DECLARATION OF JUNE ANN MUNFORD

OKTA-1016

1. My name is June Ann Munford. I am over the age of 18, have personal knowledge of the facts set forth herein, and am competent to testify to the same.

2. I earned a Master of Library and Information Science (MLIS) from the University of Wisconsin-Milwaukee in 2009. I have over ten years of experience in the library/information science field. Beginning in 2004, I have served in various positions in the public library sector including Assistant Librarian, Youth Services Librarian and Library Director. I have attached my Curriculum Vitae as Appendix CV.

3. During my career in the library profession, I have been responsible for materials acquisition for multiple libraries. In that position, I have cataloged, purchased and processed incoming library works. That includes purchasing materials directly from vendors, recording publishing data from the material in question, creating detailed material records for library catalogs and physically preparing that material for circulation. In addition to my experience in acquisitions, I was also responsible for analyzing large collections of library materials, tailoring library records for optimal catalog

search performance and creating lending agreements between libraries during my time as a Library Director.

4. I am fully familiar with the catalog record creation process in the library sector. In preparing a material for public availability, a library catalog record describing that material would be created. These records are typically written in Machine Readable Catalog (herein referred to as "MARC") code and contain information such as a physical description of the material, metadata from the material's publisher, and date of library acquisition. In particular, the 008 field of the MARC record is reserved for denoting the date of creation of the library record itself. As this typically occurs during the process of preparing materials for public access, it is my experience that an item's MARC record indicates the date of an item's public availability.

5. Typically, in creating a MARC record, a librarian would gather various bits of metadata such as book title, publisher and subject headings among others and assign each value to a relevant numerical field. For example, a book's physical description is tracked in field 300 while title/attribution is tracked in field 245. The 008 field of the MARC record is reserved for denoting the creation of the library record itself. As this is the only date reflecting the inclusion of said materials within the library's collection, it is my experience

that an item's 008 field accurately indicates the date of an item's public availability.

6. I am familiar with the Internet Archive, a digital library accessible to the general public. Among other services that the Internet Archive makes available to the general public is the Wayback Machine, an online archive of the web. The Internet Archive's Wayback Machine service archives webpages as of a certain capture date to track changes in the web over time. The Internet Archive has been in operation as a nonprofit library since 1996 and has hosted the Wayback Machine service since its inception in 2001. During my time as a librarian, I frequently used the Internet Archive's Wayback Machine for research and instruction purposes. This includes teaching instructional classes on using the Wayback Machine to library patrons and using the Wayback Machine to research reference inquiries that require hard-to-find online resources. I consider the Internet Archive's recordskeeping to be as rigorous and detailed as other formal library recordskeeping practices such as MARC records, OCLC records and Dublin Core.

7. I have reviewed Exhibit OKTA-1009, a document entitled Identity Metasystem Interoperability Version 1.0.

8. Attached hereto as Appendix OASIS01 is a screen capture of the Internet Archive Wayback Machine entry for http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.doc. I secured these screen captures myself from https://web.archive.org/web/20100707094547/http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.doc. This record was captured as of July 7, 2010.

9. Appendix OASIS01 is an Internet Archive capture of a file entitled Identity Metasystem Interoperability Version 1.0 as hosted on the Oasis website. In comparing Appendix OASIS01 to Exhibit OKTA-1009, it is my determination that Exhibit OKTA-1009 is a true and correct copy of Identity Metasystem Interoperability Version 1.0 as hosted on the Oasis website. As the Internet Archive first captured this file as of July 7, 2010, it is my determination that Identity Metasystem Interoperability Version 1.0 was first made available to the public as of July 7, 2010 via the Internet Archive and likely earlier via the original Oasis website.

10. Attached hereto as Appendix OASIS02 is a screen capture of the Internet Archive Wayback Machine entry for http://docs.oasis-

open.org/imi/identity/v1.0/os/identity-1.0-spec-os.pdf. I secured these screen captures myself from https://web.archive.org/web/20100707043418/http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.pdf. This record was captured as of July 7, 2010.

11. Appendix OASIS02 is an Internet Archive capture of a file entitled Identity Metasystem Interoperability Version 1.0 as hosted on the Oasis website. In comparing Appendix OASIS02 to Exhibit OKTA-1009, it is my determination that Exhibit OKTA-1009 is a true and correct copy of Identity Metasystem Interoperability Version 1.0 as hosted on the Oasis website. As the Internet Archive first captured this file as of July 7, 2010, it is my determination that Identity Metasystem Interoperability Version 1.0 was first made available to the public as of July 7, 2010 via the Internet Archive and likely earlier via the original Oasis website.

12. Attached hereto as Appendix OASIS03 is a screen capture of the Internet Archive Wayback Machine entry for http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.html. I secured these screen captures myself from https://web.archive.org/web/20090916031534/http://docs.oasis-

open.org/imi/identity/v1.0/os/identity-1.0-spec-os.html. This record was captured as of September 16, 2009.

13. Appendix OASIS03 is an Internet Archive capture of a web page entitled Identity Metasystem Interoperability Version 1.0 as hosted on the Oasis website. Page 2 of Appendix OASIS03 contains a link to download the DOC and PDF versions of the OASIS Standard presented in Appendices OASIS01 and OASIS02. The page captured in Appendix OASIS03 indicates the Oasis organization intended to present this standard to visitors to the Oasis website.

14. Attached hereto as Appendix OASIS04 is a screen capture of a webpage entitled Information Cards and Affirmative Statements by M. Ivkovic, et al as hosted on the Semantic Scholar website. I secured these screen captures myself from https://www.semanticscholar.org/paper/Information-Cards-and-Affirmative-Statements-Ivkovic-Centner/ab4936155796199cf7692f18eb22fe653e0eb63b.

15. Appendix OASIS04 is a paper entitled Information Cards and Affirmative Statements by M. Ivkovic, et al as presented on the Semantic Scholar website. According to page 3 of Appendix OASIS04, Information Cards and

Affirmative Statements by M. Ivkovic, et al was published in 2010 in Proceedings of the 6th International Conference on Web Information Systems and Technology. Page 9 of Appendix OASIS04 contains a citation and link to the Oasis Identity Metasystem Interoperability Version 1.0 standard.

16. Attached hereto as Appendix OASIS05 is a screen capture of a webpage entitled Magentix2: A Privacy-Enhancing Agent Platform by Jose M. Such, et al as hosted on the ScienceDirect website. I secured these screen captures myself from https://www.sciencedirect.com/science/article/abs/pii/S0952197612001522.

17. Appendix OASIS05 is a paper entitled Magentix2: A Privacy-Enhancing Agent Platform by Jose M. Such, et al as presented on the ScienceDirect website. According to page 1 of Appendix OASIS05, Magentix2: A Privacy-Enhancing Agent Platform by Jose M. Such, et al was published in January 2013 in Engineering Applications of Artificial Intelligence. Page 13 of Appendix OASIS05 contains a citation and link to the Oasis Identity Metasystem Interoperability Version 1.0 standard.

18. Attached hereto as Appendix OASIS06 is a screen capture of a webpage entitled Formal Analysis of Information Card Federated Identity-Management Protocol by Juan Wang, et al as hosted on the IEEE Xplore website. I secured these screen captures myself from https://ieeexplore.ieee.org/document/10284735.

19. Appendix OASIS06 is a paper entitled Formal Analysis of Information Card Federated Identity-Management Protocol by Juan Wang, et al as presented on the IEEE Xplore website. According to page 1 of Appendix OASIS06, Formal Analysis of Information Card Federated Identity-Management Protocol by Juan Wang, et al was published in January 2013 in the Chinese Journal of Electronics. Page 14 of Appendix OASIS06 contains a citation and link to the Oasis Identity Metasystem Interoperability Version 1.0 standard.

20. Attached hereto as Appendix OASIS07 is a screen capture of a webpage entitled OASIS Approves Interoperability Standard for Information Cards as hosted on the Xml.org website. I secured these screen captures myself from https://idtrust.xml.org/news/oasis-approves-interoperability-standard-for-information-cards.

21. Appendix OASIS07 is an article entitled Formal Analysis of Information OASIS Approves Interoperability Standard for Information Cards as presented on the xml.org website. According to page 1 of Appendix OASIS07, OASIS Approves Interoperability Standard for Information Cards was published as of July 16, 2009 on xml.org. This article describes the Oasis Identity Metasystem Interoperability Version 1.0 standard and contains the following passage: 'As with all OASIS projects, archives of the Committee's work are accessible to both members and non-members'.

22. Attached hereto as Appendix OASIS08 is a screen capture of a document entitled ITU-T Workshop on Addressing Securing Challenges on a Global Scale as hosted on the ITU-T website. I secured these screen captures myself from https://www.itu.int/dms_pub/itu-t/oth/06/35/T063500000200605PDFE.pdf.

23. Appendix OASIS08 is a presentation entitled ITU-T Workshop on Addressing Securing Challenges on a Global Scale as presented on the ITU-T website. According to page 1 of Appendix OASIS08, this document was presented as of December 6-7, 2010. Page 11 of Appendix OASIS08 contains the following passage: 'Publicly reviewed specification…Specifications are free'.

24. Attached hereto as Appendix OASIS09 is a screen capture of a webpage entitled OASIS Identity Metasystem as hosted on the Internet Identity Workshop Wiki website. I secured these screen captures myself from https://iiw.idcommons.net/OASIS_Identity_Metasystem.

25. Appendix OASIS09 is a web page entitled OASIS Identity Metasystem as presented on the IIW wiki. According to page 1 of Appendix OASIS08, this document was last updated as of December 1, 2008. Page 1 of Appendix OASIS09 contains links to a draft copy of the OASIS Approves Interoperability Standard for Information Cards.

26. Appendix OASIS03 indicates the OASIS standard was available via the OASIS website no later than September 16, 2009. Appendices OASIS04 - OASIS06 each contain a citation to the OASIS standard, indicating the general availability of the standard to the authors of those works from 2010 – 2013. Appendix OASIS07 reports the standard was approved as of July 16, 2009, Appendix OASIS08 indicates OASIS standards are broadly available to the public and Appendix OASIS09 indicates drafts of this standard were available to the public prior to being approved in July 2009. Considering each of the OASIS standards, it is my determination Formal Analysis of

Information Card Federated Identity-Management Protocol was available to the public certainly no later than July 7, 2010 and likely earlier via the OASIS website.

27. I have reviewed Exhibit OKTA-1012, a document entitled The TLS Protocol Version 1.0.

28. Attached hereto as Appendix TLS01 is a screen capture of the Internet Archive Wayback Machine entry for http://ftp.isi.edu/in-notes/rfc2246.txt. I secured these screen captures myself from https://web.archive.org/web/20011222210559/http://ftp.isi.edu/in-notes/rfc2246.txt. This record was captured as of December 22, 2001.

29. Appendix TLS01 is an Internet Archive capture of a file entitled The TLS Protocol Version 1.0 as hosted on the Network Working Group website. In comparing Appendix TLS01 to Exhibit OKTA-1012, it is my determination that Exhibit OKTA-1012 is a true and correct copy of The TLS Protocol Version 1.0 as hosted on the Network Working Group website. As the Internet Archive first captured this file as of December 22, 2001, it is my determination that The TLS Protocol Version 1.0 was first made available to

the public as of December 22, 2001 via the Internet Archive and likely earlier via the original Network Working Group website.

30. I have been retained on behalf of the Petitioner to provide assistance in the above-illustrated matter in establishing the authenticity and public availability of the documents discussed in this declaration. I am being compensated for my services in this matter at the rate of $200.00 per hour plus reasonable expenses. My statements are objective, and my compensation does not depend on the outcome of this matter.

31. I declare under penalty of perjury that the foregoing is true and correct. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Dated: 3/9/2026



June Ann Munford

Appendix
CV

June A. Munford
Curriculum Vitae

Education

University of Wisconsin-Milwaukee - MS, Library & Information Science, 2009
Milwaukee, WI

- Coursework included cataloging, metadata, data analysis, library systems, management strategies and collection development.
- Specialized in library advocacy, cataloging and public administration.

Grand Valley State University - BA, English Language & Literature, 2008
Allendale, MI

- Coursework included linguistics, documentation and literary analysis.
- Minor in political science with a focus in local-level economics and government.

Professional Experience

Researcher / Expert Witness, October 2017 – present
Freelance ● Pittsburgh, Pennsylvania & Grand Rapids, Michigan

- Material authentication and public accessibility determination. Declarations of authenticity and/or public accessibility provided upon research completion. Experienced with appeals and deposition process.

- Research provided on topics of public library operations, material publication history, digital database services and legacy web resources.

- Past clients include Alston & Bird, Arnold & Porter, Baker Botts, Fish & Richardson, Erise IP, Irell & Manella, O'Melveny & Myers, Perkins-Coie, Pillsbury Winthrop Shaw Pittman and Slayden Grubert Beard.

Library Director, February 2013 - March 2015
Dowagiac District Library ● Dowagiac, Michigan

- Executive administrator of the Dowagiac District Library. Located in Southwest Michigan, this library has a service area of 13,000, an annual operating budget of over $400,000 and total assets of approximately $1,300,000.

● Developed careful budgeting guidelines to produce a 15% surplus during the 2013-2014 & 2014-2015 fiscal years while being audited.

● Using this budget surplus, oversaw significant library investments including the purchase of property for a future building site, demolition of existing buildings and building renovation projects on the current facility.

● Led the organization and digitization of the library's archival records.

● Served as the public representative for the library, developing business relationships with local school, museum and tribal government entities.

● Developed an objective-based analysis system for measuring library services - including a full collection analysis of the library's 50,000+ circulating items and their records.

November 2010 - January 2013
Librarian & Branch Manager, Anchorage Public Library ● Anchorage, Alaska

● Headed the 2013 Anchorage Reads community reading campaign including event planning, staging public performances and creating marketing materials for mass distribution.

● Co-led the social media department of the library's marketing team, drafting social media guidelines, creating original content and instituting long-term planning via content calendars.

● Developed business relationships with The Boys & Girls Club, Anchorage School District and the US Army to establish summer reading programs for children.

June 2004 - September 2005, September 2006 - October 2013
Library Assistant, Hart Area Public Library
Hart, MI

● Responsible for verifying imported MARC records and original MARC cataloging for the local-level collection as well as the Michigan Electronic Library.

● Handled OCLC Worldcat interlibrary loan requests & fulfillment via ongoing communication with lending libraries.

Professional Involvement

Alaska Library Association - Anchorage Chapter

● Treasurer, 2012

Library Of Michigan
- ● Level VII Certification, 2008
- ● Level II Certification, 2013

Michigan Library Association Annual Conference 2014
- ● New Directors Conference Panel Member

Southwest Michigan Library Cooperative
- ● Represented the Dowagiac District Library, 2013-2015

Professional Development

Library Of Michigan Beginning Workshop, May 2008
Petoskey, MI
- ● Received training in cataloging, local history, collection management, children's literacy and reference service.

Public Library Association Intensive Library Management Training, October 2011
Nashville, TN
- ● Attended a five-day workshop focused on strategic planning, staff management, statistical analysis, collections and cataloging theory.

Alaska Library Association Annual Conference 2012 - Fairbanks, February 2012
Fairbanks, AK
- ● Attended seminars on EBSCO advanced search methods, budgeting, cataloging, database usage and marketing.

Depositions

2019 ● Fish & Richardson

Apple v. Qualcomm (Case No. IPR2018-001281, 39521-00421IP, IPR2018-01282 and 39521-00421IP2)

2019 ● Erise IP

Implicit, LLC v. Netscout Systems, Inc (Case No. 2:18-cv-53-JRG)

2019 ● Perkins-Coie

Adobe Inc. v. RAH Color Technologies LLC (Case No. IPR2019-00627, IPR2019-00628, IPR2019-00629 and IPR2019-00646)

2020 ● O'Melveny & Myers

Maxell, Ltd. v. Apple Inc. (Case No. 5:19-cv-00036-RWS)

2021 ● Pillsbury Winthrop Shaw Pittman LLP

Intel v. SRC (Case No. IPR2020-1449)

2022 ● Perkins-Coie

Realtek v. Future Link (Case No. IPR2021-01182)

2023 ● Fish & Richardson

Neuroderm Ltd. v. Abbvie, Inc (Case No. PGR2022-00040)

2023 ● Fish & Richardson

Nearmap US Inc. v. Pictometry International Corp. (Case No. IPR2022-00735)

2023 ● Fish & Richardson

Samsung Electronics v. MemoryWeb LLC (Case No. 39843-0136PS1)

2023 ● Pillsbury Winthrop Shaw Whitman LLP

Gravel Rating Systems v. Costco Wholesale Corp. (Case No. 4:21-cv-149-ALM)

2024 ● Willkie-Farr

Netflix, Inc. v. VideoLabs. Inc. (Case No. IPR2023-00628)

2024 ● Quinn Emanuel Urquhart & Sullivan

Purdue University v. Wolfspeed (Case No. 1:21-CV-840)

2024 ● Fish & Richardson

Dish Network v. Entropic Communications (Case No. IPR2024-00393)

2025 ● Perkins Coie

Amazon.com, Inc. v. Nokia Technologies Oy (Case No. IPR2024-00572)

2025 ● Fish & Richardson

Dish Network v. Entropic Communications (Case No. IPR2024-00373)

2025 ● Fish & Richardson

SK Nexilis Co., Ltd. v. Solus Advanced Materials (Case No. 2:2023-cv-00539)

2025 ● Covington & Burling

Samsung v. Pictiva (Case No. IPR2024-01222)

2026 ● Fish & Richardson

Shenzhen & Bambu Labsv. Stratasys (Case No. 2:24-cv-00644-JRG)

2026 ● Wilmer, Cutler, Pickering, Hale and Dorr

Shopify v. Lower48 (Case No. 6:22-cv-00997-DAE)

Limited Case and Clientele History

Alston & Bird

- Ericsson

  v. Collision Communications (Case No. IPR2022-01233)

- Nokia

  v. Neptune Subsea, Xtera (Case No. 1:17-cv-01876)

- Universal Electronics Inc

  v. Roku Inc (Case No. IPR2022-00818)

Alavi & Anaipakos PLLC

- Stingray Group Inc. and Stingray Music USA, Inc.

  v. Edwin A. Hernandez-Mondragon and Egla Corp. (Case No. 1:24-cv-21226-RAR)

Arnold & Porter

- Ivantis

  v. Glaukos (Case No. 8:18-cv-00620)

- Samsung

  v. Jawbone (Case No. 2:21-cv-00186)

Benesch Friedlander Coplan & Aronoff

- Voyis

  v. Cathx (Case No. 5:21-cv-00077-RWS)

Buchanan, Ingersoll & Rooney PC

- Google LLC, AT&T Service Inc., T Mobile USA Inc., Cellco Partnership, Ericsson Inc. & Nokia of America Corp.

  v. KT Corp & Pegasus Wireless Innovation LLC (Case No. IPR2025-00293)

- Google LLC

  v. K. Mizra LLC (Case No. 1:25-cv-00236)

Covington & Burling

- First Solar, Inc.

  v. JinkoSolar (Case No. 25-cv-223)

- Samsung

  v. Pictiva (Case No. IPR2024-01222)

Deschert LLP

- Smaxtec Animal Care

  v. ST Reproductive Technologies, LLC (Case No. IPR2024-00885)

Duane Morris

- Kangxi Communication Technologies

  v. Skyworks Solutions Inc. (Case No. IPR2025-00373)

- Microsoft

  v. Edge Networking Systems LLC (Case No. 1:24-cv-00215)

Erise I.P.

- Apple

  v. Ericsson Inc. (Case No. IPR2022-00715)

  v. Future Link Systems (Case No. IPRs 6317804, 6622108, 6807505, and 7917680)

  v. INVT (Case No. 20-1881)

  v. Navblazer LLC (Case No. IPR2020-01253)

  v. Qualcomm (Case No. IPR2018-001281, 39521-00421IP, IPR2018-01282, 39521-00421IP2)

  v. Quest Nettech Corp (Case No. 2:19-cv-00118-JRG)

v. Telefonaktiebolaget LM Ericsson (Case No. IPR2022-00275)

v. Theta IP, LLC (Case No. IPR2024-00818)

v. THL Holding Company LLC (Case No. 1:23-cv-00548)

- CrowdStrike

  v. Skysong Innovations (Case No. IPR2025-01398)

- Fanduel

  v. CGT (Case No. 19-1393)

- Garmin

  v. Phillips North America LLC (Case No. 2:19-cv-6301-AB-KS)

- Netscout

  v. Longhorn HD LLC (Case No. 2:20-cv-00349)

  v. Implicit, LLC (Case No. 2:18-cv-53-JRG)

- Sony Interactive Entertainment LLC

  v. Bot M8 LLC (Case No. IPR2020-01288)

  v. Infernal Technology LLC (Case No. 2:19-CV-00248-JRG)

- Tesla

  v. Charge Fusion Technologies LLC (Case No. IPR2025-00032)

- Unified Patents

  v. GE Video Compression (Case No. 2:19-cv-248)

- Apple
    - v. AliveCor (Case No. 3:21-cv-03958)
    - v. LBS Innovations (Case No. 2:19-cv-00119-JRG-RSP)
    - v. Koss Corporation (Case No. IPR2021-00305)
    - v. Masimo (Case No. IPR 50095-0012IP1, 50095-0012IP2, 50095-0013IP1, 50095-0013IP2, 50095-0006IP1, 50095-0135IP1)
    - v. Neonode (Case No. 21-cv-08872-EMC)
    - v. Qualcomm (Case No. IPR2018-001281, 39521-00421IP, IPR2018-01282, 39521-00421IP2)
    - v. Resonant Sys. (Case No. 7:23-cv-00077-ADA)

- AsusTek Computer Inc.
    - v. Videolabs, Inc. (Case No. 22-CV-00720-ADA)

- Dell
    - v. Neo Wireless (Case No. IPR2022-00616)

- Dish Network
    - v. Entropic Communications, LLC (Case No. 2:2023-CV-01043)
    - v. Entropic Communications LLC (Case No. IPR2024-00393)
    - v. Realtime Adaptive Streaming (Case No. 1:17-CV-02097-RBJ)
    - v. TQ Delta LLC (Case No. 18-1798)

- Evapco Dry Cooling
    - v. SPG Dry Cooling (Case No. IPR2021-00688)

- Generac
    - v. PSLC (Case No. 2:24-cv-01270)

- Genetec

v. Sensormatic Electronics (Case No. 1:20-CV-00760)

- Huawei

  v. Bell Northern Research LLC (Case No. IPR2019-01174)

- Kianxis

  v. Blue Yonder (Case No. 3:20-cv-03636)

- LG Electronics

  v. Bell Northern Research LLC (Case No. 3:18-cv-2864-CAB-BLM)

- MediaTek

  v. MOSAID (Case No. IPR2024-00718)

- Metaswitch

  v. Sonus Networks (Case No. IPR2018-01719)

- Microsoft

  v. Throughputer Inc (Case No. IPR2022-00757)

- Mom Enterprises

  v. Ddrops Company (Case No. 1:22-cv-00332)

- MLC Intellectual Property

  v. MicronTech (Case No. 3:14-cv-03657-SI)

- Nearmap Inc

  v. EagleView Technologies (Case No. IPR2022-01009)

- Neuroderm Ltd.

  v. Abbvie, Inc (Case No. PGR2022-00040)

- Par-Kan
  - v. Unverferth (Case No. 3:23-cv-00653)

- Posco Co Ltd.
  - v. Pascal Drillet et al. (Case No. IPR2025-00370)

- Realtek Semiconductor
  - v. Future Link (Case No. IPR2021-01182)

- Quectel
  - v. Koninklijke Philips (Case No. 1:20-cv-01710)

- Samsung
  - v. Aire Technology (Case No. IPR2022-00877)
  - v. Bell Northern Research (Case No. 2:19-cv-00286-JRG)
  - v. Communication Technologies Inc (Case No. IPR2022-01221)
  - v. Jawbone Innovations (Case No. IPR2022-00865)
  - v. MemoryWeb LLC (Case No. IPR2022-00885)
  - v. Dodots Licensing Solutions (Case No. IPR2023-00701)
  - v. Telefonaktiebolaget LM Ericsson (Case No. IPR2021-00615)

- Tesla
  - v. Intellectual Ventures (Case No. 1:24-cv-00390, 6:24-cv-00188)

- Texas Instruments
  - v. ParkerVision (Case No. IPR2024-00934)
  - v. Vantage Micro LLC (Case No. IPR2020-01261)

- Xilinx

  v. Sentient Sensors LCC (Case No. 1:22-cv-00173)

Goodwin Proctor

- Intel

  v. Proxense, LLC (Case No. 6:24-cv-00283)

- Samsung Electronics

  v. Mobile Data Technologies (Case No. 2:24-cv-00435)

- Taiwan Semiconductor Manufacturing Company Ltd.

  v. Advanced Integrated Circuit Process LLC (Case No. 2:24-cv-00623)

Groombridge Wu

- Atossa

  v. Jina & Intas Pharmaceuticals (Case No. PGR2025-00043)

- Nearmap

  v. Eagle View Technologies Inc. and Pictometry International Corp. (Case No. 2:21-cv-00283-TS-DAO)

Holland & Knight

- Wilmington Trust, N.A.

  v. Herman Segal & John Hancock Life Insurance Company of New York (Case No. 1:21-CV-01540-PKC-TAM)

Irell & Manella

- Curium

Latham & Watkins LLP

- Apple

  v. Advanced Coding Research LLC

- Google

  v. Advanced Coding Research (Case No. 2:24-cv-0035)

- Western Digital Technologies

  MR Technologies GmbH (Case No. 8:24-cv-01848-JVS)

Leydig, Voit & Mayer, Ltd.

- Public Availability Consultancy

O'Melveny & Myers

- Apple

  v. Maxell (Case No. 5:19-cv-00036-RWS)

- Disney Media & Entertainment Distribution LLC

  v. Digital Media Technology Holdings, LLC (Case No. 2:22-cv-01642)

- Micron Technology Inc.
    - v. Besang Inc (Case No. IPR2023-00900)

- Samsung
    - v. Daedalus Prime LLC (1335 Investigation)

Perkins-Coie

- Amazon
    - v. Nokia (Case No. IPR2024-00847 and IPR2024-00848)
    - v. Nokia Technologies Oy (Case No. IPR2024-00572)

- Google LLC
    - v. Clear Imaging Research LLC (Case No. 202510451)

- Heru Industries
    - v. The UAB Research Foundation (Case No. IPR2022-01148)

- Intel Corporation
    - v. BeSang Inc. (Case No. IPR2023-00991)
    - v. BitMICRO (Case No. 5:23-cv-00625)
    - v. Telefonaktiebolaget L M Ericsson (Case No. IPR2024-00610)

- Microsoft
    - v. Sandpiper (Case No. 2:25-cv-00664)

- Niantic, Inc.
    - v. ImagineAR, Inc. (C.A. No. 24-1252-JD)

- Realtek Semiconductor

  v. Future Link (Case No. IPR2021-01182)

- r-pac

  v. Adasa, Inc. (Case No. 1:2024cv06102)

- Twitter Inc

  v. VOIP-Pal.com (Case No. 3:20-cv-02397-JD)

- Twitch/Amazon

  v. RazDog (Case No. IPR2025-00308)

- TCL Industries

  v. Koninklijke Philips NV (Case No. IPR2021-00495, IPR2021-00496 and IPR2021-00497)

- VusionGroup

  v. Hanshow (Case No. IPR2024-00857)

Pillsbury Winthrop Shaw Pittman

- Intel

  v. FG SRC LLC (Case No. 6:20-cv-00315)

- Gravel Rating Systems

  v. Costco (Case No. 4:21-cv-149)
  v. Lowe's Home Centers (Case No. 4:21-cv-150)
  v. T-Mobile USA (Case No. 4:21-cv-152)
  v. Kohl's Inc. (Case No. 4:21-cv-258)
  v. Under Armor (Case No. 4:21-cv-356)

Quinn Emanuel

- Exact Sciences Corporation

  v. Geneoscopy, Inc. (Case No. IPR2024-00459 and IPR2024-1330)

- ServiceNow, Inc.

  v. InQuisient, Inc. (Case No. 22-900-CJB)

- Wolfspeed

  v. The Trustees of Purdue University (Case No. 1:2021-cv-00840)

Sheppard, Mullin, Richter & Hampton LLP

- Advanced Micro Devices, Inc and Pensando Systems, Inc.

  v. Concurrent Ventures LLC and Xtreamedge, Inc. (Case No. IPR2025-00478)

- Amazon

  v. Nokia

- Cadence Design Systems Inc.

  v. Semiconductor Design Technologies (Case No. 3:23-cv-01001)

- Marvell Semiconductor

  v. Credo Technology (Case No. IPR2025-01219)

- Sandisk Technologies, Inc.

  v. Longitude Flash Memory Solutions Ltd. (Case No. IPR2025-01283)

Shook, Hardy & Bacon LLP

- HP Inc.

  v. Universal Connectivity Technologies (Case No. 1:23-cv-01177)

Stern, Kessler, Goldstein & Fox

- Preliminary Research

Wilmer, Cutler, Pickering, Hale and Dorr

- Apple

  v. Shunock (Case No. 1:23-cv-08598)

- Dell Technologies Inc

  v. EireOg Innovations Ltd. (Case No. 1:24-cv-00416-AD)

- Dropbox

- Quest Diagnostics Inc., Haystack Oncology Inc. and Haystack Oncology GmbH

  v. Natera, Inc. (Case No. 1:23-cv-08598)

- NVIDIA

  v. Onesta IP (Case No. 1:25-cv-00586)

- Oracle

  v. EireOg Innovations Ltd. (Case No. 4-cv-00697-ADA)

- Roche Diabetes, Inc.

  v. Trividia Health, Inc. (Case No. 1:24-cv-00668)

- Shopify Inc.

  v. Lower48 IP LLC (C.A. No. 6:22-cv-00997-DAE)

- United Services Automobile Association

  v. Regions Bank (Case No. 2-25-cv-00082)

Willkie, Farr & Gallagher LLP

- Lenovo, Dell, HP

  v. Universal Connectivity Technologies Inc. (Case No. 2:2023cv00449)

- Neurent

  v. The Foundry, LLC (Case No. IPR2024-00669)

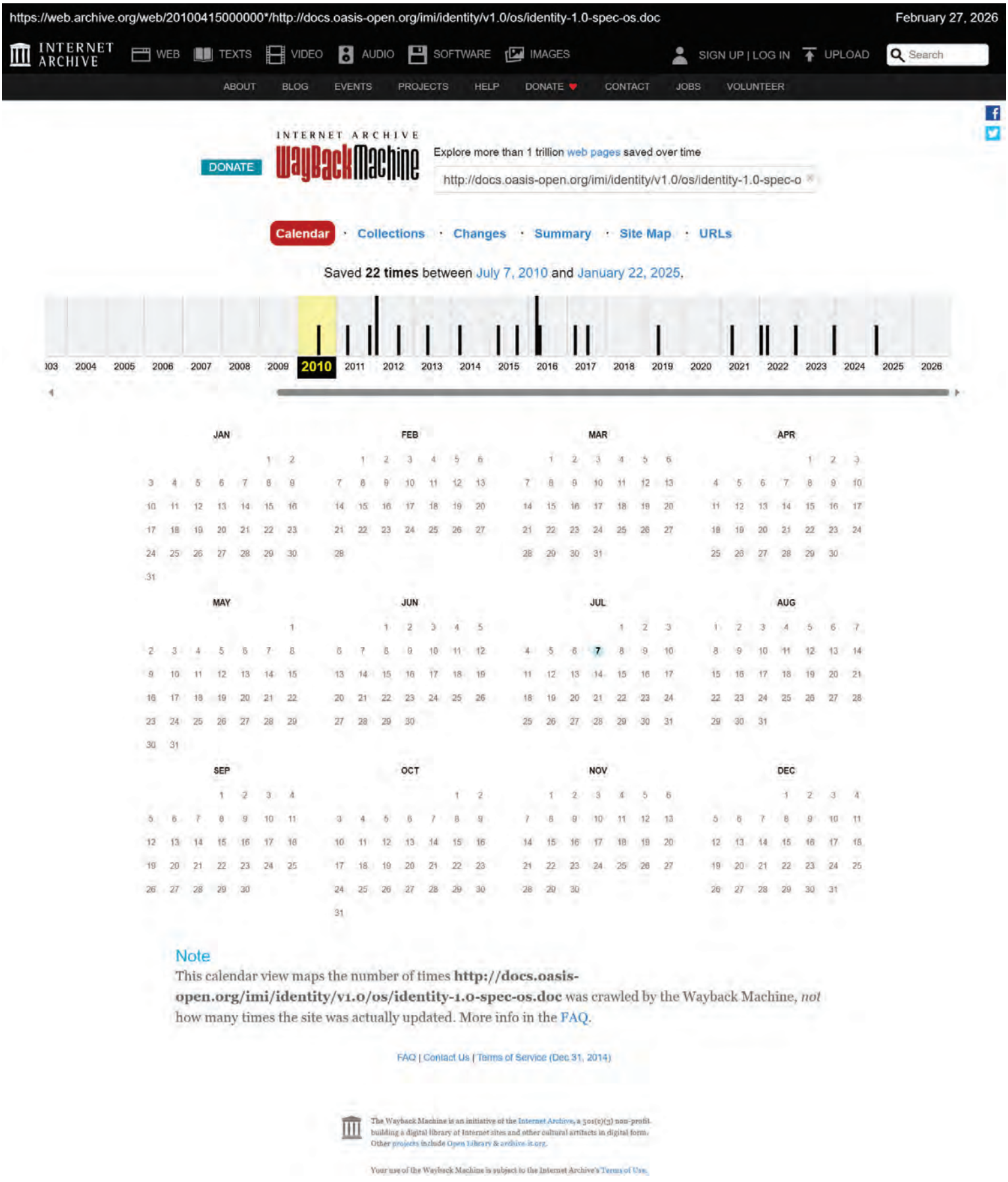
- Netflix, Inc.

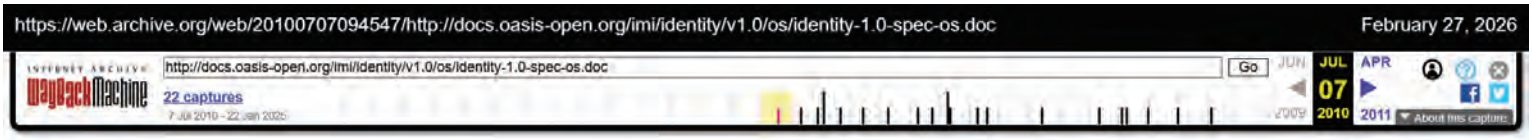
  v. DivX (Case No. 2:19-cv-1602-GW-AGR)

- Netflix, Inc.

  v. VideoLabs, Inc. (Case No. IPR2023-00628)

Appendix
OASIS01


https://web.archive.org/web/20100415000000*/http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.doc
February 27, 2026
INTERNET ARCHIVE
WEB TEXTS VIDEO AUDIO SOFTWARE IMAGES
SIGN UP | LOG IN UPLOAD
Search
ABOUT BLOG EVENTS PROJECTS HELP DONATE CONTACT JOBS VOLUNTEER
DONATE
INTERNET ARCHIVE WayBackMachine
Explore more than 1 trillion web pages saved over time
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-o
Calendar · Collections · Changes · Summary · Site Map · URLs
Saved 22 times between July 7, 2010 and January 22, 2025.
2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 2026
JAN FEB MAR APR
MAY JUN JUL AUG
SEP OCT NOV DEC
Note
This calendar view maps the number of times http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.doc was crawled by the Wayback Machine, not how many times the site was actually updated. More info in the FAQ.
FAQ | Contact Us | Terms of Service (Dec 31, 2014)
The Wayback Machine is an initiative of the Internet Archive, a 501(c)(3) non-profit, building a digital library of Internet sites and other cultural artifacts in digital form. Other projects include Open Library & archive-it.org.
Your use of the Wayback Machine is subject to the Internet Archive's Terms of Use.


https://web.archive.org/web/20100707094547/http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.doc
February 27, 2026
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.doc
Go
JUN
JUL
APR
07
2010
2011
22 captures
About this capture

Identity Metasystem Interoperability Version 1.0

OASIS Standard

1 July 2009

Specification URIs:
This Version:
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.html
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.doc (Authoritative)
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.pdf

Previous Version:
http://docs.oasis-open.org/imi/identity/v1.0/cs/identity-1.0-spec-cs-01.html
http://docs.oasis-open.org/imi/identity/v1.0/cs/identity-1.0-spec-cs-01.doc (Authoritative)
http://docs.oasis-open.org/imi/identity/v1.0/cs/identity-1.0-spec-cs-01.pdf

Latest Version:
http://docs.oasis-open.org/imi/identity/v1.0/identity.html
http://docs.oasis-open.org/imi/identity/v1.0/identity.doc
http://docs.oasis-open.org/imi/identity/v1.0/identity.pdf

Technical Committee:
OASIS Identity Metasystem Interoperability (IMI) TC

Chair(s):
Marc Goodner
Anthony Nadalin

Editor(s):
Michael B. Jones
Michael McIntosh

Related work:
This specification replaces or supersedes:

- None

This specification is related to:

- WS-Trust
- WS-SecurityPolicy
- WS-Addressing

Declared XML Namespace(s):
http://docs.oasis-open.org/imi/ns/identity-200810
http://schemas.xmlsoap.org/ws/2005/05/identity
http://schemas.xmlsoap.org/ws/2006/02/addressingidentity
http://schemas.xmlsoap.org/ws/2007/01/identity

Abstract:
This document is intended for developers and architects who wish to design identity systems and applications that interoperate using the Identity Metasystem Interoperability specification.

Copyright © OASIS® 2008-2009. All Rights Reserved.

An Identity Selector and the associated identity system components allow users to manage their Digital Identities from different Identity Providers, and employ them in various contexts to access online services. In this specification, identities are represented to users as “Information Cards”. Information Cards can be used both at applications hosted on Web sites accessed through Web browsers and rich client applications directly employing Web services.

This specification also provides a related mechanism to describe security-verifiable identity for endpoints by leveraging extensibility of the WS-Addressing specification. This is achieved via XML [XML 1.0] elements for identity provided as part of WS-Addressing Endpoint References. This mechanism enables messaging systems to support multiple trust models across networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner.

Status:

This document was last revised or approved by the Identity Metasystem Interoperability TC on the above date. The level of approval is also listed above. Check the “Latest Version” or “Latest Approved Version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the “Send A Comment” button on the Technical Committee’s web page at http://www.oasis-open.org/committees/imi/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/imi/ipr.php.

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/imi/.

Copyright © OASIS® 2008-2009. All Rights Reserved.

Notices

Copyright © OASIS® 2008-2009. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", [insert specific trademarked names and abbreviations here] are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

Table of Contents

1 Introduction 7
1.1 Notational Conventions 7
1.2 Namespaces 7
1.3 Schema 9
1.4 Terminology 9
1.5 Normative References 10
1.6 Non-Normative References 12
2 Relying Party Interactions 13
2.1 Expressing Token Requirements of Relying Party 13
2.1.1 Issuer of Tokens 13
2.1.2 Type of Proof Key in Issued Tokens 14
2.1.3 Claims in Issued Tokens 14
2.2 Expressing Privacy Policy of Relying Party 15
2.3 Employing Relying Party STSs 17
3 Identity Provider Interactions 17
3.1 Information Card 17
3.1.1 Information Card Format 17
3.1.2 Issuing Information Cards 25
3.2 Identity Provider Policy 27
3.2.1 Require Information Card Provisioning 27
3.2.2 Policy Metadata Location 27
3.3 Token Request and Response 27
3.3.1 Information Card Reference 28
3.3.2 Claims and Other Token Parameters 28
3.3.3 Token Scope 28
3.3.4 Client Pseudonym 29
3.3.5 Proof Key for Issued Token 30
3.3.6 Display Token 35
3.3.7 Token References 36
4 Authenticating to Identity Provider 36
4.1 Username and Password Credential 37
4.2 Kerberos v5 Credential 37
4.3 X.509v3 Certificate Credential 37
4.4 Self-issued Token Credential 38
5 Faults 38
5.1 Relying Party 39
5.2 Identity Provider 39
5.2.1 Identity Provider Custom Error Messages 40
6 Information Cards Transfer Format 41
6.1 Pre-Encryption Transfer Format 41
6.1.1 PIN Protected Card 43
6.1.2 Computing the ic:IssuerId 44
6.1.3 Computing the ic:IssuerName 45

Copyright © OASIS® 2008-2009. All Rights Reserved.

6.1.4 Creating the ic:HashSalt 45
6.2 Post-Encryption Transfer Format 45
7 Simple Identity Provider Profile 47
7.1 Self-Issued Information Card 47
7.2 Self-Issued Token Characteristics 47
7.3 Self-Issued Token Encryption 51
7.4 Self-Issued Token Signing Key 52
7.4.1 Processing Rules 53
7.5 Claim Types 55
7.5.1 First Name 55
7.5.2 Last Name 55
7.5.3 Email Address 55
7.5.4 Street Address 55
7.5.5 Locality Name or City 55
7.5.6 State or Province 56
7.5.7 Postal Code 56
7.5.8 Country 56
7.5.9 Primary or Home Telephone Number 56
7.5.10 Secondary or Work Telephone Number 56
7.5.11 Mobile Telephone Number 56
7.5.12 Date of Birth 57
7.5.13 Gender 57
7.5.14 Private Personal Identifier 57
7.5.15 Web Page 57
7.6 The PPID Claim 57
7.6.1 Relying Party Identifier and Relying Party PPID Seed 58
7.6.2 PPID 60
7.6.3 Friendly Identifier 60
8 Relying Parties without Certificates 61
8.1 Relying Party Identifier and Relying Party PPID Seed 61
8.2 AppliesTo Information 61
8.3 Token Signing and Encryption 62
9 Using WS-SecurityPolicy 1.2 and WS-Trust 1.3 62
9.1 Overview of Differences 62
9.2 Identity Selector Differences 62
9.3 Security Token Service Differences 63
10 Browser Behavior with Information Cards 64
10.1 Basic Protocol Flow when using an Information Card at a Web Site 64
10.2 Protocol Flow with Relying Party STS 65
10.3 User Perspective and Examples 66
10.4 Browser Perspective 67
10.5 Web Site Perspective 67
11 Invoking an Identity Selector from a Web Page 68
11.1 Syntax Alternatives: OBJECT and XHTML tags 68
11.1.1 OBJECT Syntax Examples 68

Copyright © OASIS® 2008-2009. All Rights Reserved.

11.1.2 XHTML Syntax Example 69
11.2 Identity Selector Invocation Parameters 70
11.2.1 issuer 70
11.2.2 issuerPolicy 70
11.2.3 tokenType 70
11.2.4 requiredClaims 70
11.2.5 optionalClaims 70
11.2.6 privacyUrl 70
11.2.7 privacyVersion 70
11.3 Data Types for Use with Scripting 70
11.4 Detecting and Utilizing an Information Card-enabled Browser 71
11.5 Behavior within Frames 71
11.6 Invocation Using the Document Object Model (DOM) 71
11.7 Auditing, Non-Auditing, and Auditing-Optional Cards 71
12 Endpoint Reference wsai:Identity Property 72
12.1 Default Value 72
12.2 Identity Representation 72
12.2.1 DNS Name 72
12.2.2 Service Principal Name 72
12.2.3 User Principal Name 72
12.2.4 KeyInfo 73
12.2.5 Security Token 73
12.2.6 Security Token Reference 74
13 Security Considerations 75
13.1 Protection of Information Cards by Identity Selectors 75
13.2 Relying Parties Without Certificates 75
13.3 Endpoint References 75
14 Conformance 76
A. HTTPS POST Sample Contents 77
B. Acknowledgements 80

Copyright © OASIS® 2008-2009. All Rights Reserved.

1 Introduction

The Identity Metasystem Interoperability specification prescribes a subset of the mechanisms defined in [WS-Trust 1.2], [WS-Trust 1.3], [WS-SecurityPolicy 1.1], [WS-SecurityPolicy 1.2], and [WS-MetadataExchange] to facilitate the integration of Digital Identity into an interoperable token issuance and consumption framework using the Information Card Model. It documents the Web interfaces utilized by browsers and Web applications that utilize the Information Card Model. Finally, it extends WS-Addressing's endpoint reference by providing identity information about the endpoint that can be verified through a variety of security means, such as https or the wealth of WS-Security specifications.

This profile constrains the schema elements/extensions used by the Information Card Model, and behaviors for conforming Relying Parties, Identity Providers, and Identity Selectors.

1.1 Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119].

This specification uses the following syntax to define outlines for assertions:

- The syntax appears as an XML instance, but values in italics indicate data types instead of literal values.
- Characters are appended to elements and attributes to indicate cardinality:
  - "?" (0 or 1)
  - "*" (0 or more)
  - "+" (1 or more)
- The character "|" is used to indicate a choice between alternatives.
- The characters "(" and ")" are used to indicate that contained items are to be treated as a group with respect to cardinality or choice.
- The characters "[" and "]" are used to call out references and property names.
- Ellipses (i.e., "...") indicate points of extensibility. Additional children and/or attributes MAY be added at the indicated extension points but MUST NOT contradict the semantics of the parent and/or owner, respectively. By default, if a receiver does not recognize an extension, the receiver SHOULD ignore the extension; exceptions to this processing rule, if any, are clearly indicated below.
- XML namespace prefixes (see Table 2) are used to indicate the namespace of the element being defined.

Elements and Attributes defined by this specification are referred to in the text of this document using XPath 1.0 expressions. Extensibility points are referred to using an extended version of this syntax:

- An element extensibility point is referred to using {any} in place of the element name. This indicates that any element name can be used, from any namespace other than the namespace of this specification.
- An attribute extensibility point is referred to using @{any} in place of the attribute name. This indicates that any attribute name can be used, from any namespace other than the namespace of this specification.

Extensibility points in the exemplar might not be described in the corresponding text.

1.2 Namespaces

Table 1 lists the XML namespaces that are used in this document.

Copyright © OASIS® 2008-2009. All Rights Reserved.

Prefix	XML Namespace	Specification(s)
ds	http://www.w3.org/2000/09/xmldsig#	XML Digital Signatures
ic	http://schemas.xmlsoap.org/ws/2005/05/identity	This document
ic07	http://schemas.xmlsoap.org/ws/2007/01/identity	Namespace for additional elements also defined by this document
ic08	http://docs.oasis-open.org/imi/ns/identity-200810	Namespace for new elements defined by this document
S	*May refer to either* http://schemas.xmlsoap.org/soap/envelope *or* http://www.w3.org/2003/05/soap-envelope *since both may be used*	SOAP
S11	http://schemas.xmlsoap.org/soap/envelope	SOAP 1.1 [SOAP 1.1]
S12	http://www.w3.org/2003/05/soap-envelope	SOAP 1.2 [SOAP 1.2]
saml	urn:oasis:names:tc:SAML:1.0:assertion	SAML 1.0
sp	*May refer to either* http://schemas.xmlsoap.org/ws/2005/07/securitypolicy *or* http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702 *since both may be used*	WS-SecurityPolicy
sp11	http://schemas.xmlsoap.org/ws/2005/07/securitypolicy	WS-SecurityPolicy 1.1 [WS-SecurityPolicy 1.1]
sp12	http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702	WS-SecurityPolicy 1.2 [WS-SecurityPolicy 1.2]
wsa	http://www.w3.org/2005/08/addressing	WS-Addressing [WS-Addressing]
wsai	http://schemas.xmlsoap.org/ws/2006/02/addressingidentity	Addressing Identity extension for WS-Addressing also defined by this document
wsdl	*May refer to either* http://schemas.xmlsoap.org/wsdl/ *or* http://www.w3.org/TR/wsdl20 *since both may be used*	Web Services Description Language
wsdl11	http://schemas.xmlsoap.org/wsdl/	Web Services Description Language [WSDL 1.1]
wsdl20	http://www.w3.org/TR/wsdl20	Web Services Description Language [WSDL 2.0]
wsp	http://schemas.xmlsoap.org/ws/2004/09/policy	WS-Policy [WS-Policy]
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	WS-Security Extensions [WS-Security]

Copyright © OASIS® 2008-2009. All Rights Reserved.

wst	*May refer to either* http://schemas.xmlsoap.org/ws/2005/02/trust *or* http://docs.oasis-open.org/ws-sx/ws-trust/200512 *since both may be used*	WS-Trust
wst12	http://schemas.xmlsoap.org/ws/2005/02/trust	WS-Trust 1.2 [WS-Trust 1.2]
wst13	http://docs.oasis-open.org/ws-sx/ws-trust/200512	WS-Trust 1.3 [WS-Trust 1.3]
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	WS-SecurityUtility
wsx	http://schemas.xmlsoap.org/ws/2004/09/mex	WS-MetadataExchange [WS-MetadataExchange]
xs	http://www.w3.org/2001/XMLSchema	XML Schema [Part 1, 2]

Note that the versions identified in the above table supersede versions identified in referenced specifications.

1.3 Schema

A copy of the XML Schemas for this document can be found at:

```
http://docs.oasis-open.org/imi/identity/200810/identity.xsd
http://docs.oasis-open.org/imi/identity/200810/addr-identity.xsd
http://docs.oasis-open.org/imi/identity/200810/claims.xsd
http://docs.oasis-open.org/imi/identity/200810/identity2007.xsd
```

1.4 Terminology

The following definitions establish the terminology and usage in this document.

Information Card Model – The "*Information Card Model*" refers to the use of Information Cards containing metadata for obtaining Digital Identity claims from Identity Providers and then conveying them to Relying Parties under user control.

Information Card – An *Information Card* provides a visual representation of a Digital Identity for the end user. Information Cards contain a reference to an IP/STS that issues Security Tokens containing the Claims for that Digital Identity.

Digital Identity – A "*Digital Identity*" is a set of Claims made by one party about another party.

Claim – A "*Claim*" is a piece of information about a Subject that an Identity Provider asserts about that Subject.

Subject – A "*Subject*" is an individual or entity about whom claims are made by an Identity Provider.

Service Requester – The term "*Service Requester*" means software acting on behalf of a party who wants to obtain a service through a digital network.

Relying Party – The term "*Relying Party*" (RP) means a network entity providing the desired service, and relying upon Digital Identity.

Identity Provider – The term "*Identity Provider*" (IP) means a network entity providing the Digital Identity claims used by a Relying Party.

Security Token Service – The term "*Security Token Service*" (STS) refers to a WS-Trust endpoint.

Identity Provider Security Token Service – The term "Identity Provider Security Token Service" (IP/STS) refers to the Security Token Service run by an Identity Provider to issue tokens.

Copyright © OASIS® 2008-2009. All Rights Reserved.

Relying Party Security Token Service – The term “Relying Party Security Token Service” (RP/STS) refers to a Security Token Service run by a Relying Party to accept and issue tokens.

Identity Selector – The term “*Identity Selector*” (IS) refers to a software component available to the Service Requester through which the user controls and dispatches her Digital Identities.

Trust Identity – A *trust identity* is a verifiable claim about a principal (e.g. name, identity, key, group, privilege, capability, etc).

Security Token – A *security token* represents a collection of claims.

Signed Security Token – A *signed security token* is a security token that is asserted and cryptographically endorsed by a specific authority (e.g. an X.509 certificate, a Kerberos ticket, or a self-issued Information Card).

Unsigned Security Token – An un*signed security token* is a security token that is not cryptographically endorsed by a specific authority (e.g. a security token backed by shared secrets such as usernames and passwords).

Proof-of-Possession – The *proof-of-possession* information is data that is used in a proof process to demonstrate the sender's knowledge of information that should only be known to the claiming sender of a security token.

Integrity – *Integrity* is the process by which it is guaranteed that information is not modified in transit.

Confidentiality – *Confidentiality* is the process by which data is protected such that only authorized actors or security token owners can view the data

Digest – A *digest* is a cryptographic checksum of an octet stream.

Signature - A *signature* is a cryptographic binding of a proof-of-possession and a digest. This covers both symmetric key-based and public key-based signatures. Consequently, non-repudiation is not always achieved.

Certificate – Uses of the term *certificate* in this specification refer to X.509 certificates unless otherwise qualified. Usage of certificates is dictated by the underlying protocols, e.g. HTTPS or WS-Security, except where noted.

1.5 Normative References

[DOM]

“Document Object Model (DOM)”, November 2000. http://www.w3.org/DOM/

[EV Cert]

CA / Browser Forum, “Guidelines for the Issuance and Management of Extended Validation Certificates, Version 1.1”, April 2008. http://cabforum.org/EV_Certificate_Guidelines_V11.pdf

[HTTP]

R. Fielding et al., “IETF RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1”, June 1999. http://www.ietf.org/rfc/rfc2616.txt

[HTTPS]

E. Rescorla, “RFC 2818: HTTP over TLS”, May 2000. http://www.ietf.org/rfc/rfc2818.txt

[RFC 1274]

P. Barker and S. Kille, “RFC 1274: The COSINE and Internet X.500 Schema”, November 1991. http://www.ietf.org/rfc/rfc1274.txt

[RFC 2119]

S. Bradner, “RFC 2119: Key words for use in RFCs to Indicate Requirement Levels”, March 1997. http://www.ietf.org/rfc/rfc2119.txt

Copyright © OASIS® 2008-2009. All Rights Reserved.

[RFC 2256]

M. Wahl, “RFC 2256: A Summary of the X.500(96) User Schema for use with LDAPv3”, December 1997. http://www.ietf.org/rfc/rfc2256.txt

[RFC 2459]

R. Housley, W. Ford, W. Polk, and D. Solo, “RFC 2459: Internet X.509 Public Key Infrastructure - Certificate and CRL Profile”, January 1999. http://www.ietf.org/rfc/rfc2459.txt

[RFC 2898]

B. Kaliski, “PKCS #5: Password-Based Cryptography Specification, Version 2.0”, September 2000. http://www.ietf.org/rfc/rfc2898.txt

[RFC 3066]

H. Alvestrand, “Tags for the Identification of Languages”, January 2001. http://www.faqs.org/rfcs/rfc3066.html

[SOAP 1.1]

W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[SOAP 1.2]

M. Gudgin, et al., “SOAP Version 1.2 Part 1: Messaging Framework”, June 2003. http://www.w3.org/TR/soap12-part1/

[URI]

T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998. http://www.ietf.org/rfc/rfc2396.txt

[WS-Addressing]

W3C Recommendation, “Web Service Addressing (WS-Addressing)”, 9 May 2006. http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/

[WS-MetadataExchange]

“Web Services Metadata Exchange (WS-MetadataExchange), Version 1.1”, August 2006. http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf

[WSDL 1.1]

W3C Note, "Web Services Description Language (WSDL 1.1)," 15 March 2001. http://www.w3.org/TR/wsdl

[WSDL 2.0]

“Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language”, June 2007. http://www.w3.org/TR/wsdl20

[WS-Policy]

“Web Services Policy Framework (WS-Policy), Version 1.2”, March 2006. http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf

[WS-Security]

A. Nadalin et al., “Web Services Security: SOAP Message Security 1.0”, May 2004. http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[WS-SecurityPolicy 1.1]

“Web Services Security Policy Language (WS-SecurityPolicy), Version 1.1”, July 2005. http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf

Copyright © OASIS® 2008-2009. All Rights Reserved.

[WS-SecurityPolicy 1.2]

OASIS, "WS-SecurityPolicy 1.2", July 2007. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.pdf

[WS-Trust 1.2]

"Web Services Trust Language (WS-Trust)", February 2005. http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf

[WS-Trust 1.3]

OASIS, "WS-Trust 1.3", March 2007. http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf

[XML 1.0]

W3C Recommendation, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", September 2006. http://www.w3.org/TR/xml/

[XMLDSIG]

Eastlake III, D., Reagle, J., and Solo, D., "XML-Signature Syntax and Processing", March 2002. http://www.ietf.org/rfc/rfc3275.txt

[XMLENC]

Imamura, T., Dillaway, B., and Simon, E., "XML Encryption Syntax and Processing", August 2002. http://www.w3.org/TR/xmlenc-core/

[XML Schema, Part 1]

H. Thompson et al., "XML Schema Part 1: Structures", May 2001. http://www.w3.org/TR/xmlschema-1/

[XML Schema, Part 2]

P. Biron et al., "XML Schema Part 2: Datatypes", May 2001. http://www.w3.org/TR/xmlschema-2/

1.6 Non-Normative References

[Addressing-Ext]

J. Alexander et al., "Application Note: Web Services Addressing Endpoint References and Identity", July 2008. http://schemas.xmlsoap.org/ws/2006/02/addressingidentity

[ISIP]

A. Nanda and M. Jones, "Identity Selector Interoperability Profile V1.5", July 2008. http://www.microsoft.com/downloads/details.aspx?FamilyID=b94817fc-3991-4dd0-8e85-b73e626f6764&DisplayLang=en

[ISIP Guide]

Microsoft Corporation and Ping Identity Corporation, "An Implementer's Guide to the Identity Selector Interoperability Profile V1.5", July 2008. http://www.microsoft.com/downloads/details.aspx?FamilyID=b94817fc-3991-4dd0-8e85-b73e626f6764&DisplayLang=en

[ISIP Web Guide]

M. Jones, "A Guide to Using the Identity Selector Interoperability Profile V1.5 within Web Applications and Browsers", July 2008. http://www.microsoft.com/downloads/details.aspx?FamilyID=b94817fc-3991-4dd0-8e85-b73e626f6764&DisplayLang=en

Copyright © OASIS® 2008-2009. All Rights Reserved.

2 Relying Party Interactions

This section defines the constructs used by a Relying Party Web service for specifying and conveying its Security Token requirements to the Service Requester.

2.1 Expressing Token Requirements of Relying Party

A Relying Party specifies its Security Token requirements as part of its Security Policy using the primitives and assertions defined in WS-SecurityPolicy. The primary construct in the Security Policy of the Relying Party used to specify its requirement for a Security Token from an Identity Provider is the `sp:IssuedToken` policy assertion. The basic form of the issued token policy assertion as defined in WS-SecurityPolicy is as follows.

```
<sp:IssuedToken sp:Usage="xs:anyURI" sp:IncludeToken="xs:anyURI"
    xmlns:sp="..." xmlns:wsa="..." xmlns:wsp="..." ...>
  <sp:Issuer>
    wsa:EndpointReference | xs:any
  </sp:Issuer>
  <sp:RequestSecurityTokenTemplate>
    ...
  </sp:RequestSecurityTokenTemplate>
  <wsp:Policy>
    ...
  </wsp:Policy>
  ...
</sp:IssuedToken>
```

The attributes and elements listed in the schema fragment above are described in WS-SecurityPolicy.

The ensuing subsections describe special parameters added by this profile as extensions to the `sp:IssuedToken` policy assertion that convey additional instructions to the Identity Selector available to the Service Requester.

2.1.1 Issuer of Tokens

The `sp:IssuedToken/sp:Issuer` element in an issued token policy specifies the issuer for the requested token. More specifically, it SHOULD contain the endpoint reference of an Identity Provider STS that can issue the requested token.

A Relying Party MUST specify the issuer for a requested token in one of the following ways:

- Indicate a *specific* issuer by specifying the issuer's endpoint as the value of the `sp:Issuer/wsa:Address` element.
- Indicate that the issuer is *unspecified* by omitting the `sp:Issuer` element, which means that the Service Requester should determine the appropriate issuer for the requested token with help from the user if necessary.

When requiring a specific issuer, a Relying Party MAY specify that it will accept self-issued Security Tokens by using the special URI below as the value of the `wsa:Address` element within the endpoint reference for the issuer.

URI:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self
```

Following is an example of using this URI within an issued token policy.

Example:

```
<sp:IssuedToken xmlns:sp="..." xmlns:wsa="..." ...>
  <sp:Issuer>
    <wsa:Address>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
      http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self
    </wsa:Address>
  </sp:Issuer>
  ...
</sp:IssuedToken>
```

A Relying Party MAY specify the value of the `sp:Issuer/wsa:Address` element in policy as a "logical name" of the token issuer instead of an actual network address where the token is issued. An Identity Selector SHOULD resolve the logical name to an appropriate endpoint for the token issuer by matching the issuer name in Information Cards available to it.

If a Relying Party specifies the token issuer as a network endpoint in policy, then it MUST also specify the location of issuer metadata from where the issuer's policy metadata can be obtained. This is done using the mechanism defined in [WS-Addressing] for embedding metadata within an endpoint reference. The following example shows a token policy where the issuer endpoint and its corresponding metadata location are specified.

Example:

```
<sp:IssuedToken xmlns:sp="..." xmlns:wsa="..." xmlns:wsx="..." ...>
  <sp:Issuer>
    <wsa:Address>http://contoso.com/sts</wsa:Address>
    <wsa:Metadata>
      <wsx:Metadata>
        <wsx:MetadataSection
            Dialect="http://schemas.xmlsoap.org/ws/2004/09/mex">
          <wsx:MetadataReference>
            <wsa:Address>https://contoso.com/sts/mex</wsa:Address>
          </wsx:MetadataReference>
        </wsx:MetadataSection>
      </wsx:Metadata>
    </wsa:Metadata>
  </sp:Issuer>
  ...
</sp:IssuedToken>
```

2.1.2 Type of Proof Key in Issued Tokens

If no explicit key type is specified by the Relying Party, then an Identity Selector SHOULD request an asymmetric key token from the Identity Provider to maximize user privacy and security.

A Relying Party MAY explicitly request the use of an *asymmetric* or *symmetric* key in the requested token by using the `wst:KeyType` element within its issued token policy assertion. The key type URIs are defined in [WS-Trust]. The following example illustrates the use of this element in the Relying Party's Security Policy to request a symmetric key in the issued token.

Example:

```
<sp:IssuedToken xmlns:sp="..." xmlns:wst="...">
  <sp:RequestSecurityTokenTemplate>
    <wst:KeyType>
      http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey
    </wst:KeyType>
  </sp:RequestSecurityTokenTemplate>
</sp:IssuedToken>
```

2.1.3 Claims in Issued Tokens

The claims requirement of a Relying Party can be expressed in its token policy by using the optional `wst:Claims` parameter defined in [WS-Trust 1.2] and [WS-Trust 1.3]. However, the `wst:Claims` parameter has an open content model. This profile defines the `ic:ClaimType` element for use as a child of the `wst:Claims` element. A Relying Party MAY use this element to specify an individual claim type

Copyright © OASIS® 2008-2009. All Rights Reserved.

requested. Further, each requested claim MAY be specified as being *required* or *optional*. Multiple `ic:ClaimType` elements can be included to specify multiple claim types requested.

The outline for the `ic:ClaimType` element is as follows:

Syntax:

```
<ic:ClaimType Uri="xs:anyURI" Optional="xs:boolean"? xmlns:ic="..." /> *
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:ClaimType

Indicates the requested claim type.

/ic:ClaimType/@Uri

The unique identifier of the requested claim type.

/ic:ClaimType/@Optional

Indicates if the claim can be absent in the Security Token. By default, any requested claim type is a required claim and MUST be present in the issued Security Token.

Two `<ic:ClaimType>` elements refer to the same claim type if and only if the values of their XML attribute named `Uri` are equal in a case-sensitive string comparison.

When the `ic:ClaimType` element is used within the `wst:Claims` parameter in a token policy to specify claims requirement, the `wst:Dialect` attribute on the `wst:Claims` element MUST be qualified with the URI value below.

Dialect URI:

```
http://schemas.xmlsoap.org/ws/2005/05/identity
```

The above dialect URI value indicates that the specified claim elements are to be processed according to this profile.

Following is an example of using this assertion within an issued token policy to require two claim types where one claim type is optional.

Example:

```
<sp:IssuedToken xmlns:sp="..." xmlns:wst="..." xmlns:ic="..." ...>
  ...
  <sp:RequestSecurityTokenTemplate>
    ...
    <wst:Claims
        Dialect="http://schemas.xmlsoap.org/ws/2005/05/identity">
      <ic:ClaimType
          Uri="http://.../ws/2005/05/identity/claims/givenname"/>
      <ic:ClaimType
          Uri="http://.../ws/2005/05/identity/claims/surname"
          Optional="true" />
    </wst:Claims>
  </sp:RequestSecurityTokenTemplate>
  ...
</sp:IssuedToken>
```

This profile also defines a standard set of claim types for common personal information about users that may be requested by Relying Party Web services in Security Tokens and supported by any Identity Provider. These standard claim types are defined in Section 7.5.

2.2 Expressing Privacy Policy of Relying Party

A Relying Party Web service SHOULD publish its “Privacy Policy”. Users might decide to release tokens and interact further with that service based on its Privacy Policy. No assumptions are made regarding the

Copyright © OASIS® 2008-2009. All Rights Reserved.

format and content of the Privacy Policy and an Identity Selector is NOT REQUIRED to parse, interpret or act on the Privacy Policy programmatically.

To express the location of its privacy statement, a Web service MUST use the optional policy assertion `ic:PrivacyNotice` defined below:

Syntax:

```
<ic:PrivacyNotice Version="xs:unsignedInt"? xmlns:ic="...">
  xs:anyURI
</ic:PrivacyNotice>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:PrivacyNotice

This element is used to express the location of the privacy statement of a Web service.

/ic:PrivacyNotice/@Version

This optional attribute provides a version number for the privacy statement allowing changes in its content to be reflected as a change in the version number. If present, it MUST have a minimum value of 1.

Following is an example of using this policy element to express the location of the privacy statement of a Web service.

Example:

```
<wsp:Policy xmlns:wsp="..." xmlns:ic="...">
  ...
  <ic:PrivacyNotice Version="1">
    http://www.contoso.com/privacy
  </ic:PrivacyNotice>
  ...
</wsp:Policy>
```

An Identity Selector MUST be able to accept a privacy statement location specified as an URL using the [HTTP] scheme (as illustrated above) or the [HTTPS] scheme.

Because the Privacy Policy assertion points to a “privacy statement” that applies to a service endpoint, the assertion MUST apply to [Endpoint Policy Subject]. In other words, a policy expression containing the Privacy Policy assertion MUST be attached to a `wsdl:binding` in the metadata for the service.

Further, when an Identity Selector can only render the privacy statement document in a limited number of document formats (media types), it MAY use the HTTP request-header field “Accept” in its HTTP GET request to specify the media-types it can accept. For example, the following request-header specifies that the client will accept the Privacy Policy only as a plain text or a HTML document.

```
Accept: text/plain, text/html
```

Similarly, if an Identity Selector wants to obtain the privacy statement in a specific language, it MAY use the HTTP request-header field “Accept-Language” in its HTTP GET request to specify the languages it is willing to accept. For example, the following request-header specifies that the client will accept the Privacy Policy only in Danish.

```
Accept-Language: da
```

A Web service, however, is NOT REQUIRED to be able to fulfill the document format and language requests of an Identity Selector. It MAY publish its privacy statement in a fixed set of document formats and languages.

Copyright © OASIS® 2008-2009. All Rights Reserved.

2.3 Employing Relying Party STSs

The Security Policy of a Relying Party MAY require that an issued token be obtained from a Relying Party STS. This can create a chain of STSs. The Identity Selector MUST follow the RP/STS chain, contacting each referenced STS, resolving its Policy statements and continuing to the STS it refers to.

When following a chain of STSs, when an STS with an `ic:RequireFederatedIdentityProvisioning` declaration is encountered as per Section 3.2.1, this informs the Identity Selector that the STS is an IP/STS and therefore ends the STS chain, rather than a member of the RP/STS chain. Furthermore, if an RP or RP/STS provides an incomplete Security Policy, such as no issuer or no required claims, the Identity Selector MUST be invoked so a card and requested claims can be selected by the user, enabling a Request for Security Token (RST) to be constructed and sent to the selected IP/STS.

The RP/STS's Policy is used for card matching. If the RP/STS requests a private personal identifier (PPID) claim (see Section 7.5.14), the RP/STS's certificate is used for calculating PPID, Signing Key, and Client Pseudonym (see Section 3.3.4) values – not the certificate of the Relying Party. This enables a single RP/STS to service multiple Relying Parties while always receiving the same PPID value for a given user from the Identity Selector.

Identity Selectors MUST enable users to make Relying Party trust decisions based on the identity of the Relying Party, possibly including displaying attributes from its certificate. By trusting the RP, the user is implicitly trusting the chain of RP/STSs that the RP employs.

Each RP/STS endpoint MUST provide a certificate. This certificate MAY be communicated either via Transport (such as HTTPS) or Message (such as WS-Security) Security. If Message Security is employed, transports not providing security (such as HTTP) MAY be used.

Like IP/STSs, RP/STSs publish endpoint metadata. This metadata MAY be retrieved via either WS-MetadataExchange or HTTPS GET in the same manner that IP/STS metadata can be, as described in Section 3.1.1.2.

Like IP/STSs, no changes to the syntax used to specify metadata locations occurs when RP/STS metadata is published by the Relying Party STS as a page retrievable using HTTPS GET. Relying Parties and Identity Providers MAY consequently support either or both retrieval methods for the same metadata addresses.

3 Identity Provider Interactions

This section defines the constructs used by an Identity Selector for interacting with an Identity Provider to obtain Information Cards, and to request and obtain Security Tokens.

3.1 Information Card

An Information Card represents a Digital Identity of a Subject that can be issued by an Identity Provider. It is an artifact containing metadata that represents the token issuance relationship between an Identity Provider and a Subject, and provides a visual representation of the Digital Identity. Multiple Digital Identities for a Subject from the same Identity Provider are represented by different Information Cards. Subjects may obtain an Information Card from an Identity Provider, and may have a collection of Information Cards from various Identity Providers.

3.1.1 Information Card Format

An Information Card is represented as a signed XML document that is issued by an Identity Provider. The XML schema for an Information Card is defined below:

Syntax:

```
<ic:InformationCard xml:lang="xs:language"
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
    xmlns:ic="..." xmlns:ic07="..." ...>
  <ic:InformationCardReference> ... </ic:InformationCardReference>
  <ic:CardName> xs:string </ic:CardName> ?
  <ic:CardImage MimeType="xs:string"> xs:base64Binary </ic:CardImage> ?
  <ic:Issuer> xs:anyURI </ic:Issuer>
  <ic:TimeIssued> xs:dateTime </ic:TimeIssued>
  <ic:TimeExpires> xs:dateTime </ic:TimeExpires> ?
  <ic:TokenServiceList> ... </ic:TokenServiceList>
  <ic:SupportedTokenTypeList> ... </ic:SupportedTokenTypeList> ?
  <ic:SupportedClaimTypeList> ... </ic:SupportedClaimTypeList> ?
  <ic:RequireAppliesTo ...> ... </ic:RequireAppliesTo> ?
  <ic:PrivacyNotice ...> ... </ic:PrivacyNotice> ?
  <ic07:RequireStrongRecipientIdentity /> ?
  <ic07:IssuerInformation> ... </ic07:IssuerInformation> *
  ...
</ic:InformationCard>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:InformationCard

An Information Card issued by an Identity Provider.

/ic:InformationCard/@xml:lang

A required language identifier, using the language codes specified in [RFC 3066], in which the content of localizable elements have been localized.

/ic:InformationCard/ic:InformationCardReference

This required element provides a specific reference for the Information Card by which it can be uniquely identified within the scope of an issuer. This reference MUST be included by an Identity Selector in all token requests sent to the Identity Provider based on that Information Card. The detailed schema of this element is defined in Section 3.1.1.1.

/ic:InformationCard/ic:CardName

This optional element provides a friendly textual name for the issued Information Card. The content of this element MAY be localized in a specific language.

/ic:InformationCard/ic:CardImage

This optional element contains a base64 encoded inline image that provides a graphical image for the issued Information Card. It SHOULD contain an image within the size range of 60 pixels wide by 40 pixels high and 240 pixels wide by 160 pixels high. It is RECOMMENDED that the image have an aspect ratio of 3:2 and the image size be 120 by 80 pixels.

/ic:InformationCard/ic:CardImage/@MimeType

This required attribute provides a MIME type specifying the format of the included card image. This value MUST be one of the five image formats: `image/jpeg`, `image/gif`, `image/bmp`, `image/png`, or `image/tiff`.

/ic:InformationCard/ic:Issuer

This required element provides a logical name for the issuer of the Information Card. If a Relying Party specifies a token issuer by its logical name, then the content of this element MUST be used to match the requested token issuer with an Information Card.

/ic:InformationCard/ic:TimeIssued

This required element provides the date and time when the Information Card was issued.

/ic:InformationCard/ic:TimeExpires

This optional element provides the date and time after which the Information Card SHOULD be treated as expired and invalid.

Copyright © OASIS® 2008-2009. All Rights Reserved.

/ic:InformationCard/ic:TokenServiceList

This required element provides an ordered list of Security Token Service (IP/STS) endpoints, and corresponding credential descriptors (implying the REQUIRED authentication mechanisms), where tokens can be requested. Each service endpoint MUST be tried in order by the Service Requester when requesting tokens.

/ic:InformationCard/ic:SupportedTokenTypeList

This optional element contains the list of token types that are offered by the Identity Provider.

/ic:InformationCard/ic:SupportedClaimTypeList

This optional element contains the list of claim types that are offered by the Identity Provider.

/ic:InformationCard/ic:RequireAppliesTo

This optional element indicates that token requests MUST include information identifying the Relying Party where the issued token will be used. The Relying Party information MUST be included as the content of a `wsp:AppliesTo` element in the token request.

/ic:InformationCard/ic:PrivacyNotice

This optional element provides the location of the privacy statement of the Identity Provider.

/ic:InformationCard/ic07:RequireStrongRecipientIdentity

This optional element informs the Identity Selector that it MUST only allow the card to be used at a Relying Party that presents a cryptographically protected identity, for example, an X.509v3 certificate.

/ic:InformationCard/ic07:IssuerInformation

This optional element provides information from the card issuer about the card that can be displayed by the Identity Selector user interface.

.../ic:InformationCard/@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:InformationCard` element of an `ic:RoamingStore` when representing the card in the Information Cards Transfer Format in Section 6.1.

.../ic:InformationCard/{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:InformationCard` element of an `ic:RoamingStore` when representing the card in the Information Cards Transfer Format in Section 6.1.

3.1.1.1 Information Card Reference

Every Information Card issued by an Identity Provider MUST have a unique reference by which it can be identified within the scope of the Identity Provider. This reference is included in all token requests sent to the Identity Provider based on that Information Card.

The card reference MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:InformationCardReference xmlns:ic="...">
  <ic:CardId> xs:anyURI </ic:CardId>
  <ic:CardVersion> xs:unsignedInt </ic:CardVersion>
</ic:InformationCardReference>
```

The following describes the attributes and elements listed in the schema outlined above:

Copyright © OASIS® 2008-2009. All Rights Reserved.

.../ic:InformationCardReference

A specific reference for an Information Card.

.../ic:InformationCardReference/ic:CardId

This required element provides a unique identifier in the form of a URI for the specific Information Card. The identifier provider MUST be able to identify the specific Information Card based on this identifier.

.../ic:InformationCardReference/ic:CardVersion

This required element provides a versioning epoch for the Information Card issuance infrastructure used by the Identity Provider. The minimum value for this field MUST be 1. Note that it is possible to include version information in CardId as it is a URI, and can have hierarchical content. However, it is specified as a separate value to allow the Identity Provider to change its issuance infrastructure, and thus its versioning epoch, independently without changing the CardId of all issued Information Cards. For example, when an Identity Provider makes a change to the supported claim types or any other policy pertaining to the issued cards, the version number allows the Identity Provider to determine if the Information Card needs to be refreshed. The version number is assumed to be monotonically increasing. If two Information Cards have the same CardId value but different CardVersion values, then the one with a higher numerical CardVersion value SHOULD be treated as being more up-to-date.

3.1.1.2 Token Service Endpoints and Authentication Mechanisms

Every Information Card issued by an Identity Provider MUST include an ordered list of IP/STS endpoints, and the corresponding credential type to be used, for requesting tokens. The list MUST be in a decreasing order of preference. Identity Selectors SHOULD attempt to use the endpoints in the order listed, using the first endpoint in the list for which the metadata is retrievable and the endpoint is reachable. For each endpoint, the credential type implicitly determines the authentication mechanism to be used. Each credential descriptor is personalized for the user to allow an Identity Selector to automatically locate the credential once the user has selected an Information Card.

Further, each IP/STS endpoint reference in the Information Card MUST include the Security Policy metadata for that endpoint. The policy metadata MAY be specified as a metadata location within the IP/STS endpoint reference. If a metadata location URL is specified, it MUST use the [HTTPS] transport. An Identity Selector MAY retrieve the Security Policy it will use to communicate with the IP/STS from that metadata location using the mechanism specified in [WS-MetadataExchange].

The ordered list of token service endpoints MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:TokenServiceList xmlns:ic="..." xmlns:wsa="...">
 (<ic:TokenService>
   <wsa:EndpointReference> ... </wsa:EndpointReference>
   <ic:UserCredential>
    <ic:DisplayCredentialHint> xs:string </ic:DisplayCredentialHint> ?
   (
    <ic:UsernamePasswordCredential>...</ic:UsernamePasswordCredential> |
    <ic:KerberosV5Credential>...</ic:KerberosV5Credential> |
    <ic:X509V3Credential>...</ic:X509V3Credential> |
    <ic:SelfIssuedCredential>...</ic:SelfIssuedCredential> | ...
   )
   </ic:UserCredential>
  </ic:TokenService>) +
</ic:TokenServiceList>
```

The following describes the attributes and elements listed in the schema outlined above:

Copyright © OASIS® 2008-2009. All Rights Reserved.

572 *.../ic:TokenServiceList*

573 This required element provides an ordered list of Security Token Service endpoints (in decreasing
574 order of preference), and the corresponding credential types, for requesting tokens. Each service
575 endpoint MUST be tried in order by a Service Requester.

576 *.../ic:TokenServiceList/ic:TokenService*

577 This required element describes a single token issuing endpoint.

578 *.../ic:TokenServiceList/ic:TokenService/wsa:EndpointReference*

579 This required element provides the endpoint reference for a single token issuing endpoint. For the
580 Self-issued Identity Provider, the special address value defined in Section 2.1.1 MAY be used.
581 The `wsai:Identity` extension element (see Section 12) for endpoint references MAY be used
582 to include the protection token for this endpoint to secure communications with it.

583 *.../ic:TokenServiceList/ic:TokenService/ic:UserCredential*

584 This required element indicates the credential type to use to authenticate to the token issuing
585 endpoint.

586 *.../ic:TokenServiceList/ic:TokenService/ic:UserCredential/ic:DisplayCredentialHint*

587 This optional element provides a hint (string) to be displayed to the user to prompt for the correct
588 credential (e.g. a hint to insert the right smart card). The content of this element MAY be localized
589 in a specific language.

590 *.../ic:TokenServiceList/ic:TokenService/ic:UserCredential/<credential descriptor>*

591 This required element provides an unambiguous descriptor for the credential to use for
592 authenticating to the token issuing endpoint. The schema to describe the credential is specific to
593 each credential type. This profile defines the schema elements
594 `ic:UsernamePasswordCredential`, `ic:KerberosV5Credential`,
595 `ic:X509V3Credential` or `ic:SelfIssuedCredential` later in Section 4 corresponding to
596 username/password, Kerberos v5, X.509v3 certificate and self-issued token based credential
597 types. Other credential types MAY be introduced via the extensibility point defined in the schema
598 within this element.

599 Alternatively, Identity Providers MAY publish metadata for Information Cards as WSDL documents that
600 can be retrieved by Identity Selectors via HTTPS GET operations on URLs using the HTTPS scheme. An
601 endpoint's metadata URL is communicated to Identity Selectors in a token service
602 `wsx:MetadataReference` element in an Information Card using exactly the same syntax as when WS-
603 MetadataExchange is employed to retrieve the metadata. No change occurs in the card.

604 The metadata documents published via HTTPS GET SHOULD contain the WSDL for the endpoint as the
605 top-level element of the document without any SOAP or WS-MetadataExchange elements enclosing it.

606 Identity Providers MAY publish endpoint metadata via both the HTTPS GET and WS-MetadataExchange
607 methods at the same metadata location. If they publish the metadata via multiple mechanisms, the
608 metadata delivered via both mechanisms SHOULD be the same. Likewise, Identity Selectors MAY
609 attempt to retrieve metadata via multiple mechanisms, either in sequence or in parallel.

610 The following example illustrates an Identity Provider with two endpoints for its IP/STS, one requiring
611 Kerberos (higher priority) and the other requiring username/password (lower priority) as its authentication
612 mechanism. Further, each endpoint also includes its policy metadata location as a URL using the
613 [HTTPS] scheme.

614 *Example:*

```
615 <ic:TokenServiceList xmlns:ic="..." xmlns:wsa="..." xmlns:wsai="..."
616     xmlns:wsx="...">
617   <ic:TokenService>
618     <wsa:EndpointReference>
619       <wsa:Address>http://contoso.com/sts/kerb</wsa:Address>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
      <wsai:Identity>
        <wsai:Spn>host/corp-sts.contoso.com</wsai:Spn>
      </wsai:Identity>
      <wsa:Metadata>
        <wsx:Metadata>
          <wsx:MetadataSection
              Dialect="http://schemas.xmlsoap.org/ws/2004/09/mex">
            <wsx:MetadataReference>
              <wsa:Address>https://contoso.com/sts/kerb/mex</wsa:Address>
            </wsx:MetadataReference>
          </wsx:MetadataSection>
        </wsx:Metadata>
      </wsa:Metadata>
    </wsa:EndpointReference>
    <ic:UserCredential>
      <ic:KerberosV5Credential />
    </ic:UserCredential>
  </ic:TokenService>
  <ic:TokenService>
    <wsa:EndpointReference>
      <wsa:Address>http://contoso.com/sts/pwd</wsa:Address>
      <wsa:Metadata>
        <wsx:Metadata>
          <wsx:MetadataSection
              Dialect="http://schemas.xmlsoap.org/ws/2004/09/mex">
            <wsx:MetadataReference>
              <wsa:Address>https://contoso.com/sts/pwd/mex</wsa:Address>
            </wsx:MetadataReference>
          </wsx:MetadataSection>
        </wsx:Metadata>
      </wsa:Metadata>
    </wsa:EndpointReference>
    <ic:UserCredential>
      <ic:UsernamePasswordCredential>
        <ic:Username>Zoe</ic:Username>
      </ic:UsernamePasswordCredential>
    </ic:UserCredential>
  </ic:TokenService>
</ic:TokenServiceList>
```

3.1.1.3 Token Types Offered

Every Information Card issued by an Identity Provider SHOULD include an unordered list of token types that can be issued by the Identity Provider. The set of token types offered by the Identity Provider MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:SupportedTokenTypeList xmlns:ic="..." xmlns:wst="...">
  <wst:TokenType> xs:anyURI </wst:TokenType> +
</ic:SupportedTokenTypeList>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:SupportedTokenTypeList

This optional element contains the set of token types offered by the Identity Provider.

.../ic:SupportedTokenTypeList/wst:TokenType

This required element indicates an individual token type that is offered.

The following example illustrates an Identity Provider that offers both SAML 1.1 and SAML 2.0 tokens.

Example:

```
<ic:SupportedTokenTypeList xmlns:ic="..." xmlns:wst="...">
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
  <wst:TokenType>urn:oasis:names:tc:SAML:1.0:assertion</wst:TokenType>
  <wst:TokenType>urn:oasis:names:tc:SAML:2.0:assertion</wst:TokenType>
</ic:SupportedTokenTypeList>
```

3.1.1.4 Claim Types Offered

Every Information Card issued by an Identity Provider SHOULD include an unordered list of claim types that can be issued by the Identity Provider. The set of claim types offered by the Identity Provider MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:SupportedClaimTypeList xmlns:ic="...">
  (<ic:SupportedClaimType Uri="xs:anyURI">
    <ic:DisplayTag> xs:string </ic:DisplayTag> ?
    <ic:Description> xs:string </ic:Description> ?
  </ic:SupportedClaimType>) +
</ic:SupportedClaimTypeList>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:SupportedClaimTypeList

This optional element contains the set of claim types offered by the Identity Provider.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType

This required element indicates an individual claim type that is offered.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType/@Uri

This required attribute provides the unique identifier (URI) of this individual claim type offered.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType/ic:DisplayTag

This optional element provides a friendly name for this individual claim type. The content of this element MAY be localized in a specific language.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType/ic:Description

This optional element provides a description of the semantics for this individual claim type. The content of this element MAY be localized in a specific language.

The following example illustrates an Identity Provider that offers two claim types.

Example:

```
<ic:SupportedClaimTypeList xmlns:ic="...">
  <ic:SupportedClaimType Uri=".../ws/2005/05/identity/claims/givenname">
    <ic:DisplayTag>Given Name</DisplayTag>
  </ic:SupportedClaimType>
  <ic:SupportedClaimType Uri=".../ws/2005/05/identity/claims/surname">
    <ic:DisplayTag>Last Name</DisplayTag>
  </ic:SupportedClaimType>
</ic:SupportedClaimTypeList>
```

3.1.1.5 Requiring Token Scope Information

An Identity Selector, by default, SHOULD NOT convey information about the Relying Party where an issued token will be used (i.e., target scope) when requesting Security Tokens. This helps safeguard user privacy. However, an Identity Provider MAY override that behavior.

Every Information Card issued by an Identity Provider MAY include a requirement that token requests include token scope information identifying the Relying Party where the token will be used. The requirement to submit token scope information MUST be expressed using the following schema element within an Information Card.

Syntax:

Copyright © OASIS® 2008-2009. All Rights Reserved.

```
<ic:RequireAppliesTo Optional="xs:boolean" xmlns:ic="..." /> ?
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:RequireAppliesTo

This optional element indicates a requirement for a token requester to submit token scope information in the request. Absence of this element in an Information Card means that the token requester MUST NOT submit any token scope information.

.../ic:RequireAppliesTo/@Optional

This optional attribute indicates whether the token scope information is required or is optional by the Identity Provider. An attribute value of “true” indicates that the token scope information is not required, but will be accepted by the Identity Provider if submitted. An attribute value of “false” (default) indicates that the token scope information is required.

The following example illustrates the use of this element.

Example:

```
<ic:RequireAppliesTo Optional="true" xmlns:ic="..." />
```

If token scope information is required by an Identity Provider, an Identity Selector MUST include the Relying Party identity as the content of the `wsp:AppliesTo` element in the token request. The actual behavior of an Identity Selector vis-à-vis the possible requirements that can be expressed by the above element is specified in Section 3.3.3.

3.1.1.6 Privacy Policy Location

Every Information Card issued by an Identity Provider SHOULD include a pointer to the privacy statement of the Identity Provider. The location of the privacy statement MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:PrivacyNotice Version="xs:unsignedInt" xmlns:ic="..." /> ?
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:PrivacyNotice

This optional element provides the location of the privacy statement of the Identity Provider.

.../ic:PrivacyNotice/@Version

This optional attribute indicates a version number that tracks changes in the content of the privacy statement. This field MUST have a minimum value of 1 when present.

The following example illustrates the use of this element.

Example:

```
<ic:PrivacyNotice Version="1" xmlns:ic="...">
  http://www.contoso.com/privacynotice
</ic:PrivacyNotice>
```

An Identity Selector MUST be able to accept a privacy statement location specified as an URL using the [HTTP] scheme (as illustrated above) or the [HTTPS] scheme.

3.1.1.7 Prohibiting Use at Relying Parties Not Identified by a Cryptographically Protected Identity

Information Cards issuers MAY specify that a card MUST NOT be used at Relying Parties that do not present a cryptographically protected identity, such as an X.509v3 certificate. This would typically be done when the issuer determines that the use of HTTP without Message Security would not provide a sufficiently secure environment for the use of the card.

Copyright © OASIS® 2008-2009. All Rights Reserved.

Syntax:

```
<ic07:RequireStrongRecipientIdentity xmlns:ic07="..." /> ?
```

.../ic07:RequireStrongRecipientIdentity

This optional element informs the Identity Selector that it MUST only allow the card to be used at a Relying Party that presents a cryptographically protected identity, such as an X.509v3 certificate.

3.1.1.8 Providing Custom Data to Display with the Card

Card issuers MAY supply a set of information about the card that MAY be displayed by the Identity Selector user interface.

Syntax:

```
<ic07:IssuerInformation xmlns:ic07="...">
  <ic07:IssuerInformationEntry>
    <ic07:EntryName> xs:string </ic07:EntryName>
    <ic07:EntryValue> xs:string </ic07:EntryValue>
  </ic07:IssuerInformationEntry> +
</ic07:IssuerInformation>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic07:IssuerInformation

This optional element provides a set of information from the card issuer about the card that can be displayed by the Identity Selector user interface.

.../ic07:IssuerInformation/IssuerInformationEntry

This required element provides one item of information about the card.

.../ic07:IssuerInformation/IssuerInformationEntry/EntryName

This required element provides the name of one item of information about the card.

.../ic07:IssuerInformation/IssuerInformationEntry/EntryValue

This required element provides the value of one item of information about the card.

The following example illustrates the use of this feature.

Example:

```
<ic07:IssuerInformation xmlns:ic07="...">
  <ic07:IssuerInformationEntry>
    <ic07:EntryName>Customer Service</ic07:EntryName>
    <ic07:EntryValue>+1-800-CONTOSO</ic07:EntryValue>
  </ic07:IssuerInformationEntry>
  <ic07:IssuerInformationEntry>
    <ic07:EntryName>E-mail Contact</ic07:EntryName>
    <ic07:EntryValue>cardhelp@contoso.com</ic07:EntryValue>
  </ic07:IssuerInformationEntry>
</ic07:IssuerInformation>
```

3.1.2 Issuing Information Cards

An Identity Provider can issue Information Cards to its users using any out-of-band mechanism that is mutually suitable.

In order to provide the assurance that an Information Card is indeed issued by the Identity Provider expected by the user, the Information Card MUST be carried inside a digitally signed envelope that is signed by the Identity Provider. For this, the “enveloping signature” construct (see [XMLDSIG]) MUST be used where the Information Card is included in the `ds:Object` element. The signature on the digitally

Copyright © OASIS® 2008-2009. All Rights Reserved.

signed envelope provides data origin authentication assuring the user that it came from the right Identity Provider.

The specific profile of XML digital signatures [XMLDSIG] that is RECOMMENDED for signing the envelope carrying the Information Card is as follows. Usage of other algorithms is not described.

- Use *enveloping signature* format when signing the Information Card XML document.
- Use a single `ds:Object` element within the signature to hold the `ic:InformationCard` element that represents the issued Information Card. The `ds:Object/@Id` attribute provides a convenient way for referencing the Information Card from the `ds:SignedInfo/ds:Reference` element within the signature.
- Use RSA signing and verification with the algorithm identifier given by the URI *http://www.w3.org/2000/09/xmldsig#rsa-sha1*.
- Use exclusive canonicalization with the algorithm identifier given by the URI *http://www.w3.org/2001/10/xml-exc-c14n#*.
- Use SHA1 digest method for the data elements being signed with the algorithm identifier *http://www.w3.org/2000/09/xmldsig#sha1*.
- There MUST NOT be any other transforms used in the enveloping signature for the Information Card other than the ones listed above.
- The `ds:KeyInfo` element MUST be present in the signature carrying the signing key information in the form of an X.509 v3 certificate or a X.509 v3 certificate chain specified as one or more `ds:X509Certificate` elements within a `ds:X509Data` element.

The following example shows an enveloping signature carrying an Information Card that is signed by the Identity Provider using the format outlined above. Note that whitespace (newline and space character) is included in the example only to improve readability; they might not be present in an actual implementation.

Example:

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    <SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <Reference URI="#_Object_InformationCard">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue> ... </DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue> ... </SignatureValue>
  <KeyInfo>
    <X509Data>
      <X509Certificate> ... </X509Certificate>
    </X509Data>
  </KeyInfo>
  <Object Id="_Object_InformationCard">
    <ic:InformationCard
        xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
        xml:lang="en-us">
      [Information Card content]
    </ic:InformationCard>
  </Object>
</Signature>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

An Identity Selector MUST verify the enveloping signature. The `ic:InformationCard` element can then be extracted and stored in the Information Card collection.

3.2 Identity Provider Policy

This section specifies additional policy elements and requirements introduced by this profile for an IP/STS policy metadata.

3.2.1 Require Information Card Provisioning

In the Information Card Model, an Identity Provider requires provisioning in the form of an Information Card issued by it which represents the provisioned identity of the user. In order to enable an Identity Selector to learn that such pre-provisioning is necessary before token requests can be made, the Identity Provider MUST provide an indication in its policy.

An Identity Provider issuing Information Cards MUST specify this provisioning requirement in its policy using the following schema element.

Syntax:

```
<ic:RequireFederatedIdentityProvisioning xmlns:ic="..." />
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:RequireFederatedIdentityProvisioning

This element indicates a requirement that one or more Information Cards, representing identities that can be federated, MUST be pre-provisioned before token requests can be made to the Identity Provider.

The following example illustrates the use of this policy element.

Example:

```
<wsp:Policy xmlns:wsp="..." xmlns:ic="..." xmlns:sp="...">
  ...
  <ic:RequireFederatedIdentityProvisioning />
  <sp:SymmetricBinding>
    ...
  </sp:SymmetricBinding>
  ...
</wsp:Policy>
```

3.2.2 Policy Metadata Location

In the Information Card Model, an Identity Provider MUST make the Security Policy metadata for its IP/STS endpoints available. If a metadata location is used for this purpose, the location URL MUST use the [HTTPS] scheme. An Identity Selector MAY retrieve the Security Policy it will use to communicate with the IP/STS from that metadata location using the mechanism specified in [WS-MetadataExchange].

3.3 Token Request and Response

For any given Information Card, an Identity Selector can obtain a Security Token from the IP/STS for that Information Card. Tokens MUST be requested using the “Issuance Binding” mechanism described in [WS-Trust 1.2] and [WS-Trust 1.3]. This section specifies additional constraints and extensions to the token request and response messages between the Identity Selector and the IP/STS.

The WS-Trust protocol requires that a token request be submitted by using the `wst:RequestSecurityToken` element in the request message, and that a token response be sent using the `wst:RequestSecurityTokenResponse` element in the response message. This profile refers to the “Request Security Token” message as RST and the “Request Security Token Response” message as RSTR in short.

Copyright © OASIS® 2008-2009. All Rights Reserved.

The WS-Trust protocol allows for a token response to provide multiple tokens by using the `wst:RequestSecurityTokenResponseCollection` element in the response message. This profile, however, requires that an Identity Provider MUST NOT use the `wst:RequestSecurityTokenResponseCollection` element in the response. The token response MUST consist of a single `wst:RequestSecurityTokenResponse` element.

3.3.1 Information Card Reference

When requesting a Security Token from the IP/STS, an Identity Selector MUST include the Information Card reference in the body of the RST message as a top-level element information item. The `ic:InformationCardReference` element in the Information Card, including all of its [children], [attributes] and [in-scope namespaces], MUST be copied as an immediate child of the RST element in the message as follows.

The following example illustrates the Information Card reference included in a RST message.

Example:

```
<wst:RequestSecurityToken xmlns:wst="..." xmlns:ic="...">
  ...
  <ic:InformationCardReference>
    <ic:CardId>http://xyz.com/CardId/d795621fa01d454285f9</ic:CardId>
    <ic:CardVersion>1</ic:CardVersion>
  </ic:InformationCardReference>
  ...
</wst:RequestSecurityToken>
```

The IP/STS MAY fault with `ic:InformationCardRefreshRequired` to signal to the Service Requester that the Information Card needs to be refreshed.

3.3.2 Claims and Other Token Parameters

A Relying Party's requirements of claims and other token parameters are expressed in its policy using the `sp:RequestSecurityTokenTemplate` parameter within the `sp:IssuedToken` policy assertion (see Section 2.1). If all token parameters are acceptable to the Identity Selector, it MUST copy the content of this element (i.e. all of its [children] elements) into the body of the RST message as top-level element information items. However, if optional claims are requested by the Relying Party, requests for optional claims not selected by the user MUST NOT be copied into the RST message.

3.3.3 Token Scope

The WS-Trust protocol allows a token requester to indicate the target where the issued token will be used (i.e., token scope) by using the optional element `wsp:AppliesTo` in the RST message. By default, an Identity Selector SHOULD NOT send token scope information to the Identity Provider in token requests to protect user privacy. In other words, the element `wsp:AppliesTo` is absent in the RST message.

However, if the Identity Provider requires it (see the modes of the `ic:RequireAppliesTo` element described in Section 3.1.1.5), or if the Relying Party's token policy includes the `wsp:AppliesTo` element in the `sp:RequestSecurityTokenTemplate` parameter, then an Identity Selector MUST include token scope information in its token request as per the behavior summarized in the following table.

<RequireAppliesTo> mode in Information Card	*<AppliesTo> element present in RP policy*	*Resulting behavior of Identity Selector*
Mandatory	Yes	Send <AppliesTo> value from RP policy in token request to IP.
Mandatory	No	Send the RP endpoint to which token will be sent as the value of

Copyright © OASIS® 2008-2009. All Rights Reserved.

		<AppliesTo> in token request to IP.
Optional	Yes	Send <AppliesTo> value from RP policy in token request to IP.
Optional	No	Do not send <AppliesTo> in token request to IP.
Not present	Yes	Fail
Not present	No	Do not send <AppliesTo> in token request to IP.

The following example illustrates the token scope information included in a RST message when it is sent to the Identity Provider.

Example:

```
<wst:RequestSecurityToken xmlns:wst="..." xmlns:wsp="..." xmlns:wsa="..."
    xmlns:wsai="..." xmlns:ds="...">
  <wsp:AppliesTo>
    <wsa:EndpointReference>
      <wsa:Address>http://ip.fabrikam.com</wsa:Address>
      <wsai:Identity>
        <ds:KeyInfo>
          <ds:X509Data>
            <ds:X509Certificate>...</ds:X509Certificate>
          </ds:X509Data>
        </ds:KeyInfo>
      </wsai:Identity>
    </wsa:EndpointReference>
  </wsp:AppliesTo>
  ...
</wst:RequestSecurityToken>
```

3.3.4 Client Pseudonym

A private personal identifier (PPID) claim, defined in Section 7.5.14, identifies a Subject to a Relying Party in a way such that a Subject's PPID at one Relying Party cannot be correlated with the Subject's PPID at another Relying Party. If an Identity Provider offers the PPID claim type then it MUST generate values for the claim that have this prescribed privacy characteristic using data present in the RST request.

When a Relying Party requests a PPID claim, an Identity Selector MUST provide a Client Pseudonym value via an `ic:PPID` element in the RST request that can be used by the IP/STS as input when computing the PPID claim value in the issued token. The Client Pseudonym SHOULD be produced as described in Section 3.3.4.1. It is RECOMMENDED that the IP/STS combine this Client Pseudonym value with information specific to the entity to which the card was issued as well as a secret known only by the IP/STS and pass the combination through a cryptographically non-invertible function, such as a cryptographic hash function, to generate the PPID claim value sent in the token. Alternatively, when target scope information is sent in the token request using the `wsp:AppliesTo` element, the IP/STS MAY instead choose to use that information to generate an appropriate PPID value.

When Client Pseudonym information is included by an Identity Selector in a token request, it MUST be sent using the following schema element.

Syntax:

```
<ic:ClientPseudonym xmlns:ic="...">
  <ic:PPID> xs:base64Binary </ic:PPID>
</ic:ClientPseudonym>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

The following describes the attributes and elements listed in the schema outlined above:

.../ic:ClientPseudonym

This optional top-level element contains the Client Pseudonym information item.

.../ic:ClientPseudonym/ic:PPID

This optional element contains the Client Pseudonym value that the client has submitted for use in computing the PPID claim value for the issued token. The IP/STS MAY use this value as the input (a seed) to a custom cryptographically non-invertible function, with the result used as the PPID claim value in the issued token.

The following example illustrates the Client Pseudonym information sent in a RST message.

Example:

```
<wst:RequestSecurityToken xmlns:wst="..." xmlns:ic="...">
  <ic:ClientPseudonym>
    <ic:PPID>MIIEZzCCA9CgAwIBAgIQEmtJZc0=</ic:PPID>
  </ic:ClientPseudonym >
  ...
</wst:RequestSecurityToken>
```

When the target scope information is not sent in the token request to an IP/STS, the Identity Provider MUST NOT record any Client Pseudonym values included in the RST message. It likewise MUST NOT record the PPID claim value that it generates.

3.3.4.1 PPID

When a token request for a PPID claim is sent to an IP/STS, an Identity Selector SHOULD compute the Client Pseudonym PPID information it sends in the RST message as follows:

- Construct the *RP PPID Seed* as described in Section 7.6.1.
- Decode the base64 encoded value of the `ic:HashSalt` element of the Information Card (see Section 6.1) to obtain *SaltBytes*.
- Decode the base64 encoded value of the `ic:MasterKey` element of the Information Card (see Section 6.1) to obtain *MasterKeyBytes*.
- Hash the concatenation of *MasterKeyBytes*, *RP PPID Seed*, and *SaltBytes* using the SHA256 hash function to obtain the Client Pseudonym PPID value.

  Client Pseudonym PPID = SHA256 (MasterKeyBytes + RP PPID Seed + SaltBytes)
- Convert *Client Pseudonym PPID* to a base64 encoded string and send as the value of the `ic:PPID` element in the RST request.

3.3.5 Proof Key for Issued Token

An issued token can have a *symmetric* proof key (symmetric key token), an *asymmetric* proof key (asymmetric key token), or *no* proof key (bearer token). If no key type is specified in the Relying Party policy, then an Identity Selector SHOULD request an asymmetric key token from the IP/STS by default.

The optional `wst:KeyType` element in the RST request indicates the type of proof key desired in the issued Security Token. The IP/STS MAY return the proof key and/or entropy towards the proof key in the RSTR response. This section describes the behaviors for how each proof key type is requested, who contributes entropy, and how the proof key is computed and returned.

3.3.5.1 Symmetric Proof Key

When requesting a symmetric key token, an Identity Selector MUST submit entropy towards the proof key by augmenting the RST request message as follows:

Copyright © OASIS® 2008-2009. All Rights Reserved.

- The RST SHOULD include a `wst:KeyType` element with one of the two following URI values, depending upon the version of WS-Trust being used:

  http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey

  http://docs.oasis-open.org/ws-sx/ws-trust/200512/SymmetricKey

- The RST MUST include a `wst:BinarySecret` element inside a `wst:Entropy` element containing client-side entropy to be used as partial key material. The entropy is conveyed as raw base64 encoded bits.

The size of the submitted entropy SHOULD be equal to the key size requested in the Relying Party policy. If no key size is specified by the Relying Party, then an Identity Selector SHOULD request a key at least 256-bits in size, and submit an entropy of equal size to the IP/STS.

Following is a sample RST request fragment that illustrates a symmetric key token request.

Example:

```
<wst:RequestSecurityToken xmlns:wst="...">
  ...
  <wst:KeyType>
    http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey
  </wst:KeyType>
  <wst:KeySize>256</wst:KeySize>
  <wst:Entropy>
    <wst:BinarySecret>mQlxWxEiKOcUfnHgQpylcD7LYSkJplpE=</wst:BinarySecret>
  </wst:Entropy>
</wst:RequestSecurityToken>
```

When processing the token request, the IP/STS MAY:

a) accept the client entropy as the sole key material for the proof key,

b) accept the client entropy as partial key material and contribute additional server-side entropy as partial key material to compute the proof key as a function of both partial key materials, or

c) reject the client-side entropy and use server-side entropy as the sole key material for the proof key.

For each of the cases above, the IP/STS MUST compute and return the proof key by augmenting the RSTR response message as follows.

For case (a) where IP/STS accepts client entropy as the sole key material:

- The RSTR MUST NOT include a `wst:RequestedProofToken` element. The proof key is implied and an Identity Selector MUST use the client-side entropy as the proof key.

For case (b) where IP/STS accepts client entropy and contributes additional server entropy:

- The RSTR MUST include a `wst:BinarySecret` element inside a `wst:Entropy` element containing the server-side entropy to be used as partial key material. The entropy is conveyed as raw base64 encoded bits.
- The partial key material from the IP/STS MUST be combined (by each party) with the partial key material from the client to determine the resulting proof key.
- The RSTR MUST include a `wst:RequestedProofToken` element containing a `wst:ComputedKey` element to indicate how the proof key is to be computed. It is RECOMMENDED that an Identity Selector support the P_SHA1 computed key mechanism defined in [WS-Trust 1.2] or [WS-Trust 1.3] with the particulars below. Usage of other algorithms is not described.

ComputedKey Value	*Meaning*

Copyright © OASIS® 2008-2009. All Rights Reserved.

http://schemas.xmlsoap.org/ws/2005/02/trust/CK/PSHA1 or http://docs.oasis-open.org/ws-sx/ws-trust/200512/CK/PSHA1	The key is computed using P_SHA1 from the TLS specification to generate a bit stream using entropy from both sides. The exact form is: key = P_SHA1 (Entropy$_{REQ}$, Entropy$_{RES}$)

Following is a sample RSTR response fragment that illustrates a token response with partial key material from the IP/STS and a computed proof key.

Example:

```
<wst:RequestSecurityTokenResponse xmlns:wst="...">
  ...
  <wst:Entropy>
    <wst:BinarySecret>mQlxWxEiKOcUfnHgQpylcD7LYSkJplpE=</wst:BinarySecret>
  </wst:Entropy>
  <wst:RequestedProofToken>
    <wst:ComputedKey>
       http://schemas.xmlsoap.org/ws/2005/02/trust/CK/PSHA1
    </wst:ComputedKey>
  </wst:RequestedProofToken>
</wst:RequestSecurityTokenResponse>
```

For case (c) where IP/STS contributes server entropy as the sole key material:

- The RSTR MUST include a `wst:BinarySecret` element inside a `wst:RequestedProofToken` element containing the specific proof key to be used. The proof key is conveyed as raw base64 encoded bits.

Following is a sample RSTR response fragment that illustrates a token response with fully specified proof key from the IP/STS.

Example:

```
<wst:RequestSecurityTokenResponse xmlns:wst="...">
  ...
  <wst:RequestedProofToken>
    <wst:BinarySecret>
      mQlxWxEiKOcUfnHgQpylcDKOcUfnHg7LYSkJplpE=
    </wst:BinarySecret>
  </wst:RequestedProofToken>
</wst:RequestSecurityTokenResponse>
```

The following table summarizes the symmetric proof key computation rules to be used by an Identity Selector:

Token Requester (Identity Selector)	*Token Issuer (IP/STS)*	*Results*
Provides entropy	Uses requester entropy as proof key	No <wst:RequestedProofToken> element present in RSTR. Proof key is implied.
Provides entropy	Uses requester entropy and provides additional entropy of its own	<wst:Entropy> element present in RSTR containing issuer supplied entropy. <wst:RequestedProofToken> element present in RSTR containing computed key mechanism. Requestor and Issuer compute proof key by combining both entropies using the specified computed key

Copyright © OASIS® 2008-2009. All Rights Reserved.

		mechanism.
Provides entropy	Uses own entropy as proof key (rejects requester entropy)	<wst:RequestedProofToken> element present in RSTR containing the proof key.

3.3.5.2 Asymmetric Proof Key

When requesting an asymmetric key token, it is RECOMMENDED that an Identity Selector generate an ephemeral RSA key pair. Usage of other algorithms is not described. The generated RSA key pair MUST be at least 1024-bits in size for use as the proof key. It MUST submit the public key to the IP/STS by augmenting the RST request as follows:

- The RST MUST include a `wst:KeyType` element with one of the two following URI values, depending upon the version of WS-Trust being used:

  http://schemas.xmlsoap.org/ws/2005/02/trust/PublicKey

  http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey

- The RST SOAP body MUST include a `wst:UseKey` element containing the public key to be used as proof key in the returned token. The public key is present as a raw RSA key in the form of a `ds:RSAKeyValue` element inside a `ds:KeyValue` element.

- The RST SOAP security header SHOULD include a supporting signature to prove ownership of the corresponding private key. The `ds:KeyInfo` element within the signature, if present, MUST include the same public key as in the `wst:UseKey` element in the SOAP body.

- The supporting signature, if present, MUST be placed in the SOAP security header where the signature for an endorsing supporting token would be placed as per the security header layout specified in WS-SecurityPolicy.

Following is a sample RST request fragment that illustrates an asymmetric key based token request containing the public key and proof of ownership of the corresponding private key.

Example:

```
<s:Envelope xmlns:s="..." xmlns:wsse="..." xmlns:ds="..." xmlns:wst="..."
    ... >
  <s:Header>
    ...
    <wsse:Security>
      ...
      <ds:Signature Id="_proofSignature">
        <!-- signature proving possession of submitted proof key -->
        ...
        <!-- KeyInfo in signature contains the submitted proof key -->
        <ds:KeyInfo>
          <ds:KeyValue>
            <ds:RSAKeyValue>
              <ds:Modulus>...</ds:Modulus>
              <ds:Exponent>...</ds:Exponent>
            </ds:RSAKeyValue>
          </ds:KeyValue>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </s:Header>
  <s:Body wsu:Id="req">
    <wst:RequestSecurityToken>
      ...
      <wst:KeyType>
        http://schemas.xmlsoap.org/ws/2005/02/trust/PublicKey
      </wst:KeyType>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
      <wst:UseKey Sig="#_proofSignature">
        <ds:KeyInfo>
          <ds:KeyValue>
            <ds:RSAKeyValue>
              <ds:Modulus>...</ds:Modulus>
              <ds:Exponent>...</ds:Exponent>
            </ds:RSAKeyValue>
          </ds:KeyValue>
        </ds:KeyInfo>
      </wst:UseKey>
    </wst:RequestSecurityToken>
  </s:Body>
</s:Envelope>
```

If a supporting signature for the submitted proof key is not present in the token request, the IP/STS MAY fail the request. If a supporting signature is present, the IP/STS MUST verify the signature and MUST ensure that the public key included in the `wst:UseKey` element and in the supporting signature are the same. If verification succeeds and the IP/STS accepts the submitted public key for use in the issued token, then the token response MUST NOT include a `wst:RequestedProofToken` element. The proof key is implied and an Identity Selector MUST use the public key it submitted as the proof key.

The following table summarizes the asymmetric proof key rules used by an Identity Selector:

Token Requester (Identity Selector)	*Token Issuer (IP/STS)*	*Results*
Provides ephemeral public key for use as proof key	Uses requester supplied proof key	No <wst:RequestedProofToken> element present in RSTR. Proof key is implied.

3.3.5.3 No Proof Key

When requesting a token with no proof key, an Identity Selector MUST augment the RST request message as follows:

- The RST MUST include a `wst:KeyType` element with the following URI value if [WS-Trust 1.2] is being used:

  http://schemas.xmlsoap.org/ws/2005/05/identity/NoProofKey

  or the RST MUST include a wst:KeyType element with the following URI value if [WS-Trust 1.3] is being used:

  http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer

Following is a sample RST request fragment that illustrates a bearer token request.

Example:

```
<wst:RequestSecurityToken xmlns:wst="...">
  ...
  <wst:KeyType>
    http://schemas.xmlsoap.org/ws/2005/05/identity/NoProofKey
  </wst:KeyType>
</wst:RequestSecurityToken>
```

When processing the token request, if the IP/STS issues a SAML v1.1 bearer token then:

- It MUST specify “urn:oasis:names:tc:SAML:1.0:cm:bearer” as the subject confirmation method in the token.
- It SHOULD include a `saml:AudienceRestrictionCondition` element restricting the token to the target site URL submitted in the token request.

Copyright © OASIS® 2008-2009. All Rights Reserved.

3.3.6 Display Token

An Identity Selector MAY request a Display Token – a representation of the claims carried in the issued Security Token that can be displayed in an user interface – from an IP/STS as part of the token request. To request a Display Token, the following element MUST be included in the RST message as a top-level element information item.

Syntax:

```
<ic:RequestDisplayToken xml:lang="xs:language"? xmlns:ic="..." ... />
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:RequestDisplayToken

This optional element is used to request an Identity Provider to return a Display Token corresponding to the issued token.

/ic:RequestDisplayToken/@xml:lang

This optional attribute indicates a language identifier, using the language codes specified in [RFC 3066], in which the Display Token content SHOULD be localized.

An IP/STS MAY respond to a Display Token request. If it does, it MUST use the following element to return a Display Token for the issued Security Token in the RSTR message.

Syntax:

```
<ic:RequestedDisplayToken xmlns:ic="..." ...>
  <ic:DisplayToken xml:lang="xs:language" ... >
    [ <ic:DisplayClaim Uri="xs:anyURI" ...>
        <ic:DisplayTag> xs:string </ic:DisplayTag> ?
        <ic:Description> xs:string </ic:Description> ?
        <ic:DisplayValue> xs:string </ic:DisplayValue> ?
      </ic:DisplayClaim> ] +
    |
    [ <ic:DisplayTokenText MimeType="xs:string">
        xs:string
      </ic:DisplayTokenText> ]
    ...
  </ic:DisplayToken>
</ic:RequestedDisplayToken>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:RequestedDisplayToken

This optional element is used to return a Display Token for the Security Token returned in the response.

/ic:RequestedDisplayToken/ic:DisplayToken

The returned Display Token.

/ic:RequestedDisplayToken/ic:DisplayToken/@xml:lang

This required attribute indicates a language identifier, using the language codes specified in [RFC 3066], in which the Display Token content is localized.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim

This required element indicates an individual claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/@Uri

This required attribute provides the unique identifier (URI) of the individual claim returned in the Security Token.

Copyright © OASIS® 2008-2009. All Rights Reserved.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/ic:DisplayTag

This optional element provides a friendly name for the claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/ic:Description

This optional element provides a description of the semantics for the claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/ic:DisplayValue

This optional element provides the displayable value for the claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayTokenText

This optional element provides an alternative textual representation of the entire token as a whole when the token content is not suitable for display as individual claims.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayTokenText/@MimeType

This required attribute provides a MIME type specifying the format of the Display Token content (e.g., “text/plain”).

The following example illustrates a returned Display Token corresponding to a Security Token with two claims.

Example:

```
<ic:RequestedDisplayToken xmlns:ic="...">
  <ic:DisplayToken xml:lang="en-us">
    <ic:DisplayClaim Uri="http://.../ws/2005/05/identity/claims/givenname">
      <ic:DisplayTag>Given Name</ic:DisplayTag>
      <ic:DisplayValue>John</ic:DisplayValue>
    </ic:DisplayClaim>
    <ic:DisplayClaim Uri="http://.../ws/2005/05/identity/claims/surname">
      <ic:DisplayTag>Last Name</ic:DisplayTag>
      <ic:DisplayValue>Doe</ic:DisplayValue>
    </ic:DisplayClaim>
  <ic:DisplayToken>
</ic:RequestedDisplayToken>
```

3.3.7 Token References

When an IP/STS returns the token requested by an Identity Selector, it MUST also include an attached and an un-attached token reference for the issued security token using the `wst:RequestedAttachedReference` and `wst:RequestedUnattachedReference` elements, respectively, in the RSTR response message.

An Identity Selector is truly a conduit for the security tokens issued by an IP/STS and requested by an RP, and it should remain agnostic of the type of the security token passing through it. Furthermore, a security token issued by an IP/STS MAY be encrypted directly for the RP, thus preventing visibility into the token by the Identity Selector. However, an Identity Selector (or a client application) needs to be able to use the issued security token to perform security operations (such as signature or encryption) on a message sent to an RP and thus needs a way to reference the token both when it is attached to a message and when it is not. The attached and unattached token references returned by an IP/STS in the RSTR message provide the necessary references that can be used for this purpose.

4 Authenticating to Identity Provider

The Information Card schema includes the element content necessary for an Identity Provider to express what credential the user must use in order to authenticate to the IP/STS when requesting tokens. This section defines the schema used to express the credential descriptor for each supported credential type.

Copyright © OASIS® 2008-2009. All Rights Reserved.

4.1 Username and Password Credential

When the Identity Provider requires a *username* and *password* as the credential type, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential xmlns:ic="...">
  <ic:UsernamePasswordCredential>
    <ic:Username> xs:string </ic:Username> ?
  </ic:UsernamePasswordCredential>
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:UsernamePasswordCredential

This element indicates that a username/password credential is needed.

.../ic:UsernamePasswordCredential/ic:Username

This optional element provides the username part of the credential for convenience. An Identity Selector MUST prompt the user for the password. If the username is specified, then its value MUST be copied into the username token used to authenticate to the IP/STS; else an Identity Selector MUST prompt the user for the username as well.

Furthermore, the actual Security Policy of the IP/STS (expressed in its WSDL) MUST include the `sp:UsernameToken` assertion requiring a username and password value.

4.2 Kerberos v5 Credential

When the Identity Provider requires a *Kerberos v5 service ticket* for the IP/STS as the credential type, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential xmlns:ic="...">
  <ic:KerberosV5Credential />
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:KerberosV5Credential

This element indicates that a Kerberos v5 credential is needed.

To enable the Service Requester to obtain a Kerberos v5 service ticket for the IP/STS, the endpoint reference of the IP/STS in the Information Card or in the metadata retrieved from it MUST include a "service principal name" identity claim (i.e. a `wsai:Spn` element) under the `wsai:Identity` tag as defined in Section 12.

Furthermore, the actual Security Policy of the IP/STS (expressed in its WSDL) MUST include the `sp:KerberosToken` assertion requiring a Kerberos service ticket.

4.3 X.509v3 Certificate Credential

When the Identity Provider requires an *X.509 v3 certificate* for the user as the credential type, where the certificate and keys are in a hardware-based smart card or a software-based certificate, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential xmlns:ic="..." xmlns:ds="..." xmlns:wsse="...">
  <ic:DisplayCredentialHint> xs:string </ic:DisplayCredentialHint>
  <ic:X509V3Credential>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
    <ds:X509Data>
      <wsse:KeyIdentifier
        ValueType="http://docs.oasisopen.org/wss/oasiswss-soap-
messagesecurity-1.1#ThumbPrintSHA1"
        EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis200401-wss-
soap-message-security-1.0#Base64Binary">
        xs:base64binary
      </wsse:KeyIdentifier>
    </ds:X509Data>
  </ic:X509V3Credential>
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:DisplayCredentialHint

This optional element provides a user hint string which can be used to prompt the user, for example, to insert the appropriate smart card into the reader.

.../ic:X509V3Credential

This element indicates that a X.509 certificate credential is needed.

.../ic:X509V3Credential/ds:X509Data/wsse:KeyIdentifier

This element provides a key identifier for the X.509 certificate based on the SHA1 hash of the entire certificate content expressed as a “thumbprint.” Note that the extensibility point in the `ds:X509Data` element is used to add `wsse:KeyIdentifier` as a child element.

Furthermore, the actual Security Policy of the IP/STS, expressed in its WSDL, MUST include the `sp:X509Token` assertion requiring an X.509v3 certificate.

4.4 Self-issued Token Credential

When the Identity Provider requires a *self-issued token* as the credential type, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential xmlns:ic="...">
  <ic:SelfIssuedCredential>
    <ic:PrivatePersonalIdentifier>
      xs:base64Binary
    </ic:PrivatePersonalIdentifier>
  </ic:SelfIssuedCredential>
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:SelfIssuedCredential

This element indicates that a self-issued token credential is needed.

.../ic:SelfIssuedCredential/ic:PrivatePersonalIdentifier

This required element provides the value of the PPID claim asserted in the self-issued token used previously to register with the IP/STS (see Section 7.5.14).

Furthermore, the actual Security Policy of the IP/STS (expressed in its WSDL) MUST include the `sp:IssuedToken` assertion requiring a self-issued token with exactly one claim, namely, the PPID.

5 Faults

In addition to the standard faults described in WS-Addressing, WS-Security and WS-Trust, this profile defines the following additional faults that MAY occur when interacting with an RP or an IP. The binding of the fault properties (listed below) to a SOAP 1.1 or SOAP 1.2 fault message is described in [WS-

Copyright © OASIS® 2008-2009. All Rights Reserved.

Addressing]. If the optional **[Detail]** property for a fault includes any specified content, then the corresponding schema fragment is included in the listing below.

5.1 Relying Party

The following faults MAY occur when submitting Security Tokens to an RP per its Security Policy.

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	*S:Sender*
[Subcode]	ic:RequiredClaimMissing
[Reason]	A required claim is missing from the Security Token.
[Detail]	[URI of missing claim] `<ic:ClaimType Uri="[Claim URI]" />`

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:InvalidClaimValue
[Reason]	A claim value asserted in the Security Token is invalid.
[Detail]	[URI of invalid claim] `<ic:ClaimType Uri="[Claim URI]" />`

5.2 Identity Provider

The following faults MAY occur when requesting Security Tokens from an IP using Information Cards.

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:MissingAppliesTo
[Reason]	The request is missing Relying Party identity information.
[Detail]	(None defined.)

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:InvalidProofKey
[Reason]	Invalid proof key specified in request.
[Detail]	(None defined.)

Copyright © OASIS® 2008-2009. All Rights Reserved.

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:UnknownInformationCardReference
[Reason]	Unknown Information Card reference specified in request.
[Detail]	[Unknown Information Card reference] `<ic:InformationCardReference>` `<ic:CardId>[card ID]</ic:CardId>` `<ic:CardVersion>[version]</ic:CardVersion>` `</ic:InformationCardReference>`

1381

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:FailedRequiredClaims
[Reason]	Could not satisfy required claims in request; construction of token failed
[Detail]	[URIs of claims that could not be satisfied] `<ic:ClaimType Uri="[Claim URI]" />` `...`

1382

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:InformationCardRefreshRequired
[Reason]	Stale Information Card reference specified in request; Information Card SHOULD be refreshed
[Detail]	[Information Card reference that needs refreshing] `<ic:InformationCardReference>` `<ic:CardId>[card ID]</ic:CardId>` `<ic:CardVersion>[version]</ic:CardVersion>` `</ic:InformationCardReference>`

1383 ### 5.2.1 Identity Provider Custom Error Messages

1384 Identity Providers MAY return custom error messages to Identity Selectors via SOAP faults that can be
1385 displayed by the Identity Selector user interface. The error message MUST be communicated as an
1386 `S:Text` element within the `S:Reason` element of a SOAP fault message. Multiple `S:Text` elements
1387 MAY be returned with different `xml:lang` values and the Identity Selector SHOULD use the one
1388 matching the user's locale, if possible.

1389 *Example:*

```
1390 <s:Envelope xmlns:wsa="http://www.w3.org/2005/08/addressing"
1391     xmlns:s="http://www.w3.org/2003/05/soap-envelope">
1392   <s:Header>
1393     <wsa:Action s:mustUnderstand="1">
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
      http://www.w3.org/2005/08/addressing/soap/fault
    </wsa:Action>
  </s:Header>
  <s:Body>
    <s:Fault>
      <s:Code>
        <s:Value>s:Sender</s:Value>
      </s:Code>
      <s:Reason>
        <s:Text xml:lang="en">Message in English ...</</s:Text>
        <s:Text xml:lang="es-ES">Message in the Spanish of Spain ...</s:Text>
      </s:Reason>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

6 Information Cards Transfer Format

This section defines how collections of Information Cards are transferred between Identity Selectors. The cards collection is always transferred after encrypting it with a key derived from a user specified password. Section 6.1 describes the transfer format of the collection in the clear, whereas Section 6.1.2 describes the transfer format after the necessary encryption is applied.

6.1 Pre-Encryption Transfer Format

Each Information Card in the transfer stream will contain metadata and key material maintained by the originating Identity Selector in addition to the original Information Card metadata. If an Identity Selector includes a co-resident Self-issued Identity Provider (described in Section 7), an exported self-issued card MAY also contain any associated claims information.

The XML schema used for the transfer format is defined below:

Syntax:

```
<ic:RoamingStore xmlns:ic="...">
  <ic:RoamingInformationCard> +
    <ic:InformationCardMetaData>
      [Information Card]
      <ic:IsSelfIssued> xs:boolean </ic:IsSelfIssued>
      <ic:PinDigest> xs:base64Binary </ic:PinDigest> ?
      <ic:HashSalt> xs:base64Binary </ic:HashSalt>
      <ic:TimeLastUpdated> xs:dateTime </ic:TimeLastUpdated>
      <ic:IssuerId> xs:base64Binary </ic:IssuerId>
      <ic:IssuerName> xs:string </ic:IssuerName>
      <ic:BackgroundColor> xs:int </ic:BackgroundColor>
    </ic:InformationCardMetaData>
    <ic:InformationCardPrivateData> ?
      <ic:MasterKey> xs:base64Binary </ic:MasterKey>
      <ic:ClaimValueList> ?
        <ic:ClaimValue Uri="xs:anyURI" ...> +
          <ic:Value> xs:string </ic:Value>
        </ic:ClaimValue>
      </ic:ClaimValueList>
    </ic:InformationCardPrivateData>
    ...
  </ic:RoamingInformationCard>
  ...
</ic:RoamingStore>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:RoamingStore

The collection of Information Cards selected for transfer.

Copyright © OASIS® 2008-2009. All Rights Reserved.

/ic:RoamingStore/ic:RoamingInformationCard (one or more)

An individual Information Card within the transfer stream.

For brevity, the prefix string “/ic:RoamingStore/ic:RoamingInformationCard” in the element names below is shortened to “...”.

.../ic:InformationCardMetaData

This required element contains the metadata for an Information Card.

.../ic:InformationCardMetaData/[Information Card]

The original content of the Information Card as issued by the Identity Provider (described in Section 3.1.1).

.../ic:InformationCardMetaData/ic:IsSelfIssued

This required element indicates if the card is self-issued (“true”) or not (“false”).

.../ic:InformationCardMetaData/ic:PinDigest

This optional element contains a digest of the user-specified PIN information if the card is PIN-protected. The digest contains the base64 encoded bytes of the SHA1 hash of the bytes of the user-specified PIN represented using Unicode encoding UTF-16LE with no byte order mark. Usage of other algorithms is not described.

.../ic:InformationCardMetaData/ic:HashSalt

This optional element contains a random per-card entropy value used for computing the Relying Party specific PPID claim when the card is used at a Relying Party and for computing the Client Pseudonym PPID value sent an Identity Provider.

.../ic:InformationCardMetaData/ic:TimeLastUpdated

This required element contains the date and time when the card was last updated.

.../ic:InformationCardMetaData/ic:IssuerId

This required element contains an identifier for the Identity Provider with which a self-issued credential descriptor in a card issued by that Identity Provider can be resolved to the correct self-issued card. The element content SHOULD be the empty string for self-issued cards.

.../ic:InformationCardMetaData/ic:IssuerName

This required element contains a friendly name of the card issuer.

.../ic:InformationCardMetaData/ic:BackgroundColor

This required element contains the background color used to display the card image. This value is a 3-byte RGB color value in the sRGB color space used by HTML.

.../ic:InformationCardMetaData/{any}

This is an extensibility point to allow additional metadata to be included.

.../ic:InformationCardPrivateData

This required element contains the private data for an Information Card.

.../ic:InformationCardPrivateData/ic:MasterKey

This required element contains a base64 encoded 256-bit random number that provides a “secret key” for the Information Card. This key is used for computing the Relying Party specific PPID claim when the card is used at a Relying Party and for computing the Client Pseudonym PPID value sent to an Identity Provider. This element is present both for self-issued and managed Information Cards.

.../ic:InformationCardPrivateData/ic:ClaimValueList

This optional element is a container for the set of claim types and their corresponding values embodied by a self-issued card.

Copyright © OASIS® 2008-2009. All Rights Reserved.

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue (one or more)

This required element is a container for an individual claim, *i.e.*, a claim type and its corresponding value.

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue/@Uri

This required attribute contains a URI that identifies the specific claim type.

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue/ic:Value

This required element contains the value for an individual claim type.

…/@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore/ic:RoamingInformationCard` element when updating information using the Information Cards Transfer Format.

…/{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore/ic:RoamingInformationCard` element when updating information using the Information Cards Transfer Format.

/ic:RoamingStore/@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore` element when updating information using the Information Cards Transfer Format.

/ic:RoamingStore/{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore` element when updating information using the Information Cards Transfer Format.

6.1.1 PIN Protected Card

When an Information Card is PIN protected, in addition to storing a digest of the PIN in the card data, the master key and claim values associated with the card MUST also be encrypted with a key derived from the user-specified PIN.

It is RECOMMENDED that the PKCS-5 based key derivation method be used with the input parameters summarized in the table below for deriving the encryption key from the PIN. Usage of other algorithms is not described.

Copyright © OASIS® 2008-2009. All Rights Reserved.

Key derivation method	PBKDF1 per [RFC 2898] (Section 5.1)
Input parameters:	
Password	UTF-8 encoded octets of PIN
Salt	16-byte random number (actual value stored along with master key)
Iteration count	1000 (actual value stored along with master key)
Key length	32 octets
Hash function	SHA-256

The encryption method and the corresponding parameters that MUST be used are summarized in the table below.

Encryption method	AES-256
Parameters:	
Padding	As per PKCS-7 standard
Mode	CBC
Block size	16 bytes (as REQUIRED by AES)

In a PIN-protected card, the encrypted content of the master key and the claim value fields are described below.

.../ic:InformationCardPrivateData/ic:MasterKey

This element MUST contain a base64 encoded byte array comprised of the encryption parameters and the encrypted master key serialized as per the binary structure summarized in the table below.

Field	*Offset*	*Size (bytes)*
Version (for internal use)	0	1
Salt used for key-derivation method	1	16
Iteration count used for key-derivation method	17	4
Initialization Vector (IV) used for encryption	21	16
Encrypted master key	37	master key length

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue/ic:Value

This element MUST contain a base64 encoded byte array comprised of the encrypted claim value. The encryption parameters used are taken from those serialized into the master key field and summarized in the table above.

6.1.2 Computing the ic:IssuerId

The `ic:IssuerId` value used for a card when representing it in the Information Cards Transfer Format SHOULD be computed as a function of the `ds:KeyInfo` field of the envelope digitally signed by the Identity Provider. Specifically:

- Compute *IP PPID Seed* in the same manner as *RP PPID Seed* in Section 7.6.1, except that the certificate from `ds:KeyInfo` is used, rather than the Relying Party's.

Copyright © OASIS® 2008-2009. All Rights Reserved.

Use the *IP PPID Seed* as the `ic:IssuerId` value.

The `ic:IssuerId` value SHOULD be the empty string for self-issued cards.

6.1.3 Computing the ic:IssuerName

The `ic:IssuerName` value used for a card when representing it in the Information Cards Transfer Format SHOULD be computed as a function of the `ds:KeyInfo` field of the envelope digitally signed by the Identity Provider. Specifically, if the certificate from `ds:KeyInfo` is an extended validation (EV) certificate [EV Cert], then set `ic:IssuerName` to the organizationName (O) field value from the certificate, otherwise set `ic:IssuerName` to the commonName (CN) field value from the certificate.

6.1.4 Creating the ic:HashSalt

A random `ic:HashSalt` value for a card SHOULD be created by the Identity Selector when that card is created from the `ic:InformationCard` data provided by an Identity Provider.

6.2 Post-Encryption Transfer Format

The transfer stream MUST be encrypted with a key derived from a user specified password. The XML schema used for the encrypted transfer stream is defined below:

Syntax:

```
Byte-order-mark
<?xml version="1.0" encoding="utf-8"?>
<ic:EncryptedStore xmlns:ic="..." xmlns:xenc="...">
  <ic:StoreSalt> xs:base64Binary </ic:StoreSalt>
  <xenc:EncryptedData>
    <xenc:CipherData>
      <xenc:CipherValue> ... </xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>
</ic:EncryptedStore>
...
```

The following describes the elements listed in the XML schema outlined above:

Byte-order-mark

The first three bytes in the stream containing the values {0xEF, 0xBB, 0xBF} constitutes a “byte order mark”.

/ic:EncryptedStore

The top-level container element for the encrypted transfer stream.

/ic:EncryptedStore/ic:StoreSalt

This required element contains the random salt used as a parameter for the key derivation function to derive the encryption key from a user-specified password.

/ic:EncryptedStore/xenc:EncryptedData/xenc:CipherData/xenc:CipherValue

This element contains a base64 encoded byte array containing the ciphertext corresponding to the clear text transfer stream described in Section 6.1.

@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them when updating information using the Information Cards Transfer Format.

Copyright © OASIS® 2008-2009. All Rights Reserved.

{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them when updating information using the Information Cards Transfer Format.

The remainder of this section describes the element content of the *xenc:CipherValue* element in the schema outline above. Specifically, it describes the encryption method used and the format of the encrypted content.

The following table defines two symbolic constants, namely *EncryptionKeySalt* and *IntegrityKeySalt*, and their corresponding values used by the key derivation and the encryption methods described below to encrypt the transfer stream.

EncryptionKeySalt	{ 0xd9, 0x59, 0x7b, 0x26, 0x1e, 0xd8, 0xb3, 0x44, 0x93, 0x23, 0xb3, 0x96, 0x85, 0xde, 0x95, 0xfc }
IntegrityKeySalt	{ 0xc4, 0x01, 0x7b, 0xf1, 0x6b, 0xad, 0x2f, 0x42, 0xaf, 0xf4, 0x97, 0x7d, 0x4, 0x68, 0x3, 0xdb }

The transfer stream content is encrypted with a key derived from a user-specified password. It is RECOMMENDED that the PKCS-5 based key derivation method be used with the input parameters summarized in the table below for deriving the key from the password. Usage of other algorithms is not described.

Key derivation method	PBKDF1 per [RFC 2898] (Section 5.1)
Input parameters:	
Password	UTF-8 encoded octets of user-specified password
Salt	16-byte random number (actual value stored in the *ic:StoreSalt* field)
Iteration count	1000
Key length	32 octets
Hash function	SHA-256

The PKCS-5 key derived as per the preceding table MUST be further hashed with a 16-byte salt using the SHA256 hash function, and the resulting value used as the encryption key. The order in which the values used MUST be hashed is as follows:

Encryption Key = SHA256 (EncryptionKeySalt + PKCS5-derived-key)

Further, to provide an additional integrity check at the time of import, a “hashed integrity code” MUST be computed as follows and included along with the encrypted transfer stream content.

- The PKCS-5 key derived as per the preceding table MUST be further hashed with a 16-byte salt using the SHA256 hash function, and the resulting value used as the integrity key. The order in which the values used MUST be hashed is as follows:

  Integrity Key = SHA256 (IntegrityKeySalt + PKCS5-derived-key)

- The last block of the clear text transfer stream MUST be captured and further hashed with the *integrity key (IK)* and the *initialization vector (IV)* using the SHA256 hash function, and the resulting value used as the hashed integrity code. The order in which the values used MUST be hashed is as follows:

Copyright © OASIS® 2008-2009. All Rights Reserved.

Hashed Integrity Code = SHA256 (IV + IK + Last-block-of-clear-text)

The encryption method and the corresponding parameters that MUST be used to encrypt the transfer stream are summarized in the table below.

Encryption method	AES-256
Parameters:	
Padding	As per PKCS-7 standard
Mode	CBC
Block size	16 bytes (as REQUIRED by AES)

The element content of `xenc:CipherValue` MUST be a base64 encoded byte array comprised of the initialization vector used for encryption, the hashed integrity code (as described above), and the encrypted transfer stream. It MUST be serialized as per the binary structure summarized in the table below.

Field	*Offset*	*Size (bytes)*
Initialization Vector (IV) used for encryption	0	16
Hashed integrity code	16	32
Ciphertext of transfer stream	48	Arbitrary

7 Simple Identity Provider Profile

A simple Identity Provider, called the “Self-issued Identity Provider” (SIP), is one which allows users to self-assert identity in the form of self-issued tokens. An Identity Selector MAY include a co-resident Self-issued Identity Provider that conforms to the Simple Identity Provider Profile defined in this section. This profile allows self-issued identities created within one Identity Selector to be used in another Identity Selector such that users do not have to reregister at a Relying Party when switching Identity Selectors. Because of the co-location there is data and metadata specific to an Identity Provider that need to be shareable between Identity Selectors.

7.1 Self-Issued Information Card

The `ic:Issuer` element within an Information Card provides a logical name for the issuer of the Information Card. An Information Card issued by a SIP (*i.e.*, a self-issued Information Card) MUST use the special URI below as the value of the `ic:Issuer` element in the Information Card.

URI:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self
```

7.2 Self-Issued Token Characteristics

The self-issued tokens issued by a SIP MUST have the following characteristics:

- The token type of the issued token MUST be SAML 1.1 which MUST be identified by either of the following token type URIs:
  - *urn:oasis:names:tc:SAML:1.0:assertion*, or
  - *http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1*.
- It is RECOMMENDED that the signature key used in the issued token be a 2048-bit asymmetric RSA key which identifies the issuer. Usage of other algorithms is not described.

Copyright © OASIS® 2008-2009. All Rights Reserved.

- The issuer of the token, indicated by the value of the `saml:Issuer` attribute on the `saml:Assertion` root element, MUST be identified by the following URI defined in Section 2.1.1 representing the issuer "self".

  `http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self`

- The issued token MUST contain the `saml:Conditions` element specifying:
  - the token validity interval using the `NotBefore` and `NotOnOrAfter` attributes, and
  - the `saml:AudienceRestrictionCondition` element restricting the token to a specific target scope (i.e., a specific recipient of the token).
- The `saml:NameIdentifier` element SHOULD NOT be used to specify the Subject of the token.
- The subject confirmation method MUST be specified as one of:
  - *urn:oasis:names:tc:SAML:1.0:cm:holder-of-key*, or
  - *urn:oasis:names:tc:SAML:1.0:cm:bearer* (for Browser based applications).
- When the subject confirmation method is "holder of key", the subject confirmation key (also referred to as the *proof key*) MUST be included in the token in the `ds:KeyInfo` child element under the `saml:SubjectConfirmation` element. The proof key MUST be encoded in the token as follows:
  - For *symmetric* key tokens, the proof key is encrypted to the recipient of the token in the form of a `xenc:EncryptedKey` child element. It is RECOMMENDED that an AES key with a default size of 256 bits be used, but a different size MAY be specified by the Relying Party. Usage of other algorithms is not described.
  - For *asymmetric* key tokens, it is RECOMMENDED that the proof key be a public RSA key value specified as a `ds:RSAKeyValue` child element under the `ds:KeyValue` element. The default size of the key is 2048 bits. Usage of other algorithms is not described.
- The issued token MUST contain a single attribute statement (i.e., a single `saml:AttributeStatement` element) containing the subject confirmation data and the requested claims (called *attributes* in a SAML token).
- The claim types supported by the self-issued token SHOULD include those listed in Section7.5.
- The claims asserted in the `saml:AttributeStatement` element of the issued token MUST be named as follows using the claim type definitions in the XML schema file referenced in Section7.5. For each claim represented by a `saml:Attribute` element,
  - the `AttributeName` attribute is set to the NCname of the corresponding claim type defined in the XML schema file, and
  - the `AttributeNamespace` attribute is set to the target namespace of the XML schema file, namely

    `http://schemas.xmlsoap.org/ws/2005/05/identity/claims`

It is RECOMMENDED that the XML digital signature [XMLDSIG] profile used to sign a self-issued token be as follows. Usage of other algorithms is not described.

- Uses the *enveloped signature* format identified by the transform algorithm identifier "*http://www.w3.org/2000/09/xmldsig#enveloped-signature*". The token signature contains a single `ds:Reference` containing a URI reference to the `AssertionID` attribute value of the root element of the SAML token.

Copyright © OASIS® 2008-2009. All Rights Reserved.

- Uses the RSA signature method identified by the algorithm identifier "*http://www.w3.org/2000/09/xmldsig#rsa-sha1*".
- Uses the exclusive canonicalization method identified by the algorithm identifier **"***http://www.w3.org/2001/10/xml-exc-c14n#*" for canonicalizing the token content as well as the signature content.
- Uses the SHA1 digest method identified by the algorithm identifier "*http://www.w3.org/2000/09/xmldsig#sha1*" for digesting the token content being signed.
- No other transforms, other than the ones listed above, are used in the enveloped signature.
- The `ds:KeyInfo` element is always present in the signature carrying the signing RSA public key in the form of a `ds:RSAKeyValue` child element.

Following is an example of a self-issued signed Security Token containing three claims.

Example:

```
<Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
    AssertionID="urn:uuid:08301dba-d8d5-462f-85db-dec08c5e4e17"
    Issuer="http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self"
    IssueInstant="2004-10-06T16:44:20.00Z"
    MajorVersion="1" MinorVersion="1">
  <Conditions NotBefore="2004-10-06T16:44:20.00Z"
    NotOnOrAfter="2004-10-06T16:49:20.00Z">
    <AudienceRestrictionCondition>
      <Audience>http://www.relying-party.com</Audience>
    </AudienceRestrictionCondition>
  </Conditions>
  <AttributeStatement>
    <Subject>
      <!-- Content here differs; see examples that follow -->
    </Subject>
    <Attribute AttributeName="privatepersonalidentifier"
AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <AttributeValue>
        f8301dba-d8d5a904-462f0027-85dbdec0
      </AttributeValue>
    </Attribute>
    <Attribute AttributeName="givenname"
AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <AttributeValue>dasf</AttributeValue>
    </Attribute>
    <Attribute AttributeName="emailaddress"
AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <AttributeValue>dasf@mail.com</AttributeValue>
    </Attribute>
  </AttributeStatement>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="urn:uuid:08301dba-d8d5-462f-85db-dec08c5e4e17">
        <Transforms>
          <Transform
            Algorithm="http://.../2000/09/xmldsig#enveloped-signature"/>
          <Transform
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        </Transforms>
        <DigestMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>vpnIyEi4R/S4b+1vEH4gwQ9iHsY=</DigestValue>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
      </Reference>
    </SignedInfo>
    <SignatureValue>...</SignatureValue>
    <!-- token signing key -->
    <KeyInfo>
      <KeyValue>
        <RSAKeyValue>
          <Modulus>... utnQyEi8R/S4b+1vEH4gwR9ihsV ...</Modulus>
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
</Assertion>
```

The content of the `saml:Subject` element in the self-issued token differs based on the subject confirmation method and the type of proof key used. The following examples illustrate each of the three variations of the content of this element.

The following example illustrates the content of the `saml:Subject` element when subject confirmation method is “holder of key” using a symmetric proof key.

Example:

```
<Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:ds="..."
    xmlns:wsse="..." xmlns:xenc="...">
  <SubjectConfirmation>
    <ConfirmationMethod>
      urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
    </ConfirmationMethod>
    <ds:KeyInfo>
      <!-- symmetric proof key encrypted to recipient -->
      <xenc:EncryptedKey>
        <xenc:EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
        <ds:KeyInfo>
          <ds:X509Data>
            <wsse:KeyIdentifier
              ValueType="http://docs.oasis-open.org/wss/2004/xx/oasis-2004xx-
wss-soap-message-security-1.1#ThumbprintSHA1">
                EdFoIaAeja85201XTzjNMVWy7532jUYtrx=
            </wsse:KeyIdentifier>
          </ds:X509Data>
        </ds:KeyInfo>
        <xenc:CipherData>
          <xenc:CipherValue>
            AuFhiu72+1kaJiAuFhiu72+1kaJi=
          </xenc:CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedKey>
    </ds:KeyInfo>
  </SubjectConfirmation>
</Subject>
```

The following example illustrates the content of the `saml:Subject` element when subject confirmation method is “holder of key” using an asymmetric proof key.

Example:

```
<Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:ds="...">
  <SubjectConfirmation>
    <ConfirmationMethod>
      urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
    </ConfirmationMethod>
    <ds:KeyInfo>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
      <!-- asymmetric RSA public key as proof key -->
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>... FntQyKi6R/E4b+1vDH4gwS5ihsU ...</ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
  </SubjectConfirmation>
</Subject>
```

The following example illustrates the content of the `saml:Subject` element when subject confirmation method is “bearer” using no proof key.

Example:

```
<Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
  <SubjectConfirmation>
    <ConfirmationMethod>
      urn:oasis:names:tc:SAML:1.0:cm:bearer
    </ConfirmationMethod>
  </SubjectConfirmation>
</Subject>
```

7.3 Self-Issued Token Encryption

One of the goals of the Information Card Model is to ensure that any claims are exposed only to the Relying Party intended by the user. For this reason, the SIP SHOULD encrypt the self-issued token under the key of the Relying Party. This guarantees that a token intended for one Relying Party cannot be decoded by nor be meaningful to another Relying Party. As described in Section 8.3, when the Relying Party is not identified by a certificate, because no key is available for the Relying Party in this case, the token can not be encrypted, but SHOULD still be signed.

When a self-issued token is encrypted, the XML encryption [XMLENC] standard MUST be used. The encryption construct MUST use encrypting the self-issued token with a randomly generated symmetric key which in turn is encrypted to the Relying Party’s public key taken from its X.509 v3 certificate. The encrypted symmetric key MUST be placed in an `xenc:EncryptedKey` element within the `xenc:EncryptedData` element carrying the encrypted Security Token.

It is RECOMMENDED that the XML encryption [XMLENC] profile that is used for encrypting the key and the token be as follows. Usage of other algorithms is not described.

- Uses the RSA-OAEP key wrap method identified by the algorithm identifier “*http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p*” for encrypting the encryption key.
- Uses the AES256 with CBC encryption method identified by the algorithm “*http://www.w3.org/2001/04/xmlenc#aes256-cbc*” for encrypting the token. The padding method used is as per the PKCS-7 standard in which the number of octets remaining in the last block is used as the padding octet value.
- The `ds:KeyInfo` element is present in the encrypted key specifying the encryption key information in the form of a Security Token reference.

Following is an illustration of a self-issued token encrypted to a Relying Party using the encryption structure described above.

Example:

```
<xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
    xmlns:xenc="..." xmlns:ds="..." xmlns:wsse="...">
  <xenc:EncryptionMethod
     Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc" />
  <ds:KeyInfo>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
    <xenc:EncryptedKey>
      <xenc:EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      </xenc:EncryptionMethod>
      <ds:KeyInfo>
        <wsse:SecurityTokenReference>
          <wsse:KeyIdentifier
            ValueType="http://docs.oasis-open.org/wss/2004/xx/oasis-2004xx-
wss-soap-message-security-1.1#ThumbprintSHA1"
            EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis200401-
wss-soap-message-security-1.0#Base64Binary">
            +PYbznDaB/dlhjIfqCQ458E72wA=
          </wsse:KeyIdentifier>
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>...Ukasdj8257Fjwf=</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedKey>
  </ds:KeyInfo>
  <xenc:CipherData>
    <!-- Start encrypted Content
    <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
        AssertionID="urn:uuid:08301dba-d8d5-462f-85db-dec08c5e4e17" ...>
      ...
    </Assertion>
    End encrypted content -->
    <xenc:CipherValue>...aKlh4817JerpZoDofy90=</xenc:CipherValue>
  </xenc:CipherData>
</xenc:EncryptedData>
```

7.4 Self-Issued Token Signing Key

The key used to sign a self-issued token presented to a Relying Party also represents a unique identifier for the Subject of the token. In order to prevent the key from becoming a correlation identifier across relying parties, a SIP SHOULD use a different key to sign a self-issued token for each Relying Party where the card is used. In other words, the key used to sign the self-issued token is pair-wise unique for a given Information Card and RP combination. To allow self-issued identities created by a SIP within one Identity Selector to be used in another, the signing keys used by the two SIPs SHOULD be the same.

It is RECOMMENDED that the signing key be an RSA key. Usage of other algorithms is not described.

This section specifies the “processing rules” that SHOULD be used by a SIP to derive the RSA key used to sign the self-issued token for a combination of an Information Card and an RP where the card is used. Each self-issued Information Card contains a 256-bit secret random number, called the “master key” (see Section 6.1), that is used as the secret entropy in deriving the token signing RSA key. (Managed Information Cards also have a master key that is used in the Client Pseudonym PPID calculation, as per Section 3.3.4.1.)

Key derivation is done according to the ANSI X9.31 standard for key generation which starts with requiring the use of six random values denoted by X_{p1}, X_{p2}, X_{q1}, X_{q2}, X_p, and X_q. The processing rules described here enunciate how to transform the master key in an Information Card into the six random inputs for the X9.31 key generation process. The actual key computation algorithm in the X9.31 standard is *not* reproduced here.

The values X_p and X_q are REQUIRED to be at least 512 bits and each independently carries the full entropy of any Information Card master key of up to 512 bits in length. The values X_{p1}, X_{p2}, X_{q1}, and X_{q2} have a length of only 100 to 121 bits and therefore will be shorter than the Information Card master key and hence cannot each independently carry the full master key entropy. The details of the X9.31 protocol,

Copyright © OASIS® 2008-2009. All Rights Reserved.

however, ensure that for reasonably sized master keys, full entropy will be achieved in the generated asymmetric key pair.

7.4.1 Processing Rules

This key generation mechanism can be used to generate 1024 or 2048-bit RSA keys.

Notation: If H is an *n*-bit big-endian value, the convention H[1..p] denotes bits *1* through *p* in the value of H where $p \leq n$, and bit-1 is the rightmost (least significant) bit whereas bit-n is the leftmost (most significant) bit in the value of H. Also, the convention X + Y denotes the concatenation of the big-endian bit value of X followed by the big-endian bit value of Y.

Assume that the master key for the selected Information Card (see Section 6.1) is M and the unique *RP Identifier* (derived as per Section 7.6.1) is T. The following processing rules SHOULD be used to derive the inputs for the X9.31 key generation process.

1. Define 32-bit DWORD constants C_n as follows:

   $C_n = n$, where $n = 0,1,2,...,15$

2. Compute SHA-1 hash values H_n as follows:

   If the requested key size = 1024 bits, compute

   $H_n = SHA1 (M + T + C_n)$ for $n = 0,1,2,...,9$

   If the requested key size = 2048 bits, compute

   $H_n = SHA1 (M + T + C_n)$ for $n = 0,1,2,...,15$

3. Extract the random input parameters for the X9.31 protocol as follows:

   For all key sizes, compute

   X_{p1} [112-bits long] = $H_0[1..112]$

   X_{p2} [112-bits long] = $H_1[1..112]$

   X_{q1} [112-bits long] = $H_2[1..112]$

   X_{q2} [112-bits long] = $H_3[1..112]$

   If the requested key size = 1024 bits, compute

   X_p [512-bits long] = $H_4[1..160] + H_5[1..160] + H_6[1..160] + H_0[129..160]$

   X_q [512-bits long] = $H_7[1..160] + H_8[1..160] + H_9[1..160] + H_1[129..160]$

   If the requested key size = 2048 bits, compute

   X_p [1024-bits long] = $H_4[1..160] + H_5[1..160] + H_6[1..160] + H_0[129..160] +$
   $H_{10}[1..160] + H_{11}[1..160] + H_{12}[1..160] + H_2[129..160]$

   X_q [1024-bits long] = $H_7[1..160] + H_8[1..160] + H_9[1..160] + H_1[129..160] +$
   $H_{13}[1..160] + H_{14}[1..160] + H_{15}[1..160] + H_3[129..160]$

4. The X9.31 specification (Section 4.1.2) requires that the input values X_{p1}, X_{p2}, X_{q1}, X_{q2} MUST satisfy the following conditions.

   - The large prime factors p_1, p_2, q_1, and q_2 are the first primes greater than their respective random X_{p1}, X_{p2}, X_{q1}, X_{q2} input values. They are randomly selected from the set of prime numbers between 2^{100} and 2^{120}, and each SHALL pass at least 27 iterations of Miller-Rabin.

Copyright © OASIS® 2008-2009. All Rights Reserved.

To ensure that the lower bound of 2^{100} is met, set the 101th bit of X_{p1}, X_{p2}, X_{q1}, X_{q2} to '1' (*i.e.* X_{p1}[13th byte] |= 0x10, X_{p2}[13th byte] |= 0x10, X_{q1}[13th byte] |= 0x10, X_{q2}[13th byte] |= 0x10).

5. The X9.31 specification (Section 4.1.2) requires that the input values X_p and X_q MUST satisfy the following conditions.
   - If the requested key size = 1024 bits, then
     $X_p \geq (\sqrt{2})(2^{511})$ and $X_q \geq (\sqrt{2})(2^{511})$
   - If the requested key size = 2048 bits, then
     $X_p \geq (\sqrt{2})(2^{1023})$ and $X_q \geq (\sqrt{2})(2^{1023})$

   To ensure this condition is met, set the two most significant bits of X_p and X_q to '1' (*i.e.* X_p[most significant byte] |= 0xC0, X_q[most significant byte] |= 0xC0).
6. Compute 1024 or 2048-bit keys as per the X9.31 protocol using {X_{p1}, X_{p2}, X_{q1}, X_{q2}, X_p, X_q} as the random input parameters.
7. Use a 32-bit DWORD size *public exponent* value of 65537 for the generated RSA keys.

There are three conditions as follows in the X9.31 specification which, if not met, require that one or more of the input parameters MUST be regenerated.

- (Section 4.1.2 of X9.31) $|X_p - X_q| \geq 2^{412}$ (for 1024-bit keys) or $|X_p - X_q| \geq 2^{924}$ (for 2048-bit keys). If not true, X_q MUST be regenerated and q recomputed.
- (Section 4.1.2 of X9.31) $|p - q| \geq 2^{412}$ (for 1024-bit keys) or $|p - q| \geq 2^{924}$ (for 2048-bit keys). If not true, X_q MUST be regenerated and q recomputed.
- (Section 4.1.3 of X9.31) $d > 2^{512}$ (for 1024-bit keys) or $d > 2^{1024}$ (for 2048-bit keys). If not true, X_{q1}, X_{q2}, and X_q MUST be regenerated and key generation process repeated.

When it is necessary to regenerate an input parameter as necessitated by one or more of the conditions above, it is essential that the regeneration of the input parameter be deterministic to guarantee that all implementations of the key generation mechanism will produce the same results. Furthermore, input regeneration is a potentially unlimited process. In other words, it is possible that regeneration MUST be performed more than once. In theory, one MAY need to regenerate input parameters many times before a key that meets all of the requirements can be generated.

The following processing rules MUST be used for regenerating an input parameter *X* of length *n-bits* when necessary:

a. Pad the input parameter *X* on the right, assuming a big-endian representation, with *m* zero-bits where *m* is the smallest number which satisfies *((n+m) mod 128 = 0)*.

b. Encrypt the padded value with the AES-128 (**E**lectronic **C**ode **B**ook mode) algorithm using the 16-byte constant below as the encryption key:

Encryption Key	{ 0x8b, 0xe5, 0x61, 0xf5, 0xbc, 0x3e, 0x0c, 0x4e, 0x94, 0x0d, 0x0a, 0x6d, 0xdc, 0x21, 0x9d, 0xfd }

c. Use the leftmost *n-bits* of the result above as the REQUIRED regenerated parameter.

If a regenerated parameter does not satisfy the necessary conditions, then repeat the 3-step process above (call it *RegenFunction*) to generate the parameter again by using the output of one iteration as input for the next iteration. In other words, if the output of the i^{th} iteration of the regeneration function above for an input parameter *X* is given by X_i then

$$X_{i+1} = RegenFunction\ (X_i)$$

Copyright © OASIS® 2008-2009. All Rights Reserved.

7.5 Claim Types

This section specifies a set of claim (attribute) types and the corresponding URIs that is defined by this profile for some commonly used personal information. These claim types MAY be used by a SIP, in self-issued tokens, or by other Identity Providers. Note that, wherever possible, the claims included here reuse and refer to the attribute semantics defined in other established industry standards that deal with personal information. A SIP SHOULD support these claim types at a minimum. Other Identity Providers MAY also support these claim types when appropriate. The URIs defined here MAY be used by a Relying Party to specify requested claims in its policy.

The base XML namespace URI that is used by the claim types defined here is as follows:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/claims
```

For convenience, an XML Schema for the claim types defined here can be found at:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/claims.xsd
```

7.5.1 First Name

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname*

Type: *xs:string*

Definition: (*givenName* in [RFC 2256]) Preferred name or first name of a Subject. According to RFC 2256: “This attribute is used to hold the part of a person’s name which is not their surname nor middle name.”

7.5.2 Last Name

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname*

Type: *xs:string*

Definition: (*sn* in [RFC 2256]) Surname or family name of a Subject. According to RFC 2256: “This is the X.500 surname attribute which contains the family name of a person.”

7.5.3 Email Address

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress*

Type: *xs:string*

Definition: (*mail* in inetOrgPerson) Preferred address for the “To:” field of email to be sent to the Subject, usually of the form <user>@<domain>. According to inetOrgPerson using [RFC 1274]: “This attribute type specifies an electronic mailbox attribute following the syntax specified in RFC 822.”

7.5.4 Street Address

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/streetaddress*

Type: *xs:string*

Definition: (*street* in [RFC 2256]) Street address component of a Subject’s address information. According to RFC 2256: “This attribute contains the physical address of the object to which the entry corresponds, such as an address for package delivery.” Its content is arbitrary, but typically given as a PO Box number or apartment/house number followed by a street name, *e.g.* 303 Mulberry St.

7.5.5 Locality Name or City

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/locality*

Type: *xs:string*

Copyright © OASIS® 2008-2009. All Rights Reserved.

Definition: (*l* in [RFC 2256]) Locality component of a Subject's address information. According to RFC 2256: "This attribute contains the name of a locality, such as a city, county or other geographic region." *e.g.* Redmond.

7.5.6 State or Province

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/stateorprovince*

Type: *xs:string*

Definition: (*st* in [RFC 2256]) Abbreviation for state or province name of a Subject's address information. According to RFC 2256: "This attribute contains the full name of a state or province. The values SHOULD be coordinated on a national level and if well-known shortcuts exist - like the two-letter state abbreviations in the US – these abbreviations are preferred over longer full names." *e.g.* WA.

7.5.7 Postal Code

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/postalcode*

Type: *xs:string*

Definition: (*postalCode* in X.500) Postal code or zip code component of a Subject's address information. According to X.500(2001): "The postal code attribute type specifies the postal code of the named object. If this attribute value is present, it will be part of the object's postal address - zip code in USA, postal code for other countries."

7.5.8 Country

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/country*

Type: *xs:string*

Definition: (*c* in [RFC 2256]) Country of a Subject. According to RFC 2256: "This attribute contains a two-letter ISO 3166 country code."

7.5.9 Primary or Home Telephone Number

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/homephone*

Type: *xs:string*

Definition: (*homePhone* in inetOrgPerson) Primary or home telephone number of a Subject. According to inetOrgPerson using [RFC 1274]: "This attribute type specifies a home telephone number associated with a person." Attribute values SHOULD follow the agreed format for international telephone numbers, *e.g.* +44 71 123 4567.

7.5.10 Secondary or Work Telephone Number

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/otherphone*

Type: *xs:string*

Definition: (*telephoneNumber* in X.500 Person) Secondary or work telephone number of a Subject. According to X.500(2001): "This attribute type specifies an office/campus telephone number associated with a person." Attribute values SHOULD follow the agreed format for international telephone numbers, *e.g.* +44 71 123 4567.

7.5.11 Mobile Telephone Number

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/mobilephone*

Type: *xs:string*

Definition: (*mobile* in inetOrgPerson) Mobile telephone number of a Subject. According to inetOrgPerson using [RFC 1274]: "This attribute type specifies a mobile telephone number associated

Copyright © OASIS® 2008-2009. All Rights Reserved.

with a person." Attribute values SHOULD follow the agreed format for international telephone numbers, *e.g.* +44 71 123 4567.

7.5.12 Date of Birth

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/dateofbirth*

Type: *xs:date*

Definition: The date of birth of a Subject in a form allowed by the *xs:date* data type.

7.5.13 Gender

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/gender*

Type: *xs:token*

Definition: Gender of a Subject that can have any of these exact string values – '0' (meaning unspecified), '1' (meaning Male) or '2' (meaning Female). Using these values allows them to be language neutral.

7.5.14 Private Personal Identifier

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/privatepersonalidentifier*

Type: *xs:base64binary*

Definition: A private personal identifier (PPID) that identifies the Subject to a Relying Party. The word "private" is used in the sense that the Subject identifier is specific to a given Relying Party and hence private to that Relying Party. A Subject's PPID at one Relying Party cannot be correlated with the Subject's PPID at another Relying Party. Typically, the PPID SHOULD be generated by an Identity Provider as a pair-wise pseudonym for a Subject for a given Relying Party. For a self-issued Information Card, the Self-issued Identity Provider in an Identity Selector system SHOULD generate a PPID for each Relying Party as a function of the card identifier and the Relying Party's identity. The processing rules and encoding of the PPID claim value is specified in Section 7.6.

Compatibility Note: Some existing Identity Selectors omit listing the PPID claim as an `ic:SupportedClaimType` from the `ic:SupportedClaimTypeList` when saving a self-issued Information Card in the Information Cards Transfer Format defined in Section 6.1, even though the PPID claim is supported by the card. This behavior is deprecated, as all supported claims SHOULD be listed. Nonetheless, Identity Selectors MAY choose to recognize this case and support the PPID claim for self-issued cards not explicitly listing this claim.

7.5.15 Web Page

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/webpage*

Type: *xs:string*

Definition: The Web page of a Subject expressed as a URL.

7.6 The PPID Claim

The PPID claim for a Subject user represents a unique identifier for that user at a given Relying Party that is different from all identifiers for that user at any other Relying Party. In other words, the PPID is a pair-wise unique identifier for a given user identity and Relying Party combination. Since an Information Card represents a specific user identity and a Relying Party is the organization behind a Web service or site that the user interacts with, the PPID claim is logically a function of an Information Card and the organizational identity of the Relying Party.

This section describes the processing rules that SHOULD be used by a SIP to derive a PPID claim value for a combination of an Information Card and a Relying Party where it is used.

Copyright © OASIS® 2008-2009. All Rights Reserved.

7.6.1 Relying Party Identifier and Relying Party PPID Seed

In order to derive the PPID and Signing Key as functions of the RP's organizational identity, a stable and unique identifier for the RP, called the *RP Identifier*, is needed. In the Information Card Model, the identity of a Relying Party (RP) possessing an X.509v3 certificate is presented in the form of that certificate. Therefore the organizational identity of the RP is obtained by applying a series of transformations to the identity information carried in the X.509 certificate. (See Section 8 for the specification of how to compute these values for Relying Parties not possessing a certificate.)

As specified in [RFC 2459], the subject field inside an X.509 certificate identifies the entity associated with the public key stored in the subject public key field. Where it is non-empty, the subject field MUST contain an X.500 distinguished name (DN). The DN MUST be unique for each subject entity certified by the one CA as defined by the issuer name field.

The subject field contains a DN of the form shown below:

CN=*string*, [OU=*string*, ...,] O=*string*, L=*string*, S=*string*, C=*string*

The Object Identifiers for these attributes from the DN are as follows:

Field Abbreviation	*Field Name*	*Object Identifier*
O	organizationName	2.5.4.10
L	localityName	2.5.4.7
S	stateOrProvinceName	2.5.4.8
C	countryName	2.5.4.6
CN	commonName	2.5.4.3

Note that the field names and abbreviations used in this specification may not correspond to those used by particular software but the underlying Object Identifiers (OIDs) of the attributes are unambiguous.

For an end-entity certificate, the values of the attribute types O (organizationName), L (localityName), S (stateOrProvinceName) and C (countryName) together uniquely identify the organization to which the end-entity identified by the certificate belongs. These attribute types are collectively referred to as the *organizational identifier attributes* here. The *RP Identifier* is constructed using these organizational identifier attributes as described below.

The *RP Identifier* value is used as an input to the Signing Key computation. A closely related value called the Relying Party PPID Seed is also computed, which is used as an input to the PPID claim and Client Pseudonym PPID computations. In many cases these are the same but in one case they differ.

There are four cases of how the *RP Identifier* and *RP PPID Seed* are constructed depending on which organizational identifier attributes the RP's certificate contains, if it is an extended validation (EV) certificate [EV Cert] with respect to the organizational identifier attributes, and if it chains to a trusted root certificate.

Case 1: RP's certificate *is* EV for organizational identifier attributes and chains to a trusted root certificate authority

- Convert the organizational identifier attributes in the end-entity certificate into a string, call it *OrgIdString*, of the following form:

  |O="*string*"|L="*string*"|S="*string*"|C="*string*"|

  The vertical bar character (ASCII 0x7C) is used as a delimiter at the start and end of the string as well as between the attribute types. Further, the string values of the individual attribute types are enclosed within double quote characters (ASCII 0x22). If an attribute type is absent in the subject field of the end-entity certificate, then the corresponding string value is the empty string (""). Following is an example *OrgIdString* per this convention.

Copyright © OASIS® 2008-2009. All Rights Reserved.

|O="Microsoft"|L="Redmond"|S="Washington"|C="US"|

- Encode all the characters in *OrgIdString* into a sequence of bytes, call it *OrgIdBytes*, using Unicode encoding UTF-16LE with no byte order mark.
- Hash *OrgIdBytes* using the SHA256 hash function, and use the resulting value as the *RP Identifier* and *RP PPID Seed*.

RP PPID Seed = RP Identifier = SHA256 (OrgIdBytes)

Case 2: RP's certificate *is not* EV for organizational identifier attributes, has a non-empty organizationName (O) value, and chains to a trusted root certificate authority

- Convert the organizational identifier attributes in the end-entity certificate into a string, call it *OrgIdString*, in the same manner as employed for Case 1 above.
- Let *QualifierString* be the string:

|Non-EV

- Let *QualifiedOrgIdString* be the concatenation of *QualifierString* and *OrgIdString*.

QualifiedOrgIdString = QualifierString + OrgIdString

- Encode all the characters in *QualifiedOrgIdString* into a sequence of bytes, call it *QualifiedOrgIdBytes*, using Unicode encoding UTF-16LE with no byte order mark.
- Hash *QualifiedOrgIdBytes* using the SHA256 hash function, and use the resulting value as the *RP Identifier*.

RP Identifier = SHA256 (QualifiedOrgIdBytes)

- Encode all the characters in *OrgIdString* into a sequence of bytes, call it *OrgIdBytes*, using Unicode encoding UTF-16LE with no byte order mark.
- Hash *OrgIdBytes* using the SHA256 hash function, and use the resulting value as the *Relying Party PPID Seed*.

RP PPID Seed = SHA256 (OrgIdBytes)

Case 3: RP's certificate has an empty or no organizationName (O) value and has an empty or no commonName (CN) or does not chain to a trusted root certificate authority

- Take the subject public key in the end-entity certificate, call it *PublicKey*, as a byte array.
- Hash *PublicKey* using the SHA256 hash function, and use the resulting value as the *RP Identifier* and *RP PPID Seed*.

RP PPID Seed = RP Identifier = SHA256 (PublicKey)

Case 4: RP's certificate has an empty or no organizationName (O) value but has a non-empty commonName (CN) value and chains to a trusted root certificate authority

- Convert the commonName attribute value in the end-entity certificate into a string, call it *CnIdString*, of the following form:

|CN="*string*"|

Following is an example *CnIdString* per this convention:

|CN="login.live.com"|

- Encode all the characters in *CnIdString* into a sequence of bytes, call it *CnIdBytes*, using Unicode encoding UTF-16LE with no byte order mark.
- Hash *CnIdBytes* using the SHA256 hash function, and use the resulting value as the *RP Identifier* and *RP PPID Seed*.

RP PPID Seed = RP Identifier = SHA256 (CnIdBytes)

Copyright © OASIS® 2008-2009. All Rights Reserved.

7.6.2 PPID

The PPID value SHOULD be produced as follows using the card identifier and the *RP PPID Seed* (specified in Section 7.6.1):

- Encode the value of the `ic:CardId` element of the Information Card into a sequence of bytes, call it *CardIdBytes*, using Unicode encoding UTF-16LE with no byte order mark.
- Hash *CardIdBytes* using the SHA256 hash function to obtain the canonical card identifier *CanonicalCardId*.

  CanonicalCardId = SHA256 (CardIdBytes)
- Hash the concatenation of *RP PPID Seed* and *CanonicalCardId* using the SHA256 hash function to obtain the PPID.

  PPID = SHA256 (RP PPID Seed + CanonicalCardId)

7.6.3 Friendly Identifier

The PPID provides an RP-specific identifier for a Subject that is suitable for programmatic processing, but is not a user-friendly identifier. The simple transformation rules specified in this section MAY be used by a SIP, or any other Identity Provider supporting the PPID claim, to create a friendly identifier for use within a Display Token accompanying a Security Token carrying the PPID claim.

The Friendly Identifier has the following characteristics:

- It is encoded as a 10-character alphanumeric string of the form “AAA-AAAA-AAA” grouped into three groups separated by the ‘hyphen’ character (*e.g.*, the string “6QR-97A4-WR5”). Note that the hyphens are used for punctuation only.
- The encoding alphabet does NOT use the numbers ‘0’ and ‘1’, and the letters ‘O’ and ‘I’ to avoid confusion stemming from the similar glyphs used for these numbers and characters. This leaves 8 digits and 24 letters – a total of 32 alphanumeric symbols – as the alphabet for the encoding.

The processing rules used for deriving a Friendly Identifier from a PPID are as follows:

- The PPID value is conveyed as a base64 encoded string inside tokens. Start with the base64 decoded PPID value as input.
- Hash the PPID value using the SHA1 hash function to obtain a hashed identifier.

  HashId = SHA1 (PPID)
- Let the Friendly Identifier be the string “$A_0\ A_1\ A_2$– $A_3\ A_4\ A_5\ A_6$– $A_7\ A_8\ A_9$” where each A_i is an alphanumeric character from the encoding alphabet described above.
- For *i := 0 to 9*, each A_i is determined as below:
  - Take the i^{th} octet of *HashId* (denoted as *HashId[i]*)
  - Find *RawValue = HashId[i] % 32* (where % is the remainder operation)
  - A_i = *EncodedSymbol* obtained by mapping *RawValue* to *EncodedSymbol* using the table below

Raw Value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Encoded Symbol	Q	L	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Copyright © OASIS® 2008-2009. All Rights Reserved.

Raw Value	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Encoded Symbol	G	H	J	K	M	N	P	R	S	T	U	V	W	X	Y	Z

8 Relying Parties without Certificates

While Relying Parties are typically identified by presenting a cryptographically protected identity, such as an X.509v3 certificate, the Information Card Model is also applicable in situations in which no Relying Party certificate is available. This section specifies how Information Cards are used at Relying Parties with no certificate: specifically, Web sites using the [HTTP] scheme. Also see `ic07:RequireStrongRecipientIdentity` in Section 3.1.1.7 for a means whereby card issuers can prohibit the use of cards at Relying Parties not identified by a certificate.

8.1 Relying Party Identifier and Relying Party PPID Seed

The Relying Party Identifier and Relying Party PPID Seed values for Relying Parties without certificates are computed in this manner:

- Set the string *OrgIdString* to be the fully qualified DNS host name in lowercase characters specified in the URI of the Relying Party, or if a numeric IP address was used, then a string representation of the IP address of the server. For IPv4 addresses, this string is the standard 4-byte dotted decimal representation of the address with no leading zeros, such as `131.107.55.210`. For IPv6 addresses, this string is the hexadecimal representation of the address in eight groups of four hex digits each using uppercase for the letters, with each group of four digits separated by a colon, all enclosed by square brackets, such as `[0000:1234:0000:0000:0000:000A:00BC:0DEF]`.
- Encode all the characters in *OrgIdString* into a sequence of bytes, call it *OrgIdBytes*, using the Unicode encoding UTF-16LE with no byte order mark.
- Hash *OrgIdBytes* using the SHA256 hash function, and use the resulting value as both the *RP Identifier* and the *RP PPID Seed*.

The *RP Identifier* and *RP PPID Seed* are then used in the same manner as for Relying Parties identified by certificates when computing PPID claim and Client Pseudonym PPID values.

8.2 AppliesTo Information

Under the circumstances described in Section 3.3.3 that the RP endpoint to which the token will be sent is supplied as the `wsp:AppliesTo` value to the IP, when the RP possesses no certificate, the URL of the RP is supplied as that `wsp:AppliesTo` value.

Example:

```
<wst:RequestSecurityToken xmlns:wst="..." xmlns:wsp="..." xmlns:wsa="...">
  <wsp:AppliesTo>
    <wsa:EndpointReference>
      <wsa:Address>http://login.contoso.com</wsa:Address>
    </wsa:EndpointReference>
  </wsp:AppliesTo>
  ...
</wst:RequestSecurityToken>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

8.3 Token Signing and Encryption

When the Relying Party is not identified by a certificate, tokens sent from the Self-issued Identity Provider are not encrypted, although they are still signed in the manner described in Section 7.2. Tokens generated by Identity Providers for Relying Parties not identified by a certificate are also typically not encrypted, as no encryption key is available. However, the token MAY still be encrypted if the Identity Provider has a pre-existing relationship with the Relying Party and they have mutually agreed on the use of a known encryption key. The token SHOULD still typically be signed, even when not encrypted.

9 Using WS-SecurityPolicy 1.2 and WS-Trust 1.3

Software implementing the Information Card Model SHOULD utilize the OASIS standard versions of WS-SecurityPolicy and WS-Trust – [WS-SecurityPolicy 1.2] and [WS-Trust 1.3] and MAY utilize the previous draft versions – [WS-SecurityPolicy 1.1] and [WS-Trust 1.2]. This section describes the differences between the old and standard versions of these protocols that MAY affect software implementing the Information Card Model.

9.1 Overview of Differences

The following changes between the protocol versions affect software implementing this specification:

- **Namespace changes:**
  http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702 replaces
  http://schemas.xmlsoap.org/ws/2005/07/securitypolicy.
  http://docs.oasis-open.org/ws-sx/ws-trust/200512 replaces
  http://schemas.xmlsoap.org/ws/2005/02/trust.
- **Use of RequestSecurityTokenResponseCollection:** A `wst:RequestSecurityTokenResponseCollection` element encloses the `wst:RequestSecurityTokenResponse` when WS-Trust 1.3 is used.
- **Use of SecondaryParameters:** An Identity Selector sends some information received from the Relying Party to the Identity Provider in a `wst:SecondaryParameters` element.
- **Bearer Token Request Syntax:** The new `wst:KeyType` value http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer is used to request a bearer token.

9.2 Identity Selector Differences

Identity Selectors MUST determine the WS-Trust versions used by Identity Provider STSs and Relying Party STSs using their Security Policy.

Identity Selectors supporting WS-Trust 1.3 MUST understand the new WS-Trust 1.3 elements and syntax such as `wst13:RequestSecurityTokenResponseCollection` and new URIs such as http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer. They MUST also understand that typical properties of an RST like Claims and KeyType MAY be either a direct child of the top level `wst13:RequestSecurityToken` element or contained within a `wst13:SecondaryParameters` element in the RST.

When constructing an RST for an Identity Provider using WS-Trust 1.3, the Identity Selector SHOULD send parameters received from the Relying Party in a `wst13:SecondaryParameters` element within the `wst13:RequestSecurityToken`, with these exceptions:

- The user chooses not to send optional claims. In this scenario, no SecondaryParameters element is sent in order to hide this user decision.
- No `wsp:AppliesTo` is being sent in the RST. In this scenario, no `wst13:SecondaryParameters` element is sent so that the Identity Provider does not obtain any identifying information about the Relying Party.

Copyright © OASIS® 2008-2009. All Rights Reserved.

Example:

```
<wst13:RequestSecurityToken Context="ProcessRequestSecurityToken"
    xmlns:wst13="..." xmlns:ic="...">
  <wst13:RequestType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue</wst13:RequestType>
  <ic:InformationCardReference>
  ...
  </ic:InformationCardReference>
  <wst13:Claims Dialect="http://schemas.xmlsoap.org/ws/2005/05/identity">
  ...
  </wst13:Claims>
  <wst13:KeyType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/SymmetricKey</wst13:KeyType>
  <wst13:SecondaryParameters>
    <wst13:RequestType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue</wst13:RequestType>
    <wst13:TokenType>urn:oasis:names:tc:SAML:1.0:assertion</wst13:TokenType>
    <wst13:KeyType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/SymmetricKey</wst13:KeyType>
    <wst13:KeyWrapAlgorithm>http://www.w3.org/2001/04/xmlenc#rsa-oaep-
mgf1p</wst13:KeyWrapAlgorithm>
    ...
  </wst13:SecondaryParameters>
</wst13:RequestSecurityToken>
```

The `wst13:RequestSecurityTokenResponse` constructed MUST be enclosed within a `wst13:RequestSecurityTokenResponseCollection` element.

Example:

```
<wst13:RequestSecurityTokenResponseCollection xmlns:wst13="...">
  <wst13:RequestSecurityTokenResponse>
    <wst13:TokenType>urn:oasis:names:tc:SAML:1.0:assertion</wst13:TokenType>
    <wst13:RequestedSecurityToken> ... </wst13:RequestedSecurityToken>
    ...
  </wst13:RequestSecurityTokenResponse>
</wst13:RequestSecurityTokenResponseCollection>
```

9.3 Security Token Service Differences

To utilize WS-Trust 1.3, an Identity Provider STS and Relying Party STSs MUST express their Security Policy using WS-SecurityPolicy 1.2.

STSs using WS-Trust 1.3 MUST understand the new WS-Trust 1.3 elements and syntax such as `wst13:RequestSecurityTokenResponseCollection` and new URIs such as http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer. They MUST also understand that typical properties of an RST like Claims and KeyType MAY be either a direct child of the top level `wst13:RequestSecurityToken` element or contained within a `wst13:SecondaryParameters` element in the RST.

Copyright © OASIS® 2008-2009. All Rights Reserved.

10 Browser Behavior with Information Cards

This section explains the steps that a Web browser takes when using an Information Card to authenticate to a Web site. Two cases are described. The basic case is where the Web site provides all the Relying Party functionality via HTML extensions transported over HTTPS. The second case is where the Relying Party employs a Relying Party Security Token Service (STS), which it references via HTML extensions transported over HTTPS.

10.1 Basic Protocol Flow when using an Information Card at a Web Site

This section explains the protocol flow when using an Information Card to authenticate at a Web site where no Relying Party STS is employed.

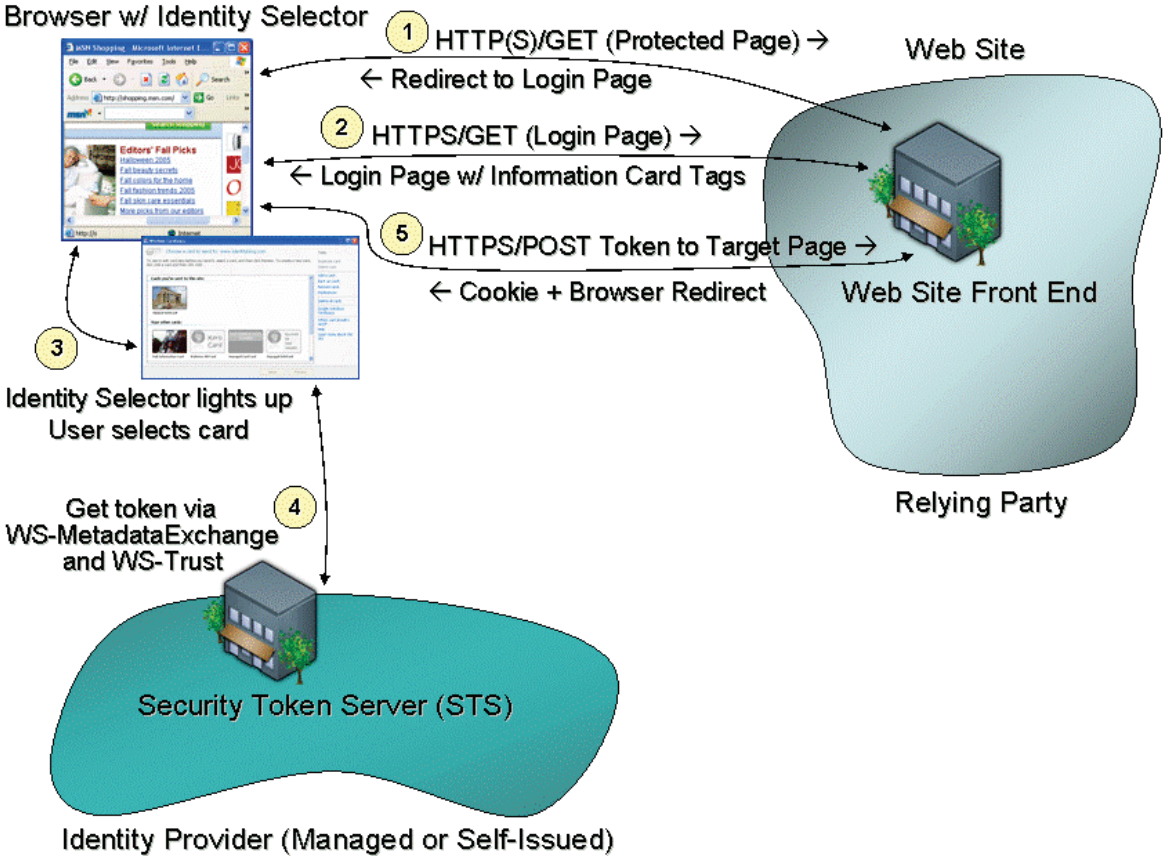


Figure 1. Basic protocol flow when using an Information Card to authenticate at a Web site

Figure 1 gives an example of the basic protocol flow when an Information Card is used to authenticate at a Web site that employs no Relying Party STS. Steps 1, 2, and 5 are essentially the same as a typical forms-based login today: (1) The user navigates to a protected page that requires authentication. (2) The site redirects the browser to a login page, which presents a Web form. (5) The browser posts the Web form that includes the login credentials supplied by the user back to the login page. The site then validates the contents of the form including the user credentials, typically writes a client-side browser cookie to the client for the protected page domain, and redirects the browser back to the protected page.

Copyright © OASIS® 2008-2009. All Rights Reserved.

The key difference between this scenario and today's site login scenarios is that the login page returned to the browser in step (2) contains an HTML tag that allows the user to choose to use an Information Card to authenticate to the site. When the user selects this tag, the browser invokes an Identity Selector, which implements the Information Card user experience and protocols, and triggers steps (3) through (5).

In Step (3), the browser Information Card support code invokes the Identity Selector, passing it parameter values supplied by the Information Card HTML tag supplied by the site in Step (2). The user then uses the Identity Selector to choose an Information Card, which represents a Digital Identity that can be used to authenticate at that site. Step (4) retrieves a Security Token that represents the Digital Identity selected by the user from the STS at the Identity Provider for that identity.

In Step (5), the browser posts the token obtained back to the Web site using a HTTPS/POST. The Web site validates the token, completing the user's Information Card-based authentication to the Web site. Following authentication, the Web site typically then writes a client-side browser cookie and redirects the browser back to the protected page.

It is worth noting that this cookie is likely to be *exactly the same cookie* as the site would have written back had the user authenticated via other means, such as a forms-based login using username/password. This is one of the ways that the goal of "minimal impact on Web sites" is achieved. Other than its authentication subsystem, the bulk of a Web site's code can remain completely unaware that Information Card-based authentication is even utilized. It just uses the same kinds of cookies as always.

10.2 Protocol Flow with Relying Party STS

In the previous scenario, the Web site communicated with the client Identity Selector using only the HTML extensions enabling Information Card use, transported over the normal browser HTTPS channel. In this scenario, the Web site also employs a Relying Party STS to do part of the work of authenticating the user, passing the result of that authentication on to the login page via HTTPS POST.

There are several reasons that a site might factor its solution this way. One is that the same Relying Party STS can be used to do the authentication work for both browser-based applications and smart client applications that are using Web services. Second, it allows the bulk of the authentication work to be done on servers dedicated to this purpose, rather than on the Web site front-end servers. Finally, this means that the front-end servers can accept site-specific tokens, rather than the potentially more general or more complicated authentication tokens issued by the Identity Providers.

Copyright © OASIS® 2008-2009. All Rights Reserved.

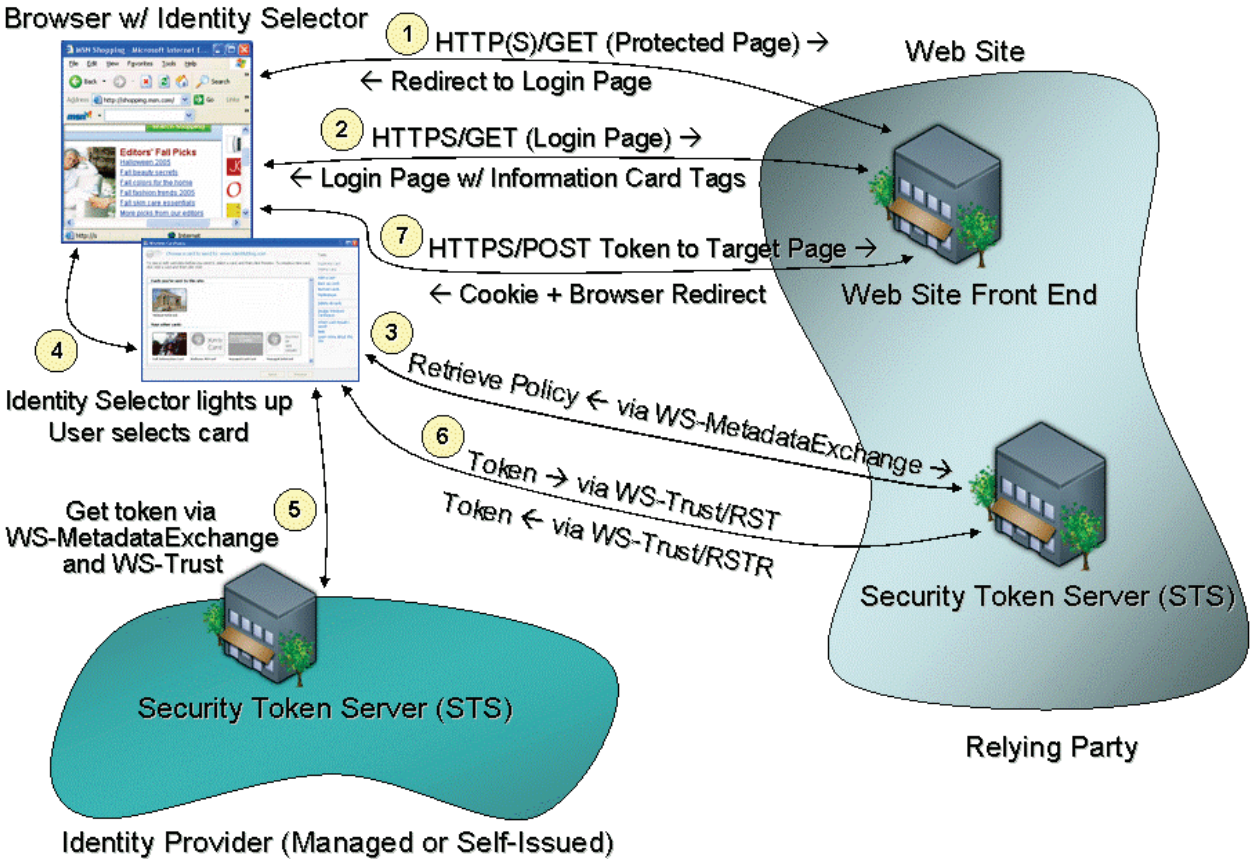


Figure 2. Protocol flow when using an Information Card to authenticate at a Web site, where the Web site employs a Relying Party STS

This scenario is similar to the previous one, with the addition of steps (3) and (6). The differences start with the Information Card information supplied to the browser by the Web site in Step (2). In the previous scenario, the site encoded its WS-SecurityPolicy information using Information Card HTML extensions and supplied them to the Information Card-extended browser directly. In this scenario, the site uses different Information Card HTML extensions in the Step (2) reply to specify which Relying Party STS SHOULD be contacted to obtain the WS-SecurityPolicy information.

In Step (3), the Identity Selector contacts the Relying Party STS specified by the Web site and obtains its WS-SecurityPolicy information via WS-MetadataExchange. In Step (4) the Identity Selector user interface is shown and the user selects an Information Card, which represents a Digital Identity to use at the site. In Step (5), the Identity Provider is contacted to obtain a Security Token for the selected Digital Identity. In Step (6), the Security Token is sent to the Web site's Relying Party STS to authenticate the user and a site-specific authentication token is returned to the Identity Selector. Finally, in Step (7), the browser posts the token obtained in Step (6) back to the Web site using HTTPS/POST. The Web site validates the token, completing the user's Information Card-based authentication to the Web site. Following authentication, the Web site typically then writes a client-side browser cookie and redirects the browser back to the protected page.

10.3 User Perspective and Examples

The Information Card user experience at Web sites is intended to be intuitive and natural enough that users' perspective on it will simply be "That's how you log in". Today, Web sites that require authentication typically ask the user to supply a username and password at login time. With Information Cards, they instead ask users to choose an Information Card. Some sites will choose to accept only

Copyright © OASIS® 2008-2009. All Rights Reserved.

Information Cards whereas others will give users the choice of Information Cards or other forms of authentication.

A site that accepts Information Cards typically has a login screen that contains button with a label such as "**Sign in with an Information Card**" or "**Log in using an Information Card**". Upon clicking this button, the user is presented with a choice of his Information Cards that are accepted at the site, and is asked to choose one. Once a card is selected and submitted to the site, the user is logged in and continues using the site, just as they would after submitting a username and password to a site.

Sites that accept both Information Cards and other forms of authentication present users with both an Information Card login choice and whatever other choices the site supports. For instance, a site login screen might display both "**Sign in with your username and password**" and "**Sign in with an Information Card**" buttons.

10.4 Browser Perspective

Very little additional support is needed from today's Web browsers to also support Information Cards. The main addition is that they MUST recognize special HTML and/or XHTML tags for invoking the Identity Selector, pass encoded parameters on to the Identity Selector on the platform, and POST back the token resulting from the user's choice of an Information Card.

10.5 Web Site Perspective

Web sites that employ Information Card-based authentication MUST support two new pieces of functionality: adding HTML or XHTML tags to their login page to request an Information Card-based login and code to log the user into the site using the POSTed credentials. In response to the Information Card-based login, the Web site typically writes the same client-side browser cookie that it would have if the login had occurred via username/password authentication or other mechanisms, and issue the same browser redirects. Thus, other than the code directly involved with user authentication, the bulk of a Web site can remain unchanged and oblivious to the site's acceptance of Information Cards as a means of authentication.

Copyright © OASIS® 2008-2009. All Rights Reserved.

11 Invoking an Identity Selector from a Web Page

11.1 Syntax Alternatives: OBJECT and XHTML tags

HTML extensions are used to signal to the browser when to invoke the Identity Selector. However, not all HTML extensions are supported by all browsers, and some commonly supported HTML extensions are disabled in browser high security configurations. For example, while the OBJECT tag is widely supported, it is also disabled by high security settings on some browsers, including Internet Explorer.

An alternative is to use an XHTML syntax that is not disabled by changing browser security settings. However, not all browsers provide full support for XHTML.

To address this situation, two HTML extension formats are specified. Browsers MAY support one or both of the extension formats.

11.1.1 OBJECT Syntax Examples

An example of the OBJECT syntax is as follows:

```
<html>
  <head>
    <title>Welcome to Fabrikam</title>
  </head>
  <body>
    <img src='fabrikam.jpg' alt="Fabrikam Logo" />
    <form name="ctl00" id="ctl00" method="post"
        action="https://www.fabrikam.com/InfoCard-Browser/Main.aspx">
      <center>
        <img src='infocard_56x39.png' alt="Information Card Icon"
            onClick='ctl00.submit()' />
        <input type="submit" name="InfoCardSignin" value="Log in"
          id="InfoCardSignin" />
      </center>
      <OBJECT type="application/x-informationCard" name="xmlToken">
        <PARAM Name="tokenType" Value="urn:oasis:names:tc:SAML:1.0:assertion">
        <PARAM Name="issuer" Value=
            "http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self">
        <PARAM Name="requiredClaims" Value=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname">
      </OBJECT>
    </form>
  </body>
</html>
```

This is an example of a page that requests that the user log in using an Information Card. The key portion of this page is the OBJECT of type "`application/x-informationCard`". Once a card is selected by the user, the resulting Security Token is included in the resulting POST as the xmlToken value of the form. Appendix A shows a sample POST resulting from using a login page similar to the preceding one. If the user cancels the authentication request, the resulting POST contains an empty xmlToken value.

Parameters of the Information Card OBJECT are used to encode the necessary WS-SecurityPolicy information in HTML. In this example, the Relying Party is requesting a SAML 1.0 token from a Self-issued Identity Provider, supplying the requested claims "`emailaddress`", "`givenname`", and "`surname`". This example uses the basic protocol described in Section 2.1 (without employing a Relying Party STS).

Copyright © OASIS® 2008-2009. All Rights Reserved.

A second example of the OBJECT syntax is as follows:

```
<html>
  <body>
    <form name="ctl01" method="post"
        action="https://www.fabrikam.com/InfoCard-Browser-STS/login.aspx"
        id="ctl01" onSubmit="fnGetCard();">
      <img src='infocard_56x39.png' alt="Information Card Icon"
          onClick='ctl01.submit()' />
      <input type="submit" name="InfoCardSignin" value="Log in"
          id="InfoCardSignin" />
      <OBJECT type="application/x-informationCard" name="xmlToken"
          ID="oCard" />
    </form>
    <script type="text/javascript">
    <!--
      function fnGetCard(){
        oCard.issuer = "http://www.fabrikam.com/sts";
        oCard.issuerPolicy = "https://www.fabrikam.com/sts/mex";
        oCard.tokenType = "urn:fabricam:custom-token-type";
      }
    //-->
    </script>
  </body>
</html>
```

This example uses the enhanced protocol described in Section 2.3, which employs a Relying Party STS. Note that in this case, the “issuer” points to a Relying Party STS. The “issuerPolicy” points to an endpoint where the Security Policy of the STS (expressed via WS-SecurityPolicy) is to be obtained using WS-MetadataExchange. Also, note that the “tokenType” parameter requests a custom token type defined by the site for its own purposes. The “tokenType” parameter could have been omitted as well, provided that the Web site is capable of understanding all token types issued by the specified STS or if the STS has prior knowledge about the token type to issue for the Web site.

The object parameters can be set in normal script code. This is equivalent to setting them using the “PARAM” declarations in the previous example.

11.1.2 XHTML Syntax Example

An example of the XHTML syntax is as follows:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity">
  <head>
    <title>Welcome to Fabrikam</title>
  </head>
  <body>
    <img src='fabrikam.jpg' alt="Fabrikam Logo" />
    <form name="ctl00" id="ctl00" method="post"
        action="https://www.fabrikam.com/InfoCard-Browser/Main.aspx">
      <ic:informationCard name='xmlToken'
          style='behavior:url(#default#informationCard)'
          issuer="http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self"
          tokenType="urn:oasis:names:tc:SAML:1.0:assertion">
        <ic:add claimType=
       "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"
            optional="false" />
        <ic:add claimType=
         "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname"
            optional="false" />
        <ic:add claimType=
         "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname"
            optional="false" />
      </ic:informationCard>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
      <center>
        <input type="submit" name="InfoCardSignin" value="Log in"
            id="InfoCardSignin" />
      </center>
    </form>
  </body>
</html>
```

11.2 Identity Selector Invocation Parameters

The parameters to the OBJECT and XHTML Information Card objects are used to encode information in HTML that is otherwise supplied as WS-SecurityPolicy information via WS-MetadataExchange when an Identity Selector is used in a Web services context.

11.2.1 issuer

This optional parameter specifies the URL of the STS from which to obtain a token. If omitted, no specific STS is requested. The special value "`http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self`" specifies that the token SHOULD come from a Self-issued Identity Provider.

11.2.2 issuerPolicy

This optional parameter specifies the URL of an endpoint from which the STS's WS-SecurityPolicy can be retrieved using WS-MetadataExchange. This endpoint MUST use HTTPS.

11.2.3 tokenType

This optional parameter specifies the type of the token to be requested from the STS as a URI. This parameter can be omitted if the STS and the Web site front-end have a mutual understanding about what token type will be provided or if the Web site is willing to accept any token type.

11.2.4 requiredClaims

This optional parameter specifies the types of claims that MUST be supplied by the identity. If omitted, there are no required claims. The value of `requiredClaims` is a space-separated list of URIs, each specifying a required claim type.

11.2.5 optionalClaims

This optional parameter specifies the types of optional claims that MAY be supplied by the identity. If omitted, there are no optional claims. The value of `optionalClaims` is a space-separated list of URIs, each specifying a claim type that can MAY be submitted.

11.2.6 privacyUrl

This optional parameter specifies the URL of the human-readable Privacy Policy of the site, if provided.

11.2.7 privacyVersion

This optional parameter specifies the Privacy Policy version. This MUST be a value greater than 0 if a privacyUrl is specified. If this value changes, the UI notifies the user and allows them review the change to the Privacy Policy.

11.3 Data Types for Use with Scripting

The object used in the Information Card HTML extensions has the following type signature, allowing it to be used by normal scripting code:

```
interface IInformationCardSigninHelper
{
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
  string issuer;              // URI specifying token issuer
  string issuerPolicy;        // MetadataExchange endpoint of issuer
  string tokenType;           // URI specifying type of token to be requested
  string [] requiredClaims;   // Array of URIs of required claim types
  string [] optionalClaims;   // Array of URIs of optional claim types
  string privacyUrl;          // URL of the Privacy Policy of the site
  string privacyVersion;      // Version number of the Privacy Policy
  boolean isInstalled;        // True when an Identity Selector is available
                              // to the browser
}
```

11.4 Detecting and Utilizing an Information Card-enabled Browser

Web sites MAY choose to detect browser and Identity Selector support for Information Cards and modify their login page contents depending upon whether Information Card support is present, and which of the OBJECT and/or XHTML syntaxes are supported by the browser and supported by the Web site. This allows Information Card capabilities to be shown when available to the user, and to be not displayed otherwise.

Detecting an Information Card-enabled browser may require detecting specific browser and Identity Selector versions and being aware of the nature of their Information Card support.

11.5 Behavior within Frames

When the object tag is specified in an embedded frame, the certificate of the frame is compared to that of the root frame. For this configuration to work, the scheme, domain, and security zone (for example https, microsoft.com, and Intranet) of the URL of the embedded frame MUST be the same as that of the root frame. If they do not match, the object tag SHOULD NOT be acted upon. This prevents a form of cross-site scripting attacks.

11.6 Invocation Using the Document Object Model (DOM)

In addition to being invokable using static HTML tags and script code, Identity Selectors can be invoked from script injected into the page using the Document Object Model [DOM]. Invocation from dynamically generated script allows the Web site's requirements to be set dynamically.

11.7 Auditing, Non-Auditing, and Auditing-Optional Cards

- **Auditing Card:** When a managed card with an `ic:RequireAppliesTo` element and no `Optional` attribute or `Optional=false` attribute is used at a Web site, the Request Security Token (RST) sent to the Identity Provider contains a `wsp:AppliesTo` element.
- **Non-Auditing Card:** When a managed card with no `ic:RequireAppliesTo` element is used at a Web site, the Request Security Token (RST) sent to the Identity Provider contains no `wsp:AppliesTo` element.
- **Auditing-Optional Card:** When a managed card with an `ic:RequireAppliesTo` element with `Optional=true` attribute is used at a Web site, the Request Security Token (RST) sent to the Identity Provider contains a `wsp:AppliesTo` element.

Copyright © OASIS® 2008-2009. All Rights Reserved.

12 Endpoint Reference wsai:Identity Property

This section adds the `wsai:Identity` property to an Endpoint Reference [WS-Addressing] and leverages extensibility of the `wsa:EndpointReferenceType` schema to include a `wsai:Identity` element as described below:

```
<wsa:EndpointReference xmlns:wsa="..." xmlns:wsai="...">
  ...
  <wsai:Identity>...identity representation...</wsai:Identity>
  ...
</wsa:EndpointReference>
```

The `wsai:Identity` element inside a `wsa:EndpointReference` can hold any of the identity representations defined in Section 12.2 below.

12.1 Default Value

If a `wsa:EndpointReference` does not contain a `wsai:Identity` element, a DNS Name representation can be assumed by extracting the hostname from the Address URI.

If the URI does not have a hostname, it does not have an implicit identity value and can not be verified by the mechanisms defined in this document.

12.2 Identity Representation

12.2.1 DNS Name

The DNS Name representation implies that the remote principal is trusted to speak for that DNS name. For instance the DNS Name representation could specify “fabrikam.com”. When challenged, the endpoint contacted MUST be able to prove its right to speak for “fabrikam.com”. The service could prove its right by proving ownership of a certificate containing a reference to fabrikam.com and signed by a trusted Certificate Authority. The following element of type `xs:string` can be used to represent a DNS Name representation within a `wsai:Identity` element.

```
<wsai:Dns xmlns:wsai="...">fabrikam.com</wsai:Dns>
```

12.2.2 Service Principal Name

The SPN representation implies that the remote principal is trusted to speak for that SPN, a mechanism common in intranet domains. Its format is <serviceClass>/<host>. For example, the SPN for a generic service running on “server1.fabrikam.com” would be “host/server1.fabrikam.com”. The client could confidentially speak to the service and verify replies back from the service by obtaining a Kerberos ticket from the realm’s domain controller. The following element of type `xs:string` can be used to represent an SPN representation within a `wsai:Identity` element.

```
<wsai:Spn xmlns:wsai="...">host/hrweb</wsai:Spn>
```

12.2.3 User Principal Name

The UPN representation implies that the remote principal is a particular user in a domain. Its format is: <user>@<domain>. For example, the UPN for a user “someone” at a domain “example.com” would be “someone@example.com”. A service could prove its UPN by providing the password for the user

Copyright © OASIS® 2008-2009. All Rights Reserved.

associated with “someone@example.com”. The following element of type `xs:string` can be used to represent a UPN representation within a `wsai:Identity` element.

```
<wsai:Upn xmlns:wsai="...">someone@example.com</wsai:Upn>
```

12.2.4 KeyInfo

This identity value is similar to the previous three, but rather than describing an attribute of the target, this mechanism describes a reference (embedded or external) to key material associated with the target. This allows confirmation of the target trust identity through encryption. These values can also be used to compare authenticated identities similar to the basic trust identity values by comparing the hash of the specified trust identity value with a hash of the authenticated identity of the service. The `ds:KeyInfo` element defined in [XMLDSIG] can be used.

```
<ds:KeyInfo xmlns:ds="...">...</ds:KeyInfo>
```

12.2.4.1 Example specifying an RSA Public Key

The PublicKey representation states the public key of the remote principal. A service could prove its ownership of the key by signing some data with the private key.

```
<wsai:Identity xmlns:wsai="..." ds:wsai="...">
  <ds:KeyInfo>
    <ds:RSAKeyValue>
      <ds:Modulus>xA7SEU+e0yQH5...</ds:Modulus>
      <ds:Exponent>AQAB</ds:Exponent>
    </ds:RSAKeyValue>
  </ds:KeyInfo>
</wsai:Identity>
```

12.2.4.2 Example specifying an X509 Certificate

This example shows a certificate of the remote principal being used as the identity value.

```
<wsai:Identity xmlns:wsai="..." xmlns:ds="...">
  <ds:KeyInfo>
    <ds:X509Data>
      <ds:X509Certificate>MIICXTCCA...</ds:X509Certificate>
    </ds:X509Data>
  </ds:KeyInfo>
</wsai:Identity>
```

12.2.5 Security Token

A security token can be an identity value representing statements about the identity of an endpoint. E.g.:

```
<wsai:Identity xmlns:wsai="..." ds:wsse="...">
      <wsse:BinarySecurityToken ValueType="...#X509v3">
            <!--base64 encoded value of the X509 certificate-->
      </wsse:BinarySecurityToken>
</wsai:Identity>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

12.2.6 Security Token Reference

Similarly to `ds:KeyInfo`, `wsse:SecurityTokenReference` element can be used within a `wsai:Identity` element to reference a token representing a collection of statements about the identity of an endpoint. E.g.:

```
<wsai:Identity xmlns:wsai="..." ds:wsse="...">
      <wsse:SecurityTokenReference>
           <wsse:KeyIdentifier ValueType="...#ThumbprintSHA1">
                <!-- thumbprint of the X509 certificate  -->
           </wsse:KeyIdentifier>
      </wsse:SecurityTokenReference>
</wsai:Identity>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

13 Security Considerations

13.1 Protection of Information Cards by Identity Selectors

It is RECOMMENDED that Identity Selectors encrypt or otherwise secure the Information Card data held by them to help protect cards from being stolen and then used by an attacker. This is particularly important for self-issued Information Cards, where possession of the unencrypted contents of a card could enable an attacker to gain access to Relying Parties accounts associated with that card.

13.2 Relying Parties Without Certificates

Because claims sent to relying parties without certificates are not encrypted, it is RECOMMENDED that sensitive claims not be released to these relying parties. Identity Providers holding sensitive user data that can be released as claim values are encouraged to issue cards containing an `ic07:RequireStrongRecipientIdentity` element to prevent transmission of sensitive claim values over an unencrypted channel.

13.3 Endpoint References

It is RECOMMENDED that Endpoint Reference elements be signed to prevent tampering.

An Endpoint Reference SHOULD NOT be accepted unless it is signed and have an associated security token to specify the signer has the right to “speak for” the endpoint. That is, the relying party SHOULD NOT use an endpoint reference unless the endpoint reference is signed and presented with sufficient credentials to pass the relying parties acceptance criteria.

It is RECOMMENDED that an endpoint reference be encrypted when it contains claims and other sensitive information.

When included in a SOAP message, endpoint references are RECOMMENDED to be protected using the mechanisms described in WS-Security [WS-Security]

Copyright © OASIS® 2008-2009. All Rights Reserved.

14 Conformance

An implementation conforms to this specification if it satisfies all of the MUST or REQUIRED level requirements defined within this specification for the portions of the specification implemented by that implementation. Furthermore, when an implementation supports functionality in which there is a RECOMMENDED algorithm or set of parameter choices, conforming implementations MUST support the RECOMMENDED algorithm and parameter choices. A SOAP Node MUST NOT use the XML namespace identifiers for this specification (listed in Section 1.2) within SOAP Envelopes unless it is compliant with this specification.

This specification references a number of other specifications. In order to comply with this specification, an implementation MUST implement the portions of referenced specifications necessary to comply with the required provisions of the portions of this specification that it implements. Additionally, the implementation of the portions of the referenced specifications that are specifically cited in this specification MUST comply with the rules for those portions as established in the referenced specification.

Additionally, normative text within this specification takes precedence over normative outlines (as described in Section 1.1), which in turn take precedence over the XML Schema [XML Schema Part 1, Part 2] and WSDL [WSDL 1.1] descriptions. That is, the normative text in this specification further constrains the schemas and/or WSDL that are part of this specification; and this specification contains further constraints on the elements defined in referenced schemas.

If an OPTIONAL message is not supported, then the implementation SHOULD Fault just as it would for any other unrecognized/unsupported message. If an OPTIONAL message is supported, then the implementation MUST satisfy all of the MUST and REQUIRED sections of the message.

Copyright © OASIS® 2008-2009. All Rights Reserved.

A. HTTPS POST Sample Contents

The contents of an HTTPS POST generated by a page like the first example in Section 4.1.1 follow:

```
POST /test/s/TokenPage.aspx HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Content-Length: 6478
Content-Type: application/x-www-form-urlencoded
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-sh
ockwave-flash, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Host: calebb-tst
Referer: https://localhost/test/s/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30)
UA-CPU: x86

InfoCardSignin=Log+in&xmlToken=%3Cenc%3AEncryptedData+Type%3D%22http%3A%2F%2F
www.w3.org%2F2001%2F04%2Fxmlenc%23Element%22+xmlns%3Aenc%3D%22http%3A%2F%2Fww
w.w3.org%2F2001%2F04%2Fxmlenc%23%22%3E%3Cenc%3AEncryptionMethod+Algorithm%3D%
22http%3A%2F%2Fwww.w3.org%2F2001%2F04%2Fxmlenc%23aes256-cbc%22+%2F%3E%3CKeyIn
fo+xmlns%3D%22http%3A%2F%2Fwww.w3.org%2F2000%2F09%2Fxmldsig%23%22%3E%3Ce%3AEn
cryptedKey+xmlns%3Ae%3D%22http%3A%2F%2Fwww.w3.org%2F2001%2F04%2Fxmlenc%23%22%
3E%3Ce%3AEncryptionMethod+Algorithm%3D%22http%3A%2F%2Fwww.w3.org%2F2001%2F04%
2Fxmlenc%23rsa-oaep-mgf1p%22%3E%3CDigestMethod+Algorithm%3D%22http%3A%2F%2Fww
w.w3.org%2F2000%2F09%2Fxmldsig%23sha1%22+%2F%3E%3C%2Fe%3AEncryptionMethod%3E%
3CKeyInfo%3E%3Co%3ASecurityTokenReference+xmlns%3Ao%3D%22http%3A%2F%2Fdocs.oa
sis-open.org%2Fwss%2F2004%2F01%2Foasis-200401-wss-wssecurity-secext-1.0.xsd%2
2%3E%3Co%3AKeyIdentifier+ValueType%3D%22http%3A%2F%2Fdocs.oasis-open.org%2Fws
s%2Foasis-wss-soap-message-security-1.1%23ThumbprintSHA1%22+EncodingType%3D%2
2http%3A%2F%2Fdocs.oasis-open.org%2Fwss%2F2004%2F01%2Foasis-200401-wss-soap-m
essage-security-1.0%23Base64Binary%22%3E%2BPYbznDaB%2FdlhjIfqCQ458E72wA%3D%3C
%2Fo%3AKeyIdentifier%3E%3C%2Fo%3ASecurityTokenReference%3E%3C%2FKeyInfo%3E%3C
e%3ACipherData%3E%3Ce%3ACipherValue%3EEq9UhAJ8C9K5l4Mr3qmgX0XnyL1ChKs2PqMj0Sk
6snw%2FIRNtXqLzmgbj2Vd3vFA4Vx1hileSTyqc1kAsskqpqBc4bMHT61w1f0NxU10HDor0DlNVcV
Dm%2FAfLcyLqEP%2Boh05B%2B5ntVIJzL8Ro3typF0eoSm3S6UnINOHIjHaVWyg%3D%3C%2Fe%3AC
ipherValue%3E%3C%2Fe%3ACipherData%3E%3C%2Fe%3AEncryptedKey%3E%3C%2FKeyInfo%3E
%3Cenc%3ACipherData%3E%3Cenc%3ACipherValue%3ErBvpZydiyDzJtzl1%2FjUFX9XAzO1mOR
q0ypPLjh%2FBagXcfZeYwWD57v4Jvn1QwGajadcDASCisazswn1skdkwgmd4IUWJpPMRH7es9zY0U
vnS4ccsakgDcmscq3pDYTrxbSBfhdvrzjDiHC2XCtowOveoHeB51C5N8UAbff18IxCNtkWO8y3wLH
VGdvwaDOSakK%2FK%2Fv1UgXIc51%2FtYvjeFGeGbbSNxo8DTqeDnAMQ%2B4Y%2B1aUGhI%2FtbSr
EyJECkDgtztcxhrumbupKO%2BogWKUTTpSt851xjOFxAMiVaPZ%2FAm8V8H3ZLsR087sX%2FJ%2Bn
bRqze%2BfbdUwimN5pNoJDdMnF%2BEDLass1dPsvhL4EXzuIp5deGBaqAIoaOMEUW7ssuh1PtwkEM
eqwlOzOhu%2FHtwP1qh3D02U59MtyQnJMD5UwIwO7sZJl6%2BPg6Zp9HHtKKUMnkguvFmhyXS4BFS
ZVxPl18i%2B0MLO1um5dejEFd4nwGO%2FmNw6yEI8DdGVjXcYOT6JhPz9rHNh9%2F%2FOj5snJfL6
j2sg0EvIYoRs%2BhT4sdHZ95tGAiwMwT6cFOXbAQZUbYTr1ZOC6XPsfL2CFwiTM3mI%2Blco4Hc%2
F7IakIA8jwAJdtnd2mGuV67ZbY1mzibM1LUApixZj59El83ixctSQbV7iyywQ4IYN2CAq%2BCLMdl
R%2BDHfgEe8O3IVaGBDUEcd2MYimEiA7Yw3NIDrC14SbLzNvU702HpVJMeYv9q6S9xIVGApSrARsw
RFXyMbkMDp5WIQaJEXon7qLcsZONpdlX9bCcmaiikdpxmCeyS638te%2FhGBLmYJSQ0stf7BhA6E0
kwDRgdwsAa88bODiWHek0vDhAN4HlXFZ%2BCxp53L9Mmvy%2FCAOI%2B9OkPL2yxS22yjWQxom%2F
yZuawsK98JHVShsIVmmbKvRM6xJwvHDSzuBAOlQKS%2FMHcFZn8vHZR4lMhm5nL3F%2B%2BumMKh0
vMuKk6JiCqG9OEj996bVIIkLzESU5Z5vT6I1Kr9Brdx8ckDElipdH3x54WVfaItHJTYU%2BsxIR1T
25fi9k%2FOc%2FMX7Q%2B6NSDs4nGqkn4rzqpez9BUWNZw7caVOrDeao85f%2FiDCGymtl0A3JaSZ
dTKfzHLGmUfSkCAlVeisdvB6R7uBw8tR%2BZlgLIGS28wppFlnUYvSK7DnPrzId%2BGfHwLfL6WA%
2FEzBMMgppb5Vi%2BauHq%2BHxpCamlkrcUkzagbwNkGV8TfafkqUvRwJbxRwNVPI%2F%2Fxs%2Fp
Lcu1dh6eKcmU00%2FNx0zNOScd9XoeEU3zsV78PgvPIBT4EDugdv4bMR6dExXvZBl%2F84b1gOMhK
ZRplF8t6EAc4LCct01ht7VOVNz25NtP27ct9QPrDJc%2FoxihT4Df6NV3l4vlTnu%2B%2BzVB%2BH
JAxNkiO9gx3uLUJM9XEZCDzZKihaBk2y%2F3RhsJpABVneUd%2B3sCRbQXhgKYNBHZyRAUGpMDLhL
qpjoF9x%2FNvUujQ5DBLJafxxzNVshG52jRz%2BikhCNhJDDbeA5MQ8Q7QsYcKDC0DBFsewtWaA%2
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
FsKxl3JU6hyTotnFS%2FoS2EzbOSvn25qZuBERsZ3w%2B5WMkRzfQadyIYOSv2Df1YoljubDKy1l9
St%2FbCIBgXbVIZKYtQ%2BLyepxxFjrN7cWo2aYFnB6YLurg4USJwhXzcGcvA3%2BR5dRT6Fr37U6
OcHc%2Fz2MaZmn1cQWiDGNxHtRVxEvirBc1x47hWfSRjrKzf3orL5LzgMlYc7Iwclw2rbeWljCqOb
oV3d71ez%2FvNz1pxEMi4w8yUAQL8p%2FRCZ%2BpzvsgORu4RWKWiSwbl7AN0J3jiWShyZgDmxd2O
DDYffXjNiuH1mQWnDTkJX1ig88mqjhOYJEal0W6L0ErwrRIy29tOiAvXZANC8kA1HexulH0e38x8E
IOaVaJtNz9mqrnmnp4GdZ38txV%2BCUeWHOZaHLF4xkdtRxMAu%2FbzQ03YmUOhgxqkTfNzV6Ymne
v2nv5VsyQGJaQsNjb0M4yOe6kX2qNTwKBN2%2Bp%2Fz3f15i8KuGCgBcfP%2BP9xBizBeo7FbFtyo
2pfFhzBPmZeSOJ6kEbF1yQKHYQAT5iZ4SyTIfqqmwGxsQpWMstx3qJF8aW8WFzU1qXcC1LmgClg19
rx9NYFaQshX4f729B9Ue5MX7gTrMgwAnlXty9BsoP7nzGbr3HSXy8pR%2BimuAFW3c2NaQSbjSH5Z
FOr7PZdLHsNVJzFIsaufAwr0CAEtvlPJUt7%2B%2FE5MQsMsVqMoXFmefgdxbvY1Ue6MX1wtuJYY1
PAX7MHTyRUR3RfJDO054EoflVTwNE1fmocUXUh5rtFFuzy2T%2F2Y6pLAARXzo8uslAuH67VkuXv%
2BEMc7e3ogbf5%2BROsgJirZS6qkcYpfEUwqHiQYLnSIP4bt%2BWI5j1bxs7yzcSCkNZ2rd%2FHWr
A41AyGMfYzqxfGcrOaxHsds3JUcByB5Zw17W58GBC32Iusqa69BFTPagEapM0Fb5CbTqXnWTNNB5J
t40BVZvLv3u5oy%2BBRaMKXZhwnbT2WUTp0Ebsn17xvte52B%2BLMlSWJn96Nl5thd%2Ft1D7PlWA
sUvpJAd0UHPizCkY8VIhcXTrsSyEwer2J2I9TQTUosmssFjoP8Lx9qMfXo0eGVmneV8kVBtu4J7N1
QmWfV%2B%2FK8vGbCwW3Gm%2FEUlOO4ZbbK39y0JgNQ7fshxHr5Hdtd%2F6S%2FQkb6NPVDwn7Srh
Y0diWujXz5QlIYBSN7vDfMun3yF%2BGbmMExZ8MkOthuYkgMS9qiFoJGUXGyELsJfxbzdcRE9iyJn
p88L4%2BCtcO3l2JxIhMAgxOZx42RfAiDV1Gbpa4f%2F0urmWQ2VK7uZ%2FlViVrGAJ2kpH0EfwYE
Mb2YYT8FFjogqEpDSJX48BLIh1TE4nMbqQVG1cksCGDc0XyGKaF5Z7Ikw493Xz0JQ0BZvaf2Kceb7
MUZlsU1DSHcQQ9X%2Bxu9RcgUePJEe9BgCMpZ5Kr6r43qyk79noBSgrsSkDhT5sg%2Fc20RHQB8OX
%2BC4r3XGQFWF2m2j0xTc%2Boy14xqUmSB2qJtuWGOXDJspejDRP1GIfFnqDFdqSO3%2FkV9AC5Ee
39iJGv8I%2B5nErtQao645bCytn4B2bJah8R2fXLs8Dd4%2BC2ykxVrLxTUmJaGqd2RK%2F6t1E47
l%2B90Vp4WEzC0CFXXt9XNqdVjo2bZsXbfKQgO2zT2q2qCsgwbxVzIF5y39R%2BrkSkX16uuz3q6w
n3I5RI9M8Hn3DCzzv6Ms4rYxYuiqxaIcb7DgjI2fk1bdyiiRjSxzpCHpK6CWjBD8DPQYdkqGr%2Bs
oWeSvHvPLMSDxEPzwlnaxysRXzKphHUeUa2CCqcpagux2mbKkwHSXemX9I3V3AhPePp5XI5eCRiy3
D4%2BcBXOydie94Nz9DIhW749hPiVD9CioAgyqgAzFwCxEEUCXKTzu9xXX4DXg9b3CUfGzwERtY7x
TGT2y%2F9i7r5Xs0lrKi9ftws4JO5v%2Be3WuAEtWv0w%2FVKCl1WwTbV9xtx%2B4RZQ3%2Fewvv%
2F0GqiiSrhiVBGuCDaQs7stwqfkF3vFgGXmmODGTIkIxvYm2fzcEfq4A6LRp5RkYyJyUTF87c56tn
Qa%2Bo3xeiX5WRJybpabrRou09vyWLdlkhcUaBElGWB7iYUJ9bCltByEdNZnuDV%2FXlfnmDARKp8
RVN028czIk57wQMuizgWrM6S9Ku20noDmLgbT554UBf7FnjRWOb%2FF9OJuPpUcARBPrfuqTcOsBq
tZr7AJl3zz%2F53mpyn9rgzw5gBLgkvrdbciabJOAacccTDEB5kEzCLuprC3SlVedhgY%2BMQ5%2F
xgN%2Faf3TtJiBKFvb1V37BlbXXGosnPFcoH8I0XbqW5FSsxmcnpg48poJcB7j5eHq7Y%2F01RLb4
iMmzNap4%2BFg2F3LrwOI0Wk7ueIjgFd5KJ1iTda1ivGU%2Fchr9aTNpM5HiLb2fDW0pZ%2FFBJcI
XxpT9eNY%2FpVj5pnTW2ubpPnBulPOQTLCi1EOxbl33wnhUIfnGiVWJdrls2j3GWgqOnrYUbP%2FX
tNJqIucnMYGqPbcGIF2QRuiwD%2FiTRMvCRCmdCsYE%2FaXjOMhskX7KYC%2B9iG%2FT1wQRbfHSK
WD%2Fpv450OVDsfc1Adq6FCr1LesDNTew%2FF8Z3SiHnWS76OVsNM2SB%2FhMP67iu5UWVkb3%2FQ
qCN0aosOPs2QX0XBCZFmN6p3FhFnXPbAbaGz9y6KzUiUxC03U0fZcToKl4y%2Bw0P4IvxpjVt4t8b
84Q9hiBxd5xu1%2BRE973a%2FyIWO%2Fit1MdUSmxWakxWuGxDnQxwkNCN7ekL%2FQ%2B6FItm86b
w9cc%2FMiI7q2fK7y7YAzM3tmamhF1%2FWJNj1lH0vh%2BhNehJlLlb4Z%2F9ZtxMWV4LVTyrFaF1
zyCEqcKUTk0jc%2FXDwyKZc%2FSV9EOoPk2fVnmzs3WkA74GB%2BWtjdvQjSmnJYtPkMNsikHw%2B
RyB1hTkYbn3iQ6BUiJ0v97j7MVZHxCa1KS3t2gx8H7ts6Tfy5il89xVUdiZwfj0w06g199qlAqUMZ
EWxh0%3D%3C%2Fenc%3ACipherValue%3E%3C%2Fenc%3ACipherData%3E%3C%2Fenc%3AEncryp
tedData%3E
```

An un-escaped and reformatted version of the preceding xmlToken value, with the encrypted value elided, is as follows:

```
<enc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element" xmlns:enc=
"http://www.w3.org/2001/04/xmlenc#">
<enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"
/>
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<e:EncryptedKey xmlns:e="http://www.w3.org/2001/04/xmlenc#">
<e:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1
p">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
</e:EncryptionMethod>
<KeyInfo>
<o:SecurityTokenReference xmlns:o="http://docs.oasis-open.org/wss/2004/01/oas
is-200401-wss-wssecurity-secext-1.0.xsd">
<o:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-mes
sage-security-1.1#ThumbprintSHA1" EncodingType="http://docs.oasis-open.org/ws
s/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary">
+PYbznDaB/dlhjIfqCQ458E72wA=
</o:KeyIdentifier>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
</o:SecurityTokenReference>
</KeyInfo>
<e:CipherData>
<e:CipherValue>
Eq9UhAJ8C9K5l4Mr3qmgX0XnyL1ChKs2PqMj0Sk6snw/IRNtXqLzmgbj2Vd3vFA4Vx1hileSTyqc1
kAsskqpqBc4bMHT61w1f0NxU10HDor0DlNVcVDm/AfLcyLqEP+oh05B+5ntVIJzL8Ro3typF0eoSm
3S6UnINOHIjHaVWyg=
</e:CipherValue>
</e:CipherData>
</e:EncryptedKey>
</KeyInfo>
<enc:CipherData>
<enc:CipherValue>
...=
</enc:CipherValue>
</enc:CipherData>
</enc:EncryptedData>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

B. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Original Authors of the initial contributions:

Arun Nanda, Microsoft Corporation
Michael B. Jones, Microsoft Corporation
Jan Alexander, Microsoft
Giovanni Della-Libera, Microsoft
Martin Gudgin, Microsoft
Kirill Gavrylyuk, Microsoft
Tomasz Janczuk, Microsoft
Michael McIntosh, IBM
Anthony Nadalin, IBM
Bruce Rich, IBM
Doug Walter, Microsoft

Participants:

John Bradley, Individual
Norman Brickman, Mitre Corporation
Jeffrey Broberg, CA
Scott Cantor Internet2
Ruchith Fernando, WSO2
Marc Goodner, Microsoft Corporation (Chair)
Patrick Harding, Ping Identity
Andrew Hodgkinson, Novell
Mario Ivkovic, A-SIT, Zentrum für Sichere Informationstechnologie - Austria
Michael B. Jones, Microsoft Corporation (Editor)
Mike Kirkwood, Polka Networks
Herbert Leitold, A-SIT, Zentrum für Sichere Informationstechnologie - Austria
Michael McIntosh, IBM (Editor)
Dale Olds, Novell
Anthony Nadalin, IBM (Chair)
Drummond Reed, Cordance
Bruce Rich ,IBM
Darran Rolls, SailPoint Technologies
Prabath Siriwardena, WSO2

Copyright © OASIS® 2008-2009. All Rights Reserved.

Appendix
OASIS02

https://web.archive.org/web/20100401000000*/http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.pdf February 27, 2026

INTERNET ARCHIVE | WEB | TEXTS | VIDEO | AUDIO | SOFTWARE | IMAGES | SIGN UP | LOG IN | UPLOAD | Search

ABOUT | BLOG | EVENTS | PROJECTS | HELP | DONATE | CONTACT | JOBS | VOLUNTEER

DONATE

INTERNET ARCHIVE WayBackMachine

Explore more than 1 trillion web pages saved over time

http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-o

Calendar · Collections · Changes · Summary · Site Map · URLs

Saved **34 times** between July 7, 2010 and September 6, 2025.



JAN

					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

FEB

	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28						

MAR

	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

APR

				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

MAY

						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

JUN

		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

JUL

				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

AUG

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

SEP

		1	2	3	4	
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

OCT

					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

NOV

	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

DEC

			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

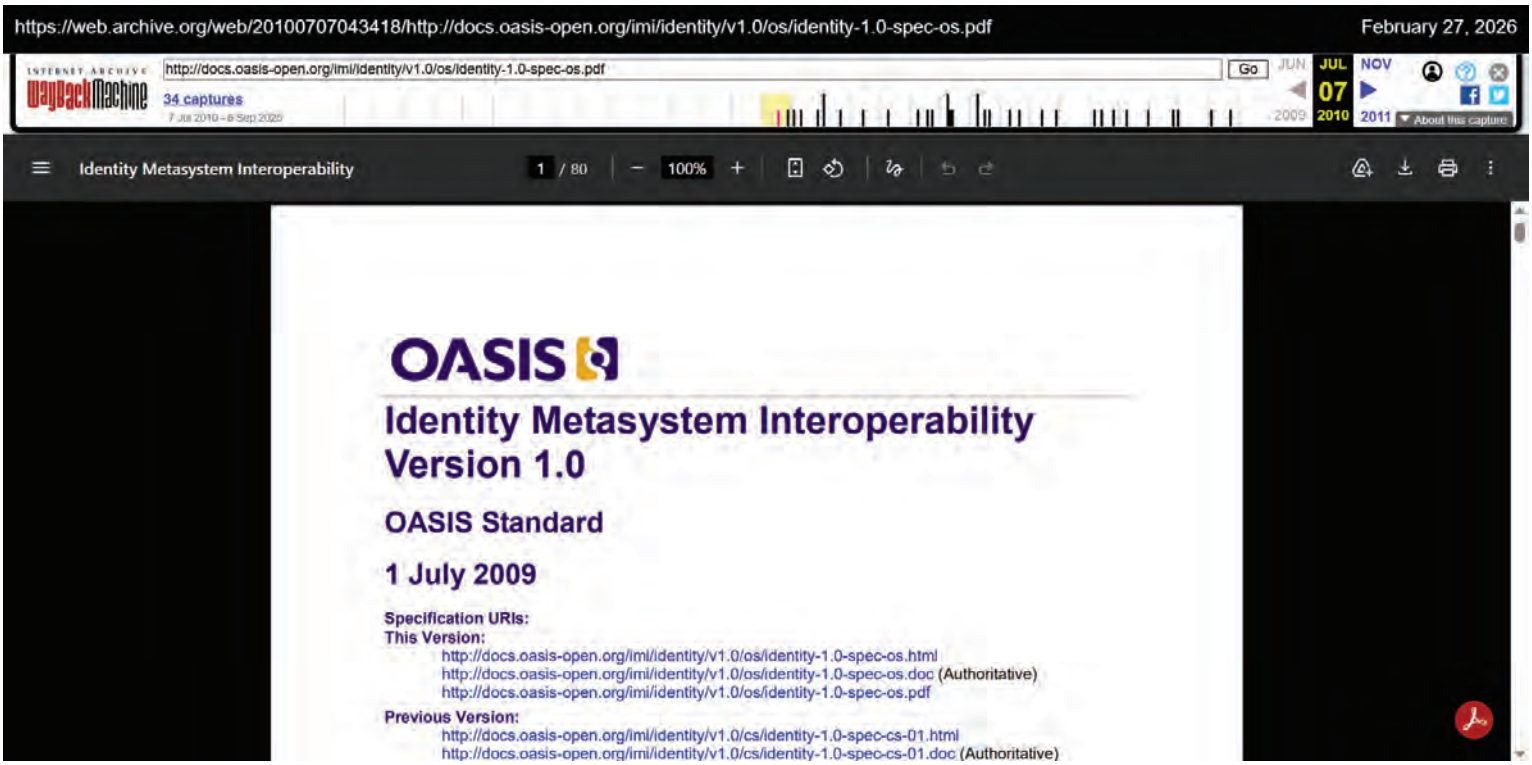
Note

This calendar view maps the number of times **http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.pdf** was crawled by the Wayback Machine, *not* how many times the site was actually updated. More info in the FAQ.

FAQ | Contact Us | Terms of Service (Dec 31, 2014)

The Wayback Machine is an initiative of the Internet Archive, a 501(c)(3) non-profit, building a digital library of Internet sites and other cultural artifacts in digital form. Other projects include Open Library & archive-it.org.

Your use of the Wayback Machine is subject to the Internet Archive's Terms of Use.


https://web.archive.org/web/20100707043418/http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.pdf
February 27, 2026
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.pdf
Go
34 captures
JUL 07 2010
NOV 2011
Identity Metasystem Interoperability
1 / 80
100%
OASIS
Identity Metasystem Interoperability Version 1.0
OASIS Standard
1 July 2009
Specification URIs:
This Version:
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.html
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.doc (Authoritative)
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.pdf
Previous Version:
http://docs.oasis-open.org/imi/identity/v1.0/cs/identity-1.0-spec-cs-01.html
http://docs.oasis-open.org/imi/identity/v1.0/cs/identity-1.0-spec-cs-01.doc (Authoritative)

Identity Metasystem Interoperability Version 1.0

OASIS Standard

1 July 2009

Specification URIs:
This Version:
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.html
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.doc (Authoritative)
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.pdf

Previous Version:
http://docs.oasis-open.org/imi/identity/v1.0/cs/identity-1.0-spec-cs-01.html
http://docs.oasis-open.org/imi/identity/v1.0/cs/identity-1.0-spec-cs-01.doc (Authoritative)
http://docs.oasis-open.org/imi/identity/v1.0/cs/identity-1.0-spec-cs-01.pdf

Latest Version:
http://docs.oasis-open.org/imi/identity/v1.0/identity.html
http://docs.oasis-open.org/imi/identity/v1.0/identity.doc
http://docs.oasis-open.org/imi/identity/v1.0/identity.pdf

Technical Committee:
OASIS Identity Metasystem Interoperability (IMI) TC

Chair(s):
Marc Goodner
Anthony Nadalin

Editor(s):
Michael B. Jones
Michael McIntosh

Related work:
This specification replaces or supersedes:

- None

This specification is related to:

- WS-Trust
- WS-SecurityPolicy
- WS-Addressing

Declared XML Namespace(s):
http://docs.oasis-open.org/imi/ns/identity-200810
http://schemas.xmlsoap.org/ws/2005/05/identity
http://schemas.xmlsoap.org/ws/2006/02/addressingidentity
http://schemas.xmlsoap.org/ws/2007/01/identity

Abstract:
This document is intended for developers and architects who wish to design identity systems and applications that interoperate using the Identity Metasystem Interoperability specification.

Copyright © OASIS® 2008-2009. All Rights Reserved.

An Identity Selector and the associated identity system components allow users to manage their Digital Identities from different Identity Providers, and employ them in various contexts to access online services. In this specification, identities are represented to users as “Information Cards”. Information Cards can be used both at applications hosted on Web sites accessed through Web browsers and rich client applications directly employing Web services.

This specification also provides a related mechanism to describe security-verifiable identity for endpoints by leveraging extensibility of the WS-Addressing specification. This is achieved via XML [XML 1.0] elements for identity provided as part of WS-Addressing Endpoint References. This mechanism enables messaging systems to support multiple trust models across networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner.

Status:

This document was last revised or approved by the Identity Metasystem Interoperability TC on the above date. The level of approval is also listed above. Check the “Latest Version” or “Latest Approved Version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the “Send A Comment” button on the Technical Committee’s web page at http://www.oasis-open.org/committees/imi/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/imi/ipr.php.

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/imi/.

Copyright © OASIS® 2008-2009. All Rights Reserved.

Notices

Copyright © OASIS® 2008-2009. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", [insert specific trademarked names and abbreviations here] are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

Table of Contents

1 Introduction 7
1.1 Notational Conventions 7
1.2 Namespaces 7
1.3 Schema 9
1.4 Terminology 9
1.5 Normative References 10
1.6 Non-Normative References 12
2 Relying Party Interactions 13
2.1 Expressing Token Requirements of Relying Party 13
2.1.1 Issuer of Tokens 13
2.1.2 Type of Proof Key in Issued Tokens 14
2.1.3 Claims in Issued Tokens 14
2.2 Expressing Privacy Policy of Relying Party 15
2.3 Employing Relying Party STSs 17
3 Identity Provider Interactions 17
3.1 Information Card 17
3.1.1 Information Card Format 17
3.1.2 Issuing Information Cards 25
3.2 Identity Provider Policy 27
3.2.1 Require Information Card Provisioning 27
3.2.2 Policy Metadata Location 27
3.3 Token Request and Response 27
3.3.1 Information Card Reference 28
3.3.2 Claims and Other Token Parameters 28
3.3.3 Token Scope 28
3.3.4 Client Pseudonym 29
3.3.5 Proof Key for Issued Token 30
3.3.6 Display Token 35
3.3.7 Token References 36
4 Authenticating to Identity Provider 36
4.1 Username and Password Credential 37
4.2 Kerberos v5 Credential 37
4.3 X.509v3 Certificate Credential 37
4.4 Self-issued Token Credential 38
5 Faults 38
5.1 Relying Party 39
5.2 Identity Provider 39
5.2.1 Identity Provider Custom Error Messages 40
6 Information Cards Transfer Format 41
6.1 Pre-Encryption Transfer Format 41
6.1.1 PIN Protected Card 43
6.1.2 Computing the ic:IssuerId 44
6.1.3 Computing the ic:IssuerName 45

Copyright © OASIS® 2008-2009. All Rights Reserved.

6.1.4 Creating the ic:HashSalt 45
6.2 Post-Encryption Transfer Format 45
7 Simple Identity Provider Profile 47
7.1 Self-Issued Information Card 47
7.2 Self-Issued Token Characteristics 47
7.3 Self-Issued Token Encryption 51
7.4 Self-Issued Token Signing Key 52
7.4.1 Processing Rules 53
7.5 Claim Types 55
7.5.1 First Name 55
7.5.2 Last Name 55
7.5.3 Email Address 55
7.5.4 Street Address 55
7.5.5 Locality Name or City 55
7.5.6 State or Province 56
7.5.7 Postal Code 56
7.5.8 Country 56
7.5.9 Primary or Home Telephone Number 56
7.5.10 Secondary or Work Telephone Number 56
7.5.11 Mobile Telephone Number 56
7.5.12 Date of Birth 57
7.5.13 Gender 57
7.5.14 Private Personal Identifier 57
7.5.15 Web Page 57
7.6 The PPID Claim 57
7.6.1 Relying Party Identifier and Relying Party PPID Seed 58
7.6.2 PPID 60
7.6.3 Friendly Identifier 60
8 Relying Parties without Certificates 61
8.1 Relying Party Identifier and Relying Party PPID Seed 61
8.2 AppliesTo Information 61
8.3 Token Signing and Encryption 62
9 Using WS-SecurityPolicy 1.2 and WS-Trust 1.3 62
9.1 Overview of Differences 62
9.2 Identity Selector Differences 62
9.3 Security Token Service Differences 63
10 Browser Behavior with Information Cards 64
10.1 Basic Protocol Flow when using an Information Card at a Web Site 64
10.2 Protocol Flow with Relying Party STS 65
10.3 User Perspective and Examples 66
10.4 Browser Perspective 67
10.5 Web Site Perspective 67
11 Invoking an Identity Selector from a Web Page 68
11.1 Syntax Alternatives: OBJECT and XHTML tags 68
11.1.1 OBJECT Syntax Examples 68

Copyright © OASIS® 2008-2009. All Rights Reserved.

11.1.2 XHTML Syntax Example 69
11.2 Identity Selector Invocation Parameters 70
11.2.1 issuer 70
11.2.2 issuerPolicy 70
11.2.3 tokenType 70
11.2.4 requiredClaims 70
11.2.5 optionalClaims 70
11.2.6 privacyUrl 70
11.2.7 privacyVersion 70
11.3 Data Types for Use with Scripting 70
11.4 Detecting and Utilizing an Information Card-enabled Browser 71
11.5 Behavior within Frames 71
11.6 Invocation Using the Document Object Model (DOM) 71
11.7 Auditing, Non-Auditing, and Auditing-Optional Cards 71
12 Endpoint Reference wsai:Identity Property 72
12.1 Default Value 72
12.2 Identity Representation 72
12.2.1 DNS Name 72
12.2.2 Service Principal Name 72
12.2.3 User Principal Name 72
12.2.4 KeyInfo 73
12.2.5 Security Token 73
12.2.6 Security Token Reference 74
13 Security Considerations 75
13.1 Protection of Information Cards by Identity Selectors 75
13.2 Relying Parties Without Certificates 75
13.3 Endpoint References 75
14 Conformance 76
A. HTTPS POST Sample Contents 77
B. Acknowledgements 80

Copyright © OASIS® 2008-2009. All Rights Reserved.

1 Introduction

The Identity Metasystem Interoperability specification prescribes a subset of the mechanisms defined in [WS-Trust 1.2], [WS-Trust 1.3], [WS-SecurityPolicy 1.1], [WS-SecurityPolicy 1.2], and [WS-MetadataExchange] to facilitate the integration of Digital Identity into an interoperable token issuance and consumption framework using the Information Card Model. It documents the Web interfaces utilized by browsers and Web applications that utilize the Information Card Model. Finally, it extends WS-Addressing's endpoint reference by providing identity information about the endpoint that can be verified through a variety of security means, such as https or the wealth of WS-Security specifications.

This profile constrains the schema elements/extensions used by the Information Card Model, and behaviors for conforming Relying Parties, Identity Providers, and Identity Selectors.

1.1 Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119].

This specification uses the following syntax to define outlines for assertions:

- The syntax appears as an XML instance, but values in italics indicate data types instead of literal values.
- Characters are appended to elements and attributes to indicate cardinality:
    - "?" (0 or 1)
    - "*" (0 or more)
    - "+" (1 or more)
- The character "|" is used to indicate a choice between alternatives.
- The characters "(" and ")" are used to indicate that contained items are to be treated as a group with respect to cardinality or choice.
- The characters "[" and "]" are used to call out references and property names.
- Ellipses (i.e., "...") indicate points of extensibility. Additional children and/or attributes MAY be added at the indicated extension points but MUST NOT contradict the semantics of the parent and/or owner, respectively. By default, if a receiver does not recognize an extension, the receiver SHOULD ignore the extension; exceptions to this processing rule, if any, are clearly indicated below.
- XML namespace prefixes (see Table 2) are used to indicate the namespace of the element being defined.

Elements and Attributes defined by this specification are referred to in the text of this document using XPath 1.0 expressions. Extensibility points are referred to using an extended version of this syntax:

- An element extensibility point is referred to using {any} in place of the element name. This indicates that any element name can be used, from any namespace other than the namespace of this specification.
- An attribute extensibility point is referred to using @{any} in place of the attribute name. This indicates that any attribute name can be used, from any namespace other than the namespace of this specification.

Extensibility points in the exemplar might not be described in the corresponding text.

1.2 Namespaces

Table 1 lists the XML namespaces that are used in this document.

Copyright © OASIS® 2008-2009. All Rights Reserved.

Prefix	XML Namespace	Specification(s)
ds	http://www.w3.org/2000/09/xmldsig#	XML Digital Signatures
ic	http://schemas.xmlsoap.org/ws/2005/05/identity	This document
ic07	http://schemas.xmlsoap.org/ws/2007/01/identity	Namespace for additional elements also defined by this document
ic08	http://docs.oasis-open.org/imi/ns/identity-200810	Namespace for new elements defined by this document
S	*May refer to either* http://schemas.xmlsoap.org/soap/envelope *or* http://www.w3.org/2003/05/soap-envelope *since both may be used*	SOAP
S11	http://schemas.xmlsoap.org/soap/envelope	SOAP 1.1 [SOAP 1.1]
S12	http://www.w3.org/2003/05/soap-envelope	SOAP 1.2 [SOAP 1.2]
saml	urn:oasis:names:tc:SAML:1.0:assertion	SAML 1.0
sp	*May refer to either* http://schemas.xmlsoap.org/ws/2005/07/securitypolicy *or* http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702 *since both may be used*	WS-SecurityPolicy
sp11	http://schemas.xmlsoap.org/ws/2005/07/securitypolicy	WS-SecurityPolicy 1.1 [WS-SecurityPolicy 1.1]
sp12	http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702	WS-SecurityPolicy 1.2 [WS-SecurityPolicy 1.2]
wsa	http://www.w3.org/2005/08/addressing	WS-Addressing [WS-Addressing]
wsai	http://schemas.xmlsoap.org/ws/2006/02/addressingidentity	Addressing Identity extension for WS-Addressing also defined by this document
wsdl	*May refer to either* http://schemas.xmlsoap.org/wsdl/ *or* http://www.w3.org/TR/wsdl20 *since both may be used*	Web Services Description Language
wsdl11	http://schemas.xmlsoap.org/wsdl/	Web Services Description Language [WSDL 1.1]
wsdl20	http://www.w3.org/TR/wsdl20	Web Services Description Language [WSDL 2.0]
wsp	http://schemas.xmlsoap.org/ws/2004/09/policy	WS-Policy [WS-Policy]
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	WS-Security Extensions [WS-Security]

Copyright © OASIS® 2008-2009. All Rights Reserved.

wst	*May refer to either* http://schemas.xmlsoap.org/ws/2005/02/trust *or* http://docs.oasis-open.org/ws-sx/ws-trust/200512 *since both may be used*	WS-Trust
wst12	http://schemas.xmlsoap.org/ws/2005/02/trust	WS-Trust 1.2 [WS-Trust 1.2]
wst13	http://docs.oasis-open.org/ws-sx/ws-trust/200512	WS-Trust 1.3 [WS-Trust 1.3]
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	WS-SecurityUtility
wsx	http://schemas.xmlsoap.org/ws/2004/09/mex	WS-MetadataExchange [WS-MetadataExchange]
xs	http://www.w3.org/2001/XMLSchema	XML Schema [Part 1, 2]

Note that the versions identified in the above table supersede versions identified in referenced specifications.

1.3 Schema

A copy of the XML Schemas for this document can be found at:

```
http://docs.oasis-open.org/imi/identity/200810/identity.xsd
http://docs.oasis-open.org/imi/identity/200810/addr-identity.xsd
http://docs.oasis-open.org/imi/identity/200810/claims.xsd
http://docs.oasis-open.org/imi/identity/200810/identity2007.xsd
```

1.4 Terminology

The following definitions establish the terminology and usage in this document.

Information Card Model – The "*Information Card Model*" refers to the use of Information Cards containing metadata for obtaining Digital Identity claims from Identity Providers and then conveying them to Relying Parties under user control.

Information Card – An *Information Card* provides a visual representation of a Digital Identity for the end user. Information Cards contain a reference to an IP/STS that issues Security Tokens containing the Claims for that Digital Identity.

Digital Identity – A "*Digital Identity*" is a set of Claims made by one party about another party.

Claim – A "*Claim*" is a piece of information about a Subject that an Identity Provider asserts about that Subject.

Subject – A "*Subject*" is an individual or entity about whom claims are made by an Identity Provider.

Service Requester – The term "*Service Requester*" means software acting on behalf of a party who wants to obtain a service through a digital network.

Relying Party – The term "*Relying Party*" (RP) means a network entity providing the desired service, and relying upon Digital Identity.

Identity Provider – The term "*Identity Provider*" (IP) means a network entity providing the Digital Identity claims used by a Relying Party.

Security Token Service – The term "*Security Token Service*" (STS) refers to a WS-Trust endpoint.

Identity Provider Security Token Service – The term "Identity Provider Security Token Service" (IP/STS) refers to the Security Token Service run by an Identity Provider to issue tokens.

Copyright © OASIS® 2008-2009. All Rights Reserved.

Relying Party Security Token Service – The term “Relying Party Security Token Service” (RP/STS) refers to a Security Token Service run by a Relying Party to accept and issue tokens.

Identity Selector – The term “*Identity Selector*” (IS) refers to a software component available to the Service Requester through which the user controls and dispatches her Digital Identities.

Trust Identity – A *trust identity* is a verifiable claim about a principal (e.g. name, identity, key, group, privilege, capability, etc).

Security Token – A *security token* represents a collection of claims.

Signed Security Token – A *signed security token* is a security token that is asserted and cryptographically endorsed by a specific authority (e.g. an X.509 certificate, a Kerberos ticket, or a self-issued Information Card).

Unsigned Security Token – An un*signed security token* is a security token that is not cryptographically endorsed by a specific authority (e.g. a security token backed by shared secrets such as usernames and passwords).

Proof-of-Possession – The *proof-of-possession* information is data that is used in a proof process to demonstrate the sender's knowledge of information that should only be known to the claiming sender of a security token.

Integrity – *Integrity* is the process by which it is guaranteed that information is not modified in transit.

Confidentiality – *Confidentiality* is the process by which data is protected such that only authorized actors or security token owners can view the data

Digest – A *digest* is a cryptographic checksum of an octet stream.

Signature - A *signature* is a cryptographic binding of a proof-of-possession and a digest. This covers both symmetric key-based and public key-based signatures. Consequently, non-repudiation is not always achieved.

Certificate – Uses of the term *certificate* in this specification refer to X.509 certificates unless otherwise qualified. Usage of certificates is dictated by the underlying protocols, e.g. HTTPS or WS-Security, except where noted.

1.5 Normative References

[DOM]

“Document Object Model (DOM)”, November 2000. http://www.w3.org/DOM/

[EV Cert]

CA / Browser Forum, “Guidelines for the Issuance and Management of Extended Validation Certificates, Version 1.1”, April 2008. http://cabforum.org/EV_Certificate_Guidelines_V11.pdf

[HTTP]

R. Fielding et al., “IETF RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1”, June 1999. http://www.ietf.org/rfc/rfc2616.txt

[HTTPS]

E. Rescorla, “RFC 2818: HTTP over TLS”, May 2000. http://www.ietf.org/rfc/rfc2818.txt

[RFC 1274]

P. Barker and S. Kille, “RFC 1274: The COSINE and Internet X.500 Schema”, November 1991. http://www.ietf.org/rfc/rfc1274.txt

[RFC 2119]

S. Bradner, “RFC 2119: Key words for use in RFCs to Indicate Requirement Levels”, March 1997. http://www.ietf.org/rfc/rfc2119.txt

Copyright © OASIS® 2008-2009. All Rights Reserved.

[RFC 2256]

M. Wahl, “RFC 2256: A Summary of the X.500(96) User Schema for use with LDAPv3”, December 1997. http://www.ietf.org/rfc/rfc2256.txt

[RFC 2459]

R. Housley, W. Ford, W. Polk, and D. Solo, “RFC 2459: Internet X.509 Public Key Infrastructure - Certificate and CRL Profile”, January 1999. http://www.ietf.org/rfc/rfc2459.txt

[RFC 2898]

B. Kaliski, “PKCS #5: Password-Based Cryptography Specification, Version 2.0”, September 2000. http://www.ietf.org/rfc/rfc2898.txt

[RFC 3066]

H. Alvestrand, “Tags for the Identification of Languages”, January 2001. http://www.faqs.org/rfcs/rfc3066.html

[SOAP 1.1]

W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[SOAP 1.2]

M. Gudgin, et al., “SOAP Version 1.2 Part 1: Messaging Framework”, June 2003. http://www.w3.org/TR/soap12-part1/

[URI]

T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998. http://www.ietf.org/rfc/rfc2396.txt

[WS-Addressing]

W3C Recommendation, “Web Service Addressing (WS-Addressing)”, 9 May 2006. http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/

[WS-MetadataExchange]

“Web Services Metadata Exchange (WS-MetadataExchange), Version 1.1”, August 2006. http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf

[WSDL 1.1]

W3C Note, "Web Services Description Language (WSDL 1.1)," 15 March 2001. http://www.w3.org/TR/wsdl

[WSDL 2.0]

“Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language”, June 2007. http://www.w3.org/TR/wsdl20

[WS-Policy]

“Web Services Policy Framework (WS-Policy), Version 1.2”, March 2006. http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf

[WS-Security]

A. Nadalin et al., “Web Services Security: SOAP Message Security 1.0”, May 2004. http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[WS-SecurityPolicy 1.1]

“Web Services Security Policy Language (WS-SecurityPolicy), Version 1.1”, July 2005. http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf

Copyright © OASIS® 2008-2009. All Rights Reserved.

[WS-SecurityPolicy 1.2]

OASIS, "WS-SecurityPolicy 1.2", July 2007. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.pdf

[WS-Trust 1.2]

"Web Services Trust Language (WS-Trust)", February 2005. http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf

[WS-Trust 1.3]

OASIS, "WS-Trust 1.3", March 2007. http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf

[XML 1.0]

W3C Recommendation, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", September 2006. http://www.w3.org/TR/xml/

[XMLDSIG]

Eastlake III, D., Reagle, J., and Solo, D., "XML-Signature Syntax and Processing", March 2002. http://www.ietf.org/rfc/rfc3275.txt

[XMLENC]

Imamura, T., Dillaway, B., and Simon, E., "XML Encryption Syntax and Processing", August 2002. http://www.w3.org/TR/xmlenc-core/

[XML Schema, Part 1]

H. Thompson et al., "XML Schema Part 1: Structures", May 2001. http://www.w3.org/TR/xmlschema-1/

[XML Schema, Part 2]

P. Biron et al., "XML Schema Part 2: Datatypes", May 2001. http://www.w3.org/TR/xmlschema-2/

1.6 Non-Normative References

[Addressing-Ext]

J. Alexander et al., "Application Note: Web Services Addressing Endpoint References and Identity", July 2008. http://schemas.xmlsoap.org/ws/2006/02/addressingidentity

[ISIP]

A. Nanda and M. Jones, "Identity Selector Interoperability Profile V1.5", July 2008. http://www.microsoft.com/downloads/details.aspx?FamilyID=b94817fc-3991-4dd0-8e85-b73e626f6764&DisplayLang=en

[ISIP Guide]

Microsoft Corporation and Ping Identity Corporation, "An Implementer's Guide to the Identity Selector Interoperability Profile V1.5", July 2008. http://www.microsoft.com/downloads/details.aspx?FamilyID=b94817fc-3991-4dd0-8e85-b73e626f6764&DisplayLang=en

[ISIP Web Guide]

M. Jones, "A Guide to Using the Identity Selector Interoperability Profile V1.5 within Web Applications and Browsers", July 2008. http://www.microsoft.com/downloads/details.aspx?FamilyID=b94817fc-3991-4dd0-8e85-b73e626f6764&DisplayLang=en

Copyright © OASIS® 2008-2009. All Rights Reserved.

2 Relying Party Interactions

This section defines the constructs used by a Relying Party Web service for specifying and conveying its Security Token requirements to the Service Requester.

2.1 Expressing Token Requirements of Relying Party

A Relying Party specifies its Security Token requirements as part of its Security Policy using the primitives and assertions defined in WS-SecurityPolicy. The primary construct in the Security Policy of the Relying Party used to specify its requirement for a Security Token from an Identity Provider is the `sp:IssuedToken` policy assertion. The basic form of the issued token policy assertion as defined in WS-SecurityPolicy is as follows.

```
<sp:IssuedToken sp:Usage="xs:anyURI" sp:IncludeToken="xs:anyURI"
    xmlns:sp="..." xmlns:wsa="..." xmlns:wsp="..." ...>
  <sp:Issuer>
    wsa:EndpointReference | xs:any
  </sp:Issuer>
  <sp:RequestSecurityTokenTemplate>
    ...
  </sp:RequestSecurityTokenTemplate>
  <wsp:Policy>
    ...
  </wsp:Policy>
  ...
</sp:IssuedToken>
```

The attributes and elements listed in the schema fragment above are described in WS-SecurityPolicy.

The ensuing subsections describe special parameters added by this profile as extensions to the `sp:IssuedToken` policy assertion that convey additional instructions to the Identity Selector available to the Service Requester.

2.1.1 Issuer of Tokens

The `sp:IssuedToken/sp:Issuer` element in an issued token policy specifies the issuer for the requested token. More specifically, it SHOULD contain the endpoint reference of an Identity Provider STS that can issue the requested token.

A Relying Party MUST specify the issuer for a requested token in one of the following ways:

- Indicate a *specific* issuer by specifying the issuer's endpoint as the value of the `sp:Issuer/wsa:Address` element.
- Indicate that the issuer is *unspecified* by omitting the `sp:Issuer` element, which means that the Service Requester should determine the appropriate issuer for the requested token with help from the user if necessary.

When requiring a specific issuer, a Relying Party MAY specify that it will accept self-issued Security Tokens by using the special URI below as the value of the `wsa:Address` element within the endpoint reference for the issuer.

URI:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self
```

Following is an example of using this URI within an issued token policy.

Example:

```
<sp:IssuedToken xmlns:sp="..." xmlns:wsa="..." ...>
  <sp:Issuer>
    <wsa:Address>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
      http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self
    </wsa:Address>
  </sp:Issuer>
  ...
</sp:IssuedToken>
```

A Relying Party MAY specify the value of the `sp:Issuer/wsa:Address` element in policy as a "logical name" of the token issuer instead of an actual network address where the token is issued. An Identity Selector SHOULD resolve the logical name to an appropriate endpoint for the token issuer by matching the issuer name in Information Cards available to it.

If a Relying Party specifies the token issuer as a network endpoint in policy, then it MUST also specify the location of issuer metadata from where the issuer's policy metadata can be obtained. This is done using the mechanism defined in [WS-Addressing] for embedding metadata within an endpoint reference. The following example shows a token policy where the issuer endpoint and its corresponding metadata location are specified.

Example:

```
<sp:IssuedToken xmlns:sp="..." xmlns:wsa="..." xmlns:wsx="..." ...>
  <sp:Issuer>
    <wsa:Address>http://contoso.com/sts</wsa:Address>
    <wsa:Metadata>
      <wsx:Metadata>
        <wsx:MetadataSection
            Dialect="http://schemas.xmlsoap.org/ws/2004/09/mex">
          <wsx:MetadataReference>
            <wsa:Address>https://contoso.com/sts/mex</wsa:Address>
          </wsx:MetadataReference>
        </wsx:MetadataSection>
      </wsx:Metadata>
    </wsa:Metadata>
  </sp:Issuer>
  ...
</sp:IssuedToken>
```

2.1.2 Type of Proof Key in Issued Tokens

If no explicit key type is specified by the Relying Party, then an Identity Selector SHOULD request an asymmetric key token from the Identity Provider to maximize user privacy and security.

A Relying Party MAY explicitly request the use of an *asymmetric* or *symmetric* key in the requested token by using the `wst:KeyType` element within its issued token policy assertion. The key type URIs are defined in [WS-Trust]. The following example illustrates the use of this element in the Relying Party's Security Policy to request a symmetric key in the issued token.

Example:

```
<sp:IssuedToken xmlns:sp="..." xmlns:wst="...">
  <sp:RequestSecurityTokenTemplate>
    <wst:KeyType>
      http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey
    </wst:KeyType>
  </sp:RequestSecurityTokenTemplate>
</sp:IssuedToken>
```

2.1.3 Claims in Issued Tokens

The claims requirement of a Relying Party can be expressed in its token policy by using the optional `wst:Claims` parameter defined in [WS-Trust 1.2] and [WS-Trust 1.3]. However, the `wst:Claims` parameter has an open content model. This profile defines the `ic:ClaimType` element for use as a child of the `wst:Claims` element. A Relying Party MAY use this element to specify an individual claim type

Copyright © OASIS® 2008-2009. All Rights Reserved.

297 requested. Further, each requested claim MAY be specified as being *required* or *optional*. Multiple
298 `ic:ClaimType` elements can be included to specify multiple claim types requested.

299 The outline for the `ic:ClaimType` element is as follows:

300 **Syntax:**

301
```
<ic:ClaimType Uri="xs:anyURI" Optional="xs:boolean"? xmlns:ic="..." /> *
```

302 The following describes the attributes and elements listed in the schema outlined above:

303 */ic:ClaimType*

304 Indicates the requested claim type.

305 */ic:ClaimType/@Uri*

306 The unique identifier of the requested claim type.

307 */ic:ClaimType/@Optional*

308 Indicates if the claim can be absent in the Security Token. By default, any requested claim type is
309 a required claim and MUST be present in the issued Security Token.

310 Two `<ic:ClaimType>` elements refer to the same claim type if and only if the values of their XML
311 attribute named `Uri` are equal in a case-sensitive string comparison.

312 When the `ic:ClaimType` element is used within the `wst:Claims` parameter in a token policy to specify
313 claims requirement, the `wst:Dialect` attribute on the `wst:Claims` element MUST be qualified with the
314 URI value below.

315 **Dialect URI:**

316
```
http://schemas.xmlsoap.org/ws/2005/05/identity
```

317 The above dialect URI value indicates that the specified claim elements are to be processed according to
318 this profile.

319 Following is an example of using this assertion within an issued token policy to require two claim types
320 where one claim type is optional.

321 *Example:*

322
```
<sp:IssuedToken xmlns:sp="..." xmlns:wst="..." xmlns:ic="..." ...>
  ...
  <sp:RequestSecurityTokenTemplate>
    ...
    <wst:Claims
        Dialect="http://schemas.xmlsoap.org/ws/2005/05/identity">
      <ic:ClaimType
          Uri="http://.../ws/2005/05/identity/claims/givenname"/>
      <ic:ClaimType
          Uri="http://.../ws/2005/05/identity/claims/surname"
          Optional="true" />
    </wst:Claims>
  </sp:RequestSecurityTokenTemplate>
  ...
</sp:IssuedToken>
```

337 This profile also defines a standard set of claim types for common personal information about users that
338 may be requested by Relying Party Web services in Security Tokens and supported by any Identity
339 Provider. These standard claim types are defined in Section 7.5.

340 ## 2.2 Expressing Privacy Policy of Relying Party

341 A Relying Party Web service SHOULD publish its “Privacy Policy”. Users might decide to release tokens
342 and interact further with that service based on its Privacy Policy. No assumptions are made regarding the

Copyright © OASIS® 2008-2009. All Rights Reserved.

format and content of the Privacy Policy and an Identity Selector is NOT REQUIRED to parse, interpret or act on the Privacy Policy programmatically.

To express the location of its privacy statement, a Web service MUST use the optional policy assertion `ic:PrivacyNotice` defined below:

Syntax:

```
<ic:PrivacyNotice Version="xs:unsignedInt"? xmlns:ic="...">
  xs:anyURI
</ic:PrivacyNotice>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:PrivacyNotice

This element is used to express the location of the privacy statement of a Web service.

/ic:PrivacyNotice/@Version

This optional attribute provides a version number for the privacy statement allowing changes in its content to be reflected as a change in the version number. If present, it MUST have a minimum value of 1.

Following is an example of using this policy element to express the location of the privacy statement of a Web service.

Example:

```
<wsp:Policy xmlns:wsp="..." xmlns:ic="...">
  ...
  <ic:PrivacyNotice Version="1">
    http://www.contoso.com/privacy
  </ic:PrivacyNotice>
  ...
</wsp:Policy>
```

An Identity Selector MUST be able to accept a privacy statement location specified as an URL using the [HTTP] scheme (as illustrated above) or the [HTTPS] scheme.

Because the Privacy Policy assertion points to a “privacy statement” that applies to a service endpoint, the assertion MUST apply to [Endpoint Policy Subject]. In other words, a policy expression containing the Privacy Policy assertion MUST be attached to a `wsdl:binding` in the metadata for the service.

Further, when an Identity Selector can only render the privacy statement document in a limited number of document formats (media types), it MAY use the HTTP request-header field “Accept” in its HTTP GET request to specify the media-types it can accept. For example, the following request-header specifies that the client will accept the Privacy Policy only as a plain text or a HTML document.

```
Accept: text/plain, text/html
```

Similarly, if an Identity Selector wants to obtain the privacy statement in a specific language, it MAY use the HTTP request-header field “Accept-Language” in its HTTP GET request to specify the languages it is willing to accept. For example, the following request-header specifies that the client will accept the Privacy Policy only in Danish.

```
Accept-Language: da
```

A Web service, however, is NOT REQUIRED to be able to fulfill the document format and language requests of an Identity Selector. It MAY publish its privacy statement in a fixed set of document formats and languages.

Copyright © OASIS® 2008-2009. All Rights Reserved.

2.3 Employing Relying Party STSs

The Security Policy of a Relying Party MAY require that an issued token be obtained from a Relying Party STS. This can create a chain of STSs. The Identity Selector MUST follow the RP/STS chain, contacting each referenced STS, resolving its Policy statements and continuing to the STS it refers to.

When following a chain of STSs, when an STS with an `ic:RequireFederatedIdentityProvisioning` declaration is encountered as per Section 3.2.1, this informs the Identity Selector that the STS is an IP/STS and therefore ends the STS chain, rather than a member of the RP/STS chain. Furthermore, if an RP or RP/STS provides an incomplete Security Policy, such as no issuer or no required claims, the Identity Selector MUST be invoked so a card and requested claims can be selected by the user, enabling a Request for Security Token (RST) to be constructed and sent to the selected IP/STS.

The RP/STS's Policy is used for card matching. If the RP/STS requests a private personal identifier (PPID) claim (see Section 7.5.14), the RP/STS's certificate is used for calculating PPID, Signing Key, and Client Pseudonym (see Section 3.3.4) values – not the certificate of the Relying Party. This enables a single RP/STS to service multiple Relying Parties while always receiving the same PPID value for a given user from the Identity Selector.

Identity Selectors MUST enable users to make Relying Party trust decisions based on the identity of the Relying Party, possibly including displaying attributes from its certificate. By trusting the RP, the user is implicitly trusting the chain of RP/STSs that the RP employs.

Each RP/STS endpoint MUST provide a certificate. This certificate MAY be communicated either via Transport (such as HTTPS) or Message (such as WS-Security) Security. If Message Security is employed, transports not providing security (such as HTTP) MAY be used.

Like IP/STSs, RP/STSs publish endpoint metadata. This metadata MAY be retrieved via either WS-MetadataExchange or HTTPS GET in the same manner that IP/STS metadata can be, as described in Section 3.1.1.2.

Like IP/STSs, no changes to the syntax used to specify metadata locations occurs when RP/STS metadata is published by the Relying Party STS as a page retrievable using HTTPS GET. Relying Parties and Identity Providers MAY consequently support either or both retrieval methods for the same metadata addresses.

3 Identity Provider Interactions

This section defines the constructs used by an Identity Selector for interacting with an Identity Provider to obtain Information Cards, and to request and obtain Security Tokens.

3.1 Information Card

An Information Card represents a Digital Identity of a Subject that can be issued by an Identity Provider. It is an artifact containing metadata that represents the token issuance relationship between an Identity Provider and a Subject, and provides a visual representation of the Digital Identity. Multiple Digital Identities for a Subject from the same Identity Provider are represented by different Information Cards. Subjects may obtain an Information Card from an Identity Provider, and may have a collection of Information Cards from various Identity Providers.

3.1.1 Information Card Format

An Information Card is represented as a signed XML document that is issued by an Identity Provider. The XML schema for an Information Card is defined below:

Syntax:

```
<ic:InformationCard xml:lang="xs:language"
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
    xmlns:ic="..." xmlns:ic07="..." ...>
  <ic:InformationCardReference> ... </ic:InformationCardReference>
  <ic:CardName> xs:string </ic:CardName> ?
  <ic:CardImage MimeType="xs:string"> xs:base64Binary </ic:CardImage> ?
  <ic:Issuer> xs:anyURI </ic:Issuer>
  <ic:TimeIssued> xs:dateTime </ic:TimeIssued>
  <ic:TimeExpires> xs:dateTime </ic:TimeExpires> ?
  <ic:TokenServiceList> ... </ic:TokenServiceList>
  <ic:SupportedTokenTypeList> ... </ic:SupportedTokenTypeList> ?
  <ic:SupportedClaimTypeList> ... </ic:SupportedClaimTypeList> ?
  <ic:RequireAppliesTo ...> ... </ic:RequireAppliesTo> ?
  <ic:PrivacyNotice ...> ... </ic:PrivacyNotice> ?
  <ic07:RequireStrongRecipientIdentity /> ?
  <ic07:IssuerInformation> ... </ic07:IssuerInformation> *
  ...
</ic:InformationCard>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:InformationCard

An Information Card issued by an Identity Provider.

/ic:InformationCard/@xml:lang

A required language identifier, using the language codes specified in [RFC 3066], in which the content of localizable elements have been localized.

/ic:InformationCard/ic:InformationCardReference

This required element provides a specific reference for the Information Card by which it can be uniquely identified within the scope of an issuer. This reference MUST be included by an Identity Selector in all token requests sent to the Identity Provider based on that Information Card. The detailed schema of this element is defined in Section 3.1.1.1.

/ic:InformationCard/ic:CardName

This optional element provides a friendly textual name for the issued Information Card. The content of this element MAY be localized in a specific language.

/ic:InformationCard/ic:CardImage

This optional element contains a base64 encoded inline image that provides a graphical image for the issued Information Card. It SHOULD contain an image within the size range of 60 pixels wide by 40 pixels high and 240 pixels wide by 160 pixels high. It is RECOMMENDED that the image have an aspect ratio of 3:2 and the image size be 120 by 80 pixels.

/ic:InformationCard/ic:CardImage/@MimeType

This required attribute provides a MIME type specifying the format of the included card image. This value MUST be one of the five image formats: `image/jpeg`, `image/gif`, `image/bmp`, `image/png`, or `image/tiff`.

/ic:InformationCard/ic:Issuer

This required element provides a logical name for the issuer of the Information Card. If a Relying Party specifies a token issuer by its logical name, then the content of this element MUST be used to match the requested token issuer with an Information Card.

/ic:InformationCard/ic:TimeIssued

This required element provides the date and time when the Information Card was issued.

/ic:InformationCard/ic:TimeExpires

This optional element provides the date and time after which the Information Card SHOULD be treated as expired and invalid.

Copyright © OASIS® 2008-2009. All Rights Reserved.

/ic:InformationCard/ic:TokenServiceList

This required element provides an ordered list of Security Token Service (IP/STS) endpoints, and corresponding credential descriptors (implying the REQUIRED authentication mechanisms), where tokens can be requested. Each service endpoint MUST be tried in order by the Service Requester when requesting tokens.

/ic:InformationCard/ic:SupportedTokenTypeList

This optional element contains the list of token types that are offered by the Identity Provider.

/ic:InformationCard/ic:SupportedClaimTypeList

This optional element contains the list of claim types that are offered by the Identity Provider.

/ic:InformationCard/ic:RequireAppliesTo

This optional element indicates that token requests MUST include information identifying the Relying Party where the issued token will be used. The Relying Party information MUST be included as the content of a `wsp:AppliesTo` element in the token request.

/ic:InformationCard/ic:PrivacyNotice

This optional element provides the location of the privacy statement of the Identity Provider.

/ic:InformationCard/ic07:RequireStrongRecipientIdentity

This optional element informs the Identity Selector that it MUST only allow the card to be used at a Relying Party that presents a cryptographically protected identity, for example, an X.509v3 certificate.

/ic:InformationCard/ic07:IssuerInformation

This optional element provides information from the card issuer about the card that can be displayed by the Identity Selector user interface.

.../ic:InformationCard/@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:InformationCard` element of an `ic:RoamingStore` when representing the card in the Information Cards Transfer Format in Section 6.1.

.../ic:InformationCard/{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:InformationCard` element of an `ic:RoamingStore` when representing the card in the Information Cards Transfer Format in Section 6.1.

3.1.1.1 Information Card Reference

Every Information Card issued by an Identity Provider MUST have a unique reference by which it can be identified within the scope of the Identity Provider. This reference is included in all token requests sent to the Identity Provider based on that Information Card.

The card reference MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:InformationCardReference xmlns:ic="...">
  <ic:CardId> xs:anyURI </ic:CardId>
  <ic:CardVersion> xs:unsignedInt </ic:CardVersion>
</ic:InformationCardReference>
```

The following describes the attributes and elements listed in the schema outlined above:

Copyright © OASIS® 2008-2009. All Rights Reserved.

.../ic:InformationCardReference

A specific reference for an Information Card.

.../ic:InformationCardReference/ic:CardId

This required element provides a unique identifier in the form of a URI for the specific Information Card. The identifier provider MUST be able to identify the specific Information Card based on this identifier.

.../ic:InformationCardReference/ic:CardVersion

This required element provides a versioning epoch for the Information Card issuance infrastructure used by the Identity Provider. The minimum value for this field MUST be 1. Note that it is possible to include version information in CardId as it is a URI, and can have hierarchical content. However, it is specified as a separate value to allow the Identity Provider to change its issuance infrastructure, and thus its versioning epoch, independently without changing the CardId of all issued Information Cards. For example, when an Identity Provider makes a change to the supported claim types or any other policy pertaining to the issued cards, the version number allows the Identity Provider to determine if the Information Card needs to be refreshed. The version number is assumed to be monotonically increasing. If two Information Cards have the same CardId value but different CardVersion values, then the one with a higher numerical CardVersion value SHOULD be treated as being more up-to-date.

3.1.1.2 Token Service Endpoints and Authentication Mechanisms

Every Information Card issued by an Identity Provider MUST include an ordered list of IP/STS endpoints, and the corresponding credential type to be used, for requesting tokens. The list MUST be in a decreasing order of preference. Identity Selectors SHOULD attempt to use the endpoints in the order listed, using the first endpoint in the list for which the metadata is retrievable and the endpoint is reachable. For each endpoint, the credential type implicitly determines the authentication mechanism to be used. Each credential descriptor is personalized for the user to allow an Identity Selector to automatically locate the credential once the user has selected an Information Card.

Further, each IP/STS endpoint reference in the Information Card MUST include the Security Policy metadata for that endpoint. The policy metadata MAY be specified as a metadata location within the IP/STS endpoint reference. If a metadata location URL is specified, it MUST use the [HTTPS] transport. An Identity Selector MAY retrieve the Security Policy it will use to communicate with the IP/STS from that metadata location using the mechanism specified in [WS-MetadataExchange].

The ordered list of token service endpoints MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:TokenServiceList xmlns:ic="..." xmlns:wsa="...">
 (<ic:TokenService>
   <wsa:EndpointReference> ... </wsa:EndpointReference>
   <ic:UserCredential>
    <ic:DisplayCredentialHint> xs:string </ic:DisplayCredentialHint> ?
   (
    <ic:UsernamePasswordCredential>...</ic:UsernamePasswordCredential> |
    <ic:KerberosV5Credential>...</ic:KerberosV5Credential> |
    <ic:X509V3Credential>...</ic:X509V3Credential> |
    <ic:SelfIssuedCredential>...</ic:SelfIssuedCredential> | ...
   )
   </ic:UserCredential>
  </ic:TokenService>) +
</ic:TokenServiceList>
```

The following describes the attributes and elements listed in the schema outlined above:

Copyright © OASIS® 2008-2009. All Rights Reserved.

.../ic:TokenServiceList

This required element provides an ordered list of Security Token Service endpoints (in decreasing order of preference), and the corresponding credential types, for requesting tokens. Each service endpoint MUST be tried in order by a Service Requester.

.../ic:TokenServiceList/ic:TokenService

This required element describes a single token issuing endpoint.

.../ic:TokenServiceList/ic:TokenService/wsa:EndpointReference

This required element provides the endpoint reference for a single token issuing endpoint. For the Self-issued Identity Provider, the special address value defined in Section 2.1.1 MAY be used. The `wsai:Identity` extension element (see Section 12) for endpoint references MAY be used to include the protection token for this endpoint to secure communications with it.

.../ic:TokenServiceList/ic:TokenService/ic:UserCredential

This required element indicates the credential type to use to authenticate to the token issuing endpoint.

.../ic:TokenServiceList/ic:TokenService/ic:UserCredential/ic:DisplayCredentialHint

This optional element provides a hint (string) to be displayed to the user to prompt for the correct credential (e.g. a hint to insert the right smart card). The content of this element MAY be localized in a specific language.

.../ic:TokenServiceList/ic:TokenService/ic:UserCredential/<credential descriptor>

This required element provides an unambiguous descriptor for the credential to use for authenticating to the token issuing endpoint. The schema to describe the credential is specific to each credential type. This profile defines the schema elements `ic:UsernamePasswordCredential`, `ic:KerberosV5Credential`, `ic:X509V3Credential` or `ic:SelfIssuedCredential` later in Section 4 corresponding to username/password, Kerberos v5, X.509v3 certificate and self-issued token based credential types. Other credential types MAY be introduced via the extensibility point defined in the schema within this element.

Alternatively, Identity Providers MAY publish metadata for Information Cards as WSDL documents that can be retrieved by Identity Selectors via HTTPS GET operations on URLs using the HTTPS scheme. An endpoint's metadata URL is communicated to Identity Selectors in a token service `wsx:MetadataReference` element in an Information Card using exactly the same syntax as when WS-MetadataExchange is employed to retrieve the metadata. No change occurs in the card.

The metadata documents published via HTTPS GET SHOULD contain the WSDL for the endpoint as the top-level element of the document without any SOAP or WS-MetadataExchange elements enclosing it.

Identity Providers MAY publish endpoint metadata via both the HTTPS GET and WS-MetadataExchange methods at the same metadata location. If they publish the metadata via multiple mechanisms, the metadata delivered via both mechanisms SHOULD be the same. Likewise, Identity Selectors MAY attempt to retrieve metadata via multiple mechanisms, either in sequence or in parallel.

The following example illustrates an Identity Provider with two endpoints for its IP/STS, one requiring Kerberos (higher priority) and the other requiring username/password (lower priority) as its authentication mechanism. Further, each endpoint also includes its policy metadata location as a URL using the [HTTPS] scheme.

Example:

```
<ic:TokenServiceList xmlns:ic="..." xmlns:wsa="..." xmlns:wsai="..."
    xmlns:wsx="...">
  <ic:TokenService>
    <wsa:EndpointReference>
      <wsa:Address>http://contoso.com/sts/kerb</wsa:Address>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
      <wsai:Identity>
        <wsai:Spn>host/corp-sts.contoso.com</wsai:Spn>
      </wsai:Identity>
      <wsa:Metadata>
        <wsx:Metadata>
          <wsx:MetadataSection
              Dialect="http://schemas.xmlsoap.org/ws/2004/09/mex">
            <wsx:MetadataReference>
              <wsa:Address>https://contoso.com/sts/kerb/mex</wsa:Address>
            </wsx:MetadataReference>
          </wsx:MetadataSection>
        </wsx:Metadata>
      </wsa:Metadata>
    </wsa:EndpointReference>
    <ic:UserCredential>
      <ic:KerberosV5Credential />
    </ic:UserCredential>
  </ic:TokenService>
  <ic:TokenService>
    <wsa:EndpointReference>
      <wsa:Address>http://contoso.com/sts/pwd</wsa:Address>
      <wsa:Metadata>
        <wsx:Metadata>
          <wsx:MetadataSection
              Dialect="http://schemas.xmlsoap.org/ws/2004/09/mex">
            <wsx:MetadataReference>
              <wsa:Address>https://contoso.com/sts/pwd/mex</wsa:Address>
            </wsx:MetadataReference>
          </wsx:MetadataSection>
        </wsx:Metadata>
      </wsa:Metadata>
    </wsa:EndpointReference>
    <ic:UserCredential>
      <ic:UsernamePasswordCredential>
        <ic:Username>Zoe</ic:Username>
      </ic:UsernamePasswordCredential>
    </ic:UserCredential>
  </ic:TokenService>
</ic:TokenServiceList>
```

3.1.1.3 Token Types Offered

Every Information Card issued by an Identity Provider SHOULD include an unordered list of token types that can be issued by the Identity Provider. The set of token types offered by the Identity Provider MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:SupportedTokenTypeList xmlns:ic="..." xmlns:wst="...">
  <wst:TokenType> xs:anyURI </wst:TokenType> +
</ic:SupportedTokenTypeList>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:SupportedTokenTypeList

This optional element contains the set of token types offered by the Identity Provider.

.../ic:SupportedTokenTypeList/wst:TokenType

This required element indicates an individual token type that is offered.

The following example illustrates an Identity Provider that offers both SAML 1.1 and SAML 2.0 tokens.

Example:

```
<ic:SupportedTokenTypeList xmlns:ic="..." xmlns:wst="...">
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
  <wst:TokenType>urn:oasis:names:tc:SAML:1.0:assertion</wst:TokenType>
  <wst:TokenType>urn:oasis:names:tc:SAML:2.0:assertion</wst:TokenType>
</ic:SupportedTokenTypeList>
```

3.1.1.4 Claim Types Offered

Every Information Card issued by an Identity Provider SHOULD include an unordered list of claim types that can be issued by the Identity Provider. The set of claim types offered by the Identity Provider MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:SupportedClaimTypeList xmlns:ic="...">
  (<ic:SupportedClaimType Uri="xs:anyURI">
    <ic:DisplayTag> xs:string </ic:DisplayTag> ?
    <ic:Description> xs:string </ic:Description> ?
  </ic:SupportedClaimType>) +
</ic:SupportedClaimTypeList>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:SupportedClaimTypeList

This optional element contains the set of claim types offered by the Identity Provider.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType

This required element indicates an individual claim type that is offered.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType/@Uri

This required attribute provides the unique identifier (URI) of this individual claim type offered.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType/ic:DisplayTag

This optional element provides a friendly name for this individual claim type. The content of this element MAY be localized in a specific language.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType/ic:Description

This optional element provides a description of the semantics for this individual claim type. The content of this element MAY be localized in a specific language.

The following example illustrates an Identity Provider that offers two claim types.

Example:

```
<ic:SupportedClaimTypeList xmlns:ic="...">
  <ic:SupportedClaimType Uri=".../ws/2005/05/identity/claims/givenname">
    <ic:DisplayTag>Given Name</DisplayTag>
  </ic:SupportedClaimType>
  <ic:SupportedClaimType Uri=".../ws/2005/05/identity/claims/surname">
    <ic:DisplayTag>Last Name</DisplayTag>
  </ic:SupportedClaimType>
</ic:SupportedClaimTypeList>
```

3.1.1.5 Requiring Token Scope Information

An Identity Selector, by default, SHOULD NOT convey information about the Relying Party where an issued token will be used (i.e., target scope) when requesting Security Tokens. This helps safeguard user privacy. However, an Identity Provider MAY override that behavior.

Every Information Card issued by an Identity Provider MAY include a requirement that token requests include token scope information identifying the Relying Party where the token will be used. The requirement to submit token scope information MUST be expressed using the following schema element within an Information Card.

Syntax:

Copyright © OASIS® 2008-2009. All Rights Reserved.

```
<ic:RequireAppliesTo Optional="xs:boolean" xmlns:ic="..." /> ?
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:RequireAppliesTo

This optional element indicates a requirement for a token requester to submit token scope information in the request. Absence of this element in an Information Card means that the token requester MUST NOT submit any token scope information.

.../ic:RequireAppliesTo/@Optional

This optional attribute indicates whether the token scope information is required or is optional by the Identity Provider. An attribute value of “true” indicates that the token scope information is not required, but will be accepted by the Identity Provider if submitted. An attribute value of “false” (default) indicates that the token scope information is required.

The following example illustrates the use of this element.

Example:

```
<ic:RequireAppliesTo Optional="true" xmlns:ic="..." />
```

If token scope information is required by an Identity Provider, an Identity Selector MUST include the Relying Party identity as the content of the `wsp:AppliesTo` element in the token request. The actual behavior of an Identity Selector vis-à-vis the possible requirements that can be expressed by the above element is specified in Section 3.3.3.

3.1.1.6 Privacy Policy Location

Every Information Card issued by an Identity Provider SHOULD include a pointer to the privacy statement of the Identity Provider. The location of the privacy statement MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:PrivacyNotice Version="xs:unsignedInt" xmlns:ic="..." /> ?
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:PrivacyNotice

This optional element provides the location of the privacy statement of the Identity Provider.

.../ic:PrivacyNotice/@Version

This optional attribute indicates a version number that tracks changes in the content of the privacy statement. This field MUST have a minimum value of 1 when present.

The following example illustrates the use of this element.

Example:

```
<ic:PrivacyNotice Version="1" xmlns:ic="...">
  http://www.contoso.com/privacynotice
</ic:PrivacyNotice>
```

An Identity Selector MUST be able to accept a privacy statement location specified as an URL using the [HTTP] scheme (as illustrated above) or the [HTTPS] scheme.

3.1.1.7 Prohibiting Use at Relying Parties Not Identified by a Cryptographically Protected Identity

Information Cards issuers MAY specify that a card MUST NOT be used at Relying Parties that do not present a cryptographically protected identity, such as an X.509v3 certificate. This would typically be done when the issuer determines that the use of HTTP without Message Security would not provide a sufficiently secure environment for the use of the card.

Copyright © OASIS® 2008-2009. All Rights Reserved.

Syntax:

```
<ic07:RequireStrongRecipientIdentity xmlns:ic07="..." /> ?
```

.../ic07:RequireStrongRecipientIdentity

This optional element informs the Identity Selector that it MUST only allow the card to be used at a Relying Party that presents a cryptographically protected identity, such as an X.509v3 certificate.

3.1.1.8 Providing Custom Data to Display with the Card

Card issuers MAY supply a set of information about the card that MAY be displayed by the Identity Selector user interface.

Syntax:

```
<ic07:IssuerInformation xmlns:ic07="...">
  <ic07:IssuerInformationEntry>
    <ic07:EntryName> xs:string </ic07:EntryName>
    <ic07:EntryValue> xs:string </ic07:EntryValue>
  </ic07:IssuerInformationEntry> +
</ic07:IssuerInformation>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic07:IssuerInformation

This optional element provides a set of information from the card issuer about the card that can be displayed by the Identity Selector user interface.

.../ic07:IssuerInformation/IssuerInformationEntry

This required element provides one item of information about the card.

.../ic07:IssuerInformation/IssuerInformationEntry/EntryName

This required element provides the name of one item of information about the card.

.../ic07:IssuerInformation/IssuerInformationEntry/EntryValue

This required element provides the value of one item of information about the card.

The following example illustrates the use of this feature.

Example:

```
<ic07:IssuerInformation xmlns:ic07="...">
  <ic07:IssuerInformationEntry>
    <ic07:EntryName>Customer Service</ic07:EntryName>
    <ic07:EntryValue>+1-800-CONTOSO</ic07:EntryValue>
  </ic07:IssuerInformationEntry>
  <ic07:IssuerInformationEntry>
    <ic07:EntryName>E-mail Contact</ic07:EntryName>
    <ic07:EntryValue>cardhelp@contoso.com</ic07:EntryValue>
  </ic07:IssuerInformationEntry>
</ic07:IssuerInformation>
```

3.1.2 Issuing Information Cards

An Identity Provider can issue Information Cards to its users using any out-of-band mechanism that is mutually suitable.

In order to provide the assurance that an Information Card is indeed issued by the Identity Provider expected by the user, the Information Card MUST be carried inside a digitally signed envelope that is signed by the Identity Provider. For this, the “enveloping signature” construct (see [XMLDSIG]) MUST be used where the Information Card is included in the `ds:Object` element. The signature on the digitally

Copyright © OASIS® 2008-2009. All Rights Reserved.

809 signed envelope provides data origin authentication assuring the user that it came from the right Identity
810 Provider.

811 The specific profile of XML digital signatures [XMLDSIG] that is RECOMMENDED for signing the
812 envelope carrying the Information Card is as follows. Usage of other algorithms is not described.

813 - Use *enveloping signature* format when signing the Information Card XML document.

814 - Use a single `ds:Object` element within the signature to hold the `ic:InformationCard`
815 element that represents the issued Information Card. The `ds:Object/@Id` attribute provides a
816 convenient way for referencing the Information Card from the `ds:SignedInfo/ds:Reference`
817 element within the signature.

818 - Use RSA signing and verification with the algorithm identifier given by the URI
819 *http://www.w3.org/2000/09/xmldsig#rsa-sha1*.

820 - Use exclusive canonicalization with the algorithm identifier given by the URI
821 *http://www.w3.org/2001/10/xml-exc-c14n#*.

822 - Use SHA1 digest method for the data elements being signed with the algorithm identifier
823 *http://www.w3.org/2000/09/xmldsig#sha1*.

824 - There MUST NOT be any other transforms used in the enveloping signature for the Information
825 Card other than the ones listed above.

826 - The `ds:KeyInfo` element MUST be present in the signature carrying the signing key information
827 in the form of an X.509 v3 certificate or a X.509 v3 certificate chain specified as one or more
828 `ds:X509Certificate` elements within a `ds:X509Data` element.

829 The following example shows an enveloping signature carrying an Information Card that is signed by the
830 Identity Provider using the format outlined above. Note that whitespace (newline and space character) is
831 included in the example only to improve readability; they might not be present in an actual
832 implementation.

833 *Example:*

```
834 <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
835   <SignedInfo>
836     <CanonicalizationMethod
837       Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
838     <SignatureMethod
839       Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
840     <Reference URI="#_Object_InformationCard">
841       <Transforms>
842         <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
843       </Transforms>
844       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
845       <DigestValue> ... </DigestValue>
846     </Reference>
847   </SignedInfo>
848   <SignatureValue> ... </SignatureValue>
849   <KeyInfo>
850     <X509Data>
851       <X509Certificate> ... </X509Certificate>
852     </X509Data>
853   </KeyInfo>
854   <Object Id="_Object_InformationCard">
855     <ic:InformationCard
856         xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
857         xml:lang="en-us">
858       [Information Card content]
859     </ic:InformationCard>
860   </Object>
861 </Signature>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

An Identity Selector MUST verify the enveloping signature. The `ic:InformationCard` element can then be extracted and stored in the Information Card collection.

3.2 Identity Provider Policy

This section specifies additional policy elements and requirements introduced by this profile for an IP/STS policy metadata.

3.2.1 Require Information Card Provisioning

In the Information Card Model, an Identity Provider requires provisioning in the form of an Information Card issued by it which represents the provisioned identity of the user. In order to enable an Identity Selector to learn that such pre-provisioning is necessary before token requests can be made, the Identity Provider MUST provide an indication in its policy.

An Identity Provider issuing Information Cards MUST specify this provisioning requirement in its policy using the following schema element.

Syntax:

```
<ic:RequireFederatedIdentityProvisioning xmlns:ic="..." />
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:RequireFederatedIdentityProvisioning

This element indicates a requirement that one or more Information Cards, representing identities that can be federated, MUST be pre-provisioned before token requests can be made to the Identity Provider.

The following example illustrates the use of this policy element.

Example:

```
<wsp:Policy xmlns:wsp="..." xmlns:ic="..." xmlns:sp="...">
  ...
  <ic:RequireFederatedIdentityProvisioning />
  <sp:SymmetricBinding>
    ...
  </sp:SymmetricBinding>
  ...
</wsp:Policy>
```

3.2.2 Policy Metadata Location

In the Information Card Model, an Identity Provider MUST make the Security Policy metadata for its IP/STS endpoints available. If a metadata location is used for this purpose, the location URL MUST use the [HTTPS] scheme. An Identity Selector MAY retrieve the Security Policy it will use to communicate with the IP/STS from that metadata location using the mechanism specified in [WS-MetadataExchange].

3.3 Token Request and Response

For any given Information Card, an Identity Selector can obtain a Security Token from the IP/STS for that Information Card. Tokens MUST be requested using the “Issuance Binding” mechanism described in [WS-Trust 1.2] and [WS-Trust 1.3]. This section specifies additional constraints and extensions to the token request and response messages between the Identity Selector and the IP/STS.

The WS-Trust protocol requires that a token request be submitted by using the `wst:RequestSecurityToken` element in the request message, and that a token response be sent using the `wst:RequestSecurityTokenResponse` element in the response message. This profile refers to the “Request Security Token” message as RST and the “Request Security Token Response” message as RSTR in short.

Copyright © OASIS® 2008-2009. All Rights Reserved.

The WS-Trust protocol allows for a token response to provide multiple tokens by using the `wst:RequestSecurityTokenResponseCollection` element in the response message. This profile, however, requires that an Identity Provider MUST NOT use the `wst:RequestSecurityTokenResponseCollection` element in the response. The token response MUST consist of a single `wst:RequestSecurityTokenResponse` element.

3.3.1 Information Card Reference

When requesting a Security Token from the IP/STS, an Identity Selector MUST include the Information Card reference in the body of the RST message as a top-level element information item. The `ic:InformationCardReference` element in the Information Card, including all of its [children], [attributes] and [in-scope namespaces], MUST be copied as an immediate child of the RST element in the message as follows.

The following example illustrates the Information Card reference included in a RST message.

Example:

```
<wst:RequestSecurityToken xmlns:wst="..." xmlns:ic="...">
  ...
  <ic:InformationCardReference>
    <ic:CardId>http://xyz.com/CardId/d795621fa01d454285f9</ic:CardId>
    <ic:CardVersion>1</ic:CardVersion>
  </ic:InformationCardReference>
  ...
</wst:RequestSecurityToken>
```

The IP/STS MAY fault with `ic:InformationCardRefreshRequired` to signal to the Service Requester that the Information Card needs to be refreshed.

3.3.2 Claims and Other Token Parameters

A Relying Party's requirements of claims and other token parameters are expressed in its policy using the `sp:RequestSecurityTokenTemplate` parameter within the `sp:IssuedToken` policy assertion (see Section 2.1). If all token parameters are acceptable to the Identity Selector, it MUST copy the content of this element (i.e. all of its [children] elements) into the body of the RST message as top-level element information items. However, if optional claims are requested by the Relying Party, requests for optional claims not selected by the user MUST NOT be copied into the RST message.

3.3.3 Token Scope

The WS-Trust protocol allows a token requester to indicate the target where the issued token will be used (i.e., token scope) by using the `optional` element `wsp:AppliesTo` in the RST message. By default, an Identity Selector SHOULD NOT send token scope information to the Identity Provider in token requests to protect user privacy. In other words, the element `wsp:AppliesTo` is absent in the RST message.

However, if the Identity Provider requires it (see the modes of the `ic:RequireAppliesTo` element described in Section 3.1.1.5), or if the Relying Party's token policy includes the `wsp:AppliesTo` element in the `sp:RequestSecurityTokenTemplate` parameter, then an Identity Selector MUST include token scope information in its token request as per the behavior summarized in the following table.

<RequireAppliesTo> mode in Information Card	*<AppliesTo> element present in RP policy*	*Resulting behavior of Identity Selector*
Mandatory	Yes	Send <AppliesTo> value from RP policy in token request to IP.
Mandatory	No	Send the RP endpoint to which token will be sent as the value of

Copyright © OASIS® 2008-2009. All Rights Reserved.

		<AppliesTo> in token request to IP.
Optional	Yes	Send <AppliesTo> value from RP policy in token request to IP.
Optional	No	Do not send <AppliesTo> in token request to IP.
Not present	Yes	Fail
Not present	No	Do not send <AppliesTo> in token request to IP.

The following example illustrates the token scope information included in a RST message when it is sent to the Identity Provider.

Example:

```
<wst:RequestSecurityToken xmlns:wst="..." xmlns:wsp="..." xmlns:wsa="..."
    xmlns:wsai="..." xmlns:ds="...">
  <wsp:AppliesTo>
    <wsa:EndpointReference>
      <wsa:Address>http://ip.fabrikam.com</wsa:Address>
      <wsai:Identity>
        <ds:KeyInfo>
          <ds:X509Data>
            <ds:X509Certificate>...</ds:X509Certificate>
          </ds:X509Data>
        </ds:KeyInfo>
      </wsai:Identity>
    </wsa:EndpointReference>
  </wsp:AppliesTo>
  ...
</wst:RequestSecurityToken>
```

3.3.4 Client Pseudonym

A private personal identifier (PPID) claim, defined in Section 7.5.14, identifies a Subject to a Relying Party in a way such that a Subject's PPID at one Relying Party cannot be correlated with the Subject's PPID at another Relying Party. If an Identity Provider offers the PPID claim type then it MUST generate values for the claim that have this prescribed privacy characteristic using data present in the RST request.

When a Relying Party requests a PPID claim, an Identity Selector MUST provide a Client Pseudonym value via an `ic:PPID` element in the RST request that can be used by the IP/STS as input when computing the PPID claim value in the issued token. The Client Pseudonym SHOULD be produced as described in Section 3.3.4.1. It is RECOMMENDED that the IP/STS combine this Client Pseudonym value with information specific to the entity to which the card was issued as well as a secret known only by the IP/STS and pass the combination through a cryptographically non-invertible function, such as a cryptographic hash function, to generate the PPID claim value sent in the token. Alternatively, when target scope information is sent in the token request using the `wsp:AppliesTo` element, the IP/STS MAY instead choose to use that information to generate an appropriate PPID value.

When Client Pseudonym information is included by an Identity Selector in a token request, it MUST be sent using the following schema element.

Syntax:

```
<ic:ClientPseudonym xmlns:ic="...">
  <ic:PPID> xs:base64Binary </ic:PPID>
</ic:ClientPseudonym>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

The following describes the attributes and elements listed in the schema outlined above:

.../ic:ClientPseudonym

This optional top-level element contains the Client Pseudonym information item.

.../ic:ClientPseudonym/ic:PPID

This optional element contains the Client Pseudonym value that the client has submitted for use in computing the PPID claim value for the issued token. The IP/STS MAY use this value as the input (a seed) to a custom cryptographically non-invertible function, with the result used as the PPID claim value in the issued token.

The following example illustrates the Client Pseudonym information sent in a RST message.

Example:

```
<wst:RequestSecurityToken xmlns:wst="..." xmlns:ic="...">
  <ic:ClientPseudonym>
    <ic:PPID>MIIEZzCCA9CgAwIBAgIQEmtJZc0=</ic:PPID>
  </ic:ClientPseudonym >
  ...
</wst:RequestSecurityToken>
```

When the target scope information is not sent in the token request to an IP/STS, the Identity Provider MUST NOT record any Client Pseudonym values included in the RST message. It likewise MUST NOT record the PPID claim value that it generates.

3.3.4.1 PPID

When a token request for a PPID claim is sent to an IP/STS, an Identity Selector SHOULD compute the Client Pseudonym PPID information it sends in the RST message as follows:

- Construct the *RP PPID Seed* as described in Section 7.6.1.
- Decode the base64 encoded value of the `ic:HashSalt` element of the Information Card (see Section 6.1) to obtain *SaltBytes*.
- Decode the base64 encoded value of the `ic:MasterKey` element of the Information Card (see Section 6.1) to obtain *MasterKeyBytes*.
- Hash the concatenation of *MasterKeyBytes*, *RP PPID Seed*, and *SaltBytes* using the SHA256 hash function to obtain the Client Pseudonym PPID value.

  Client Pseudonym PPID = SHA256 (MasterKeyBytes + RP PPID Seed + SaltBytes)
- Convert *Client Pseudonym PPID* to a base64 encoded string and send as the value of the `ic:PPID` element in the RST request.

3.3.5 Proof Key for Issued Token

An issued token can have a *symmetric* proof key (symmetric key token), an *asymmetric* proof key (asymmetric key token), or *no* proof key (bearer token). If no key type is specified in the Relying Party policy, then an Identity Selector SHOULD request an asymmetric key token from the IP/STS by default.

The `optional wst:KeyType` element in the RST request indicates the type of proof key desired in the issued Security Token. The IP/STS MAY return the proof key and/or entropy towards the proof key in the RSTR response. This section describes the behaviors for how each proof key type is requested, who contributes entropy, and how the proof key is computed and returned.

3.3.5.1 Symmetric Proof Key

When requesting a symmetric key token, an Identity Selector MUST submit entropy towards the proof key by augmenting the RST request message as follows:

Copyright © OASIS® 2008-2009. All Rights Reserved.

- The RST SHOULD include a `wst:KeyType` element with one of the two following URI values, depending upon the version of WS-Trust being used:

  http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey

  http://docs.oasis-open.org/ws-sx/ws-trust/200512/SymmetricKey

- The RST MUST include a `wst:BinarySecret` element inside a `wst:Entropy` element containing client-side entropy to be used as partial key material. The entropy is conveyed as raw base64 encoded bits.

The size of the submitted entropy SHOULD be equal to the key size requested in the Relying Party policy. If no key size is specified by the Relying Party, then an Identity Selector SHOULD request a key at least 256-bits in size, and submit an entropy of equal size to the IP/STS.

Following is a sample RST request fragment that illustrates a symmetric key token request.

Example:

```
<wst:RequestSecurityToken xmlns:wst="...">
  ...
  <wst:KeyType>
    http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey
  </wst:KeyType>
  <wst:KeySize>256</wst:KeySize>
  <wst:Entropy>
    <wst:BinarySecret>mQlxWxEiKOcUfnHgQpylcD7LYSkJplpE=</wst:BinarySecret>
  </wst:Entropy>
</wst:RequestSecurityToken>
```

When processing the token request, the IP/STS MAY:

a) accept the client entropy as the sole key material for the proof key,

b) accept the client entropy as partial key material and contribute additional server-side entropy as partial key material to compute the proof key as a function of both partial key materials, or

c) reject the client-side entropy and use server-side entropy as the sole key material for the proof key.

For each of the cases above, the IP/STS MUST compute and return the proof key by augmenting the RSTR response message as follows.

For case (a) where IP/STS accepts client entropy as the sole key material:

- The RSTR MUST NOT include a `wst:RequestedProofToken` element. The proof key is implied and an Identity Selector MUST use the client-side entropy as the proof key.

For case (b) where IP/STS accepts client entropy and contributes additional server entropy:

- The RSTR MUST include a `wst:BinarySecret` element inside a `wst:Entropy` element containing the server-side entropy to be used as partial key material. The entropy is conveyed as raw base64 encoded bits.
- The partial key material from the IP/STS MUST be combined (by each party) with the partial key material from the client to determine the resulting proof key.
- The RSTR MUST include a `wst:RequestedProofToken` element containing a `wst:ComputedKey` element to indicate how the proof key is to be computed. It is RECOMMENDED that an Identity Selector support the P_SHA1 computed key mechanism defined in [WS-Trust 1.2] or [WS-Trust 1.3] with the particulars below. Usage of other algorithms is not described.

ComputedKey Value	*Meaning*

Copyright © OASIS® 2008-2009. All Rights Reserved.

http://schemas.xmlsoap.org/ws/2005/02/trust/CK/PSHA1 or http://docs.oasis-open.org/ws-sx/ws-trust/200512/CK/PSHA1	The key is computed using P_SHA1 from the TLS specification to generate a bit stream using entropy from both sides. The exact form is: key = P_SHA1 ($Entropy_{REQ}$, $Entropy_{RES}$)

Following is a sample RSTR response fragment that illustrates a token response with partial key material from the IP/STS and a computed proof key.

Example:

```
<wst:RequestSecurityTokenResponse xmlns:wst="...">
  ...
  <wst:Entropy>
    <wst:BinarySecret>mQlxWxEiKOcUfnHgQpylcD7LYSkJplpE=</wst:BinarySecret>
  </wst:Entropy>
  <wst:RequestedProofToken>
    <wst:ComputedKey>
       http://schemas.xmlsoap.org/ws/2005/02/trust/CK/PSHA1
    </wst:ComputedKey>
  </wst:RequestedProofToken>
</wst:RequestSecurityTokenResponse>
```

For case (c) where IP/STS contributes server entropy as the sole key material:

- The RSTR MUST include a `wst:BinarySecret` element inside a `wst:RequestedProofToken` element containing the specific proof key to be used. The proof key is conveyed as raw base64 encoded bits.

Following is a sample RSTR response fragment that illustrates a token response with fully specified proof key from the IP/STS.

Example:

```
<wst:RequestSecurityTokenResponse xmlns:wst="...">
  ...
  <wst:RequestedProofToken>
    <wst:BinarySecret>
      mQlxWxEiKOcUfnHgQpylcDKOcUfnHg7LYSkJplpE=
    </wst:BinarySecret>
  </wst:RequestedProofToken>
</wst:RequestSecurityTokenResponse>
```

The following table summarizes the symmetric proof key computation rules to be used by an Identity Selector:

Token Requester (Identity Selector)	*Token Issuer (IP/STS)*	*Results*
Provides entropy	Uses requester entropy as proof key	No <wst:RequestedProofToken> element present in RSTR. Proof key is implied.
Provides entropy	Uses requester entropy and provides additional entropy of its own	<wst:Entropy> element present in RSTR containing issuer supplied entropy. <wst:RequestedProofToken> element present in RSTR containing computed key mechanism. Requestor and Issuer compute proof key by combining both entropies using the specified computed key

Copyright © OASIS® 2008-2009. All Rights Reserved.

		mechanism.
Provides entropy	Uses own entropy as proof key (rejects requester entropy)	<wst:RequestedProofToken> element present in RSTR containing the proof key.

3.3.5.2 Asymmetric Proof Key

When requesting an asymmetric key token, it is RECOMMENDED that an Identity Selector generate an ephemeral RSA key pair. Usage of other algorithms is not described. The generated RSA key pair MUST be at least 1024-bits in size for use as the proof key. It MUST submit the public key to the IP/STS by augmenting the RST request as follows:

- The RST MUST include a `wst:KeyType` element with one of the two following URI values, depending upon the version of WS-Trust being used:

  http://schemas.xmlsoap.org/ws/2005/02/trust/PublicKey

  http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey

- The RST SOAP body MUST include a `wst:UseKey` element containing the public key to be used as proof key in the returned token. The public key is present as a raw RSA key in the form of a `ds:RSAKeyValue` element inside a `ds:KeyValue` element.

- The RST SOAP security header SHOULD include a supporting signature to prove ownership of the corresponding private key. The `ds:KeyInfo` element within the signature, if present, MUST include the same public key as in the `wst:UseKey` element in the SOAP body.

- The supporting signature, if present, MUST be placed in the SOAP security header where the signature for an endorsing supporting token would be placed as per the security header layout specified in WS-SecurityPolicy.

Following is a sample RST request fragment that illustrates an asymmetric key based token request containing the public key and proof of ownership of the corresponding private key.

Example:

```
<s:Envelope xmlns:s="..." xmlns:wsse="..." xmlns:ds="..." xmlns:wst="..."
    ... >
  <s:Header>
    ...
    <wsse:Security>
      ...
      <ds:Signature Id="_proofSignature">
        <!-- signature proving possession of submitted proof key -->
        ...
        <!-- KeyInfo in signature contains the submitted proof key -->
        <ds:KeyInfo>
          <ds:KeyValue>
            <ds:RSAKeyValue>
              <ds:Modulus>...</ds:Modulus>
              <ds:Exponent>...</ds:Exponent>
            </ds:RSAKeyValue>
          </ds:KeyValue>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </s:Header>
  <s:Body wsu:Id="req">
    <wst:RequestSecurityToken>
      ...
      <wst:KeyType>
        http://schemas.xmlsoap.org/ws/2005/02/trust/PublicKey
      </wst:KeyType>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
      <wst:UseKey Sig="#_proofSignature">
        <ds:KeyInfo>
          <ds:KeyValue>
            <ds:RSAKeyValue>
              <ds:Modulus>...</ds:Modulus>
              <ds:Exponent>...</ds:Exponent>
            </ds:RSAKeyValue>
          </ds:KeyValue>
        </ds:KeyInfo>
      </wst:UseKey>
    </wst:RequestSecurityToken>
  </s:Body>
</s:Envelope>
```

If a supporting signature for the submitted proof key is not present in the token request, the IP/STS MAY fail the request. If a supporting signature is present, the IP/STS MUST verify the signature and MUST ensure that the public key included in the `wst:UseKey` element and in the supporting signature are the same. If verification succeeds and the IP/STS accepts the submitted public key for use in the issued token, then the token response MUST NOT include a `wst:RequestedProofToken` element. The proof key is implied and an Identity Selector MUST use the public key it submitted as the proof key.

The following table summarizes the asymmetric proof key rules used by an Identity Selector:

Token Requester (Identity Selector)	*Token Issuer (IP/STS)*	*Results*
Provides ephemeral public key for use as proof key	Uses requester supplied proof key	No <wst:RequestedProofToken> element present in RSTR. Proof key is implied.

3.3.5.3 No Proof Key

When requesting a token with no proof key, an Identity Selector MUST augment the RST request message as follows:

- The RST MUST include a `wst:KeyType` element with the following URI value if [WS-Trust 1.2] is being used:

  http://schemas.xmlsoap.org/ws/2005/05/identity/NoProofKey

  or the RST MUST include a wst:KeyType element with the following URI value if [WS-Trust 1.3] is being used:

  http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer

Following is a sample RST request fragment that illustrates a bearer token request.

Example:

```
<wst:RequestSecurityToken xmlns:wst="...">
  ...
  <wst:KeyType>
    http://schemas.xmlsoap.org/ws/2005/05/identity/NoProofKey
  </wst:KeyType>
</wst:RequestSecurityToken>
```

When processing the token request, if the IP/STS issues a SAML v1.1 bearer token then:

- It MUST specify “urn:oasis:names:tc:SAML:1.0:cm:bearer” as the subject confirmation method in the token.
- It SHOULD include a `saml:AudienceRestrictionCondition` element restricting the token to the target site URL submitted in the token request.

Copyright © OASIS® 2008-2009. All Rights Reserved.

3.3.6 Display Token

An Identity Selector MAY request a Display Token – a representation of the claims carried in the issued Security Token that can be displayed in an user interface – from an IP/STS as part of the token request. To request a Display Token, the following element MUST be included in the RST message as a top-level element information item.

Syntax:

```
<ic:RequestDisplayToken xml:lang="xs:language"? xmlns:ic="..." ... />
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:RequestDisplayToken

This optional element is used to request an Identity Provider to return a Display Token corresponding to the issued token.

/ic:RequestDisplayToken/@xml:lang

This optional attribute indicates a language identifier, using the language codes specified in [RFC 3066], in which the Display Token content SHOULD be localized.

An IP/STS MAY respond to a Display Token request. If it does, it MUST use the following element to return a Display Token for the issued Security Token in the RSTR message.

Syntax:

```
<ic:RequestedDisplayToken xmlns:ic="..." ...>
  <ic:DisplayToken xml:lang="xs:language" ... >
    [ <ic:DisplayClaim Uri="xs:anyURI" ...>
        <ic:DisplayTag> xs:string </ic:DisplayTag> ?
        <ic:Description> xs:string </ic:Description> ?
        <ic:DisplayValue> xs:string </ic:DisplayValue> ?
      </ic:DisplayClaim> ] +
    |
    [ <ic:DisplayTokenText MimeType="xs:string">
        xs:string
      </ic:DisplayTokenText> ]
    ...
  </ic:DisplayToken>
</ic:RequestedDisplayToken>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:RequestedDisplayToken

This optional element is used to return a Display Token for the Security Token returned in the response.

/ic:RequestedDisplayToken/ic:DisplayToken

The returned Display Token.

/ic:RequestedDisplayToken/ic:DisplayToken/@xml:lang

This required attribute indicates a language identifier, using the language codes specified in [RFC 3066], in which the Display Token content is localized.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim

This required element indicates an individual claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/@Uri

This required attribute provides the unique identifier (URI) of the individual claim returned in the Security Token.

Copyright © OASIS® 2008-2009. All Rights Reserved.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/ic:DisplayTag

This optional element provides a friendly name for the claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/ic:Description

This optional element provides a description of the semantics for the claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/ic:DisplayValue

This optional element provides the displayable value for the claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayTokenText

This optional element provides an alternative textual representation of the entire token as a whole when the token content is not suitable for display as individual claims.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayTokenText/@MimeType

This required attribute provides a MIME type specifying the format of the Display Token content (e.g., “text/plain”).

The following example illustrates a returned Display Token corresponding to a Security Token with two claims.

Example:

```
<ic:RequestedDisplayToken xmlns:ic="...">
  <ic:DisplayToken xml:lang="en-us">
    <ic:DisplayClaim Uri="http://.../ws/2005/05/identity/claims/givenname">
      <ic:DisplayTag>Given Name</ic:DisplayTag>
      <ic:DisplayValue>John</ic:DisplayValue>
    </ic:DisplayClaim>
    <ic:DisplayClaim Uri="http://.../ws/2005/05/identity/claims/surname">
      <ic:DisplayTag>Last Name</ic:DisplayTag>
      <ic:DisplayValue>Doe</ic:DisplayValue>
    </ic:DisplayClaim>
  <ic:DisplayToken>
</ic:RequestedDisplayToken>
```

3.3.7 Token References

When an IP/STS returns the token requested by an Identity Selector, it MUST also include an attached and an un-attached token reference for the issued security token using the `wst:RequestedAttachedReference` and `wst:RequestedUnattachedReference` elements, respectively, in the RSTR response message.

An Identity Selector is truly a conduit for the security tokens issued by an IP/STS and requested by an RP, and it should remain agnostic of the type of the security token passing through it. Furthermore, a security token issued by an IP/STS MAY be encrypted directly for the RP, thus preventing visibility into the token by the Identity Selector. However, an Identity Selector (or a client application) needs to be able to use the issued security token to perform security operations (such as signature or encryption) on a message sent to an RP and thus needs a way to reference the token both when it is attached to a message and when it is not. The attached and unattached token references returned by an IP/STS in the RSTR message provide the necessary references that can be used for this purpose.

4 Authenticating to Identity Provider

The Information Card schema includes the element content necessary for an Identity Provider to express what credential the user must use in order to authenticate to the IP/STS when requesting tokens. This section defines the schema used to express the credential descriptor for each supported credential type.

Copyright © OASIS® 2008-2009. All Rights Reserved.

4.1 Username and Password Credential

When the Identity Provider requires a *username* and *password* as the credential type, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential xmlns:ic="...">
  <ic:UsernamePasswordCredential>
    <ic:Username> xs:string </ic:Username> ?
  </ic:UsernamePasswordCredential>
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:UsernamePasswordCredential

This element indicates that a username/password credential is needed.

.../ic:UsernamePasswordCredential/ic:Username

This optional element provides the username part of the credential for convenience. An Identity Selector MUST prompt the user for the password. If the username is specified, then its value MUST be copied into the username token used to authenticate to the IP/STS; else an Identity Selector MUST prompt the user for the username as well.

Furthermore, the actual Security Policy of the IP/STS (expressed in its WSDL) MUST include the `sp:UsernameToken` assertion requiring a username and password value.

4.2 Kerberos v5 Credential

When the Identity Provider requires a *Kerberos v5 service ticket* for the IP/STS as the credential type, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential xmlns:ic="...">
  <ic:KerberosV5Credential />
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:KerberosV5Credential

This element indicates that a Kerberos v5 credential is needed.

To enable the Service Requester to obtain a Kerberos v5 service ticket for the IP/STS, the endpoint reference of the IP/STS in the Information Card or in the metadata retrieved from it MUST include a "service principal name" identity claim (i.e. a `wsai:Spn` element) under the `wsai:Identity` tag as defined in Section 12.

Furthermore, the actual Security Policy of the IP/STS (expressed in its WSDL) MUST include the `sp:KerberosToken` assertion requiring a Kerberos service ticket.

4.3 X.509v3 Certificate Credential

When the Identity Provider requires an *X.509 v3 certificate* for the user as the credential type, where the certificate and keys are in a hardware-based smart card or a software-based certificate, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential xmlns:ic="..." xmlns:ds="..." xmlns:wsse="...">
  <ic:DisplayCredentialHint> xs:string </ic:DisplayCredentialHint>
  <ic:X509V3Credential>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
    <ds:X509Data>
      <wsse:KeyIdentifier
        ValueType="http://docs.oasisopen.org/wss/oasiswss-soap-
messagesecurity-1.1#ThumbPrintSHA1"
        EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis200401-wss-
soap-message-security-1.0#Base64Binary">
        xs:base64binary
      </wsse:KeyIdentifier>
    </ds:X509Data>
  </ic:X509V3Credential>
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:DisplayCredentialHint

This optional element provides a user hint string which can be used to prompt the user, for example, to insert the appropriate smart card into the reader.

.../ic:X509V3Credential

This element indicates that a X.509 certificate credential is needed.

.../ic:X509V3Credential/ds:X509Data/wsse:KeyIdentifier

This element provides a key identifier for the X.509 certificate based on the SHA1 hash of the entire certificate content expressed as a "thumbprint." Note that the extensibility point in the `ds:X509Data` element is used to add `wsse:KeyIdentifier` as a child element.

Furthermore, the actual Security Policy of the IP/STS, expressed in its WSDL, MUST include the `sp:X509Token` assertion requiring an X.509v3 certificate.

4.4 Self-issued Token Credential

When the Identity Provider requires a *self-issued token* as the credential type, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential xmlns:ic="...">
  <ic:SelfIssuedCredential>
    <ic:PrivatePersonalIdentifier>
      xs:base64Binary
    </ic:PrivatePersonalIdentifier>
  </ic:SelfIssuedCredential>
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:SelfIssuedCredential

This element indicates that a self-issued token credential is needed.

.../ic:SelfIssuedCredential/ic:PrivatePersonalIdentifier

This required element provides the value of the PPID claim asserted in the self-issued token used previously to register with the IP/STS (see Section 7.5.14).

Furthermore, the actual Security Policy of the IP/STS (expressed in its WSDL) MUST include the `sp:IssuedToken` assertion requiring a self-issued token with exactly one claim, namely, the PPID.

5 Faults

In addition to the standard faults described in WS-Addressing, WS-Security and WS-Trust, this profile defines the following additional faults that MAY occur when interacting with an RP or an IP. The binding of the fault properties (listed below) to a SOAP 1.1 or SOAP 1.2 fault message is described in [WS-

Copyright © OASIS® 2008-2009. All Rights Reserved.

1372 Addressing]. If the optional **[Detail]** property for a fault includes any specified content, then the
1373 corresponding schema fragment is included in the listing below.

1374 5.1 Relying Party

1375 The following faults MAY occur when submitting Security Tokens to an RP per its Security Policy.

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	*S:Sender*
[Subcode]	ic:RequiredClaimMissing
[Reason]	A required claim is missing from the Security Token.
[Detail]	[URI of missing claim] `<ic:ClaimType Uri="[Claim URI]" />`

1376

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:InvalidClaimValue
[Reason]	A claim value asserted in the Security Token is invalid.
[Detail]	[URI of invalid claim] `<ic:ClaimType Uri="[Claim URI]" />`

1377 5.2 Identity Provider

1378 The following faults MAY occur when requesting Security Tokens from an IP using Information Cards.

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:MissingAppliesTo
[Reason]	The request is missing Relying Party identity information.
[Detail]	(None defined.)

1379

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:InvalidProofKey
[Reason]	Invalid proof key specified in request.
[Detail]	(None defined.)

1380

Copyright © OASIS® 2008-2009. All Rights Reserved.

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:UnknownInformationCardReference
[Reason]	Unknown Information Card reference specified in request.
[Detail]	[Unknown Information Card reference] `<ic:InformationCardReference>` `<ic:CardId>[card ID]</ic:CardId>` `<ic:CardVersion>[version]</ic:CardVersion>` `</ic:InformationCardReference>`

1381

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:FailedRequiredClaims
[Reason]	Could not satisfy required claims in request; construction of token failed
[Detail]	[URIs of claims that could not be satisfied] `<ic:ClaimType Uri="[Claim URI]" />` `...`

1382

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:InformationCardRefreshRequired
[Reason]	Stale Information Card reference specified in request; Information Card SHOULD be refreshed
[Detail]	[Information Card reference that needs refreshing] `<ic:InformationCardReference>` `<ic:CardId>[card ID]</ic:CardId>` `<ic:CardVersion>[version]</ic:CardVersion>` `</ic:InformationCardReference>`

1383 ### 5.2.1 Identity Provider Custom Error Messages

1384 Identity Providers MAY return custom error messages to Identity Selectors via SOAP faults that can be
1385 displayed by the Identity Selector user interface. The error message MUST be communicated as an
1386 `S:Text` element within the `S:Reason` element of a SOAP fault message. Multiple `S:Text` elements
1387 MAY be returned with different `xml:lang` values and the Identity Selector SHOULD use the one
1388 matching the user's locale, if possible.

1389 *Example:*

```
1390 <s:Envelope xmlns:wsa="http://www.w3.org/2005/08/addressing"
1391     xmlns:s="http://www.w3.org/2003/05/soap-envelope">
1392   <s:Header>
1393     <wsa:Action s:mustUnderstand="1">
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
      http://www.w3.org/2005/08/addressing/soap/fault
    </wsa:Action>
  </s:Header>
  <s:Body>
    <s:Fault>
      <s:Code>
        <s:Value>s:Sender</s:Value>
      </s:Code>
      <s:Reason>
        <s:Text xml:lang="en">Message in English ...</</s:Text>
        <s:Text xml:lang="es-ES">Message in the Spanish of Spain ...</s:Text>
      </s:Reason>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

6 Information Cards Transfer Format

This section defines how collections of Information Cards are transferred between Identity Selectors. The cards collection is always transferred after encrypting it with a key derived from a user specified password. Section 6.1 describes the transfer format of the collection in the clear, whereas Section 6.1.2 describes the transfer format after the necessary encryption is applied.

6.1 Pre-Encryption Transfer Format

Each Information Card in the transfer stream will contain metadata and key material maintained by the originating Identity Selector in addition to the original Information Card metadata. If an Identity Selector includes a co-resident Self-issued Identity Provider (described in Section 7), an exported self-issued card MAY also contain any associated claims information.

The XML schema used for the transfer format is defined below:

Syntax:

```
<ic:RoamingStore xmlns:ic="...">
  <ic:RoamingInformationCard> +
    <ic:InformationCardMetaData>
      [Information Card]
      <ic:IsSelfIssued> xs:boolean </ic:IsSelfIssued>
      <ic:PinDigest> xs:base64Binary </ic:PinDigest> ?
      <ic:HashSalt> xs:base64Binary </ic:HashSalt>
      <ic:TimeLastUpdated> xs:dateTime </ic:TimeLastUpdated>
      <ic:IssuerId> xs:base64Binary </ic:IssuerId>
      <ic:IssuerName> xs:string </ic:IssuerName>
      <ic:BackgroundColor> xs:int </ic:BackgroundColor>
    </ic:InformationCardMetaData>
    <ic:InformationCardPrivateData> ?
      <ic:MasterKey> xs:base64Binary </ic:MasterKey>
      <ic:ClaimValueList> ?
        <ic:ClaimValue Uri="xs:anyURI" ...> +
          <ic:Value> xs:string </ic:Value>
        </ic:ClaimValue>
      </ic:ClaimValueList>
    </ic:InformationCardPrivateData>
    ...
  </ic:RoamingInformationCard>
  ...
</ic:RoamingStore>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:RoamingStore

The collection of Information Cards selected for transfer.

Copyright © OASIS® 2008-2009. All Rights Reserved.

/ic:RoamingStore/ic:RoamingInformationCard (one or more)

An individual Information Card within the transfer stream.

For brevity, the prefix string "/ic:RoamingStore/ic:RoamingInformationCard" in the element names below is shortened to "...".

.../ic:InformationCardMetaData

This required element contains the metadata for an Information Card.

.../ic:InformationCardMetaData/[Information Card]

The original content of the Information Card as issued by the Identity Provider (described in Section 3.1.1).

.../ic:InformationCardMetaData/ic:IsSelfIssued

This required element indicates if the card is self-issued ("true") or not ("false").

.../ic:InformationCardMetaData/ic:PinDigest

This optional element contains a digest of the user-specified PIN information if the card is PIN-protected. The digest contains the base64 encoded bytes of the SHA1 hash of the bytes of the user-specified PIN represented using Unicode encoding UTF-16LE with no byte order mark. Usage of other algorithms is not described.

.../ic:InformationCardMetaData/ic:HashSalt

This optional element contains a random per-card entropy value used for computing the Relying Party specific PPID claim when the card is used at a Relying Party and for computing the Client Pseudonym PPID value sent an Identity Provider.

.../ic:InformationCardMetaData/ic:TimeLastUpdated

This required element contains the date and time when the card was last updated.

.../ic:InformationCardMetaData/ic:IssuerId

This required element contains an identifier for the Identity Provider with which a self-issued credential descriptor in a card issued by that Identity Provider can be resolved to the correct self-issued card. The element content SHOULD be the empty string for self-issued cards.

.../ic:InformationCardMetaData/ic:IssuerName

This required element contains a friendly name of the card issuer.

.../ic:InformationCardMetaData/ic:BackgroundColor

This required element contains the background color used to display the card image. This value is a 3-byte RGB color value in the sRGB color space used by HTML.

.../ic:InformationCardMetaData/{any}

This is an extensibility point to allow additional metadata to be included.

.../ic:InformationCardPrivateData

This required element contains the private data for an Information Card.

.../ic:InformationCardPrivateData/ic:MasterKey

This required element contains a base64 encoded 256-bit random number that provides a "secret key" for the Information Card. This key is used for computing the Relying Party specific PPID claim when the card is used at a Relying Party and for computing the Client Pseudonym PPID value sent to an Identity Provider. This element is present both for self-issued and managed Information Cards.

.../ic:InformationCardPrivateData/ic:ClaimValueList

This optional element is a container for the set of claim types and their corresponding values embodied by a self-issued card.

Copyright © OASIS® 2008-2009. All Rights Reserved.

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue (one or more)

This required element is a container for an individual claim, *i.e.*, a claim type and its corresponding value.

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue/@Uri

This required attribute contains a URI that identifies the specific claim type.

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue/ic:Value

This required element contains the value for an individual claim type.

…/@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore/ic:RoamingInformationCard` element when updating information using the Information Cards Transfer Format.

…/{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore/ic:RoamingInformationCard` element when updating information using the Information Cards Transfer Format.

/ic:RoamingStore/@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore` element when updating information using the Information Cards Transfer Format.

/ic:RoamingStore/{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore` element when updating information using the Information Cards Transfer Format.

6.1.1 PIN Protected Card

When an Information Card is PIN protected, in addition to storing a digest of the PIN in the card data, the master key and claim values associated with the card MUST also be encrypted with a key derived from the user-specified PIN.

It is RECOMMENDED that the PKCS-5 based key derivation method be used with the input parameters summarized in the table below for deriving the encryption key from the PIN. Usage of other algorithms is not described.

Copyright © OASIS® 2008-2009. All Rights Reserved.

Key derivation method	PBKDF1 per [RFC 2898] (Section 5.1)
Input parameters:	
Password	UTF-8 encoded octets of PIN
Salt	16-byte random number (actual value stored along with master key)
Iteration count	1000 (actual value stored along with master key)
Key length	32 octets
Hash function	SHA-256

The encryption method and the corresponding parameters that MUST be used are summarized in the table below.

Encryption method	AES-256
Parameters:	
Padding	As per PKCS-7 standard
Mode	CBC
Block size	16 bytes (as REQUIRED by AES)

In a PIN-protected card, the encrypted content of the master key and the claim value fields are described below.

.../ic:InformationCardPrivateData/ic:MasterKey

This element MUST contain a base64 encoded byte array comprised of the encryption parameters and the encrypted master key serialized as per the binary structure summarized in the table below.

Field	*Offset*	*Size (bytes)*
Version (for internal use)	0	1
Salt used for key-derivation method	1	16
Iteration count used for key-derivation method	17	4
Initialization Vector (IV) used for encryption	21	16
Encrypted master key	37	master key length

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue/ic:Value

This element MUST contain a base64 encoded byte array comprised of the encrypted claim value. The encryption parameters used are taken from those serialized into the master key field and summarized in the table above.

6.1.2 Computing the ic:IssuerId

The `ic:IssuerId` value used for a card when representing it in the Information Cards Transfer Format SHOULD be computed as a function of the `ds:KeyInfo` field of the envelope digitally signed by the Identity Provider. Specifically:

- Compute *IP PPID Seed* in the same manner as *RP PPID Seed* in Section 7.6.1, except that the certificate from `ds:KeyInfo` is used, rather than the Relying Party's.

Copyright © OASIS® 2008-2009. All Rights Reserved.

Use the *IP PPID Seed* as the `ic:IssuerId` value.

The `ic:IssuerId` value SHOULD be the empty string for self-issued cards.

6.1.3 Computing the ic:IssuerName

The `ic:IssuerName` value used for a card when representing it in the Information Cards Transfer Format SHOULD be computed as a function of the `ds:KeyInfo` field of the envelope digitally signed by the Identity Provider. Specifically, if the certificate from `ds:KeyInfo` is an extended validation (EV) certificate [EV Cert], then set `ic:IssuerName` to the organizationName (O) field value from the certificate, otherwise set `ic:IssuerName` to the commonName (CN) field value from the certificate.

6.1.4 Creating the ic:HashSalt

A random `ic:HashSalt` value for a card SHOULD be created by the Identity Selector when that card is created from the `ic:InformationCard` data provided by an Identity Provider.

6.2 Post-Encryption Transfer Format

The transfer stream MUST be encrypted with a key derived from a user specified password. The XML schema used for the encrypted transfer stream is defined below:

Syntax:

```
Byte-order-mark
<?xml version="1.0" encoding="utf-8"?>
<ic:EncryptedStore xmlns:ic="..." xmlns:xenc="...">
  <ic:StoreSalt> xs:base64Binary </ic:StoreSalt>
  <xenc:EncryptedData>
    <xenc:CipherData>
      <xenc:CipherValue> ... </xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>
</ic:EncryptedStore>
...
```

The following describes the elements listed in the XML schema outlined above:

Byte-order-mark

The first three bytes in the stream containing the values {0xEF, 0xBB, 0xBF} constitutes a “byte order mark”.

/ic:EncryptedStore

The top-level container element for the encrypted transfer stream.

/ic:EncryptedStore/ic:StoreSalt

This required element contains the random salt used as a parameter for the key derivation function to derive the encryption key from a user-specified password.

/ic:EncryptedStore/xenc:EncryptedData/xenc:CipherData/xenc:CipherValue

This element contains a base64 encoded byte array containing the ciphertext corresponding to the clear text transfer stream described in Section 6.1.

@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them when updating information using the Information Cards Transfer Format.

Copyright © OASIS® 2008-2009. All Rights Reserved.

{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them when updating information using the Information Cards Transfer Format.

The remainder of this section describes the element content of the *xenc:CipherValue* element in the schema outline above. Specifically, it describes the encryption method used and the format of the encrypted content.

The following table defines two symbolic constants, namely *EncryptionKeySalt* and *IntegrityKeySalt*, and their corresponding values used by the key derivation and the encryption methods described below to encrypt the transfer stream.

EncryptionKeySalt	{ 0xd9, 0x59, 0x7b, 0x26, 0x1e, 0xd8, 0xb3, 0x44, 0x93, 0x23, 0xb3, 0x96, 0x85, 0xde, 0x95, 0xfc }
IntegrityKeySalt	{ 0xc4, 0x01, 0x7b, 0xf1, 0x6b, 0xad, 0x2f, 0x42, 0xaf, 0xf4, 0x97, 0x7d, 0x4, 0x68, 0x3, 0xdb }

The transfer stream content is encrypted with a key derived from a user-specified password. It is RECOMMENDED that the PKCS-5 based key derivation method be used with the input parameters summarized in the table below for deriving the key from the password. Usage of other algorithms is not described.

Key derivation method	PBKDF1 per [RFC 2898] (Section 5.1)
Input parameters:	
Password	UTF-8 encoded octets of user-specified password
Salt	16-byte random number (actual value stored in the *ic:StoreSalt* field)
Iteration count	1000
Key length	32 octets
Hash function	SHA-256

The PKCS-5 key derived as per the preceding table MUST be further hashed with a 16-byte salt using the SHA256 hash function, and the resulting value used as the encryption key. The order in which the values used MUST be hashed is as follows:

Encryption Key = SHA256 (EncryptionKeySalt + PKCS5-derived-key)

Further, to provide an additional integrity check at the time of import, a “hashed integrity code” MUST be computed as follows and included along with the encrypted transfer stream content.

- The PKCS-5 key derived as per the preceding table MUST be further hashed with a 16-byte salt using the SHA256 hash function, and the resulting value used as the integrity key. The order in which the values used MUST be hashed is as follows:

  Integrity Key = SHA256 (IntegrityKeySalt + PKCS5-derived-key)

- The last block of the clear text transfer stream MUST be captured and further hashed with the *integrity key (IK)* and the *initialization vector (IV)* using the SHA256 hash function, and the resulting value used as the hashed integrity code. The order in which the values used MUST be hashed is as follows:

Copyright © OASIS® 2008-2009. All Rights Reserved.

Hashed Integrity Code = SHA256 (IV + IK + Last-block-of-clear-text)

The encryption method and the corresponding parameters that MUST be used to encrypt the transfer stream are summarized in the table below.

Encryption method	AES-256
Parameters:	
Padding	As per PKCS-7 standard
Mode	CBC
Block size	16 bytes (as REQUIRED by AES)

The element content of `xenc:CipherValue` MUST be a base64 encoded byte array comprised of the initialization vector used for encryption, the hashed integrity code (as described above), and the encrypted transfer stream. It MUST be serialized as per the binary structure summarized in the table below.

Field	*Offset*	*Size (bytes)*
Initialization Vector (IV) used for encryption	0	16
Hashed integrity code	16	32
Ciphertext of transfer stream	48	Arbitrary

7 Simple Identity Provider Profile

A simple Identity Provider, called the “Self-issued Identity Provider” (SIP), is one which allows users to self-assert identity in the form of self-issued tokens. An Identity Selector MAY include a co-resident Self-issued Identity Provider that conforms to the Simple Identity Provider Profile defined in this section. This profile allows self-issued identities created within one Identity Selector to be used in another Identity Selector such that users do not have to reregister at a Relying Party when switching Identity Selectors. Because of the co-location there is data and metadata specific to an Identity Provider that need to be shareable between Identity Selectors.

7.1 Self-Issued Information Card

The `ic:Issuer` element within an Information Card provides a logical name for the issuer of the Information Card. An Information Card issued by a SIP (*i.e.*, a self-issued Information Card) MUST use the special URI below as the value of the `ic:Issuer` element in the Information Card.

URI:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self
```

7.2 Self-Issued Token Characteristics

The self-issued tokens issued by a SIP MUST have the following characteristics:

- The token type of the issued token MUST be SAML 1.1 which MUST be identified by either of the following token type URIs:
  - *urn:oasis:names:tc:SAML:1.0:assertion*, or
  - *http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1*.
- It is RECOMMENDED that the signature key used in the issued token be a 2048-bit asymmetric RSA key which identifies the issuer. Usage of other algorithms is not described.

Copyright © OASIS® 2008-2009. All Rights Reserved.

- The issuer of the token, indicated by the value of the `saml:Issuer` attribute on the `saml:Assertion` root element, MUST be identified by the following URI defined in Section 2.1.1 representing the issuer "self".

  `http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self`

- The issued token MUST contain the `saml:Conditions` element specifying:
  - the token validity interval using the `NotBefore` and `NotOnOrAfter` attributes, and
  - the `saml:AudienceRestrictionCondition` element restricting the token to a specific target scope (i.e., a specific recipient of the token).
- The `saml:NameIdentifier` element SHOULD NOT be used to specify the Subject of the token.
- The subject confirmation method MUST be specified as one of:
  - *urn:oasis:names:tc:SAML:1.0:cm:holder-of-key*, or
  - *urn:oasis:names:tc:SAML:1.0:cm:bearer* (for Browser based applications).
- When the subject confirmation method is "holder of key", the subject confirmation key (also referred to as the *proof key*) MUST be included in the token in the `ds:KeyInfo` child element under the `saml:SubjectConfirmation` element. The proof key MUST be encoded in the token as follows:
  - For *symmetric* key tokens, the proof key is encrypted to the recipient of the token in the form of a `xenc:EncryptedKey` child element. It is RECOMMENDED that an AES key with a default size of 256 bits be used, but a different size MAY be specified by the Relying Party. Usage of other algorithms is not described.
  - For *asymmetric* key tokens, it is RECOMMENDED that the proof key be a public RSA key value specified as a `ds:RSAKeyValue` child element under the `ds:KeyValue` element. The default size of the key is 2048 bits. Usage of other algorithms is not described.
- The issued token MUST contain a single attribute statement (i.e., a single `saml:AttributeStatement` element) containing the subject confirmation data and the requested claims (called *attributes* in a SAML token).
- The claim types supported by the self-issued token SHOULD include those listed in Section7.5.
- The claims asserted in the `saml:AttributeStatement` element of the issued token MUST be named as follows using the claim type definitions in the XML schema file referenced in Section7.5. For each claim represented by a `saml:Attribute` element,
  - the `AttributeName` attribute is set to the NCname of the corresponding claim type defined in the XML schema file, and
  - the `AttributeNamespace` attribute is set to the target namespace of the XML schema file, namely

    `http://schemas.xmlsoap.org/ws/2005/05/identity/claims`

It is RECOMMENDED that the XML digital signature [XMLDSIG] profile used to sign a self-issued token be as follows. Usage of other algorithms is not described.

- Uses the *enveloped signature* format identified by the transform algorithm identifier "*http://www.w3.org/2000/09/xmldsig#enveloped-signature*". The token signature contains a single `ds:Reference` containing a URI reference to the `AssertionID` attribute value of the root element of the SAML token.

Copyright © OASIS® 2008-2009. All Rights Reserved.

- Uses the RSA signature method identified by the algorithm identifier "*http://www.w3.org/2000/09/xmldsig#rsa-sha1*".
- Uses the exclusive canonicalization method identified by the algorithm identifier **"***http://www.w3.org/2001/10/xml-exc-c14n#*" for canonicalizing the token content as well as the signature content.
- Uses the SHA1 digest method identified by the algorithm identifier "*http://www.w3.org/2000/09/xmldsig#sha1*" for digesting the token content being signed.
- No other transforms, other than the ones listed above, are used in the enveloped signature.
- The `ds:KeyInfo` element is always present in the signature carrying the signing RSA public key in the form of a `ds:RSAKeyValue` child element.

Following is an example of a self-issued signed Security Token containing three claims.

Example:

```
<Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
    AssertionID="urn:uuid:08301dba-d8d5-462f-85db-dec08c5e4e17"
    Issuer="http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self"
    IssueInstant="2004-10-06T16:44:20.00Z"
    MajorVersion="1" MinorVersion="1">
  <Conditions NotBefore="2004-10-06T16:44:20.00Z"
    NotOnOrAfter="2004-10-06T16:49:20.00Z">
    <AudienceRestrictionCondition>
      <Audience>http://www.relying-party.com</Audience>
    </AudienceRestrictionCondition>
  </Conditions>
  <AttributeStatement>
    <Subject>
      <!-- Content here differs; see examples that follow -->
    </Subject>
    <Attribute AttributeName="privatepersonalidentifier"
AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <AttributeValue>
        f8301dba-d8d5a904-462f0027-85dbdec0
      </AttributeValue>
    </Attribute>
    <Attribute AttributeName="givenname"
AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <AttributeValue>dasf</AttributeValue>
    </Attribute>
    <Attribute AttributeName="emailaddress"
AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <AttributeValue>dasf@mail.com</AttributeValue>
    </Attribute>
  </AttributeStatement>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="urn:uuid:08301dba-d8d5-462f-85db-dec08c5e4e17">
        <Transforms>
          <Transform
            Algorithm="http://.../2000/09/xmldsig#enveloped-signature"/>
          <Transform
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        </Transforms>
        <DigestMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>vpnIyEi4R/S4b+1vEH4gwQ9iHsY=</DigestValue>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
      </Reference>
    </SignedInfo>
    <SignatureValue>...</SignatureValue>
    <!-- token signing key -->
    <KeyInfo>
      <KeyValue>
        <RSAKeyValue>
          <Modulus>... utnQyEi8R/S4b+1vEH4gwR9ihsV ...</Modulus>
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
</Assertion>
```

The content of the `saml:Subject` element in the self-issued token differs based on the subject confirmation method and the type of proof key used. The following examples illustrate each of the three variations of the content of this element.

The following example illustrates the content of the `saml:Subject` element when subject confirmation method is “holder of key” using a symmetric proof key.

Example:

```
<Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:ds="..."
    xmlns:wsse="..." xmlns:xenc="...">
  <SubjectConfirmation>
    <ConfirmationMethod>
      urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
    </ConfirmationMethod>
    <ds:KeyInfo>
      <!-- symmetric proof key encrypted to recipient -->
      <xenc:EncryptedKey>
        <xenc:EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
        <ds:KeyInfo>
          <ds:X509Data>
            <wsse:KeyIdentifier
              ValueType="http://docs.oasis-open.org/wss/2004/xx/oasis-2004xx-
wss-soap-message-security-1.1#ThumbprintSHA1">
                EdFoIaAeja85201XTzjNMVWy7532jUYtrx=
            </wsse:KeyIdentifier>
          </ds:X509Data>
        </ds:KeyInfo>
        <xenc:CipherData>
          <xenc:CipherValue>
            AuFhiu72+1kaJiAuFhiu72+1kaJi=
          </xenc:CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedKey>
    </ds:KeyInfo>
  </SubjectConfirmation>
</Subject>
```

The following example illustrates the content of the `saml:Subject` element when subject confirmation method is “holder of key” using an asymmetric proof key.

Example:

```
<Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:ds="...">
  <SubjectConfirmation>
    <ConfirmationMethod>
      urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
    </ConfirmationMethod>
    <ds:KeyInfo>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
      <!-- asymmetric RSA public key as proof key -->
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>... FntQyKi6R/E4b+1vDH4gwS5ihsU ...</ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
  </SubjectConfirmation>
</Subject>
```

The following example illustrates the content of the `saml:Subject` element when subject confirmation method is “bearer” using no proof key.

Example:

```
<Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
  <SubjectConfirmation>
    <ConfirmationMethod>
      urn:oasis:names:tc:SAML:1.0:cm:bearer
    </ConfirmationMethod>
  </SubjectConfirmation>
</Subject>
```

7.3 Self-Issued Token Encryption

One of the goals of the Information Card Model is to ensure that any claims are exposed only to the Relying Party intended by the user. For this reason, the SIP SHOULD encrypt the self-issued token under the key of the Relying Party. This guarantees that a token intended for one Relying Party cannot be decoded by nor be meaningful to another Relying Party. As described in Section 8.3, when the Relying Party is not identified by a certificate, because no key is available for the Relying Party in this case, the token can not be encrypted, but SHOULD still be signed.

When a self-issued token is encrypted, the XML encryption [XMLENC] standard MUST be used. The encryption construct MUST use encrypting the self-issued token with a randomly generated symmetric key which in turn is encrypted to the Relying Party’s public key taken from its X.509 v3 certificate. The encrypted symmetric key MUST be placed in an `xenc:EncryptedKey` element within the `xenc:EncryptedData` element carrying the encrypted Security Token.

It is RECOMMENDED that the XML encryption [XMLENC] profile that is used for encrypting the key and the token be as follows. Usage of other algorithms is not described.

- Uses the RSA-OAEP key wrap method identified by the algorithm identifier “*http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p*” for encrypting the encryption key.
- Uses the AES256 with CBC encryption method identified by the algorithm “*http://www.w3.org/2001/04/xmlenc#aes256-cbc*” for encrypting the token. The padding method used is as per the PKCS-7 standard in which the number of octets remaining in the last block is used as the padding octet value.
- The `ds:KeyInfo` element is present in the encrypted key specifying the encryption key information in the form of a Security Token reference.

Following is an illustration of a self-issued token encrypted to a Relying Party using the encryption structure described above.

Example:

```
<xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
    xmlns:xenc="..." xmlns:ds="..." xmlns:wsse="...">
  <xenc:EncryptionMethod
     Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc" />
  <ds:KeyInfo>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
    <xenc:EncryptedKey>
      <xenc:EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      </xenc:EncryptionMethod>
      <ds:KeyInfo>
        <wsse:SecurityTokenReference>
          <wsse:KeyIdentifier
            ValueType="http://docs.oasis-open.org/wss/2004/xx/oasis-2004xx-
wss-soap-message-security-1.1#ThumbprintSHA1"
            EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis200401-
wss-soap-message-security-1.0#Base64Binary">
            +PYbznDaB/dlhjIfqCQ458E72wA=
          </wsse:KeyIdentifier>
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>...Ukasdj8257Fjwf=</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedKey>
  </ds:KeyInfo>
  <xenc:CipherData>
    <!-- Start encrypted Content
    <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
        AssertionID="urn:uuid:08301dba-d8d5-462f-85db-dec08c5e4e17" ...>
      ...
    </Assertion>
    End encrypted content -->
    <xenc:CipherValue>...aKlh4817JerpZoDofy90=</xenc:CipherValue>
  </xenc:CipherData>
</xenc:EncryptedData>
```

7.4 Self-Issued Token Signing Key

The key used to sign a self-issued token presented to a Relying Party also represents a unique identifier for the Subject of the token. In order to prevent the key from becoming a correlation identifier across relying parties, a SIP SHOULD use a different key to sign a self-issued token for each Relying Party where the card is used. In other words, the key used to sign the self-issued token is pair-wise unique for a given Information Card and RP combination. To allow self-issued identities created by a SIP within one Identity Selector to be used in another, the signing keys used by the two SIPs SHOULD be the same.

It is RECOMMENDED that the signing key be an RSA key. Usage of other algorithms is not described.

This section specifies the “processing rules” that SHOULD be used by a SIP to derive the RSA key used to sign the self-issued token for a combination of an Information Card and an RP where the card is used. Each self-issued Information Card contains a 256-bit secret random number, called the “master key” (see Section 6.1), that is used as the secret entropy in deriving the token signing RSA key. (Managed Information Cards also have a master key that is used in the Client Pseudonym PPID calculation, as per Section 3.3.4.1.)

Key derivation is done according to the ANSI X9.31 standard for key generation which starts with requiring the use of six random values denoted by X_{p1}, X_{p2}, X_{q1}, X_{q2}, X_p, and X_q. The processing rules described here enunciate how to transform the master key in an Information Card into the six random inputs for the X9.31 key generation process. The actual key computation algorithm in the X9.31 standard is *not* reproduced here.

The values X_p and X_q are REQUIRED to be at least 512 bits and each independently carries the full entropy of any Information Card master key of up to 512 bits in length. The values X_{p1}, X_{p2}, X_{q1}, and X_{q2} have a length of only 100 to 121 bits and therefore will be shorter than the Information Card master key and hence cannot each independently carry the full master key entropy. The details of the X9.31 protocol,

Copyright © OASIS® 2008-2009. All Rights Reserved.

however, ensure that for reasonably sized master keys, full entropy will be achieved in the generated asymmetric key pair.

7.4.1 Processing Rules

This key generation mechanism can be used to generate 1024 or 2048-bit RSA keys.

Notation: If H is an *n*-bit big-endian value, the convention H[1..p] denotes bits *1* through *p* in the value of H where $p \leq n$, and bit-1 is the rightmost (least significant) bit whereas bit-n is the leftmost (most significant) bit in the value of H. Also, the convention X + Y denotes the concatenation of the big-endian bit value of X followed by the big-endian bit value of Y.

Assume that the master key for the selected Information Card (see Section 6.1) is M and the unique *RP Identifier* (derived as per Section 7.6.1) is T. The following processing rules SHOULD be used to derive the inputs for the X9.31 key generation process.

1. Define 32-bit DWORD constants C_n as follows:

   $C_n = n$, where $n = 0,1,2,...,15$

2. Compute SHA-1 hash values H_n as follows:

   If the requested key size = 1024 bits, compute

   $H_n = SHA1 (M + T + C_n)$ for $n = 0,1,2,...,9$

   If the requested key size = 2048 bits, compute

   $H_n = SHA1 (M + T + C_n)$ for $n = 0,1,2,...,15$

3. Extract the random input parameters for the X9.31 protocol as follows:

   For all key sizes, compute

   X_{p1} [112-bits long] = $H_0[1..112]$

   X_{p2} [112-bits long] = $H_1[1..112]$

   X_{q1} [112-bits long] = $H_2[1..112]$

   X_{q2} [112-bits long] = $H_3[1..112]$

   If the requested key size = 1024 bits, compute

   X_p [512-bits long] = $H_4[1..160] + H_5[1..160] + H_6[1..160] + H_0[129..160]$

   X_q [512-bits long] = $H_7[1..160] + H_8[1..160] + H_9[1..160] + H_1[129..160]$

   If the requested key size = 2048 bits, compute

   X_p [1024-bits long] = $H_4[1..160] + H_5[1..160] + H_6[1..160] + H_0[129..160] +$
   $H_{10}[1..160] + H_{11}[1..160] + H_{12}[1..160] + H_2[129..160]$

   X_q [1024-bits long] = $H_7[1..160] + H_8[1..160] + H_9[1..160] + H_1[129..160] +$
   $H_{13}[1..160] + H_{14}[1..160] + H_{15}[1..160] + H_3[129..160]$

4. The X9.31 specification (Section 4.1.2) requires that the input values X_{p1}, X_{p2}, X_{q1}, X_{q2} MUST satisfy the following conditions.

   - The large prime factors p_1, p_2, q_1, and q_2 are the first primes greater than their respective random X_{p1}, X_{p2}, X_{q1}, X_{q2} input values. They are randomly selected from the set of prime numbers between 2^{100} and 2^{120}, and each SHALL pass at least 27 iterations of Miller-Rabin.

Copyright © OASIS® 2008-2009. All Rights Reserved.

To ensure that the lower bound of 2^{100} is met, set the 101^{th} bit of X_{p1}, X_{p2}, X_{q1}, X_{q2} to '1' (*i.e.* X_{p1}[13^{th} byte] |= 0x10, X_{p2}[13^{th} byte] |= 0x10, X_{q1}[13^{th} byte] |= 0x10, X_{q2}[13^{th} byte] |= 0x10).

5. The X9.31 specification (Section 4.1.2) requires that the input values X_p and X_q MUST satisfy the following conditions.
   - If the requested key size = 1024 bits, then
     $X_p \geq (\sqrt{2})(2^{511})$ and $X_q \geq (\sqrt{2})(2^{511})$
   - If the requested key size = 2048 bits, then
     $X_p \geq (\sqrt{2})(2^{1023})$ and $X_q \geq (\sqrt{2})(2^{1023})$

   To ensure this condition is met, set the two most significant bits of X_p and X_q to '1' (*i.e.* X_p[most significant byte] |= 0xC0, X_q[most significant byte] |= 0xC0).
6. Compute 1024 or 2048-bit keys as per the X9.31 protocol using $\{X_{p1}, X_{p2}, X_{q1}, X_{q2}, X_p, X_q\}$ as the random input parameters.
7. Use a 32-bit DWORD size *public exponent* value of 65537 for the generated RSA keys.

There are three conditions as follows in the X9.31 specification which, if not met, require that one or more of the input parameters MUST be regenerated.

- (Section 4.1.2 of X9.31) $|X_p - X_q| \geq 2^{412}$ (for 1024-bit keys) or $|X_p - X_q| \geq 2^{924}$ (for 2048-bit keys). If not true, X_q MUST be regenerated and q recomputed.
- (Section 4.1.2 of X9.31) $|p - q| \geq 2^{412}$ (for 1024-bit keys) or $|p - q| \geq 2^{924}$ (for 2048-bit keys). If not true, X_q MUST be regenerated and q recomputed.
- (Section 4.1.3 of X9.31) $d > 2^{512}$ (for 1024-bit keys) or $d > 2^{1024}$ (for 2048-bit keys). If not true, X_{q1}, X_{q2}, and X_q MUST be regenerated and key generation process repeated.

When it is necessary to regenerate an input parameter as necessitated by one or more of the conditions above, it is essential that the regeneration of the input parameter be deterministic to guarantee that all implementations of the key generation mechanism will produce the same results. Furthermore, input regeneration is a potentially unlimited process. In other words, it is possible that regeneration MUST be performed more than once. In theory, one MAY need to regenerate input parameters many times before a key that meets all of the requirements can be generated.

The following processing rules MUST be used for regenerating an input parameter *X* of length *n-bits* when necessary:

a. Pad the input parameter *X* on the right, assuming a big-endian representation, with *m* zero-bits where *m* is the smallest number which satisfies *((n+m) mod 128 = 0)*.

b. Encrypt the padded value with the AES-128 (**E**lectronic **C**ode **B**ook mode) algorithm using the 16-byte constant below as the encryption key:

Encryption Key	{ 0x8b, 0xe5, 0x61, 0xf5, 0xbc, 0x3e, 0x0c, 0x4e, 0x94, 0x0d, 0x0a, 0x6d, 0xdc, 0x21, 0x9d, 0xfd }

c. Use the leftmost *n-bits* of the result above as the REQUIRED regenerated parameter.

If a regenerated parameter does not satisfy the necessary conditions, then repeat the 3-step process above (call it *RegenFunction*) to generate the parameter again by using the output of one iteration as input for the next iteration. In other words, if the output of the i^{th} iteration of the regeneration function above for an input parameter *X* is given by X_i then

$$X_{i+1} = RegenFunction\ (X_i)$$

Copyright © OASIS® 2008-2009. All Rights Reserved.

7.5 Claim Types

This section specifies a set of claim (attribute) types and the corresponding URIs that is defined by this profile for some commonly used personal information. These claim types MAY be used by a SIP, in self-issued tokens, or by other Identity Providers. Note that, wherever possible, the claims included here reuse and refer to the attribute semantics defined in other established industry standards that deal with personal information. A SIP SHOULD support these claim types at a minimum. Other Identity Providers MAY also support these claim types when appropriate. The URIs defined here MAY be used by a Relying Party to specify requested claims in its policy.

The base XML namespace URI that is used by the claim types defined here is as follows:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/claims
```

For convenience, an XML Schema for the claim types defined here can be found at:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/claims.xsd
```

7.5.1 First Name

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname*

Type: *xs:string*

Definition: (*givenName* in [RFC 2256]) Preferred name or first name of a Subject. According to RFC 2256: “This attribute is used to hold the part of a person’s name which is not their surname nor middle name.”

7.5.2 Last Name

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname*

Type: *xs:string*

Definition: (*sn* in [RFC 2256]) Surname or family name of a Subject. According to RFC 2256: “This is the X.500 surname attribute which contains the family name of a person.”

7.5.3 Email Address

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress*

Type: *xs:string*

Definition: (*mail* in inetOrgPerson) Preferred address for the “To:” field of email to be sent to the Subject, usually of the form <user>@<domain>. According to inetOrgPerson using [RFC 1274]: “This attribute type specifies an electronic mailbox attribute following the syntax specified in RFC 822.”

7.5.4 Street Address

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/streetaddress*

Type: *xs:string*

Definition: (*street* in [RFC 2256]) Street address component of a Subject’s address information. According to RFC 2256: “This attribute contains the physical address of the object to which the entry corresponds, such as an address for package delivery.” Its content is arbitrary, but typically given as a PO Box number or apartment/house number followed by a street name, *e.g.* 303 Mulberry St.

7.5.5 Locality Name or City

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/locality*

Type: *xs:string*

Copyright © OASIS® 2008-2009. All Rights Reserved.

Definition: (*l* in [RFC 2256]) Locality component of a Subject's address information. According to RFC 2256: "This attribute contains the name of a locality, such as a city, county or other geographic region." *e.g.* Redmond.

7.5.6 State or Province

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/stateorprovince*

Type: *xs:string*

Definition: (*st* in [RFC 2256]) Abbreviation for state or province name of a Subject's address information. According to RFC 2256: "This attribute contains the full name of a state or province. The values SHOULD be coordinated on a national level and if well-known shortcuts exist - like the two-letter state abbreviations in the US – these abbreviations are preferred over longer full names." *e.g.* WA.

7.5.7 Postal Code

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/postalcode*

Type: *xs:string*

Definition: (*postalCode* in X.500) Postal code or zip code component of a Subject's address information. According to X.500(2001): "The postal code attribute type specifies the postal code of the named object. If this attribute value is present, it will be part of the object's postal address - zip code in USA, postal code for other countries."

7.5.8 Country

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/country*

Type: *xs:string*

Definition: (*c* in [RFC 2256]) Country of a Subject. According to RFC 2256: "This attribute contains a two-letter ISO 3166 country code."

7.5.9 Primary or Home Telephone Number

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/homephone*

Type: *xs:string*

Definition: (*homePhone* in inetOrgPerson) Primary or home telephone number of a Subject. According to inetOrgPerson using [RFC 1274]: "This attribute type specifies a home telephone number associated with a person." Attribute values SHOULD follow the agreed format for international telephone numbers, *e.g.* +44 71 123 4567.

7.5.10 Secondary or Work Telephone Number

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/otherphone*

Type: *xs:string*

Definition: (*telephoneNumber* in X.500 Person) Secondary or work telephone number of a Subject. According to X.500(2001): "This attribute type specifies an office/campus telephone number associated with a person." Attribute values SHOULD follow the agreed format for international telephone numbers, *e.g.* +44 71 123 4567.

7.5.11 Mobile Telephone Number

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/mobilephone*

Type: *xs:string*

Definition: (*mobile* in inetOrgPerson) Mobile telephone number of a Subject. According to inetOrgPerson using [RFC 1274]: "This attribute type specifies a mobile telephone number associated

Copyright © OASIS® 2008-2009. All Rights Reserved.

with a person." Attribute values SHOULD follow the agreed format for international telephone numbers, *e.g.* +44 71 123 4567.

7.5.12 Date of Birth

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/dateofbirth*

Type: *xs:date*

Definition: The date of birth of a Subject in a form allowed by the *xs:date* data type.

7.5.13 Gender

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/gender*

Type: *xs:token*

Definition: Gender of a Subject that can have any of these exact string values – '0' (meaning unspecified), '1' (meaning Male) or '2' (meaning Female). Using these values allows them to be language neutral.

7.5.14 Private Personal Identifier

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/privatepersonalidentifier*

Type: *xs:base64binary*

Definition: A private personal identifier (PPID) that identifies the Subject to a Relying Party. The word "private" is used in the sense that the Subject identifier is specific to a given Relying Party and hence private to that Relying Party. A Subject's PPID at one Relying Party cannot be correlated with the Subject's PPID at another Relying Party. Typically, the PPID SHOULD be generated by an Identity Provider as a pair-wise pseudonym for a Subject for a given Relying Party. For a self-issued Information Card, the Self-issued Identity Provider in an Identity Selector system SHOULD generate a PPID for each Relying Party as a function of the card identifier and the Relying Party's identity. The processing rules and encoding of the PPID claim value is specified in Section 7.6.

Compatibility Note: Some existing Identity Selectors omit listing the PPID claim as an `ic:SupportedClaimType` from the `ic:SupportedClaimTypeList` when saving a self-issued Information Card in the Information Cards Transfer Format defined in Section 6.1, even though the PPID claim is supported by the card. This behavior is deprecated, as all supported claims SHOULD be listed. Nonetheless, Identity Selectors MAY choose to recognize this case and support the PPID claim for self-issued cards not explicitly listing this claim.

7.5.15 Web Page

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/webpage*

Type: *xs:string*

Definition: The Web page of a Subject expressed as a URL.

7.6 The PPID Claim

The PPID claim for a Subject user represents a unique identifier for that user at a given Relying Party that is different from all identifiers for that user at any other Relying Party. In other words, the PPID is a pair-wise unique identifier for a given user identity and Relying Party combination. Since an Information Card represents a specific user identity and a Relying Party is the organization behind a Web service or site that the user interacts with, the PPID claim is logically a function of an Information Card and the organizational identity of the Relying Party.

This section describes the processing rules that SHOULD be used by a SIP to derive a PPID claim value for a combination of an Information Card and a Relying Party where it is used.

Copyright © OASIS® 2008-2009. All Rights Reserved.

7.6.1 Relying Party Identifier and Relying Party PPID Seed

In order to derive the PPID and Signing Key as functions of the RP's organizational identity, a stable and unique identifier for the RP, called the *RP Identifier*, is needed. In the Information Card Model, the identity of a Relying Party (RP) possessing an X.509v3 certificate is presented in the form of that certificate. Therefore the organizational identity of the RP is obtained by applying a series of transformations to the identity information carried in the X.509 certificate. (See Section 8 for the specification of how to compute these values for Relying Parties not possessing a certificate.)

As specified in [RFC 2459], the subject field inside an X.509 certificate identifies the entity associated with the public key stored in the subject public key field. Where it is non-empty, the subject field MUST contain an X.500 distinguished name (DN). The DN MUST be unique for each subject entity certified by the one CA as defined by the issuer name field.

The subject field contains a DN of the form shown below:

CN=*string*, [OU=*string*, ...,] O=*string*, L=*string*, S=*string*, C=*string*

The Object Identifiers for these attributes from the DN are as follows:

Field Abbreviation	*Field Name*	*Object Identifier*
O	organizationName	2.5.4.10
L	localityName	2.5.4.7
S	stateOrProvinceName	2.5.4.8
C	countryName	2.5.4.6
CN	commonName	2.5.4.3

Note that the field names and abbreviations used in this specification may not correspond to those used by particular software but the underlying Object Identifiers (OIDs) of the attributes are unambiguous.

For an end-entity certificate, the values of the attribute types O (organizationName), L (localityName), S (stateOrProvinceName) and C (countryName) together uniquely identify the organization to which the end-entity identified by the certificate belongs. These attribute types are collectively referred to as the *organizational identifier attributes* here. The *RP Identifier* is constructed using these organizational identifier attributes as described below.

The *RP Identifier* value is used as an input to the Signing Key computation. A closely related value called the Relying Party PPID Seed is also computed, which is used as an input to the PPID claim and Client Pseudonym PPID computations. In many cases these are the same but in one case they differ.

There are four cases of how the *RP Identifier* and *RP PPID Seed* are constructed depending on which organizational identifier attributes the RP's certificate contains, if it is an extended validation (EV) certificate [EV Cert] with respect to the organizational identifier attributes, and if it chains to a trusted root certificate.

Case 1: RP's certificate *is* EV for organizational identifier attributes and chains to a trusted root certificate authority

- Convert the organizational identifier attributes in the end-entity certificate into a string, call it *OrgIdString*, of the following form:

  |O="*string*"|L="*string*"|S="*string*"|C="*string*"|

  The vertical bar character (ASCII 0x7C) is used as a delimiter at the start and end of the string as well as between the attribute types. Further, the string values of the individual attribute types are enclosed within double quote characters (ASCII 0x22). If an attribute type is absent in the subject field of the end-entity certificate, then the corresponding string value is the empty string (""). Following is an example *OrgIdString* per this convention.

Copyright © OASIS® 2008-2009. All Rights Reserved.

2147 |O="Microsoft"|L="Redmond"|S="Washington"|C="US"|

2148 • Encode all the characters in *OrgIdString* into a sequence of bytes, call it *OrgIdBytes*, using
2149 Unicode encoding UTF-16LE with no byte order mark.

2150 • Hash *OrgIdBytes* using the SHA256 hash function, and use the resulting value as the *RP*
2151 *Identifier* and *RP PPID Seed*.

2152 *RP PPID Seed = RP Identifier = SHA256 (OrgIdBytes)*

2153 **Case 2: RP's certificate *is not* EV for organizational identifier attributes, has a non-empty**
2154 **organizationName (O) value, and chains to a trusted root certificate authority**

2155 • Convert the organizational identifier attributes in the end-entity certificate into a string, call it
2156 *OrgIdString*, in the same manner as employed for Case 1 above.

2157 • Let *QualifierString* be the string:

2158 |Non-EV

2159 • Let *QualifiedOrgIdString* be the concatenation of *QualifierString* and *OrgIdString*.

2160 *QualifiedOrgIdString = QualifierString + OrgIdString*

2161 • Encode all the characters in *QualifiedOrgIdString* into a sequence of bytes, call it
2162 *QualifiedOrgIdBytes*, using Unicode encoding UTF-16LE with no byte order mark.

2163 • Hash *QualifiedOrgIdBytes* using the SHA256 hash function, and use the resulting value as the
2164 *RP Identifier*.

2165 *RP Identifier = SHA256 (QualifiedOrgIdBytes)*

2166 • Encode all the characters in *OrgIdString* into a sequence of bytes, call it *OrgIdBytes*, using
2167 Unicode encoding UTF-16LE with no byte order mark.

2168 • Hash *OrgIdBytes* using the SHA256 hash function, and use the resulting value as the *Relying*
2169 *Party PPID Seed*.

2170 *RP PPID Seed = SHA256 (OrgIdBytes)*

2171 **Case 3: RP's certificate has an empty or no organizationName (O) value and has an empty or no**
2172 **commonName (CN) or does not chain to a trusted root certificate authority**

2173 • Take the subject public key in the end-entity certificate, call it *PublicKey*, as a byte array.

2174 • Hash *PublicKey* using the SHA256 hash function, and use the resulting value as the *RP Identifier*
2175 and *RP PPID Seed*.

2176 *RP PPID Seed = RP Identifier = SHA256 (PublicKey)*

2177 **Case 4: RP's certificate has an empty or no organizationName (O) value but has a non-empty**
2178 **commonName (CN) value and chains to a trusted root certificate authority**

2179 • Convert the commonName attribute value in the end-entity certificate into a string, call it
2180 *CnIdString*, of the following form:

2181 |CN="*string*"|

2182 Following is an example *CnIdString* per this convention:

2183 |CN="login.live.com"|

2184 • Encode all the characters in *CnIdString* into a sequence of bytes, call it *CnIdBytes*, using Unicode
2185 encoding UTF-16LE with no byte order mark.

2186 • Hash *CnIdBytes* using the SHA256 hash function, and use the resulting value as the *RP Identifier*
2187 and *RP PPID Seed*.

2188 *RP PPID Seed = RP Identifier = SHA256 (CnIdBytes)*

Copyright © OASIS® 2008-2009. All Rights Reserved.

7.6.2 PPID

The PPID value SHOULD be produced as follows using the card identifier and the *RP PPID Seed* (specified in Section 7.6.1):

- Encode the value of the `ic:CardId` element of the Information Card into a sequence of bytes, call it *CardIdBytes*, using Unicode encoding UTF-16LE with no byte order mark.
- Hash *CardIdBytes* using the SHA256 hash function to obtain the canonical card identifier *CanonicalCardId*.

  CanonicalCardId = SHA256 (CardIdBytes)
- Hash the concatenation of *RP PPID Seed* and *CanonicalCardId* using the SHA256 hash function to obtain the PPID.

  PPID = SHA256 (RP PPID Seed + CanonicalCardId)

7.6.3 Friendly Identifier

The PPID provides an RP-specific identifier for a Subject that is suitable for programmatic processing, but is not a user-friendly identifier. The simple transformation rules specified in this section MAY be used by a SIP, or any other Identity Provider supporting the PPID claim, to create a friendly identifier for use within a Display Token accompanying a Security Token carrying the PPID claim.

The Friendly Identifier has the following characteristics:

- It is encoded as a 10-character alphanumeric string of the form “AAA-AAAA-AAA” grouped into three groups separated by the ‘hyphen’ character (*e.g.*, the string “6QR-97A4-WR5”). Note that the hyphens are used for punctuation only.
- The encoding alphabet does NOT use the numbers ‘0’ and ‘1’, and the letters ‘O’ and ‘I’ to avoid confusion stemming from the similar glyphs used for these numbers and characters. This leaves 8 digits and 24 letters – a total of 32 alphanumeric symbols – as the alphabet for the encoding.

The processing rules used for deriving a Friendly Identifier from a PPID are as follows:

- The PPID value is conveyed as a base64 encoded string inside tokens. Start with the base64 decoded PPID value as input.
- Hash the PPID value using the SHA1 hash function to obtain a hashed identifier.

  HashId = SHA1 (PPID)
- Let the Friendly Identifier be the string “$A_0\ A_1\ A_2$– $A_3\ A_4\ A_5\ A_6$– $A_7\ A_8\ A_9$” where each A_i is an alphanumeric character from the encoding alphabet described above.
- For *i := 0 to 9*, each A_i is determined as below:
  - Take the i^{th} octet of *HashId* (denoted as *HashId[i]*)
  - Find *RawValue = HashId[i] % 32* (where % is the remainder operation)
  - A_i = *EncodedSymbol* obtained by mapping *RawValue* to *EncodedSymbol* using the table below

Raw Value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Encoded Symbol	Q	L	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Copyright © OASIS® 2008-2009. All Rights Reserved.

Raw Value	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Encoded Symbol	G	H	J	K	M	N	P	R	S	T	U	V	W	X	Y	Z

8 Relying Parties without Certificates

While Relying Parties are typically identified by presenting a cryptographically protected identity, such as an X.509v3 certificate, the Information Card Model is also applicable in situations in which no Relying Party certificate is available. This section specifies how Information Cards are used at Relying Parties with no certificate: specifically, Web sites using the [HTTP] scheme. Also see `ic07:RequireStrongRecipientIdentity` in Section 3.1.1.7 for a means whereby card issuers can prohibit the use of cards at Relying Parties not identified by a certificate.

8.1 Relying Party Identifier and Relying Party PPID Seed

The Relying Party Identifier and Relying Party PPID Seed values for Relying Parties without certificates are computed in this manner:

- Set the string *OrgIdString* to be the fully qualified DNS host name in lowercase characters specified in the URI of the Relying Party, or if a numeric IP address was used, then a string representation of the IP address of the server. For IPv4 addresses, this string is the standard 4-byte dotted decimal representation of the address with no leading zeros, such as `131.107.55.210`. For IPv6 addresses, this string is the hexadecimal representation of the address in eight groups of four hex digits each using uppercase for the letters, with each group of four digits separated by a colon, all enclosed by square brackets, such as `[0000:1234:0000:0000:0000:000A:00BC:0DEF]`.
- Encode all the characters in *OrgIdString* into a sequence of bytes, call it *OrgIdBytes*, using the Unicode encoding UTF-16LE with no byte order mark.
- Hash *OrgIdBytes* using the SHA256 hash function, and use the resulting value as both the *RP Identifier* and the *RP PPID Seed*.

The *RP Identifier* and *RP PPID Seed* are then used in the same manner as for Relying Parties identified by certificates when computing PPID claim and Client Pseudonym PPID values.

8.2 AppliesTo Information

Under the circumstances described in Section 3.3.3 that the RP endpoint to which the token will be sent is supplied as the `wsp:AppliesTo` value to the IP, when the RP possesses no certificate, the URL of the RP is supplied as that `wsp:AppliesTo` value.

Example:

```
<wst:RequestSecurityToken xmlns:wst="..." xmlns:wsp="..." xmlns:wsa="...">
  <wsp:AppliesTo>
    <wsa:EndpointReference>
      <wsa:Address>http://login.contoso.com</wsa:Address>
    </wsa:EndpointReference>
  </wsp:AppliesTo>
  ...
</wst:RequestSecurityToken>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

8.3 Token Signing and Encryption

When the Relying Party is not identified by a certificate, tokens sent from the Self-issued Identity Provider are not encrypted, although they are still signed in the manner described in Section 7.2. Tokens generated by Identity Providers for Relying Parties not identified by a certificate are also typically not encrypted, as no encryption key is available. However, the token MAY still be encrypted if the Identity Provider has a pre-existing relationship with the Relying Party and they have mutually agreed on the use of a known encryption key. The token SHOULD still typically be signed, even when not encrypted.

9 Using WS-SecurityPolicy 1.2 and WS-Trust 1.3

Software implementing the Information Card Model SHOULD utilize the OASIS standard versions of WS-SecurityPolicy and WS-Trust – [WS-SecurityPolicy 1.2] and [WS-Trust 1.3] and MAY utilize the previous draft versions – [WS-SecurityPolicy 1.1] and [WS-Trust 1.2]. This section describes the differences between the old and standard versions of these protocols that MAY affect software implementing the Information Card Model.

9.1 Overview of Differences

The following changes between the protocol versions affect software implementing this specification:

- **Namespace changes:**
  http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702 replaces
  http://schemas.xmlsoap.org/ws/2005/07/securitypolicy.
  http://docs.oasis-open.org/ws-sx/ws-trust/200512 replaces
  http://schemas.xmlsoap.org/ws/2005/02/trust.
- **Use of RequestSecurityTokenResponseCollection:** A `wst:RequestSecurityTokenResponseCollection` element encloses the `wst:RequestSecurityTokenResponse` when WS-Trust 1.3 is used.
- **Use of SecondaryParameters:** An Identity Selector sends some information received from the Relying Party to the Identity Provider in a `wst:SecondaryParameters` element.
- **Bearer Token Request Syntax:** The new `wst:KeyType` value http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer is used to request a bearer token.

9.2 Identity Selector Differences

Identity Selectors MUST determine the WS-Trust versions used by Identity Provider STSs and Relying Party STSs using their Security Policy.

Identity Selectors supporting WS-Trust 1.3 MUST understand the new WS-Trust 1.3 elements and syntax such as `wst13:RequestSecurityTokenResponseCollection` and new URIs such as http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer. They MUST also understand that typical properties of an RST like Claims and KeyType MAY be either a direct child of the top level `wst13:RequestSecurityToken` element or contained within a `wst13:SecondaryParameters` element in the RST.

When constructing an RST for an Identity Provider using WS-Trust 1.3, the Identity Selector SHOULD send parameters received from the Relying Party in a `wst13:SecondaryParameters` element within the `wst13:RequestSecurityToken`, with these exceptions:

- The user chooses not to send optional claims. In this scenario, no SecondaryParameters element is sent in order to hide this user decision.
- No `wsp:AppliesTo` is being sent in the RST. In this scenario, no `wst13:SecondaryParameters` element is sent so that the Identity Provider does not obtain any identifying information about the Relying Party.

Copyright © OASIS® 2008-2009. All Rights Reserved.

Example:

```
<wst13:RequestSecurityToken Context="ProcessRequestSecurityToken"
    xmlns:wst13="..." xmlns:ic="...">
  <wst13:RequestType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue</wst13:RequestType>
  <ic:InformationCardReference>
  ...
  </ic:InformationCardReference>
  <wst13:Claims Dialect="http://schemas.xmlsoap.org/ws/2005/05/identity">
  ...
  </wst13:Claims>
  <wst13:KeyType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/SymmetricKey</wst13:KeyType>
  <wst13:SecondaryParameters>
    <wst13:RequestType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue</wst13:RequestType>
    <wst13:TokenType>urn:oasis:names:tc:SAML:1.0:assertion</wst13:TokenType>
    <wst13:KeyType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/SymmetricKey</wst13:KeyType>
    <wst13:KeyWrapAlgorithm>http://www.w3.org/2001/04/xmlenc#rsa-oaep-
mgf1p</wst13:KeyWrapAlgorithm>
    ...
  </wst13:SecondaryParameters>
</wst13:RequestSecurityToken>
```

The `wst13:RequestSecurityTokenResponse` constructed MUST be enclosed within a `wst13:RequestSecurityTokenResponseCollection` element.

Example:

```
<wst13:RequestSecurityTokenResponseCollection xmlns:wst13="...">
  <wst13:RequestSecurityTokenResponse>
    <wst13:TokenType>urn:oasis:names:tc:SAML:1.0:assertion</wst13:TokenType>
    <wst13:RequestedSecurityToken> ... </wst13:RequestedSecurityToken>
    ...
  </wst13:RequestSecurityTokenResponse>
</wst13:RequestSecurityTokenResponseCollection>
```

9.3 Security Token Service Differences

To utilize WS-Trust 1.3, an Identity Provider STS and Relying Party STSs MUST express their Security Policy using WS-SecurityPolicy 1.2.

STSs using WS-Trust 1.3 MUST understand the new WS-Trust 1.3 elements and syntax such as `wst13:RequestSecurityTokenResponseCollection` and new URIs such as http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer. They MUST also understand that typical properties of an RST like Claims and KeyType MAY be either a direct child of the top level `wst13:RequestSecurityToken` element or contained within a `wst13:SecondaryParameters` element in the RST.

Copyright © OASIS® 2008-2009. All Rights Reserved.

10 Browser Behavior with Information Cards

This section explains the steps that a Web browser takes when using an Information Card to authenticate to a Web site. Two cases are described. The basic case is where the Web site provides all the Relying Party functionality via HTML extensions transported over HTTPS. The second case is where the Relying Party employs a Relying Party Security Token Service (STS), which it references via HTML extensions transported over HTTPS.

10.1 Basic Protocol Flow when using an Information Card at a Web Site

This section explains the protocol flow when using an Information Card to authenticate at a Web site where no Relying Party STS is employed.

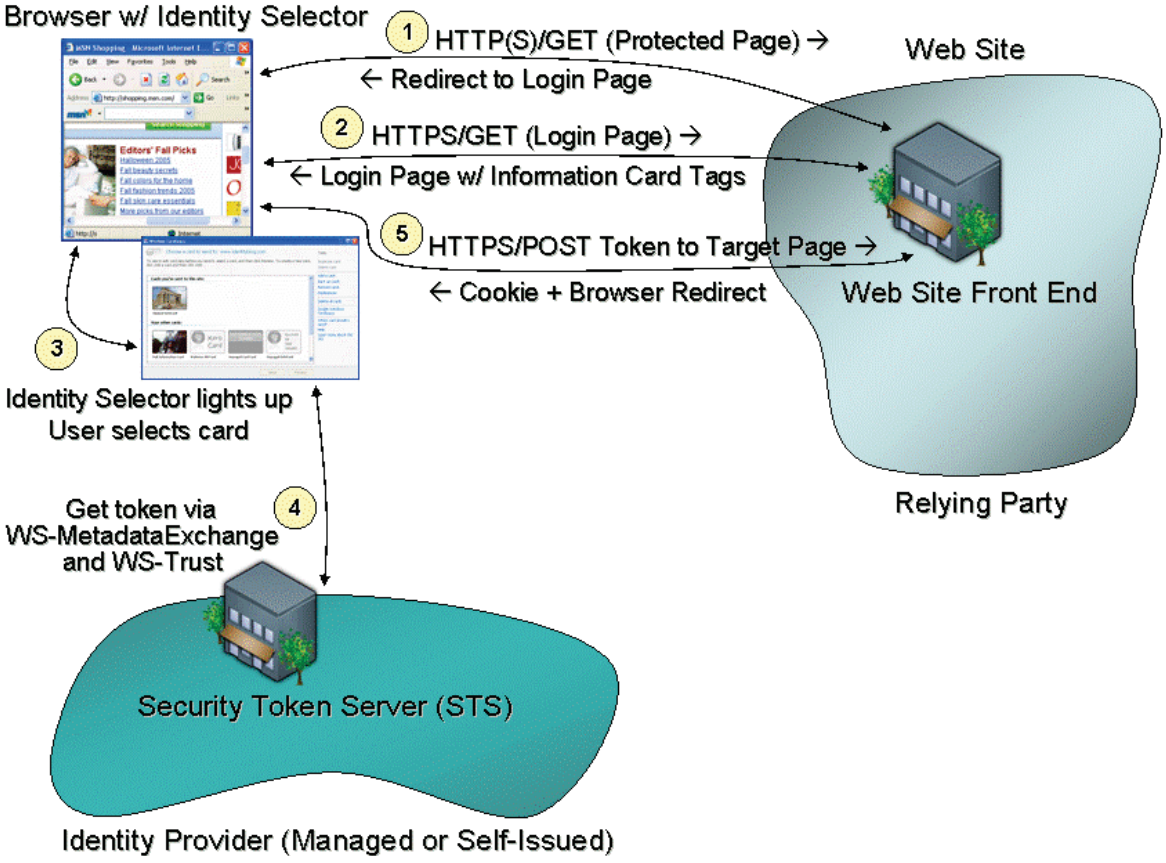


Figure 1. Basic protocol flow when using an Information Card to authenticate at a Web site

Figure 1 gives an example of the basic protocol flow when an Information Card is used to authenticate at a Web site that employs no Relying Party STS. Steps 1, 2, and 5 are essentially the same as a typical forms-based login today: (1) The user navigates to a protected page that requires authentication. (2) The site redirects the browser to a login page, which presents a Web form. (5) The browser posts the Web form that includes the login credentials supplied by the user back to the login page. The site then validates the contents of the form including the user credentials, typically writes a client-side browser cookie to the client for the protected page domain, and redirects the browser back to the protected page.

Copyright © OASIS® 2008-2009. All Rights Reserved.

The key difference between this scenario and today's site login scenarios is that the login page returned to the browser in step (2) contains an HTML tag that allows the user to choose to use an Information Card to authenticate to the site. When the user selects this tag, the browser invokes an Identity Selector, which implements the Information Card user experience and protocols, and triggers steps (3) through (5).

In Step (3), the browser Information Card support code invokes the Identity Selector, passing it parameter values supplied by the Information Card HTML tag supplied by the site in Step (2). The user then uses the Identity Selector to choose an Information Card, which represents a Digital Identity that can be used to authenticate at that site. Step (4) retrieves a Security Token that represents the Digital Identity selected by the user from the STS at the Identity Provider for that identity.

In Step (5), the browser posts the token obtained back to the Web site using a HTTPS/POST. The Web site validates the token, completing the user's Information Card-based authentication to the Web site. Following authentication, the Web site typically then writes a client-side browser cookie and redirects the browser back to the protected page.

It is worth noting that this cookie is likely to be *exactly the same cookie* as the site would have written back had the user authenticated via other means, such as a forms-based login using username/password. This is one of the ways that the goal of "minimal impact on Web sites" is achieved. Other than its authentication subsystem, the bulk of a Web site's code can remain completely unaware that Information Card-based authentication is even utilized. It just uses the same kinds of cookies as always.

10.2 Protocol Flow with Relying Party STS

In the previous scenario, the Web site communicated with the client Identity Selector using only the HTML extensions enabling Information Card use, transported over the normal browser HTTPS channel. In this scenario, the Web site also employs a Relying Party STS to do part of the work of authenticating the user, passing the result of that authentication on to the login page via HTTPS POST.

There are several reasons that a site might factor its solution this way. One is that the same Relying Party STS can be used to do the authentication work for both browser-based applications and smart client applications that are using Web services. Second, it allows the bulk of the authentication work to be done on servers dedicated to this purpose, rather than on the Web site front-end servers. Finally, this means that the front-end servers can accept site-specific tokens, rather than the potentially more general or more complicated authentication tokens issued by the Identity Providers.

Copyright © OASIS® 2008-2009. All Rights Reserved.

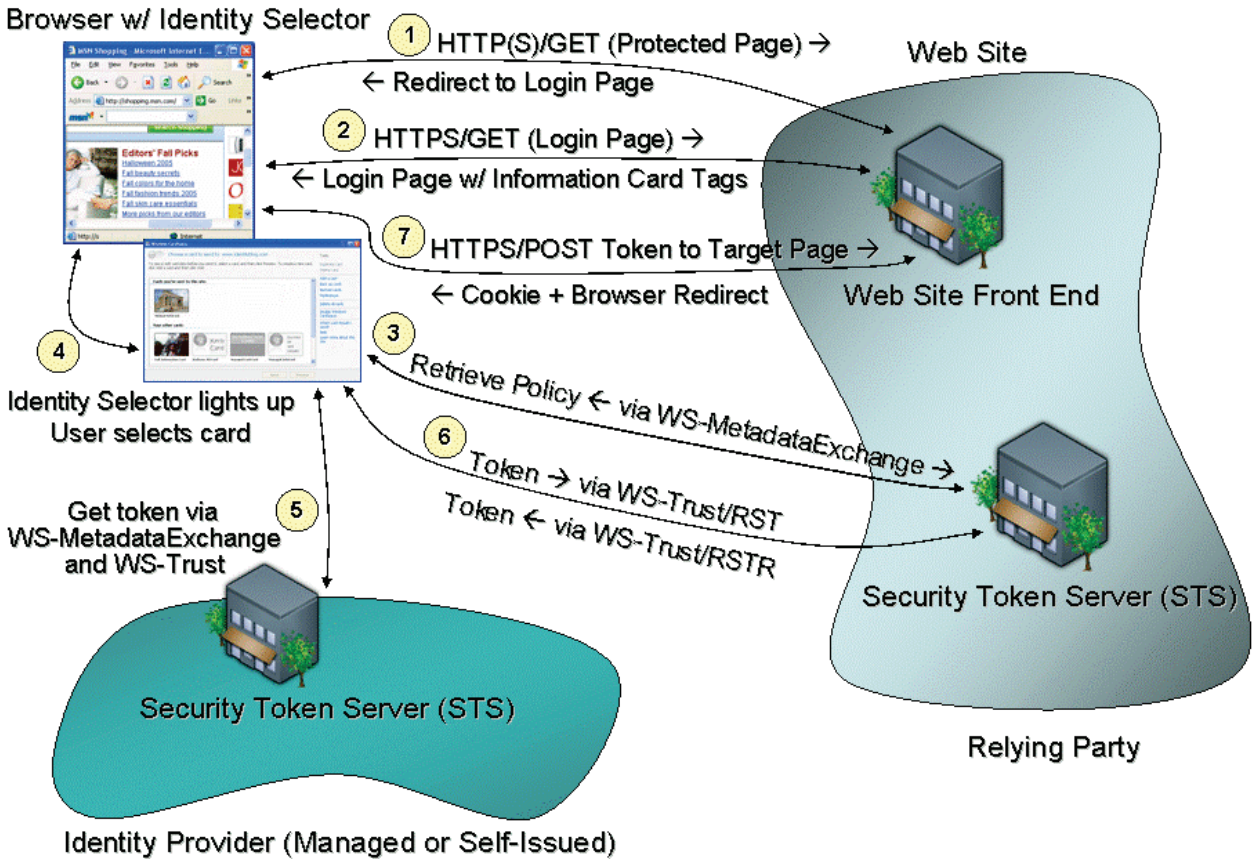


Figure 2. Protocol flow when using an Information Card to authenticate at a Web site, where the Web site employs a Relying Party STS

This scenario is similar to the previous one, with the addition of steps (3) and (6). The differences start with the Information Card information supplied to the browser by the Web site in Step (2). In the previous scenario, the site encoded its WS-SecurityPolicy information using Information Card HTML extensions and supplied them to the Information Card-extended browser directly. In this scenario, the site uses different Information Card HTML extensions in the Step (2) reply to specify which Relying Party STS SHOULD be contacted to obtain the WS-SecurityPolicy information.

In Step (3), the Identity Selector contacts the Relying Party STS specified by the Web site and obtains its WS-SecurityPolicy information via WS-MetadataExchange. In Step (4) the Identity Selector user interface is shown and the user selects an Information Card, which represents a Digital Identity to use at the site. In Step (5), the Identity Provider is contacted to obtain a Security Token for the selected Digital Identity. In Step (6), the Security Token is sent to the Web site's Relying Party STS to authenticate the user and a site-specific authentication token is returned to the Identity Selector. Finally, in Step (7), the browser posts the token obtained in Step (6) back to the Web site using HTTPS/POST. The Web site validates the token, completing the user's Information Card-based authentication to the Web site. Following authentication, the Web site typically then writes a client-side browser cookie and redirects the browser back to the protected page.

10.3 User Perspective and Examples

The Information Card user experience at Web sites is intended to be intuitive and natural enough that users' perspective on it will simply be "That's how you log in". Today, Web sites that require authentication typically ask the user to supply a username and password at login time. With Information Cards, they instead ask users to choose an Information Card. Some sites will choose to accept only

Copyright © OASIS® 2008-2009. All Rights Reserved.

Information Cards whereas others will give users the choice of Information Cards or other forms of authentication.

A site that accepts Information Cards typically has a login screen that contains button with a label such as "**Sign in with an Information Card**" or "**Log in using an Information Card**". Upon clicking this button, the user is presented with a choice of his Information Cards that are accepted at the site, and is asked to choose one. Once a card is selected and submitted to the site, the user is logged in and continues using the site, just as they would after submitting a username and password to a site.

Sites that accept both Information Cards and other forms of authentication present users with both an Information Card login choice and whatever other choices the site supports. For instance, a site login screen might display both "**Sign in with your username and password**" and "**Sign in with an Information Card**" buttons.

10.4 Browser Perspective

Very little additional support is needed from today's Web browsers to also support Information Cards. The main addition is that they MUST recognize special HTML and/or XHTML tags for invoking the Identity Selector, pass encoded parameters on to the Identity Selector on the platform, and POST back the token resulting from the user's choice of an Information Card.

10.5 Web Site Perspective

Web sites that employ Information Card-based authentication MUST support two new pieces of functionality: adding HTML or XHTML tags to their login page to request an Information Card-based login and code to log the user into the site using the POSTed credentials. In response to the Information Card-based login, the Web site typically writes the same client-side browser cookie that it would have if the login had occurred via username/password authentication or other mechanisms, and issue the same browser redirects. Thus, other than the code directly involved with user authentication, the bulk of a Web site can remain unchanged and oblivious to the site's acceptance of Information Cards as a means of authentication.

Copyright © OASIS® 2008-2009. All Rights Reserved.

11 Invoking an Identity Selector from a Web Page

11.1 Syntax Alternatives: OBJECT and XHTML tags

HTML extensions are used to signal to the browser when to invoke the Identity Selector. However, not all HTML extensions are supported by all browsers, and some commonly supported HTML extensions are disabled in browser high security configurations. For example, while the OBJECT tag is widely supported, it is also disabled by high security settings on some browsers, including Internet Explorer.

An alternative is to use an XHTML syntax that is not disabled by changing browser security settings. However, not all browsers provide full support for XHTML.

To address this situation, two HTML extension formats are specified. Browsers MAY support one or both of the extension formats.

11.1.1 OBJECT Syntax Examples

An example of the OBJECT syntax is as follows:

```
<html>
  <head>
    <title>Welcome to Fabrikam</title>
  </head>
  <body>
    <img src='fabrikam.jpg' alt="Fabrikam Logo" />
    <form name="ctl00" id="ctl00" method="post"
        action="https://www.fabrikam.com/InfoCard-Browser/Main.aspx">
      <center>
        <img src='infocard_56x39.png' alt="Information Card Icon"
            onClick='ctl00.submit()' />
        <input type="submit" name="InfoCardSignin" value="Log in"
          id="InfoCardSignin" />
      </center>
      <OBJECT type="application/x-informationCard" name="xmlToken">
        <PARAM Name="tokenType" Value="urn:oasis:names:tc:SAML:1.0:assertion">
        <PARAM Name="issuer" Value=
            "http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self">
        <PARAM Name="requiredClaims" Value=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname">
      </OBJECT>
    </form>
  </body>
</html>
```

This is an example of a page that requests that the user log in using an Information Card. The key portion of this page is the OBJECT of type "`application/x-informationCard`". Once a card is selected by the user, the resulting Security Token is included in the resulting POST as the xmlToken value of the form. Appendix A shows a sample POST resulting from using a login page similar to the preceding one. If the user cancels the authentication request, the resulting POST contains an empty xmlToken value.

Parameters of the Information Card OBJECT are used to encode the necessary WS-SecurityPolicy information in HTML. In this example, the Relying Party is requesting a SAML 1.0 token from a Self-issued Identity Provider, supplying the requested claims "`emailaddress`", "`givenname`", and "`surname`". This example uses the basic protocol described in Section 2.1 (without employing a Relying Party STS).

Copyright © OASIS® 2008-2009. All Rights Reserved.

A second example of the OBJECT syntax is as follows:

```
<html>
  <body>
    <form name="ctl01" method="post"
        action="https://www.fabrikam.com/InfoCard-Browser-STS/login.aspx"
        id="ctl01" onSubmit="fnGetCard();">
      <img src='infocard_56x39.png' alt="Information Card Icon"
          onClick='ctl01.submit()' />
      <input type="submit" name="InfoCardSignin" value="Log in"
          id="InfoCardSignin" />
      <OBJECT type="application/x-informationCard" name="xmlToken"
          ID="oCard" />
    </form>
    <script type="text/javascript">
    <!--
      function fnGetCard(){
        oCard.issuer = "http://www.fabrikam.com/sts";
        oCard.issuerPolicy = "https://www.fabrikam.com/sts/mex";
        oCard.tokenType = "urn:fabricam:custom-token-type";
      }
    //-->
    </script>
  </body>
</html>
```

This example uses the enhanced protocol described in Section 2.3, which employs a Relying Party STS. Note that in this case, the “issuer” points to a Relying Party STS. The “issuerPolicy” points to an endpoint where the Security Policy of the STS (expressed via WS-SecurityPolicy) is to be obtained using WS-MetadataExchange. Also, note that the “tokenType” parameter requests a custom token type defined by the site for its own purposes. The “tokenType” parameter could have been omitted as well, provided that the Web site is capable of understanding all token types issued by the specified STS or if the STS has prior knowledge about the token type to issue for the Web site.

The object parameters can be set in normal script code. This is equivalent to setting them using the “PARAM” declarations in the previous example.

11.1.2 XHTML Syntax Example

An example of the XHTML syntax is as follows:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity">
  <head>
    <title>Welcome to Fabrikam</title>
  </head>
  <body>
    <img src='fabrikam.jpg' alt="Fabrikam Logo" />
    <form name="ctl00" id="ctl00" method="post"
        action="https://www.fabrikam.com/InfoCard-Browser/Main.aspx">
      <ic:informationCard name='xmlToken'
          style='behavior:url(#default#informationCard)'
          issuer="http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self"
          tokenType="urn:oasis:names:tc:SAML:1.0:assertion">
        <ic:add claimType=
       "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"
            optional="false" />
        <ic:add claimType=
         "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname"
            optional="false" />
        <ic:add claimType=
         "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname"
            optional="false" />
      </ic:informationCard>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
      <center>
        <input type="submit" name="InfoCardSignin" value="Log in"
            id="InfoCardSignin" />
      </center>
    </form>
  </body>
</html>
```

11.2 Identity Selector Invocation Parameters

The parameters to the OBJECT and XHTML Information Card objects are used to encode information in HTML that is otherwise supplied as WS-SecurityPolicy information via WS-MetadataExchange when an Identity Selector is used in a Web services context.

11.2.1 issuer

This optional parameter specifies the URL of the STS from which to obtain a token. If omitted, no specific STS is requested. The special value "`http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self`" specifies that the token SHOULD come from a Self-issued Identity Provider.

11.2.2 issuerPolicy

This optional parameter specifies the URL of an endpoint from which the STS's WS-SecurityPolicy can be retrieved using WS-MetadataExchange. This endpoint MUST use HTTPS.

11.2.3 tokenType

This optional parameter specifies the type of the token to be requested from the STS as a URI. This parameter can be omitted if the STS and the Web site front-end have a mutual understanding about what token type will be provided or if the Web site is willing to accept any token type.

11.2.4 requiredClaims

This optional parameter specifies the types of claims that MUST be supplied by the identity. If omitted, there are no required claims. The value of `requiredClaims` is a space-separated list of URIs, each specifying a required claim type.

11.2.5 optionalClaims

This optional parameter specifies the types of optional claims that MAY be supplied by the identity. If omitted, there are no optional claims. The value of `optionalClaims` is a space-separated list of URIs, each specifying a claim type that can MAY be submitted.

11.2.6 privacyUrl

This optional parameter specifies the URL of the human-readable Privacy Policy of the site, if provided.

11.2.7 privacyVersion

This optional parameter specifies the Privacy Policy version. This MUST be a value greater than 0 if a privacyUrl is specified. If this value changes, the UI notifies the user and allows them review the change to the Privacy Policy.

11.3 Data Types for Use with Scripting

The object used in the Information Card HTML extensions has the following type signature, allowing it to be used by normal scripting code:

```
interface IInformationCardSigninHelper
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
{
  string issuer;              // URI specifying token issuer
  string issuerPolicy;        // MetadataExchange endpoint of issuer
  string tokenType;           // URI specifying type of token to be requested
  string [] requiredClaims;   // Array of URIs of required claim types
  string [] optionalClaims;   // Array of URIs of optional claim types
  string privacyUrl;          // URL of the Privacy Policy of the site
  string privacyVersion;      // Version number of the Privacy Policy
  boolean isInstalled;        // True when an Identity Selector is available
                              // to the browser
}
```

11.4 Detecting and Utilizing an Information Card-enabled Browser

Web sites MAY choose to detect browser and Identity Selector support for Information Cards and modify their login page contents depending upon whether Information Card support is present, and which of the OBJECT and/or XHTML syntaxes are supported by the browser and supported by the Web site. This allows Information Card capabilities to be shown when available to the user, and to be not displayed otherwise.

Detecting an Information Card-enabled browser may require detecting specific browser and Identity Selector versions and being aware of the nature of their Information Card support.

11.5 Behavior within Frames

When the object tag is specified in an embedded frame, the certificate of the frame is compared to that of the root frame. For this configuration to work, the scheme, domain, and security zone (for example https, microsoft.com, and Intranet) of the URL of the embedded frame MUST be the same as that of the root frame. If they do not match, the object tag SHOULD NOT be acted upon. This prevents a form of cross-site scripting attacks.

11.6 Invocation Using the Document Object Model (DOM)

In addition to being invokable using static HTML tags and script code, Identity Selectors can be invoked from script injected into the page using the Document Object Model [DOM]. Invocation from dynamically generated script allows the Web site's requirements to be set dynamically.

11.7 Auditing, Non-Auditing, and Auditing-Optional Cards

- **Auditing Card:** When a managed card with an `ic:RequireAppliesTo` element and no `Optional` attribute or `Optional=false` attribute is used at a Web site, the Request Security Token (RST) sent to the Identity Provider contains a `wsp:AppliesTo` element.
- **Non-Auditing Card:** When a managed card with no `ic:RequireAppliesTo` element is used at a Web site, the Request Security Token (RST) sent to the Identity Provider contains no `wsp:AppliesTo` element.
- **Auditing-Optional Card:** When a managed card with an `ic:RequireAppliesTo` element with `Optional=true` attribute is used at a Web site, the Request Security Token (RST) sent to the Identity Provider contains a `wsp:AppliesTo` element.

Copyright © OASIS® 2008-2009. All Rights Reserved.

12 Endpoint Reference wsai:Identity Property

This section adds the `wsai:Identity` property to an Endpoint Reference [WS-Addressing] and leverages extensibility of the `wsa:EndpointReferenceType` schema to include a `wsai:Identity` element as described below:

```
<wsa:EndpointReference xmlns:wsa="..." xmlns:wsai="...">
  ...
  <wsai:Identity>...identity representation...</wsai:Identity>
  ...
</wsa:EndpointReference>
```

The `wsai:Identity` element inside a `wsa:EndpointReference` can hold any of the identity representations defined in Section 12.2 below.

12.1 Default Value

If a `wsa:EndpointReference` does not contain a `wsai:Identity` element, a DNS Name representation can be assumed by extracting the hostname from the Address URI.

If the URI does not have a hostname, it does not have an implicit identity value and can not be verified by the mechanisms defined in this document.

12.2 Identity Representation

12.2.1 DNS Name

The DNS Name representation implies that the remote principal is trusted to speak for that DNS name. For instance the DNS Name representation could specify “fabrikam.com”. When challenged, the endpoint contacted MUST be able to prove its right to speak for “fabrikam.com”. The service could prove its right by proving ownership of a certificate containing a reference to fabrikam.com and signed by a trusted Certificate Authority. The following element of type `xs:string` can be used to represent a DNS Name representation within a `wsai:Identity` element.

```
<wsai:Dns xmlns:wsai="...">fabrikam.com</wsai:Dns>
```

12.2.2 Service Principal Name

The SPN representation implies that the remote principal is trusted to speak for that SPN, a mechanism common in intranet domains. Its format is <serviceClass>/<host>. For example, the SPN for a generic service running on “server1.fabrikam.com” would be “host/server1.fabrikam.com”. The client could confidentially speak to the service and verify replies back from the service by obtaining a Kerberos ticket from the realm’s domain controller. The following element of type `xs:string` can be used to represent an SPN representation within a `wsai:Identity` element.

```
<wsai:Spn xmlns:wsai="...">host/hrweb</wsai:Spn>
```

12.2.3 User Principal Name

The UPN representation implies that the remote principal is a particular user in a domain. Its format is: <user>@<domain>. For example, the UPN for a user “someone” at a domain “example.com” would be “someone@example.com”. A service could prove its UPN by providing the password for the user

Copyright © OASIS® 2008-2009. All Rights Reserved.

associated with “someone@example.com”. The following element of type `xs:string` can be used to represent a UPN representation within a `wsai:Identity` element.

```
<wsai:Upn xmlns:wsai="...">someone@example.com</wsai:Upn>
```

12.2.4 KeyInfo

This identity value is similar to the previous three, but rather than describing an attribute of the target, this mechanism describes a reference (embedded or external) to key material associated with the target. This allows confirmation of the target trust identity through encryption. These values can also be used to compare authenticated identities similar to the basic trust identity values by comparing the hash of the specified trust identity value with a hash of the authenticated identity of the service. The `ds:KeyInfo` element defined in [XMLDSIG] can be used.

```
<ds:KeyInfo xmlns:ds="...">...</ds:KeyInfo>
```

12.2.4.1 Example specifying an RSA Public Key

The PublicKey representation states the public key of the remote principal. A service could prove its ownership of the key by signing some data with the private key.

```
<wsai:Identity xmlns:wsai="..." ds:wsai="...">
  <ds:KeyInfo>
    <ds:RSAKeyValue>
      <ds:Modulus>xA7SEU+e0yQH5...</ds:Modulus>
      <ds:Exponent>AQAB</ds:Exponent>
    </ds:RSAKeyValue>
  </ds:KeyInfo>
</wsai:Identity>
```

12.2.4.2 Example specifying an X509 Certificate

This example shows a certificate of the remote principal being used as the identity value.

```
<wsai:Identity xmlns:wsai="..." xmlns:ds="...">
  <ds:KeyInfo>
    <ds:X509Data>
      <ds:X509Certificate>MIICXTCCA...</ds:X509Certificate>
    </ds:X509Data>
  </ds:KeyInfo>
</wsai:Identity>
```

12.2.5 Security Token

A security token can be an identity value representing statements about the identity of an endpoint. E.g.:

```
<wsai:Identity xmlns:wsai="..." ds:wsse="...">
      <wsse:BinarySecurityToken ValueType="...#X509v3">
            <!--base64 encoded value of the X509 certificate-->
      </wsse:BinarySecurityToken>
</wsai:Identity>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

12.2.6 Security Token Reference

Similarly to `ds:KeyInfo`, `wsse:SecurityTokenReference` element can be used within a `wsai:Identity` element to reference a token representing a collection of statements about the identity of an endpoint. E.g.:

```
<wsai:Identity xmlns:wsai="..." ds:wsse="...">
      <wsse:SecurityTokenReference>
           <wsse:KeyIdentifier ValueType="...#ThumbprintSHA1">
                <!-- thumbprint of the X509 certificate  -->
           </wsse:KeyIdentifier>
      </wsse:SecurityTokenReference>
</wsai:Identity>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

13 Security Considerations

13.1 Protection of Information Cards by Identity Selectors

It is RECOMMENDED that Identity Selectors encrypt or otherwise secure the Information Card data held by them to help protect cards from being stolen and then used by an attacker. This is particularly important for self-issued Information Cards, where possession of the unencrypted contents of a card could enable an attacker to gain access to Relying Parties accounts associated with that card.

13.2 Relying Parties Without Certificates

Because claims sent to relying parties without certificates are not encrypted, it is RECOMMENDED that sensitive claims not be released to these relying parties. Identity Providers holding sensitive user data that can be released as claim values are encouraged to issue cards containing an `ic07:RequireStrongRecipientIdentity` element to prevent transmission of sensitive claim values over an unencrypted channel.

13.3 Endpoint References

It is RECOMMENDED that Endpoint Reference elements be signed to prevent tampering.

An Endpoint Reference SHOULD NOT be accepted unless it is signed and have an associated security token to specify the signer has the right to “speak for” the endpoint. That is, the relying party SHOULD NOT use an endpoint reference unless the endpoint reference is signed and presented with sufficient credentials to pass the relying parties acceptance criteria.

It is RECOMMENDED that an endpoint reference be encrypted when it contains claims and other sensitive information.

When included in a SOAP message, endpoint references are RECOMMENDED to be protected using the mechanisms described in WS-Security [WS-Security]

Copyright © OASIS® 2008-2009. All Rights Reserved.

14 Conformance

An implementation conforms to this specification if it satisfies all of the MUST or REQUIRED level requirements defined within this specification for the portions of the specification implemented by that implementation. Furthermore, when an implementation supports functionality in which there is a RECOMMENDED algorithm or set of parameter choices, conforming implementations MUST support the RECOMMENDED algorithm and parameter choices. A SOAP Node MUST NOT use the XML namespace identifiers for this specification (listed in Section 1.2) within SOAP Envelopes unless it is compliant with this specification.

This specification references a number of other specifications. In order to comply with this specification, an implementation MUST implement the portions of referenced specifications necessary to comply with the required provisions of the portions of this specification that it implements. Additionally, the implementation of the portions of the referenced specifications that are specifically cited in this specification MUST comply with the rules for those portions as established in the referenced specification.

Additionally, normative text within this specification takes precedence over normative outlines (as described in Section 1.1), which in turn take precedence over the XML Schema [XML Schema Part 1, Part 2] and WSDL [WSDL 1.1] descriptions. That is, the normative text in this specification further constrains the schemas and/or WSDL that are part of this specification; and this specification contains further constraints on the elements defined in referenced schemas.

If an OPTIONAL message is not supported, then the implementation SHOULD Fault just as it would for any other unrecognized/unsupported message. If an OPTIONAL message is supported, then the implementation MUST satisfy all of the MUST and REQUIRED sections of the message.

Copyright © OASIS® 2008-2009. All Rights Reserved.

A. HTTPS POST Sample Contents

The contents of an HTTPS POST generated by a page like the first example in Section 4.1.1 follow:

```
POST /test/s/TokenPage.aspx HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Content-Length: 6478
Content-Type: application/x-www-form-urlencoded
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-sh
ockwave-flash, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Host: calebb-tst
Referer: https://localhost/test/s/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30)
UA-CPU: x86

InfoCardSignin=Log+in&xmlToken=%3Cenc%3AEncryptedData+Type%3D%22http%3A%2F%2F
www.w3.org%2F2001%2F04%2Fxmlenc%23Element%22+xmlns%3Aenc%3D%22http%3A%2F%2Fww
w.w3.org%2F2001%2F04%2Fxmlenc%23%22%3E%3Cenc%3AEncryptionMethod+Algorithm%3D%
22http%3A%2F%2Fwww.w3.org%2F2001%2F04%2Fxmlenc%23aes256-cbc%22+%2F%3E%3CKeyIn
fo+xmlns%3D%22http%3A%2F%2Fwww.w3.org%2F2000%2F09%2Fxmldsig%23%22%3E%3Ce%3AEn
cryptedKey+xmlns%3Ae%3D%22http%3A%2F%2Fwww.w3.org%2F2001%2F04%2Fxmlenc%23%22%
3E%3Ce%3AEncryptionMethod+Algorithm%3D%22http%3A%2F%2Fwww.w3.org%2F2001%2F04%
2Fxmlenc%23rsa-oaep-mgf1p%22%3E%3CDigestMethod+Algorithm%3D%22http%3A%2F%2Fww
w.w3.org%2F2000%2F09%2Fxmldsig%23sha1%22+%2F%3E%3C%2Fe%3AEncryptionMethod%3E%
3CKeyInfo%3E%3Co%3ASecurityTokenReference+xmlns%3Ao%3D%22http%3A%2F%2Fdocs.oa
sis-open.org%2Fwss%2F2004%2F01%2Foasis-200401-wss-wssecurity-secext-1.0.xsd%2
2%3E%3Co%3AKeyIdentifier+ValueType%3D%22http%3A%2F%2Fdocs.oasis-open.org%2Fws
s%2Foasis-wss-soap-message-security-1.1%23ThumbprintSHA1%22+EncodingType%3D%2
2http%3A%2F%2Fdocs.oasis-open.org%2Fwss%2F2004%2F01%2Foasis-200401-wss-soap-m
essage-security-1.0%23Base64Binary%22%3E%2BPYbznDaB%2FdlhjIfqCQ458E72wA%3D%3C
%2Fo%3AKeyIdentifier%3E%3C%2Fo%3ASecurityTokenReference%3E%3C%2FKeyInfo%3E%3C
e%3ACipherData%3E%3Ce%3ACipherValue%3EEq9UhAJ8C9K5l4Mr3qmgX0XnyL1ChKs2PqMj0Sk
6snw%2FIRNtXqLzmgbj2Vd3vFA4Vx1hileSTyqc1kAsskqpqBc4bMHT61w1f0NxU10HDor0DlNVcV
Dm%2FAfLcyLqEP%2Boh05B%2B5ntVIJzL8Ro3typF0eoSm3S6UnINOHIjHaVWyg%3D%3C%2Fe%3AC
ipherValue%3E%3C%2Fe%3ACipherData%3E%3C%2Fe%3AEncryptedKey%3E%3C%2FKeyInfo%3E
%3Cenc%3ACipherData%3E%3Cenc%3ACipherValue%3ErBvpZydiyDzJtzl1%2FjUFX9XAzO1mOR
q0ypPLjh%2FBagXcfZeYwWD57v4Jvn1QwGajadcDASCisazswn1skdkwgmd4IUWJpPMRH7es9zY0U
vnS4ccsakgDcmscq3pDYTrxbSBfhdvrzjDiHC2XCtowOveoHeB51C5N8UAbff18IxCNtkWO8y3wLH
VGdvwaDOSakK%2FK%2Fv1UgXIc51%2FtYvjeFGeGbbSNxo8DTqeDnAMQ%2B4Y%2B1aUGhI%2FtbSr
EyJECkDgtztcxhrumbupKO%2BogWKUTTpSt851xjOFxAMiVaPZ%2FAm8V8H3ZLsR087sX%2FJ%2Bn
bRqze%2BfbdUwimN5pNoJDdMnF%2BEDLass1dPsvhL4EXzuIp5deGBaqAIoaOMEUW7ssuh1PtwkEM
eqwlOzOhu%2FHtwP1qh3D02U59MtyQnJMD5UwIwO7sZJl6%2BPg6Zp9HHtKKUMnkguvFmhyXS4BFS
ZVxPl18i%2B0MLO1um5dejEFd4nwGO%2FmNw6yEI8DdGVjXcYOT6JhPz9rHNh9%2F%2FOj5snJfL6
j2sg0EvIYoRs%2BhT4sdHZ95tGAiwMwT6cFOXbAQZUbYTr1ZOC6XPsfL2CFwiTM3mI%2Blco4Hc%2
F7IakIA8jwAJdtnd2mGuV67ZbY1mzibM1LUApixZj59El83ixctSQbV7iyywQ4IYN2CAq%2BCLMdl
R%2BDHfgEe8O3IVaGBDUEcd2MYimEiA7Yw3NIDrC14SbLzNvU702HpVJMeYv9q6S9xIVGApSrARsw
RFXyMbkMDp5WIQaJEXon7qLcsZONpdlX9bCcmaiikdpxmCeyS638te%2FhGBLmYJSQ0stf7BhA6E0
kwDRgdwsAa88bODiWHek0vDhAN4HlXFZ%2BCxp53L9Mmvy%2FCAOI%2B9OkPL2yxS22yjWQxom%2F
yZuawsK98JHVShsIVmmbKvRM6xJwvHDSzuBAOlQKS%2FMHcFZn8vHZR4lMhm5nL3F%2B%2BumMKh0
vMuKk6JiCqG9OEj996bVIIkLzESU5Z5vT6I1Kr9Brdx8ckDElipdH3x54WVfaItHJTYU%2BsxIR1T
25fi9k%2FOc%2FMX7Q%2B6NSDs4nGqkn4rzqpez9BUWNZw7caVOrDeao85f%2FiDCGymtl0A3JaSZ
dTKfzHLGmUfSkCAlVeisdvB6R7uBw8tR%2BZlgLIGS28wppFlnUYvSK7DnPrzId%2BGfHwLfL6WA%
2FEzBMMgppb5Vi%2BauHq%2BHxpCamlkrcUkzagbwNkGV8TfafkqUvRwJbxRwNVPI%2F%2Fxs%2Fp
Lcu1dh6eKcmU00%2FNx0zNOScd9XoeEU3zsV78PgvPIBT4EDugdv4bMR6dExXvZBl%2F84b1gOMhK
ZRplF8t6EAc4LCct01ht7VOVNz25NtP27ct9QPrDJc%2FoxihT4Df6NV3l4vlTnu%2B%2BzVB%2BH
JAxNkiO9gx3uLUJM9XEZCDzZKihaBk2y%2F3RhsJpABVneUd%2B3sCRbQXhgKYNBHZyRAUGpMDLhL
qpjoF9x%2FNvUujQ5DBLJafxxzNVshG52jRz%2BikhCNhJDDbeA5MQ8Q7QsYcKDC0DBFsewtWaA%2
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
FsKxl3JU6hyTotnFS%2FoS2EzbOSvn25qZuBERsZ3w%2B5WMkRzfQadyIYOSv2Df1YoljubDKy1l9
St%2FbCIBgXbVIZKYtQ%2BLyepxxFjrN7cWo2aYFnB6YLurg4USJwhXzcGcvA3%2BR5dRT6Fr37U6
OcHc%2Fz2MaZmn1cQWiDGNxHtRVxEvirBc1x47hWfSRjrKzf3orL5LzgMlYc7Iwclw2rbeWljCqOb
oV3d71ez%2FvNz1pxEMi4w8yUAQL8p%2FRCZ%2BpzvsgORu4RWKWiSwbl7AN0J3jiWShyZgDmxd2O
DDYffXjNiuH1mQWnDTkJX1ig88mqjhOYJEal0W6L0ErwrRIy29tOiAvXZANC8kA1HexulH0e38x8E
IOaVaJtNz9mqrnmnp4GdZ38txV%2BCUeWHOZaHLF4xkdtRxMAu%2FbzQ03YmUOhgxqkTfNzV6Ymne
v2nv5VsyQGJaQsNjb0M4yOe6kX2qNTwKBN2%2Bp%2Fz3f15i8KuGCgBcfP%2BP9xBizBeo7FbFtyo
2pfFhzBPmZeSOJ6kEbF1yQKHYQAT5iZ4SyTIfqqmwGxsQpWMstx3qJF8aW8WFzU1qXcC1LmgClg19
rx9NYFaQshX4f729B9Ue5MX7gTrMgwAnlXty9BsoP7nzGbr3HSXy8pR%2BimuAFW3c2NaQSbjSH5Z
FOr7PZdLHsNVJzFIsaufAwr0CAEtvlPJUt7%2B%2FE5MQsMsVqMoXFmefgdxbvY1Ue6MX1wtuJYY1
PAX7MHTyRUR3RfJDO054EoflVTwNE1fmocUXUh5rtFFuzy2T%2F2Y6pLAARXzo8uslAuH67VkuXv%
2BEMc7e3ogbf5%2BROsgJirZS6qkcYpfEUwqHiQYLnSIP4bt%2BWI5j1bxs7yzcSCkNZ2rd%2FHWr
A41AyGMfYzqxfGcrOaxHsds3JUcByB5Zw17W58GBC32Iusqa69BFTPagEapM0Fb5CbTqXnWTNNB5J
t40BVZvLv3u5oy%2BBRaMKXZhwnbT2WUTp0Ebsn17xvte52B%2BLMlSWJn96Nl5thd%2Ft1D7PlWA
sUvpJAd0UHPizCkY8VIhcXTrsSyEwer2J2I9TQTUosmssFjoP8Lx9qMfXo0eGVmneV8kVBtu4J7N1
QmWfV%2B%2FK8vGbCwW3Gm%2FEUlOO4ZbbK39y0JgNQ7fshxHr5Hdtd%2F6S%2FQkb6NPVDwn7Srh
Y0diWujXz5QlIYBSN7vDfMun3yF%2BGbmMExZ8MkOthuYkgMS9qiFoJGUXGyELsJfxbzdcRE9iyJn
p88L4%2BCtcO3l2JxIhMAgxOZx42RfAiDV1Gbpa4f%2F0urmWQ2VK7uZ%2FlViVrGAJ2kpH0EfwYE
Mb2YYT8FFjogqEpDSJX48BLIh1TE4nMbqQVG1cksCGDc0XyGKaF5Z7Ikw493Xz0JQ0BZvaf2Kceb7
MUZlsU1DSHcQQ9X%2Bxu9RcgUePJEe9BgCMpZ5Kr6r43qyk79noBSgrsSkDhT5sg%2Fc20RHQB8OX
%2BC4r3XGQFWF2m2j0xTc%2Boy14xqUmSB2qJtuWGOXDJspejDRP1GIfFnqDFdqSO3%2FkV9AC5Ee
39iJGv8I%2B5nErtQao645bCytn4B2bJah8R2fXLs8Dd4%2BC2ykxVrLxTUmJaGqd2RK%2F6t1E47
l%2B90Vp4WEzC0CFXXt9XNqdVjo2bZsXbfKQgO2zT2q2qCsgwbxVzIF5y39R%2BrkSkX16uuz3q6w
n3I5RI9M8Hn3DCzzv6Ms4rYxYuiqxaIcb7DgjI2fk1bdyiiRjSxzpCHpK6CWjBD8DPQYdkqGr%2Bs
oWeSvHvPLMSDxEPzwlnaxysRXzKphHUeUa2CCqcpagux2mbKkwHSXemX9I3V3AhPePp5XI5eCRiy3
D4%2BcBXOydie94Nz9DIhW749hPiVD9CioAgyqgAzFwCxEEUCXKTzu9xXX4DXg9b3CUfGzwERtY7x
TGT2y%2F9i7r5Xs0lrKi9ftws4JO5v%2Be3WuAEtWv0w%2FVKCl1WwTbV9xtx%2B4RZQ3%2Fewvv%
2F0GqiiSrhiVBGuCDaQs7stwqfkF3vFgGXmmODGTIkIxvYm2fzcEfq4A6LRp5RkYyJyUTF87c56tn
Qa%2Bo3xeiX5WRJybpabrRou09vyWLdlkhcUaBElGWB7iYUJ9bCltByEdNZnuDV%2FXlfnmDARKp8
RVN028czIk57wQMuizgWrM6S9Ku20noDmLgbT554UBf7FnjRWOb%2FF9OJuPpUcARBPrfuqTcOsBq
tZr7AJl3zz%2F53mpyn9rgzw5gBLgkvrdbciabJOAacccTDEB5kEzCLuprC3SlVedhgY%2BMQ5%2F
xgN%2Faf3TtJiBKFvb1V37BlbXXGosnPFcoH8I0XbqW5FSsxmcnpg48poJcB7j5eHq7Y%2F01RLb4
iMmzNap4%2BFg2F3LrwOI0Wk7ueIjgFd5KJ1iTda1ivGU%2Fchr9aTNpM5HiLb2fDW0pZ%2FFBJcI
XxpT9eNY%2FpVj5pnTW2ubpPnBulPOQTLCi1EOxbl33wnhUIfnGiVWJdrls2j3GWgqOnrYUbP%2FX
tNJqIucnMYGqPbcGIF2QRuiwD%2FiTRMvCRCmdCsYE%2FaXjOMhskX7KYC%2B9iG%2FT1wQRbfHSK
WD%2Fpv450OVDsfc1Adq6FCr1LesDNTew%2FF8Z3SiHnWS76OVsNM2SB%2FhMP67iu5UWVkb3%2FQ
qCN0aosOPs2QX0XBCZFmN6p3FhFnXPbAbaGz9y6KzUiUxC03U0fZcToKl4y%2Bw0P4IvxpjVt4t8b
84Q9hiBxd5xu1%2BRE973a%2FyIWO%2Fit1MdUSmxWakxWuGxDnQxwkNCN7ekL%2FQ%2B6FItm86b
w9cc%2FMiI7q2fK7y7YAzM3tmamhF1%2FWJNj1lH0vh%2BhNehJlLlb4Z%2F9ZtxMWV4LVTyrFaF1
zyCEqcKUTk0jc%2FXDwyKZc%2FSV9EOoPk2fVnmzs3WkA74GB%2BWtjdvQjSmnJYtPkMNsikHw%2B
RyB1hTkYbn3iQ6BUiJ0v97j7MVZHxCa1KS3t2gx8H7ts6Tfy5il89xVUdiZwfj0w06g199qlAqUMZ
EWxh0%3D%3C%2Fenc%3ACipherValue%3E%3C%2Fenc%3ACipherData%3E%3C%2Fenc%3AEncryp
tedData%3E
```

An un-escaped and reformatted version of the preceding xmlToken value, with the encrypted value elided, is as follows:

```
<enc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element" xmlns:enc=
"http://www.w3.org/2001/04/xmlenc#">
<enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"
/>
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<e:EncryptedKey xmlns:e="http://www.w3.org/2001/04/xmlenc#">
<e:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1
p">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
</e:EncryptionMethod>
<KeyInfo>
<o:SecurityTokenReference xmlns:o="http://docs.oasis-open.org/wss/2004/01/oas
is-200401-wss-wssecurity-secext-1.0.xsd">
<o:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-mes
sage-security-1.1#ThumbprintSHA1" EncodingType="http://docs.oasis-open.org/ws
s/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary">
+PYbznDaB/dlhjIfqCQ458E72wA=
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

```
</o:KeyIdentifier>
</o:SecurityTokenReference>
</KeyInfo>
<e:CipherData>
<e:CipherValue>
Eq9UhAJ8C9K5l4Mr3qmgX0XnyL1ChKs2PqMj0Sk6snw/IRNtXqLzmgbj2Vd3vFA4Vx1hileSTyqc1
kAsskqpqBc4bMHT61w1f0NxU10HDor0DlNVcVDm/AfLcyLqEP+oh05B+5ntVIJzL8Ro3typF0eoSm
3S6UnINOHIjHaVWyg=
</e:CipherValue>
</e:CipherData>
</e:EncryptedKey>
</KeyInfo>
<enc:CipherData>
<enc:CipherValue>
...=
</enc:CipherValue>
</enc:CipherData>
</enc:EncryptedData>
```


Copyright © OASIS® 2008-2009. All Rights Reserved.

B. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Original Authors of the initial contributions:

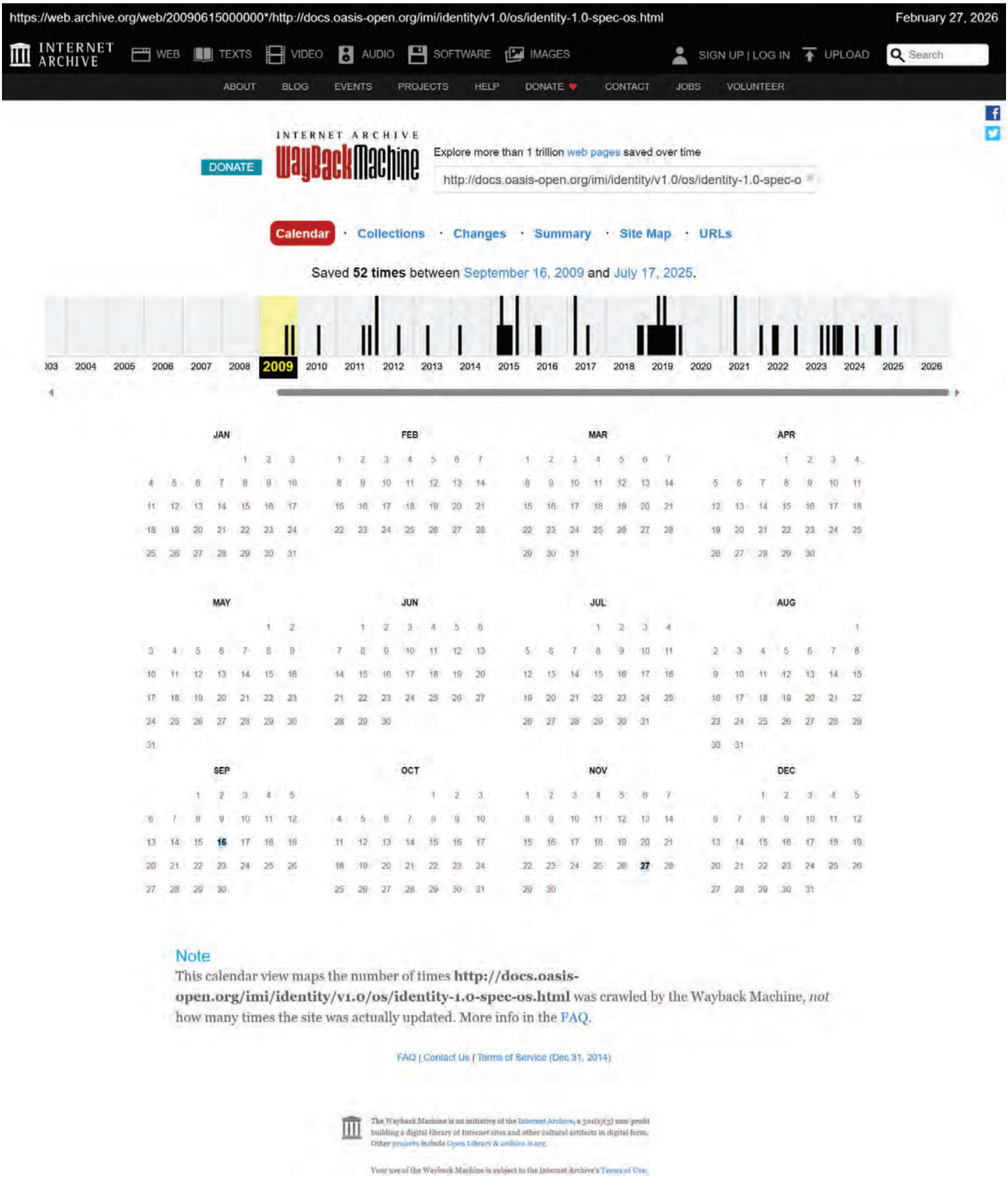
Arun Nanda, Microsoft Corporation
Michael B. Jones, Microsoft Corporation
Jan Alexander, Microsoft
Giovanni Della-Libera, Microsoft
Martin Gudgin, Microsoft
Kirill Gavrylyuk, Microsoft
Tomasz Janczuk, Microsoft
Michael McIntosh, IBM
Anthony Nadalin, IBM
Bruce Rich, IBM
Doug Walter, Microsoft

Participants:

John Bradley, Individual
Norman Brickman, Mitre Corporation
Jeffrey Broberg, CA
Scott Cantor Internet2
Ruchith Fernando, WSO2
Marc Goodner, Microsoft Corporation (Chair)
Patrick Harding, Ping Identity
Andrew Hodgkinson, Novell
Mario Ivkovic, A-SIT, Zentrum für Sichere Informationstechnologie - Austria
Michael B. Jones, Microsoft Corporation (Editor)
Mike Kirkwood, Polka Networks
Herbert Leitold, A-SIT, Zentrum für Sichere Informationstechnologie - Austria
Michael McIntosh, IBM (Editor)
Dale Olds, Novell
Anthony Nadalin, IBM (Chair)
Drummond Reed, Cordance
Bruce Rich ,IBM
Darran Rolls, SailPoint Technologies
Prabath Siriwardena, WSO2

Copyright © OASIS® 2008-2009. All Rights Reserved.

Appendix
OASIS03


https://web.archive.org/web/20090615000000*/http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.html
February 27, 2026
INTERNET ARCHIVE
WEB TEXTS VIDEO AUDIO SOFTWARE IMAGES
SIGN UP | LOG IN UPLOAD
Search
ABOUT BLOG EVENTS PROJECTS HELP DONATE CONTACT JOBS VOLUNTEER
DONATE
INTERNET ARCHIVE WayBackMachine
Explore more than 1 trillion web pages saved over time
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-o
Calendar · Collections · Changes · Summary · Site Map · URLs
Saved 52 times between September 16, 2009 and July 17, 2025.
2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 2026
JAN FEB MAR APR
MAY JUN JUL AUG
SEP OCT NOV DEC
Note
This calendar view maps the number of times http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.html was crawled by the Wayback Machine, not how many times the site was actually updated. More info in the FAQ.
FAQ | Contact Us | Terms of Service (Dec 31, 2014)
The Wayback Machine is an initiative of the Internet Archive, a 501(c)(3) non-profit building a digital library of Internet sites and other cultural artifacts in digital form. Other projects include Open Library & archive-it.org.
Your use of the Wayback Machine is subject to the Internet Archive's Terms of Use.

https://web.archive.org/web/20090916031534/http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.html February 27, 2026

INTERNET ARCHIVE Wayback Machine http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.html Go AUG SEP NOV
52 captures 16 Sep 2009 - 17 Jul 2025 2008 16 2009 2011 About this capture

OASIS

Identity Metasystem Interoperability Version 1.0

OASIS Standard

1 July 2009

Specification URIs:
This Version:
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.html
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.doc (Authoritative)
http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.pdf

Previous Version:
http://docs.oasis-open.org/imi/identity/v1.0/cs/identity-1.0-spec-cs-01.html
http://docs.oasis-open.org/imi/identity/v1.0/cs/identity-1.0-spec-cs-01.doc (Authoritative)
http://docs.oasis-open.org/imi/identity/v1.0/cs/identity-1.0-spec-cs-01.pdf

Latest Version:
http://docs.oasis-open.org/imi/identity/v1.0/identity.html
http://docs.oasis-open.org/imi/identity/v1.0/identity.doc
http://docs.oasis-open.org/imi/identity/v1.0/identity.pdf

Technical Committee:
OASIS Identity Metasystem Interoperability (IMI) TC

Chair(s):
Marc Goodner
Anthony Nadalin

Editor(s):
Michael B. Jones
Michael McIntosh

Related work:
This specification replaces or supersedes:

- None

This specification is related to:

- WS-Trust
- WS-SecurityPolicy
- WS-Addressing

Declared XML Namespace(s):
http://docs.oasis-open.org/imi/ns/identity-200810
http://schemas.xmlsoap.org/ws/2005/05/identity
http://schemas.xmlsoap.org/ws/2006/02/addressingidentity
http://schemas.xmlsoap.org/ws/2007/01/identity

Abstract:

This document is intended for developers and architects who wish to design identity systems and applications that interoperate using the Identity Metasystem Interoperability specification.

An Identity Selector and the associated identity system components allow users to manage their Digital Identities from different Identity Providers, and employ them in various contexts to access online services. In this specification, identities are represented to users as "Information Cards". Information Cards can be used both at applications hosted on Web sites accessed through Web browsers and rich client applications directly employing Web services.

This specification also provides a related mechanism to describe security-verifiable identity for endpoints by leveraging extensibility of the WS-Addressing specification. This is achieved via XML [XML 1.0] elements for identity provided as part of WS-Addressing Endpoint References. This mechanism enables messaging systems to support multiple trust models across networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner.

Status:

This document was last revised or approved by the Identity Metasystem Interoperability TC on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/imi/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/imi/ipr.php.

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/imi/.

Notices

Copyright © OASIS® 2008-2009. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

Table of Contents

1 Introduction

1.1 Notational Conventions
1.2 Namespaces
1.3 Schema
1.4 Terminology
1.5 Normative References
1.6 Non-Normative References
2 Relying Party Interactions
2.1 Expressing Token Requirements of Relying Party
2.1.1 Issuer of Tokens
2.1.2 Type of Proof Key in Issued Tokens
2.1.3 Claims in Issued Tokens
2.2 Expressing Privacy Policy of Relying Party
2.3 Employing Relying Party STSs
3 Identity Provider Interactions
3.1 Information Card
3.1.1 Information Card Format
3.1.2 Issuing Information Cards
3.2 Identity Provider Policy
3.2.1 Require Information Card Provisioning
3.2.2 Policy Metadata Location
3.3 Token Request and Response
3.3.1 Information Card Reference
3.3.2 Claims and Other Token Parameters
3.3.3 Token Scope
3.3.4 Client Pseudonym
3.3.5 Proof Key for Issued Token
3.3.6 Display Token
3.3.7 Token References
4 Authenticating to Identity Provider
4.1 Username and Password Credential
4.2 Kerberos v5 Credential
4.3 X.509v3 Certificate Credential
4.4 Self-issued Token Credential
5 Faults
5.1 Relying Party
5.2 Identity Provider
5.2.1 Identity Provider Custom Error Messages
6 Information Cards Transfer Format
6.1 Pre-Encryption Transfer Format
6.1.1 PIN Protected Card
6.1.2 Computing the ic:IssuerId
6.1.3 Computing the ic:IssuerName
6.1.4 Creating the ic:HashSalt
6.2 Post-Encryption Transfer Format
7 Simple Identity Provider Profile
7.1 Self-Issued Information Card
7.2 Self-Issued Token Characteristics
7.3 Self-Issued Token Encryption
7.4 Self-Issued Token Signing Key
7.4.1 Processing Rules
7.5 Claim Types
7.5.1 First Name
7.5.2 Last Name
7.5.3 Email Address
7.5.4 Street Address
7.5.5 Locality Name or City
7.5.6 State or Province
7.5.7 Postal Code
7.5.8 Country
7.5.9 Primary or Home Telephone Number
7.5.10 Secondary or Work Telephone Number
7.5.11 Mobile Telephone Number
7.5.12 Date of Birth
7.5.13 Gender
7.5.14 Private Personal Identifier
7.5.15 Web Page
7.6 The PPID Claim
7.6.1 Relying Party Identifier and Relying Party PPID Seed
7.6.2 PPID
7.6.3 Friendly Identifier
8 Relying Parties without Certificates
8.1 Relying Party Identifier and Relying Party PPID Seed
8.2 AppliesTo Information
8.3 Token Signing and Encryption
9 Using WS-SecurityPolicy 1.2 and WS-Trust 1.3
9.1 Overview of Differences
9.2 Identity Selector Differences
9.3 Security Token Service Differences
10 Browser Behavior with Information Cards
10.1 Basic Protocol Flow when using an Information Card at a Web Site
10.2 Protocol Flow with Relying Party STS
10.3 User Perspective and Examples
10.4 Browser Perspective
10.5 Web Site Perspective
11 Invoking an Identity Selector from a Web Page

11.1 Syntax Alternatives: OBJECT and XHTML tags
11.1.1 OBJECT Syntax Examples
11.1.2 XHTML Syntax Example
11.2 Identity Selector Invocation Parameters
11.2.1 issuer
11.2.2 issuerPolicy
11.2.3 tokenType
11.2.4 requiredClaims
11.2.5 optionalClaims
11.2.6 privacyUrl
11.2.7 privacyVersion
11.3 Data Types for Use with Scripting
11.4 Detecting and Utilizing an Information Card-enabled Browser
11.5 Behavior within Frames
11.6 Invocation Using the Document Object Model (DOM)
11.7 Auditing, Non-Auditing, and Auditing-Optional Cards
12 Endpoint Reference wsai:Identity Property
12.1 Default Value
12.2 Identity Representation
12.2.1 DNS Name
12.2.2 Service Principal Name
12.2.3 User Principal Name
12.2.4 KeyInfo
12.2.5 Security Token
12.2.6 Security Token Reference
13 Security Considerations
13.1 Protection of Information Cards by Identity Selectors
13.2 Relying Parties Without Certificates
13.3 Endpoint References
14 Conformance
A. HTTPS POST Sample Contents
B. Acknowledgements

1 Introduction

The Identity Metasystem Interoperability specification prescribes a subset of the mechanisms defined in [WS-Trust 1.2], [WS-Trust 1.3], [WS-SecurityPolicy 1.1], [WS-SecurityPolicy 1.2], and [WS-MetadataExchange] to facilitate the integration of Digital Identity into an interoperable token issuance and consumption framework using the Information Card Model. It documents the Web interfaces utilized by browsers and Web applications that utilize the Information Card Model. Finally, it extends WS-Addressing's endpoint reference by providing identity information about the endpoint that can be verified through a variety of security means, such as https or the wealth of WS-Security specifications.

This profile constrains the schema elements/extensions used by the Information Card Model, and behaviors for conforming Relying Parties, Identity Providers, and Identity Selectors.

1.1 Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119].

This specification uses the following syntax to define outlines for assertions:

- The syntax appears as an XML instance, but values in italics indicate data types instead of literal values.
- Characters are appended to elements and attributes to indicate cardinality:
  - "?" (0 or 1)
  - "*" (0 or more)
  - "+" (1 or more)
- The character "|" is used to indicate a choice between alternatives.
- The characters "(" and ")" are used to indicate that contained items are to be treated as a group with respect to cardinality or choice.
- The characters "[" and "]" are used to call out references and property names.
- Ellipses (i.e., "...") indicate points of extensibility. Additional children and/or attributes MAY be added at the indicated extension points but MUST NOT contradict the semantics of the parent and/or owner, respectively. By default, if a receiver does not recognize an extension, the receiver SHOULD ignore the extension; exceptions to this processing rule, if any, are clearly indicated below.
- XML namespace prefixes (see Table 2) are used to indicate the namespace of the element being defined.

Elements and Attributes defined by this specification are referred to in the text of this document using XPath 1.0 expressions. Extensibility points are referred to using an extended version of this syntax:

- An element extensibility point is referred to using {any} in place of the element name. This indicates that any element name can be used, from any namespace other than the namespace of this specification.
- An attribute extensibility point is referred to using @{any} in place of the attribute name. This indicates that any attribute name can be used, from any namespace other than the namespace of this specification.

Extensibility points in the exemplar might not be described in the corresponding text.

1.2 Namespaces

Table 1 lists the XML namespaces that are used in this document.

Prefix	XML Namespace	Specification(s)
ds	http://www.w3.org/2000/09/xmldsig#	XML Digital Signatures
ic	http://schemas.xmlsoap.org/ws/2005/05/identity	This document
ic07	http://schemas.xmlsoap.org/ws/2007/01/identity	Namespace for additional elements also defined by this document
ic08	http://docs.oasis-open.org/imi/ns/identity-200810	Namespace for new elements defined by this document
S	*May refer to either* http://schemas.xmlsoap.org/soap/envelope *or* http://www.w3.org/2003/05/soap-envelope *since both may be used*	SOAP
S11	http://schemas.xmlsoap.org/soap/envelope	SOAP 1.1 [SOAP 1.1]
S12	http://www.w3.org/2003/05/soap-envelope	SOAP 1.2 [SOAP 1.2]
saml	urn:oasis:names:tc:SAML:1.0:assertion	SAML 1.0
sp	*May refer to either*	WS-SecurityPolicy

	http://schemas.xmlsoap.org/ws/2005/07/securitypolicy *or* http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702 *since both may be used*	
sp11	http://schemas.xmlsoap.org/ws/2005/07/securitypolicy	WS-SecurityPolicy 1.1 [WS-SecurityPolicy 1.1]
sp12	http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702	WS-SecurityPolicy 1.2 [WS-SecurityPolicy 1.2]
wsa	http://www.w3.org/2005/08/addressing	WS-Addressing [WS-Addressing]
wsai	http://schemas.xmlsoap.org/ws/2006/02/addressingidentity	Addressing Identity extension for WS-Addressing also defined by this document
wsdl	*May refer to either* http://schemas.xmlsoap.org/wsdl/ *or* http://www.w3.org/TR/wsdl20 *since both may be used*	Web Services Description Language
wsdl11	http://schemas.xmlsoap.org/wsdl/	Web Services Description Language [WSDL 1.1]
wsdl20	http://www.w3.org/TR/wsdl20	Web Services Description Language [WSDL 2.0]
wsp	http://schemas.xmlsoap.org/ws/2004/09/policy	WS-Policy [WS-Policy]
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	WS-Security Extensions [WS-Security]
wst	*May refer to either* http://schemas.xmlsoap.org/ws/2005/02/trust *or* http://docs.oasis-open.org/ws-sx/ws-trust/200512 *since both may be used*	WS-Trust
wst12	http://schemas.xmlsoap.org/ws/2005/02/trust	WS-Trust 1.2 [WS-Trust 1.2]
wst13	http://docs.oasis-open.org/ws-sx/ws-trust/200512	WS-Trust 1.3 [WS-Trust 1.3]
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	WS-SecurityUtility
wsx	http://schemas.xmlsoap.org/ws/2004/09/mex	WS-MetadataExchange [WS-MetadataExchange]
xs	http://www.w3.org/2001/XMLSchema	XML Schema [Part 1, 2]

Note that the versions identified in the above table supersede versions identified in referenced specifications.

1.3 Schema

A copy of the XML Schemas for this document can be found at:

```
http://docs.oasis-open.org/imi/identity/200810/identity.xsd
http://docs.oasis-open.org/imi/identity/200810/addr-identity.xsd
http://docs.oasis-open.org/imi/identity/200810/claims.xsd
http://docs.oasis-open.org/imi/identity/200810/identity2007.xsd
```

1.4 Terminology

The following definitions establish the terminology and usage in this document.

Information Card Model – The "*Information Card Model*" refers to the use of Information Cards containing metadata for obtaining Digital Identity claims from Identity Providers and then conveying them to Relying Parties under user control.

Information Card – An *Information Card* provides a visual representation of a Digital Identity for the end user. Information Cards contain a reference to an IP/STS that issues Security Tokens containing the Claims for that Digital Identity.

Digital Identity – A "*Digital Identity*" is a set of Claims made by one party about another party.

Claim – A "*Claim*" is a piece of information about a Subject that an Identity Provider asserts about that Subject.

Subject – A "*Subject*" is an individual or entity about whom claims are made by an Identity Provider.

Service Requester – The term "*Service Requester*" means software acting on behalf of a party who wants to obtain a service through a digital network.

Relying Party – The term "*Relying Party*" (RP) means a network entity providing the desired service, and relying upon Digital Identity.

Identity Provider – The term "*Identity Provider*" (IP) means a network entity providing the Digital Identity claims used by a Relying Party.

Security Token Service – The term "*Security Token Service*" (STS) refers to a WS-Trust endpoint.

Identity Provider Security Token Service – The term "Identity Provider Security Token Service" (IP/STS) refers to the Security Token Service run by an Identity Provider to issue tokens.

Relying Party Security Token Service – The term "Relying Party Security Token Service" (RP/STS) refers to a Security Token Service run by a Relying Party to accept and issue tokens.

Identity Selector – The term "*Identity Selector*" (IS) refers to a software component available to the Service Requester through which the user controls and dispatches her Digital Identities.

Trust Identity – A *trust identity* is a verifiable claim about a principal (e.g. name, identity, key, group, privilege, capability, etc).

Security Token – A *security token* represents a collection of claims.

Signed Security Token – A *signed security token* is a security token that is asserted and cryptographically endorsed by a specific authority (e.g. an X.509 certificate, a Kerberos ticket, or a self-issued Information Card).

Unsigned Security Token – An un*signed security token* is a security token that is not cryptographically endorsed by a specific authority (e.g. a security token backed by shared secrets such as usernames and passwords).

Proof-of-Possession – The *proof-of-possession* information is data that is used in a proof process to demonstrate the sender's knowledge of information that should only be known to the claiming sender of a security token.

Integrity – *Integrity* is the process by which it is guaranteed that information is not modified in transit.

Confidentiality – *Confidentiality* is the process by which data is protected such that only authorized actors or security token owners can view the data

Digest – A *digest* is a cryptographic checksum of an octet stream.

Signature - A *signature* is a cryptographic binding of a proof-of-possession and a digest. This covers both symmetric key-based and public key-based signatures. Consequently, non-repudiation is not always achieved.

Certificate – Uses of the term *certificate* in this specification refer to X.509 certificates unless otherwise qualified. Usage of certificates is dictated by the underlying protocols, e.g. HTTPS or WS-Security, except where noted.

1.5 Normative References

[DOM]

"Document Object Model (DOM)", November 2000. http://www.w3.org/DOM/

[EV Cert]

CA / Browser Forum, "Guidelines for the Issuance and Management of Extended Validation Certificates, Version 1.1", April 2008. http://cabforum.org/EV_Certificate_Guidelines_V11.pdf

[HTTP]
R. Fielding et al., "IETF RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1", June 1999. http://www.ietf.org/rfc/rfc2616.txt

[HTTPS]
E. Rescorla, "RFC 2818: HTTP over TLS", May 2000. http://www.ietf.org/rfc/rfc2818.txt

[RFC 1274]
P. Barker and S. Kille, "RFC 1274: The COSINE and Internet X.500 Schema", November 1991. http://www.ietf.org/rfc/rfc1274.txt

[RFC 2119]
S. Bradner, "RFC 2119: Key words for use in RFCs to Indicate Requirement Levels", March 1997. http://www.ietf.org/rfc/rfc2119.txt

[RFC 2256]
M. Wahl, "RFC 2256: A Summary of the X.500(96) User Schema for use with LDAPv3", December 1997. http://www.ietf.org/rfc/rfc2256.txt

[RFC 2459]
R. Housley, W. Ford, W. Polk, and D. Solo, "RFC 2459: Internet X.509 Public Key Infrastructure - Certificate and CRL Profile", January 1999. http://www.ietf.org/rfc/rfc2459.txt

[RFC 2898]
B. Kaliski, "PKCS #5: Password-Based Cryptography Specification, Version 2.0", September 2000. http://www.ietf.org/rfc/rfc2898.txt

[RFC 3066]
H. Alvestrand, "Tags for the Identification of Languages", January 2001. http://www.faqs.org/rfcs/rfc3066.html

[SOAP 1.1]
W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[SOAP 1.2]
M. Gudgin, et al., "SOAP Version 1.2 Part 1: Messaging Framework", June 2003. http://www.w3.org/TR/soap12-part1/

[URI]
T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998. http://www.ietf.org/rfc/rfc2396.txt

[WS-Addressing]
W3C Recommendation, "Web Service Addressing (WS-Addressing)", 9 May 2006. http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/

[WS-MetadataExchange]
"Web Services Metadata Exchange (WS-MetadataExchange), Version 1.1", August 2006. http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf

[WSDL 1.1]
W3C Note, "Web Services Description Language (WSDL 1.1)," 15 March 2001. http://www.w3.org/TR/wsdl

[WSDL 2.0]
"Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language", June 2007. http://www.w3.org/TR/wsdl20

[WS-Policy]
"Web Services Policy Framework (WS-Policy), Version 1.2", March 2006. http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf

[WS-Security]
A. Nadalin et al., "Web Services Security: SOAP Message Security 1.0", May 2004. http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[WS-SecurityPolicy 1.1]
"Web Services Security Policy Language (WS-SecurityPolicy), Version 1.1", July 2005. http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf

[WS-SecurityPolicy 1.2]
OASIS, "WS-SecurityPolicy 1.2", July 2007. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.pdf

[WS-Trust 1.2]
"Web Services Trust Language (WS-Trust)", February 2005. http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf

[WS-Trust 1.3]
OASIS, "WS-Trust 1.3", March 2007. http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf

[XML 1.0]
W3C Recommendation, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", September 2006. http://www.w3.org/TR/xml/

[XMLDSIG]
Eastlake III, D., Reagle, J., and Solo, D., "XML-Signature Syntax and Processing", March 2002. http://www.ietf.org/rfc/rfc3275.txt

[XMLENC]
Imamura, T., Dillaway, B., and Simon, E., "XML Encryption Syntax and Processing", August 2002. http://www.w3.org/TR/xmlenc-core/

[XML Schema, Part 1]
H. Thompson et al., "XML Schema Part 1: Structures", May 2001. http://www.w3.org/TR/xmlschema-1/

[XML Schema, Part 2]
P. Biron et al., "XML Schema Part 2: Datatypes", May 2001. http://www.w3.org/TR/xmlschema-2/

1.6 Non-Normative References

[Addressing-Ext]
J. Alexander et al., "Application Note: Web Services Addressing Endpoint References and Identity", July 2008. http://schemas.xmlsoap.org/ws/2006/02/addressingidentity

[ISIP]
A. Nanda and M. Jones, "Identity Selector Interoperability Profile V1.5", July 2008. http://www.microsoft.com/downloads/details.aspx?FamilyID=b94817fc-3991-4dd0-8e85-b73e626f6764&DisplayLang=en

[ISIP Guide]
Microsoft Corporation and Ping Identity Corporation, "An Implementer's Guide to the Identity Selector Interoperability Profile V1.5", July 2008. http://www.microsoft.com/downloads/details.aspx?FamilyID=b94817fc-3991-4dd0-8e85-b73e626f6764&DisplayLang=en

[ISIP Web Guide]
M. Jones, "A Guide to Using the Identity Selector Interoperability Profile V1.5 within Web Applications and Browsers", July 2008. http://www.microsoft.com/downloads/details.aspx?FamilyID=b94817fc-3991-4dd0-8e85-b73e626f6764&DisplayLang=en

---

2 Relying Party Interactions

This section defines the constructs used by a Relying Party Web service for specifying and conveying its Security Token requirements to the Service Requester.

2.1 Expressing Token Requirements of Relying Party

A Relying Party specifies its Security Token requirements as part of its Security Policy using the primitives and assertions defined in WS-SecurityPolicy. The primary construct in the Security Policy of the Relying Party used to specify its requirement for a Security Token from an Identity Provider is the `sp:IssuedToken` policy assertion. The basic form of the issued token policy assertion as defined in WS-SecurityPolicy is as follows.

```
<sp:IssuedToken sp:Usage="xs:anyURI" sp:IncludeToken="xs:anyURI"
    xmlns:sp="..." xmlns:wsa="..." xmlns:wsp="..." ...>
  <sp:Issuer>
    wsa:EndpointReference | xs:any
  </sp:Issuer>
  <sp:RequestSecurityTokenTemplate>
```

```
    ...
  </sp:RequestSecurityTokenTemplate>
  <wsp:Policy>
    ...
  </wsp:Policy>
  ...
</sp:IssuedToken>
```

The attributes and elements listed in the schema fragment above are described in WS-SecurityPolicy.

The ensuing subsections describe special parameters added by this profile as extensions to the `sp:IssuedToken` policy assertion that convey additional instructions to the Identity Selector available to the Service Requester.

2.1.1 Issuer of Tokens

The `sp:IssuedToken/sp:Issuer` element in an issued token policy specifies the issuer for the requested token. More specifically, it SHOULD contain the endpoint reference of an Identity Provider STS that can issue the requested token.

A Relying Party MUST specify the issuer for a requested token in one of the following ways:

- Indicate a ***specific*** issuer by specifying the issuer's endpoint as the value of the `sp:Issuer/wsa:Address` element.
- Indicate that the issuer is ***unspecified*** by omitting the `sp:Issuer` element, which means that the Service Requester should determine the appropriate issuer for the requested token with help from the user if necessary.

When requiring a specific issuer, a Relying Party MAY specify that it will accept self-issued Security Tokens by using the special URI below as the value of the `wsa:Address` element within the endpoint reference for the issuer.

URI:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self
```

Following is an example of using this URI within an issued token policy.

Example:

```
<sp:IssuedToken xmlns:sp="..." xmlns:wsa="..." ...>
  <sp:Issuer>
    <wsa:Address>
      http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self
    </wsa:Address>
  </sp:Issuer>
  ...
</sp:IssuedToken>
```

A Relying Party MAY specify the value of the `sp:Issuer/wsa:Address` element in policy as a "logical name" of the token issuer instead of an actual network address where the token is issued. An Identity Selector SHOULD resolve the logical name to an appropriate endpoint for the token issuer by matching the issuer name in Information Cards available to it.

If a Relying Party specifies the token issuer as a network endpoint in policy, then it MUST also specify the location of issuer metadata from where the issuer's policy metadata can be obtained. This is done using the mechanism defined in [WS-Addressing] for embedding metadata within an endpoint reference. The following example shows a token policy where the issuer endpoint and its corresponding metadata location are specified.

Example:

```
<sp:IssuedToken xmlns:sp="..." xmlns:wsa="..." xmlns:wsx="..." ...>
  <sp:Issuer>
    <wsa:Address>http://contoso.com/sts</wsa:Address>
    <wsa:Metadata>
      <wsx:Metadata>
        <wsx:MetadataSection
            Dialect="http://schemas.xmlsoap.org/ws/2004/09/mex">
          <wsx:MetadataReference>
            <wsa:Address>https://contoso.com/sts/mex</wsa:Address>
          </wsx:MetadataReference>
        </wsx:MetadataSection>
      </wsx:Metadata>
    </wsa:Metadata>
  </sp:Issuer>
  ...
</sp:IssuedToken>
```

2.1.2 Type of Proof Key in Issued Tokens

If no explicit key type is specified by the Relying Party, then an Identity Selector SHOULD request an asymmetric key token from the Identity Provider to maximize user privacy and security.

A Relying Party MAY explicitly request the use of an ***asymmetric*** or ***symmetric*** key in the requested token by using the `wst:KeyType` element within its issued token policy assertion. The key type URIs are defined in [WS-Trust]. The following example illustrates the use of this element in the Relying Party's Security Policy to request a symmetric key in the issued token.

Example:

```
<sp:IssuedToken xmlns:sp="..." xmlns:wst="...">
  <sp:RequestSecurityTokenTemplate>
    <wst:KeyType>
      http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey
    </wst:KeyType>
  </sp:RequestSecurityTokenTemplate>
</sp:IssuedToken>
```

2.1.3 Claims in Issued Tokens

The claims requirement of a Relying Party can be expressed in its token policy by using the optional `wst:Claims` parameter defined in [WS-Trust 1.2] and [WS-Trust 1.3]. However, the `wst:Claims` parameter has an open content model. This profile defines the `ic:ClaimType` element for use as a child of the `wst:Claims` element. A Relying Party MAY use this element to specify an individual claim type requested. Further, each requested claim MAY be specified as being ***required*** or ***optional***. Multiple `ic:ClaimType` elements can be included to specify multiple claim types requested.

The outline for the `ic:ClaimType` element is as follows:

Syntax:

```
<ic:ClaimType Uri="xs:anyURI" Optional="xs:boolean"? xmlns:ic="..." /> *
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:ClaimType

Indicates the requested claim type.

/ic:ClaimType/@Uri

The unique identifier of the requested claim type.

/ic:ClaimType/@Optional

Indicates if the claim can be absent in the Security Token. By default, any requested claim type is a required claim and MUST be present in the issued Security Token.

Two `<ic:ClaimType>` elements refer to the same claim type if and only if the values of their XML attribute named `Uri` are equal in a case-sensitive string comparison.

When the `ic:ClaimType` element is used within the `wst:Claims` parameter in a token policy to specify claims requirement, the `wst:Dialect` attribute on the `wst:Claims` element MUST be qualified with the URI value below.

Dialect URI:

```
http://schemas.xmlsoap.org/ws/2005/05/identity
```

The above dialect URI value indicates that the specified claim elements are to be processed according to this profile.

Following is an example of using this assertion within an issued token policy to require two claim types where one claim type is optional.

Example:

```
<sp:IssuedToken xmlns:sp="..." xmlns:wst="..." xmlns:ic="..." ...>
  ...
  <sp:RequestSecurityTokenTemplate>
```

```
    ...
    <wst:Claims
        Dialect="http://schemas.xmlsoap.org/ws/2005/05/identity">
      <ic:ClaimType
          Uri="http://.../ws/2005/05/identity/claims/givenname"/>
      <ic:ClaimType
          Uri="http://.../ws/2005/05/identity/claims/surname"
          Optional="true" />
    </wst:Claims>
  </sp:RequestSecurityTokenTemplate>
  ...
</sp:IssuedToken>
```

This profile also defines a standard set of claim types for common personal information about users that may be requested by Relying Party Web services in Security Tokens and supported by any Identity Provider. These standard claim types are defined in Section 7.5.

2.2 Expressing Privacy Policy of Relying Party

A Relying Party Web service SHOULD publish its "Privacy Policy". Users might decide to release tokens and interact further with that service based on its Privacy Policy. No assumptions are made regarding the format and content of the Privacy Policy and an Identity Selector is NOT REQUIRED to parse, interpret or act on the Privacy Policy programmatically.

To express the location of its privacy statement, a Web service MUST use the optional policy assertion `ic:PrivacyNotice` defined below:

Syntax:

```
<ic:PrivacyNotice Version="xs:unsignedInt"? xmlns:ic="...">
  xs:anyURI
</ic:PrivacyNotice>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:PrivacyNotice

This element is used to express the location of the privacy statement of a Web service.

/ic:PrivacyNotice/@Version

This optional attribute provides a version number for the privacy statement allowing changes in its content to be reflected as a change in the version number. If present, it MUST have a minimum value of 1.

Following is an example of using this policy element to express the location of the privacy statement of a Web service.

Example:

```
<wsp:Policy xmlns:wsp="..." xmlns:ic="...">
  ...
  <ic:PrivacyNotice Version="1">
    http://www.contoso.com/privacy
  </ic:PrivacyNotice>
  ...
</wsp:Policy>
```

An Identity Selector MUST be able to accept a privacy statement location specified as an URL using the [HTTP] scheme (as illustrated above) or the [HTTPS] scheme.

Because the Privacy Policy assertion points to a "privacy statement" that applies to a service endpoint, the assertion MUST apply to [Endpoint Policy Subject]. In other words, a policy expression containing the Privacy Policy assertion MUST be attached to a `wsdl:binding` in the metadata for the service.

Further, when an Identity Selector can only render the privacy statement document in a limited number of document formats (media types), it MAY use the HTTP request-header field "Accept" in its HTTP GET request to specify the media-types it can accept. For example, the following request-header specifies that the client will accept the Privacy Policy only as a plain text or a HTML document.

```
Accept: text/plain, text/html
```

Similarly, if an Identity Selector wants to obtain the privacy statement in a specific language, it MAY use the HTTP request-header field "Accept-Language" in its HTTP GET request to specify the languages it is willing to accept. For example, the following request-header specifies that the client will accept the Privacy Policy only in Danish.

```
Accept-Language: da
```

A Web service, however, is NOT REQUIRED to be able to fulfill the document format and language requests of an Identity Selector. It MAY publish its privacy statement in a fixed set of document formats and languages.

2.3 Employing Relying Party STSs

The Security Policy of a Relying Party MAY require that an issued token be obtained from a Relying Party STS. This can create a chain of STSs. The Identity Selector MUST follow the RP/STS chain, contacting each referenced STS, resolving its Policy statements and continuing to the STS it refers to.

When following a chain of STSs, when an STS with an `ic:RequireFederatedIdentityProvisioning` declaration is encountered as per Section 3.2.1, this informs the Identity Selector that the STS is an IP/STS and therefore ends the STS chain, rather than a member of the RP/STS chain. Furthermore, if an RP or RP/STS provides an incomplete Security Policy, such as no issuer or no required claims, the Identity Selector MUST be invoked so a card and requested claims can be selected by the user, enabling a Request for Security Token (RST) to be constructed and sent to the selected IP/STS.

The RP/STS's Policy is used for card matching. If the RP/STS requests a private personal identifier (PPID) claim (see Section 7.5.14), the RP/STS's certificate is used for calculating PPID, Signing Key, and Client Pseudonym (see Section 3.3.4) values – not the certificate of the Relying Party. This enables a single RP/STS to service multiple Relying Parties while always receiving the same PPID value for a given user from the Identity Selector.

Identity Selectors MUST enable users to make Relying Party trust decisions based on the identity of the Relying Party, possibly including displaying attributes from its certificate. By trusting the RP, the user is implicitly trusting the chain of RP/STSs that the RP employs.

Each RP/STS endpoint MUST provide a certificate. This certificate MAY be communicated either via Transport (such as HTTPS) or Message (such as WS-Security) Security. If Message Security is employed, transports not providing security (such as HTTP) MAY be used.

Like IP/STSs, RP/STSs publish endpoint metadata. This metadata MAY be retrieved via either WS-MetadataExchange or HTTPS GET in the same manner that IP/STS metadata can be, as described in Section 3.1.1.2.

Like IP/STSs, no changes to the syntax used to specify metadata locations occurs when RP/STS metadata is published by the Relying Party STS as a page retrievable using HTTPS GET. Relying Parties and Identity Providers MAY consequently support either or both retrieval methods for the same metadata addresses.

3 Identity Provider Interactions

This section defines the constructs used by an Identity Selector for interacting with an Identity Provider to obtain Information Cards, and to request and obtain Security Tokens.

3.1 Information Card

An Information Card represents a Digital Identity of a Subject that can be issued by an Identity Provider. It is an artifact containing metadata that represents the token issuance relationship between an Identity Provider and a Subject, and provides a visual representation of the Digital Identity. Multiple Digital Identities for a Subject from the same Identity Provider are represented by different Information Cards. Subjects may obtain an Information Card from an Identity Provider, and may have a collection of Information Cards from various Identity Providers.

3.1.1 Information Card Format

An Information Card is represented as a signed XML document that is issued by an Identity Provider. The XML schema for an Information Card is defined below:

Syntax:

```
<ic:InformationCard xml:lang="xs:language"
    xmlns:ic="..." xmlns:ic07="..." ...>
  <ic:InformationCardReference> ... </ic:InformationCardReference>
  <ic:CardName> xs:string </ic:CardName> ?
  <ic:CardImage MimeType="xs:string"> xs:base64Binary </ic:CardImage> ?
  <ic:Issuer> xs:anyURI </ic:Issuer>
  <ic:TimeIssued> xs:dateTime </ic:TimeIssued>
  <ic:TimeExpires> xs:dateTime </ic:TimeExpires> ?
  <ic:TokenServiceList> ... </ic:TokenServiceList>
  <ic:SupportedTokenTypeList> ... </ic:SupportedTokenTypeList> ?
  <ic:SupportedClaimTypeList> ... </ic:SupportedClaimTypeList> ?
  <ic:RequireAppliesTo ...> ... </ic:RequireAppliesTo> ?
  <ic:PrivacyNotice ...> ... </ic:PrivacyNotice> ?
  <ic07:RequireStrongRecipientIdentity /> ?
  <ic07:IssuerInformation> ... </ic07:IssuerInformation> *
  ...
</ic:InformationCard>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:InformationCard

An Information Card issued by an Identity Provider.

/ic:InformationCard/@xml:lang

A required language identifier, using the language codes specified in [RFC 3066], in which the content of localizable elements have been localized.

/ic:InformationCard/ic:InformationCardReference

This required element provides a specific reference for the Information Card by which it can be uniquely identified within the scope of an issuer. This reference MUST be included by an Identity Selector in all token requests sent to the Identity Provider based on that Information Card. The detailed schema of this element is defined in Section 3.1.1.1.

/ic:InformationCard/ic:CardName

This optional element provides a friendly textual name for the issued Information Card. The content of this element MAY be localized in a specific language.

/ic:InformationCard/ic:CardImage

This optional element contains a base64 encoded inline image that provides a graphical image for the issued Information Card. It SHOULD contain an image within the size range of 60 pixels wide by 40 pixels high and 240 pixels wide by 160 pixels high. It is RECOMMENDED that the image have an aspect ratio of 3:2 and the image size be 120 by 80 pixels.

/ic:InformationCard/ic:CardImage/@MimeType

This required attribute provides a MIME type specifying the format of the included card image. This value MUST be one of the five image formats: `image/jpeg`, `image/gif`, `image/bmp`, `image/png`, or `image/tiff`.

/ic:InformationCard/ic:Issuer

This required element provides a logical name for the issuer of the Information Card. If a Relying Party specifies a token issuer by its logical name, then the content of this element MUST be used to match the requested token issuer with an Information Card.

/ic:InformationCard/ic:TimeIssued

This required element provides the date and time when the Information Card was issued.

/ic:InformationCard/ic:TimeExpires

This optional element provides the date and time after which the Information Card SHOULD be treated as expired and invalid.

/ic:InformationCard/ic:TokenServiceList

This required element provides an ordered list of Security Token Service (IP/STS) endpoints, and corresponding credential descriptors (implying the REQUIRED authentication mechanisms), where tokens can be requested. Each service endpoint MUST be tried in order by the Service Requester when requesting tokens.

/ic:InformationCard/ic:SupportedTokenTypeList

This optional element contains the list of token types that are offered by the Identity Provider.

/ic:InformationCard/ic:SupportedClaimTypeList

This optional element contains the list of claim types that are offered by the Identity Provider.

/ic:InformationCard/ic:RequireAppliesTo

This optional element indicates that token requests MUST include information identifying the Relying Party where the issued token will be used. The Relying Party information MUST be included as the content of a `wsp:AppliesTo` element in the token request.

/ic:InformationCard/ic:PrivacyNotice

This optional element provides the location of the privacy statement of the Identity Provider.

/ic:InformationCard/ic07:RequireStrongRecipientIdentity

This optional element informs the Identity Selector that it MUST only allow the card to be used at a Relying Party that presents a cryptographically protected identity, for example, an X.509v3 certificate.

/ic:InformationCard/ic07:IssuerInformation

This optional element provides information from the card issuer about the card that can be displayed by the Identity Selector user interface.

.../ic:InformationCard/@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:InformationCard` element of an `ic:RoamingStore` when representing the card in the Information Cards Transfer Format in Section 6.1.

.../ic:InformationCard/{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:InformationCard` element of an `ic:RoamingStore` when representing the card in the Information Cards Transfer Format in Section 6.1.

3.1.1.1 Information Card Reference

Every Information Card issued by an Identity Provider MUST have a unique reference by which it can be identified within the scope of the Identity Provider. This reference is included in all token requests sent to the Identity Provider based on that Information Card.

The card reference MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:InformationCardReference xmlns:ic="...">
  <ic:CardId> xs:anyURI </ic:CardId>
  <ic:CardVersion> xs:unsignedInt </ic:CardVersion>
</ic:InformationCardReference>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:InformationCardReference

A specific reference for an Information Card.

.../ic:InformationCardReference/ic:CardId

This required element provides a unique identifier in the form of a URI for the specific Information Card. The identifier provider MUST be able to identify the specific Information Card based on this identifier.

.../ic:InformationCardReference/ic:CardVersion

This required element provides a versioning epoch for the Information Card issuance infrastructure used by the Identity Provider. The minimum value for this field MUST be 1. Note that it is possible to include version information in CardId as it is a URI, and can have hierarchical content. However, it is specified as a separate value to allow the Identity Provider to change its issuance infrastructure, and thus its versioning epoch, independently without changing the CardId of all issued Information Cards. For example, when an Identity Provider makes a change to the supported claim types or any other policy pertaining to the issued cards, the version number allows the Identity Provider to determine if the Information Card needs to be refreshed. The version number is assumed to be monotonically increasing. If two Information Cards have the same CardId value but different CardVersion values, then the one with a higher numerical CardVersion value SHOULD be treated as being more up-to-date.

3.1.1.2 Token Service Endpoints and Authentication Mechanisms

Every Information Card issued by an Identity Provider MUST include an ordered list of IP/STS endpoints, and the corresponding credential type to be used, for requesting tokens. The list MUST be in a decreasing order of preference. Identity Selectors SHOULD attempt to use the endpoints in the order listed, using the first endpoint in the list for which the metadata is retrievable and the endpoint is reachable. For each endpoint, the credential type implicitly determines the authentication mechanism to be used. Each credential descriptor is personalized for the user to allow an Identity Selector to automatically locate the credential once the user has selected an Information Card.

Further, each IP/STS endpoint reference in the Information Card MUST include the Security Policy metadata for that endpoint. The policy metadata MAY be specified as a metadata location within the IP/STS endpoint reference. If a metadata location URL is specified, it MUST use the [HTTPS] transport. An Identity Selector MAY retrieve the Security Policy it will use to communicate with the IP/STS from that metadata location using the mechanism specified in [WS-MetadataExchange].

The ordered list of token service endpoints MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:TokenServiceList xmlns:ic="..." xmlns:wsa="...">
 (<ic:TokenService>
   <wsa:EndpointReference> ... </wsa:EndpointReference>
   <ic:UserCredential>
    <ic:DisplayCredentialHint> xs:string </ic:DisplayCredentialHint> ?
   (
    <ic:UsernamePasswordCredential>...</ic:UsernamePasswordCredential> |
    <ic:KerberosV5Credential>...</ic:KerberosV5Credential> |
    <ic:X509V3Credential>...</ic:X509V3Credential> |
    <ic:SelfIssuedCredential>...</ic:SelfIssuedCredential> |
```

```
    )
   </ic:UserCredential>
  </ic:TokenService>) +
</ic:TokenServiceList>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:TokenServiceList

This required element provides an ordered list of Security Token Service endpoints (in decreasing order of preference), and the corresponding credential types, for requesting tokens. Each service endpoint MUST be tried in order by a Service Requester.

.../ic:TokenServiceList/ic:TokenService

This required element describes a single token issuing endpoint.

.../ic:TokenServiceList/ic:TokenService/wsa:EndpointReference

This required element provides the endpoint reference for a single token issuing endpoint. For the Self-issued Identity Provider, the special address value defined in Section 2.1.1 MAY be used. The `wsai:Identity` extension element (see Section 12) for endpoint references MAY be used to include the protection token for this endpoint to secure communications with it.

.../ic:TokenServiceList/ic:TokenService/ic:UserCredential

This required element indicates the credential type to use to authenticate to the token issuing endpoint.

.../ic:TokenServiceList/ic:TokenService/ic:UserCredential/ic:DisplayCredentialHint

This optional element provides a hint (string) to be displayed to the user to prompt for the correct credential (e.g. a hint to insert the right smart card). The content of this element MAY be localized in a specific language.

.../ic:TokenServiceList/ic:TokenService/ic:UserCredential/<credential descriptor>

This required element provides an unambiguous descriptor for the credential to use for authenticating to the token issuing endpoint. The schema to describe the credential is specific to each credential type. This profile defines the schema elements `ic:UsernamePasswordCredential`, `ic:KerberosV5Credential`, `ic:X509V3Credential` or `ic:SelfIssuedCredential` later in Section 4 corresponding to username/password, Kerberos v5, X.509v3 certificate and self-issued token based credential types. Other credential types MAY be introduced via the extensibility point defined in the schema within this element.

Alternatively, Identity Providers MAY publish metadata for Information Cards as WSDL documents that can be retrieved by Identity Selectors via HTTPS GET operations on URLs using the HTTPS scheme. An endpoint's metadata URL is communicated to Identity Selectors in a token service `wsx:MetadataReference` element in an Information Card using exactly the same syntax as when WS-MetadataExchange is employed to retrieve the metadata. No change occurs in the card.

The metadata documents published via HTTPS GET SHOULD contain the WSDL for the endpoint as the top-level element of the document without any SOAP or WS-MetadataExchange elements enclosing it.

Identity Providers MAY publish endpoint metadata via both the HTTPS GET and WS-MetadataExchange methods at the same metadata location. If they publish the metadata via multiple mechanisms, the metadata delivered via both mechanisms SHOULD be the same. Likewise, Identity Selectors MAY attempt to retrieve metadata via multiple mechanisms, either in sequence or in parallel.

The following example illustrates an Identity Provider with two endpoints for its IP/STS, one requiring Kerberos (higher priority) and the other requiring username/password (lower priority) as its authentication mechanism. Further, each endpoint also includes its policy metadata location as a URL using the [HTTPS] scheme.

Example:

```
<ic:TokenServiceList xmlns:ic="..." xmlns:wsa="..." xmlns:wsai="..."
    xmlns:wsx="...">
  <ic:TokenService>
    <wsa:EndpointReference>
      <wsa:Address>http://contoso.com/sts/kerb</wsa:Address>
      <wsai:Identity>
        <wsai:Spn>host/corp-sts.contoso.com</wsai:Spn>
      </wsai:Identity>
      <wsa:Metadata>
        <wsx:Metadata>
          <wsx:MetadataSection
              Dialect="http://schemas.xmlsoap.org/ws/2004/09/mex">
            <wsx:MetadataReference>
              <wsa:Address>https://contoso.com/sts/kerb/mex</wsa:Address>
            </wsx:MetadataReference>
          </wsx:MetadataSection>
        </wsx:Metadata>
      </wsa:Metadata>
    </wsa:EndpointReference>
    <ic:UserCredential>
      <ic:KerberosV5Credential />
    </ic:UserCredential>
  </ic:TokenService>
  <ic:TokenService>
    <wsa:EndpointReference>
      <wsa:Address>http://contoso.com/sts/pwd</wsa:Address>
      <wsa:Metadata>
        <wsx:Metadata>
          <wsx:MetadataSection
              Dialect="http://schemas.xmlsoap.org/ws/2004/09/mex">
            <wsx:MetadataReference>
              <wsa:Address>https://contoso.com/sts/pwd/mex</wsa:Address>
            </wsx:MetadataReference>
          </wsx:MetadataSection>
        </wsx:Metadata>
      </wsa:Metadata>
    </wsa:EndpointReference>
    <ic:UserCredential>
      <ic:UsernamePasswordCredential>
        <ic:Username>Zoe</ic:Username>
      </ic:UsernamePasswordCredential>
    </ic:UserCredential>
  </ic:TokenService>
</ic:TokenServiceList>
```

3.1.1.3 Token Types Offered

Every Information Card issued by an Identity Provider SHOULD include an unordered list of token types that can be issued by the Identity Provider. The set of token types offered by the Identity Provider MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:SupportedTokenTypeList xmlns:ic="..." xmlns:wst="...">
  <wst:TokenType> xs:anyURI </wst:TokenType> +
</ic:SupportedTokenTypeList>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:SupportedTokenTypeList

This optional element contains the set of token types offered by the Identity Provider.

.../ic:SupportedTokenTypeList/wst:TokenType

This required element indicates an individual token type that is offered.

The following example illustrates an Identity Provider that offers both SAML 1.1 and SAML 2.0 tokens.

Example:

```
<ic:SupportedTokenTypeList xmlns:ic="..." xmlns:wst="...">
  <wst:TokenType>urn:oasis:names:tc:SAML:1.0:assertion</wst:TokenType>
  <wst:TokenType>urn:oasis:names:tc:SAML:2.0:assertion</wst:TokenType>
</ic:SupportedTokenTypeList>
```

3.1.1.4 Claim Types Offered

Every Information Card issued by an Identity Provider SHOULD include an unordered list of claim types that can be issued by the Identity Provider. The set of claim types offered by the Identity Provider MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:SupportedClaimTypeList xmlns:ic="...">
  (<ic:SupportedClaimType Uri="xs:anyURI">
    <ic:DisplayTag> xs:string </ic:DisplayTag> ?
    <ic:Description> xs:string </ic:Description> ?
  </ic:SupportedClaimType>) +
</ic:SupportedClaimTypeList>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:SupportedClaimTypeList

This optional element contains the set of claim types offered by the Identity Provider.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType

This required element indicates an individual claim type that is offered.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType/@Uri

This required attribute provides the unique identifier (URI) of this individual claim type offered.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType/ic:DisplayTag

This optional element provides a friendly name for this individual claim type. The content of this element MAY be localized in a specific language.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType/ic:Description

This optional element provides a description of the semantics for this individual claim type. The content of this element MAY be localized in a specific language.

The following example illustrates an Identity Provider that offers two claim types.

Example:

```
<ic:SupportedClaimTypeList xmlns:ic="...">
  <ic:SupportedClaimType Uri=".../ws/2005/05/identity/claims/givenname">
    <ic:DisplayTag>Given Name</DisplayTag>
  </ic:SupportedClaimType>
  <ic:SupportedClaimType Uri=".../ws/2005/05/identity/claims/surname">
    <ic:DisplayTag>Last Name</DisplayTag>
  </ic:SupportedClaimType>
</ic:SupportedClaimTypeList>
```

3.1.1.5 Requiring Token Scope Information

An Identity Selector, by default, SHOULD NOT convey information about the Relying Party where an issued token will be used (i.e., target scope) when requesting Security Tokens. This helps safeguard user privacy. However, an Identity Provider MAY override that behavior.

Every Information Card issued by an Identity Provider MAY include a requirement that token requests include token scope information identifying the Relying Party where the token will be used. The requirement to submit token scope information MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:RequireAppliesTo Optional="xs:boolean" xmlns:ic="..." /> ?
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:RequireAppliesTo

This optional element indicates a requirement for a token requester to submit token scope information in the request. Absence of this element in an Information Card means that the token requester MUST NOT submit any token scope information.

.../ic:RequireAppliesTo/@Optional

This optional attribute indicates whether the token scope information is required or is optional by the Identity Provider. An attribute value of "true" indicates that the token scope information is not required, but will be accepted by the Identity Provider if submitted. An attribute value of "false" (default) indicates that the token scope information is required.

The following example illustrates the use of this element.

Example:

```
<ic:RequireAppliesTo Optional="true" xmlns:ic="..." />
```

If token scope information is required by an Identity Provider, an Identity Selector MUST include the Relying Party identity as the content of the `wsp:AppliesTo` element in the token request. The actual behavior of an Identity Selector vis-à-vis the possible requirements that can be expressed by the above element is specified in Section 3.3.3.

3.1.1.6 Privacy Policy Location

Every Information Card issued by an Identity Provider SHOULD include a pointer to the privacy statement of the Identity Provider. The location of the privacy statement MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:PrivacyNotice Version="xs:unsignedInt" xmlns:ic="..." /> ?
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:PrivacyNotice

This optional element provides the location of the privacy statement of the Identity Provider.

.../ic:PrivacyNotice/@Version

This optional attribute indicates a version number that tracks changes in the content of the privacy statement. This field MUST have a minimum value of 1 when present.

The following example illustrates the use of this element.

Example:

```
<ic:PrivacyNotice Version="1" xmlns:ic="...">
  http://www.contoso.com/privacynotice
</ic:PrivacyNotice>
```

An Identity Selector MUST be able to accept a privacy statement location specified as an URL using the [HTTP] scheme (as illustrated above) or the [HTTPS] scheme.

3.1.1.7 Prohibiting Use at Relying Parties Not Identified by a Cryptographically Protected Identity

Information Cards issuers MAY specify that a card MUST NOT be used at Relying Parties that do not present a cryptographically protected identity, such as an X.509v3 certificate. This would typically be done when the issuer determines that the use of HTTP without Message Security would not provide a sufficiently secure environment for the use of the card.

Syntax:

```
<ic07:RequireStrongRecipientIdentity xmlns:ic07="..." /> ?
```

.../ic07:RequireStrongRecipientIdentity

This optional element informs the Identity Selector that it MUST only allow the card to be used at a Relying Party that presents a cryptographically protected identity, such as an X.509v3 certificate.

3.1.1.8 Providing Custom Data to Display with the Card

Card issuers MAY supply a set of information about the card that MAY be displayed by the Identity Selector user interface.

Syntax:

```
<ic07:IssuerInformation xmlns:ic07="...">
  <ic07:IssuerInformationEntry>
    <ic07:EntryName> xs:string </ic07:EntryName>
    <ic07:EntryValue> xs:string </ic07:EntryValue>
  </ic07:IssuerInformationEntry> +
</ic07:IssuerInformation>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic07:IssuerInformation

This optional element provides a set of information from the card issuer about the card that can be displayed by the Identity Selector user interface.

.../ic07:IssuerInformation/IssuerInformationEntry

This required element provides one item of information about the card.

.../ic07:IssuerInformation/IssuerInformationEntry/EntryName

This required element provides the name of one item of information about the card.

.../ic07:IssuerInformation/IssuerInformationEntry/EntryValue

This required element provides the value of one item of information about the card.

The following example illustrates the use of this feature.

Example:

```
<ic07:IssuerInformation xmlns:ic07="...">
  <ic07:IssuerInformationEntry>
    <ic07:EntryName>Customer Service</ic07:EntryName>
    <ic07:EntryValue>+1-800-CONTOSO</ic07:EntryValue>
  </ic07:IssuerInformationEntry>
  <ic07:IssuerInformationEntry>
    <ic07:EntryName>E-mail Contact</ic07:EntryName>
    <ic07:EntryValue>cardhelp@contoso.com</ic07:EntryValue>
  </ic07:IssuerInformationEntry>
</ic07:IssuerInformation>
```

3.1.2 Issuing Information Cards

An Identity Provider can issue Information Cards to its users using any out-of-band mechanism that is mutually suitable.

In order to provide the assurance that an Information Card is indeed issued by the Identity Provider expected by the user, the Information Card MUST be carried inside a digitally signed envelope that is signed by the Identity Provider. For this, the "enveloping signature" construct (see [XMLDSIG]) MUST be used where the Information Card is included in the `ds:Object` element. The signature on the digitally signed envelope provides data origin authentication assuring the user that it came from the right Identity Provider.

The specific profile of XML digital signatures [XMLDSIG] that is RECOMMENDED for signing the envelope carrying the Information Card is as follows. Usage of other algorithms is not described.

- Use *enveloping signature* format when signing the Information Card XML document.
- Use a single `ds:Object` element within the signature to hold the `ic:InformationCard` element that represents the issued Information Card. The `ds:Object/@Id` attribute provides a convenient way for referencing the Information Card from the `ds:SignedInfo/ds:Reference` element within the signature.
- Use RSA signing and verification with the algorithm identifier given by the URI *http://www.w3.org/2000/09/xmldsig#rsa-sha1*.
- Use exclusive canonicalization with the algorithm identifier given by the URI *http://www.w3.org/2001/10/xml-exc-c14n#*.
- Use SHA1 digest method for the data elements being signed with the algorithm identifier *http://www.w3.org/2000/09/xmldsig#sha1*.
- There MUST NOT be any other transforms used in the enveloping signature for the Information Card other than the ones listed above.
- The `ds:KeyInfo` element MUST be present in the signature carrying the signing key information in the form of an X.509 v3 certificate or a X.509 v3 certificate chain specified as one or more `ds:X509Certificate` elements within a `ds:X509Data` element.

The following example shows an enveloping signature carrying an Information Card that is signed by the Identity Provider using the format outlined above. Note that whitespace (newline and space character) is included in the example only to improve readability; they might not be present in an actual implementation.

Example:

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    <SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <Reference URI="#_Object_InformationCard">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue> ... </DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue> ... </SignatureValue>
  <KeyInfo>
    <X509Data>
      <X509Certificate> ... </X509Certificate>
    </X509Data>
  </KeyInfo>
  <Object Id="_Object_InformationCard">
    <ic:InformationCard
        xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
        xml:lang="en-us">
      [Information Card content]
    </ic:InformationCard>
  </Object>
</Signature>
```

An Identity Selector MUST verify the enveloping signature. The `ic:InformationCard` element can then be extracted and stored in the Information Card collection.

3.2 Identity Provider Policy

This section specifies additional policy elements and requirements introduced by this profile for an IP/STS policy metadata.

3.2.1 Require Information Card Provisioning

In the Information Card Model, an Identity Provider requires provisioning in the form of an Information Card issued by it which represents the provisioned identity of the user. In order to enable an Identity Selector to learn that such pre-provisioning is necessary before token requests can be made, the Identity Provider MUST provide an indication in its policy.

An Identity Provider issuing Information Cards MUST specify this provisioning requirement in its policy using the following schema element.

Syntax:

```
<ic:RequireFederatedIdentityProvisioning xmlns:ic="..." />
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:RequireFederatedIdentityProvisioning

This element indicates a requirement that one or more Information Cards, representing identities that can be federated, MUST be pre-provisioned before token requests can be made to the Identity Provider.

The following example illustrates the use of this policy element.

Example:

```
<wsp:Policy xmlns:wsp="..." xmlns:ic="..." xmlns:sp="...">
  ...
  <ic:RequireFederatedIdentityProvisioning />
  <sp:SymmetricBinding>
    ...
  </sp:SymmetricBinding>
  ...
</wsp:Policy>
```

3.2.2 Policy Metadata Location

In the Information Card Model, an Identity Provider MUST make the Security Policy metadata for its IP/STS endpoints available. If a metadata location is used for this purpose, the location URL MUST use the [HTTPS] scheme. An Identity Selector MAY retrieve the Security Policy it will use to communicate with the IP/STS from that metadata location using the mechanism specified in [WS-MetadataExchange].

3.3 Token Request and Response

For any given Information Card, an Identity Selector can obtain a Security Token from the IP/STS for that Information Card. Tokens MUST be requested using the "Issuance Binding" mechanism described in [WS-

Trust 1.2] and [WS-Trust 1.3]. This section specifies additional constraints and extensions to the token request and response messages between the Identity Selector and the IP/STS.

The WS-Trust protocol requires that a token request be submitted by using the `wst:RequestSecurityToken` element in the request message, and that a token response be sent using the `wst:RequestSecurityTokenResponse` element in the response message. This profile refers to the "Request Security Token" message as RST and the "Request Security Token Response" message as RSTR in short.

The WS-Trust protocol allows for a token response to provide multiple tokens by using the `wst:RequestSecurityTokenResponseCollection` element in the response message. This profile, however, requires that an Identity Provider MUST NOT use the `wst:RequestSecurityTokenResponseCollection` element in the response. The token response MUST consist of a single `wst:RequestSecurityTokenResponse` element.

3.3.1 Information Card Reference

When requesting a Security Token from the IP/STS, an Identity Selector MUST include the Information Card reference in the body of the RST message as a top-level element information item. The `ic:InformationCardReference` element in the Information Card, including all of its [children], [attributes] and [in-scope namespaces], MUST be copied as an immediate child of the RST element in the message as follows.

The following example illustrates the Information Card reference included in a RST message.

Example:

```
<wst:RequestSecurityToken xmlns:wst="..." xmlns:ic="...">
  ...
  <ic:InformationCardReference>
    <ic:CardId>http://xyz.com/CardId/d795621fa01d454285f9</ic:CardId>
    <ic:CardVersion>1</ic:CardVersion>
  </ic:InformationCardReference>
  ...
</wst:RequestSecurityToken>
```

The IP/STS MAY fault with `ic:InformationCardRefreshRequired` to signal to the Service Requester that the Information Card needs to be refreshed.

3.3.2 Claims and Other Token Parameters

A Relying Party's requirements of claims and other token parameters are expressed in its policy using the `sp:RequestSecurityTokenTemplate` parameter within the `sp:IssuedToken` policy assertion (see Section 2.1). If all token parameters are acceptable to the Identity Selector, it MUST copy the content of this element (i.e. all of its [children] elements) into the body of the RST message as top-level element information items. However, if optional claims are requested by the Relying Party, requests for optional claims not selected by the user MUST NOT be copied into the RST message.

3.3.3 Token Scope

The WS-Trust protocol allows a token requester to indicate the target where the issued token will be used (i.e., token scope) by using the optional element `wsp:AppliesTo` in the RST message. By default, an Identity Selector SHOULD NOT send token scope information to the Identity Provider in token requests to protect user privacy. In other words, the element `wsp:AppliesTo` is absent in the RST message.

However, if the Identity Provider requires it (see the modes of the `ic:RequireAppliesTo` element described in Section 3.1.1.5), or if the Relying Party's token policy includes the `wsp:AppliesTo` element in the `sp:RequestSecurityTokenTemplate` parameter, then an Identity Selector MUST include token scope information in its token request as per the behavior summarized in the following table.

<RequireAppliesTo> mode in Information Card	*<AppliesTo> element present in RP policy*	*Resulting behavior of Identity Selector*
Mandatory	Yes	Send <AppliesTo> value from RP policy in token request to IP.
Mandatory	No	Send the RP endpoint to which token will be sent as the value of <AppliesTo> in token request to IP.
Optional	Yes	Send <AppliesTo> value from RP policy in token request to IP.
Optional	No	Do not send <AppliesTo> in token request to IP.
Not present	Yes	Fail
Not present	No	Do not send <AppliesTo> in token request to IP.

The following example illustrates the token scope information included in a RST message when it is sent to the Identity Provider.

Example:

```
<wst:RequestSecurityToken xmlns:wst="..." xmlns:wsp="..." xmlns:wsa="..."
    xmlns:wsai="..." xmlns:ds="...">
  <wsp:AppliesTo>
    <wsa:EndpointReference>
      <wsa:Address>http://ip.fabrikam.com</wsa:Address>
      <wsai:Identity>
        <ds:KeyInfo>
          <ds:X509Data>
            <ds:X509Certificate>...</ds:X509Certificate>
          </ds:X509Data>
        </ds:KeyInfo>
      </wsai:Identity>
    </wsa:EndpointReference>
  </wsp:AppliesTo>
  ...
</wst:RequestSecurityToken>
```

3.3.4 Client Pseudonym

A private personal identifier (PPID) claim, defined in Section 7.5.14, identifies a Subject to a Relying Party in a way such that a Subject's PPID at one Relying Party cannot be correlated with the Subject's PPID at another Relying Party. If an Identity Provider offers the PPID claim type then it MUST generate values for the claim that have this prescribed privacy characteristic using data present in the RST request.

When a Relying Party requests a PPID claim, an Identity Selector MUST provide a Client Pseudonym value via an `ic:PPID` element in the RST request that can be used by the IP/STS as input when computing the PPID claim value in the issued token. The Client Pseudonym SHOULD be produced as described in Section 3.3.4.1. It is RECOMMENDED that the IP/STS combine this Client Pseudonym value with information specific to the entity to which the card was issued as well as a secret known only by the IP/STS and pass the combination through a cryptographically non-invertible function, such as a cryptographic hash function, to generate the PPID claim value sent in the token. Alternatively, when target scope information is sent in the token request using the `wsp:AppliesTo` element, the IP/STS MAY instead choose to use that information to generate an appropriate PPID value.

When Client Pseudonym information is included by an Identity Selector in a token request, it MUST be sent using the following schema element.

Syntax:

```
<ic:ClientPseudonym xmlns:ic="...">
  <ic:PPID> xs:base64Binary </ic:PPID>
</ic:ClientPseudonym>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:ClientPseudonym

This optional top-level element contains the Client Pseudonym information item.

.../ic:ClientPseudonym/ic:PPID

This optional element contains the Client Pseudonym value that the client has submitted for use in computing the PPID claim value for the issued token. The IP/STS MAY use this value as the input (a seed) to a custom cryptographically non-invertible function, with the result used as the PPID claim value in the issued token.

The following example illustrates the Client Pseudonym information sent in a RST message.

Example:

```
<wst:RequestSecurityToken xmlns:wst="..." xmlns:ic="...">
  <ic:ClientPseudonym>
```

```
    <ic:PPID>...</ic:PPID>
  </ic:ClientPseudonym >
  ...
</wst:RequestSecurityToken>
```

When the target scope information is not sent in the token request to an IP/STS, the Identity Provider MUST NOT record any Client Pseudonym values included in the RST message. It likewise MUST NOT record the PPID claim value that it generates.

3.3.4.1 PPID

When a token request for a PPID claim is sent to an IP/STS, an Identity Selector SHOULD compute the Client Pseudonym PPID information it sends in the RST message as follows:

- Construct the *RP PPID Seed* as described in Section 7.6.1.
- Decode the base64 encoded value of the `ic:HashSalt` element of the Information Card (see Section 6.1) to obtain *SaltBytes*.
- Decode the base64 encoded value of the `ic:MasterKey` element of the Information Card (see Section 6.1) to obtain *MasterKeyBytes*.
- Hash the concatenation of *MasterKeyBytes*, *RP PPID Seed*, and *SaltBytes* using the SHA256 hash function to obtain the Client Pseudonym PPID value.
  Client Pseudonym PPID = SHA256 (MasterKeyBytes + RP PPID Seed + SaltBytes)
- Convert *Client Pseudonym PPID* to a base64 encoded string and send as the value of the `ic:PPID` element in the RST request.

3.3.5 Proof Key for Issued Token

An issued token can have a *symmetric* proof key (symmetric key token), an *asymmetric* proof key (asymmetric key token), or *no* proof key (bearer token). If no key type is specified in the Relying Party policy, then an Identity Selector SHOULD request an asymmetric key token from the IP/STS by default.

The optional `wst:KeyType` element in the RST request indicates the type of proof key desired in the issued Security Token. The IP/STS MAY return the proof key and/or entropy towards the proof key in the RSTR response. This section describes the behaviors for how each proof key type is requested, who contributes entropy, and how the proof key is computed and returned.

3.3.5.1 Symmetric Proof Key

When requesting a symmetric key token, an Identity Selector MUST submit entropy towards the proof key by augmenting the RST request message as follows:

- The RST SHOULD include a `wst:KeyType` element with one of the two following URI values, depending upon the version of WS-Trust being used:
  http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey
  http://docs.oasis-open.org/ws-sx/ws-trust/200512/SymmetricKey
- The RST MUST include a `wst:BinarySecret` element inside a `wst:Entropy` element containing client-side entropy to be used as partial key material. The entropy is conveyed as raw base64 encoded bits.

The size of the submitted entropy SHOULD be equal to the key size requested in the Relying Party policy. If no key size is specified by the Relying Party, then an Identity Selector SHOULD request a key at least 256-bits in size, and submit an entropy of equal size to the IP/STS.

Following is a sample RST request fragment that illustrates a symmetric key token request.

Example:

```
<wst:RequestSecurityToken xmlns:wst="...">
  ...
  <wst:KeyType>
    http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey
  </wst:KeyType>
  <wst:KeySize>256</wst:KeySize>
  <wst:Entropy>
    <wst:BinarySecret>mQlxWxEiKOcUfnHgQpylcD7LYSkJplpE=</wst:BinarySecret>
  </wst:Entropy>
</wst:RequestSecurityToken>
```

When processing the token request, the IP/STS MAY:

a) accept the client entropy as the sole key material for the proof key,

b) accept the client entropy as partial key material and contribute additional server-side entropy as partial key material to compute the proof key as a function of both partial key materials, or

c) reject the client-side entropy and use server-side entropy as the sole key material for the proof key.

For each of the cases above, the IP/STS MUST compute and return the proof key by augmenting the RSTR response message as follows.

For case (a) where IP/STS accepts client entropy as the sole key material:

- The RSTR MUST NOT include a `wst:RequestedProofToken` element. The proof key is implied and an Identity Selector MUST use the client-side entropy as the proof key.

For case (b) where IP/STS accepts client entropy and contributes additional server entropy:

- The RSTR MUST include a `wst:BinarySecret` element inside a `wst:Entropy` element containing the server-side entropy to be used as partial key material. The entropy is conveyed as raw base64 encoded bits.
- The partial key material from the IP/STS MUST be combined (by each party) with the partial key material from the client to determine the resulting proof key.
- The RSTR MUST include a `wst:RequestedProofToken` element containing a `wst:ComputedKey` element to indicate how the proof key is to be computed. It is RECOMMENDED that an Identity Selector support the P_SHA1 computed key mechanism defined in [WS-Trust 1.2] or [WS-Trust 1.3] with the particulars below. Usage of other algorithms is not described.

ComputedKey Value	*Meaning*
http://schemas.xmlsoap.org/ws/2005/02/trust/CK/PSHA1 or http://docs.oasis-open.org/ws-sx/ws-trust/200512/CK/PSHA1	The key is computed using P_SHA1 from the TLS specification to generate a bit stream using entropy from both sides. The exact form is: key = P_SHA1 ($Entropy_{REQ}$, $Entropy_{RES}$)

Following is a sample RSTR response fragment that illustrates a token response with partial key material from the IP/STS and a computed proof key.

Example:

```
<wst:RequestSecurityTokenResponse xmlns:wst="...">
  ...
  <wst:Entropy>
    <wst:BinarySecret>mQlxWxEiKOcUfnHgQpylcD7LYSkJplpE=</wst:BinarySecret>
  </wst:Entropy>
  <wst:RequestedProofToken>
    <wst:ComputedKey>
      http://schemas.xmlsoap.org/ws/2005/02/trust/CK/PSHA1
    </wst:ComputedKey>
  </wst:RequestedProofToken>
</wst:RequestSecurityTokenResponse>
```

For case (c) where IP/STS contributes server entropy as the sole key material:

- The RSTR MUST include a `wst:BinarySecret` element inside a `wst:RequestedProofToken` element containing the specific proof key to be used. The proof key is conveyed as raw base64 encoded bits.

Following is a sample RSTR response fragment that illustrates a token response with fully specified proof key from the IP/STS.

Example:

```
<wst:RequestSecurityTokenResponse xmlns:wst="...">
  ...
  <wst:RequestedProofToken>
    <wst:BinarySecret>
      mQlxWxEiKOcUfnHgQpylcDKOcUfnHg7LYSkJplpE=
    </wst:BinarySecret>
  </wst:RequestedProofToken>
</wst:RequestSecurityTokenResponse>
```

The following table summarizes the symmetric proof key computation rules to be used by an Identity Selector:

Token Requester (Identity	*Token Issuer (IP/STS)*	*Results*

Selector)		
Provides entropy	Uses requester entropy as proof key	No <wst:RequestedProofToken> element present in RSTR. Proof key is implied.
Provides entropy	Uses requester entropy and provides additional entropy of its own	<wst:Entropy> element present in RSTR containing issuer supplied entropy. <wst:RequestedProofToken> element present in RSTR containing computed key mechanism. Requestor and Issuer compute proof key by combining both entropies using the specified computed key mechanism.
Provides entropy	Uses own entropy as proof key (rejects requester entropy)	<wst:RequestedProofToken> element present in RSTR containing the proof key.

3.3.5.2 Asymmetric Proof Key

When requesting an asymmetric key token, it is RECOMMENDED that an Identity Selector generate an ephemeral RSA key pair. Usage of other algorithms is not described. The generated RSA key pair MUST be at least 1024-bits in size for use as the proof key. It MUST submit the public key to the IP/STS by augmenting the RST request as follows:

- The RST MUST include a `wst:KeyType` element with one of the two following URI values, depending upon the version of WS-Trust being used:

  http://schemas.xmlsoap.org/ws/2005/02/trust/PublicKey

  http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey
- The RST SOAP body MUST include a `wst:UseKey` element containing the public key to be used as proof key in the returned token. The public key is present as a raw RSA key in the form of a `ds:RSAKeyValue` element inside a `ds:KeyValue` element.
- The RST SOAP security header SHOULD include a supporting signature to prove ownership of the corresponding private key. The `ds:KeyInfo` element within the signature, if present, MUST include the same public key as in the `wst:UseKey` element in the SOAP body.
- The supporting signature, if present, MUST be placed in the SOAP security header where the signature for an endorsing supporting token would be placed as per the security header layout specified in WS-SecurityPolicy.

Following is a sample RST request fragment that illustrates an asymmetric key based token request containing the public key and proof of ownership of the corresponding private key.

Example:

```
<s:Envelope xmlns:s="..." xmlns:wsse="..." xmlns:ds="..." xmlns:wst="..."
    ... >
  <s:Header>
    ...
    <wsse:Security>
      ...
      <ds:Signature Id="_proofSignature">
        <!-- signature proving possession of submitted proof key -->
        ...
        <!-- KeyInfo in signature contains the submitted proof key -->
        <ds:KeyInfo>
          <ds:KeyValue>
            <ds:RSAKeyValue>
              <ds:Modulus>...</ds:Modulus>
              <ds:Exponent>...</ds:Exponent>
            </ds:RSAKeyValue>
          </ds:KeyValue>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </s:Header>
  <s:Body wsu:Id="req">
    <wst:RequestSecurityToken>
      ...
      <wst:KeyType>
        http://schemas.xmlsoap.org/ws/2005/02/trust/PublicKey
      </wst:KeyType>
      <wst:UseKey Sig="#_proofSignature">
        <ds:KeyInfo>
          <ds:KeyValue>
            <ds:RSAKeyValue>
              <ds:Modulus>...</ds:Modulus>
              <ds:Exponent>...</ds:Exponent>
            </ds:RSAKeyValue>
          </ds:KeyValue>
        </ds:KeyInfo>
      </wst:UseKey>
    </wst:RequestSecurityToken>
  </s:Body>
</s:Envelope>
```

If a supporting signature for the submitted proof key is not present in the token request, the IP/STS MAY fail the request. If a supporting signature is present, the IP/STS MUST verify the signature and MUST ensure that the public key included in the `wst:UseKey` element and in the supporting signature are the same. If verification succeeds and the IP/STS accepts the submitted public key for use in the issued token, then the token response MUST NOT include a `wst:RequestedProofToken` element. The proof key is implied and an Identity Selector MUST use the public key it submitted as the proof key.

The following table summarizes the asymmetric proof key rules used by an Identity Selector:

Token Requester (Identity Selector)	*Token Issuer (IP/STS)*	*Results*
Provides ephemeral public key for use as proof key	Uses requester supplied proof key	No <wst:RequestedProofToken> element present in RSTR. Proof key is implied.

3.3.5.3 No Proof Key

When requesting a token with no proof key, an Identity Selector MUST augment the RST request message as follows:

- The RST MUST include a `wst:KeyType` element with the following URI value if [WS-Trust 1.2] is being used:

  http://schemas.xmlsoap.org/ws/2005/05/identity/NoProofKey

  or the RST MUST include a wst:KeyType element with the following URI value if [WS-Trust 1.3] is being used:

  http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer

Following is a sample RST request fragment that illustrates a bearer token request.

Example:

```
<wst:RequestSecurityToken xmlns:wst="...">
  ...
  <wst:KeyType>
    http://schemas.xmlsoap.org/ws/2005/05/identity/NoProofKey
  </wst:KeyType>
</wst:RequestSecurityToken>
```

When processing the token request, if the IP/STS issues a SAML v1.1 bearer token then:

- It MUST specify "urn:oasis:names:tc:SAML:1.0:cm:bearer" as the subject confirmation method in the token.
- It SHOULD include a `saml:AudienceRestrictionCondition` element restricting the token to the target site URI submitted in the token request.

- It SHOULD include a saml:AudienceRestrictionCondition element restricting the token to the target site URL submitted in the token request.

3.3.6 Display Token

An Identity Selector MAY request a Display Token – a representation of the claims carried in the issued Security Token that can be displayed in an user interface – from an IP/STS as part of the token request. To request a Display Token, the following element MUST be included in the RST message as a top-level element information item.

Syntax:

```
<ic:RequestDisplayToken xml:lang="xs:language"? xmlns:ic="..." ... />
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:RequestDisplayToken

This optional element is used to request an Identity Provider to return a Display Token corresponding to the issued token.

/ic:RequestDisplayToken/@xml:lang

This optional attribute indicates a language identifier, using the language codes specified in [RFC 3066], in which the Display Token content SHOULD be localized.

An IP/STS MAY respond to a Display Token request. If it does, it MUST use the following element to return a Display Token for the issued Security Token in the RSTR message.

Syntax:

```
<ic:RequestedDisplayToken xmlns:ic="..." ...>
  <ic:DisplayToken xml:lang="xs:language" ... >
    [ <ic:DisplayClaim Uri="xs:anyURI" ...>
        <ic:DisplayTag> xs:string </ic:DisplayTag> ?
        <ic:Description> xs:string </ic:Description> ?
        <ic:DisplayValue> xs:string </ic:DisplayValue> ?
      </ic:DisplayClaim> ] +
    |
    [ <ic:DisplayTokenText MimeType="xs:string">
        xs:string
      </ic:DisplayTokenText> ]
    ...
  </ic:DisplayToken>
</ic:RequestedDisplayToken>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:RequestedDisplayToken

This optional element is used to return a Display Token for the Security Token returned in the response.

/ic:RequestedDisplayToken/ic:DisplayToken

The returned Display Token.

/ic:RequestedDisplayToken/ic:DisplayToken/@xml:lang

This required attribute indicates a language identifier, using the language codes specified in [RFC 3066], in which the Display Token content is localized.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim

This required element indicates an individual claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/@Uri

This required attribute provides the unique identifier (URI) of the individual claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/ic:DisplayTag

This optional element provides a friendly name for the claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/ic:Description

This optional element provides a description of the semantics for the claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/ic:DisplayValue

This optional element provides the displayable value for the claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayTokenText

This optional element provides an alternative textual representation of the entire token as a whole when the token content is not suitable for display as individual claims.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayTokenText/@MimeType

This required attribute provides a MIME type specifying the format of the Display Token content (e.g., "text/plain").

The following example illustrates a returned Display Token corresponding to a Security Token with two claims.

Example:

```
<ic:RequestedDisplayToken xmlns:ic="...">
  <ic:DisplayToken xml:lang="en-us">
    <ic:DisplayClaim Uri="http://.../ws/2005/05/identity/claims/givenname">
      <ic:DisplayTag>Given Name</ic:DisplayTag>
      <ic:DisplayValue>John</ic:DisplayValue>
    </ic:DisplayClaim>
    <ic:DisplayClaim Uri="http://.../ws/2005/05/identity/claims/surname">
      <ic:DisplayTag>Last Name</ic:DisplayTag>
      <ic:DisplayValue>Doe</ic:DisplayValue>
    </ic:DisplayClaim>
  </ic:DisplayToken>
</ic:RequestedDisplayToken>
```

3.3.7 Token References

When an IP/STS returns the token requested by an Identity Selector, it MUST also include an attached and an un-attached token reference for the issued security token using the `wst:RequestedAttachedReference` and `wst:RequestedUnattachedReference` elements, respectively, in the RSTR response message.

An Identity Selector is truly a conduit for the security tokens issued by an IP/STS and requested by an RP, and it should remain agnostic of the type of the security token passing through it. Furthermore, a security token issued by an IP/STS MAY be encrypted directly for the RP, thus preventing visibility into the token by the Identity Selector. However, an Identity Selector (or a client application) needs to be able to use the issued security token to perform security operations (such as signature or encryption) on a message sent to an RP and thus needs a way to reference the token both when it is attached to a message and when it is not. The attached and unattached token references returned by an IP/STS in the RSTR message provide the necessary references that can be used for this purpose.

4 Authenticating to Identity Provider

The Information Card schema includes the element content necessary for an Identity Provider to express what credential the user must use in order to authenticate to the IP/STS when requesting tokens. This section defines the schema used to express the credential descriptor for each supported credential type.

4.1 Username and Password Credential

When the Identity Provider requires a *username* and *password* as the credential type, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential xmlns:ic="...">
  <ic:UsernamePasswordCredential>
    <ic:Username> xs:string </ic:Username> ?
  </ic:UsernamePasswordCredential>
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:UsernamePasswordCredential

This element indicates that a username/password credential is needed.

.../ic:UsernamePasswordCredential/ic:Username

This optional element provides the username part of the credential for convenience. An Identity Selector MUST prompt the user for the password. If the username is specified, then its value MUST be copied into the username token used to authenticate to the IP/STS; else an Identity Selector MUST prompt the user for the username as well.

Furthermore, the actual Security Policy of the IP/STS (expressed in its WSDL) MUST include the `sp:UsernameToken` assertion requiring a username and password value.

4.2 Kerberos v5 Credential

When the Identity Provider requires a *Kerberos v5 service ticket* for the IP/STS as the credential type, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential xmlns:ic="...">
  <ic:KerberosV5Credential />
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:KerberosV5Credential

This element indicates that a Kerberos v5 credential is needed.

To enable the Service Requester to obtain a Kerberos v5 service ticket for the IP/STS, the endpoint reference of the IP/STS in the Information Card or in the metadata retrieved from it MUST include a "service principal name" identity claim (i.e. a `wsai:Spn` element) under the `wsai:Identity` tag as defined in Section 12.

Furthermore, the actual Security Policy of the IP/STS (expressed in its WSDL) MUST include the `sp:KerberosToken` assertion requiring a Kerberos service ticket.

4.3 X.509v3 Certificate Credential

When the Identity Provider requires an *X.509 v3 certificate* for the user as the credential type, where the certificate and keys are in a hardware-based smart card or a software-based certificate, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential xmlns:ic="..." xmlns:ds="..." xmlns:wsse="...">
  <ic:DisplayCredentialHint> xs:string </ic:DisplayCredentialHint>
  <ic:X509V3Credential>
    <ds:X509Data>
      <wsse:KeyIdentifier
        ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#ThumbPrintSHA1"
        EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary">
        xs:base64binary
      </wsse:KeyIdentifier>
    </ds:X509Data>
  </ic:X509V3Credential>
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:DisplayCredentialHint

This optional element provides a user hint string which can be used to prompt the user, for example, to insert the appropriate smart card into the reader.

.../ic:X509V3Credential

This element indicates that a X.509 certificate credential is needed.

.../ic:X509V3Credential/ds:X509Data/wsse:KeyIdentifier

This element provides a key identifier for the X.509 certificate based on the SHA1 hash of the entire certificate content expressed as a "thumbprint." Note that the extensibility point in the `ds:X509Data` element is used to add `wsse:KeyIdentifier` as a child element.

Furthermore, the actual Security Policy of the IP/STS, expressed in its WSDL, MUST include the `sp:X509Token` assertion requiring an X.509v3 certificate.

4.4 Self-issued Token Credential

When the Identity Provider requires a *self-issued token* as the credential type, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential xmlns:ic="...">
  <ic:SelfIssuedCredential>
    <ic:PrivatePersonalIdentifier>
      xs:base64Binary
    </ic:PrivatePersonalIdentifier>
  </ic:SelfIssuedCredential>
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:SelfIssuedCredential

This element indicates that a self-issued token credential is needed.

.../ic:SelfIssuedCredential/ic:PrivatePersonalIdentifier

This required element provides the value of the PPID claim asserted in the self-issued token used previously to register with the IP/STS (see Section 7.5.14).

Furthermore, the actual Security Policy of the IP/STS (expressed in its WSDL) MUST include the `sp:IssuedToken` assertion requiring a self-issued token with exactly one claim, namely, the PPID.

5 Faults

In addition to the standard faults described in WS-Addressing, WS-Security and WS-Trust, this profile defines the following additional faults that MAY occur when interacting with an RP or an IP. The binding of the fault properties (listed below) to a SOAP 1.1 or SOAP 1.2 fault message is described in [WS-Addressing]. If the optional **[Detail]** property for a fault includes any specified content, then the corresponding schema fragment is included in the listing below.

5.1 Relying Party

The following faults MAY occur when submitting Security Tokens to an RP per its Security Policy.

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	*S:Sender*
[Subcode]	ic:RequiredClaimMissing
[Reason]	A required claim is missing from the Security Token.
[Detail]	[URI of missing claim] `<ic:ClaimType Uri="[Claim URI]" />`

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:InvalidClaimValue
[Reason]	A claim value asserted in the Security Token is invalid.
[Detail]	[URI of invalid claim] `<ic:ClaimType Uri="[Claim URI]" />`

5.2 Identity Provider

The following faults MAY occur when requesting Security Tokens from an IP using Information Cards.

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender

[Subcode]	ic:MissingAppliesTo
[Reason]	The request is missing Relying Party identity information.
[Detail]	(None defined.)

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:InvalidProofKey
[Reason]	Invalid proof key specified in request.
[Detail]	(None defined.)

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:UnknownInformationCardReference
[Reason]	Unknown Information Card reference specified in request.
[Detail]	[Unknown Information Card reference] `<ic:InformationCardReference>` `<ic:CardId>[card ID]</ic:CardId>` `<ic:CardVersion>[version]</ic:CardVersion>` `</ic:InformationCardReference>`

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:FailedRequiredClaims
[Reason]	Could not satisfy required claims in request; construction of token failed
[Detail]	[URIs of claims that could not be satisfied] `<ic:ClaimType Uri="[Claim URI]" />` `...`

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:InformationCardRefreshRequired
[Reason]	Stale Information Card reference specified in request; Information Card SHOULD be refreshed
[Detail]	[Information Card reference that needs refreshing] `<ic:InformationCardReference>` `<ic:CardId>[card ID]</ic:CardId>` `<ic:CardVersion>[version]</ic:CardVersion>` `</ic:InformationCardReference>`

5.2.1 Identity Provider Custom Error Messages

Identity Providers MAY return custom error messages to Identity Selectors via SOAP faults that can be displayed by the Identity Selector user interface. The error message MUST be communicated as an `S:Text` element within the `S:Reason` element of a SOAP fault message. Multiple `S:Text` elements MAY be returned with different `xml:lang` values and the Identity Selector SHOULD use the one matching the user's locale, if possible.

Example:

```
<s:Envelope xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:s="http://www.w3.org/2003/05/soap-envelope">
  <s:Header>
    <wsa:Action s:mustUnderstand="1">
      http://www.w3.org/2005/08/addressing/soap/fault
    </wsa:Action>
  </s:Header>
  <s:Body>
    <s:Fault>
      <s:Code>
        <s:Value>s:Sender</s:Value>
      </s:Code>
      <s:Reason>
        <s:Text xml:lang="en">Message in English ...</s:Text>
        <s:Text xml:lang="es-ES">Message in the Spanish of Spain ...</s:Text>
      </s:Reason>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

6 Information Cards Transfer Format

This section defines how collections of Information Cards are transferred between Identity Selectors. The cards collection is always transferred after encrypting it with a key derived from a user specified password. Section 6.1 describes the transfer format of the collection in the clear, whereas Section 6.1.2 describes the transfer format after the necessary encryption is applied.

6.1 Pre-Encryption Transfer Format

Each Information Card in the transfer stream will contain metadata and key material maintained by the originating Identity Selector in addition to the original Information Card metadata. If an Identity Selector includes a co-resident Self-issued Identity Provider (described in Section 7), an exported self-issued card MAY also contain any associated claims information.

The XML schema used for the transfer format is defined below:

Syntax:

```
<ic:RoamingStore xmlns:ic="...">
  <ic:RoamingInformationCard> +
    <ic:InformationCardMetaData>
      [Information Card]
      <ic:IsSelfIssued> xs:boolean </ic:IsSelfIssued>
      <ic:PinDigest> xs:base64Binary </ic:PinDigest> ?
      <ic:HashSalt> xs:base64Binary </ic:HashSalt>
      <ic:TimeLastUpdated> xs:dateTime </ic:TimeLastUpdated>
      <ic:IssuerId> xs:base64Binary </ic:IssuerId>
      <ic:IssuerName> xs:string </ic:IssuerName>
      <ic:BackgroundColor> xs:int </ic:BackgroundColor>
```

```
    </ic:InformationCardMetaData>
    <ic:InformationCardPrivateData> ?
      <ic:MasterKey> xs:base64Binary </ic:MasterKey>
      <ic:ClaimValueList> ?
        <ic:ClaimValue Uri="xs:anyURI" ...> +
          <ic:Value> xs:string </ic:Value>
        </ic:ClaimValue>
      </ic:ClaimValueList>
    </ic:InformationCardPrivateData>
    ...
  </ic:RoamingInformationCard>
  ...
</ic:RoamingStore>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:RoamingStore

The collection of Information Cards selected for transfer.

/ic:RoamingStore/ic:RoamingInformationCard (one or more)

An individual Information Card within the transfer stream.

For brevity, the prefix string "/ic:RoamingStore/ic:RoamingInformationCard" in the element names below is shortened to "...".

.../ic:InformationCardMetaData

This required element contains the metadata for an Information Card.

.../ic:InformationCardMetaData/{Information Card}

The original content of the Information Card as issued by the Identity Provider (described in Section 3.1.1).

.../ic:InformationCardMetaData/ic:IsSelfIssued

This required element indicates if the card is self-issued ("true") or not ("false").

.../ic:InformationCardMetaData/ic:PinDigest

This optional element contains a digest of the user-specified PIN information if the card is PIN-protected. The digest contains the base64 encoded bytes of the SHA1 hash of the bytes of the user-specified PIN represented using Unicode encoding UTF-16LE with no byte order mark. Usage of other algorithms is not described.

.../ic:InformationCardMetaData/ic:HashSalt

This optional element contains a random per-card entropy value used for computing the Relying Party specific PPID claim when the card is used at a Relying Party and for computing the Client Pseudonym PPID value sent an Identity Provider.

.../ic:InformationCardMetaData/ic:TimeLastUpdated

This required element contains the date and time when the card was last updated.

.../ic:InformationCardMetaData/ic:IssuerId

This required element contains an identifier for the Identity Provider with which a self-issued credential descriptor in a card issued by that Identity Provider can be resolved to the correct self-issued card. The element content SHOULD be the empty string for self-issued cards.

.../ic:InformationCardMetaData/ic:IssuerName

This required element contains a friendly name of the card issuer.

.../ic:InformationCardMetaData/ic:BackgroundColor

This required element contains the background color used to display the card image. This value is a 3-byte RGB color value in the sRGB color space used by HTML.

.../ic:InformationCardMetaData/{any}

This is an extensibility point to allow additional metadata to be included.

.../ic:InformationCardPrivateData

This required element contains the private data for an Information Card.

.../ic:InformationCardPrivateData/ic:MasterKey

This required element contains a base64 encoded 256-bit random number that provides a "secret key" for the Information Card. This key is used for computing the Relying Party specific PPID claim when the card is used at a Relying Party and for computing the Client Pseudonym PPID value sent to an Identity Provider. This element is present both for self-issued and managed Information Cards.

.../ic:InformationCardPrivateData/ic:ClaimValueList

This optional element is a container for the set of claim types and their corresponding values embodied by a self-issued card.

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue (one or more)

This required element is a container for an individual claim, *i.e.*, a claim type and its corresponding value.

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue/@Uri

This required attribute contains a URI that identifies the specific claim type.

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue/ic:Value

This required element contains the value for an individual claim type.

.../@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore/ic:RoamingInformationCard` element when updating information using the Information Cards Transfer Format.

.../{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore/ic:RoamingInformationCard` element when updating information using the Information Cards Transfer Format.

/ic:RoamingStore/@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore` element when updating information using the Information Cards Transfer Format.

/ic:RoamingStore/{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore` element when updating information using the Information Cards Transfer Format.

6.1.1 PIN Protected Card

When an Information Card is PIN protected, in addition to storing a digest of the PIN in the card data, the master key and claim values associated with the card MUST also be encrypted with a key derived from the user-specified PIN.

It is RECOMMENDED that the PKCS-5 based key derivation method be used with the input parameters summarized in the table below for deriving the encryption key from the PIN. Usage of other algorithms is not described.

Key derivation method	PBKDF1 per [RFC 2898] (Section 5.1)
Input parameters:	
Password	UTF-8 encoded octets of PIN
Salt	16-byte random number (actual value stored along with master key)
Iteration count	1000 (actual value stored along with master key)
Key length	32 octets
Hash function	SHA-256

The encryption method and the corresponding parameters that MUST be used are summarized in the table below.

Encryption method	AES-256

Parameters:	
Padding	As per PKCS-7 standard
Mode	CBC
Block size	16 bytes (as REQUIRED by AES)

In a PIN-protected card, the encrypted content of the master key and the claim value fields are described below.

.../ic:InformationCardPrivateData/ic:MasterKey

This element MUST contain a base64 encoded byte array comprised of the encryption parameters and the encrypted master key serialized as per the binary structure summarized in the table below.

Field	*Offset*	*Size (bytes)*
Version (for internal use)	0	1
Salt used for key-derivation method	1	16
Iteration count used for key-derivation method	17	4
Initialization Vector (IV) used for encryption	21	16
Encrypted master key	37	master key length

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue/ic:Value

This element MUST contain a base64 encoded byte array comprised of the encrypted claim value. The encryption parameters used are taken from those serialized into the master key field and summarized in the table above.

6.1.2 Computing the ic:IssuerId

The `ic:IssuerId` value used for a card when representing it in the Information Cards Transfer Format SHOULD be computed as a function of the `ds:KeyInfo` field of the envelope digitally signed by the Identity Provider. Specifically:

- Compute *IP PPID Seed* in the same manner as *RP PPID Seed* in Section 7.6.1, except that the certificate from `ds:KeyInfo` is used, rather than the Relying Party's.

Use the *IP PPID Seed* as the `ic:IssuerId` value.

The `ic:IssuerId` value SHOULD be the empty string for self-issued cards.

6.1.3 Computing the ic:IssuerName

The `ic:IssuerName` value used for a card when representing it in the Information Cards Transfer Format SHOULD be computed as a function of the `ds:KeyInfo` field of the envelope digitally signed by the Identity Provider. Specifically, if the certificate from `ds:KeyInfo` is an extended validation (EV) certificate [EV Cert], then set `ic:IssuerName` to the organizationName (O) field value from the certificate, otherwise set `ic:IssuerName` to the commonName (CN) field value from the certificate.

6.1.4 Creating the ic:HashSalt

A random `ic:HashSalt` value for a card SHOULD be created by the Identity Selector when that card is created from the `ic:InformationCard` data provided by an Identity Provider.

6.2 Post-Encryption Transfer Format

The transfer stream MUST be encrypted with a key derived from a user specified password. The XML schema used for the encrypted transfer stream is defined below:

Syntax:

```
Byte-order-mark
<?xml version="1.0" encoding="utf-8"?>
<ic:EncryptedStore xmlns:ic="..." xmlns:xenc="...">
  <ic:StoreSalt> xs:base64Binary </ic:StoreSalt>
  <xenc:EncryptedData>
    <xenc:CipherData>
      <xenc:CipherValue> ... </xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>
</ic:EncryptedStore>
...
```

The following describes the elements listed in the XML schema outlined above:

Byte-order-mark

The first three bytes in the stream containing the values {0xEF, 0xBB, 0xBF} constitutes a "byte order mark".

/ic:EncryptedStore

The top-level container element for the encrypted transfer stream.

/ic:EncryptedStore/ic:StoreSalt

This required element contains the random salt used as a parameter for the key derivation function to derive the encryption key from a user-specified password.

/ic:EncryptedStore/xenc:EncryptedData/xenc:CipherData/xenc:CipherValue

This element contains a base64 encoded byte array containing the ciphertext corresponding to the clear text transfer stream described in Section 6.1.

@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them when updating information using the Information Cards Transfer Format.

{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them when updating information using the Information Cards Transfer Format.

The remainder of this section describes the element content of the *xenc:CipherValue* element in the schema outline above. Specifically, it describes the encryption method used and the format of the encrypted content.

The following table defines two symbolic constants, namely *EncryptionKeySalt* and *IntegrityKeySalt*, and their corresponding values used by the key derivation and the encryption methods described below to encrypt the transfer stream.

EncryptionKeySalt	{ 0xd9, 0x59, 0x7b, 0x26, 0x1e, 0xd8, 0xb3, 0x44, 0x93, 0x23, 0xb3, 0x96, 0x85, 0xde, 0x95, 0xfc }
IntegrityKeySalt	{ 0xc4, 0x01, 0x7b, 0xf1, 0x6b, 0xad, 0x2f, 0x42, 0xaf, 0xf4, 0x97, 0x7d, 0x4, 0x68, 0x3, 0xdb }

The transfer stream content is encrypted with a key derived from a user-specified password. It is RECOMMENDED that the PKCS-5 based key derivation method be used with the input parameters summarized in the table below for deriving the key from the password. Usage of other algorithms is not described.

Key derivation method	PBKDF1 per [RFC 2898] (Section 5.1)
Input parameters:	
Password	UTF-8 encoded octets of user-specified password
Salt	16-byte random number (actual value stored in the *ic:StoreSalt* field)
Iteration count	1000
Key length	32 octets
Hash function	SHA-256

The PKCS-5 key derived as per the preceding table MUST be further hashed with a 16-byte salt using the SHA256 hash function, and the resulting value used as the encryption key. The order in which the values used MUST be hashed is as follows:

Encryption Key = SHA256 (EncryptionKeySalt + PKCS5-derived-key)

Further, to provide an additional integrity check at the time of import, a "hashed integrity code" MUST be computed as follows and included along with the encrypted transfer stream content.

- The PKCS-5 key derived as per the preceding table MUST be further hashed with a 16-byte salt using the SHA256 hash function, and the resulting value used as the integrity key. The order in which the values used MUST be hashed is as follows:

  Integrity Key = SHA256 (IntegrityKeySalt + PKCS5-derived-key)

- The last block of the clear text transfer stream MUST be captured and further hashed with the *integrity key (IK)* and the *initialization vector (IV)* using the SHA256 hash function, and the resulting value used as the hashed integrity code. The order in which the values used MUST be hashed is as follows:

  Hashed Integrity Code = SHA256 (IV + IK + Last-block-of-clear-text)

The encryption method and the corresponding parameters that MUST be used to encrypt the transfer stream are summarized in the table below.

Encryption method	AES-256
Parameters:	
Padding	As per PKCS-7 standard
Mode	CBC
Block size	16 bytes (as REQUIRED by AES)

The element content of `xenc:CipherValue` MUST be a base64 encoded byte array comprised of the initialization vector used for encryption, the hashed integrity code (as described above), and the encrypted transfer stream. It MUST be serialized as per the binary structure summarized in the table below.

Field	*Offset*	*Size (bytes)*
Initialization Vector (IV) used for encryption	0	16
Hashed integrity code	16	32
Ciphertext of transfer stream	48	Arbitrary

7 Simple Identity Provider Profile

A simple Identity Provider, called the "Self-issued Identity Provider" (SIP), is one which allows users to self-assert identity in the form of self-issued tokens. An Identity Selector MAY include a co-resident Self-issued Identity Provider that conforms to the Simple Identity Provider Profile defined in this section. This profile allows self-issued identities created within one Identity Selector to be used in another Identity Selector such that users do not have to reregister at a Relying Party when switching Identity Selectors. Because of the co-location there is data and metadata specific to an Identity Provider that need to be shareable between Identity Selectors.

7.1 Self-Issued Information Card

The `ic:Issuer` element within an Information Card provides a logical name for the issuer of the Information Card. An Information Card issued by a SIP (*i.e.*, a self-issued Information Card) MUST use the special URI below as the value of the `ic:Issuer` element in the Information Card.

URI:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self
```

7.2 Self-Issued Token Characteristics

The self-issued tokens issued by a SIP MUST have the following characteristics:

- The token type of the issued token MUST be SAML 1.1 which MUST be identified by either of the following token type URIs:
  - *urn:oasis:names:tc:SAML:1.0:assertion*, or
  - *http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1*.
- It is RECOMMENDED that the signature key used in the issued token be a 2048-bit asymmetric RSA key which identifies the issuer. Usage of other algorithms is not described.
- The issuer of the token, indicated by the value of the `saml:Issuer` attribute on the `saml:Assertion` root element, MUST be identified by the following URI defined in Section 2.1.1 representing the issuer "self".

  `http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self`

- The issued token MUST contain the `saml:Conditions` element specifying:
  - the token validity interval using the `NotBefore` and `NotOnOrAfter` attributes, and
  - the `saml:AudienceRestrictionCondition` element restricting the token to a specific target scope (i.e., a specific recipient of the token).
- The `saml:NameIdentifier` element SHOULD NOT be used to specify the Subject of the token.
- The subject confirmation method MUST be specified as one of:
  - *urn:oasis:names:tc:SAML:1.0:cm:holder-of-key*, or
  - *urn:oasis:names:tc:SAML:1.0:cm:bearer* (for Browser based applications).
- When the subject confirmation method is "holder of key", the subject confirmation key (also referred to as the *proof key*) MUST be included in the token in the `ds:KeyInfo` child element under the `saml:SubjectConfirmation` element. The proof key MUST be encoded in the token as follows:
  - For *symmetric* key tokens, the proof key is encrypted to the recipient of the token in the form of a `xenc:EncryptedKey` child element. It is RECOMMENDED that an AES key with a default size of 256 bits be used, but a different size MAY be specified by the Relying Party. Usage of other algorithms is not described.
  - For *asymmetric* key tokens, it is RECOMMENDED that the proof key be a public RSA key value specified as a `ds:RSAKeyValue` child element under the `ds:KeyValue` element. The default size of the key is 2048 bits. Usage of other algorithms is not described.
- The issued token MUST contain a single attribute statement (i.e., a single `saml:AttributeStatement` element) containing the subject confirmation data and the requested claims (called *attributes* in a SAML token).
- The claim types supported by the self-issued token SHOULD include those listed in Section7.5.
- The claims asserted in the `saml:AttributeStatement` element of the issued token MUST be named as follows using the claim type definitions in the XML schema file referenced in Section7.5. For each claim represented by a `saml:Attribute` element,
  - the `AttributeName` attribute is set to the NCname of the corresponding claim type defined in the XML schema file, and
  - the `AttributeNamespace` attribute is set to the target namespace of the XML schema file, namely

    `http://schemas.xmlsoap.org/ws/2005/05/identity/claims`

It is RECOMMENDED that the XML digital signature [XMLDSIG] profile used to sign a self-issued token be as follows. Usage of other algorithms is not described.

- Uses the *enveloped signature* format identified by the transform algorithm identifier "*http://www.w3.org/2000/09/xmldsig#enveloped-signature*". The token signature contains a single `ds:Reference` containing a URI reference to the `AssertionID` attribute value of the root element of the SAML token.
- Uses the RSA signature method identified by the algorithm identifier "*http://www.w3.org/2000/09/xmldsig#rsa-sha1*".
- Uses the exclusive canonicalization method identified by the algorithm identifier "*http://www.w3.org/2001/10/xml-exc-c14n#*" for canonicalizing the token content as well as the signature content.
- Uses the SHA1 digest method identified by the algorithm identifier "*http://www.w3.org/2000/09/xmldsig#sha1*" for digesting the token content being signed.
- No other transforms, other than the ones listed above, are used in the enveloped signature.
- The `ds:KeyInfo` element is always present in the signature carrying the signing RSA public key in the form of a `ds:RSAKeyValue` child element.

Following is an example of a self-issued signed Security Token containing three claims.

Example:

```
<Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
    AssertionID="urn:uuid:08301dba-d8d5-462f-85db-dec08c5e4e17"
    Issuer="http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self"
    IssueInstant="2004-10-06T16:44:20.00Z"
    MajorVersion="1" MinorVersion="1">
  <Conditions NotBefore="2004-10-06T16:44:20.00Z"
    NotOnOrAfter="2004-10-06T16:49:20.00Z">
    <AudienceRestrictionCondition>
      <Audience>http://www.relying-party.com</Audience>
    </AudienceRestrictionCondition>
  </Conditions>
  <AttributeStatement>
    <Subject>
      <!-- Content here differs; see examples that follow -->
    </Subject>
    <Attribute AttributeName="privatepersonalidentifier" AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <AttributeValue>
        f8301dba-d8d5a904-462f0027-85dbdec0
      </AttributeValue>
    </Attribute>
    <Attribute AttributeName="givenname" AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <AttributeValue>dasf</AttributeValue>
    </Attribute>
    <Attribute AttributeName="emailaddress" AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <AttributeValue>dasf@mail.com</AttributeValue>
    </Attribute>
  </AttributeStatement>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="urn:uuid:08301dba-d8d5-462f-85db-dec08c5e4e17">
        <Transforms>
          <Transform
            Algorithm="http://.../2000/09/xmldsig#enveloped-signature"/>
          <Transform
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        </Transforms>
        <DigestMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>vpnIyEi4R/S4b+1vEH4gwQ9iHsY=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>...</SignatureValue>
    <!-- token signing key -->
    <KeyInfo>
      <KeyValue>
        <RSAKeyValue>
          <Modulus>... utnQyEi8R/S4b+1vEH4gwR9ihsV ...</Modulus>
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
</Assertion>
```

The content of the `saml:Subject` element in the self-issued token differs based on the subject confirmation method and the type of proof key used. The following examples illustrate each of the three variations of the content of this element.

The following example illustrates the content of the `saml:Subject` element when subject confirmation method is "holder of key" using a symmetric proof key.

Example:

```
<Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:ds="..."
    xmlns:wsse="..." xmlns:xenc="...">
  <SubjectConfirmation>
    <ConfirmationMethod>
      urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
    </ConfirmationMethod>
    <ds:KeyInfo>
      <!-- symmetric proof key encrypted to recipient -->
      <xenc:EncryptedKey>
        <xenc:EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
        <ds:KeyInfo>
          <ds:X509Data>
            <wsse:KeyIdentifier
              ValueType="http://docs.oasis-open.org/wss/2004/xx/oasis-2004xx-wss-soap-message-security-1.1#ThumbprintSHA1">
                EdFoIaAeja85201XTzjNMVWy7532jUYtrx=
            </wsse:KeyIdentifier>
          </ds:X509Data>
        </ds:KeyInfo>
        <xenc:CipherData>
          <xenc:CipherValue>
            AuFhiu72+lkaJiAuFhiu72+lkaJi=
          </xenc:CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedKey>
    </ds:KeyInfo>
  </SubjectConfirmation>
</Subject>
```

The following example illustrates the content of the `saml:Subject` element when subject confirmation method is "holder of key" using an asymmetric proof key.

Example:

```
<Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:ds="...">
  <SubjectConfirmation>
    <ConfirmationMethod>
      urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
    </ConfirmationMethod>
    <ds:KeyInfo>
      <!-- asymmetric RSA public key as proof key -->
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>... FntQyKi6R/E4b+1vDH4gwS5ihsU ...</ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
  </SubjectConfirmation>
</Subject>
```

The following example illustrates the content of the `saml:Subject` element when subject confirmation method is "bearer" using no proof key.

Example:

```
<Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
  <SubjectConfirmation>
    <ConfirmationMethod>
      urn:oasis:names:tc:SAML:1.0:cm:bearer
    </ConfirmationMethod>
  </SubjectConfirmation>
</Subject>
```

7.3 Self-Issued Token Encryption

One of the goals of the Information Card Model is to ensure that any claims are exposed only to the Relying Party intended by the user. For this reason, the SIP SHOULD encrypt the self-issued token under the key of the Relying Party. This guarantees that a token intended for one Relying Party cannot be decoded by nor be meaningful to another Relying Party. As described in Section 8.3, when the Relying Party is not identified by a certificate, because no key is available for the Relying Party in this case, the token can not be encrypted, but SHOULD still be signed.

When a self-issued token is encrypted, the XML encryption [XMLENC] standard MUST be used. The encryption construct MUST use encrypting the self-issued token with a randomly generated symmetric key which in turn is encrypted to the Relying Party's public key taken from its X.509 v3 certificate. The encrypted symmetric key MUST be placed in an `xenc:EncryptedKey` element within the `xenc:EncryptedData` element carrying the encrypted Security Token.

It is RECOMMENDED that the XML encryption [XMLENC] profile that is used for encrypting the key and the token be as follows. Usage of other algorithms is not described.

- Uses the RSA-OAEP key wrap method identified by the algorithm identifier "*http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p*" for encrypting the encryption key.
- Uses the AES256 with CBC encryption method identified by the algorithm "*http://www.w3.org/2001/04/xmlenc#aes256-cbc*" for encrypting the token. The padding method used is as per the PKCS-7 standard in which the number of octets remaining in the last block is used as the padding octet value.
- The `ds:KeyInfo` element is present in the encrypted key specifying the encryption key information in the form of a Security Token reference.

Following is an illustration of a self-issued token encrypted to a Relying Party using the encryption structure described above.

Example:

```
<xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
    xmlns:xenc="..." xmlns:ds="..." xmlns:wsse="...">
  <xenc:EncryptionMethod
     Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc" />
  <ds:KeyInfo>
    <xenc:EncryptedKey>
      <xenc:EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      </xenc:EncryptionMethod>
      <ds:KeyInfo>
        <wsse:SecurityTokenReference>
          <wsse:KeyIdentifier
            ValueType="http://docs.oasis-open.org/wss/2004/xx/oasis-2004xx-wss-soap-message-security-1.1#ThumbprintSHA1"
            EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis200401-wss-soap-message-security-1.0#Base64Binary">
            +PYbznDaB/dlhjIfqCQ458B72wA=
          </wsse:KeyIdentifier>
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>...Ukasdj8257fjWf=</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedKey>
  </ds:KeyInfo>
  <xenc:CipherData>
    <!-- Start encrypted Content
    <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
       AssertionID="Urn:uuid:0830ldbd-2bd6-467a-85db-dac08c5b4a17" ...>
       ...
    </Assertion>
    End encrypted content -->
    <xenc:CipherValue>...aKlh4817JerpZoDofy90=</xenc:CipherValue>
  </xenc:CipherData>
</xenc:EncryptedData>
```

7.4 Self-Issued Token Signing Key

The key used to sign a self-issued token presented to a Relying Party also represents a unique identifier for the Subject of the token. In order to prevent the key from becoming a correlation identifier across relying parties, a SIP SHOULD use a different key to sign a self-issued token for each Relying Party where the card is used. In other words, the key used to sign the self-issued token is pair-wise unique for a given Information Card and RP combination. To allow self-issued identities created by a SIP within one Identity Selector to be used in another, the signing keys used by the two SIPs SHOULD be the same.

It is RECOMMENDED that the signing key be an RSA key. Usage of other algorithms is not described.

This section specifies the "processing rules" that SHOULD be used by a SIP to derive the RSA key used to sign the self-issued token for a combination of an Information Card and an RP where the card is used. Each self-issued Information Card contains a 256-bit secret random number, called the "master key" (see Section 6.1), that is used as the secret entropy in deriving the token signing RSA key. (Managed Information Cards also have a master key that is used in the Client Pseudonym PPID calculation, as per Section 3.3.4.1.)

Key derivation is done according to the ANSI X9.31 standard for key generation which starts with requiring the use of six random values denoted by X_{p1}, X_{p2}, X_{q1}, X_{q2}, X_p, and X_q. The processing rules described here enunciate how to transform the master key in an Information Card into the six random inputs for the X9.31 key generation process. The actual key computation algorithm in the X9.31 standard is *not* reproduced here.

The values X_p and X_q are REQUIRED to be at least 512 bits and each independently carries the full entropy of any Information Card master key of up to 512 bits in length. The values X_{p1}, X_{p2}, X_{q1}, and X_{q2} have a length of only 100 to 121 bits and therefore will be shorter than the Information Card master key and hence cannot each independently carry the full master key entropy. The details of the X9.31 protocol, however, ensure that for reasonably sized master keys, full entropy will be achieved in the generated asymmetric key pair.

7.4.1 Processing Rules

This key generation mechanism can be used to generate 1024 or 2048-bit RSA keys.

Notation: If H is an *n*-bit big-endian value, the convention H[1..p] denotes bits *1* through *p* in the value of H where $p \leq n$, and bit-1 is the rightmost (least significant) bit whereas bit-n is the leftmost (most significant) bit in the value of H. Also, the convention X + Y denotes the concatenation of the big-endian bit value of X followed by the big-endian bit value of Y.

Assume that the master key for the selected Information Card (see Section 6.1) is M and the unique *RP Identifier* (derived as per Section 7.6.1) is T. The following processing rules SHOULD be used to derive the inputs for the X9.31 key generation process.

1. Define 32-bit DWORD constants C_n as follows:

   $C_n = n$, where $n = 0,1,2,...,15$

2. Compute SHA-1 hash values H_n as follows:

   If the requested key size = 1024 bits, compute

   $H_n = \text{SHA1}(M + T + C_n)$ for $n = 0,1,2,...,9$

   If the requested key size = 2048 bits, compute

   $H_n = \text{SHA1}(M + T + C_n)$ for $n = 0,1,2,...,15$

3. Extract the random input parameters for the X9.31 protocol as follows:

   For all key sizes, compute

   X_{p1} [112-bits long] = H_0[1..112]

   X_{p2} [112-bits long] = H_1[1..112]

   X_{q1} [112-bits long] = H_2[1..112]

   X_{q2} [112-bits long] = H_3[1..112]

   If the requested key size = 1024 bits, compute

   X_p [512-bits long] = H_4[1..160] + H_5[1..160] + H_6[1..160] + H_0[129..160]

   X_q [512-bits long] = H_7[1..160] + H_8[1..160] + H_9[1..160] + H_1[129..160]

   If the requested key size = 2048 bits, compute

   X_p [1024-bits long] = H_4[1..160] + H_5[1..160] + H_6[1..160] + H_0[129..160] +
   H_{10}[1..160] + H_{11}[1..160] + H_{12}[1..160] + H_2[129..160]

X_q [1024-bits long] = $H_7[1..160] + H_8[1..160] + H_9[1..160] + H_1[129..160] +$
$H_{13}[1..160] + H_{14}[1..160] + H_{15}[1..160] + H_3[129..160]$

4. The X9.31 specification (Section 4.1.2) requires that the input values X_{p1}, X_{p2}, X_{q1}, X_{q2} MUST satisfy the following conditions.
   - The large prime factors p_1, p_2, q_1, and q_2 are the first primes greater than their respective random X_{p1}, X_{p2}, X_{q1}, X_{q2} input values. They are randomly selected from the set of prime numbers between 2^{100} and 2^{120}, and each SHALL pass at least 27 iterations of Miller-Rabin.

   To ensure that the lower bound of 2^{100} is met, set the 101th bit of X_{p1}, X_{p2}, X_{q1}, X_{q2} to '1' (*i.e.* X_{p1}[13th byte] |= 0x10, X_{p2}[13th byte] |= 0x10, X_{q1}[13th byte] |= 0x10, X_{q2}[13th byte] |= 0x10).
5. The X9.31 specification (Section 4.1.2) requires that the input values X_p and X_q MUST satisfy the following conditions.
   - If the requested key size = 1024 bits, then
     $X_p \geq (\sqrt{2})(2^{511})$ and $X_q \geq (\sqrt{2})(2^{511})$
   - If the requested key size = 2048 bits, then
     $X_p \geq (\sqrt{2})(2^{1023})$ and $X_q \geq (\sqrt{2})(2^{1023})$

   To ensure this condition is met, set the two most significant bits of X_p and X_q to '1' (*i.e.* X_p[most significant byte] |= 0xC0, X_q[most significant byte] |= 0xC0).
6. Compute 1024 or 2048-bit keys as per the X9.31 protocol using {X_{p1}, X_{p2}, X_{q1}, X_{q2}, X_p, X_q} as the random input parameters.
7. Use a 32-bit DWORD size *public exponent* value of 65537 for the generated RSA keys.

There are three conditions as follows in the X9.31 specification which, if not met, require that one or more of the input parameters MUST be regenerated.

- (Section 4.1.2 of X9.31) $|X_p - X_q| \geq 2^{412}$ (for 1024-bit keys) or $|X_p - X_q| \geq 2^{924}$ (for 2048-bit keys). If not true, X_q MUST be regenerated and q recomputed.
- (Section 4.1.2 of X9.31) $|p-q| \geq 2^{412}$ (for 1024-bit keys) or $|p-q| \geq 2^{924}$ (for 2048-bit keys). If not true, X_q MUST be regenerated and q recomputed.
- (Section 4.1.3 of X9.31) $d > 2^{512}$ (for 1024-bit keys) or $d > 2^{1024}$ (for 2048-bit keys). If not true, X_{q1}, X_{q2}, and X_q MUST be regenerated and key generation process repeated.

When it is necessary to regenerate an input parameter as necessitated by one or more of the conditions above, it is essential that the regeneration of the input parameter be deterministic to guarantee that all implementations of the key generation mechanism will produce the same results. Furthermore, input regeneration is a potentially unlimited process. In other words, it is possible that regeneration MUST be performed more than once. In theory, one MAY need to regenerate input parameters many times before a key that meets all of the requirements can be generated.

The following processing rules MUST be used for regenerating an input parameter *X* of length *n-bits* when necessary:

a. Pad the input parameter *X* on the right, assuming a big-endian representation, with *m* zero-bits where *m* is the smallest number which satisfies *((n+m) mod 128 = 0)*.

b. Encrypt the padded value with the AES-128 (**E**lectronic **C**ode **B**ook mode) algorithm using the 16-byte constant below as the encryption key:

Encryption Key	{ 0x8b, 0xe5, 0x61, 0xf5, 0xbc, 0x3e, 0x0c, 0x4e, 0x94, 0x0d, 0x0a, 0x6d, 0xdc, 0x21, 0x9d, 0xfd }

c. Use the leftmost *n-bits* of the result above as the REQUIRED regenerated parameter.

If a regenerated parameter does not satisfy the necessary conditions, then repeat the 3-step process above (call it *RegenFunction*) to generate the parameter again by using the output of one iteration as input for the next iteration. In other words, if the output of the i^{th} iteration of the regeneration function above for an input parameter *X* is given by X_i then

$$X_{i+1} = RegenFunction\ (X_i)$$

7.5 Claim Types

This section specifies a set of claim (attribute) types and the corresponding URIs that is defined by this profile for some commonly used personal information. These claim types MAY be used by a SIP, in self-issued tokens, or by other Identity Providers. Note that, wherever possible, the claims included here reuse and refer to the attribute semantics defined in other established industry standards that deal with personal information. A SIP SHOULD support these claim types at a minimum. Other Identity Providers MAY also support these claim types when appropriate. The URIs defined here MAY be used by a Relying Party to specify requested claims in its policy.

The base XML namespace URI that is used by the claim types defined here is as follows:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/claims
```

For convenience, an XML Schema for the claim types defined here can be found at:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/claims.xsd
```

7.5.1 First Name

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname*

Type: *xs:string*

Definition: (*givenName* in [RFC 2256]) Preferred name or first name of a Subject. According to RFC 2256: "This attribute is used to hold the part of a person's name which is not their surname nor middle name."

7.5.2 Last Name

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname*

Type: *xs:string*

Definition: (*sn* in [RFC 2256]) Surname or family name of a Subject. According to RFC 2256: "This is the X.500 surname attribute which contains the family name of a person."

7.5.3 Email Address

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress*

Type: *xs:string*

Definition: (*mail* in inetOrgPerson) Preferred address for the "To:" field of email to be sent to the Subject, usually of the form <user>@<domain>. According to inetOrgPerson using [RFC 1274]: "This attribute type specifies an electronic mailbox attribute following the syntax specified in RFC 822."

7.5.4 Street Address

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/streetaddress*

Type: *xs:string*

Definition: (*street* in [RFC 2256]) Street address component of a Subject's address information. According to RFC 2256: "This attribute contains the physical address of the object to which the entry corresponds, such as an address for package delivery." Its content is arbitrary, but typically given as a PO Box number or apartment/house number followed by a street name, ***e.g.*** 303 Mulberry St.

7.5.5 Locality Name or City

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/locality*

Type: *xs:string*

Definition: (*l* in [RFC 2256]) Locality component of a Subject's address information. According to RFC 2256: "This attribute contains the name of a locality, such as a city, county or other geographic region," ***e.g.*** Redmond.

7.5.6 State or Province

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/stateorprovince*

Type: *xs:string*

Definition: (*st* in [RFC 2256]) Abbreviation for state or province name of a Subject's address information. According to RFC 2256: "This attribute contains the full name of a state or province. The values SHOULD be coordinated on a national level and if well-known shortcuts exist - like the two-letter state abbreviations in the US – these abbreviations are preferred over longer full names." ***e.g.*** WA.

7.5.7 Postal Code

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/postalcode*

Type: *xs:string*

Definition: (*postalCode* in X.500) Postal code or zip code component of a Subject's address information. According to X.500(2001): "The postal code attribute type specifies the postal code of the named object. If

this attribute value is present, it will be part of the object's postal address - zip code in USA, postal code for other countries.

7.5.8 Country

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/country*

Type: *xs:string*

Definition: (*c* in [RFC 2256]) Country of a Subject. According to RFC 2256: "This attribute contains a two-letter ISO 3166 country code."

7.5.9 Primary or Home Telephone Number

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/homephone*

Type: *xs:string*

Definition: (*homePhone* in inetOrgPerson) Primary or home telephone number of a Subject. According to inetOrgPerson using [RFC 1274]: "This attribute type specifies a home telephone number associated with a person." Attribute values SHOULD follow the agreed format for international telephone numbers, *e.g.* +44 71 123 4567.

7.5.10 Secondary or Work Telephone Number

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/otherphone*

Type: *xs:string*

Definition: (*telephoneNumber* in X.500 Person) Secondary or work telephone number of a Subject. According to X.500(2001): "This attribute type specifies an office/campus telephone number associated with a person." Attribute values SHOULD follow the agreed format for international telephone numbers, *e.g.* +44 71 123 4567.

7.5.11 Mobile Telephone Number

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/mobilephone*

Type: *xs:string*

Definition: (*mobile* in inetOrgPerson) Mobile telephone number of a Subject. According to inetOrgPerson using [RFC 1274]: "This attribute type specifies a mobile telephone number associated with a person." Attribute values SHOULD follow the agreed format for international telephone numbers, *e.g.* +44 71 123 4567.

7.5.12 Date of Birth

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/dateofbirth*

Type: *xs:date*

Definition: The date of birth of a Subject in a form allowed by the *xs:date* data type.

7.5.13 Gender

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/gender*

Type: *xs:token*

Definition: Gender of a Subject that can have any of these exact string values – '0' (meaning unspecified), '1' (meaning Male) or '2' (meaning Female). Using these values allows them to be language neutral.

7.5.14 Private Personal Identifier

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/privatepersonalidentifier*

Type: *xs:base64binary*

Definition: A private personal identifier (PPID) that identifies the Subject to a Relying Party. The word "private" is used in the sense that the Subject identifier is specific to a given Relying Party and hence private to that Relying Party. A Subject's PPID at one Relying Party cannot be correlated with the Subject's PPID at another Relying Party. Typically, the PPID SHOULD be generated by an Identity Provider as a pair-wise pseudonym for a Subject for a given Relying Party. For a self-issued Information Card, the Self-issued Identity Provider in an Identity Selector system SHOULD generate a PPID for each Relying Party as a function of the card identifier and the Relying Party's identity. The processing rules and encoding of the PPID claim value is specified in Section 7.6.

Compatibility Note: Some existing Identity Selectors omit listing the PPID claim as an `ic:SupportedClaimType` from the `ic:SupportedClaimTypeList` when saving a self-issued Information Card in the Information Cards Transfer Format defined in Section 6.1, even though the PPID claim is supported by the card. This behavior is deprecated, as all supported claims SHOULD be listed. Nonetheless, Identity Selectors MAY choose to recognize this case and support the PPID claim for self-issued cards not explicitly listing this claim.

7.5.15 Web Page

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/webpage*

Type: *xs:string*

Definition: The Web page of a Subject expressed as a URL.

7.6 The PPID Claim

The PPID claim for a Subject user represents a unique identifier for that user at a given Relying Party that is different from all identifiers for that user at any other Relying Party. In other words, the PPID is a pair-wise unique identifier for a given user identity and Relying Party combination. Since an Information Card represents a specific user identity and a Relying Party is the organization behind a Web service or site that the user interacts with, the PPID claim is logically a function of an Information Card and the organizational identity of the Relying Party.

This section describes the processing rules that SHOULD be used by a SIP to derive a PPID claim value for a combination of an Information Card and a Relying Party where it is used.

7.6.1 Relying Party Identifier and Relying Party PPID Seed

In order to derive the PPID and Signing Key as functions of the RP's organizational identity, a stable and unique identifier for the RP, called the *RP Identifier*, is needed. In the Information Card Model, the identity of a Relying Party (RP) possessing an X.509v3 certificate is presented in the form of that certificate. Therefore the organizational identity of the RP is obtained by applying a series of transformations to the identity information carried in the X.509 certificate. (See Section 8 for the specification of how to compute these values for Relying Parties not possessing a certificate.)

As specified in [RFC 2459], the subject field inside an X.509 certificate identifies the entity associated with the public key stored in the subject public key field. Where it is non-empty, the subject field MUST contain an X.500 distinguished name (DN). The DN MUST be unique for each subject entity certified by the one CA as defined by the issuer name field.

The subject field contains a DN of the form shown below:

CN=*string*, [OU=*string*, ...,] O=*string*, L=*string*, S=*string*, C=*string*

The Object Identifiers for these attributes from the DN are as follows:

Field Abbreviation	*Field Name*	*Object Identifier*
O	organizationName	2.5.4.10
L	localityName	2.5.4.7
S	stateOrProvinceName	2.5.4.8
C	countryName	2.5.4.6
CN	commonName	2.5.4.3

Note that the field names and abbreviations used in this specification may not correspond to those used by particular software but the underlying Object Identifiers (OIDs) of the attributes are unambiguous.

For an end-entity certificate, the values of the attribute types O (organizationName), L (localityName), S (stateOrProvinceName) and C (countryName) together uniquely identify the organization to which the end-entity identified by the certificate belongs. These attribute types are collectively referred to as the *organizational identifier attributes* here. The *RP Identifier* is constructed using these organizational identifier attributes as described below.

The *RP Identifier* value is used as an input to the Signing Key computation. A closely related value called the Relying Party PPID Seed is also computed, which is used as an input to the PPID claim and Client Pseudonym PPID computations. In many cases these are the same but in one case they differ.

There are four cases of how the *RP Identifier* and *RP PPID Seed* are constructed depending on which organizational identifier attributes the RP's certificate contains, if it is an extended validation (EV) certificate [EV Cert] with respect to the organizational identifier attributes, and if it chains to a trusted root certificate.

Case 1: RP's certificate *is* EV for organizational identifier attributes and chains to a trusted root certificate authority

- Convert the organizational identifier attributes in the end-entity certificate into a string, call it *OrgIdString*, of the following form:

  |O="*string*"|L="*string*"|S="*string*"|C="*string*"|

  The vertical bar character (ASCII 0x7C) is used as a delimiter at the start and end of the string as well as between the attribute types. Further, the string values of the individual attribute types are enclosed within double quote characters (ASCII 0x22). If an attribute type is absent in the subject field of the end-entity certificate, then the corresponding string value is the empty string (""). Following is an example *OrgIdString* per this convention:

OrgIdString per this convention:

|O="Microsoft"|L="Redmond"|S="Washington"|C="US"|

- Encode all the characters in *OrgIdString* into a sequence of bytes, call it *OrgIdBytes*, using Unicode encoding UTF-16LE with no byte order mark.
- Hash *OrgIdBytes* using the SHA256 hash function, and use the resulting value as the *RP Identifier* and *RP PPID Seed*.

  RP PPID Seed = RP Identifier = SHA256 (OrgIdBytes)

Case 2: RP's certificate *is not* EV for organizational identifier attributes, has a non-empty organizationName (O) value, and chains to a trusted root certificate authority

- Convert the organizational identifier attributes in the end-entity certificate into a string, call it *OrgIdString*, in the same manner as employed for Case 1 above.
- Let *QualifierString* be the string:

  |Non-EV
- Let *QualifiedOrgIdString* be the concatenation of *QualifierString* and *OrgIdString*.

  QualifiedOrgIdString = QualifierString + OrgIdString
- Encode all the characters in *QualifiedOrgIdString* into a sequence of bytes, call it *QualifiedOrgIdBytes*, using Unicode encoding UTF-16LE with no byte order mark.
- Hash *QualifiedOrgIdBytes* using the SHA256 hash function, and use the resulting value as the *RP Identifier*.

  RP Identifier = SHA256 (QualifiedOrgIdBytes)
- Encode all the characters in *OrgIdString* into a sequence of bytes, call it *OrgIdBytes*, using Unicode encoding UTF-16LE with no byte order mark.
- Hash *OrgIdBytes* using the SHA256 hash function, and use the resulting value as the *Relying Party PPID Seed*.

  RP PPID Seed = SHA256 (OrgIdBytes)

Case 3: RP's certificate has an empty or no organizationName (O) value and has an empty or no commonName (CN) or does not chain to a trusted root certificate authority

- Take the subject public key in the end-entity certificate, call it *PublicKey*, as a byte array.
- Hash *PublicKey* using the SHA256 hash function, and use the resulting value as the *RP Identifier* and *RP PPID Seed*.

  RP PPID Seed = RP Identifier = SHA256 (PublicKey)

Case 4: RP's certificate has an empty or no organizationName (O) value but has a non-empty commonName (CN) value and chains to a trusted root certificate authority

- Convert the commonName attribute value in the end-entity certificate into a string, call it *CnIdString*, of the following form:

  |CN="*string*"|

  Following is an example *CnIdString* per this convention:

  |CN="login.live.com"|
- Encode all the characters in *CnIdString* into a sequence of bytes, call it *CnIdBytes*, using Unicode encoding UTF-16LE with no byte order mark.
- Hash *CnIdBytes* using the SHA256 hash function, and use the resulting value as the *RP Identifier* and *RP PPID Seed*.

  RP PPID Seed = RP Identifier = SHA256 (CnIdBytes)

7.6.2 PPID

The PPID value SHOULD be produced as follows using the card identifier and the *RP PPID Seed* (specified in Section 7.6.1):

- Encode the value of the `ic:CardId` element of the Information Card into a sequence of bytes, call it *CardIdBytes*, using Unicode encoding UTF-16LE with no byte order mark.
- Hash *CardIdBytes* using the SHA256 hash function to obtain the canonical card identifier *CanonicalCardId*.

  CanonicalCardId = SHA256 (CardIdBytes)
- Hash the concatenation of *RP PPID Seed* and *CanonicalCardId* using the SHA256 hash function to obtain the PPID.

  PPID = SHA256 (RP PPID Seed + CanonicalCardId)

7.6.3 Friendly Identifier

The PPID provides an RP-specific identifier for a Subject that is suitable for programmatic processing, but is not a user-friendly identifier. The simple transformation rules specified in this section MAY be used by a SIP, or any other Identity Provider supporting the PPID claim, to create a friendly identifier for use within a Display Token accompanying a Security Token carrying the PPID claim.

The Friendly Identifier has the following characteristics:

- It is encoded as a 10-character alphanumeric string of the form "AAA-AAAA-AAA" grouped into three groups separated by the 'hyphen' character (*e.g.*, the string "6QR-97A4-WR5"). Note that the hyphens are used for punctuation only.
- The encoding alphabet does NOT use the numbers '0' and '1', and the letters 'O' and 'I' to avoid confusion stemming from the similar glyphs used for these numbers and characters. This leaves 8 digits and 24 letters – a total of 32 alphanumeric symbols – as the alphabet for the encoding.

The processing rules used for deriving a Friendly Identifier from a PPID are as follows:

- The PPID value is conveyed as a base64 encoded string inside tokens. Start with the base64 decoded PPID value as input.
- Hash the PPID value using the SHA1 hash function to obtain a hashed identifier.

  HashId = SHA1 (PPID)
- Let the Friendly Identifier be the string "$A_0 A_1 A_2$–$A_3 A_4 A_5 A_6$–$A_7 A_8 A_9$" where each A_i is an alphanumeric character from the encoding alphabet described above.
- For *i := 0 to 9*, each A_i is determined as below:
  - Take the i^{th} octet of *HashId* (denoted as *HashId[i]*)
  - Find *RawValue = HashId[i] % 32* (where % is the remainder operation)
  - A_i = *EncodedSymbol* obtained by mapping *RawValue* to *EncodedSymbol* using the table below

Raw Value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Encoded Symbol	Q	L	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Raw Value	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Encoded Symbol	G	H	J	K	M	N	P	R	S	T	U	V	W	X	Y	Z

8 Relying Parties without Certificates

While Relying Parties are typically identified by presenting a cryptographically protected identity, such as an X.509v3 certificate, the Information Card Model is also applicable in situations in which no Relying Party certificate is available. This section specifies how Information Cards are used at Relying Parties with no certificate: specifically, Web sites using the [HTTP] scheme. Also see `ic07:RequireStrongRecipientIdentity` in Section 3.1.1.7 for a means whereby card issuers can prohibit the use of cards at Relying Parties not identified by a certificate.

8.1 Relying Party Identifier and Relying Party PPID Seed

The Relying Party Identifier and Relying Party PPID Seed values for Relying Parties without certificates are computed in this manner:

- Set the string *OrgIdString* to be the fully qualified DNS host name in lowercase characters specified in the URI of the Relying Party, or if a numeric IP address was used, then a string representation of the IP address of the server. For IPv4 addresses, this string is the standard 4-byte dotted decimal representation of the address with no leading zeros, such as `131.107.55.210`. For IPv6 addresses, this string

is the hexadecimal representation of the address in eight groups of four hex digits each using uppercase for the letters, with each group of four digits separated by a colon, all enclosed by square brackets, such as `[0000:1234:0000:0000:0000:000A:00BC:0DEF]`.

- Encode all the characters in *OrgIdString* into a sequence of bytes, call it *OrgIdBytes*, using the Unicode encoding UTF-16LE with no byte order mark.
- Hash *OrgIdBytes* using the SHA256 hash function, and use the resulting value as both the *RP Identifier* and the *RP PPID Seed*.

The *RP Identifier* and *RP PPID Seed* are then used in the same manner as for Relying Parties identified by certificates when computing PPID claim and Client Pseudonym PPID values.

8.2 AppliesTo Information

Under the circumstances described in Section 3.3.3 that the RP endpoint to which the token will be sent is supplied as the `wsp:AppliesTo` value to the IP, when the RP possesses no certificate, the URL of the RP is supplied as that `wsp:AppliesTo` value.

Example:

```
<wst:RequestSecurityToken xmlns:wst="..." xmlns:wsp="..." xmlns:wsa="...">
  <wsp:AppliesTo>
    <wsa:EndpointReference>
      <wsa:Address>http://login.contoso.com</wsa:Address>
    </wsa:EndpointReference>
  </wsp:AppliesTo>
  ...
</wst:RequestSecurityToken>
```

8.3 Token Signing and Encryption

When the Relying Party is not identified by a certificate, tokens sent from the Self-issued Identity Provider are not encrypted, although they are still signed in the manner described in Section 7.2. Tokens generated by Identity Providers for Relying Parties not identified by a certificate are also typically not encrypted, as no encryption key is available. However, the token MAY still be encrypted if the Identity Provider has a pre-existing relationship with the Relying Party and they have mutually agreed on the use of a known encryption key. The token SHOULD still typically be signed, even when not encrypted.

9 Using WS-SecurityPolicy 1.2 and WS-Trust 1.3

Software implementing the Information Card Model SHOULD utilize the OASIS standard versions of WS-SecurityPolicy and WS-Trust – [WS-SecurityPolicy 1.2] and [WS-Trust 1.3] and MAY utilize the previous draft versions – [WS-SecurityPolicy 1.1] and [WS-Trust 1.2]. This section describes the differences between the old and standard versions of these protocols that MAY affect software implementing the Information Card Model.

9.1 Overview of Differences

The following changes between the protocol versions affect software implementing this specification:

- **Namespace changes:**
  http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702 replaces http://schemas.xmlsoap.org/ws/2005/07/securitypolicy.
  http://docs.oasis-open.org/ws-sx/ws-trust/200512 replaces http://schemas.xmlsoap.org/ws/2005/02/trust.
- **Use of RequestSecurityTokenResponseCollection:** A `wst:RequestSecurityTokenResponseCollection` element encloses the `wst:RequestSecurityTokenResponse` when WS-Trust 1.3 is used.
- **Use of SecondaryParameters:** An Identity Selector sends some information received from the Relying Party to the Identity Provider in a `wst:SecondaryParameters` element.
- **Bearer Token Request Syntax:** The new `wst:KeyType` value http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer is used to request a bearer token.

9.2 Identity Selector Differences

Identity Selectors MUST determine the WS-Trust versions used by Identity Provider STSs and Relying Party STSs using their Security Policy.

Identity Selectors supporting WS-Trust 1.3 MUST understand the new WS-Trust 1.3 elements and syntax such as `wst13:RequestSecurityTokenResponseCollection` and new URIs such as http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer. They MUST also understand that typical properties of an RST like Claims and KeyType MAY be either a direct child of the top level `wst13:RequestSecurityToken` element or contained within a `wst13:SecondaryParameters` element in the RST.

When constructing an RST for an Identity Provider using WS-Trust 1.3, the Identity Selector SHOULD send parameters received from the Relying Party in a `wst13:SecondaryParameters` element within the `wst13:RequestSecurityToken`, with these exceptions:

- The user chooses not to send optional claims. In this scenario, no SecondaryParameters element is sent in order to hide this user decision.
- No `wsp:AppliesTo` is being sent in the RST. In this scenario, no `wst13:SecondaryParameters` element is sent so that the Identity Provider does not obtain any identifying information about the Relying Party.

Example:

```
<wst13:RequestSecurityToken Context="ProcessRequestSecurityToken"
    xmlns:wst13="..." xmlns:ic="...">
  <wst13:RequestType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue</wst13:RequestType>
  <ic:InformationCardReference>
  ...
  </ic:InformationCardReference>
  <wst13:Claims Dialect="http://schemas.xmlsoap.org/ws/2005/05/identity">
  ...
  </wst13:Claims>
  <wst13:KeyType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/SymmetricKey</wst13:KeyType>
  <wst13:SecondaryParameters>
    <wst13:RequestType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue</wst13:RequestType>
    <wst13:TokenType>urn:oasis:names:tc:SAML:1.0:assertion</wst13:TokenType>
    <wst13:KeyType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/SymmetricKey</wst13:KeyType>
    <wst13:KeyWrapAlgorithm>http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p</wst13:KeyWrapAlgorithm>
    ...
  </wst13:SecondaryParameters>
</wst13:RequestSecurityToken>
```

The `wst13:RequestSecurityTokenResponse` constructed MUST be enclosed within a `wst13:RequestSecurityTokenResponseCollection` element.

Example:

```
<wst13:RequestSecurityTokenResponseCollection xmlns:wst13="...">
  <wst13:RequestSecurityTokenResponse>
    <wst13:TokenType>urn:oasis:names:tc:SAML:1.0:assertion</wst13:TokenType>
    <wst13:RequestedSecurityToken> ... </wst13:RequestedSecurityToken>
    ...
  </wst13:RequestSecurityTokenResponse>
</wst13:RequestSecurityTokenResponseCollection>
```

9.3 Security Token Service Differences

To utilize WS-Trust 1.3, an Identity Provider STS and Relying Party STSs MUST express their Security Policy using WS-SecurityPolicy 1.2.

STSs using WS-Trust 1.3 MUST understand the new WS-Trust 1.3 elements and syntax such as `wst13:RequestSecurityTokenResponseCollection` and new URIs such as http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer. They MUST also understand that typical properties of an RST like Claims and KeyType MAY be either a direct child of the top level `wst13:RequestSecurityToken` element or contained within a `wst13:SecondaryParameters` element in the RST.

10 Browser Behavior with Information Cards

This section explains the steps that a Web browser takes when using an Information Card to authenticate to a Web site. Two cases are described. The basic case is where the Web site provides all the Relying Party functionality via HTML extensions transported over HTTPS. The second case is where the Relying Party employs a Relying Party Security Token Service (STS), which it references via HTML extensions transported over HTTPS.

10.1 Basic Protocol Flow when using an Information Card at a Web Site

This section explains the protocol flow when using an Information Card to authenticate at a Web site where no Relying Party STS is employed.

Browser w/ Identity Selector

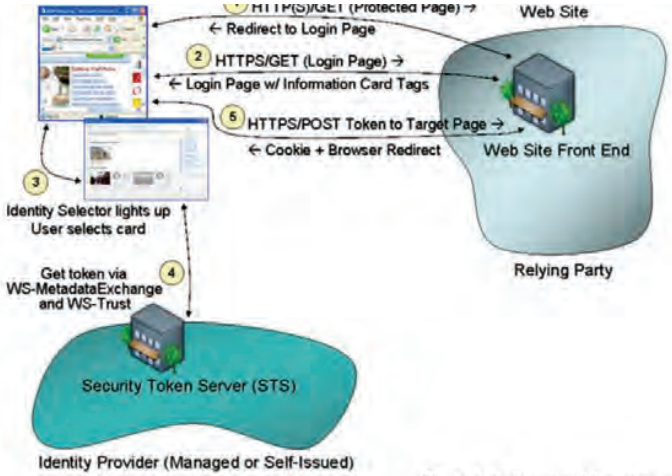


Figure 1. Basic protocol flow when using an Information Card to authenticate at a Web site

Figure 1 gives an example of the basic protocol flow when an Information Card is used to authenticate at a Web site that employs no Relying Party STS. Steps 1, 2, and 5 are essentially the same as a typical forms-based login today: (1) The user navigates to a protected page that requires authentication. (2) The site redirects the browser to a login page, which presents a Web form. (5) The browser posts the Web form that includes the login credentials supplied by the user back to the login page. The site then validates the contents of the form including the user credentials, typically writes a client-side browser cookie to the client for the protected page domain, and redirects the browser back to the protected page.

The key difference between this scenario and today's site login scenarios is that the login page returned to the browser in step (2) contains an HTML tag that allows the user to choose to use an Information Card to authenticate to the site. When the user selects this tag, the browser invokes an Identity Selector, which implements the Information Card user experience and protocols, and triggers steps (3) through (5).

In Step (3), the browser Information Card support code invokes the Identity Selector, passing it parameter values supplied by the Information Card HTML tag supplied by the site in Step (2). The user then uses the Identity Selector to choose an Information Card, which represents a Digital Identity that can be used to authenticate at that site. Step (4) retrieves a Security Token that represents the Digital Identity selected by the user from the STS at the Identity Provider for that identity.

In Step (5), the browser posts the token obtained back to the Web site using a HTTPS/POST. The Web site validates the token, completing the user's Information Card-based authentication to the Web site. Following authentication, the Web site typically then writes a client-side browser cookie and redirects the browser back to the protected page.

It is worth noting that this cookie is likely to be *exactly the same cookie* as the site would have written back had the user authenticated via other means, such as a forms-based login using username/password. This is one of the ways that the goal of "minimal impact on Web sites" is achieved. Other than its authentication subsystem, the bulk of a Web site's code can remain completely unaware that Information Card-based authentication is even utilized. It just uses the same kinds of cookies as always.

10.2 Protocol Flow with Relying Party STS

In the previous scenario, the Web site communicated with the client Identity Selector using only the HTML extensions enabling Information Card use, transported over the normal browser HTTPS channel. In this scenario, the Web site also employs a Relying Party STS to do part of the work of authenticating the user, passing the result of that authentication on to the login page via HTTPS POST.

There are several reasons that a site might factor its solution this way. One is that the same Relying Party STS can be used to do the authentication work for both browser-based applications and smart client applications that are using Web services. Second, it allows the bulk of the authentication work to be done on servers dedicated to this purpose, rather than on the Web site front-end servers. Finally, this means that the front-end servers can accept site-specific tokens, rather than the potentially more general or more complicated authentication tokens issued by the Identity Providers.

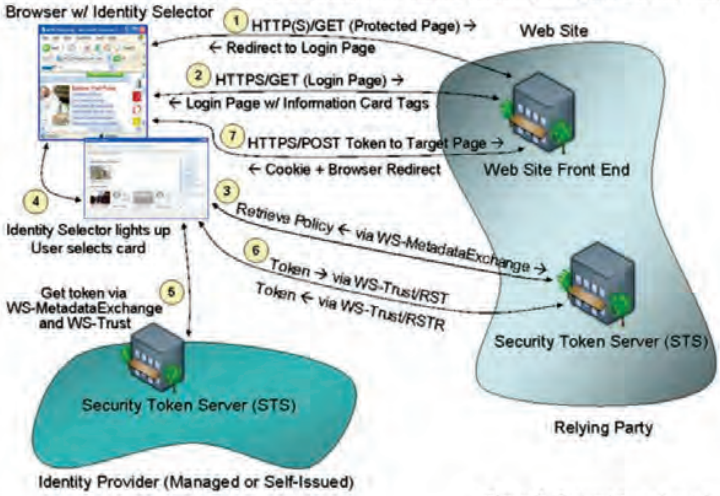


Figure 2. Protocol flow when using an Information Card to authenticate at a Web site, where the Web site employs a Relying Party STS

This scenario is similar to the previous one, with the addition of steps (3) and (6). The differences start with the Information Card information supplied to the browser by the Web site in Step (2). In the previous scenario, the site encoded its WS-SecurityPolicy information using Information Card HTML extensions and supplied them to the Information Card-extended browser directly. In this scenario, the site uses different Information Card HTML extensions in the Step (2) reply to specify which Relying Party STS SHOULD be contacted to obtain the WS-SecurityPolicy information.

In Step (3), the Identity Selector contacts the Relying Party STS specified by the Web site and obtains its WS-SecurityPolicy information via WS-MetadataExchange. In Step (4) the Identity Selector user interface is shown and the user selects an Information Card, which represents a Digital Identity to use at the site. In Step (5), the Identity Provider is contacted to obtain a Security Token for the selected Digital Identity. In Step (6), the Security Token is sent to the Web site's Relying Party STS to authenticate the user and a site-specific authentication token is returned to the Identity Selector. Finally, in Step (7), the browser posts the token obtained in Step (6) back to the Web site using HTTPS/POST. The Web site validates the token, completing the user's Information Card-based authentication to the Web site. Following authentication, the Web site typically then writes a client-side browser cookie and redirects the browser back to the protected page.

10.3 User Perspective and Examples

The Information Card user experience at Web sites is intended to be intuitive and natural enough that users' perspective on it will simply be "That's how you log in". Today, Web sites that require authentication typically ask the user to supply a username and password at login time. With Information Cards, they instead ask users to choose an Information Card. Some sites will choose to accept only Information Cards whereas others will give users the choice of Information Cards or other forms of authentication.

A site that accepts Information Cards typically has a login screen that contains button with a label such as "**Sign in with an Information Card**" or "**Log in using an Information Card**". Upon clicking this button, the user is presented with a choice of his Information Cards that are accepted at the site, and is asked to choose one. Once a card is selected and submitted to the site, the user is logged in and continues using the site, just as they would after submitting a username and password to a site.

Sites that accept both Information Cards and other forms of authentication present users with both an Information Card login choice and whatever other choices the site supports. For instance, a site login screen might display both "**Sign in with your username and password**" and "**Sign in with an Information Card**" buttons.

10.4 Browser Perspective

Very little additional support is needed from today's Web browsers to also support Information Cards. The main addition is that they MUST recognize special HTML and/or XHTML tags for invoking the Identity Selector, pass encoded parameters on to the Identity Selector on the platform, and POST back the token resulting from the user's choice of an Information Card.

10.5 Web Site Perspective

Web sites that employ Information Card-based authentication MUST support two new pieces of functionality: adding HTML or XHTML tags to their login page to request an Information Card-based login and code to log the user into the site using the POSTed credentials. In response to the Information Card-based login, the Web site typically writes the same client-side browser cookie that it would have if the login had occurred via username/password authentication or other mechanisms, and issue the same browser redirects. Thus, other than the code directly involved with user authentication, the bulk of a Web site can remain unchanged and oblivious to the site's acceptance of Information Cards as a means of authentication.

---

11 Invoking an Identity Selector from a Web Page

11.1 Syntax Alternatives: OBJECT and XHTML tags

HTML extensions are used to signal to the browser when to invoke the Identity Selector. However, not all HTML extensions are supported by all browsers, and some commonly supported HTML extensions are disabled in browser high security configurations. For example, while the OBJECT tag is widely supported, it is also disabled by high security settings on some browsers, including Internet Explorer.

An alternative is to use an XHTML syntax that is not disabled by changing browser security settings. However, not all browsers provide full support for XHTML.

To address this situation, two HTML extension formats are specified. Browsers MAY support one or both of the extension formats.

11.1.1 OBJECT Syntax Examples

An example of the OBJECT syntax is as follows:

```
<html>
  <head>
    <title>Welcome to Fabrikam</title>
  </head>
  <body>
    <img src='fabrikam.jpg' alt="Fabrikam Logo" />
    <form name="ctl00" id="ctl00" method="post"
       action="https://www.fabrikam.com/InfoCard-Browser/Main.aspx">
      <center>
        <img src='infocard_56x39.png' alt="Information Card Icon"
           onClick='ctl00.submit()' />
        <input type="submit" name="InfoCardSignin" value="Log in"
          id="InfoCardSignin" />
      </center>
      <OBJECT type="application/x-informationCard" name="xmlToken">
        <PARAM Name="tokenType" Value="urn:oasis:names:tc:SAML:1.0:assertion">
        <PARAM Name="issuer" Value=
           "http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self">
        <PARAM Name="requiredClaims" Value=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname">
      </OBJECT>
    </form>
  </body>
</html>
```

This is an example of a page that requests that the user log in using an Information Card. The key portion of this page is the OBJECT of type "`application/x-informationCard`". Once a card is selected by the user, the resulting Security Token is included in the resulting POST as the xmlToken value of the form. Appendix A shows a sample POST resulting from using a login page similar to the preceding one. If the user cancels the authentication request, the resulting POST contains an empty xmlToken value.

Parameters of the Information Card OBJECT are used to encode the necessary WS-SecurityPolicy information in HTML. In this example, the Relying Party is requesting a SAML 1.0 token from a Self-issued Identity Provider, supplying the requested claims "`emailaddress`", "`givenname`", and "`surname`". This example uses the basic protocol described in Section 2.1 (without employing a Relying Party STS).

A second example of the OBJECT syntax is as follows:

```
<html>
  <body>
    <form name="ctl01" method="post"
       action="https://www.fabrikam.com/InfoCard-Browser-STS/login.aspx"
       id="ctl01" onSubmit="fnGetCard();">
      <img src='infocard_56x39.png' alt="Information Card Icon"
         onClick='ctl01.submit()' />
      <input type="submit" name="InfoCardSignin" value="Log in"
         id="InfoCardSignin" />
      <OBJECT type="application/x-informationCard" name="xmlToken"
         ID="oCard" />
    </form>
    <script type="text/javascript">
    <!--
      function fnGetCard(){
        oCard.issuer = "http://www.fabrikam.com/sts";
        oCard.issuerPolicy = "https://www.fabrikam.com/sts/mex";
        oCard.tokenType = "urn:fabricam:custom-token-type";
      }
    //-->
    </script>
  </body>
</html>
```

This example uses the enhanced protocol described in Section 2.3, which employs a Relying Party STS. Note that in this case, the "issuer" points to a Relying Party STS. The "issuerPolicy" points to an endpoint where the Security Policy of the STS (expressed via WS-SecurityPolicy) is to be obtained using WS-MetadataExchange. Also, note that the "tokenType" parameter requests a custom token type defined by the site for its own purposes. The "tokenType" parameter could have been omitted as well, provided that the Web site is capable of understanding all token types issued by the specified STS or if the STS has prior knowledge about the token type to issue for the Web site.

The object parameters can be set in normal script code. This is equivalent to setting them using the "PARAM" declarations in the previous example.

11.1.2 XHTML Syntax Example

An example of the XHTML syntax is as follows:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity">
  <head>
    <title>Welcome to Fabrikam</title>
  </head>
  <body>
    <img src='fabrikam.jpg' alt="Fabrikam Logo" />
    <form name="ctl00" id="ctl00" method="post"
       action="https://www.fabrikam.com/InfoCard-Browser/Main.aspx">
      <ic:informationCard name='xmlToken'
         style='behavior:url(#default#informationCard)'
         issuer="http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self"
         tokenType="urn:oasis:names:tc:SAML:1.0:assertion">
        <ic:add claimType=
       "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"
          optional="false" />
        <ic:add claimType=
         "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname"
          optional="false" />
        <ic:add claimType=
         "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname"
          optional="false" />
      </ic:informationCard>
      <center>
        <input type="submit" name="InfoCardSignin" value="Log in"
          id="InfoCardSignin" />
```

```
    </center>
  </form>
 </body>
</html>
```

11.2 Identity Selector Invocation Parameters

The parameters to the OBJECT and XHTML Information Card objects are used to encode information in HTML that is otherwise supplied as WS-SecurityPolicy information via WS-MetadataExchange when an Identity Selector is used in a Web services context.

11.2.1 issuer

This optional parameter specifies the URL of the STS from which to obtain a token. If omitted, no specific STS is requested. The special value "`http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self`" specifies that the token SHOULD come from a Self-issued Identity Provider.

11.2.2 issuerPolicy

This optional parameter specifies the URL of an endpoint from which the STS's WS-SecurityPolicy can be retrieved using WS-MetadataExchange. This endpoint MUST use HTTPS.

11.2.3 tokenType

This optional parameter specifies the type of the token to be requested from the STS as a URI. This parameter can be omitted if the STS and the Web site front-end have a mutual understanding about what token type will be provided or if the Web site is willing to accept any token type.

11.2.4 requiredClaims

This optional parameter specifies the types of claims that MUST be supplied by the identity. If omitted, there are no required claims. The value of `requiredClaims` is a space-separated list of URIs, each specifying a required claim type.

11.2.5 optionalClaims

This optional parameter specifies the types of optional claims that MAY be supplied by the identity. If omitted, there are no optional claims. The value of `optionalClaims` is a space-separated list of URIs, each specifying a claim type that can MAY be submitted.

11.2.6 privacyUrl

This optional parameter specifies the URL of the human-readable Privacy Policy of the site, if provided.

11.2.7 privacyVersion

This optional parameter specifies the Privacy Policy version. This MUST be a value greater than 0 if a privacyUrl is specified. If this value changes, the UI notifies the user and allows them review the change to the Privacy Policy.

11.3 Data Types for Use with Scripting

The object used in the Information Card HTML extensions has the following type signature, allowing it to be used by normal scripting code:

```
interface IInformationCardSigninHelper
{
  string issuer;             // URI specifying token issuer
  string issuerPolicy;       // MetadataExchange endpoint of issuer
  string tokenType;          // URI specifying type of token to be requested
  string [] requiredClaims;  // Array of URIs of required claim types
  string [] optionalClaims;  // Array of URIs of optional claim types
  string privacyUrl;         // URL of the Privacy Policy of the site
  string privacyVersion;     // Version number of the Privacy Policy
  boolean isInstalled;       // True when an Identity Selector is available
                             // to the browser
}
```

11.4 Detecting and Utilizing an Information Card-enabled Browser

Web sites MAY choose to detect browser and Identity Selector support for Information Cards and modify their login page contents depending upon whether Information Card support is present, and which of the OBJECT and/or XHTML syntaxes are supported by the browser and supported by the Web site. This allows Information Card capabilities to be shown when available to the user, and to be not displayed otherwise.

Detecting an Information Card-enabled browser may require detecting specific browser and Identity Selector versions and being aware of the nature of their Information Card support.

11.5 Behavior within Frames

When the object tag is specified in an embedded frame, the certificate of the frame is compared to that of the root frame. For this configuration to work, the scheme, domain, and security zone (for example https, microsoft.com, and Intranet) of the URL of the embedded frame MUST be the same as that of the root frame. If they do not match, the object tag SHOULD NOT be acted upon. This prevents a form of cross-site scripting attacks.

11.6 Invocation Using the Document Object Model (DOM)

In addition to being invokable using static HTML tags and script code, Identity Selectors can be invoked from script injected into the page using the Document Object Model [DOM]. Invocation from dynamically generated script allows the Web site's requirements to be set dynamically.

11.7 Auditing, Non-Auditing, and Auditing-Optional Cards

- **Auditing Card:** When a managed card with an `ic:RequireAppliesTo` element and no `Optional` attribute or `Optional=false` attribute is used at a Web site, the Request Security Token (RST) sent to the Identity Provider contains a `wsp:AppliesTo` element.
- **Non-Auditing Card:** When a managed card with no `ic:RequireAppliesTo` element is used at a Web site, the Request Security Token (RST) sent to the Identity Provider contains no `wsp:AppliesTo` element.
- **Auditing-Optional Card:** When a managed card with an `ic:RequireAppliesTo` element with `Optional=true` attribute is used at a Web site, the Request Security Token (RST) sent to the Identity Provider contains a `wsp:AppliesTo` element.

12 Endpoint Reference wsai:Identity Property

This section adds the `wsai:Identity` property to an Endpoint Reference [WS-Addressing] and leverages extensibility of the `wsa:EndpointReferenceType` schema to include a `wsai:Identity` element as described below:

```
<wsa:EndpointReference xmlns:wsa="..." xmlns:wsai="...">
  ...
  <wsai:Identity>...identity representation...</wsai:Identity>
  ...
</wsa:EndpointReference>
```

The `wsai:Identity` element inside a `wsa:EndpointReference` can hold any of the identity representations defined in Section 12.2 below.

12.1 Default Value

If a `wsa:EndpointReference` does not contain a `wsai:Identity` element, a DNS Name representation can be assumed by extracting the hostname from the Address URI.

If the URI does not have a hostname, it does not have an implicit identity value and can not be verified by the mechanisms defined in this document.

12.2 Identity Representation

12.2.1 DNS Name

The DNS Name representation implies that the remote principal is trusted to speak for that DNS name. For instance the DNS Name representation could specify "fabrikam.com". When challenged, the endpoint contacted MUST be able to prove its right to speak for "fabrikam.com". The service could prove its right by proving ownership of a certificate containing a reference to fabrikam.com and signed by a trusted Certificate Authority. The following element of type `xs:string` can be used to represent a DNS Name representation within a `wsai:Identity` element.

```
<wsai:Dns xmlns:wsai="...">fabrikam.com</wsai:Dns>
```

12.2.2 Service Principal Name

The SPN representation implies that the remote principal is trusted to speak for that SPN, a mechanism common in intranet domains. Its format is <serviceClass>/<host>. For example, the SPN for a generic service running on "server1.fabrikam.com" would be "host/server1.fabrikam.com". The client could confidentially speak to the service and verify replies back from the service by obtaining a Kerberos ticket from the realm's domain controller. The following element of type `xs:string` can be used to represent an SPN representation within a `wsai:Identity` element.

```
<wsai:Spn xmlns:wsai="...">host/hrweb</wsai:Spn>
```

12.2.3 User Principal Name

The UPN representation implies that the remote principal is a particular user in a domain. Its format is: <user>@<domain>. For example, the UPN for a user "someone" at a domain "example.com" would be "someone@example.com". A service could prove its UPN by providing the password for the user associated with "someone@example.com". The following element of type `xs:string` can be used to represent a UPN representation within a `wsai:Identity` element.

```
<wsai:Upn xmlns:wsai="...">someone@example.com</wsai:Upn>
```

12.2.4 KeyInfo

This identity value is similar to the previous three, but rather than describing an attribute of the target, this mechanism describes a reference (embedded or external) to key material associated with the target. This allows confirmation of the target trust identity through encryption. These values can also be used to compare authenticated identities similar to the basic trust identity values by comparing the hash of the specified trust identity value with a hash of the authenticated identity of the service. The `ds:KeyInfo` element defined in [XMLDSIG] can be used.

```
<ds:KeyInfo xmlns:ds="...">...</ds:KeyInfo>
```

12.2.4.1 Example specifying an RSA Public Key

The PublicKey representation states the public key of the remote principal. A service could prove its ownership of the key by signing some data with the private key.

```
<wsai:Identity xmlns:wsai="..." ds:wsai="...">
  <ds:KeyInfo>
    <ds:RSAKeyValue>
      <ds:Modulus>xA7SEU+e0yQH5...</ds:Modulus>
      <ds:Exponent>AQAB</ds:Exponent>
    </ds:RSAKeyValue>
  </ds:KeyInfo>
</wsai:Identity>
```

12.2.4.2 Example specifying an X509 Certificate

This example shows a certificate of the remote principal being used as the identity value.

```
<wsai:Identity xmlns:wsai="..." xmlns:ds="...">
  <ds:KeyInfo>
    <ds:X509Data>
      <ds:X509Certificate>MIICXTCCA...</ds:X509Certificate>
    </ds:X509Data>
  </ds:KeyInfo>
</wsai:Identity>
```

12.2.5 Security Token

A security token can be an identity value representing statements about the identity of an endpoint. E.g.:

```
<wsai:Identity xmlns:wsai="..." ds:wsse="...">
     <wsse:BinarySecurityToken ValueType="...#X509v3">
          <!--base64 encoded value of the X509 certificate-->
     </wsse:BinarySecurityToken>
</wsai:Identity>
```

12.2.6 Security Token Reference

Similarly to `ds:KeyInfo`, `wsse:SecurityTokenReference` element can be used within a `wsai:Identity` element to reference a token representing a collection of statements about the identity of an endpoint. E.g.:

```
<wsai:Identity xmlns:wsai="..." ds:wsse="...">
     <wsse:SecurityTokenReference>
          <wsse:KeyIdentifier ValueType="...#ThumbprintSHA1">
               <!-- thumbprint of the X509 certificate  -->
          </wsse:KeyIdentifier>
     </wsse:SecurityTokenReference>
</wsai:Identity>
```

13 Security Considerations

13.1 Protection of Information Cards by Identity Selectors

It is RECOMMENDED that Identity Selectors encrypt or otherwise secure the Information Card data held by them to help protect cards from being stolen and then used by an attacker. This is particularly important for self-issued Information Cards, where possession of the unencrypted contents of a card could enable an attacker to gain access to Relying Parties accounts associated with that card.

13.2 Relying Parties Without Certificates

Because claims sent to relying parties without certificates are not encrypted, it is RECOMMENDED that sensitive claims not be released to these relying parties. Identity Providers holding sensitive user data that can be released as claim values are encouraged to issue cards containing an `ic07:RequireStrongRecipientIdentity` element to prevent transmission of sensitive claim values over an unencrypted channel.

13.3 Endpoint References

It is RECOMMENDED that Endpoint Reference elements be signed to prevent tampering.

An Endpoint Reference SHOULD NOT be accepted unless it is signed and have an associated security token to specify the signer has the right to "speak for" the endpoint. That is, the relying party SHOULD NOT use an endpoint reference unless the endpoint reference is signed and presented with sufficient credentials to pass the relying parties acceptance criteria.

It is RECOMMENDED that an endpoint reference be encrypted when it contains claims and other sensitive information.

When included in a SOAP message, endpoint references are RECOMMENDED to be protected using the mechanisms described in WS-Security [WS-Security]

14 Conformance

An implementation conforms to this specification if it satisfies all of the MUST or REQUIRED level requirements defined within this specification for the portions of the specification implemented by that implementation. Furthermore, when an implementation supports functionality in which there is a RECOMMENDED algorithm or set of parameter choices, conforming implementations MUST support the RECOMMENDED algorithm and parameter choices. A SOAP Node MUST NOT use the XML namespace identifiers for this specification (listed in Section 1.2) within SOAP Envelopes unless it is compliant with this specification.

This specification references a number of other specifications. In order to comply with this specification, an implementation MUST implement the portions of referenced specifications necessary to comply with the required provisions of the portions of this specification that it implements. Additionally, the implementation of the portions of the referenced specifications that are specifically cited in this specification MUST comply with the rules for those portions as established in the referenced specification.

Additionally, normative text within this specification takes precedence over normative outlines (as described in Section 1.1), which in turn take precedence over the XML Schema [XML Schema Part 1, Part 2] and WSDL [WSDL 1.1] descriptions. That is, the normative text in this specification further constrains the schemas and/or WSDL that are part of this specification; and this specification contains further constraints on the elements defined in referenced schemas.

If an OPTIONAL message is not supported, then the implementation SHOULD Fault just as it would for any other unrecognized/unsupported message. If an OPTIONAL message is supported, then the implementation MUST satisfy all of the MUST and REQUIRED sections of the message.

A. HTTPS POST Sample Contents

The contents of an HTTPS POST generated by a page like the first example in Section 4.1.1 follow:

```
POST /test/s/TokenPage.aspx HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Content-Length: 6478
Content-Type: application/x-www-form-urlencoded
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-sh
ockwave-flash, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Host: calebb-tst
Referer: https://localhost/test/s/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30)
UA-CPU: x86

InfoCardSignin=Log+in&xmlToken=%3Cenc%3AEncryptedData+Type%3D%22http%3A%2F%2F
www.W3.org%2F2001%2F04%2Fxmlenc%23Element%22+xmlns%3Aenc%3D%22http%3A%2F%2Fww
w.w3.org%2F2001%2F04%2Fxmlenc%23%22%3E%3Cenc%3AEncryptionMethod+Algorithm%3D%
22http%3A%2F%2Fwww.w3.org%2F2001%2F04%2Fxmlenc%23aes256-cbc%22+%2F%3E%3CKeyIn
fo+xmlns%3D%22http%3A%2F%2Fwww.w3.org%2F2000%2F09%2Fxmldsig%23%22%3E%3Ce%3AEn
cryptedKey+xmlns%3Ae%3D%22http%3A%2F%2Fwww.w3.org%2F2001%2F04%2Fxmlenc%23%22%
3E%3Ce%3AEncryptionMethod+Algorithm%3D%22http%3A%2F%2Fwww.w3.org%2F2001%2F04%
2Fxmlenc%23rsa-oaep-mgf1p%22%3E%3CDigestMethod+Algorithm%3D%22http%3A%2F%2Fww
w.w3.org%2F2000%2F09%2Fxmldsig%23sha1%22+%2F%3E%3C%2Fe%3AEncryptionMethod%3E%
3CKeyInfo%3E%3Co%3ASecurityTokenReference+xmlns%3Ao%3D%22http%3A%2F%2Fdocs.oa
sis-open.org%2Fwss%2F2004%2F01%2Foasis-200401-wss-wssecurity-secext-1.0.xsd%2
2%3E%3Co%3AKeyIdentifier+ValueType%3D%22http%3A%2F%2Fdocs.oasis-open.org%2Fws
s%2Foasis-wss-soap-message-security-1.1%23ThumbprintSHA1%22+EncodingType%3D%2
2http%3A%2F%2Fdocs.oasis-open.org%2Fwss%2F2004%2F01%2Foasis-200401-wss-soap-m
essage-security-1.0%23Base64Binary%22%3E%2BPYbznDaB%2FdlhjIfqCQ458B72wA%3D%3C
%2Fo%3AKeyIdentifier%3E%3C%2Fo%3ASecurityTokenReference%3E%3C%2FKeyInfo%3E%3C
e%3ACipherData%3E%3Ce%3ACipherValue%3EEq9UhAJ8C9K514Mr3qmgX0XnyL1ChKs2PqMjOSk
6snw%2FIRNtXqLzmgbj2Vd3vFA4VxlhileSTyqc1kAsskqpqBc4bMHT61w1f0NxU10HDor0DlNVcV
Dm%2FAfLcyLqEP%2Boh05B%2B5ntVIJzL8Ro3typF0eoSm3S6UnINOMIjHaVWyg%3D%3C%2Fe%3AC
ipherValue%3E%3C%2Fe%3ACipherData%3E%3C%2Fe%3AEncryptedKey%3E%3C%2FKeyInfo%3E
%3Cenc%3ACipherData%3E%3Cenc%3ACipherValue%3ErBvpZydiyDzJtzl1%2FjUFX9XAzO1mOR
qUypPLjh%2FBagXcf2eYwWD57v4JvnlQwGajadcDASCisazswnlskdkWgmd4IUWJpPMRH7es9zY0U
vnS4ccsakgDcmscqJpDYTrxbSBfhdvrzjDiHC2XCtoWOveoHeB51C5N8UAbff18IxCNtkWO8y3WLH
VGdVWaDOsAkK%2FK%2Fv1UgXIc51%2FtYvjefGeGbb5Nxo8DTqeDnAMQ%2B4Y%2BlaUGh1%2Ftbsr
EyJECkDgtztcxhrumbupKO%2BogWKUTTpSt851xjOFxAMiVaPZ%2FAm8V8H3ZLsR087sX%2FJ%2Bn
bRqze%2BfbdUwimN5pNoJDdMnF%2BEDLassldPsvhL4EXzuIp5deGBaqAIoaOMEUW7ssuh1PtwkEM
eqwlOzOhu%2FHtwPlqh3D02U59MtyQnJMD5UwIwO7s8J16%2BPg6Zp9HHtKKUMnkguvFmhyXS4BFS
ZVxPli8i%2B0MLO1um5dejEFd4nwGO%2FmNw6yEI8DdGVjXcYOT6JhPz9rHNh9%2F%2FOj5snJfL6
j2sg0EvIYoRs%2BhT4sdHZ95tGAiWMwT6cFOXbAQZUbYTr1ZOC6XPsfL2CFwiTM3mI%2Blco4Hc%2
F7IakIA8jwAJdtnd2mGuV67ZbY1mzibM1LUApixZj59El83ixctSQbV7iyywQ4IYN2CAq%2BCLMd1
R%2BDHfgEe8O3IVaGBDUEcd2MYimEiA7Yw3NIDrC14SbLzNvU702NpVJMeYv9q6S9xIVGApSrARsw
RFXyMbkMDp5WIQaJEXon7qLcsZONpdlX9bCcmsiikdpxmCeyS638te%2FhGBLmYJSQ0stf7BhA6B0
kwDRqdwsAa88bODiWHek0vDhAN4HlXF2%2BCxp53L9Mmvy%2FCAOI%2B9OkPL2yxS22yjWQxom%2F
yZuawsK98JHVShsIVmmbKvRM6xJwvHDSzuBAOlQKS%2FMHcFzn8vHZR4lMhm5nL3r%2B%2BumMKh0
vMuKk6JiCqG9OEj996bVIIkLzESU5Z5vT6I1Kr9Brdx8ckDElipdH3x54WVfaItHJTYU%2BsxIR1T
25fi9k%2FOc%2FMX7Q%2B6NSDs4nGqkn4rzqpez9BUWNZw7caVOrDeao85f%2FiDCGymt10A3JaSZ
dTKfzHLGmUfSkCAlVeisdvB6R7uBw8tR%2BZlgLIGS28wppFlnUYv9K7DnPrzId%2B0fHWLfL6WA%
2FEzBMMgppb5Vi%2BauHq%2BHxpCamlkrcUkzagbWNkGV8TfafkqUvRwJbxRwNVPI%2F%2Fxs%2Fp
Lcu1dh6eKcmU00%2FNx0zNOScd9XoeEU3zsV78PgvPIBT4EDuqdv4bMR6dExXvZBl%2F84blgOmhK
ZRp1F8t6EAc4LCot01ht7VOVNz25NtP27ct9QPrDJc%2FoxihT4Df6NV314vlTnu%2B%2BzVB%2BH
JAxNkiO9gx3uLUJM9XEZCDzZKihaBk2y%2F3RhsJpABVneUd%2B3sCRbQXhqKYNBHZyRAUGpMDLhL
qpjoF9x%2FNvUujQ5DBLJafxxzNVshG52jRz%2BikhCNhJDDbeA5MQ8Q7QsYcKDC0DBFsewtWaA%2
FsKxl3JU6hyTotnFS%2FoS2EzbOSvn25qZuBERsZ3W%2B5WMkRzfQadyIYOSv2DflYoljubDKyl19
St%2FbCIBgXbVIZKYtQ%2BLyepxxFjrN7cWo2aYFnB6YLurg4USJwhXzcGcvA3%2BR5dRT6Fr37U6
OcHc%2Fz2MaZmn1cQWiDGNxHtRVxEvirBc1x47hWf8RjrKzf3orL5LzgMlYc7IwclW2rbeWljCqOb
oV3d7lez%2FvNz1pxEMi4w8yUAQL8p%2FRCZ%2BpzvsgORu4RWKWiSwb17AN0JJjiW5hyZgDmxd2O
DDYffXjNiUHlmQWnDTkJXlig88mqjhOYJEal0W6L0ErwrRIy29tOiAvXZANC8kAlHexulHOe38x8E
IOaVaJtNz9mqrnmnp4Gd238txV%2BCUeWHOZaHLF4xkdtRxMAu%2FbzQ03YmUOhgxqkTfNzV6Ymne
v2nv5vsyQGJaQsNjb0M4yOe6kX2qNTwKBN2%2Bp%2Fz3f15i8KuGCgBcfP%2BP9xBizBeo7FbFtyo
2pfFhzBPmZeSOJ6kEbF1yQKHYQAT5i24SyTIfqqmwGxsQpWMstx3qJF8aW8WFzU1qXoC1LmgC1g19
rx9NYFaQshX4f729B9Ue5MX7gTrMgwAnlXty9BsoP7nzGbr3HSXy8pR%2BimuAFW3c2NaQSbjSH5Z
FOr7PZdLHsNVJzFIsaufAWr0CAEtvlPJUt7%2B%2FE5MQsMsVqMoXFmefgdxbvYlUe6MX1wtuJYYl
PAX7MHTyRUR3RfJDO054EoflVTwNE1fmocUXUh5rtFFuzy2T%2F2Y6pLAARXzo8uslAuH67VkuXv%
2BEMc7e3ogbf5%2BROsgJirZS6qkcYpfEUWqHiQYLnSIP4bt%2BW15j1bxs7yzcSCkNZ2rd%2FHWr
A41AyGMfYzqxfGcrOaxHsds3JUcByB5Zw17W58GBC32Iusqa69BFTPagEapM0Fb5CbTqXnWTNNB5J
t40BVZvLv3u5oy%2BBRaMKXZhwnbT2WUTp0Ebsn17xvte52B%2BLMlSWJn96Nl5thd%2Fi1D7PlWA
sUvpJAd0UHPizCkY8VIhcXTrs8yEwer2J2I9TQTUosmssFjoP8Lx9qMfXo0eGVmneV8kVBtu4J7Nl
QmWfV%2B%2FK8vGbCwW3Gm%2FEU1OO4ZbbK39y0JgNQ7fshxHr5Hdtd%2F6S%2FQkb6NPVDwn7Srh
Y0diWujXz5QlIYB5N7vDfMun3yF%2BGbmMExZ8MkOthuYkgMS9qiFoJGUXGyELsJfxbzdcRE9iyJn
p88L4%2BCtcO3lZJxIhMAgxOZx42RfAiDV1Gbpa4f%2F0urmWQ2VK7uZ%2FlViVrGAJ2kpH0EfwYE
Mb2YYT8FFjogqEpDSJX48BLIh1TE4nMbqQVGlckxCGDc0XyGKaF5Z7Ikw493XzOJQ0BZvaf2Kceb7
MU2lsU1DSHcQQ9X%2Bxu9RcgUePJEe9BgCMp25Kr6r43qYk79noBSgrs9kDhT5sg%2Fc20RHQB8OX
%2BC4r3XGQFWF2m2j0xTc%2Boy14xqUmSB2qJtuWGOXDJspejDRPlGIfFnqDFdqSO3%2FkV9AC5Ee
39iJGv8I%2B5nErtQao645bCytn4B2bJah8R2fXLs8Dd4%2BC2ykxVrLxTUmJaGqd2RK%2F6t1E47
1%2B90Vp4WEzC0CFXXt9XNqdVjo2bZsXbfKQgO2zT2q2qCsgwbxVzIF5y39R%2Brk8kXl6uuz3q6w
n3I5RI9M8Hn3DCzzv6Ms4rYxYuiqxaIeb7DgjI2fk1bdyiiRjSxzpCHpK6CWjBD8DPQYdkqGr%2Bs
oWeSvHvPLMSDxEPzwlnaxysRXzKphHUeUa2CCqcpagux2mbKkWHSXemX9I3V3AhPePp5XI5eCRiy3
D4%2BcBXOydie94Nz9DIhW749hPiVD9CiOAgyqgAzFwCxEEUCXKTzu9xXX4DXg9b3CUfGzwERtY7x
TGT2y%2F9I7r5Xs0lrKi9ftws4JO5v%2Be3WuAEtWv0w%2FVKClWwYbV9xtx%2B4RZQ3%2Fewvv%
2FOGqiiSrhiVBGuCDaQs7stwqfkF3vFgGXmmODGTIkIxvYm2fzcEfq4A6LRp5RkYyJyUTF87c56tn
Qa%2Bo3xeiX5WRJybpabrRou09vyWLdlkhcUaEElGWB7iYUJ9bC1tByEdNZnuDV%2FXlfnmDARKp8
RVNO28czIk57wQMuizgWrM6S9Ku2OnoDmLgbT554UBf7FnjRWOb%2FF9OJuPpUcARBPrfuqTcOsBq
tZr7AJl3zz%2F53mpyn9rgzw5gBLgkvrdbciabJOAacccTDEB5kEzCLuprC3SlVedhgY%2BMQ5%2F
xgN%2Faf3TtJiBKFVb1V37BlbXXGosnPFcoH8IOXbqW5FSsxmcnpg48poJcB7j5eHq7Y%2F01Rlb4
iMmzNap4%2BFg2F3LrwOIOWk7ueIjgFd5KJliTdalivGU%2Fchr9aTNpM5HiLb2fDW0pZ%2FFBJcI
XxpT9eNY%2FpVj5pnTW2ubpPnBulPOQTLCi1EOxbl33wnhUIfnGiVWJdrls2j3GWqqOnrYUbP%2FX
tNJqIucnMYGqPbcGIF2QRuiwD%2FiTRMvCRCmdCsYE%2FaXjOMhskX7KYC%2B9iG%2FTlwQRbfHSK
WD%2Fpv450OVDsfc1Adq6FCr1LesDNTew%2FF8Z3SiHnWS76OVsNM2SB%2FhMP67iu5UWVkb3%2FQ
qCN0aosOPs2QXOXBCZFmN6p3fhFnXPbAbaGz9y6KzUiUxC03U0fZcToKl4y%2BW0P4IvxpjVt4t8
84Q9hiBxd5xul%2BRE973a%2FyIWO%2Fit1MdUSmxWakxWuGxDnQxwkNCN7ekL%2FQ%2B6FItm86b
w9cc%2FMiI7q2IK7y7YAzM3tmamhFi%2FWJNjllHOvh%2BhNehJlLlb4Z%2F9ZtxMWV4LVIyrFaFl
```

```
RyBlhTkYbn3iQ6BUiJ0v97j7MVZHxCa1KS3t2gx8H7ts6Tfy5il89xVUdiZwfj0w06g199q1AqUMZ
EWxh0%3D%3C%2Fenc%3ACipherValue%3E%3C%2Fenc%3ACipherData%3E%3C%2Fenc%3AEncryp
tedData%3E
```

An un-escaped and reformatted version of the preceding xmlToken value, with the encrypted value elided, is as follows:

```
<enc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element" xmlns:enc=
"http://www.w3.org/2001/04/xmlenc#">
<enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"
/>
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<e:EncryptedKey xmlns:e="http://www.w3.org/2001/04/xmlenc#">
<e:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1
p">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
</e:EncryptionMethod>
<KeyInfo>
<o:SecurityTokenReference xmlns:o="http://docs.oasis-open.org/wss/2004/01/oas
is-200401-wss-wssecurity-secext-1.0.xsd">
<o:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-mes
sage-security-1.1#ThumbprintSHA1" EncodingType="http://docs.oasis-open.org/ws
s/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary">
+PYbznDaB/dlhjIfqCQ458E72wA=
</o:KeyIdentifier>
</o:SecurityTokenReference>
</KeyInfo>
<e:CipherData>
<e:CipherValue>
Eq9UhAJ8C9K514Mr3qmgX0XnyL1ChKs2PqMj0Sk6snw/IRNtXqLzmgbj2Vd3vFA4Vx1hileSTyqcl
kAsskqpqBc4bMHT61w1f0NxU10HDor0D1NVcVDm/AfLcyLqEP+oh05B+5ntVIJzL8Ro3typF0eoSm
3S6UnINOHIjHaVWyg=
</e:CipherValue>
</e:CipherData>
</e:EncryptedKey>
</KeyInfo>
<enc:CipherData>
<enc:CipherValue>
...=
</enc:CipherValue>
</enc:CipherData>
</enc:EncryptedData>
```

B. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Original Authors of the initial contributions:

Arun Nanda, Microsoft Corporation
Michael B. Jones, Microsoft Corporation
Jan Alexander, Microsoft
Giovanni Della-Libera, Microsoft
Martin Gudgin, Microsoft
Kirill Gavrylyuk, Microsoft
Tomasz Janczuk, Microsoft
Michael McIntosh, IBM
Anthony Nadalin, IBM
Bruce Rich, IBM
Doug Walter, Microsoft

Participants:

John Bradley, Individual
Norman Brickman, Mitre Corporation
Jeffrey Broberg, CA
Scott Cantor Internet2
Ruchith Fernando, WSO2
Marc Goodner, Microsoft Corporation (Chair)
Patrick Harding, Ping Identity
Andrew Hodgkinson, Novell
Mario Ivkovic, A-SIT, Zentrum für Sichere Informationstechnologie - Austria
Michael B. Jones, Microsoft Corporation (Editor)
Mike Kirkwood, Polka Networks
Herbert Leitold, A-SIT, Zentrum für Sichere Informationstechnologie - Austria
Michael McIntosh, IBM (Editor)
Dale Olds, Novell
Anthony Nadalin, IBM (Chair)
Drummond Reed, Cordance
Bruce Rich ,IBM
Darran Rolls, SailPoint Technologies
Prabath Siriwardena, WSO2

Appendix
OASIS04

DOI: 10.5220/0002804203330339 • Corpus ID: 5426930

Share

Information Cards and Affirmative Statements

M. Ivkovic, Martin Centner • Published in International Conference on... 2010 • Computer Science, Law

TLDR This paper presents an approach to use Citizen Cards together with Information Cards for identification and authentication in e-government services. Expand

View PDF Save to Library Create Alert Cite

Figures Topics 6 References Related Papers

Figures from this paper

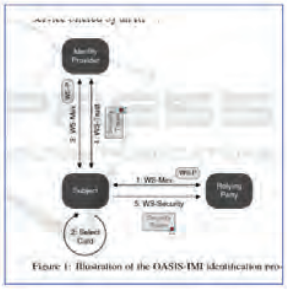


Figure 1



Figure 2



Figure 3

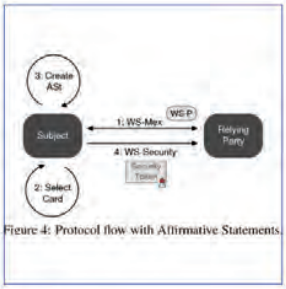


Figure 4: Protocol flow with Affirmative Statements.

Topics

AI-Generated

Authentication Information Cards E-government Services

6 References

Date Range Citation Type Has PDF Author More Filters Sort by Most Influe...

The Laws of Identity

Kim Cameron A. Durand +33 authors W. Heath Computer Science, Law 2005

TLDR The approach has been to develop a formal understanding of the dynamics causing digital identity systems to succeed or fail in various contexts, expressed as the Laws of Identity, which define a unifying identity metasystem that can offer the Internet the identity layer it so obviously requires. Expand

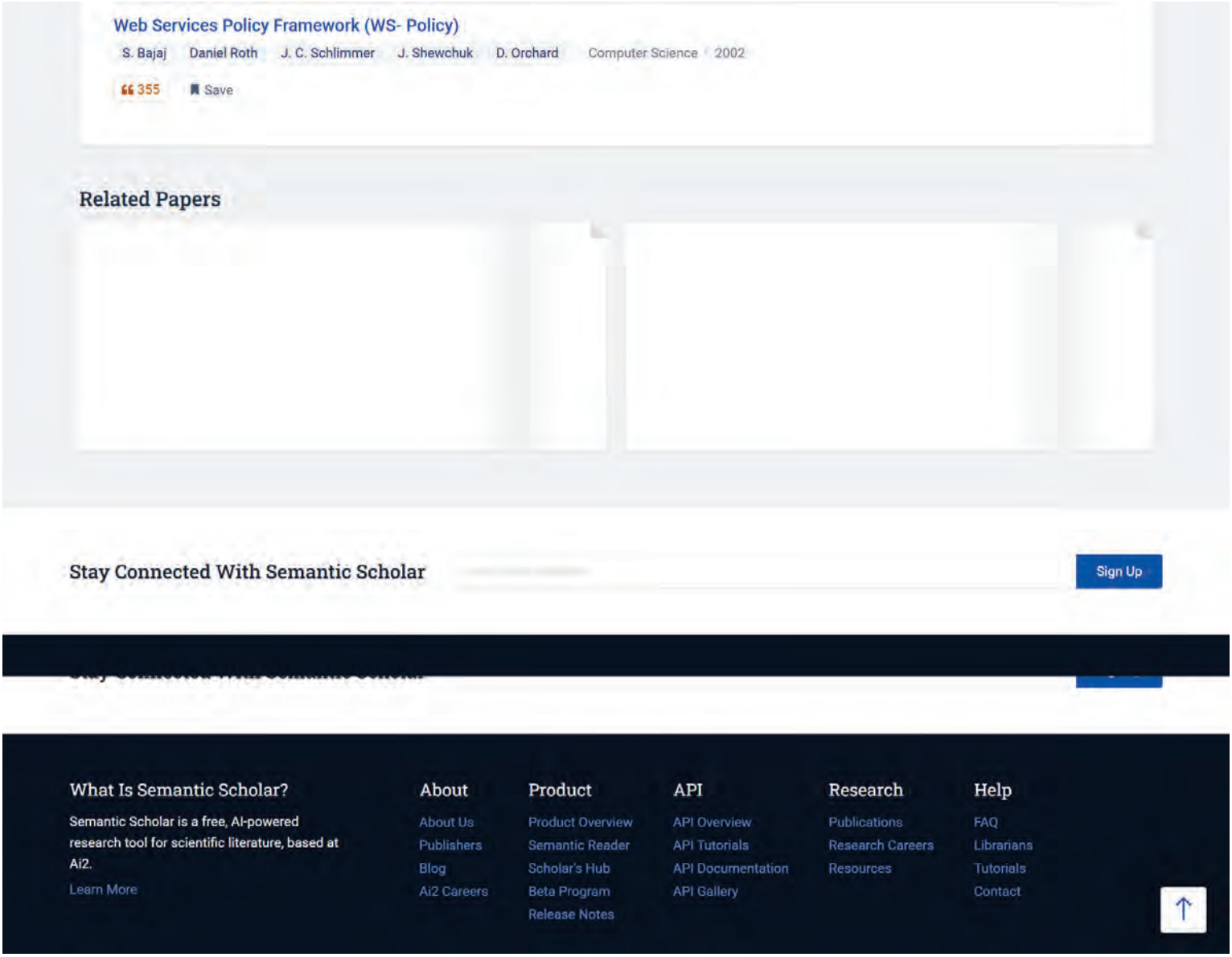
295 3 Excerpts Save

Web services security: soap message security

A. Nadalin Chris Kaler Phillip Hallam-Baker Ronald Monzillo Computer Science 2003

TLDR This specification describes enhancements to SOAP messaging to provide message integrity and confidentiality and includes extensibility mechanisms that can be used to further describe the characteristics of the tokens that are included with a message. Expand

331 2 Excerpts Save


Web Services Policy Framework (WS- Policy)
S. Bajaj Daniel Roth J. C. Schlimmer J. Shewchuk D. Orchard Computer Science 2002
355 Save
Related Papers
Stay Connected With Semantic Scholar
Sign Up
What Is Semantic Scholar?
Semantic Scholar is a free, AI-powered research tool for scientific literature, based at Ai2.
Learn More
About
About Us
Publishers
Blog
Ai2 Careers
Product
Product Overview
Semantic Reader
Scholar's Hub
Beta Program
Release Notes
API
API Overview
API Tutorials
API Documentation
API Gallery
Research
Publications
Research Careers
Resources
Help
FAQ
Librarians
Tutorials
Contact

INFORMATION CARDS AND AFFIRMATIVE STATEMENTS

Mario Ivkovic and Martin Centner
Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Austria

Keywords: Affirmative statements, Citizen cards, e-Government, Identity metasystem, Information cards.

Abstract: E-government services require strong methods of identification and authentication in order to protect personal rights and to comply with corresponding laws. The requirements for the authentication process can be fulfilled by electronic signatures. Identification in e-government applications often relies on government-issued identifiers provided by electronic identity (eID) cards. An eID card with signature creation capabilities is typically called Citizen Card. The Information Cards technology, a recently introduced user-centric identity management framework, gains more and more importance if the field of eID. Expecting a high importance of Information Cards in the future, it would be very reasonable to utilize them for e-government services. In this paper we present an approach to use Citizen Cards together with Information Cards for identification and authentication in e-government services.

1 INTRODUCTION

The Identity Metasystem concept gains more and more importance in the field of eID. For several reasons we think that Information Cards and the Identity Metasystem will play an important role in the near future. The first reason is that Information Cards is an open standard which is the basis for a wide distribution. OASIS published version 1.0 of the *Identity Metasystem Interoperability (IMI)* standard in July 2009. The second indicator for a possible success is the increasing number of implementations. Despite Microsoft's CardSpace, which was the first available implementation on the market, several other vendors released their commercial and non-commercial Identity Metasystem products (e.g. Novell's DigitalMe[1], IBM's Higgins[2], Azigo[3], etc.). The third important reason is the industry interest. On the one hand, some editors of the OASIS standard represent big market players and on the other hand, the so-called Information Card Foundation[4] including industry leaders with the aim to advance the use of Information Cards has been created.

Expecting the wide distribution of the Identity Metasystem in the future, it would be very reasonable to utilize it for e-government services. Identification and authentication in e-government application often relies on government-issued identifiers provided by eID cards with signature creation capabilities. This combination of eID card and signature creation device is typically called Citizen Card. The goal is now to combine identification and authentication mechanisms provided by Citizen Cards with the Information Cards technology. Unfortunately, for several reasons this cannot be done directly. Some e-government services for example, call for written statements signed by the user with a qualified electronic signature which is not supported by the OASIS-IMI specification. Furthermore, in some countries it is not intended that providers of government-issued identifiers are directly involved in the identification process. In other words, because of several reasons some countries don't want to conduct an IdP as defined in OASIS-IMI. OASIS-IMI supports self-issued identifiers and identifiers issued by an IdP during the identification process. However, the IMI protocol does not support the required pre-issued governmental identifiers. Additionally, OASIS-IMI only describes the usage of the RSA signature algorithm. Many Citizen Cards (e.g. the Austrian Citizen Card) use other algorithms, which would result in interoperability issues.

In this paper we show how this issues can be solved by our new introduced *Affirmative Statement*

[1] http://code.bandit-project.org/trac/wiki/DigitalMe
[2] http://eclipse.org/higgins/
[3] http://www.azigo.com/
[4] http://informationcard.net/

Ivkovic M. and Centner M.
INFORMATION CARDS AND AFFIRMATIVE STATEMENTS.
DOI: 10.5220/0002804203330339
In *Proceedings of the 6th International Conference on Web Information Systems and Technology (WEBIST 2010)*, page
ISBN: 978-989-674-025-2
Copyright © 2010 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

Claim. The Affirmative Statement Claim contains a statement in natural language signed by the user with a qualified electronic signature. The signed Claim also contains the required government-issued identifier. The Affirmative Statement Claim can be added like any other Claim without the need to change the existing specification or implementations. To verify our approach we have extended an existing implementation and successfully tested with existing service providers and a self-written service provider.

Having solved these problems, Information Cards can help to improve or enhance existing solutions. On the one hand we consider general aspects of Information Cards where existing eID solutions can benefit from and on the other hand we consider features and characteristics that could enhance the Austrian eID landscape in particular.

Same Look&Feel for Users: The Austrian eID solution, like other national eID implementations, is partly based on specially developed identification and authentication protocols (see Section 2). If the Information Card technology could be used for e-government purposes, people would have only one means for identification and authentication. They could then use the same technology for Austrian e-government services as well as for private-sector services already utilizing the OASIS-IMI protocol.

No Additional Software for Users Required: Not least because of mobility reasons (e.g. access to services in internet cafes or at kiosks), it is very appropriate to avoid the need for specific software on the client side to access a service. Identity Metasystem implementations that are already integrated into the operating system, could help to clear this hurdle.

No Additional Software for Relying Parties Required: A Relying Party (RP) offering a service usually wants to have wide appeal, but do not want to implement a wealth of identification and authentication protocols. Moreover, it is not very likely that global service providers will implement national solutions. Therefore, the application of Information Cards could be a way towards the propagation of national eID solutions.

Card Roll-out: In Austria the penetration of cards that can be used as Citizen Card is very high (e.g. approximately 8.5 million[5] social security cards and more than 7 million bank cards[6]). If the Austrian eID solution could be combined with the OASIS-IMI protocol, service providers would have a high number of potential users.

The rest of this paper is structured as follows. In the next section we describe the requirements on electronic signatures to fulfill the needs for identification and authentication in e-government applications and we briefly explain the Austrian Citizen Card concept. The third section of this paper describes the concept of Information Cards and the underlying Identity Metasystem. Starting from the idea and the evolution of the Identity Metasystem, we explain the main concepts behind it and describe the protocol flow of a typical authentication process. In the fourth section (Affirmative Statements), we present how the Austrian Citizen Card could be used with Information Cards for identification and authentication in e-government services. In the last section, we finalize the paper with some conclusions.

2 E-GOVERNMENT SERVICES AND THE AUSTRIAN CITIZEN CARD

E-Government services – such as services that provide access to personal and sensitive data – usually require strong methods of identification and authentication in order to protect personal rights and to comply with corresponding laws.

Electronic signatures provide this means of identification and authentication in many electronic authentication schemes. The EU Directive on electronic signatures (Directive 1999/93/EC, 1999) defines a legal framework for electronic signatures. This Directive has been adopted into national legislation by all EU Member States. It defines special requirements for *advanced electronic signatures*, *qualified certificates* and *secure signature-creation devices*. Advanced electronic signatures based on a qualified certificate created with a secure signature-creation device (hereafter referenced to as *qualified signatures*) satisfy the same legal requirements as handwritten signatures and are admissible as evidence in legal proceedings.

The EU Directive on services in the internal market (Directive 2006/123/EC, 2006) requires Member States to ensure that foreign service providers wishing to provide their services in the respective Member State are able to complete all procedures and formalities at a point of single contact. As the Service Directive calls for providing electronic means for completing procedures and formalities, the recognition of qualified signatures by foreign EU Member States is

[5] According to http://www.chipkarte.at

[6] According to http://www.bankomatkarte.at

of ever-growing importance.

Considering the above mentioned EU Directives qualified signatures can be a very strong instrument for identification and authentication with recognition by foreign EU Member States. They are therefore very well suited for identification and authentication in e-government services.

Identification in e-government applications usually relies on government-issued identifiers. Such an identifier may be based on the social insurance number, the tax account number, the number in the register of residents, etc. – whatever allows to uniquely identify a citizen or legal entity. The identifier provided to a particular service may also be a derived identifier that uniquely identifies a citizen within a specific sector of public administration but is different from the identifiers used in other sectors. Such sector specific identifiers were introduced in some countries (e.g. in Austria) to prevent systematic correlation of personal data from different sectors of public administration.

Government-issued identifiers may be provided by electronic identity cards. Such electronic identity cards are often implemented as smart cards that also act as secure signature-creation device for creating qualified signatures. The combination of electronic identity card and secure signature-creation device is often called *Citizen Card*.

The Austrian E-Government Act (EGov-Act, 2004) defines a Citizen Card as a logical unit, independent of its technical implementation that combines qualified electronic signatures with an *Identity Link*. The Identity Link provides a mechanism for electronic identification by linking a qualified certificate to the citizen's government-issued identifier (the so-called *source PIN* derived from the citizen's number in the register of residents). The source PIN may never directly be used as identifier in e-government services. Instead, a sector-specific personal identifier has to be derived from the source PIN.

Two major types of Citizen Cards are currently available in Austria: Citizen Cards based on smart cards such as the electronic health insurance card, bank cards and other electronic signature cards and Citizen Cards based on mobile phones for authorizing a server generated electronic signature.

All Austrian Citizen Card implementations provide their functions via a common interface (*Security Layer*) specified in the Austrian Citizen Card specification (Hollosi and Karlinger, 2004). Smart card based implementations require a software (called *Citizen Card Environment*) to externally offer the Security Layer interface. Such software is available from different vendors and has to be installed on the citizen's PC. In addition, one implementation is available that does not require the installation on the citizen's PC but is provided by a server and uses a lightweight component, executed in the citizen's browser, to access the smart card.

The Security Layer interface is designed to be directly accessible by applications on the citizen's PC or by web applications via the citizen's web browser. For this purpose the Security Layer interface provides a simple XML based protocol over an HTTP interface.

Identification and authentication with the Austrian Citizen Card involves two steps:

1. Deriving a sector specific identifier, reading and validating the Identity Link.
2. Creating and validating a qualified signature.

The first step serves as method for electronic identification based on sector specific identities as discussed above. The second step provides authentication, based on a qualified signature.

The Austrian Electronic Signature Act (Sig-G, 1999) demands that the secure signature-creation device must not prevent the signatory from viewing the data to-be signed. Therefore, for any data to-be signed a meaningful way of displaying it to the signatory must exist. Authentication based on Austrian Citizen Cards fulfils this requirement by requesting the citizen to sign a text written in natural language. The text may look like this:

> I *name and date of birth* apply for access to *the service* with my electronic signature.
> *Sector specific personal identifier*
> *Date and time instant*

Some e-government services have additional legal requirements that must be fulfilled. Some services for example call for written statements. As discussed at the beginning in this section, qualified signatures are able to fulfil the same legal requirements as handwritten signatures. Therefore, electronic applications signed with qualified signatures are able to meet the requirements of written form. The needs of E-Government services that require written form for access (e.g. services that provide access to personal and sensitive data) could be met by providing appropriate text to be signed with a qualified signature.

3 INFORMATION CARDS

The idea to develop a new identity system was born in order to banish the use of passwords together with all its disadvantages from the Web. Therefore, Kim Cameron, nowadays the Chief Architect of Identity

in the Identity and Security Division at Microsoft, started a blog[7] to publicly discuss the issues and requirements for a possible solution. Out of this long and extensive discussion he extracted the widely known 7 laws of identity and described the concept of the Identity Metasystem, a user-centric identity management framework (Cameron, 2005).

From a user's point of view, the Identity Metasystem serves as digital equivalent of a wallet which is filled with various types of documents representing the user's identity (e.g. identity card, driving license, credit card, ...). During an identification process the user is able choose which identity document, and thus, which identity data to disclose. The digital equivalent of an identity document within the Identity Metasystem is called Information Card.

The Identity Metasystem has been designed as a system of systems and it is not based on any specific technology (Cameron, 2005):

> "We need a unifying identity metasystem that can protect applications from the internal complexities of specific implementations and allow digital identity to become loosely coupled. This metasystem is in effect a system of systems that exposes a unified interface much like a device driver or network socket does. That allows one-offs to evolve towards standardized technologies that work within a metasystem framework without requiring the whole world to agree a priori."

The concept of the Identity Metasystem has been then taken up by OASIS[8] and the Identity Metasystem Interoperability (IMI) technical committee has been formed to develop an open and public standard. Version 1.0 of the Identity Metasystem Interoperability standard has been published in July 2009 (OASIS-IMI, 2009). All remaining descriptions and deliberations of this paper refer to the OASIS standard.

In the following, the basic concepts and involved parties that are necessary for the following protocol description and this paper in general are described.

Subject. A Subject is a person or entity that wants to use a service offered by an RP. A subject owns a Digital Identity.

Identity Provider. An Identity Provider (IdP) is an entity that issues Digital Identities to Subjects. An IdP could be a government agency that issues Digital Identities to the citizens, or a credit card provider for example. It is also possible that a Subject issues a Digital Identity to itself, which is then called a self-issued identity.

Relying Party. An RP is an entity offering a service to Subjects and relies on Digital Identities.

Information Card. An Information Card is a visualization of a Subject's Digital Identity.

Claim. A Claim is a specific information about a Subject asserted by an IdP. Typical Claims are a person's given name, last name or date of birth.

Digital Identity. A Digital Identity is a set of Claims concerning a particular Subject.

Security Token. A Security Token is basically the representation of an asserted Digital Identity that contains a set of Claims. Typically, a Security Token is encoded as a SAML v1.1 Assertion[9].

Figure 1 shows the basic communication steps of an authentication process using the OASIS-IMI protocol. In this example a Subject wants access to a restricted service offered by an RP[10].

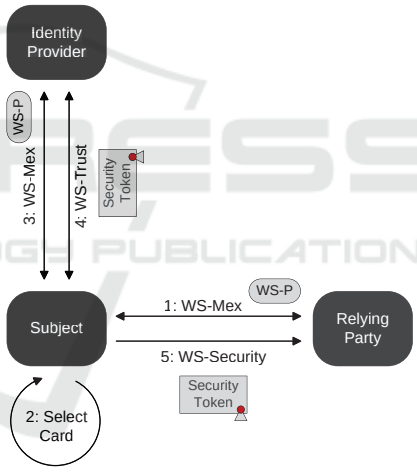


Figure 1: Illustration of the OASIS-IMI identification protocol flow.

In Step (1), the Subject and the RP negotiate the policy that should be applied for this authentication process using WS-MetadataExchange (K. Ballinger et

[7]http://www.identiyblog.com

[8]OASIS, Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org/

[9]Security Assertion Markup Language (SAML) v1.1, http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf

[10]Please note that this protocol description relates to the generic protocol flow using WS-* protocols between all parties. In case a user wants to access a restricted Web site using a Web browser, which typically doesn't support WS-* protocols, the protocol flow differs from this one. In such a case the communication between browser and RP would be plain HTTP or HTTPS, respectively (OASIS-IMI, 2009).

al., 2006). The policy itself is in turn defined using WS-Policy (S. Bajaj et al., 2006). Based on the received policy the Subject selects an Information Card and thus the Digital Identity that should be disclosed in Step (2). The component that holds the Information Cards and allows to choose which identity should be disclosed is called Identity Selector or Card Selector. The selected card, in case it is not a personal card (a card representing a self-issued digital identity), contains the information that is necessary to contact the responsible IdP. In Step (3), the Subject, or to be precise the Identity Selector on behalf of the Subject, negotiates the policy with the IdP, again using WS-MetadataExchange and WS-Policy. The Subject requests a Security Token based on the previously received policy using WS-Trust (OASIS WS-Trust, 2007) in Step (4). In order to request the token, the Subject needs to authenticate to the IdP. In the OASIS-IMI standard, four different authentication mechanisms are defined. These are username and password, kerberos v5 credential, X.509 certificate credential, and a self-issued Security Token (OASIS-IMI, 2009). A self-issued Security Token is a Security Token issued by the subjects Card Selector using a personal card and not by an IdP as in case of a managed card. In Step (5), the Subject transmits the received Security Token to the RP using WS-Security (A. Nadalin et al., 2004). After receiving the Security Token the RP verifies the token and grants access to the desired service.

4 AFFIRMATIVE STATEMENTS

The previous sections gave an introduction to Information Cards, identification and authentication in e-government and the Austrian Citizen Card. In this section we will present our approach to use a Citizen Card together with Information Cards for identification and authentication in e-government services.

Our aim is to combine the strong means of authentication attained by qualified electronic signatures and identification provided by a Citizen Card with the common user experience of Information Cards. Therefore, it is desirable to have the user's Citizen Card appearing as just another card in the Card Selector, such that the Citizen Card may be used in a manner similar to the use of any other information card.

A Citizen Card carries the same information that is also typically available with Information Cards. Given name, family name and date of birth are Claims often required by RPs. In addition the qualified certificate on the Citizen Card often includes the signatory's e-mail address – another Claim that is typically required by RPs. For RPs providing e-government services, the most important Claim will however be the government-issued identifier (or an derived identifier as discussed in section 2).

As already mentioned Citizen Card based identification relies on pre-issued identities. The only third party interaction required upon identification and authentication is obtaining revocation status information for the qualified certificate. There is no identity provider involved in the identification and authentication process.

The Information Card model introduces the concept of *self-issued* Security Tokens for authentication without third-party IdP interaction. In the Information Card model, as defined in OASIS-IMI (OASIS-IMI, 2009), self-issued Security Tokens always base on self-asserted identities. We are however going to use self-issued Security Tokens to provide government-issued, and therefore government-asserted and not self-asserted, identity to the RP.

Security Tokens are signed by the issuer before they are used for authentication at the RP. As our aim is to use qualified electronic signatures for authentication, at a first glance it seems obvious to apply a qualified electronic signature directly on the self-issued Security Token. Considering the following issues we have however chosen to follow a different approach.

- OASIS-IMI does not describe the usage of signature algorithms other than RSA. However, many Citizen Cards use different signature algorithms (e.g. ECDSA). Therefore, using such signature algorithms would most likely result in interoperability issues with existing RP implementations.
- As discussed in section 2, national implementations of the Signature Directive require that the signatory must have the ability to view the data before signing. However, there is no standard representation of a Security Token in a form meaningful for an end user. Transformation of the Security Token in an end user readable form before signing would violate the OASIS-IMI specification and would therefore again result in incompatibility to existing RP implementations.

For the given reasons we have decided to define a new *Affirmative Statement* (ASt) Claim. The Claim includes a statement in natural language similar to the one shown at the end of section 2 and has to be signed by the user with a qualified signature for the following authentication process. The ASt Claim also includes the government-issued identifier (either the identifier itself or a derived identifier thereof). This generated Claim is added to a self-issued Security Token, which

is then signed as specified in OASIS-IMI (see Figure 2). Therefore, a token-signing key has to be derived from the secret master key of the corresponding information card. If this master key in turn is derived from a secret property of the Citizen Card, there is no need to store any information about the Citizen Card in the Card Selector. Thus, a Citizen Card backed Information Card behaves just like a *roaming* card, which may be easily used with different Card Selectors without the need to transfer information.

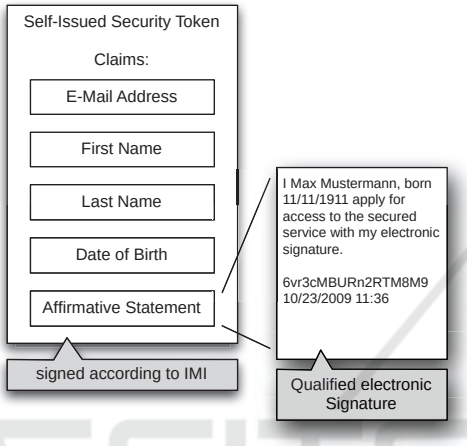


Figure 2: Self-issued Security Token with an Affirmative Statement.

Security Tokens with the additional ASt Claim can be processed as regular tokens by standard RP implementations.The Affirmative Statement Claim can be retrieved from the RP implementation like any other Claim. It is now the responsibility of a Citizen Card aware service to validate the Affirmative Statement Claim and extract the government-issued identifier. Of course this moves some responsibility to the service implementation. However, RPs which do not require this additional Affirmative Statement Claim can simply ignore it and treat the token like a regular self-asserted Security Token.

Card Selector implementations must be extended to support the introduced Affirmative Statement Claim. They need to be able to retrieve the required information from the Citizen Card and means for requesting the user to sign the Affirmative Statement with a qualified electronic signature. To verify our approach and to gain experience we have extended a Card Selector provided by the *Higgins Project*. Higgins offers several Card Selector implementations (e.g. GTK and Cocoa Selector, Firefox-Embedded Selector, iPhone Selector, Android Selector, . . .). Because of existing Java libraries for the required Citizen Card interaction, we have decided to extend the Higgins Java RCP Card Selector[11] implementation.



Figure 3: Citizen Card as Personal Card within the Higgins Card Selector.

To adapt the Higgins RCP Selector for our needs, we have implemented an additional *Personal Card*, in our case called Citizen Card, (see Figure 3) that supports the ASt Claim. If an RP requests an ASt Claim, the extended selector highlights the Citizen Card Information Card as supported card. In case the user selects this card, a Security Layer request (see Section 2 is submitted to retrieve the first name, last name, and date of birth stored on the user's Citizen Card. After retrieving this information the user can decide whether to transmit this card or not. If the user then decides to transmit this card by pressing the submit button, a second Security Layer request is generated and sent to the Citizen Card. With this request the user creates an Affirmative Statement with an applied qualified electronic signature as described above. This Affirmative Statement is then packed into the Security Token which in turn is signed according to the OASIS-IMI specification and transmitted to the RP. Figure 4 illustrates the protocol flow described above.

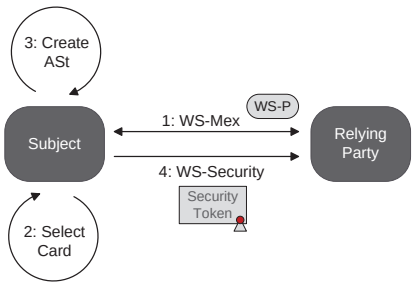


Figure 4: Protocol flow with Affirmative Statements.

[11] http://wiki.eclipse.org/RCP_Selector_1.0

5 CONCLUSIONS

With the presented approach it is possible to use Information Cards for identification and authentication with a Citizen Card without requiring any changes to the underlying protocols. RP implementations do not need to be aware of the Affirmative Statement Claim but need only to be able to hand it over to the application for further processing. Card Selectors however, must be extended to support Citizen Card backed Information Cards. While such integration is not provided in standard Card Selectors, the user may decide to install an extended Citizen Card enabled Card Selector as discussed above. However, it would be possible to integrate standard support for Affirmative Statements in Card Selectors on the same bases as signature creation devices in different operating systems.

Our concept of an Affirmative Statement Claim that solves the issues with existing e-government services has been proposed to the OASIS-IMI technical committee. OASIS added this Claim to their list of issues and the need for an Affirmative Statement Claim has been discussed and perceived in the technical committee.

The reliability and trustworthiness of an IdP is essential for the security of authentication schemes requiring IdP interaction. The approach we presented does not involve IdP interaction but uses qualified signatures for authentication. Thus, it relies on a well established legal framework as well as a system for accreditation and supervision of certification authorities. Therefore, RPs do not need to establish trust to any further parties.

The presented approach also provides Information Cards with a high degree of portability. A Citizen Card may be used with any supporting Card Selector without the need of prior information transfer.

REFERENCES

A. Nadalin et al. (2004). Web Services Security: SOAP Message Security 1.0. http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf.

Cameron, K. (2005). The Laws of Identity. http://www.identityblog.com/stories/2005/05/13/TheLawsOfIdentity.pdf.

Directive 1999/93/EC (1999). Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31999L0093:EN:HTML.

Directive 2006/123/EC (2006). Directive 2006/123/EC of the European Parliament and of the Council of 12 December 2006 on services in the internal market. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32006L0123:EN:HTML.

EGov-Act (2004). The Austrian E-Government Act – Federal Act on Provisions Facilitating Electronic Communications with Public Bodies. http://www.digitales.oesterreich.gv.at/DocView.axd?CobId=31191.

Hollosi, A. and Karlinger, G. (2004). The Austrian Citizen Card. http://www.buergerkarte.at/konzept/securitylayer/spezifikation/20040514/introduction/Introduction.en.html.

K. Ballinger et al. (2006). Web Services Metadata Exchange (WS-MetadataExchange), Version 1.1. http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf.

OASIS-IMI (2009). Identity Metasystem Interoperability, Version 1.0, OASIS Standard. http://docs.oasis-open.org/imi/identity/v1.0/os/identity-1.0-spec-os.pdf.

OASIS WS-Trust (2007). WS-Trust 1.3, OASIS Standard. http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf.

S. Bajaj et al. (2006). Web Services Policy Framework (WS-Policy), Version 1.2. http:// specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf.

Sig-G (1999). Bundesgesetz über elektronische Signaturen (Signaturgesetz – SigG). available in German only, http:// www.ris.bka.gv.at/ GeltendeFassung.wxe?Abfrage = Bundesnormen&Gesetzesnummer=10003685.

Appendix
OASIS05

ScienceDirect Journals & Books Help Search My account Sign in

Access through your organization Purchase PDF

Article preview

Abstract

Introduction

Section snippets

References (73)

Cited by (22)

Engineering Applications of Artificial Intelligence

Volume 26, Issue 1, January 2013, Pages 96-109

Magentix2: A privacy-enhancing Agent Platform

Jose M. Such, Ana García-Fornes, Agustín Espinosa, Joan Bellver

Show more

Add to Mendeley Share Cite

https://doi.org/10.1016/j.engappai.2012.06.009 Get rights and content

Abstract

Agent Platforms are the software that supports the development and execution of Multi-agent Systems. There are many Agent Platforms developed by the agent community, but they hardly consider privacy. This leads to agent-based applications that invade users' privacy. Privacy can be threatened by two main information activities: information collection and information processing. Information collection can be prevented using traditional security mechanisms. Information processing can be prevented by minimizing data identifiability, i.e., the degree by which personal information can be directly attributed to a particular individual. However, minimizing data identifiability may directly affect other crucial issues in Multi-agent Systems, such as accountability, trust, and reputation. In this paper, we present the support that the Magentix2 Agent Platform provides for preserving privacy. Specifically, it provides mechanisms to avoid information collection and information processing when they are not desired. Moreover, Magentix2 provides these mechanisms without compromising accountability, trust, and reputation. We also provide in this paper an application built on top of Magentix2 that exploits its support for preserving privacy. Finally, we provide an extensive evaluation of the support that Magentix2 provides for preserving privacy based on that application. We specifically test whether or not privacy loss can be minimized by using the support that Magentix2 provides, whether or not this support introduces a bearable performance overhead, and whether or not existing trust and reputation models can be implemented on top of Magentix2.

Highlights

► We present the support that Magentix2 provides to enhance privacy. ► Magentix2 avoids information collection and information processing. ► Information processing is avoided without compromising accountability, trust and reputation. ► We describe an application that exploits the support that Magentix2 provides. ► We provide an extensive evaluation of the privacy features of Magentix2.

Recommended articles

Social media research in the industrial marketing field: Revie...
Industrial Marketing Management, Volum...
Jari Salo

Investigation and comparison of the anti-tumor activities of...
Journal of Dairy Science, Volume 102, Issu...
H.Y. Li, ..., N. Zheng

Forecasting stock index price using the CEEMDAN-LSTM model
The North American Jou
Yu Lin, ..., Feng Ma

FEEDBACK

Show 3 more articles

Article Metrics

Citations

Citation Indexes 22

Captures

Mendeley Readers 32

PLUMX View details

Introduction

A Multi-agent System (MAS) consists of a number of agents that interact with one-another (Wooldridge, 2002). MAS represents a key issue, especially from the development point of view in Distributed Artificial Intelligence (DAI). This is because the MAS community has produced both methodologies and actual frameworks to make the implementation of agent-based applications possible. In particular, Agent Platforms (APs) are the software that supports the development and execution of MAS. APs provide all the basic infrastructure (for message handling, tracing and monitoring, run-time management, and so on) required to create MAS (Wooldridge, 2002).

There are many APs developed by the MAS community—for an overview of current APs and the features they provide refer to Alberola et al. (2010). However, privacy is seldom considered (Piolle et al., 2007, Such et al., in press). This leads to agent-based applications that invade individuals' privacy. This is due to the fact that an agent usually encapsulates personal information describing its principal[1] Fasli (2007a), such as preferences, names, and other information. Moreover, agents carry out interactions on behalf of their principals so that they exchange personal information. For instance, agents act on behalf of their principals in agent-mediated e-commerce (Sierra, 2004), as personal assistants (Mitchell et al., 1994), in virtual worlds like Second Life[2] (Weitnauer et al., 2008), as recommenders (Montaner et al., 2003), and so on.

The modern conception of privacy started more than a hundred years ago, with the seminal work of Warren and Brandeis (1890) *The right of privacy*. These two lawyers defined privacy as "the right to be let alone". They were pioneers in considering the implications of technology in privacy. Specifically, they were very concerned about the implications of instantaneous photographs and portraits in injuring the feelings of the people in those photographs and portraits. Privacy was later recognized as a fundamental human right by the United Nations Declaration of Human Rights, the International Covenant on Civil and Political Rights, the Charter of Fundamental Rights of the European Union, and many other international treaties (Acquisti et al., 2008).

In the second part of the 20th century, Alan Westin defined privacy as "the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated" (Westin, 1967). This is what is currently known as the informational self-determination right (Rannenberg et al., 2009). The concept of informational self-determination changed the right to privacy from the right to be let alone to its current incarnation as a means to limit the abuse of personal data (Schermer, 2007). Informational self-determination represents today's European understanding and regulation of privacy in the context of information and communication technology (EU Directives 95/46/EC, 45/2001/EC, and 2002/58/EC).

Despite all these regulations, as the Internet has no governing or regulating body, privacy breaches are still possible. Nowadays, in the era of global connectivity (everything is inter-connected anytime and everywhere) with more than 2 billion world-wide users with connection to the Internet as of 2011,[3] privacy is of great concern. In the real world, everyone decides (at least implicitly) what to tell other

people about themselves. In the digital world, users have more or less lost effective control over their personal data. Users are therefore exposed to constant personal data collection and processing without even being aware of it Fischer-Hübner and Hedbom (2008). Garfinkel (2001) suggests that nowadays users have only one option to preserve their privacy: becoming hermits and not using online social networks, e-commerce sites, etc. Considering the increasing power and sophistication of computer applications that offer many advantages to individuals, becoming a hermit may not really be an option. However, all of these advantages come at a significant loss of privacy (Borking et al., 1999). Recent studies show that 90% of users are concerned or very concerned about privacy (Taylor, 2003). Moreover, almost 95% of web users admitted they have declined to provide personal information to web sites at one time or another when asked (Hoffman et al., 1999).

In this paper, we describe the support that the Magentix2[4] AP provides for preserving privacy. The remainder of this paper is organized as follows. Section 2 introduces the main concepts treated in this paper. Section 3 gives a brief overview of the Magentix2 AP. Section 4 presents the support that Magentix2 provides for avoiding information processing. Section 5 presents the support that Magentix2 provides for avoiding information collection. Section 6 presents an application that takes advantage of the support for preserving privacy that Magentix2 provides. Section 7 presents the evaluation we carried out. Section 8 presents related relevant works. Finally, Section 9 presents some concluding remarks and future work.

Access through your organization
Check access to the full text by signing in through your organization.
Access through your organization

Section snippets

Background

In this paper we consider two information-related activities that can represent a major threat for privacy: information collection and information processing (Rannenberg et al., 2009, Such et al., in press). These activities can lead to many privacy breaches (Solove, 2006). We now introduce both activities and outline how these activities can be prevented when they are not desired. We also detail the implications that preventing these activities may have in accountability, trust, and reputation. ...

The Magentix2 agent platform

Magentix2[7] is an AP that focuses on providing support for open MAS. Magentix2 provides support at three levels, as suggested in Luck et al. (2005): (i) organization level: technologies and techniques related to agent societies; (ii) interaction level: technologies and techniques related to communications between agents; and (iii) agent level: technologies and techniques related to individual agents (such as reasoning and learning).

...

Protection against information processing

As we detailed in Section 2, information processing can be minimized by minimizing data identifiability, i.e., using many pseudonyms (even complete anonymity can be achieve by using a different pseudonym for each interaction). However, this introduces problems regarding accountability, trust and reputation. We describe in this section the support that Magentix2 provides for avoiding information processing but preserving accountability and avoiding identity-related vulnerabilities of trust and ...

Protection against information collection

In this section, we detail the support that Magentix2 provides for avoiding information collection. As we explained in Section 2, information collection by unauthorized parties can be avoided by means of confidentiality. Therefore, we explain in this section how Magentix2 makes agent communications confidential. Moreover, we detail how this confidential agent communication integrates with the agent identity management model presented in the previous section.

Agent communication in Magentix2 is ...

Application to an agent-based e-marketplace for wines

In this section, we detail an application built on top of Magentix2. It is an agent-based e-commerce application. Agent-mediated e-commerce (Fasli, 2007b, Sierra, 2004) refers to electronic commerce in which agent technologies are applied to provide personalized, continuously running, semi-autonomous behavior. In agent-mediated e-commerce, agents can be distributed among different remote locations over the Internet. For instance, buyer agents mostly act as personal assistants that usually run ...

Evaluation

In this section, we provide an extensive evaluation of the support that Magentix2 provides for preserving privacy based on the application we just presented in the previous section. In particular, we test whether or not information processing can be minimized by using the support that Magentix2 provides, whether or not this support introduces a bearable performance overhead, and whether or not existing trust and reputation models can be implemented on top of Magentix2. Finally, we also provide ...

Privacy-enhancing Agent Platforms

There are many Agent Platforms (APs) developed by the agent community (Alberola et al., 2010). However, only a few of them currently take security concerns into account. For instance, Jade (JADE Board, 2005), Magentix[12] (Such et al., 2011b), AgentScape (Quillinan et al., 2008), SECMAP (Ugurlu and Erdogan, 2005), Tryllian ADK (Xu and Shatz, 2003), Cougaar (...

Conclusions

In this paper, we present the support that the Magentix2 AP provides for enhancing privacy in the applications built on top of it. This support also allows agents to avoid identity-related vulnerabilities of

trust and reputation models. Moreover, this support avoids the lack of accountability of the principals involved. All these features are crucial for encouraging principals' trust in agent-based applications.

We experimentally show that the support that Magentix2 provides can be used for ...

Acknowledgments

This work has been partially supported by CONSOLIDER-INGENIO 2010 under Grant CSD2007-00022, and project TIN2009-13839-C03-01.

We acknowledge ITMAS 2011 as the forum in which the main ideas behind this paper were preliminary discussed. ...

References (73)

M. Hansen *et al.*
Privacy-enhancing identity management
Inf. Secur. Tech. Rep. (2004)

A. Jøsang *et al.*
A survey of trust and reputation systems for online service provision
Decis. Support Syst. (2007)

J.M. Such *et al.*
Partial identities as a foundation for trust and reputation
Eng. Appl. Artif. Intell. (2011)

Acquisti, A., Gritzalis, S., Lambrinoudakis, C., di Vimercati, S. (Eds.), 2008. Digital Privacy: Theory, Technologies,...

E. Aïmeur *et al.*
Secure anonymous physical delivery
IADIS Int. J. WWW/Internet (2006)

J.M. Alberola *et al.*
A performance evaluation of three multiagent platforms
Artif. Intell. Rev. (2010)

Aydoan, R., Yolum, P., 2010. Learning opponents preferences for effective negotiation: an approach based on concept...

Balke, T., Eymann, T., 2008. The conclusion of contracts by software agents in the eyes of the law. In: Proceedings of...

A. Bhargav-Spantzel *et al.*
User centricity: a taxonomy and open issues
J. Comput. Secur. (2007)

Bilge, L., Strufe, T., Balzarotti, D., Kirda, E., 2009. All your contacts are belong to us: automated identity theft...

View more references

Cited by (22)

A survey of privacy in multi-agent systems ↗
2014, Knowledge Engineering Review

Logic-based technologies for multi-agent systems: a systematic literature review ↗
2021, Autonomous Agents and Multi Agent Systems

Cooperative multi-Agent planning: A survey ↗
2018, ACM Computing Surveys

Survey on computational trust and reputation models ↗
2019, ACM Computing Surveys

Applying an Unified Access Control for IoT-based Intelligent Agent Systems ↗
2016, Proceedings 2015 IEEE 8th International Conference on Service Oriented Computing and Applications Soca 2015

FMAP: Distributed cooperative multi-agent planning ↗
2014, Applied Intelligence

View all citing articles on Scopus ↗

View full text

Copyright © 2012 Elsevier Ltd. All rights reserved.

About ScienceDirect ↗ Remote access Contact and support ↗ Terms and conditions ↗ Privacy policy ↗ Cookie settings

All content on this site: Copyright © 2026 Elsevier B.V., its licensors, and contributors. All rights are reserved, including those for text and data mining, AI training, and similar technologies. For all open access content, the relevant licensing terms apply.

Magentix2: a Privacy-enhancing Agent Platform

Jose M. Such, Ana García-Fornes, Agustín Espinosa, Joan Bellver

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València, Camí de Vera s/n, València, Spain
{*jsuch,agarcia,aespinos,jbellver*}*@dsic.upv.es*

Abstract

Agent Platforms are the software that supports the development and execution of Multi-agent Systems. There are many Agent Platforms developed by the agent community, but they hardly consider privacy. This leads to agent-based applications that invade users' privacy. Privacy can be threatened by two main information activities: information collection and information processing. Information collection can be prevented using traditional security mechanisms. Information processing can be prevented by minimizing data identifiability, i.e., the degree by which personal information can be directly attributed to a particular individual. However, minimizing data identifiability may directly affect other crucial issues in Multi-agent Systems, such as accountability, trust, and reputation. In this paper, we present the support that the Magentix2 Agent Platform provides for preserving privacy. Specifically, It provides mechanisms to avoid information collection and information processing when they are not desired. Moreover, Magentix2 provides these mechanisms without compromising accountability, trust, and reputation. We also provide in this paper an application built on top of Magentix2 that exploits its support for preserving privacy. Finally, we provide an extensive evaluation of the support that Magentix2 provides for preserving privacy based on that application. We specifically test whether or not privacy loss can be minimized by using the support that Magentix2 provides, whether or not this support introduces a bearable performance overhead, and whether or not existing trust and reputation models can be implemented on top of Magentix2.

Keywords: Privacy, Agent Platforms, Multi-agent Systems, Security, Trust, Reputation

1. Introduction

A Multiagent System (MAS) consists of a number of agents that interact with one-another [1]. MAS represents a key issue, especially from the development point of view in Distributed Artificial Intelligence (DAI). This is because the MAS community has produced both methodologies and actual frameworks to make the implementation of agent-based applications possible. In particular, Agent Platforms (APs) are the software that supports the development and execution of MAS. APs provide all the basic infrastructure (for message handling, tracing and monitoring, run-time management, and so on) required to create MAS [1].

There are many APs developed by the MAS community – for an overview of current APs and the features they provide refer to [2]. However, privacy is seldom considered [3, 4]. This leads to agent-based applications that invade individuals' privacy. This is due to the fact that an agent usually encapsulates personal information describing its principal[1] [5], such as preferences, names, and other informa-

[☆]NOTICE: this is the author's version of a work that was accepted for publication in Engineering Applications of Artificial Intelligence. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. A definitive version was subsequently published: Jose M. Such, Ana Garcia-Fornes, Agustin Espinosa and Joan Bellver. Magentix2: a Privacy-enhancing Agent Platform. Engineering Applications of Artificial Intelligence, Vol. 26 N. 1 pp. 96-109 (2013). `http://www.sciencedirect.com/science/article/pii/S0952197612001522`

[1] In this paper, we use the terms principal and user indistinctly to refer to the user that the agent is acting on behalf

tion. Moreover, agents carry out interactions on behalf of their principals so that they exchange personal information. For instance, agents act on behalf of their principals in agent-mediated e-commerce [6], as personal assistants [7], in virtual worlds like Second Life[2] [8], as recommenders [9], and so on.

The modern conception of privacy started more than a hundred years ago, with the seminal work of Warren & Brandeis [10] *The right of privacy*. These two lawyers defined privacy as "the right to be let alone". They were pioneers in considering the implications of technology in privacy. Specifically, they were very concerned about the implications of instantaneous photographs and portraits in injuring the feelings of the people in those photographs and portraits. Privacy was later recognized as a fundamental human right by the United Nations Declaration of Human Rights, the International Covenant on Civil and Political Rights, the Charter of Fundamental Rights of the European Union, and many other international treaties [11].

In the second part of the twentieth century, Alan Westin defined privacy as "the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated" [12]. This is what is currently known as the informational self-determination right [13]. The concept of informational self-determination changed the right to privacy from the right to be let alone to its current incarnation as a means to limit the abuse of personal data [14]. Informational self-determination represents today's European understanding and regulation of privacy in the context of information and communication technology (EU Directives 95/46/EC, 45/2001/EC, and 2002/58/EC).

Despite all these regulations, as the Internet has no governing or regulating body, privacy breaches are still possible. Nowadays, in the era of global connectivity (everything is inter-connected anytime and everywhere) with more than 2 billion world-wide users with connection to the Internet as of 2011[3], privacy is of great concern. In the real world, everyone decides (at least implicitly) what to tell other people about themselves. In the digital world, users have more or less lost effective control over their personal data. Users are therefore exposed to constant personal data collection and processing without even being aware of it [15]. Garfinkel [16] suggests that nowadays users have only one option to preserve their privacy: becoming hermits and not using online social networks, e-commerce sites, etc. Considering the increasing power and sophistication of computer applications that offer many advantages to individuals, becoming a hermit may not really be an option. However, all of these advantages come at a significant loss of privacy [17]. Recent studies show that 90% of users are concerned or very concerned about privacy [18]. Moreover, almost 95% of web users admitted they have declined to provide personal information to web sites at one time or another when asked [19].

In this paper, we describe the support that the Magentix2[4] AP provides for preserving privacy. The remainder of this paper is organized as follows. Section 2 introduces the main concepts treated in this article. Section 3 gives a brief overview of the Magentix2 AP. Section 4 presents the support that Magentix2 provides for avoiding information processing. Section 5 presents the support that Magentix2 provides for avoiding information collection. Section 6 presents an application that takes advantage of the support for preserving privacy that Magentix2 provides. Section 7 presents the evaluation we carried out. Section 8 presents related relevant works. Finally, Section 9 presents some concluding remarks and future work.

2. Background

In this paper we consider two information-related activities that can represent a major threat for privacy: information collection and information processing [13, 4]. These activities can lead to many privacy breaches [20]. We now introduce both activities and outline how these activities can be prevented when they are not desired. We also detail the implications that preventing these activities may have in accountability, trust, and reputation.

2.1. Information Collection

Information collection refers to the process of gathering and storing data about an individual. Personal data is transferred on-line even across the Internet.

of. Principals are also called agent owners, or simply users in the related literature.

[2] `http://secondlife.com/`

[3] `http://www.internetworldstats.com/stats.htm` to consult updated statistics on world Internet users and population.

[4] `http://magentix2.gti-ia.upv.es`

Without appropriate protection mechanisms a potential attacker could easily obtain information about principals without their consent. For instance, an attacker can be listening to transferred information over the network (files, messages, e-mails, etc) and simply gather the information flowing in the network [21]. Moreover, the attacker could even use the information it gathers about an individual to impersonate her/his, which is known as *identity theft* [22]. For instance, in [23] the authors present how to clone an existing account in an online social network and to establish a friendship connection with the victim in order to obtain information about her/him.

In order to avoid undesired information collection, sensitive personal information must be protected from access by any other third party that is different from the agent to which the information is directed to. Therefore, avoiding information collection requires security to control the access to personal information [24]. In particular, confidentiality is a security property of a system that ensures the prevention of unauthorized reading of information [25]. In distributed environments, confidentiality usually means that sensitive information is encrypted into a piece of data so that only parties that can decrypt that piece of data can access the sensitive information.

Confidentiality can be achieved by using existing secure data transfer technologies such as Kerberos [26], SSL [27], and TLS [28]. These technologies allow the encryption of messages before transferring them and the decryption of messages once they are received. As a result, if an agent A sends a message to an agent B using these technologies, A is sure that B will be the only one able to read this message.

Confidentiality is a necessary condition to preserve privacy, but it is not sufficient. It prevents undesired information collection from unauthorized third parties. If an agent A sends personal information to an agent B in a confidential fashion, external third parties will not be able to access it. However, agent B will obviously receive this personal information. The point is that agent B can then process the received personal information, unless specific measures for preventing information processing are adopted before sending this information.

2.2. Information Processing

Information processing refers to the use or transformation of data that has already been collected [29], even though this information has been collected by mutual consent between two parties. An example of information processing is profiling [30]: "*the process of 'discovering' patterns in data that can be used to identify or represent a human or nonhuman subject (individual or group) and/or the application of profiles (sets of correlated data) to individuate and represent an individual subject or to identify a subject as a member of a group (which can be an existing community or a discovered category) and/or the application of profiles to individuate and represent individuals or groups*".

One of the most common types of profiling is called buyer profiling in e-commerce environments, in which vendors obtain detailed profiles of their customers and tailor their offers regarding customers' tastes. These profiles can represent a serious threat to privacy. For instance, these profiles can be used to perform *price discrimination* [31]. Vendors could charge customers different prices for the same good according to the customers' profiles, i.e., if a vendor knows that some good is of great interest to one customer, the vendor could charge this customer more money for this good than other customers for the same good. For instance, in 2000, Amazon started to charge customers different prices for the same DVD titles [32]. When the story became public, Amazon claimed that this was part of a simple price test and discontinued this practice. Another example of privacy threat due to the use of these profiles is what is known as *poor judgment* [33]. This is when individuals are judged and subsequently treated according to decisions made automatically based on incorrect or partial personal data. For instance, companies usually divide their potential customers into similar groups based on customers' characteristics (known as customer segmentation). This practice can lead to exclusion of people from services based on potentially distorted judgments [29].

Most of the work on protecting against the processing of information already collected is based on minimizing data identifiability. Identifiability can be defined as "the degree to which (personal) data can be directly linked to an individual" [29]. The degree of privacy of a system is inversely related to the degree of user data identifiability. The more identifiable data that exists about a person, the less she/he is able to control access to information about herself/himself, and the greater the privacy risks. Identifiability ranges from complete identifiability to anonymity.

Pseudonymity [34] is the use of pseudonyms as

identifiers. A pseudonym is an identifier of a subject other than one of the subject's real names. Pseudonyms have been broadly used by human beings in the real world. For instance, in the 19th century when writing was a male-dominated profession, some female writers used male names for their writings. Nowadays, in the digital world, there is a great number of pseudonyms such as usernames, nicknames, e-mail addresses, sequence numbers, public keys, etc. [35].

The most important trait of pseudonymity is that it comprises all degrees of identifiability of a subject (from identified to anonymous) depending on the nature of the pseudonyms being used. Complete identification is when the linking between a pseudonym and its holder is publicly known. Anonymity can be achieved by using a different pseudonym for each different interaction. This is known as transaction pseudonyms [34]. For instance, let us suppose that two agents A and B act as a buyer and a seller, respectively, in a e-marketplace. Agent A can use a different pseudonym (e.g. a random generated numeric identifier) for each specific interaction with agent B. Hence, Agent B collects information about the interactions performed but is unable to relate different interactions to each other or relate any of these interactions to agent A[5].

As we detail later on in this paper (Section 4), Magentix2 allows agents to use as many pseudonyms as they need to prevent information processing. We refer to these pseudonyms as *regular* pseudonyms.

2.3. Implications in Accountability

Minimizing data identifiability may have a direct impact on accountability. Accountability refers to the ability to hold entities responsible for their actions [38]. Accountability usually requires an unambiguous identification of the principal involved [39]. Then, this principal can be held liable for their acts. For instance, a buyer agent pays a seller agent for a good. The seller commits to shipping the good to the customer agent's principal. In the event that the customer agent's principal does not receive the good, the seller agent's principal[6] may be held liable for this. Although determining exactly who should be held liable for this depends on the applicable laws in the specific country, it usually requires the identification of the seller agent's principal. Then, the seller agent's principal can be sued for fraud.

Many systems (such as commercial systems) emphasize accountability because, in these environments, principals can be subject to serious losses such as money loss. Moreover, the sense of impunity generated by the lack of accountability could even encourage abuse. Thus, accountability is of crucial importance for agent-based applications because it helps to promote trust in these applications, which is needed for principals to be willing to engage with and delegate tasks to agents [42].

Pseudonyms can be utilized to implement accountability [43]. However, this implies that there is a trusted entity that issues pseudonyms and knows the association between pseudonyms and the real identity of the principal that is behind an agent. Therefore, when an agent misbehaves using a given pseudonym, this pseudonym can be traced back to the principal behind this agent for law enforcement. One approach to do this would be that the AP is the one that issues pseudonyms. Therefore, the AP itself can disclose the principal behind the pseudonym, removing pseudonymity and producing identity and accountability as a result. The main drawback of this approach is that the AP itself knows the relation of pseudonyms to each other and to the principal involved. As APs are usually to be run by the same company that hosts the specific system (e.g. eBay), information processing could still be easy. We think that decoupling the place in which the identities are issued and the place in which the identities are effectively used can make this harder. Therefore, we

[5] Even when buyer agents are able to change its pseudonym, purchases may include information that can be used to relate different purchases to each other and to the buyer agent, e.g., the credit card number to perform the payments and the shipment address may be the same for different transactions. We assume that payments are carried out using some kind of anonymous payment mechanism and deliveries are carried out using some anonymous delivery system. Hence, credit card numbers and delivery addresses do not need to be disclosed when an agent acquires a product. For instance, the untraceable electronic cash presented by Chaum et al. [36] can be used for anonymous payments. For anonymous deliveries, the privacy-preserving physical delivery system presented by Aïmeur et al. [37] can be used.

[6] Software entities (intelligent agents, virtual organizations, etc.) cannot have real identities because, until now, they could not be held liable for their acts in front of the law. However, this may change in the future if they finally achieve some kind of legal personhood, as suggested by [40] and [41]. In this case, software entities may be provided with legal personhood to be (partially) held liable for their acts. The point is that according to the law, someone must be liable for frauds like this.

consider that pseudonyms are issued and validated by external trusted third parties (as explained later on in Section 4) that do not participate in any way in the specific system.

2.4. Implications in Trust and Reputation

There is also the need to equip agents with models to reason about and assess trust towards other agents and their reputation in a MAS [5]. These models allow agents to select the best and most reliable partners in a specific situation and to avoid partners of previous unsuccessful interactions. Trust and reputation are even more important in open systems, in which previously unknown parties may interact. For instance, if a buyer agent enters into an e-marketplace for the first time, it will need to choose among all of the available seller agents. As the buyer agent has no previous interactions with the seller agent, the reputation of the seller agent in the e-marketplace can play a crucial role for the buyer agent to choose an specific seller agent.

The agent community has developed a vast number of trust and reputation models [44, 45]. However, current trust and reputation models are based on the assumption that identities are long-lived, so that ratings about a particular agent from the past are related to the same agent in the future. However, when such models are actually used in real domains this assumption is no longer valid and the so-called identity-related vulnerabilities emerge. These vulnerabilities can be exploited by means of *whitewashing* attacks [46] (also called *change of identities* [47]). For instance, an agent that has a low reputation due to its cheating behavior may be really interested in changing its identity and restarting its reputation from scratch. These vulnerabilities can also be exploited by means of *sybil* attacks [48]. An example of this attack could be an agent that holds multiple identities in a marketplace and attempts to sell the same product through each of them, increasing the probability of being chosen by a potential buyer.

Due to the identity-related vulnerabilities of trust and reputation models, malicious agents may be able to modify the expected behavior of trust and reputation models. As a result, these reputation models may become completely useless. For instance, in our previous example, a seller agent may be able to cheat the reputation model used by the buyer agent. Thus, the buyer agent may end up interacting with a malicious agent instead of what it believes a reputable agent. These has the potential to cause many damages such as money lose. Therefore, these vulnerabilities have the potential to place the whole system in jeopardy.

A possible solution for these vulnerabilities is the use of *once-in-a-lifetime* pseudonyms [49]. Agents can only hold one *once-in-a-lifetime* pseudonym in each marketplace. Therefore, they cannot get rid of the trust and reputation ratings they got from other agents in the marketplace. However, minimizing data identifiability requires that agents have the ability to hold and use many pseudonyms. Thus, it could seem that a system that allows the prevention of information processing may have important difficulties to also prevent identity-related vulnerabilities.

To address this, Magentix2 bases on the agent identity model proposed in [50], as we explain later on in Section 4. Agents in Magentix2 can hold two kinds of pseudonyms: permanent pseudonyms (that correspond to *once-in-a-lifetime* pseudonyms), which avoid identity-related vulnerabilities; and regular pseudonyms which agents can use without any limitation in number to obtain their desired degree of privacy.

3. The Magentix2 Agent Platform

Magentix2[7] is an AP that focuses on providing support for open MAS. Magentix2 provides support at three levels, as suggested in [51]: (i) organization level: technologies and techniques related to agent societies; (ii) interaction level: technologies and techniques related to communications between agents; and (iii) agent level: technologies and techniques related to individual agents (such as reasoning and learning).

In this paper we focus on the technologies that Magentix2 provides at interaction level to enhance the privacy of the applications that are developed and executed on top of it. Firstly, we briefly introduce the Magentix2 agent communication mechanism in order for the reader to have a bird's-eye view of the system. Then, we thoroughly describe the support that Magentix2 provides to avoid information processing and information collection in Sections 4 and 5 respectively.

[7] `http://users.dsic.upv.es/grupos/ia/sma/tools/magentix2/index.php`

3.1. Agent Communication

Magentix2 uses AMQP[8] [52] as a foundation for agent communication. This standard facilitates the interoperability between heterogeneous entities. Magentix2 allows heterogeneous agents to interact with each other via messages that are represented following the FIPA-ACL [53] standard, which are exchanged using the AMQP standard.

Magentix2 uses the Apache Qpid[9] open-source implementation of AMQP for Agent Communication. Apache Qpid provides two AMQP servers, implemented in C++ (the one we use) and Java. Qpid also provides AMQP Client APIs that support the following languages: C++, Java, C# .NET, Ruby, and Python. Qpid allows distributed applications made up of different parts written in any of these languages to communicate with each other. What is more, any client that is developed using one of the Qpid Client APIs is able to communicate with any client that is developed using any other AMQP-compliant API via any AMQP server implementation, as long as both server and clients implement the same version of the AMQP standard.

Figure 1 shows an overview of the Magentix2 agent communication architecture. Magentix2 is composed by one or more (in this case federated) AMQP Servers (QPid brokers). Magentix2 agents act as AMQP Clients (using Qpid Client APIs) that connect to the Qpid broker and are then able to communicate with each other. Magentix2 agents can be located in any Internet location, they only need to know the host on which the Qpid broker (or one of the federated Qpid brokers) is running.

Magentix2 provides a Java library, which is called the Magentix2 Agent Library (MAL), to facilitate the development of agents. This API allows agent programmers to focus on creating FIPA-ACL messages and sending and receiving them, without dealing directly with the Qpid Client Java API. Currently, this API is only written in Java, but the existence of multiple QPid Client APIs for several programming languages enables the development of agents written in different programming languages. Moreover, any proprietary implementation that follows both AMQP and FIPA-ACL standards would be interoperable with Magentix2 agents.

[8] http://www.amqp.org/
[9] http://qpid.apache.org/

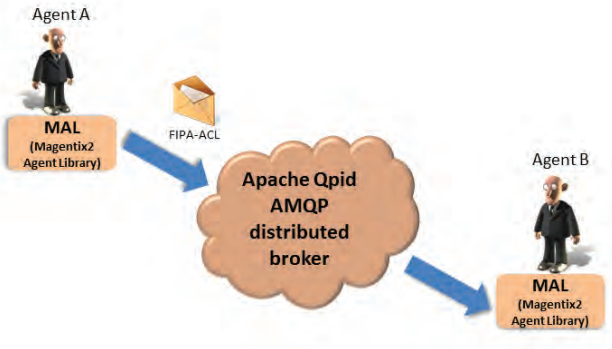


Figure 1: Magentix2 Agent Comunication Architecture

4. Protection against Information Processing

As we detailed in section 2, information processing can be minimized by minimizing data identifiability, i.e., using many pseudonyms (even complete anonymity can be achieve by using a different pseudonym for each interaction). However, this introduces problems regarding accountability, trust and reputation. We describe in this section the support that Magentix2 provides for avoiding information processing but preserving accountability and avoiding identity-related vulnerabilities of trust and reputation models.

The building block of the Magentix2 support for avoiding information processing is how it manages the identities of the agents. In particular,the agent identity management in Magentix2 is based on the partial identities model presented in [50]. In a nutshell and informally speaking, a partial identity can be seen as a set of attributes that identifies an entity in a given context. They are composed of a pseudonym that is unique within a context and other attributes that describe the entity within that context (roles, location, preferences, etc.).

Magentix2 considers two kinds of partial identities:

- **Regular Partial Identities (RPIs)** A RPI must contain a regular pseudonym. Agents have no limitation in the number of regular pseudonyms that they are able to hold in a given system. Thus, agents can hold as many as RPIs as they need to achieve they desired data identifiability level. For instance, an agent could use a different RPI in each interaction to achieve anonymity.
- **Permanent Partial Identities (PPIs)** must contain a permanent pseudonym (*once-in-a-lifetime*

pseudonym). An agent can only hold one permanent pseudonym for a given system. Thus, if trust and reputation is established trough PPIs, identity-related vulnerabilities of trust and reputation are avoided. This is because an agent can only hold one PPI in a given system.

Although both kinds of partial identities enable trust and reputation relationships, only PPIs guarantee that identity-related vulnerabilities are avoided. Therefore, agents will choose to establish trust and reputation through PPIs if they want to avoid identity-related vulnerabilities. However, if they want to avoid information processing, they can use as many RPIs as needed. For instance, in an e-marketplace a seller agent could be very interested in interacting with buyer agents using a PPI. This is because buyer agents would know that the seller agent is not able to perform attacks that exploit identity-vulnerabilities, e.g., the seller agent is not able to change its PPI. However, a buyer agent can be very interested in using a different RPI each time it interacts with a seller agent, so that the seller agent is not able to make a profile of the buyer agent's tastes and then use this profile to perform practices such as price discrimination, poor judgement, etc.

Magentix2 also considers the concept of real identities. Real identities identify entities that can be liable for their acts in front of the law, such as human beings, companies, etc. Real identities are used for accountability concerns such as law enforcement. For this reason, real identities are restricted to only legal persons. A real identity, for example, would be: *Bob Andrew Miller, born in Los Angeles, CA, USA on July 7, 1975.* Agents might also have a real identity for accountability concerns if they finally achieve some kind of legal personality, as suggested by [40] and [41]. In the event of this, the support that Magentix2 provides would not need any modification.

4.1. Identity Management Architecture

Magentix2 complies with the client part of the Identity Metasystem Interoperability standard[10]. This standard specifies the interfaces for the secure web services provided by User-Centric Privacy-Enhancing Identity Management Systems [54]. These systems support the process of management of partial identities. In particular, they provide the following facilities:

- **Identity Providers (IdPs)**, which issue partial identities and validate these identities to other Relying Parties.
- **Relying Parties**, which are a set of APIs for verifying partial identities against an Identity Provider.
- **Identity Selectors**, which provide a simple way to manage partial identities and choose which partial identity to use in a given context.
- **Attribute Services**, which allow the specification of access control rights of relying parties over the attributes in a partial identity.

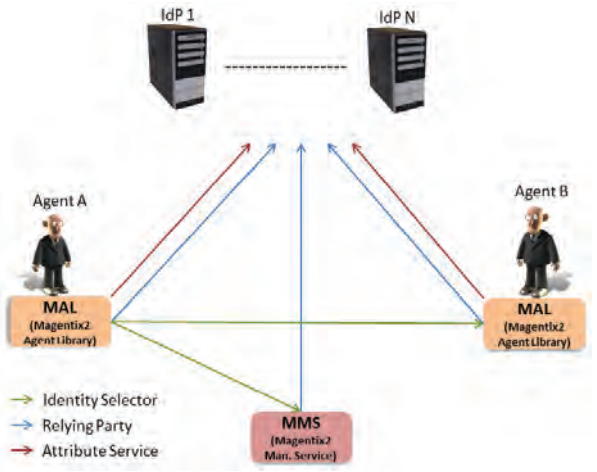


Figure 2: The Magentix2 agent identity management architecture.

Figure 2 shows an overview of the Magentix2 agent identity management support. As one can observe, Magentix2 components act as clients of IdPs. The Magentix2 Agent Library (MAL) implements clients for Identity Selectors, Relying Parties, and Attribute Services. Therefore, agents in Magentix2 can select the partial identity to use in a given interaction, verify the partial identities of other agents, and specify access control for attributes in their partial identities.

We can also see in Figure 2 that there is another component of Magentix2 that is called the Magentix2 Management Service (MMS). The MMS is a secure web service that acts as a Relying Party, i.e., it is able to request IdPs to verify partial identities. The MMS

[10] `http://docs.oasis-open.org/imi/identity/v1.0/identity.html`

is in charge of dynamically signing digital certificates for agents to communicate to each other in a confidential fashion, and thus, avoiding information collection from unauthorized parties (as detailed in section 5). Agents request the signing of digital certificates to the MMS using one of their partial identities. The MMS must verify the partial identity that the agent used before signing the digital certificate.

IdPs are classified according to the type of partial identities they issue. The Permanent Identity Provider (PIdP) is an IdP (or a federation of IdPs[11]) that issues PPIs to the agents taking part in the specific marketplace. Agents must register using a real identity that the PIdP will not reveal to other agents or to Magentix2. The PIdP is also in charge of forcing agents to only hold a single PPI in this specific marketplace. Moreover, an agent is able to know when it validates a partial identity of another agent that this partial identity is effectively a PPI so that the second agent cannot change it and is unable to perform identity-related attacks.

Regular Identity Providers (RIdPs) issue RPIs to agents. Agents request RPIs by providing either a real identity, or a PPI that RIdPs will not reveal to others. There is no limitation in the number of RIdPs per marketplace or in the number of RPIs per agent and per marketplace.

4.2. Obtaining Partial Identities

We now detail the dynamics of agent identity management in Magentix2. Users provide their real identity when they launch an agent. This real identity is only used by the local instance of the MAL. Therefore, it is only in the local computer of the user that launches the agent. Moreover, the API offered by the MAL allows the user to choose an identity for the newly created agent. Then, the MAL calls to a PIdP/RIdP to obtain a PPI/RPI. From then on, and as we will explain in section 5, the agent can use this PPI/RPI to interact with other agents.

The important point is that neither Magentix2 nor the agents running in Magentix2 have access to the real identity of the user. They are only able to know the PPI/RPI that the agent is using. In case of law enforcement, a court could ask IdPs to disclose the real identity behind a particular PPI/RPI. Thus, accountability is preserved.

Whenever an agent wants to change its RPI (recall that PPIs cannot be changed), it only needs to call to the MAL API. Then, this API will be in charge of contacting the appropriate RIdP and obtaining a new RPI.

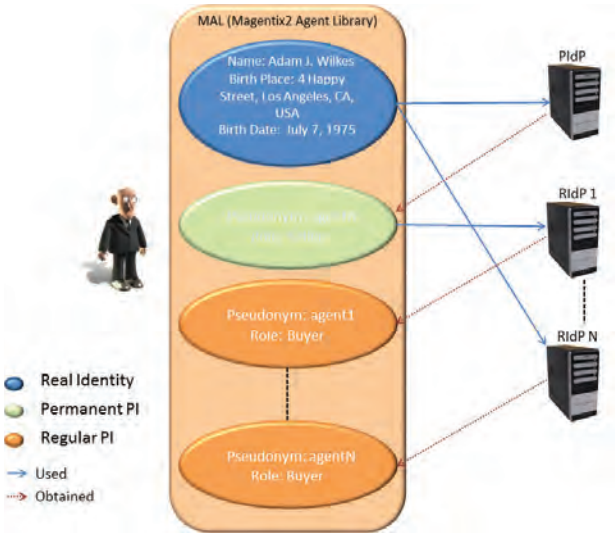


Figure 3: An example of the Partial Identities of an agent.

Figure 3 shows an example of an agent and its partial identities. The agent's principal has a real identity with an attribute name *Adam John Wilkes*. Using this real identity, the agent has obtained a PPI from the PIdP that includes two attributes: pseudonym `A` and role `seller`. This entity has also obtained N RPIs from N different RIdPs. Some of the RPIs are obtained by providing a PPI (such as RPI 1, with pseudonym `agent1` and role `buyer`) and other RPIs are obtained using a real identity (such as RPI N, with pseudonym `agentN` and role `buyer`).

5. Protection against Information Collection

In this section, we detail the support that Magentix2 provides for avoiding information collection. As we explained in Section 2, information collection by

[11] User-Centric Identity Management Systems support the federation of IdPs that belong to the same and also different remote security domains across the Internet. Therefore, a PIdP can be implemented as a federation of IdPs instead of only one IdP, minimizing the typical drawbacks of a centralized trusted third party, such as being a single point of failure (SPOF) and a possible efficiency bottleneck. Examples of identity federation standards are the Liberty Alliance Identity Federation Framework `http://projectliberty.org/resource_center/specifications/liberty_alliance_id_ff_1_2_specifications/` and WS-Federation `http://www.ibm.com/developerworks/library/specification/ws-fed/`.

unauthorized parties can be avoided by means of confidentiality. Therefore, we explain in this section how Magentix2 makes agent communications confidential. Moreover, we detail how this confidential agent communication integrates with the agent identity management model presented in the previous section.

Agent communication in Magentix2 is based on AMQP. The AMQP standard specifies secure communication by tunneling AMQP connections through SSL [27] (so-called AMQPS). Apache Qpid implements SSL support for AMQP. SSL authenticates communicating parties based on digital certificates. Thus, it needs a configured Public Key Infrastructure (PKI). The Magentix2 PKI is set during installation time. Firstly, the Magentix2 certificate authority (MCA) is created. Secondly, certificates for the Magentix2 Management Service (MMS) and the Qpid Broker are created using this certificate authority. Digital certificates for agents are created automatically by the MAL and dynamically signed by the MCA through the MMS at execution time (as described below).

The MMS is a front-end of the MCA. It is implemented as a secure web service. The MMS is in charge of dynamically signing digital certificates for agents, which can use these certificates to communicate securely. The MMS service needs two inputs: the agent pseudonym and a non-signed digital certificate. The first input is the pseudonym in the permanent or regular partial identity (issued by a permanent or regular IdP) that the agent uses to invoke the MMS. The second input is a non-signed certificate that contains the agent's public key (this is the certificate that is to be signed). The agent key pair (private and public key) and this certificate are created by the MAL *locally* for each agent and for each new partial identity.

The MMS produces one output: the digital certificate *signed* by the MCA. The MMS produces this output after: (i) verifying that the pseudonym is the same as the one in the partial identity used to invoke the secure web service; (ii) verifying the partial identity against the IdP that issued it; (iii) and finally signing the certificate using the MCA. Agents can then use this signed certificate to communicate to other Magentix2 agents.

The AMQP connection of every agent to the Qpid broker is tunneled through SSL. Hence, the communication between two Magentix2 agents is provided with confidentiality and integrity out of the box. To ensure the authenticity of the sender pseudonym in a FIPA-ACL message (recall that in Magentix2 FIPA-ACL messages are encapsulated into AMQP messages), an agent must verify that the pseudonym of the sender in the AMQP sender message field is the same as the pseudonym of the sender in the FIPA-ACL sender message field upon receiving a new message. This is performed automatically by the Magentix2 agent library.

Each time an agent obtains a new partial identity (RPI or PPI) as described in Section 4, the MAL repeats the process for obtaining a MCA-signed certificate. This certificate is for the agent to communicate in a confidential fashion when it uses this partial identity. Moreover, agents can also choose the partial identity to be used in each communication. Therefore, the MAL is in charge of using each time the certificate that corresponds to the partial identity that the agent intends to use.

Figure 4 shows an example of an agent that holds a RPI with two attributes: the pseudonym A, and the role buyer. The MAL automatically creates a certificate that contains the pseudonym A and calls to the MMS using the agent's RPI. The MMS validates the RPI to the IdP that issued it. Then, it signs the certificate using the MCA and sends back the signed certificate to the A agent. From this moment on, when the A agent sends a message, the MAL automatically send this message trough an AMQPS connection established with the signed certificate received from the MMS. Thus, agent A can communicate with agent B in a confidential fashion.

6. Application to an Agent-based E-marketplace for Wines

In this section, we detail an application built on top of Magentix2. It is an agent-based e-commerce application. Agent-mediated e-commerce [42, 6] refers to electronic commerce in which agent technologies are applied to provide personalized, continuously running, semi-autonomous behavior. In agent-mediated e-commerce, agents can be distributed among different remote locations over the Internet. For instance, buyer agents mostly act as personal assistants that usually run on a local mobile/personal device/computer and interact with other seller agents that usually run in remote servers.

Agents partially or fully automate some of the processes involved in e-commerce, providing one or more of the following (and also other) facilities [42]:

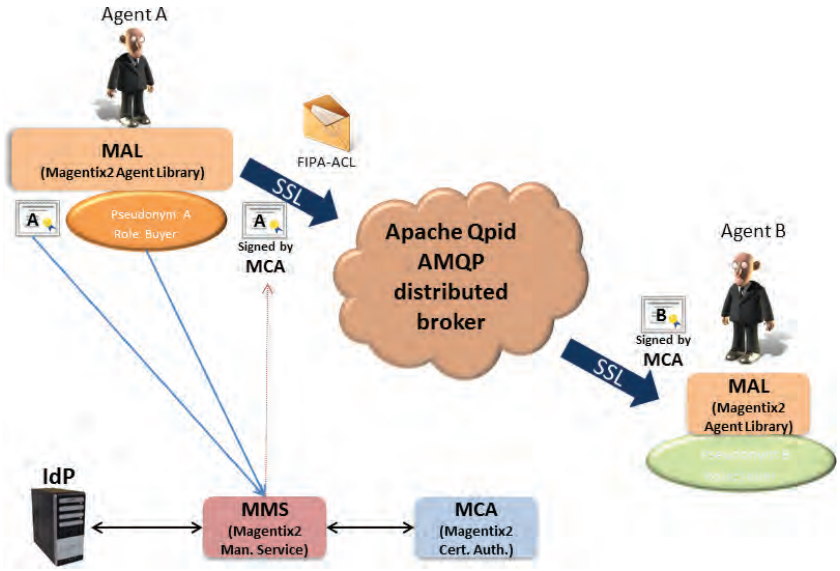


Figure 4: Secure Agent Communication (information collection avoidance).

- find, recommend and compare products, vendors, or services;
- participate in electronic markets and negotiate the price/terms of transactions or contracts with other participants;
- perform transactions on behalf of their users;
- track the user's interests and offer personalized services;
- monitor conditions and provide notifications;
- retrieve, filter, and mine information and knowledge;
- produce and deliver e-services, such as information gathering, processing and management.

In agent-based e-commerce, agents encapsulate personal information describing their principals [42]. They usually have a detailed profile of their principals' names, preferences, roles in organizations and institutions, location, transactions performed, and other personal information. Moreover, agents carry out interactions on behalf of their principals, so they exchange this personal information. Therefore, privacy is of crucial importance in this kind of applications.

In this paper, we consider an electronic market where seller agents and buyer agents trade wines on behalf of their users. Seller agents act on behalf of wine merchants. Buyer agents act on behalf of the users that are willing to buy wines. As shown in Figure 5, each buyer agent can interact with any of the seller agents that is available to buy wines. Moreover, each agent is supposed to be running in a different Internet location.

Agents in the e-marketplace follow the negotiation protocol depicted in Figure 6 for each purchase of a bottle of wine. A buyer agent makes a request to buy a bottle of wine with a *request* message. This message can be replied by the seller agent with either a *model* message (which means that the requested wine is available and includes its price) or an *alternative* message (which means that the requested wine is not available but there is another one that is very similar). Then, the buyer agent can reply to both messages with: an *accept* message (which means that the buyer agent accepts the wine offered), a *quit* message (which means that the negotiation was broken by the buyer agent), or a *request* message (which means that the agent request a different bottle of wine).

We based on the wine attributes considered in the preference modeling approach described in [55]. Thus, we consider the following attributes to describe wines: color, body, flavor, sugar, and country. The possible values for each of these attributes are shown in Table 1.

6.1. Privacy

Privacy can be a great concern in this application. This is because agents are subject to informa-

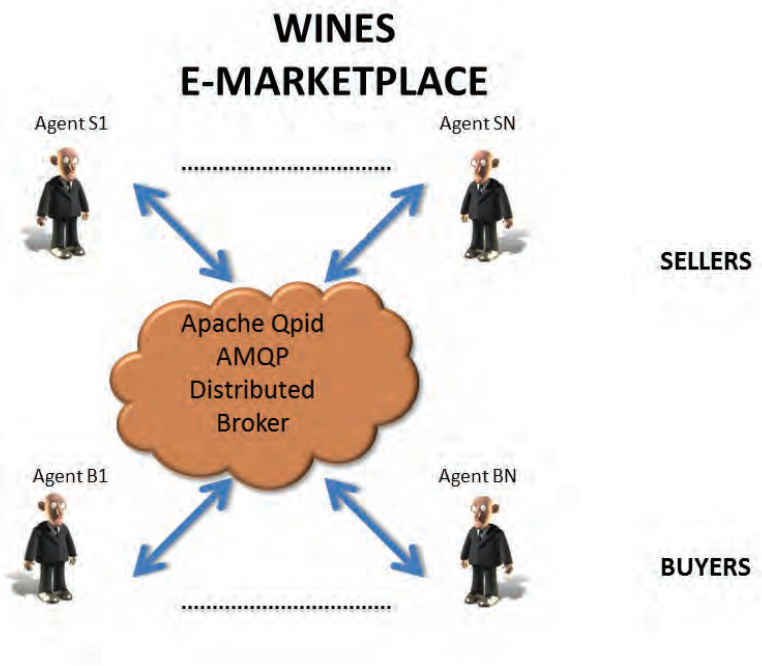


Figure 5: Wines e-Marketplace

Attribute	Values
Color	red, rose, white
Body	light, medium, full
Flavor	delicate, moderate, strong
Sugar	dry, offDry, sweet
Country	France, Portugal, Spain, Italy, USA, Germany, Australia, NewZeeland

Table 1: Considered Wine Attributes.



Figure 6: Negotiation Protocol for the Wine e-marketplace scenario.

tion collection and processing. Specifically, agents can exchange personal information when purchasing bottles of wine. This information must be protected from unauthorized parties, i.e., the information that a seller agent and a buyer agent exchange in a purchase should not be accessible for other agents in the e-marketplace.

Regarding information processing, seller agents could perform buyer profiling, i.e., seller agents obtain detailed profiles of buyer agents and tailor their offers regarding buyer agents's tastes. Seller agents could even charge buyer agents different prices for the same wine according to the customers' profiles (price discrimination), i.e., if a seller agent knows that some wine is of great interest to one buyer agent, the seller agent could charge this customer more money for a bottle of wine than other buyer agents for the very same bottle of wine.

We assume that seller agents follow an approach to build buyer agents' preference profiles similar to [56]. Over the course of a negotiation, the seller agent marks wines that are not accepted by a buyer agent as a negative instance (class "-"), while a seller agent marks the wine that the buyer agent accepts to buy (i.e., the wine of the last step in the nego-

tiation protocol) as a positive instance (class "+"). The seller agent use all the collected instances about a buyer agent to train a statistical classifier. The resulting trained classifier models the buyer agent's preferences with a given accuracy. Thus, the seller agent has a detailed profile on what wines the buyer agent likes/dislikes. Moreover, the seller agent can use this profile in future purchases from the buyer agent, e.g., the seller agent could then perform price discrimination to a buyer agent with respect to an item that the seller agent knows that is of great interest for the buyer agent.

6.2. Trust and Reputation

Trust and reputation also play a crucial role in this application. Buyer agents must be able to choose among seller agents that sell the same wines. One of the important dimensions that a buyer will take into account in its decision is the trust it has in each seller agent. This trust can be based on successful previous purchases with the same seller agent. A buyer agent can trust in a seller agent regarding past interactions by measuring: whether or not the seller agent provisioned the wine on time, the overall quality of the wine bought, if there were hidden costs, etc. A buyer agent can also trust in a seller agent regarding some attributes of the seller agent's partial identity in the electronic market: registration date, corporate title, skills, etc.

Another important dimension that a buyer agent will take into account in its decision of buying a bottle of wine is the reputation of the seller agent. This is particularly useful when the buyer agent had no previous transactions with a given seller agent. In this case, it is not what an agent thinks of a given seller agent but what it is generally said about the seller agent in the e-marketplace.

In this scenario, malicious seller agents could take advantage of the identity-related vulnerabilities of trust and reputation models. Thus, seller agents should not be able to get rid of their trust and reputation assessments. This could cause important money loss. For instance, a seller agent could ship bottles of wine not on time. This obviously decreases the trust and reputation that buyer agents have in this seller agent. Hence, this seller agent decides to quit the electronic market and re-entry into it with a new fresh identity, restarting its trust and reputation assessments from scratch. Another example would be a seller agent which sell the same wines under different partial identities. In this sense, the probability that a buyer agent chooses one of their partial identities as the provider of the wine increases.

6.3. Accountability

As in many other applications, accountability must be preserved. Moreover, accountability is even more important in applications that involve commercial transactions, such as the one that we present in this paper. Indeed, the lack of accountability could cause principals not to be willing to participate in the e-marketplace. This is because if there is not a guarantee that accountability will be preserved, principals may be subject to many types of fraud. For instance, a buyer agent could purchase a bottle of wine from a seller agent. However, the seller agent may not ship that bottle of wine. If we consider a great number of bottles of wine (or a very expensive bottle of wine), the seller agent's principal should be sued for fraud. Although the final punishment may depend on the applicable laws for such a case, the important point is that the principal behind the seller agent should be completely identifiable.

7. Evaluation

In this section, we provide an extensive evaluation of the support that Magentix2 provides for preserving privacy based on the application we just presented in the previous section. In particular, we test whether or not information processing can be minimized by using the support that Magentix2 provides, whether or not this support introduces a bearable performance overhead, and whether or not existing trust and reputation models can be implemented on top of Magentix2. Finally, we also provide a discussion at the end of this section in which we argument how Magentix2 complies with all of the stated requirements of the application presented in Section 6 regarding privacy, trust and reputation, and accountability.

7.1. Avoiding Information Processing

Our aim in this section is to experimentally demonstrate that information processing, and thus, its possible undesired effects, can be minimized by changing RPIs. To this aim, we designed an experiment in which 10 different buyer agents purchase bottles of wine from a seller agent. The primary objective is for buyer agents to avoid that the seller agent is capable of obtaining a preference model from them.

As we explained in the previous section (Section 6), the seller agent trains an statistical classifier that models buyer agents' preferences (following the approach described in [56]). The statistical classifier is trained considering instances that correspond to wines that have been offered by the seller agent and that buyer agents have either accepted or declined during the negotiation protocol for a given purchase. The resulting trained classifier models the buyer agents' preferences with a given accuracy.

The aim of the experiment is to demonstrate that changing RPIs can significantly reduce the accuracy of the preference models obtained by the seller agent. Therefore, the seller agent cannot take advantage of these models to abuse buyer agents, e.g., performing price discrimination on buyer agents. In order to prove that changing RPIs can reduce information processing, we performed our experiment in which 10 buyer agent repeat 100 purchases of a bottle of wine with a varying number of RPI changes. That is, buyer agents start with 100 purchases and without any RPI change and end up using a different RPI for each of the 100 purchases. For each number of RPI changes we calculate the accuracy of the resulting classifier.

For our experiment, we considered the parameters that we sum up in Table 2. Each buyer agent has different preferences with respect to the wines that it likes. The specific preferences for each agent are shown in Table 3. According to that preferences, each buyer agent performs 100 different purchases of a bottle of wine. Each purchase involves a negotiation with a seller agent to get the desired wine. We assume that negotiations are always successful. However, we consider that negotiations can randomly involve from 1 up to 10 rounds of the protocol. That is, we simulate negotiations on which a buyer agent and a seller agent perform a maximum number of 10 rounds of the protocol. Based on this, a seller agent marks wines that are not accepted by a buyer agent as a negative instance (class "-"), while a seller agent marks the wine of the last step in the protocol (i.e., the wine that the buyer agent accepts to buy) as a positive instance (class "+"). After the 100 purchases, the seller agent use all the collected instances about a buyer agent to train a classifier. We measure the accuracy of the resulting trained classifier as the percent of correctly classified instances from an extra set of test instances (positive and negative) that we generate according to the buyer preferences.

We implemented the seller agent so that it uses a different classifier to obtain a model of the buyer agents' preferences based on the previous negotiations with them. In this way, we repeat the overall experiment to obtain the results regarding three different classifiers. Specifically, we consider the same classifiers as in [56]: the J48 decision tree algorithm (an implementation of the C.45 algorithm), the NNge classification rules algorithm (Nearest neighbor like algorithm using non-nested generalized exemplars) and the BayesNet classifier that is a classifier based on Bayesian networks.

Parameter	Description	Value
Ni	# of interactions per negotiation	10
Nn	# of negotiations	100
Nre	# repetitions of the experiment	100
Nbu	# of buyer agents	10
Nse	# of seller agents	1
Alg	Learning algorithms used	J48, NNge, BayesNet

Table 2: Parameters used in the privacy preservation experiments.

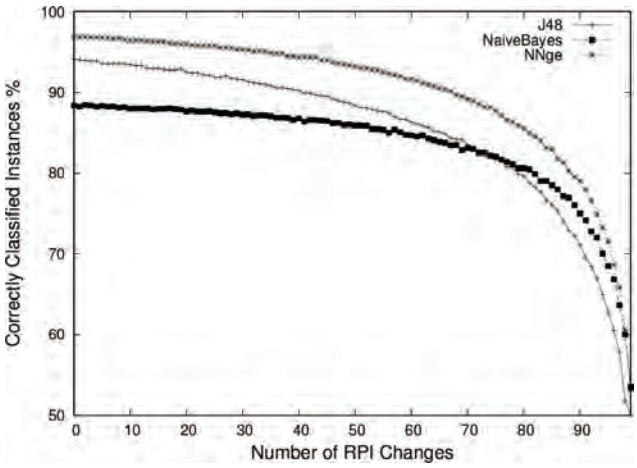


Figure 7: Privacy Preservation when changing RPIs

Figure 7 shows the results obtained for our experiment. These results are the average of the results obtained for each individual buyer agent. We can see that the percent of correctly classified instances behaves very similar regardless of the learning algorithm used. As expected, the more a buyer agent changes its RPI, the less the accuracy of the learning algorithms. In other words, the more a buyer agent changes its RPI, the less the seller agent is able to obtain a preference model of the buyer agent. Therefore, buyer agents can avoid that the seller agent performs information processing. Thus, it cannot make

Agent	Preferences
1	$(body = light \wedge flavor = delicate) \vee$ $(sugar = dry \wedge country = Portugal)$
2	$(color = red \wedge body = full \wedge flavor = strong) \vee$ $(color = white \wedge body = light \wedge flavor = moderate) \vee$ $(sugar = offDry \wedge country = Germany)$
3	$(sugar = sweet \wedge country = France) \vee$ $(sugar = dry \wedge country = Spain) \vee$ $(color = rose \wedge flavor = moderate)$
4	$(color = red \wedge body = medium \wedge flavor = moderate) \vee$ $(color = rose \wedge body = light \wedge sugar = dry) \vee$ $(color = white \wedge body = full \wedge sugar = dry)$
5	$(color = red \wedge body = medium \wedge flavor = moderate) \vee$ $(color = rose \wedge body = full \wedge country = Italy)$
6	$(sugar = dry \wedge color = red \wedge body = medium \wedge flavor = moderate \wedge country = France) \vee$ $(sugar = sweet \wedge color = white \wedge body = light \wedge flavor = delicate \wedge country = USA)$
7	$(sugar = dry \wedge color = red \wedge body = medium \wedge flavor = moderate \wedge country = France) \vee$ $(sugar = sweet \wedge color = white \wedge body = light \wedge flavor = delicate \wedge country = USA) \vee$ $(sugar = sweet \wedge color = rose \wedge body = medium \wedge country = Portugal)$
8	$(sugar = dry \wedge color = red \wedge body = medium \wedge flavor = moderate \wedge country = France) \vee$ $(sugar = offDry \wedge color = white \wedge body = medium \wedge flavor = delicate \wedge country = Australia)$
9	$(sugar = dry \wedge color = red \wedge body = medium \wedge flavor = moderate \wedge country = France) \vee$ $(sugar = sweet \wedge color = white \wedge body = light \wedge flavor = delicate \wedge country = USA) \vee$ $(sugar = sweet \wedge color = rose \wedge body = medium \wedge country = Australia)$
10	$(sugar = dry \wedge color = red \wedge body = medium \wedge flavor = moderate \wedge country = France) \vee$ $(sugar = sweet \wedge color = white \wedge body = light \wedge flavor = delicate \wedge country = USA) \vee$ $(sugar = sweet \wedge color = rose \wedge body = medium \wedge country = Portugal) \vee$ $(sugar = offDry \wedge color = red \wedge body = full \wedge flavor = strong \wedge country = NewZeeland)$

Table 3: Buyer agent's preferences

any secondary use of the processed data.

Another remarkable phenomenon is that it seems that there is a threshold in the number of RPI changes (≈ 90 RPI changes) from which the accuracy of learning algorithms decreases at a faster rate. This reinforces the thesis of some privacy-enhancing technologies researchers that encourage users to change their identities as often as possible. Moreover, it is clear that the maximum privacy preservation is achieved when buyer agents change their RPI for each new interaction, which is known as transaction pseudonyms in the privacy-enhancing technologies literature.

7.2. Performance Evaluation

In the previous section, we described an experiment that demonstrates that changing RPIs can minimize information processing. However, changing RPIs could also have costs associated to the change, e.g., a temporal cost. If this temporal cost is too high, it could discourage agents from using the RPI change capability offered by Magentix2. In this section, we describe the experiment that we carried out in order to evaluate the temporal cost of changing RPIs in the Magentix2 agent platform.

We performed a similar experiment as the one presented in the previous section, in which agents change their RPI a number of times in order to reduce information processing. In this case, we only focus on two agents, one buyer agent and one seller agent. Moreover, we do not calculate the accuracy of the preference model that the seller obtains but we calculate the temporal cost for the buyer to change its RPI a specific number of times. This is in order to ascertain whether or not it is temporally feasible for a buyer agent to change their RPI as many times as needed to prevent the seller agent from constructing a detailed model on the buyer agent's preferences.

We performed a simulation in which the buyer agent carries out 100 different purchases of a bottle of wine. Each purchase involves a negotiation with the seller agent to get the desired wine. We assume that negotiations are always successful. Moreover, we consider that negotiations can randomly involve from 1 up to 10 rounds of the protocol. The buyer agent repeats the 100 purchases with a varying number of RPI changes. That is, the buyer agent starts with 100 purchases and without any RPI change, and it ends up using a different RPI for each of the 100 purchases (transaction pseudonyms). For each number of RPI changes we calculate the RTT time of the messages exchanged between the buyer agent and the seller agent.

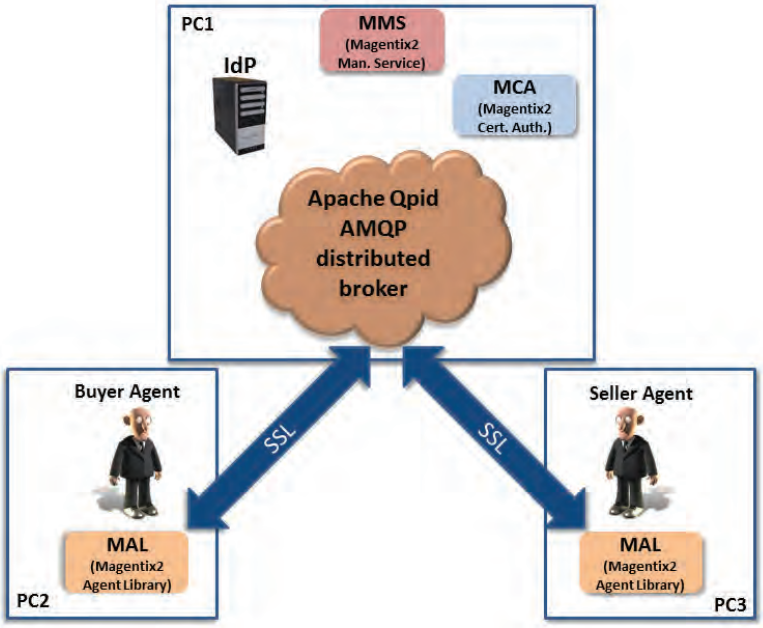


Figure 8: Location of the different components.

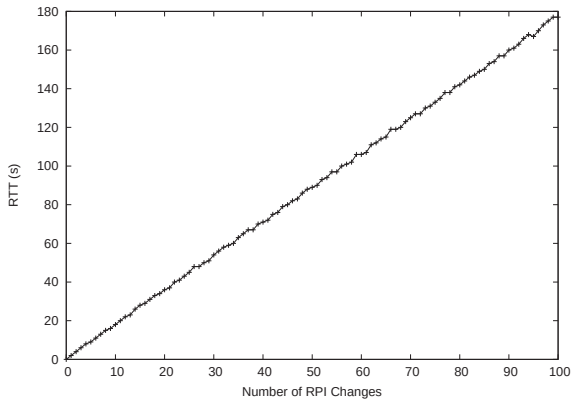


Figure 9: Performance per number of RPI changes

In order for the experiment to be in an absolutely controlled environment, we do not use any external IdP but we use the IdP prototype described in [50] as both the PIdP (the IdP that issues PPIs) and the RIdP (the IdP that issues RPIs). Moreover, we used 3 PCs Intel(R) Core(TM) 2 Duo CPU @ 2.60GHz, 1GB RAM, Ubuntu 11.04 (x86_64) and Linux Kernel 2.6.38. The computers are connected to each other via a 100Mb Ethernet switch. The security parameters are the following: certificate keys are 1024 bits RSA keys, SHA-1 hash function with 96-bit keys to perform HMAC computations, and the saml2 tokens to be issued by the IdP contain keys of 256 bits. The location of the different components is shown in Figure 8: PC number 1 runs the Qpid Broker, the MMS, the MCA, and the IdP; PC number 2 runs the buyer agent; and finally, PC number 3 runs the seller agent.

Figure 9 shows the results obtained. These results mean that changing a RPI has a temporal cost that is linear with the number of changes to be made. These results also mean that the temporal cost of a single change is constant, and thus, it is not related to the number of previously performed changes. Therefore, we can claim that agents developed in Magentix2 can minimize information processing about their princi-

pals' data (as shown in the previous section) without incurring in a not affordable temporal cost. Moreover, as the cost of one single change is constant, a buyer agent can predict in advance the temporal cost of the changes it requires to reduce seller agents from processing its data, and then, decide if it performs the required changes or not.

7.3. Building Trust and Reputation

Finally, we also wanted to test whether trust and reputation models can be implemented using partial identities. The aim is to validate that agents in Magentix2 can use the trust and reputation model that they prefer. Thus, if two agents establish trust or reputation trough permanent partial identities, identity-related vulnerabilities are avoided because IdPs do not allow agents to change their permanent partial identity.

We implemented one seller agent and three buyer agents. Each buyer agent uses its own trust and reputation machinery to model the trustworthiness of the seller agent based on the previous interactions and the personal attributes of the seller agent. The PPIs issued by the PIdP take values for two attributes: name and role. Both seller agents and buyer agents register into the system using the PPI that the PIdP issued for them – so that the system does not know the real identity of the legal person that agents are acting on behalf of. In this way, buyer agents are able to identify providers from previous interactions and build their own trust and reputation models, being sure that the seller agent will not be able to hold any other PPI.

The seller agent follows a normal distribution with a mean of 0 and standard deviation of 1 to model whether it carries out the service requested in the way that buyer agents expect it. In this sense, when a buyer agent requests a service to the seller, if the value returned is in the interval [-1,1], the buyer agent considers that the seller performed as expected. If the value returned is out of this interval the buyer agent considers that the seller agent did not perform as expected. When the seller agent performs as expected, the buyer agent rates it with 1. When the seller agent does not perform as expected, the buyer agent rates it with 0. These ratings are inputs of the trust and reputation model each buyer agent has.

Each buyer agent runs a different trust and reputation model that is fed using past interactions with the seller agent and attributes from the seller agent's partial identity. We implemented three models (each one for each buyer agent), one simply using a mean of all the previous performances to compute a trust value, one using the SinAlpha trust model [57] that considers previous interactions, and finally, one using the Fire trust and reputation model [58] which uses, among other information, previous interactions and the role of the agents to be trusted. Figure 10 presents trust values for each buyer after 100 interactions with the seller agent. These trust values are the result of each buyer agent's trust and reputation model given the results of the interactions with the seller agent. We can observe that the trust models perform as expected and the tendency is very similar in all of the models implemented . This tendency corresponds to whether or not the seller agent performs as buyers expect. Thus, we can conclude that the trust models considered can be implemented using partial identities.

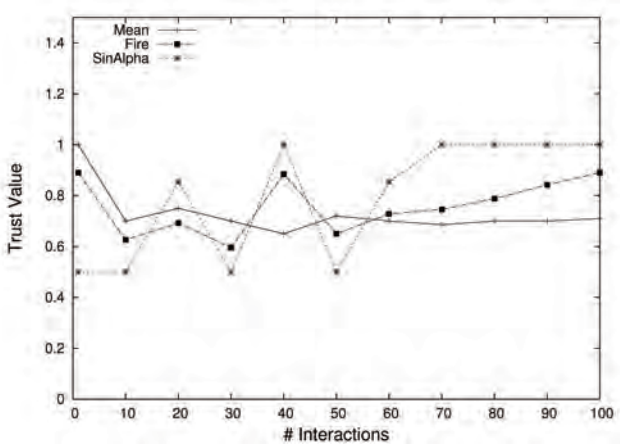


Figure 10: Trust values for the seller agent obtained by each trust model (implemented by each buyer agent).

7.4. Discussion

The agent-based e-marketplace for wine application benefits from the features that Magentix2 provides for preserving privacy and accountability as well as avoiding identity-related vulnerabilities of trust and reputation. Magentix2 provides support for preserving privacy by means of mechanisms that can be used by agents to avoid information collection and processing.

Regarding information collection, the information that a buyer agent and a seller agent exchange is confidential. To this aim, Magentix2 is in charge of automatically generating a X.509 certificate for each agent

partial identity so that they can communicate trough SSL. Therefore, communications are encrypted from one agent to another so that the information contained in such communications cannot be accessed by any other unauthorized agent.

Regarding information processing, buyer agents can hold multiple RPIs. Therefore, they can avoid information processing as shown in Section 7.1. Indeed, the most privacy preserving option is for buyer agents to change their RPIs for each different purchase of a bottle of wine. Moreover, as shown in Section 7.2, RPI changes come at a bearable efficiency cost. Thus, agents can change their RPIs without an important efficiency penalty.

In Section 7.3, we demonstrate that trust and reputation models can be implemented using partial identities. Buyers and sellers are able to authenticate their partial identities (both PPIs and RPIs). Therefore, they are allowed to recognize to each other from interaction to interaction and establish trust and reputation relationships. Moreover, if trust and reputation is established using PPIs, identity-related vulnerabilities of trust and reputation models are avoided. This is due to the fact that there is no chance for a buyer or a seller to have two different PPIs.

Finally, accountability is preserved. IdPs conceal real identities. Therefore, both Magentix2 and the rest of the agents are a priori not able to link a given partial identity (PPI or RPI) to the corresponding original real identity. However, if an agent misbehaves when using one of its partial identities (PPI or RPI), a court could require the IdP that issued the partial identity to disclose the real identity that is behind the partial identity. For instance, if an agent performs fraud, the real identity of its principal could be known so that this principal can be sued for fraud.

8. Related Work

8.1. Privacy-enhancing Agent Platforms

There are many Agent Platforms (APs) developed by the agent community [2]. However, only a few of them currently take security concerns into account. For instance, Jade [59], Magentix[12] [60], AgentScape [61], SECMAP [62], Tryllian ADK [63], Cougaar [64], SeMoA [65], and Voyager [66] are security-concerned APs.

Current security-concerned APs provide confidentiality for the messages exchanged by the agents running on top of them. To this aim, APs use existing secure data transfer technologies such as Kerberos [26], SSL [27], and TLS [28]. These technologies allow the encryption of messages before transferring them and the decryption of messages once they are received. As a result, if an agent A sends a message to an agent B using these technologies, A is sure that B will be the only one able to read this message.

Confidentiality is a necessary condition to preserve privacy, it can prevent information collection but it is not sufficient to prevent information processing. Only a few of the security-concerned APs explained above implement some kind of support to prevent information processing. Specifically, only Magentix, Secmap, AgentScape, and Cougaar allow agents to authenticate each other using their unique agent identity. With this identity, agents can act pseudonymously, i.e., agents can act on behalf of their principal without using the identity of their principal. However, agents cannot hold more than one pseudonym, i.e., principals should use a different agent each time they want to use a different pseudonym.

Warnier and Brazier [67] also present a mechanism for the AgentScape AP that offers pseudonymity by means of what they call *handles*. Handles are pseudonyms that agents can use to send/receive messages to/from other agents. At will, agents can request new handles to the AP. Moreover, the AP is the only one that knows the association between handles and GUIDs (global unique identities of the agents). An agent can also obtain anonymity by simply using a different handle for each transaction (transaction pseudonyms). AgentScape also offers an automatic anonymity service. Agents can send messages anonymously without having to manage pseudonyms. This service is provided by agents called *anonymizers*. When an agent wants to send a message anonymously, this message is redirected to an anonymizer. Then, this anonymizer is in charge of removing the original handle of the sender from the message, replacing it with another (possibly new) handle, and sending the message to the intended recipient. If the intended recipient replies, this reply is forwarded to the sender of the original message. The original sender of the message must notify when a transaction ends. For each new transaction the anonymizer

[12]Note that the support we present in this paper is for Magentix2, which is a completely redesigned version of Magentix [60].

generates a new handle.

APs that provide support for pseudonymity (e.g. by providing APIs to create and manage pseudonyms) do not consider that pseudonyms can be issued by external third parties. That is, APs themselves are in charge of issuing the pseudonyms. Thus, the AP itself (and the anonymizer agents for the case of AgentScape) knows the relationship of pseudonyms to each other and to the principal involved. This usually implies that the organization or company that hosts the specific system (e.g. eBay in the case of an e-marketplace) knows the association of pseudonyms to each other and to principals. Therefore, information processing could still be easy. In Magentix2 partial identities are issued by IdPs, and then, these partial identities are used in Magentix2. Therefore, identity management is decoupled from the system where the identities are to be used. Note that this may not completely prevent information processing. There is still the possibility that an agent running on Magentix2 or Magentix2 itself could collude with some of the IdPs in order to be able to link a partial identity to its corresponding real identity. However, agents could (partially) address this by obtaining different partial identities from different IdPs so as to decrease the probability of being traced back in case of collusion.

Finally, there are also approaches that have been implemented on top of APs that by themselves do not provide support to avoid information collection and processing. In this way, Lee et al. [68] present an approach based on P3P[13] for the Jade AP. The privacy-enhancing agent (PEA) is in charge of automatically retrieving P3P policies of service providers and evaluates whether or not these policies are compliant with its principal's policy. When a principal attempts to access a website, PEA automatically retrieves the website P3P policy and compares it to its principal's preferences. If PEA detects potential privacy violations (i.e., the principal's preferences and the website's P3P policy do not match) or is unable to read the policy of the website, it notifies its principal so that the principal can decide to desist in accessing the website. This approach does not consider that a website may not comply with its announced policy, and, thus, principals' privacy breaches are still possible. Moreover, this approach is not a generic support from the AP and may not be suitable in other domains.

[13]The Platform for Privacy Preferences 1.0 `http://www.w3.org/TR/P3P/`

8.2. Accountability Preservation

Secure APs that do not provide support for pseudonymity (such as Jade [59], Tryllian ADK [63], SeMoA [65], and Voyager [66]) usually preserve accountability. This is because they base authentication on the the identity of the agents' users. Therefore, the identity of the agents' users is always available. If an agent misbehaves, its users identity can be known.

Accountability is more difficult in secure APs that provide support for pseudonymity (such as Magentix [60], Secmap [62], AgentScape [61], and Cougaar [64]). This is because these APs authenticate based on the identity (pseudonym) of the agents. Thus, the identity of the corresponding user may not be known. In order to avoid a lack of accountability that could cause a sense of impunity and encourage abuse, Magentix and AgentScape keep track of the association between principals and pseudonyms.

Magentix2 does not keep track of the association between principals and pseudonyms. It relies on trusted external identity providers to keep this information. However, accountability is preserved because these trusted external identity providers can reveal the corresponding real identities for law enforcement.

8.3. Identity-related vulnerabilities of Trust and Reputation Models

There have been some different approaches in the related literature to tackle identity-related vulnerabilities of trust and reputation [69]. There is one approach that bases on adding a monetary cost for entering a given system [49]. Thus, an potential malicious agent would have a sufficient incentive (if the fee is high enough compared to the benefit expected) not to re-entry the system with a new identity. The main problem of this approach is that if the cost for entering the particular system is too high, even potentially benevolent agents may choose not to enter the system because of the high cost associated to it.

Yu et al. [70] present an approach based on social networks represented as a graph in which nodes represent pseudonyms and edges represent human-established trust relationships among them in the real world. They claim that malicious users can create many pseudonyms but few trust relationships. They exploit this property to bound the number of pseudonyms to be considered for trust and reputation. However, this approach is not appropriate for

open MAS in which agents act on behalf of principals that may not be known in the real world.

There is another approach that consist of reputation models specifically designed to meet some mathematical properties that are proved to avoid identity-related vulnerabilities. For instance, Cheng et al. [71] have demonstrated several conditions using graph theory that must be satisfied when calculating reputation in order for reputation models to be resilient to sybil attacks. The only drawback of this kind of approaches is that they usually need a particular and specific way to calculate reputation ratings about an individual. Thus, this approach cannot be applied to reputation models that follow other approaches for managing reputation ratings.

Magentix2 follows a different approach, it bases on completely avoiding the possibility for agents to change their identity. In this way, agents can only hold one PPI for a given system. Therefore, when trust and reputation are established trough PPIs, agents cannot get rid of the trust and reputation ratings they got from other agents in the system.

9. Conclusions

In this paper, we present the support that the Magentix2 AP provides for enhancing privacy in the applications built on top of it. This support also allows agents to avoid identity-related vulnerabilities of trust and reputation models. Moreover, this support avoids the lack of accountability of the principals involved. All these features are crucial for encouraging principals' trust in agent-based applications.

We experimentally show that the support that Magentix2 provides can be used for agents to minimize information processing. Moreover, we demonstrate that the efficiency cost of changing a RPI is absolutely bearable. Finally, we also demonstrate that current trust and reputation models can be implemented in Magentix2 by implementing some of them.

Agents running on Magentix2 can use the support it provides to enhance privacy while preserving accountability and avoiding identity-related vulnerabilities of trust and reputation at will depending on their principals' needs. An agent can create as many RPIs as needed to avoid information processing. Otherwise, an agent can use a PPI if it is interested in building trust and reputation. Thus, other agents can trust in this agent while being sure that it cannot perform whitewashing and sibyl attacks because PPI cannot be changed.

As future work, we would like to explore the possibility of agents automatically deciding whether or not they change their RPI for their next interaction. This decision may be based on the privacy that is to be lost if the RPI is not changed (e.g., an estimation of the model that other agents could have about their preferences) and the possible benefits of not changing the RPI. In particular, the decision of whether or not changing a RPI could be based on disclosure decision making models that consider a tradeoff between the privacy that is lost and the benefit of doing so, such as privacy-utility tradeoff models [72] and privacy-intimacy tradeoff models [73].

10. Acknowledgments

This work has been partially supported by CONSOLIDER INGENIO 2010 under grant CSD2007-00022, and project TIN2009-13839-C03-01.

References

[1] M. Wooldridge, An Introduction to MultiAgent Systems, Wiley, 2002.

[2] J. M. Alberola, J. M. Such, A. Garcia-Fornes, A. Espinosa, V. Botti, A performance evaluation of three multiagent platforms, Artificial Intelligence Review 34 (2010) 145–176.

[3] G. Piolle, Y. Demazeau, J. Caelen, Privacy management in user-centred multi-agent systems, in: G. O'Hare, A. Ricci, M. O'Grady, O. Dikenelli (Eds.), Engineering Societies in the Agents World VII, Vol. 4457 of LNCS, Springer Berlin / Heidelberg, 2007, pp. 354–367.

[4] J. M. Such, A. Espinosa, A. Garcia-Fornes, A survey of privacy in multi-agent systems, Knowledge Engineering Review (2012) In press.

[5] M. Fasli, On agent technology for e-commerce: trust, security and legal issues, Knowledge Engineering Review 22 (1) (2007) 3–35.

[6] C. Sierra, Agent-mediated electronic commerce, Autonomous Agents and Multi-Agent Systems 9 (3) (2004) 285–301.

[7] T. Mitchell, R. Caruana, D. Freitag, J. McDermott, D. Zabowski, Experience with a learning personal assistant, Communications of the ACM 37 (7) (1994) 80–91.

[8] E. Weitnauer, N. Thomas, F. Rabe, S. Kopp, Intelligent agents living in social virtual environments bringing max into second life, in: Intelligent Virtual Agents (IVA), Springer Berlin / Heidelberg, 2008, pp. 552–553.

[9] M. Montaner, B. López, J. De La Rosa, A taxonomy of recommender agents on the internet, Artificial intelligence review 19 (4) (2003) 285–330.

[10] S. Warren, L. Brandeis, The right to privacy, Hardvard Law Review 4 (5).

[11] A. Acquisti, S. Gritzalis, C. Lambrinoudakis, S. di Vimercati (Eds.), Digital Privacy: Theory, Technologies, and Practices, Auerbach Publications, 2008.
[12] A. Westin, Privacy and Freedom, New York Atheneum, 1967.
[13] K. Rannenberg, D. Royer, A. Deuker (Eds.), The Future of Identity in the Information Society: Challenges and Opportunities, Springer Publishing Company, Incorporated, 2009.
[14] B. Schermer, Software agents, surveillance, and the right to privacy: a legislative framework for agent-enabled surveillance, Amsterdam Univ Pr, 2007.
[15] S. Fischer-Hübner, H. Hedbom, Benefits of privacy-enhancing identity management, Asia-Pacific Business Review 10 (4) (2008) 36–52.
[16] S. Garfinkel, Database nation: the death of privacy in the 21st century, O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.
[17] J. Borking, B. Van Eck, P. Siepel, D. Bedrijf, Intelligent software agents: Turning a privacy threat into a privacy protector, Registratiekamer, The Hague, 1999.
[18] H. Taylor, Most People Are "Privacy Pragmatists" Who, While Concerned about Privacy, Will Sometimes Trade It Off for Other Benefits., Harris Interactive, 2003.
[19] D. Hoffman, T. Novak, M. Peralta, Building consumer trust online, Communications of the ACM 42 (4) (1999) 80–85.
[20] D. Solove, A taxonomy of privacy, University of Pennsylvania Law Review 154 (3) (2006) 477–560.
[21] W. Stallings, Network security essentials : applications and standards, Prentice Hall, 2010.
[22] B.-J. Koops, R. Leenes, Identity theft, identity fraud and/or identity-related crime, Datenschutz und Datensicherheit - DuD 30 (2006) 553–556.
[23] L. Bilge, T. Strufe, D. Balzarotti, E. Kirda, All your contacts are belong to us: automated identity theft attacks on social networks, in: Proceedings of the 18th international conference on World wide web (WWW), ACM, New York, NY, USA, 2009, pp. 551–560.
[24] M. Petkovic, W. Jonker (Eds.), Security, Privacy and Trust in Modern Data Management (Data-Centric Systems and Applications), Springer-Verlag, 2007.
[25] M. Stamp, Information Security: Principles and Practice, Wiley-Interscience, 2006.
[26] C. Neuman, T. Yu, S. Hartman, K. Raeburn, The Kerberos Network Authentication Service (V5), no. 4120 in Request for Comments, IETF, 2005.
[27] A. Frier, P. Karlton, P. Kocher, The secure socket layer, Tech. Rep. MSU-CSE-00-2, Netscape Communications (1996).
[28] T. Dierks, C. Allen, The tls protocol version 1.0, RFC 2246 (1999).
URL `http://www.ietf.org/rfc/rfc2246.txt`
[29] S. Spiekermann, L. F. Cranor, Engineering privacy, IEEE Transactions on Software Engineering 35 (1) (2009) 67–82.
[30] M. Hildebrandt, S. Gutwirth, Profiling the European Citizen: Cross-Disciplinary Perspectives, Springer Publishing Company, Inc., 2008.
[31] A. Odlyzko, Privacy, economics, and price discrimination on the internet, in: Proceedings of the 5th international conference on Electronic commerce (ICEC), ACM, New York, NY, USA, 2003, pp. 355–366.
[32] S. Spiekermann, Individual price discriminaton - an impossibility?, in: Proceedings of the Workshop on Privacy and Personalization held with the International Conference for Human-Computer Interaction (CHI), 2006, pp. 47–52.
[33] H. J. Smith, S. J. Milberg, Information privacy: measuring individuals' concerns about organizational practices, MIS Quarterly 20 (1996) 167–196.
[34] D. Chaum, Security without identification: transaction systems to make big brother obsolete, Commun. ACM 28 (1985) 1030–1044.
[35] A. Pfitzmann, M. Hansen, A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management, http://dud.inf.tu-dresden.de/Anon_Terminology.shtml, v0.34 (Aug. 2010).
[36] D. Chaum, A. Fiat, M. Naor, Untraceable electronic cash, in: Proceedings on Advances in cryptology(CRYPTO), Springer-Verlag New York, Inc., New York, NY, USA, 1990, pp. 319–327.
[37] E. Aïmeur, G. Brassard, F. Onana, Secure anonymous physical delivery, IADIS International Journal on WWW/Internet 4 (1) (2006) 55–59.
[38] A. Bhargav-Spantzel, J. Camenisch, T. Gross, D. Sommer, User centricity: A taxonomy and open issues, J. Comput. Secur. 15 (2007) 493–527.
[39] M. Bishop, Computer Security: Art and Science, Addison-Wesley, 2002.
[40] S. Chopra, L. White, Artificial agents - personhood in law and philosophy, in: Proceedings of The 13th European Conference on Artificial Intelligence (ECAI), IOS press,, 2004, pp. 635–639.
[41] T. Balke, T. Eymann, The conclusion of contracts by software agents in the eyes of the law, in: Proc. of The 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS), IFAAMAS, 2008, pp. 771–778.
[42] M. Fasli, Agent Technology For E-Commerce, John Wiley & Sons, 2007.
[43] M. Hansen, P. Berlich, J. Camenisch, S. Clau, A. Pfitzmann, M. Waidner, Privacy-enhancing identity management, Information Security Technical Report 9 (1) (2004) 35 – 44.
[44] I. Pinyol, J. Sabater-Mir, Computational trust and reputation models for open multi-agent systems: a review, Artificial Intelligence Review (2011) In press. DOI 10.1007/s10462–011–9277–z.
[45] J. Sabater, C. Sierra, Review on computational trust and reputation models, Artificial Intelligence Review 24 (2005) 33–60.
[46] E. Carrara, G. Hogben, Reputation-based systems: a security analysis, ENISA Position Paper (2007).
[47] A. Jøsang, R. Ismail, C. Boyd, A survey of trust and reputation systems for online service provision, Decis. Support Syst. 43 (2) (2007) 618–644.
[48] A. Jøsang, J. Golbeck, Challenges for Robust Trust and Reputation Systems, in: Proceedings of the 5th International Workshop on Security and Trust Management (STM), 2009, pp. 1–12.
[49] E. J. Friedman, P. Resnick, The social cost of cheap pseudonyms, Journal of Economics and Management Strategy 10 (1998) 173–199.
[50] J. M. Such, A. Espinosa, A. Garcia-Fornes, V. Botti, Par-

tial identities as a foundation for trust and reputation, Engineering Applications of Artificial Intelligence 24 (7) (2011) 1128–1136.

[51] M. Luck, P. McBurney, O. Shehory, S. Willmott, Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing), AgentLink, 2005.

[52] S. Vinoski, Advanced message queuing protocol, IEEE Internet Computing 10 (6) (2006) 87–89. doi:http://doi.ieeecomputersociety.org/10.1109/MIC.2006.116.

[53] FIPA, FIPA ACL Message Structure Specification, FIPA (2001).
URL http://www.fipa.org/specs/fipa00061/

[54] S. Clauβ, D. Kesdogan, T. Kölsch, Privacy enhancing identity management: protection against re-identification and profiling, in: Proceedings of the workshop on Digital identity management (DIM), ACM, New York, NY, USA, 2005, pp. 84–93.

[55] R. Aydoan, P. Yolum, Learning opponents preferences for effective negotiation: an approach based on concept learning, Autonomous Agents and Multi-Agent Systems (2010) 1–3710.1007/s10458-010-9147-0.
URL http://dx.doi.org/10.1007/s10458-010-9147-0

[56] E. Serrano, M. Rovatsos, J. Botia, Mining qualitative context models from multiagent interactions (extended abstract), in: Proceedings of the tenth international joint conference on Autonomous agents and multiagent systems (AAMAS), IFAAMAS, 2011, pp. 1215–1216.

[57] J. Urbano, A. P. Rocha, E. Oliveira, Computing confidence values: Does trust dynamics matter?, in: EPIA '09: Proceedings of the 14th Portuguese Conference on Artificial Intelligence, Springer-Verlag, 2009, pp. 520–531.

[58] T. D. Huynh, N. R. Jennings, N. R. Shadbolt, An integrated trust and reputation model for open multi-agent systems, Autonomous Agents and Multi-Agent Systems 13 (2) (2006) 119–154.

[59] JADE Board, Jade security guide, http://jade.tilab.com (2005).

[60] J. M. Such, J. M. Alberola, A. Espinosa, A. Garcia-Fornes, A Group-oriented Secure Multiagent Platform, Software: Practice and Experience 41 (11) (2011) 1289–1302.

[61] T. B. Quillinan, M. Warnier, M. Oey, R. Timmer, F. Brazier, Enforcing security in the agentscape middleware, in: Proceedings of the workshop on Middleware security (MidSec), ACM, New York, NY, USA, 2008, pp. 25–30.

[62] S. Ugurlu, N. Erdogan, An overview of secmap secure mobile agent platform, in: Proceedings of Second International Workshop on Safety and Security in Multiagent Systems, 2005.

[63] H. Xu, S. M. Shatz, Adk: An agent development kit based on a formal design model for multi-agent systems, Journal of Automated Software Engineering 10 (2003) 337–365.

[64] A. E. Newman, Cougaar developers' guide, http://www.cougaar.org (2004).

[65] V. Roth, M. Jalali-Sohi, Concepts and architecture of a security-centric mobile agent server, in: Proceedings of the Fifth International Symposium on Autonomous Decentralized Systems (ISADS), IEEE Computer Society, Washington, DC, USA, 2001, pp. 435–444.

[66] Recursion Software Inc., Voyager security guide, http://www.recursionsw.com/ (2008).

[67] M. Warnier, F. Brazier, Anonymity services for multiagent systems, Web Intelligence and Agent Systems 8 (2) (2010) 219–232.

[68] H.-H. Lee, M. Stamp, An agent-based privacy-enhancing model, Inf. Manag. Comput. Security 16 (3) (2008) 305–319.

[69] K. Hoffman, D. Zage, C. Nita-Rotaru, A survey of attack and defense techniques for reputation systems, ACM Comput. Surv. 42 (2009) 1:1–1:31. doi:http://doi.acm.org/10.1145/1592451.1592452.
URL http://doi.acm.org/10.1145/1592451.1592452

[70] H. Yu, M. Kaminsky, P. B. Gibbons, A. Flaxman, Sybilguard: defending against sybil attacks via social networks, in: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM), ACM, New York, NY, USA, 2006, pp. 267–278.

[71] A. Cheng, E. Friedman, Sybilproof reputation mechanisms, in: Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems, P2PECON '05, ACM, New York, NY, USA, 2005, pp. 128–132. doi:http://doi.acm.org/10.1145/1080192.1080202.
URL http://doi.acm.org/10.1145/1080192.1080202

[72] A. Krause, E. Horvitz, A utility-theoretic approach to privacy and personalization, in: Proceedings of the 23rd national conference on Artificial intelligence (AAAI), AAAI Press, 2008, pp. 1181–1188.

[73] J. M. Such, A. Espinosa, A. Garcia-Fornes, C. Sierra, Self-disclosure decision making based on intimacy and privacy, Information Sciences (2012) DOI: 10.1016/j.ins.2012.05.003.

Appendix
OASIS06

https://ieeexplore.ieee.org/document/10284735 February 27, 2026
IEEE.org IEEE Xplore IEEE SA IEEE Spectrum More Sites Subscribe Donate Cart Create Account Personal Sign In
IEEE Xplore® Browse My Settings Help Institutional Sign In IEEE
All
ADVANCED SEARCH

Formal Analysis of Information Card Federated Identity-Management Protocol

Publisher: CIE Cite This PDF

Juan Wang ; Hongxin Hu ; Bo Zhao ; Fei Yan ; Huanguo Zhang ; Qianhong Wu All Authors

1 Cites in Paper
135 Full Text Views
Open Access

Abstract
Document Sections
I. Introduction
II. Infocard Protocol
III. Formal Verification of Infocard Protocol
IV. Related Work
V. Conclusion
Authors
Figures
References
Citations
Keywords
Metrics
Footnotes

Abstract:
Information Card (InfoCard) is a user-centric identity management metasystem. It has been accepted as a standard of OASIS Identity Metasystem Inter-operability Technical Committee. However, there is currently a lack of security analysis to InfoCard protocol, especially, with formal methods. In this paper, we accommodate such a requirement by analyzing security properties of InfoCard protocol adopting a formal protocol analysis tool. Our analysis result discovers that current InfoCard protocol is vulnerable against the session replay attack. Furthermore, we reveal the importance of two optional elements in InfoCard metasystem, token scope and proof key, and found that InfoCard protocol will be susceptible to man-in-the-middle attack and token replay attack if these two optional elements lack.

Published in: Chinese Journal of Electronics (Volume: 22 , Issue: 1, January 2013)

Page(s): 83 - 88 **DOI:** 10.23919/CJE.2013.10284735

Date of Publication: January 2013 **Publisher:** CIE

ISSN Information:

Funding Agency:

More Like This

A Gen2-Based RFID Authentication Protocol for Security and Privacy
IEEE Transactions on Mobile Computing
Published: 2009

On the Security of Privacy-Enhanced Authentication Protocol for Federated Learning in VANETs
IEEE Transactions on Information Forensics and Security
Published: 2024

Show More

SECTION I.
Introduction

In open and collaborative Internet, managing the necessary digital identities is a complex hurdle. Traditionally, Service providers collect and manage user identity related information which has led to proven attacks like identity theft and privacy violation. Recently, user-centric identity management approaches have emerged. User-centric identity management allows users to control their own digital identities. Users can select their credentials when responding to an authenticator or attribute requester, giving users more rights and responsibility over their identity information. Information Card (InfoCard)[1]- [3] is a user centric federated identity

management metasystem for reducing the reliance on passwords and improving the privacy of personal information.

InfoCard protocol is as the foundation of an InfoCard system implementation and its security should be guaranteed. However, the design of security protocols is highly error-prone. Security flaws have been found in many protocols[4]. For InfoCard protocol, security issues have not yet been explored in depth, especially using formal methods. In this paper, we focus on the formal analysis of InfoCard protocol. We first provide an informal description about the detailed communication proceeds and message format of InfoCard protocol and thoroughly analyze the implementation mechanism of security goals. Then, we translate the informal description of InfoCard protocol and its security goals to a formal specification based on High level protocol specification language (HLPSL). Finally, we verify the protocol using an automatic formal analysis tool, Automated validation of Internet security protocols and applications (AVISPA)[5], [6].

Our contribution is three-fold. First, we model and analyze the authentication and secrecy goals of InfoCard protocol using a formal method. Second, we uncover a session replay attack. Finally, we reveal the importance of two main optional elements, token scope and proof key, in an InfoCard metasystem. We found that InfoCard protocol is susceptible to man-in-the-middle attack and token replay attack if these two optional elements lack in the implementation of InfoCard protocol. To the best of our knowledge, this is the first time that above attacks to Infocard protocol are identified.

The remainder of this paper is organized as follows. In Section II, we describe the detailed communication proceeds and message format of InfoCard protocol and analyze the implementation mechanism of security goals. In Section III, we analyze and verify InfoCard protocol by a formal analysis tool, AVISPA, and discuss the importance of two main optional elements, token scope and proof key. Section IV discusses the related work. Finally, Section V provides conclusions.

SECTION II.
Infocard Protocol

In accordance with the specification of OASIS Identity Metasystem Interoperability standard, the InfoCard protocol involves five components: User (U), Identity selector (IS), Client application (A), Relying party (RP) and Identity provider (IP).

The interaction process of the InfoCard system is depicted in Fig. 1.

(1) A user accesses a relying party (a website) which allows the user to log in the relying party with InfoCard.

(2) The relying party returns the requested page and sends the relying party's policy.

(3) The client application calls the identity selector and forwards the relying party's policy.

(4) In accordance with the relying party's policy, the identity selector searches in the user's information cards collection and finds the information cards which satisfy the requested claims in the relying party's policy. The user then selects one of these cards to log on the relying party.

(5) The identity selector sends a token request to the corresponding identity provider who issued the selected card.

(6) The identity provider authenticates the user by the provided authentication credential. If the user has passed the authentication, the identity provider

issues a security token to the identity selector.

(7) The identity selector sends the service request message and the security token to the relying party.

(8) The relying party successfully authenticates the user and sends desired service or resource to the client.

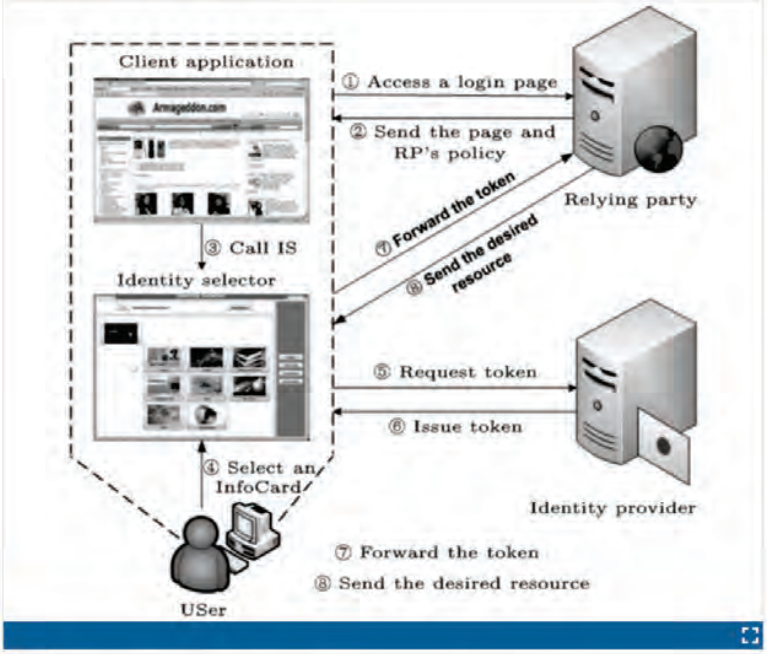


Fig. 1.
Framework of InfoCard

Show All

Although the InfoCard system includes five components and eight operating steps, but in fact it is a typical three party authentication protocol. We can combine the user, identity selector and client application as "Client", and neglect the steps used to retrieve the policy from the point view of security protocol analysis, just considering the steps using the mutual authentication process. Moreover, it must be explained that we just consider the managed card protocol in this paper because the selfissued card protocol does not have a separate identity provider and can be regarded as a special case of the managed card protocol.

On the basis of above considerations, we can give the main authentication procedure of managed card protocol. The main authentication procedure and message format are according to Microsoft's implementation of InforCard protocol[7], [8] and the OASIS identity metasystem specification[1].

The elements of n_1 to n_9, n_{ce} and n_{se} are random numbers. k_1, k_2 and k_t are symmetric keys. PK_{IP} represents the public key of IP and PK_{IP}^{-1} represents the private key of IP. Ck_{sig} is the signature key of client. Ck_{enc} is the encryption key and $C_{kendorse}$ is the endorsing signature key. RPk_{sig} is the signature key of RP and RPk_{enc} is the encryption key. M_{rst} indicates request message and M_{resp} indicates response message. M_{tok} indicates the issued token by identity provider. SAML is the type of issued token. M_{enctok} represents the encyphered token. $cardId$ is the unique identity of a card. $types_{RP}$ stands for the types of the required claims. $claims_U$ represents claims with the attribute values for the required $types_{RP}$. M_{mac} is the message signature. M_{proof} is the endorsing signing message proving possession of proof key.

The detailed communication proceeds and message format of managed card protocol are described as following:

(1) C->IP: $\{\{k_1\}_{PK_{IP}}, n_1, n_2, \{\{M_{user}, M_{rst}\}_{Ck_{sig}}, M_{user}, M_{rst}\}_{Ck_{enc}}\}$

$$M_{user} = (U, pwd), M_{rst} = (cardId, types_{RP}, RP, n_{ce})$$

$$Ck_{sig} = \text{PSHA1}(k_1, n_1)$$

$$Ck_{enc} = \text{PSHA1}(k_1, n_1)$$

(2) IP->C:
$\{n_3, n_4, \{\{M_{rstr}\}_{IPk_{sig}}, M_{rstr}\}_{IPk_{enc}}\} M_{rstr} = (\{k_t\}_{PKRP}, \{SAML(M_{tok},$

$\{M_{tok}\}^{-1}_{PK_{IP}})\}_{k_t}, n_{se}) M_{tok} = (IP, \{PSHA1(n_{ce}, n_{se})\}_{PK_{RP}}, claims_U,$

$$IPk_{sig} = PSHA1(k_1, n_3), IPk_{sec} = PSHA1(k_1, n_4)$$

(3) C->RP:
$\{\{k_2\}_{PK_{RP}}, n_5, n_6, n_7, M_{enctok}, \{M_{mac}, M_{proof}, M_{req}\}_{Ck_{enc2}}\} M_{enctok} == (\{I$

$$Ck_{sig2} = PSHA1(k_2, n_5),$$

$$k_{proof} = PSHA1(n_{ce}, n_{se}), Ck_{endorse} = PSHA1(k_{proof}, n_7)$$

(4) $\text{RP} -> \text{C} : \{n_8, n_9, \{\{M_{resp}\}_{RPk_{sig}}, M_{resp}\}_{RPk_{enc}}\} RPk_{enc} = PSHA1(k_2, n_5$

SECTION III.
Formal Verification of Infocard Protocol

In this paper, we use the formal tool-Automated validation of Internet security protocols and applications (AVISPA)[5], [6] to analyze InfoCard protocol.

1. Modeling Infocard Protocol and Security Goals

AVISPA is a formal tool for building and analyzing security protocols. AVISPA integrates four different formal backends: On-the-fly model-checker (OFMC), CL-based Attack searcher (CL-AtSe), SAT-based model-checker (SATMC) and Tree-automata-based protocol analyzer (TA4SP). These backends implement different automated reasoning techniques for formally analyzing protocol specification. To our knowledge, no other tool exhibits the same level of scope and robustness while being with the same performance and scalability.

AVISPA uses a special language called High level protocol specification language (HLPSL)[9], [10] to model the analyzed protocol. A HLPSL specification is composed of three parts: a list of definitions of basic roles (the principals of protocol), protocol session, Intruder capabilities and a list of declarations of goals.

(1) Modeling Infocard Protocol in HLPSL
Specification of Protocol Basic Roles
The specification of InfoCard protocol contains three basic roles: client, Identity provider (IP), and Relying party (RP). Every role owns a list of global variables, a list of local variables and a lot of transitions. Global variables represent the public and share initial knowledge. Messages are exchanged via channels, Sending channel (SND) and Receiving channel (RCV). dy, the channel of type, stands for a channel of Dolev-Yao[11] which is under complete control of intruder. The following is the HLPSL specification of client.

role client (C, IP, RP: agent,

Kip, Krp: *public_key*, Muser, Cardid: text,

SND, RCV: channel (dy),

PSHA1: *hash_func*) played_by C def=

local State: nat, $N1, N2, N3, N4, N5, N6, N7, N8, N9$, Nce, Nse: text,
$K1, Kt, K2$: symmetric_key,

$Ipksig,\ Ipkenc,\ Cksig,\ Ckenc,\ Cksig1,\ Ckenc1,\ Rpksig,\ Rpkenc : hash$

Kproof: text,

Ckendorse: hash(text. text), Mresp, Mreq: message,

SAMLtoken:agent. *{text}_public_key.text.agent.text.{agent.*
{text}_public_key.text.agent.text}_inv(public_key)

The transition section of basic roles describes the procedure of message reception and emission (behavior of principals). Every transition has the form "precondition =>action", which represents that when precondition is satisfied, action will be executed. For instance, the state transition of the client is as following: transition

0 State = 0 ∧RCV(start) = | >

State':= $2 \wedge N1' := new() \wedge N2' := new() \wedge K1' := new()$

$\wedge Cksig' := PSHA1(K1'.N1') \wedge Ckenc' := PSHA1$

$(K1'.N2') \wedge SND(\{K1'\}\_Kip.\ N1'.N2'.\ \{\{Muser.Ca$

$rdid.typesrp.RP.Nce'\}\_Cksig'$. Muser. Cardid. type-srp. RP.
$Nce'\} - Ckenc')$

2.State= $2 \wedge$ RCV$(N3'.N4'.\{\{\{Kt'\}_Krp.\{SAMLtoke-$

$n'\}\_Kt'.Nse'\}\_Ipksig'.\{Kt'\}\_Krp.\{SAMLtoken'\}\_$

$Kt'.Nse'\}\_Ipkenc') \wedge Ipksig' = PSHA1(K1.N3') \wedge$

$Ipkenc' = PSHA1(K1.N4') = | >$

State':== $4 \wedge K2' := new() \wedge N5' := new() \wedge N6' := new()$

$\wedge N7' := new() \wedge Cksig1' := PSHA1(K2'.N5')/\backslash Cke-$

$nc1' := PSHA1(K2'.N6') \wedge Kproof' := PSHA1(Nce.$

$Nse) \wedge Ckendorse' := PSHA1(Kproof'.N7') \wedge SND$

$(\{K2'\}\_Krp.\ N5'.N6'.N7'.\{Kt'\}\_Krp.\{SAMLtoken'\}$

$\_Kt'.\{Mreq.\{Mreq\}\_Cksig1'.\{\{Mreq\}\_Cksig1'\}\_Cke - ndorse'\} - Ck$

4.State=4

$\wedge RCV(N8'.N9'.\{\{Mresp\} - Rpksig'.\ Mre - sp\}\_Rpkenc') \wedge Rpksig' =$

State':= 6

end role

Protocol Session

A session is used for describing how to combine roles which is to run roles in parallel. The run of a basic role is a process. In the session of InfoCard protocol, the processes of three basic roles (client, IP, RP) run parallel which is described as following:

Role session $(C,\ IP,\ RP$: *agent, Kip, Krp: public_key, Muser, Cardid: text, PSHA1: hash_func) def=* local $SC,\ RC,\ SIP,\ RIP,\ SRP,\ RRP$: *channel (dy)* composition client
$(C,\ IP,\ RP,\ Kip,\ Krp,\ Muser,\ Cardid,\ SC,\ RC, PSHA1) \wedge$ ip$(C,\ IP$

$PSHA1) \wedge$ rp $(C,\ IP,\ RP,\ Kip,\ Krp,\ SRP,\ RRP,\ PSHA1)$ end role

Intruder Capabilities

Intruder capabilities are defined in the environment role. This role comprises global constant and a composition of session, where the intruder may play some roles as legal users. Composition session includes two parallel protocol running in which the intruder is participating. The definition of environment role is as follows:

role environment()def= const

client, ip, rp: agent,

$krip,\ kip,\ ki$: *public_key*, typesrp, muser, cardid, claimsu, ppid: text,

samltoken: *protocol_id*, pshal: *hash_func*
$intruder\_knowledge = \{client,\ ip,\ rp,\ krip,\ kip,\ ki,\ inv(ki), psha1\}$

composition

session $(client,\ ip,\ rp,\ kip,\ krp,\ muser,\ cardid,\ pshal)$

$\wedge$session $client,\ ip,\ rp,\ kip,\ krp,\ muser,\ cardid,\ psha1)$

end role

(2) Modeling Security Goals in HLPSL

Authentication and secrecy are two basic security goals of InfoCard protocol. We model them as following:

request (IP, C, muser, Muser) witness(C, IP, muser, Muser)

request (IP, C, cardid, Cardid) witness(C, IP, cardid, Cardid)

request (RP, C, samltoken, SAMLtoken')

witness(C, RP, samltoken, SAMLtoken')

secret (SAMLtoken, samltoken, {IP, C}) $request(IP, C, muser, Muser)$
means that IP receives

the message $Muser$) from the $client(C)$

the message M (user) from the client(C).

$witness(C, IP, muser, Muser)$ means that the client sends the message ($Muser$)) to IP. The paired predicates *request and witness*) describe strong authentication[4] properties in HLPSL.

$secret(SAMLtoken, samltoken, \{IP, C\})$ represents the main secrecy goal that SAML type token should be confidential between the IP and the client. An attacker can not know the token.

2. Security Verification and Verification Results

Given a HLPSL specification, the next step is to translate it into a lower level specification. This is automatically done by the translator hlpsl2if, generating a specification in an intermediate format (IF). In the end, IF specification is verified by the underlying automated reasoning module. During the verification process of InfoCard protocol, we mainly used the OFMC (On-the-fly model-checker)[12], which is the most important and powerful reasoning module of AVISPA.

We have found three types of attacks: session replay attack, man-in-the-middle attack and token replay attack. The man-in-the-middle attack and token replay attack are found based on the assumption that two optional elements, token scope and proof key, lack in InfoCard protocol implementation.

(1) Session Replay Attack

This attack is found by authentication goal. An attacker can intercept the old message from a client to IP and replay it to IP in a new session. The attack is showed as follows. i-> (client, 3): start (client, 3)-> i:

$\{K1(1)\} - kip.N1(1).N2(1).\{\{muser.cardid.typesrp.rp.Nce(1)\}\_(psh$

$i- > (\text{ip}, 3)$:

$\{K1(1)\} - kip.N1(1).N2(1).\{\{muser.cardid.typesrp.rp.Nce(1)\}\_(psh$

$(\text{ip}, 3)- > i : N3(2).N4(2).\{\{\{Kt(2)\}\_krp.\{ip.\{psha1(Nce(1).Nse(2))$

$(2))) \{Kt \quad (2)\}\_krp.\{ip. \{psha1(Nce(1).Nse(2))\}\_krp.clai-$

$msu.rp.ppid.\{ip.\{psha1(Nce(1).Nse(2))\}\_krp.claimsu.rp.$

$ppid\}\_inv(kip)\}\_Kt(2).Nse(2)\}\_(psha1(K1(1).N4(2)))$

$i- > (\text{ip}, 7) : \{K1(1)\}\_kip.N1(1).N2(1).\{\{muser.cardid.tvpesrp.rv.N$

$(\text{ip}, 7)- > i$:

$N3(3).N4(3).\{\{\{Kt(3)\}\_krp.\{ip.\{psha1(Nce(1).Nse(3))\}\_krp.claimsu$

$(3).Nse(3)\}\_(psha1(K1(1).N4(3)))$

Where i represents an intruder. "(client, 3) - <i, i-> (ip, 3) and (ip, 3) -> i" is a round execution of protocol. The intruder is a man in middle, who can intercept the all of message from client to identity provider. "i-> (ip, 7) and (ip, 7) - >i"; is another round execution, in which the intruder replay the old message intercepted from the former round.

Although the intruder can not decrypt the token issued by IP in the above attack, the attack has breached the authentication property. It also induces DOS attack if the intruder replays continuously the old authentication request message to IP.

(2) Man-in-the-Middle Attack

In the specification of OASIS identity metasystem interoperability, two optional elements, token scope and proof key, are extremely critical. We model and analyze the revised InfoCard protocol where these two optional elements lack. We found that InfoCard protocol is susceptible to man-in-the-middle attack if the element of token scope lacks in the implementation of InfoCard protocol. We also found the attack of replaying the token without the optional element of proof key.

Token Scope element which represents the identity of RP in the request token message is optional. An Identity Selector, by default, should not convey information about the relying party where an issued token will be used (i.e., target scope) when requesting security tokens. This helps safeguard user privacy. However, if token scope lacks in request-token message, our analysis can find a man-in-the-middle attack that allows a dishonest replying party (an intruder) to impersonate a user at another replying party. The attack is shown as Fig. 2.

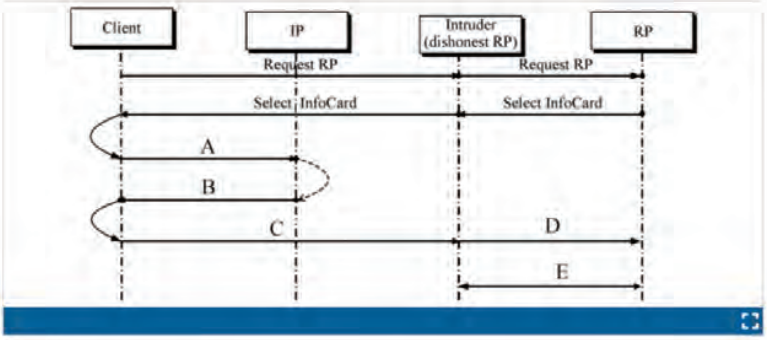


Fig. 2.

Man-in-the-middle attack.

$A - \{\{k_1\}_{PKIP}, n_1, n_2, \{\{M_{user}, M_{rst}\}_{Cksig}, M_{user}, M_{rst}\}_{Ckenc}\}$; $B - \{n_3, n_4,$

Show All

In Fig. 2, *Pki* represents the public key of the intruder. *M_tok* does not contain the identifier of RP. Hence when the intruder receives the token, he can forward directly the signed token and forge other items of the message to another RP. As a result, intruder impersonates a legal client at another RP to get the resource of RP. If token scope is included in issued token, the dishonest RP can not impersonate a legal user at another RP, because the dishonest RP can not tamper the identifier of RP contained in the signed token by private key of IP. Therefore, the dishonest RP can not abuse the token issued by IP at another RP.

(3) Token Replay Attack

Proof key is a subject confirmation key. In identity meta-system interoperability version 1.0, proof key is an optional element, but it is very important. Proof key is a credential between client and IP. Meanwhile it is included in the encyphered token. In the third step of InfoCard protocol, a client uses the proof key to produce the endorse key which is used to sign the MAC of request message. Thus, RP authenticates client based on the token and the proof key. Although an attacker can get the signed token, he can not tamper the signed token. It means that he can not tamper the proof key included in signed token. Therefore the attacker can not impersonate a legal user without the proof key. Consequently, proof keys provide stronger authentication guarantee.

If the proof key lack in InfoCard protocol, an intruder who have stolen the token by some attack methods, such as DNS dynamic pharming attack[18], can impersonate legal user and replay the signed token to legal RP. If we add the token to the knowledge of intruder and delete proof key of exchanged messages in the formal

model of InfoCard protocol, our analysis can find an attack to authentication goal as follows where an intruder, represented as i, can pass the authentication to RP by the stolen token. $i > (\mathrm{rp}, 6)$:

$$\{K2(1)\}\_Krp.\ N5(1).\ N6(1).\ N7(1).\ \{Kt(1)\}\_Krp.\ \{IP.$$

$$claimsu.\ RP.\ ppid.\ \{IP.\ claimsu.\ RP.\ ppid\}\_inv(Kip)\}\_Kt(1).\ \{$$

$$Mreq.\ \{Mreq\}\_psha1(K2(1).\ N5(1))\} - psha1(K2(1).\ N6(1))(\mathrm{rp}, 6) > i:$$

$$N8(2).\ N9(2).\ \{\{Mresp\} - psha1(K2(1).\ N8(2)).\ Mresp\}\_psha1(K2(1).$$

SECTION IV.
Related Work

Gajek *et al.*[13] analyzed the attacks to InfoCard based on weak security policy enforcements in commodity browsers. Hoang *et al.*[14] discussed the problem InfoCard secure Roaming across different terminals. Alrodhan and Mitchell[15] analyzed its privacy property of card space and suggested a zero-knowledge cryptographic technique to improve privacy protection. Ahn *et al.*[16] proposed a user preference expression language that is crucial to manage user's privacy in federated identity management. Ahn *et al.*[17] provided category-based privacy preference approach to enhance the privacy of user-centric identity management systems.

The work most closely related to ours is Ref.[18]. In this paper, Bhargavan *et al.* formally verified a reference implementation of InfoCard with ProVerif, a formal tool based on pi calculus. They found a man-in-the-middle attack where attacks get the unencyphered signature and replace it with own signature. Their analysis is based on the assumption configuration that signed messages were not encyphered. Without the assumption, their method can not find any attack. However in most cases, signed messages should be encrypted when protocols are designed, because attacks can get the unencyphered signature and replace it with own signature. In our analysis, the configuration of InfoCard protocol is encryption after signature. Therefore, the new attacks found in the paper are different from Ref. [18].

SECTION V.
Conclusion

In this paper, we have successfully analyzed the authentication and secrecy properties of Infocard protocol using a formal analysis tool, AVISPA. Our analysis showed that InfoCard protocol is easily subjected to a session replay attack because this protocol is lack of freshness numbers. In addition, we analyze the importance of token scope and proof key, the two main optional elements in the specification of OASIS Identity Metasystem Interoperability. Our analysis indicated that a man-in-the-middle attack where a dishonest RP can impersonate a legal user with acquired token to pass authentication of another RP, and token replay attack where a intruder can replay the stolen security token, are possible if these two optional elements lack in InfoCard protocol. We have implemented *an* InfoCard identity selector using Java on android platforms. Meanwhile, we are implementing an entire InfoCard system. In the future, we would like to verify the privacy properties for federated identity-management protocols based on our implementation using formal methods.

Chinese Journal of Electronics
Vol.22, No.1, Jan. 2013

Formal Analysis of Information Card Federated Identity-Management Protocol*

WANG Juan[1,2], HU Hongxin[3], ZHAO Bo[1,2], YAN Fei[1,2], ZHANG Huanguo[1,2] and WU Qianhong[1,2]

(1.*Computer School, Wuhan University, Wuhan 430072, China*)

(2.*Key Laboratory of Aerospace Information Security and Trusted Computing Ministry of Education, Wuhan 430072, China*)

(3.*SEFCOM, Arizona State University, AZ 85287, USA*)

Abstract — Information Card (InfoCard) is a user-centric identity management metasystem. It has been accepted as a standard of OASIS Identity Metasystem Interoperability Technical Committee. However, there is currently a lack of security analysis to InfoCard protocol, especially, with formal methods. In this paper, we accommodate such a requirement by analyzing security properties of InfoCard protocol adopting a formal protocol analysis tool. Our analysis result discovers that current InfoCard protocol is vulnerable against the session replay attack. Furthermore, we reveal the importance of two optional elements in InfoCard metasystem, token scope and proof key, and found that InfoCard protocol will be susceptible to man-in-the-middle attack and token replay attack if these two optional elements lack.

Key words — Information card, User-centric, Identity, Automated validation of Internet security protocols and applications (AVISPA).

I. Introduction

In open and collaborative Internet, managing the necessary digital identities is a complex hurdle. Traditionally, Service providers collect and manage user identity related information which has led to proven attacks like identity theft and privacy violation. Recently, user-centric identity management approaches have emerged. User-centric identity management allows users to control their own digital identities. Users can select their credentials when responding to an authenticator or attribute requester, giving users more rights and responsibility over their identity information. Information Card (InfoCard)[1−3] is a user centric federated identity management metasystem for reducing the reliance on passwords and improving the privacy of personal information.

InfoCard protocol is as the foundation of an InfoCard system implementation and its security should be guaranteed. However, the design of security protocols is highly error-prone. Security flaws have been found in many protocols[4]. For InfoCard protocol, security issues have not yet been explored in depth, especially using formal methods. In this paper, we focus on the formal analysis of InfoCard protocol. We first provide an informal description about the detailed communication proceeds and message format of InfoCard protocol and thoroughly analyze the implementation mechanism of security goals. Then, we translate the informal description of InfoCard protocol and its security goals to a formal specification based on High level protocol specification language (HLPSL). Finally, we verify the protocol using an automatic formal analysis tool, Automated validation of Internet security protocols and applications (AVISPA)[5,6].

Our contribution is three-fold. First, we model and analyze the authentication and secrecy goals of InfoCard protocol using a formal method. Second, we uncover a session replay attack. Finally, we reveal the importance of two main optional elements, token scope and proof key, in an InfoCard metasystem. We found that InfoCard protocol is susceptible to man-in-the-middle attack and token replay attack if these two optional elements lack in the implementation of InfoCard protocol. To the best of our knowledge, this is the first time that above attacks to Infocard protocol are identified.

The remainder of this paper is organized as follows. In Section II, we describe the detailed communication proceeds and message format of InfoCard protocol and analyze the implementation mechanism of security goals. In Section III, we analyze and verify InfoCard protocol by a formal analysis tool, AVISPA, and discuss the importance of two main optional elements, token scope and proof key. Section IV discusses the related work. Finally, Section V provides conclusions.

II. Infocard Protocol

In accordance with the specification of OASIS Identity Metasystem Interoperability standard, the InfoCard protocol involves five components: User (U), Identity selector (IS), Client application (A), Relying party (RP) and Identity provider (IP).

The interaction process of the InfoCard system is depicted in Fig.1.

(1) A user accesses a relying party (a website) which allows

*Manuscript Received Oct. 2011; Accepted Apr. 2012. This work is supported by the National Natural Science Foundation of China (No.61003268, No.61173138) the Fundamental Research Funds for the Central Universities (No.211274629).

the user to log in the relying party with InfoCard.

(2) The relying party returns the requested page and sends the relying party's policy.

(3) The client application calls the identity selector and forwards the relying party's policy.

(4) In accordance with the relying party's policy, the identity selector searches in the user's information cards collection and finds the information cards which satisfy the requested claims in the relying party's policy. The user then selects one of these cards to log on the relying party.

(5) The identity selector sends a token request to the corresponding identity provider who issued the selected card.

(6) The identity provider authenticates the user by the provided authentication credential. If the user has passed the authentication, the identity provider issues a security token to the identity selector.

(7) The identity selector sends the service request message and the security token to the relying party.

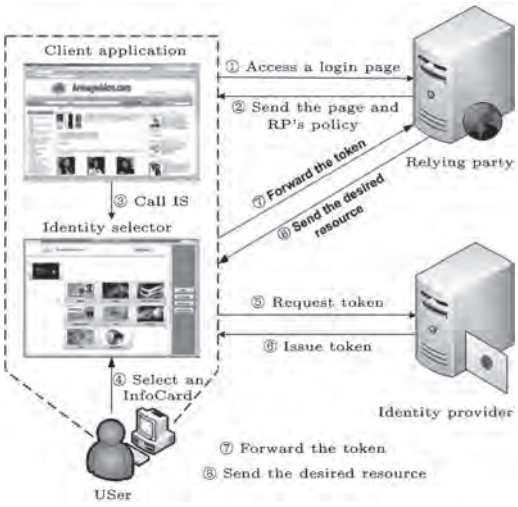


Fig. 1. Framework of InfoCard

(8) The relying party successfully authenticates the user and sends desired service or resource to the client.

Although the InfoCard system includes five components and eight operating steps, but in fact it is a typical three party authentication protocol. We can combine the user, identity selector and client application as "Client", and neglect the steps used to retrieve the policy from the point view of security protocol analysis, just considering the steps using the mutual authentication process. Moreover, it must be explained that we just consider the managed card protocol in this paper because the selfissued card protocol does not have a separate identity provider and can be regarded as a special case of the managed card protocol.

On the basis of above considerations, we can give the main authentication procedure of managed card protocol. The main authentication procedure and message format are according to Microsoft's implementation of InforCard protocol[7,8] and the OASIS identity metasystem specification[1].

The elements of n_1 to n_9, n_{ce} and n_{se} are random numbers. k_1, k_2 and k_t are symmetric keys. PK_{IP} represents the public key of IP and PK_{IP}^{-1} represents the private key of IP. Ck_{sig} is the signature key of client. Ck_{enc} is the encryption key and $C_{kendorse}$ is the endorsing signature key. RPk_{sig} is the signature key of RP and RPk_{enc} is the encryption key. M_{rst} indicates request message and M_{resp} indicates response message. M_{tok} indicates the issued token by identity provider. SAML is the type of issued token. M_{enctok} represents the encyphered token. $cardId$ is the unique identity of a card. $types_{RP}$ stands for the types of the required claims. $claims_U$ represents claims with the attribute values for the required $types_{RP}$. M_{mac} is the message signature. M_{proof} is the endorsing signing message proving possession of proof key.

The detailed communication proceeds and message format of managed card protocol are described as following:

(1) C->IP: $\{\{k_1\}_{PK_{IP}}, n_1, n_2, \{\{M_{user}, M_{rst}\}_{Ck_{sig}}, M_{user}, M_{rst}\}_{Ck_{enc}}\}$

$M_{user} = (U, pwd)$,

$M_{rst} = (cardId, types_{RP}, RP, n_{ce})$

$Ck_{sig} = \text{PSHA1}(k_1, n_1)$,

$Ck_{enc} = \text{PSHA1}(k_1, n_2)$

(2) IP->C: $\{n_3, n_4, \{\{M_{rstr}\}_{IPk_{sig}}, M_{rstr}\}_{IPk_{enc}}\}$

$M_{rstr} = (\{k_t\}_{PKRP}, \{SAML(M_{tok}, \{M_{tok}\}^{-1}_{PK_{IP}})\}_{k_t}, n_{se})$

$M_{tok} = (IP, \{PSHA1(n_{ce}, n_{se})\}_{PK_{RP}}, claims_U, RP, ppid)$

$IPk_{sig} = PSHA1(k_1, n_3)$,

$IPk_{enc} = PSHA1(k_1, n_4)$

(3) C->RP: $\{\{k_2\}_{PK_{RP}}, n_5, n_6, n_7, M_{enctok}, \{M_{mac}, M_{proof}, M_{req}\}_{Ck_{enc2}}\}$

$M_{enctok} == (\{k_t\}_{PK_{RP}}, \{SAML(M_{tok}, \{M_{tok}\}^{-1}_{PK_{IP}})\}_{k_t})$,

$M_{mac} = \{M_{req}\}_{Ck_{sig2}}$,

$M_{proof} = \{M_{mac}\}_{Ck_{endorse}}$,

$Ck_{enc2} = PSHA1(k_2, n_6)$,

$Ck_{sig2} = PSHA1(k_2, n_5)$,

$k_{proof} = PSHA1(n_{ce}, n_{se})$,

$Ck_{endorse} = PSHA1(k_{proof}, n_7)$

(4) RP->C: $\{n_8, n_9, \{\{M_{resp}\}_{RPk_{sig}}, M_{resp}\}_{RPk_{enc}}\}$

$RPk_{enc} = PSHA1(k_2, n_9)$,

$RPk_{sig} = PSHA1(k_2, n_8)$

III. Formal Verification of Infocard Protocol

In this paper, we use the formal tool-Automated validation of Internet security protocols and applications (AVISPA)[5,6] to analyze InfoCard protocol.

1. Modeling InfoCard protocol and security goals

AVISPA is a formal tool for building and analyzing security protocols. AVISPA integrates four different formal backends: On-the-fly model-checker (OFMC), CL-based Attack searcher (CL-AtSe), SAT-based model-checker (SATMC) and Tree-automata-based protocol analyzer (TA4SP). These backends implement different automated reasoning techniques for

formally analyzing protocol specification. To our knowledge, no other tool exhibits the same level of scope and robustness while being with the same performance and scalability.

AVISPA uses a special language called High level protocol specification language (HLPSL)[9,10] to model the analyzed protocol. A HLPSL specification is composed of three parts: a list of definitions of basic roles (the principals of protocol), protocol session, Intruder capabilities and a list of declarations of goals.

(1) Modeling InfoCard protocol in HLPSL

Specification of protocol basic roles. The specification of InfoCard protocol contains three basic roles: client, Identity provider (IP), and Relying party (RP). Every role owns a list of global variables, a list of local variables and a lot of transitions. Global variables represent the public and share initial knowledge. Messages are exchanged via channels, Sending channel (SND) and Receiving channel (RCV). dy, the channel of type, stands for a channel of Dolev-Yao[11] which is under complete control of intruder. The following is the HLPSL specification of client.

role client (C, IP, RP : agent,
Kip, Krp : *public_key*,
Muser, Cardid : text,
SND, RCV : channel (dy),
PSHA1 : *hash_func*)
played_by C def=
local State : nat,
$N1, N2, N3, N4, N5, N6, N7, N8, N9$, Nce, Nse : text,
$K1, Kt, K2$: symmetric_key,
Ipksig, *Ipkenc*, *Cksig*, *Ckenc*, *Cksig*1, *Ckenc*1, *Rpksig*, *Rpkenc* :
hash(*symmetric_key.text*),
Kproof : text,
Ckendorse : hash(text. text),
Mresp, Mreq: message,
SAMLtoken:agent.{*text*}_*public_key.text.agent.text.*{*agent.* {*text*}_*public_key.text.agent.text*}_*inv*(*public_key*)

The transition section of basic roles describes the procedure of message reception and emission (behavior of principals). Every transition has the form "precondition =>action", which represents that when precondition is satisfied, action will be executed. For instance, the state transition of the client is as following:

transition
0. State = 0 ∧ RCV(start) = | >
State':=2 ∧ $N1'$:=*new*() ∧ $N2'$:=*new*()∧ $K1'$:= *new*()
∧ *Cksig*′:=$PSHA1(K1'.N1')$∧ *Ckenc*′ :=$PSHA1$
$(K1'.N2')$∧ $SND(\{K1'\}\_Kip.\ N1'.N2'.\ \{\{Muser.Cardid.\ typesrp.RP.Nce'\}\_Cksig'.$ Muser. Cardid. typesrp. RP. $Nce'\}\_Ckenc')$
2.State=2∧RCV($N3'.N4'.\{\{\{Kt'\}\_Krp.\{SAMLtoken'\}\_\ Kt'.Nse'\}\_Ipksig'.\{Kt'\}\_Krp.\{SAMLtoken'\}\_\ Kt'.Nse'\}\ \_Ipkenc'$) ∧ $Ipksig' = PSHA1(K1.N3')$∧
$Ipkenc' = PSHA1(K1.N4')$ = | >
State':=4∧$K2'$:= *new*()∧$N5'$:= *new*()∧$N6'$:= *new*()
∧$N7'$:= *new*() ∧*Cksig*1′ := $PSHA1(K2'.N5')/\backslash$ *Ckenc*1′ := $PSHA1\ (K2'.N6')$ ∧*Kproof*′ := $PSHA1(Nce.Nse)$∧ *Ckendorse*′ := $PSHA1\ (Kproof'.N7')$ ∧*SND*
$(\{K2'\}\_Krp.\ N5'.N6'.N7'.\{Kt'\}\_Krp.\ \{SAMLtoken'\}\_Kt'.\{Mreq.\{Mreq\}\_Cksig1'.\{\{Mreq\}\_Cksig1'\}\_Ckendorse'\}\ \_Ckenc1')$
4.State=4 ∧ $RCV(N8'.N9'.\{\{Mresp\}\ \_Rpksig'.\ Mresp\}\_Rpkenc')$∧
$Rpksig' = PSHA1(K2.N8')$ ∧ $Rpkenc' = PSHA1$
$(K2.N9')$ = | >
State':= 6
end role

Protocol session. A session is used for describing how to combine roles which is to run roles in parallel. The run of a basic role is a process. In the session of InfoCard protocol, the processes of three basic roles (client, IP, RP) run parallel which is described as following:

Role session(*C, IP, RP: agent, Kip, Krp: public_key, Muser, Cardid: text, PSHA1*: *hash_func*) *def*=
local *SC, RC, SIP, RIP, SRP, RRP: channel* (*dy*)
composition
client(*C, IP, RP, Kip, Krp, Muser, Cardid, SC, RC, PSHA*1)
∧ ip(*C, IP, RP, Kip, Krp, Muser, Cardid, SIP, RIP, PSHA*1)
∧ rp(*C, IP, RP, Kip, Krp, SRP, RRP, PSHA*1)
end role

Intruder capabilities. Intruder capabilities are defined in the environment role. This role comprises global constant and a composition of session, where the intruder may play some roles as legal users. Composition session includes two parallel protocol running in which the intruder is participating. The definition of environment role is as follows:

role environment() def=
const
client, *ip, rp* : agent,
krp, kip, ki : *public_key*,
typesrp, muser, cardid, claimsu, ppid : text,
samltoken :*protocol_id*,
psha1 :*hash_func*

intruder_knowledge = {*client, ip, rp, krp, kip, ki, inv*(*ki*), *psha*1}
composition
session(*client, ip, rp, kip, krp, muser, cardid, psha*1)
∧ session(*client, ip, rp, kip, krp, muser, cardid, psha*1)
end role

(2) Modeling security goals in HLPSL

Authentication and secrecy are two basic security goals of InfoCard protocol. We model them as following:

request (IP, C, muser, Muser) witness(C, IP, muser, Muser)

request (IP, C, cardid, Cardid) witness(C, IP, cardid, Cardid)

request(RP,C,samltoken,SAMLtoken')

witness(C,RP,samltoken,SAMLtoken')

secret (SAMLtoken, samltoken, {IP, C})

request($IP, C, muser, Muser$) means that IP receives the message($Muser$) from the *client*(C). *witness*($C, IP, muser, Muser$) means that the client sends the message ($Muser$) to IP. The paired predicates (*request and witness*) describe strong authentication[4] properties in HLPSL.

$secret(SAMLtoken, samltoken, \{IP, C\})$ represents the main secrecy goal that SAML type token should be confidential between the IP and the client. An attacker can not know the token.

2. Security verification and verification results

Given a HLPSL specification, the next step is to translate it into a lower level specification. This is automatically done by the translator hlpsl2if, generating a specification in an intermediate format (IF). In the end, IF specification is verified by the underlying automated reasoning module. During the verification process of InfoCard protocol, we mainly used the OFMC (On-the-fly model-checker)[12], which is the most important and powerful reasoning module of AVISPA.

We have found three types of attacks: session replay attack, man-in-the-middle attack and token replay attack. The man-in-the-middle attack and token replay attack are found based on the assumption that two optional elements, token scope and proof key, lack in InfoCard protocol implementation.

(1) Session replay attack

This attack is found by authentication goal. An attacker can intercept the old message from a client to IP and replay it to IP in a new session. The attack is showed as follows.

i -> (client, 3): start

(client,3)-> i:

$\{K1(1)\}\_kip.N1(1).N2(1).\{\{muser.cardid.typesrp.rp.Nce(1)\}\_(psha1(K1(1).N1(1))).muser.cardid.typesrp.rp.Nce(1)\}\_(psha1(K1(1).N2(1)))$

i->(ip,3):

$\{K1(1)\}\_kip.N1(1).N2(1).\{\{muser.cardid.typesrp.rp.Nce(1)\}\_(psha1(K1(1).N1(1))).muser.cardid.typesrp.rp.Nce(1)\}\_(psha1(K1(1).N2(1)))$

(ip,3)-> i:

$N3(2).N4(2).\{\{\{Kt(2)\}\_krp.\{ip.\{psha1(Nce(1).Nse(2))\}\_krp.claimsu.rp.ppid.\{ip.\{psha1(Nce(1).Nse(2))\}\_krp.claimsu.rp.ppid\}\_inv(kip)\}\_Kt(2).Nse(2)\}\_(psha1(K1(1).N3(2))).\{Kt(2)\}\_krp.\{ip.\{psha1(Nce(1).Nse(2))\}\_krp.claimsu.rp.ppid.\{ip.\{psha1(Nce(1).Nse(2))\}\_krp.claimsu.rp.ppid\}\_inv(kip)\}\_Kt(2).Nse(2)\}\_(psha1(K1(1).N4(2)))$

i->(ip,7):

$\{K1(1)\}\_kip.N1(1).N2(1).\{\{muser.cardid.typesrp.rp.Nce(1)\}\_(psha1(K1(1).N1(1))).muser.cardid.typesrp.rp.Nce(1)\}\_(psha1(K1(1).N2(1)))$

(ip,7)-> i:

$N3(3).N4(3).\{\{\{Kt(3)\}\_krp.\{ip.\{psha1(Nce(1).Nse(3))\}\_krp.claimsu.rp.ppid.\{ip.\{psha1(Nce(1).Nse(3))\}\_krp.claimsu.rp.ppid\}\_inv(kip)\}\_Kt(3).Nse(3)\}\_(psha1(K1(1).N3(3))).\{Kt(3)\}\_krp.\{ip.\{psha1(Nce(1).Nse(3))\}\_krp.claimsu.rp.ppid.\{ip.\{psha1(Nce(1).Nse(3))\}\_krp.claimsu.rp.ppid\}\_inv(kip)\}\_Kt(3).Nse(3)\}\_(psha1(K1(1).N4(3)))$

where i represents an intruder. "(client, 3) -> i, i-> (ip, 3) and (ip, 3) -> i" is a round execution of protocol. The intruder is a man in middle, who can intercept the all of message from client to identity provider. "i -> (ip, 7) and (ip, 7) -> i" is another round execution, in which the intruder replay the old message intercepted from the former round.

Although the intruder can not decrypt the token issued by IP in the above attack, the attack has breached the authentication property. It also induces DOS attack if the intruder replays continuously the old authentication request message to IP.

(2) Man-in-the-middle attack

In the specification of OASIS identity metasystem interoperability, two optional elements, token scope and proof key, are extremely critical. We model and analyze the revised InfoCard protocol where these two optional elements lack. We found that InfoCard protocol is susceptible to man-in-the-middle attack if the element of token scope lacks in the implementation of InfoCard protocol. We also found the attack of replaying the token without the optional element of proof key.

Token Scope element which represents the identity of RP in the request token message is optional. An Identity Selector, by default, should not convey information about the relying party where an issued token will be used (i.e., target scope) when requesting security tokens. This helps safeguard user privacy. However, if token scope lacks in request-token message, our analysis can find a man-in-the-middle attack that allows a dishonest replying party (an intruder) to impersonate a user at another replying party. The attack is shown as Fig.2.

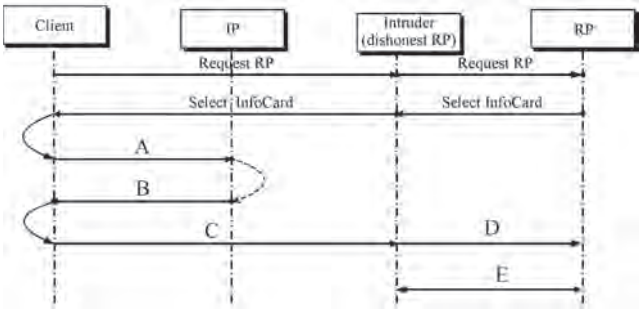


Fig. 2. Man-in-the-middle attack. A—$\{\{k_1\}_{PKIP}, n_1, n_2, \{\{M_{user}, M_{rst}\}_{Cksig}, M_{user}, M_{rst}\}_{Ckenc}\}$; B—$\{n_3, n_4, \{\{M_{rstr}\}_{IPksig}, M_{rstr}\}_{IPkenc}\}$; C—$\{\{k_2\}_{PKi}, n_5, n_6, n_7, M_{enctok}, \{M_{mac}, M_{proof}, M_{req}\}_{Ckenc2}\}$; D—$\{\{k_2\}_{PKRP}, n_5, n_6, n_7, M_{enctok}, \{M_{mac}, M_{proof}, M_{req}\}_{Ckenc2}\}$; E—$\{n_8, n_9, \{\{M_{resp}\}_{RPksig}, M_{resp}\}_{RPkenc}\}$

In Fig.2, Pki represents the public key of the intruder. M_tok does not contain the identifier of RP. Hence when the intruder receives the token, he can forward directly the signed token and forge other items of the message to another RP. As a result, intruder impersonates a legal client at another RP to get the resource of RP. If token scope is included in issued token, the dishonest RP can not impersonate a legal user at another RP, because the dishonest RP can not tamper the identifier of RP contained in the signed token by private key of IP. Therefore, the dishonest RP can not abuse the token issued by IP at another RP.

(3) Token replay attack

Proof key is a subject confirmation key. In identity metasystem interoperability version 1.0, proof key is an optional element, but it is very important. Proof key is a credential between client and IP. Meanwhile it is included in the encyphered token. In the third step of InfoCard protocol, a client uses the proof key to produce the endorse key which is used to sign the MAC of request message. Thus, RP authenticates client based on the token and the proof key. Although an attacker

can get the signed token, he can not tamper the signed token. It means that he can not tamper the proof key included in signed token. Therefore the attacker can not impersonate a legal user without the proof key. Consequently, proof keys provide stronger authentication guarantee.

If the proof key lack in InfoCard protocol, an intruder who have stolen the token by some attack methods, such as DNS dynamic pharming attack[18], can impersonate legal user and replay the signed token to legal RP. If we add the token to the knowledge of intruder and delete proof key of exchanged messages in the formal model of InfoCard protocol, our analysis can find an attack to authentication goal as follows where an intruder, represented as i, can pass the authentication to RP by the stolen token.

i >(rp,6):

$\{K2(1)\}\_Krp.N5(1).N6(1).N7(1).\{Kt(1)\}\_Krp.\{IP.claimsu.RP.ppid.\{IP.claimsu.RP.ppid\}\_inv(Kip)\}\_Kt(1).\{Mreq.\{Mreq\}\_psha1(K2(1).N5(1))\}\_psha1(K2(1).N6(1))$

(rp,6)> i:

$N8(2).N9(2).\{\{Mresp\}\_psha1(K2(1).N8(2)).Mresp\}\_psha1(K2(1).N9(2))$

IV. Related Work

Gajek *et al.*[13] analyzed the attacks to InfoCard based on weak security policy enforcements in commodity browsers. Hoang *et al.*[14] discussed the problem InfoCard secure Roaming across different terminals. Alrodhan and Mitchell[15] analyzed its privacy property of card space and suggested a zero-knowledge cryptographic technique to improve privacy protection. Ahn *et al.*[16] proposed a user preference expression language that is crucial to manage user's privacy in federated identity management. Ahn *et al.*[17] provided category-based privacy preference approach to enhance the privacy of user-centric identity management systems.

The work most closely related to ours is Ref.[18]. In this paper, Bhargavan *et al.* formally verified a reference implementation of InfoCard with ProVerif, a formal tool based on pi calculus. They found a man-in-the-middle attack where attacks get the unencyphered signature and replace it with own signature. Their analysis is based on the assumption configuration that signed messages were not encyphered. Without the assumption, their method can not find any attack. However in most cases, signed messages should be encrypted when protocols are designed, because attacks can get the unencyphered signature and replace it with own signature. In our analysis, the configuration of InfoCard protocol is encryption after signature. Therefore, the new attacks found in the paper are different from Ref.[18].

V. Conclusion

In this paper, we have successfully analyzed the authentication and secrecy properties of Infocard protocol using a formal analysis tool, AVISPA. Our analysis showed that InfoCard protocol is easily subjected to a session replay attack because this protocol is lack of freshness numbers. In addition, we analyze the importance of token scope and proof key, the two main optional elements in the specification of OASIS Identity Metasystem Interoperability. Our analysis indicated that a man-in-the-middle attack where a dishonest RP can impersonate a legal user with acquired token to pass authentication of another RP, and token replay attack where a intruder can replay the stolen security token, are possible if these two optional elements lack in InfoCard protocol. We have implemented an InfoCard identity selector using Java on android platforms. Meanwhile, we are implementing an entire InfoCard system. In the future, we would like to verify the privacy properties for federated identity-management protocols based on our implementation using formal methods.

References

[1] Identity Metasystem Interoperability Version 1.0. OASIS Standard. *http://docs. oasis-open. org/imi/identity/v1.0/ identity. html.*

[2] Higgins Open Source Identity Framework, *http: //www. eclipse. org/ higgins/.*

[3] Information Card Foundation, *http://informationcard. net/.*

[4] G. Lowe, "Breaking and fixing the Needham-Schroeder public-key protocol using FDR", *Software-Concept and Tools*, Vol.17, pp.93–102, 1996.

[5] AVISPA. The AVISPA User Manual. *http://avispa-project. org/publications.*

[6] A. Armando *et al.*, "The AVISPA tool for the automated validation of Internet security protocols and applications", *Proc. of the 17th International Conference on Computer Aided Verification* (*CAV'05*), Scotland, UK, Springer-Verlag. 2005.

[7] A. Nanda, "A technical reference for the information card profile V1.0", *Technical Report*, Microsoft Corporation, 2006.

[8] K. Bhargavan, C. Fournet, A.D. Gordon and N. Swamy, "Verified implementations of the information card federated identity-management protocol", *Proc. of ASIACCS '08*, Akihabara Convention Hall, Tokyo, ACM, pp.123–135, 2008.

[9] Y. Chevalier, L. Compagna *et al.*, "A high level protocol specification language for industrial security sensitive protocols", *Proc. of SAPS'04*, Austrian Computer Society, pp.1–13, 2004.

[10] Yannick Chevalier and Laurent Vigneron, "Rule-based programs describing Internet security protocols", *Electronic Notes in Theoretical Computer Science*, Vol.124, No.1, pp.113–132, 2005.

[11] Dolev, A. Yao, "On the security of public-key protocols", *IEEE Transactions on Information Theory*, Vol.29, No.2, pp.198–208, 1983.

[12] D. Basin, S. Mödersheim, L. Vigan‘o., "OFMC: A symbolic model-checker for security protocols", *International Journal of Information Security*, No.4, pp.181–208, 2005.

[13] Sebastian Gajek, Jörg Schwenk and Xuan Chen, "On the insecurity of microsoft's identity metasystem cardspace", *Technical Report*, Ruhr University, Bochum, Germany, 2008.

[14] L.N. Hoang, P. Laitinen and N. Asokan, "Secure roaming with identity metasystems", *Proc. of IDtrust '08*, New York, USA, ACM, pp.36–47, 2008.

[15] Waleed A. Alrodhan and Chris J. Mitchell, "Improving the security of cardspace", *EURASIP Journal on Information Security*, Vol.2009, Article ID 167216, pp.1–8, 2009.

[16] GailJoon Ahn, John Lam, "Managing privacy preferences for federated identity management", *Proc. of DIM'05*, Fairfax, Virginia, USA, pp.28–36, 2005.

[17] Gail-Joon Ahn, Moonam Ko, Mohamed Shehab, "Privacy-enhanced user-centric identity management", *Proc. of IEEE International Conference on Communication*, Dresden, Germany, pp.1–, 2009.

[18] C. Karlof, J. Tygar, D. Wagner, and U. Shankar, "Dynamic pharming attacks and locked same-origin policies for web

browsers", *Proc. of the 14th ACM Conference on Computer and Communications Security* (*CCS*), Virginia, USA, pp.58–71, 2007.



WANG Juan was born in China, in 1976. She is an associate professor of Computer School of Wuhan University and Key Laboratory of Aerospace Information Security and Trusted Computing Ministry of Education. She received the B.S., M.S. and Ph.D. degrees from Computer School of Wuhan University. From 2010 to 2011, as a visiting scholar, she did some research on information security in the Laboratory of Security Engineering for Future Computing at Arizona State University, USA. Her current research interests include identity authentication, security protocol, access control, trusted computing and cloud security. (Email: jwang@whu.edu.cn)



HU Hongxin was born in China, in 1974. He is currently a Ph.D. student in Computer Science and Engineering, Ira. A. Fulton School of Engineering at Arizona State University, USA. His current research interests include security and privacy in healthcare systems and social networks, network and system security, security in distributed, cloud and mobile computing, access control models and mechanisms.



ZHAO Bo was born in China, in 1972. He is a professor of Computer School of Wuhan University and Key Laboratory of Aerospace Information Security and Trusted Computing Ministry of Education. His current research interests include trusted computing and cloud security.



YAN Fei was born in China, in 1980. He is an associate professor of Computer School of Wuhan University and Key Laboratory of Aerospace Information Security and Trusted Computing Ministry of Education. His current research interests include trusted computing, access control and cloud security.

ZHANG Huanguo is a professor of Computer School of Wuhan University and Key Laboratory of Aerospace Information Security and Trusted Computing Ministry of Education. His current research interests include trusted computing and cloud security.

WU Qianhong is a professor of Computer School of Wuhan University and Key Laboratory of Aerospace Information Security and Trusted Computing Ministry of Education. His current research interests include cryptography and information security.

Appendix
OASIS07

OASIS Approves Interoperability Standard for Information Cards

News Submitted by carolgeyer on Thu, 07/16/2009 - 20:46.

OASIS members have approved Identity Metasystem Interoperability (IMI) version 1.0 as an OASIS Standard, a status that signifies the highest level of ratification. IMI is best known as the technical protocol that enables Information Cards, a new way for people to register, login, and share information with websites without needing a new username and password for each site.

IMI was advanced by representatives of IBM, Microsoft, CA, EMC, Novell, and others.

"Information Cards offer the best of both worlds—greater privacy and easier access," said Marc Goodner of Microsoft, chair of the OASIS IMI Technical Committee. "As an approved OASIS Standard, IMI assures interoperability across platforms and services, which will result in Information Card usage becoming even more widespread."

Information Cards are the digital equivalent of the cards people carry in their wallet or purse. They are stored in a new application called a card selector and carry information about a person's identity and relationship to the card provider, such as their employer, bank, library, store, etc.

"Most people will have a variety of Information Cards in their card selector just like they have in their wallet today. With one click they can select the one they want to use at a particular website —no more need to remember multiple usernames and passwords," said Drummond Reed, Executive Director of the Information Card Foundation and a member of the IMI Technical Committee. "They are also the first standard way to prove information about you (such as your age, voting district, employer, or credit rating) on-line. IMI 1.0 becoming an OASIS Standard is a big step forward towards an easier, safer, and more productive Internet."

"We are seeing an international convergence on issues of identity, privacy and security. The OECD pointed out in 2008, in Seoul, that we need better coordination on identity theft. The recent spate of cyber-attacks makes it clear that we need better systems in place--and that means the kind of crisp standards offered by IMI," added Laurent Liscia, OASIS executive director.

IMI is offered for implementation on a royalty-free basis. Participation in the OASIS IMI Technical Committee is open to all companies, non-profit groups, governments, academic institutions, and individuals. As with all OASIS projects, archives of the Committee's work are accessible to both members and non-members, and OASIS hosts an open mail list for public comment.

Support for IMI:

"The creation of the IMI standard is a significant development in the maturing and industry acceptance of the Identity Metasystem vision," said Kim Cameron, Microsoft's Chief Identity Architect. "The standard provides assurances that Information Card implementations will interoperate based on industry standards, enabling claims-based identity to be employed in real-world scenarios."

"CA is happy to have helped evolve the new standard for interoperable implementations of Information Cards for our customers," said Jeffrey Broberg, senior director at CA, Inc. "With IMI, it will be much easier for Internet sites and service providers to offer their customers easy-to-use, claims-based privacy and security that they require across a wide range of services and platforms."

URL:
http://www.oasis-open.org/committees/imi/

Login or register to post comments 8767 reads
News

© 1993-2023 OASIS. All rights reserved. | Terms of Use | Code of Conduct | Privacy Statement

Appendix
OASIS08


ITU-T Workshop on
Addressing
security
challenges
on a global scale
6 – 7 December 2010 Geneva, Switzerland
ITU

Security Standardization

- Oasis Standards Consortium

Speaker

- Anil Saldhana
  - Co-Chair, Oasis Identity In The Cloud TC
  - Member, Oasis IDTrust Steering Committee
    - *Member of Security Technical Committees spanning Federated Identity, Key Management, Health Care Privacy*
    - anil.saldhana@redhat.com

About Oasis

- "*Organization for the Advancement of Structured Information Standards*"
  - Founded in 1993 as SGML Open
  - Not for profit consortium
  - Global Representation
    - 5000+ Participants.
    - 600+ Organizations/Individual Members.
    - 100+ Countries.

Oasis Member Sections

- Member Sections include
  - IDTrust
  - Egov
  - Blue (SmartGrids/Intelligent Buildings)
  - WS-I etc.

Oasis IDTrust Member Section

- Promotes greater understanding and adoption of standards for
  - Identity and Trusted Infrastructure
    - Technologies, Policies and Practices

Oasis IDTrust Member Section

- Technical Committees
  - Identity Metasystems/Internet Identity
    - Identity In The Cloud (IDCloud)
    - Identity Metasystem Interoperability (IMI)
    - Extensible Resource Identifier(XRI)
    - XRI Data Interchange (XDI)

Oasis IDTrust Member Section

- Technical Committees
  - Encryption/Key Management
    - Key Mgmt. Interoperability Protocol (KMIP)
    - Enterprise Key Mgmt. Infrastructure (EKMI)

Oasis IDTrust Member Section

- Technical Committees
  - Trust/Privacy
    - Open Reputation Mgmt. Systems (ORMS)
    - Privacy Mgmt. Reference Model (PMRM)
    - Digital Signature Services Extended (DSS-X)
    - Cross Enterprise Security & Privacy Authorization (XSPA)

Oasis Security TC

- Technical Committees outside IDTrust
  - Extensible Access Control Markup Language (XACML)
  - Security Services (SAML)
  - Web Services Trust (WS-Trust)
  - Web Services Federation (WS-Fed)
  - Web Services Secure Exchange (WS-SX)

Oasis Benefits

- Open
  - Publicly archived Mailing Lists
  - Publicly reviewed specifications
  - International and neutral
  - Committees advance standards, adoption and interoperability
  - Specifications are free

References

- Resources
  - http://www.oasis-open.org
  - http://www.oasis-idtrust.org

Appendix
OASIS09

OASIS Identity Metasystem

Convener: John Bradley, D Reed, Mike Jones

Notes-taker: Mike

Discussion notes, key understandings, outstanding questions, observations, and, if appropriate to this discussion: action items, next steps:

This session discussed the goals and work plan of the OASIS Identity Metasystem Interoperability Technical Committee (IMI TC).

We shared that the committee had completed the first committee draft of the Identity Metasystem Interoperability specification and that it is available for public review. That spec consists of the merger of the content of the Identity Selector Interoperability Profile V1.5, A Guide to Using the Identity Selector Interoperability Profile V1.5 within Web Applications and Browsers, and Application Note: Web Services Addressing Endpoint References and Identity. The Security Considerations and Conformance sections of the committee draft are known to require more work on the part of the committee. This draft is available at:

- http://www.oasis-open.org/committees/download.php/29978/identity-1.0-spec-cd-01.doc
- http://www.oasis-open.org/committees/download.php/29979/identity-1.0-spec-cd-01.pdf

We also shared the output of this specification is intended to be backwards-compatible with these input documents.

We discussed the coordination with the OASIS Security Services TC to create a SAML 2.0 token profile for use with Information Cards, and that this will likely be the second work to come out of this TC. Also, we discussed that specifications for such innovations as Higgins Relationship Cards and the CardSpace "Geneva" CardTile are likely to be submitted by their inventors to the TC.

DeWitt Clinton led a discussion of the OASIS IPR Regime that the work is being conducted under. It was noted that the IPR Regime for the IMI TC work is the same as that for the underlying specifications, such as WS-Trust.

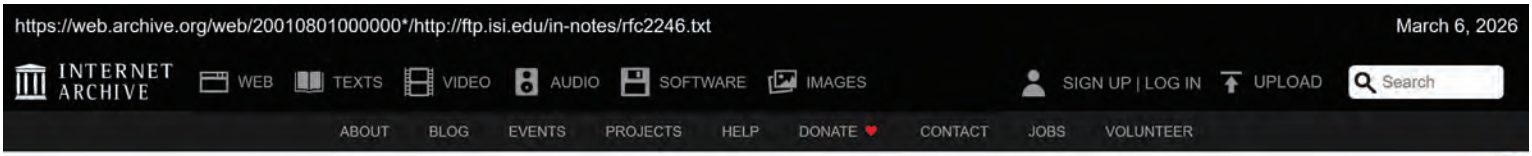
– Mike

This page was last edited on 1 December 2008, at 06:37. Content is available under Attribution-Share Alike 3.0 Unported unless otherwise noted. Privacy policy About IIW Disclaimers

MediaWiki spam blocked by CleanTalk

Appendix
TLS01

https://web.archive.org/web/20010801000000*/http://ftp.isi.edu/in-notes/rfc2246.txt March 6, 2026





DONATE

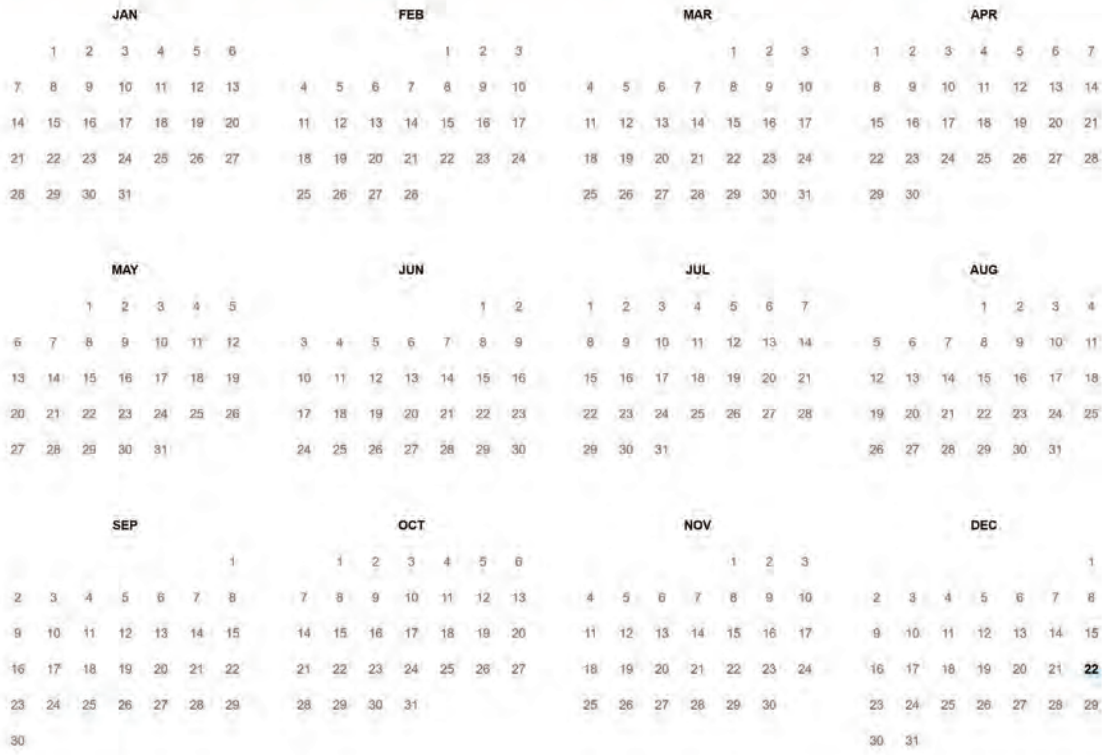
Explore more than 1 trillion web pages saved over time

http://ftp.isi.edu/in-notes/rfc2246.txt

Calendar · Collections · Changes · Summary · Site Map · URLs

Saved **7 times** between December 22, 2001 and October 22, 2023.





Note

This calendar view maps the number of times **http://ftp.isi.edu/in-notes/rfc2246.txt** was crawled by the Wayback Machine, *not* how many times the site was actually updated. More info in the FAQ.

FAQ | Contact Us | Terms of Service (Dec 31, 2014)

The Wayback Machine is an initiative of the Internet Archive, a 501(c)(3) non-profit building a digital library of Internet sites and other cultural artifacts in digital form. Other projects include Open Library & archive-it.org

Your use of the Wayback Machine is subject to the Internet Archive's Terms of Use.

https://web.archive.org/web/20011222210559/http://ftp.isi.edu/in-notes/rfc2246.txt March 6, 2026
INTERNET ARCHIVE Wayback Machine http://ftp.isi.edu/in-notes/rfc2246.txt Go NOV DEC AUG
7 captures 22 Dec 2001 - 22 Oct 2023 22 2000 2001 2005 About this capture

Network Working Group
Request for Comments: 2246
Category: Standards Track

T. Dierks
Certicom
C. Allen
Certicom
January 1999

The TLS Protocol
Version 1.0

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (1999). All Rights Reserved.

Abstract

This document specifies Version 1.0 of the Transport Layer Security (TLS) protocol. The TLS protocol provides communications privacy over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

Table of Contents

1. Introduction 3
2. Goals 4
3. Goals of this document 5
4. Presentation language 5
4.1. Basic block size 6
4.2. Miscellaneous 6
4.3. Vectors 6
4.4. Numbers 7
4.5. Enumerateds 7
4.6. Constructed types 8
4.6.1. Variants 9
4.7. Cryptographic attributes 10
4.8. Constants 11
5. HMAC and the pseudorandom function 11
6. The TLS Record Protocol 13
6.1. Connection states 14

6.2. Record layer 16
6.2.1. Fragmentation 16
6.2.2. Record compression and decompression 17
6.2.3. Record payload protection 18
6.2.3.1. Null or standard stream cipher 19
6.2.3.2. CBC block cipher 19
6.3. Key calculation 21
6.3.1. Export key generation example 22
7. The TLS Handshake Protocol 23
7.1. Change cipher spec protocol 24
7.2. Alert protocol 24
7.2.1. Closure alerts 25
7.2.2. Error alerts 26
7.3. Handshake Protocol overview 29
7.4. Handshake protocol 32
7.4.1. Hello messages 33
7.4.1.1. Hello request 33
7.4.1.2. Client hello 34
7.4.1.3. Server hello 36
7.4.2. Server certificate 37
7.4.3. Server key exchange message 39
7.4.4. Certificate request 41
7.4.5. Server hello done 42
7.4.6. Client certificate 43
7.4.7. Client key exchange message 43
7.4.7.1. RSA encrypted premaster secret message 44
7.4.7.2. Client Diffie-Hellman public value 45
7.4.8. Certificate verify 45
7.4.9. Finished 46
8. Cryptographic computations 47
8.1. Computing the master secret 47
8.1.1. RSA 48
8.1.2. Diffie-Hellman 48
9. Mandatory Cipher Suites 48
10. Application data protocol 48
A. Protocol constant values 49
A.1. Record layer 49
A.2. Change cipher specs message 50
A.3. Alert messages 50

A.4. Handshake protocol 51
A.4.1. Hello messages 51
A.4.2. Server authentication and key exchange messages 52
A.4.3. Client authentication and key exchange messages 53
A.4.4. Handshake finalization message 54
A.5. The CipherSuite 54
A.6. The Security Parameters 56
B. Glossary 57
C. CipherSuite definitions 61

D. Implementation Notes 64
D.1. Temporary RSA keys 64
D.2. Random Number Generation and Seeding 64
D.3. Certificates and authentication 65
D.4. CipherSuites 65
E. Backward Compatibility With SSL 66
E.1. Version 2 client hello 67
E.2. Avoiding man-in-the-middle version rollback 68
F. Security analysis 69
F.1. Handshake protocol 69
F.1.1. Authentication and key exchange 69
F.1.1.1. Anonymous key exchange 69
F.1.1.2. RSA key exchange and authentication 70
F.1.1.3. Diffie-Hellman key exchange with authentication 71
F.1.2. Version rollback attacks 71
F.1.3. Detecting attacks against the handshake protocol 72
F.1.4. Resuming sessions 72
F.1.5. MD5 and SHA 72
F.2. Protecting application data 72
F.3. Final notes 73
G. Patent Statement 74
Security Considerations 75
References 75
Credits 77
Comments 78
Full Copyright Statement 80

1. Introduction

The primary goal of the TLS Protocol is to provide privacy and data integrity between two communicating applications. The protocol is composed of two layers: the TLS Record Protocol and the TLS Handshake Protocol. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP[TCP]), is the TLS Record Protocol. The TLS Record Protocol provides connection security that has two basic properties:

- The connection is private. Symmetric cryptography is used for data encryption (e.g., DES [DES], RC4 [RC4], etc.) The keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by another protocol (such as the TLS Handshake Protocol). The Record Protocol can also be used without encryption.

- The connection is reliable. Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g., SHA, MD5, etc.) are used for MAC computations. The Record Protocol can operate without a MAC, but is generally only used in

this mode while another protocol is using the Record Protocol as a transport for negotiating security parameters.

The TLS Record Protocol is used for encapsulation of various higher level protocols. One such encapsulated protocol, the TLS Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. The TLS Handshake Protocol provides connection security that has three basic properties:

- The peer's identity can be authenticated using asymmetric, or public key, cryptography (e.g., RSA [RSA], DSS [DSS], etc.). This authentication can be made optional, but is generally required for at least one of the peers.

- The negotiation of a shared secret is secure: the negotiated secret is unavailable to eavesdroppers, and for any authenticated connection the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection.

- The negotiation is reliable: no attacker can modify the negotiation communication without being detected by the parties to the communication.

One advantage of TLS is that it is application protocol independent. Higher level protocols can layer on top of the TLS Protocol transparently. The TLS standard, however, does not specify how protocols add security with TLS; the decisions on how to initiate TLS handshaking and how to interpret the authentication certificates exchanged are left up to the judgment of the designers and implementors of protocols which run on top of TLS.

2. Goals

The goals of TLS Protocol, in order of their priority, are:

1. Cryptographic security: TLS should be used to establish a secure connection between two parties.

2. Interoperability: Independent programmers should be able to develop applications utilizing TLS that will then be able to successfully exchange cryptographic parameters without knowledge of one another's code.

3. Extensibility: TLS seeks to provide a framework into which new public key and bulk encryption methods can be incorporated as necessary. This will also accomplish two sub-goals: to prevent

the need to create a new protocol (and risking the introduction of possible new weaknesses) and to avoid the need to implement an entire new security library.

4. Relative efficiency: Cryptographic operations tend to be highly CPU intensive, particularly public key operations. For this reason, the TLS protocol has incorporated an optional session caching scheme to reduce the number of connections that need to be established from scratch. Additionally, care has been taken to reduce network activity.

3. Goals of this document

This document and the TLS protocol itself are based on the SSL 3.0 Protocol Specification as published by Netscape. The differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough that TLS 1.0 and SSL 3.0 do not interoperate (although TLS 1.0 does incorporate a mechanism by which a TLS implementation can back down to SSL 3.0). This document is intended primarily for readers who will be implementing the protocol and those doing cryptographic analysis of it. The specification has been written with this in mind, and it is intended to reflect the needs of those two groups. For that reason, many of the algorithm-dependent data structures and rules are included in the body of the text (as opposed to in an appendix), providing easier access to them.

This document is not intended to supply any details of service definition nor interface definition, although it does cover select areas of policy as they are required for the maintenance of solid security.

4. Presentation language

This document deals with the formatting of data in an external representation. The following very basic and somewhat casually defined presentation syntax will be used. The syntax draws from several sources in its structure. Although it resembles the programming language "C" in its syntax and XDR [XDR] in both its syntax and intent, it would be risky to draw too many parallels. The purpose of this presentation language is to document TLS only, not to have general application beyond that particular goal.

4.1. Basic block size

The representation of all data items is explicitly specified. The basic data block size is one byte (i.e. 8 bits). Multiple byte data items are concatenations of bytes, from left to right, from top to bottom. From the bytestream a multi-byte item (a numeric in the example) is formed (using C notation) by:

```
value = (byte[0] << 8*(n-1)) | (byte[1] << 8*(n-2)) |
        ... | byte[n-1];
```

This byte ordering for multi-byte values is the commonplace network byte order or big endian format.

4.2. Miscellaneous

Comments begin with "/*" and end with "*/".

Optional components are denoted by enclosing them in "[[ ]]" double brackets.

Single byte entities containing uninterpreted data are of type opaque.

4.3. Vectors

A vector (single dimensioned array) is a stream of homogeneous data elements. The size of the vector may be specified at documentation time or left unspecified until runtime. In either case the length declares the number of bytes, not the number of elements, in the vector. The syntax for specifying a new type T' that is a fixed length vector of type T is

```
T T'[n];
```

Here T' occupies n bytes in the data stream, where n is a multiple of the size of T. The length of the vector is not included in the encoded stream.

In the following example, Datum is defined to be three consecutive bytes that the protocol does not interpret, while Data is three consecutive Datum, consuming a total of nine bytes.

```
opaque Datum[3];      /* three uninterpreted bytes */
Datum Data[9];        /* 3 consecutive 3 byte vectors */
```


Variable length vectors are defined by specifying a subrange of legal lengths, inclusively, using the notation <floor..ceiling>. When encoded, the actual length precedes the vector's contents in the byte stream. The length will be in the form of a number consuming as many bytes as required to hold the vector's specified maximum (ceiling) length. A variable length vector with an actual length field of zero is referred to as an empty vector.

```
T T'<floor..ceiling>;
```

In the following example, mandatory is a vector that must contain between 300 and 400 bytes of type opaque. It can never be empty. The actual length field consumes two bytes, a uint16, sufficient to represent the value 400 (see Section 4.4). On the other hand, longer can represent up to 800 bytes of data, or 400 uint16 elements, and it may be empty. Its encoding will include a two byte actual length field prepended to the vector. The length of an encoded vector must be an even multiple of the length of a single element (for example, a 17 byte vector of uint16 would be illegal).

```
opaque mandatory<300..400>;
      /* length field is 2 bytes, cannot be empty */
uint16 longer<0..800>;
      /* zero to 400 16-bit unsigned integers */
```

4.4. Numbers

The basic numeric data type is an unsigned byte (uint8). All larger numeric data types are formed from fixed length series of bytes concatenated as described in Section 4.1 and are also unsigned. The following numeric types are predefined.

```
uint8 uint16[2];
uint8 uint24[3];
uint8 uint32[4];
uint8 uint64[8];
```

All values, here and elsewhere in the specification, are stored in "network" or "big-endian" order; the uint32 represented by the hex bytes 01 02 03 04 is equivalent to the decimal value 16909060.

4.5. Enumerateds

An additional sparse data type is available called enum. A field of type enum can only assume the values declared in the definition. Each definition is a different type. Only enumerateds of the same type may be assigned or compared. Every element of an enumerated must

be assigned a value, as demonstrated in the following example. Since the elements of the enumerated are not ordered, they can be assigned any unique value, in any order.

```
enum { e1(v1), e2(v2), ... , en(vn) [[, (n)]] } Te;
```

Enumerateds occupy as much space in the byte stream as would its maximal defined ordinal value. The following definition would cause one byte to be used to carry fields of type Color.

```
enum { red(3), blue(5), white(7) } Color;
```

One may optionally specify a value without its associated tag to force the width definition without defining a superfluous element. In the following example, Taste will consume two bytes in the data stream but can only assume the values 1, 2 or 4.

```
enum { sweet(1), sour(2), bitter(4), (32000) } Taste;
```

The names of the elements of an enumeration are scoped within the defined type. In the first example, a fully qualified reference to the second element of the enumeration would be Color.blue. Such qualification is not required if the target of the assignment is well specified.

```
Color color = Color.blue;     /* overspecified, legal */
Color color = blue;           /* correct, type implicit */
```

For enumerateds that are never converted to external representation, the numerical information may be omitted.

```
enum { low, medium, high } Amount;
```

4.6. Constructed types

Structure types may be constructed from primitive types for convenience. Each specification declares a new, unique type. The syntax for definition is much like that of C.

```
struct {
  T1 f1;
  T2 f2;
  ...
  Tn fn;
} [[T]];
```


The fields within a structure may be qualified using the type's name using a syntax much like that available for enumerateds. For example, T.f2 refers to the second field of the previous declaration. Structure definitions may be embedded.

4.6.1. Variants

Defined structures may have variants based on some knowledge that is available within the environment. The selector must be an enumerated type that defines the possible variants the structure defines. There must be a case arm for every element of the enumeration declared in the select. The body of the variant structure may be given a label for reference. The mechanism by which the variant is selected at runtime is not prescribed by the presentation language.

```
struct {
    T1 f1;
    T2 f2;
    ....
    Tn fn;
    select (E) {
        case e1: Te1;
        case e2: Te2;
        ....
        case en: Ten;
    } [[fv]];
} [[Tv]];
```

For example:

```
enum { apple, orange } VariantTag;
struct {
    uint16 number;
    opaque string<0..10>; /* variable length */
} V1;
struct {
    uint32 number;
    opaque string[10];    /* fixed length */
} V2;
struct {
    select (VariantTag) { /* value of selector is implicit */
        case apple: V1;   /* VariantBody, tag = apple */
        case orange: V2;  /* VariantBody, tag = orange */
    } variant_body;       /* optional label on variant */
} VariantRecord;
```

Variant structures may be qualified (narrowed) by specifying a value for the selector prior to the type. For example, a


```
orange VariantRecord
```

is a narrowed type of a VariantRecord containing a variant_body of type V2.

4.7. Cryptographic attributes

The four cryptographic operations digital signing, stream cipher encryption, block cipher encryption, and public key encryption are designated digitally-signed, stream-ciphered, block-ciphered, and public-key-encrypted, respectively. A field's cryptographic processing is specified by prepending an appropriate key word designation before the field's type specification. Cryptographic keys are implied by the current session state (see Section 6.1).

In digital signing, one-way hash functions are used as input for a signing algorithm. A digitally-signed element is encoded as an opaque vector <0..2^16-1>, where the length is specified by the signing algorithm and key.

In RSA signing, a 36-byte structure of two hashes (one SHA and one MD5) is signed (encrypted with the private key). It is encoded with PKCS #1 block type 0 or type 1 as described in [PKCS1].

In DSS, the 20 bytes of the SHA hash are run directly through the Digital Signing Algorithm with no additional hashing. This produces two values, r and s. The DSS signature is an opaque vector, as above, the contents of which are the DER encoding of:

```
Dss-Sig-Value  ::=  SEQUENCE  {
```

```
        r       INTEGER,
        s       INTEGER
    }
```

In stream cipher encryption, the plaintext is exclusive-ORed with an identical amount of output generated from a cryptographically-secure keyed pseudorandom number generator.

In block cipher encryption, every block of plaintext encrypts to a block of ciphertext. All block cipher encryption is done in CBC (Cipher Block Chaining) mode, and all items which are block-ciphered will be an exact multiple of the cipher block length.

In public key encryption, a public key algorithm is used to encrypt data in such a way that it can be decrypted only with the matching private key. A public-key-encrypted element is encoded as an opaque vector <0..2^16-1>, where the length is specified by the signing algorithm and key.

An RSA encrypted value is encoded with PKCS #1 block type 2 as described in [PKCS1].

In the following example:

```
    stream-ciphered struct {
        uint8 field1;
        uint8 field2;
        digitally-signed opaque hash[20];
    } UserType;
```

The contents of hash are used as input for the signing algorithm, then the entire structure is encrypted with a stream cipher. The length of this structure, in bytes would be equal to 2 bytes for field1 and field2, plus two bytes for the length of the signature, plus the length of the output of the signing algorithm. This is known due to the fact that the algorithm and key used for the signing are known prior to encoding or decoding this structure.

4.8. Constants

Typed constants can be defined for purposes of specification by declaring a symbol of the desired type and assigning values to it. Under-specified types (opaque, variable length vectors, and structures that contain opaque) cannot be assigned values. No fields of a multi-element structure or vector may be elided.

For example,

```
    struct {
        uint8 f1;
        uint8 f2;
    } Example1;

    Example1 ex1 = {1, 4};  /* assigns f1 = 1, f2 = 4 */
```

5. HMAC and the pseudorandom function

A number of operations in the TLS record and handshake layer required a keyed MAC; this is a secure digest of some data protected by a secret. Forging the MAC is infeasible without knowledge of the MAC secret. The construction we use for this operation is known as HMAC, described in [HMAC].

HMAC can be used with a variety of different hash algorithms. TLS uses it in the handshake with two different algorithms: MD5 and SHA-1, denoting these as HMAC_MD5(secret, data) and HMAC_SHA(secret,

data). Additional hash algorithms can be defined by cipher suites and used to protect record data, but MD5 and SHA-1 are hard coded into the description of the handshaking for this version of the protocol.

In addition, a construction is required to do expansion of secrets into blocks of data for the purposes of key generation or validation. This pseudo-random function (PRF) takes as input a secret, a seed, and an identifying label and produces an output of arbitrary length.

In order to make the PRF as secure as possible, it uses two hash algorithms in a way which should guarantee its security if either algorithm remains secure.

First, we define a data expansion function, P_hash(secret, data) which uses a single hash function to expand a secret and seed into an arbitrary quantity of output:

```
    P_hash(secret, seed) = HMAC_hash(secret, A(1) + seed) +
                           HMAC_hash(secret, A(2) + seed) +
                           HMAC_hash(secret, A(3) + seed) + ...
```

Where + indicates concatenation.

A() is defined as:

```
    A(0) = seed
    A(i) = HMAC_hash(secret, A(i-1))
```

P_hash can be iterated as many times as is necessary to produce the required quantity of data. For example, if P_SHA-1 was being used to create 64 bytes of data, it would have to be iterated 4 times (through A(4)), creating 80 bytes of output data; the last 16 bytes of the final iteration would then be discarded, leaving 64 bytes of output data.

TLS's PRF is created by splitting the secret into two halves and using one half to generate data with P_MD5 and the other half to generate data with P_SHA-1, then exclusive-or'ing the outputs of these two expansion functions together.

S1 and S2 are the two halves of the secret and each is the same length. S1 is taken from the first half of the secret, S2 from the second half. Their length is created by rounding up the length of the overall secret divided by two; thus, if the original secret is an odd number of bytes long, the last byte of S1 will be the same as the first byte of S2.

```
L_S = length in bytes of secret;
L_S1 = L_S2 = ceil(L_S / 2);
```


The secret is partitioned into two halves (with the possibility of one shared byte) as described above, S1 taking the first L_S1 bytes and S2 the last L_S2 bytes.

The PRF is then defined as the result of mixing the two pseudorandom streams by exclusive-or'ing them together.

```
PRF(secret, label, seed) = P_MD5(S1, label + seed) XOR
                           P_SHA-1(S2, label + seed);
```

The label is an ASCII string. It should be included in the exact form it is given without a length byte or trailing null character. For example, the label "slithy toves" would be processed by hashing the following bytes:

```
73 6C 69 74 68 79 20 74 6F 76 65 73
```

Note that because MD5 produces 16 byte outputs and SHA-1 produces 20 byte outputs, the boundaries of their internal iterations will not be aligned; to generate a 80 byte output will involve P_MD5 being iterated through A(5), while P_SHA-1 will only iterate through A(4).

6. The TLS Record Protocol

The TLS Record Protocol is a layered protocol. At each layer, messages may include fields for length, description, and content. The Record Protocol takes messages to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, and transmits the result. Received data is decrypted, verified, decompressed, and reassembled, then delivered to higher level clients.

Four record protocol clients are described in this document: the handshake protocol, the alert protocol, the change cipher spec protocol, and the application data protocol. In order to allow extension of the TLS protocol, additional record types can be supported by the record protocol. Any new record types should allocate type values immediately beyond the ContentType values for the four record types described here (see Appendix A.2). If a TLS implementation receives a record type it does not understand, it should just ignore it. Any protocol designed for use over TLS must be carefully designed to deal with all possible attacks against it. Note that because the type and length of a record are not protected by encryption, care should be take to minimize the value of traffic analysis of these values.

6.1. Connection states

A TLS connection state is the operating environment of the TLS Record Protocol. It specifies a compression algorithm, encryption algorithm, and MAC algorithm. In addition, the parameters for these algorithms are known: the MAC secret and the bulk encryption keys and IVs for the connection in both the read and the write directions. Logically, there are always four connection states outstanding: the current read and write states, and the pending read and write states. All records are processed under the current read and write states. The security parameters for the pending states can be set by the TLS Handshake Protocol, and the Handshake Protocol can selectively make either of the pending states current, in which case the appropriate current state is disposed of and replaced with the pending state; the pending state is then reinitialized to an empty state. It is illegal to make a state which has not been initialized with security parameters a current state. The initial current state always specifies that no encryption, compression, or MAC will be used.

The security parameters for a TLS Connection read and write state are set by providing the following values:

connection end
: Whether this entity is considered the "client" or the "server" in this connection.

bulk encryption algorithm
An algorithm to be used for bulk encryption. This specification includes the key size of this algorithm, how much of that key is secret, whether it is a block or stream cipher, the block size of the cipher (if appropriate), and whether it is considered an "export" cipher.

MAC algorithm
An algorithm to be used for message authentication. This specification includes the size of the hash which is returned by the MAC algorithm.

compression algorithm
An algorithm to be used for data compression. This specification must include all information the algorithm requires to do compression.

master secret
A 48 byte secret shared between the two peers in the connection.

client random
A 32 byte value provided by the client.

server random
A 32 byte value provided by the server.

These parameters are defined in the presentation language as:

```
enum { server, client } ConnectionEnd;

enum { null, rc4, rc2, des, 3des, des40 } BulkCipherAlgorithm;

enum { stream, block } CipherType;

enum { true, false } IsExportable;

enum { null, md5, sha } MACAlgorithm;

enum { null(0), (255) } CompressionMethod;

/* The algorithms specified in CompressionMethod,
   BulkCipherAlgorithm, and MACAlgorithm may be added to. */

struct {
    ConnectionEnd          entity;
    BulkCipherAlgorithm    bulk_cipher_algorithm;
    CipherType             cipher_type;
    uint8                  key_size;
    uint8                  key_material_length;
    IsExportable           is_exportable;
    MACAlgorithm           mac_algorithm;
    uint8                  hash_size;
    CompressionMethod      compression_algorithm;
    opaque                 master_secret[48];
    opaque                 client_random[32];
    opaque                 server_random[32];
} SecurityParameters;
```

The record layer will use the security parameters to generate the following six items:

client write MAC secret
server write MAC secret
client write key
server write key
client write IV (for block ciphers only)
server write IV (for block ciphers only)

The client write parameters are used by the server when receiving and processing records and vice-versa. The algorithm used for generating these items from the security parameters is described in section 6.3.

Once the security parameters have been set and the keys have been generated, the connection states can be instantiated by making them the current states. These current states must be updated for each record processed. Each connection state includes the following elements:

compression state
The current state of the compression algorithm.

cipher state
The current state of the encryption algorithm. This will consist of the scheduled key for that connection. In addition, for block ciphers running in CBC mode (the only mode specified for TLS), this will initially contain the IV for that connection state and be updated to contain the ciphertext of the last block encrypted or decrypted as records are processed. For stream ciphers, this will contain whatever the necessary state information is to allow the stream to continue to encrypt or decrypt data.

MAC secret
The MAC secret for this connection as generated above.

sequence number
Each connection state contains a sequence number, which is maintained separately for read and write states. The sequence number must be set to zero whenever a connection state is made the active state. Sequence numbers are of type uint64 and may not exceed 2^64-1. A sequence number is incremented after each record: specifically, the first record which is transmitted under a particular connection state should use sequence number 0.

6.2. Record layer

The TLS Record Layer receives uninterpreted data from higher layers in non-empty blocks of arbitrary size.

6.2.1. Fragmentation

The record layer fragments information blocks into TLSPlaintext records carrying data in chunks of 2^14 bytes or less. Client message boundaries are not preserved in the record layer (i.e., multiple client messages of the same ContentType may be coalesced into a single TLSPlaintext record, or a single message may be fragmented across several records).

```
struct {
    uint8 major, minor;
} ProtocolVersion;
```



```
enum {
    change_cipher_spec(20), alert(21), handshake(22),
    application_data(23), (255)
} ContentType;

struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
    opaque fragment[TLSPlaintext.length];
} TLSPlaintext;
```

type
The higher level protocol used to process the enclosed fragment.

version
The version of the protocol being employed. This document describes TLS Version 1.0, which uses the version { 3, 1 }. The version value 3.1 is historical: TLS version 1.0 is a minor modification to the SSL 3.0 protocol, which bears the version value 3.0. (See Appendix A.1).

length
The length (in bytes) of the following TLSPlaintext.fragment. The length should not exceed 2^14.

fragment
The application data. This data is transparent and treated as an independent block to be dealt with by the higher level protocol specified by the type field.

Note: Data of different TLS Record layer content types may be interleaved. Application data is generally of lower precedence for transmission than other content types.

6.2.2. Record compression and decompression

All records are compressed using the compression algorithm defined in the current session state. There is always an active compression algorithm; however, initially it is defined as CompressionMethod.null. The compression algorithm translates a TLSPlaintext structure into a TLSCompressed structure. Compression functions are initialized with default state information whenever a connection state is made active.

Compression must be lossless and may not increase the content length by more than 1024 bytes. If the decompression function encounters a TLSCompressed.fragment that would decompress to a length in excess of 2^14 bytes, it should report a fatal decompression failure error.

```
struct {
    ContentType type;       /* same as TLSPlaintext.type */
    ProtocolVersion version;/* same as TLSPlaintext.version */
    uint16 length;
    opaque fragment[TLSCompressed.length];
} TLSCompressed;
```

length
The length (in bytes) of the following TLSCompressed.fragment. The length should not exceed 2^14 + 1024.

fragment
The compressed form of TLSPlaintext.fragment.

Note: A CompressionMethod.null operation is an identity operation; no fields are altered.

Implementation note:
Decompression functions are responsible for ensuring that messages cannot cause internal buffer overflows.

6.2.3. Record payload protection

The encryption and MAC functions translate a TLSCompressed structure into a TLSCiphertext. The decryption functions reverse the process. The MAC of the record also includes a sequence number so that missing, extra or repeated messages are detectable.

```
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
    select (CipherSpec.cipher_type) {
        case stream: GenericStreamCipher;
        case block: GenericBlockCipher;
    } fragment;
} TLSCiphertext;
```

type
The type field is identical to TLSCompressed.type.

version
The version field is identical to TLSCompressed.version.

length
The length (in bytes) of the following TLSCiphertext.fragment. The length may not exceed 2^14 + 2048.

fragment
The encrypted form of TLSCompressed.fragment, with the MAC.

6.2.3.1. Null or standard stream cipher

Stream ciphers (including BulkCipherAlgorithm.null - see Appendix A.6) convert TLSCompressed.fragment structures to and from stream TLSCiphertext.fragment structures.

```
stream-ciphered struct {
    opaque content[TLSCompressed.length];
    opaque MAC[CipherSpec.hash_size];
} GenericStreamCipher;
```

The MAC is generated as:

```
HMAC_hash(MAC_write_secret, seq_num + TLSCompressed.type +
                TLSCompressed.version + TLSCompressed.length +
                TLSCompressed.fragment));
```

where "+" denotes concatenation.

seq_num
The sequence number for this record.

hash
The hashing algorithm specified by SecurityParameters.mac_algorithm.

Note that the MAC is computed before encryption. The stream cipher encrypts the entire block, including the MAC. For stream ciphers that do not use a synchronization vector (such as RC4), the stream cipher state from the end of one record is simply used on the subsequent packet. If the CipherSuite is TLS_NULL_WITH_NULL_NULL, encryption consists of the identity operation (i.e., the data is not encrypted and the MAC size is zero implying that no MAC is used). TLSCiphertext.length is TLSCompressed.length plus CipherSpec.hash_size.

6.2.3.2. CBC block cipher

For block ciphers (such as RC2 or DES), the encryption and MAC functions convert TLSCompressed.fragment structures to and from block TLSCiphertext.fragment structures.


```
block-ciphered struct {
    opaque content[TLSCompressed.length];
    opaque MAC[CipherSpec.hash_size];
    uint8 padding[GenericBlockCipher.padding_length];
    uint8 padding_length;
} GenericBlockCipher;
```

The MAC is generated as described in Section 6.2.3.1.

padding
Padding that is added to force the length of the plaintext to be an integral multiple of the block cipher's block length. The padding may be any length up to 255 bytes long, as long as it results in the TLSCiphertext.length being an integral multiple of the block length. Lengths longer than necessary might be desirable to frustrate attacks on a protocol based on analysis of

the lengths of exchanged messages. Each uint8 in the padding data vector must be filled with the padding length value.

padding_length
The padding length should be such that the total size of the GenericBlockCipher structure is a multiple of the cipher's block length. Legal values range from zero to 255, inclusive. This length specifies the length of the padding field exclusive of the padding_length field itself.

The encrypted data length (TLSCiphertext.length) is one more than the sum of TLSCompressed.length, CipherSpec.hash_size, and padding_length.

Example: If the block length is 8 bytes, the content length (TLSCompressed.length) is 61 bytes, and the MAC length is 20 bytes, the length before padding is 82 bytes. Thus, the padding length modulo 8 must be equal to 6 in order to make the total length an even multiple of 8 bytes (the block length). The padding length can be 6, 14, 22, and so on, through 254. If the padding length were the minimum necessary, 6, the padding would be 6 bytes, each containing the value 6. Thus, the last 8 octets of the GenericBlockCipher before block encryption would be xx 06 06 06 06 06 06 06, where xx is the last octet of the MAC.

Note: With block ciphers in CBC mode (Cipher Block Chaining) the initialization vector (IV) for the first record is generated with the other keys and secrets when the security parameters are set. The IV for subsequent records is the last ciphertext block from the previous record.

6.3. Key calculation

The Record Protocol requires an algorithm to generate keys, IVs, and MAC secrets from the security parameters provided by the handshake protocol.

The master secret is hashed into a sequence of secure bytes, which are assigned to the MAC secrets, keys, and non-export IVs required by the current connection state (see Appendix A.6). CipherSpecs require a client write MAC secret, a server write MAC secret, a client write key, a server write key, a client write IV, and a server write IV, which are generated from the master secret in that order. Unused values are empty.

When generating keys and MAC secrets, the master secret is used as an entropy source, and the random values provide unencrypted salt material and IVs for exportable ciphers.

To generate the key material, compute

```
key_block = PRF(SecurityParameters.master_secret,
                "key expansion",
                SecurityParameters.server_random +
                SecurityParameters.client_random);
```

until enough output has been generated. Then the key_block is partitioned as follows:

```
client_write_MAC_secret[SecurityParameters.hash_size]
server_write_MAC_secret[SecurityParameters.hash_size]
client_write_key[SecurityParameters.key_material_length]
server_write_key[SecurityParameters.key_material_length]
client_write_IV[SecurityParameters.IV_size]
server_write_IV[SecurityParameters.IV_size]
```

The client_write_IV and server_write_IV are only generated for non-export block ciphers. For exportable block ciphers, the initialization vectors are generated later, as described below. Any extra key_block material is discarded.

Implementation note:
The cipher spec which is defined in this document which requires the most material is 3DES_EDE_CBC_SHA: it requires 2 x 24 byte keys, 2 x 20 byte MAC secrets, and 2 x 8 byte IVs, for a total of 104 bytes of key material.

Exportable encryption algorithms (for which CipherSpec.is_exportable is true) require additional processing as follows to derive their final write keys:

```
final_client_write_key =
PRF(SecurityParameters.client_write_key,
                          "client write key",
                          SecurityParameters.client_random +
                          SecurityParameters.server_random);
final_server_write_key =
PRF(SecurityParameters.server_write_key,
                          "server write key",
                          SecurityParameters.client_random +
```

```
                                SecurityParameters.server_random);
```

Exportable encryption algorithms derive their IVs solely from the random values from the hello messages:

```
    iv_block = PRF("", "IV block", SecurityParameters.client_random +
                   SecurityParameters.server_random);
```

The iv_block is partitioned into two initialization vectors as the key_block was above:

```
    client_write_IV[SecurityParameters.IV_size]
    server_write_IV[SecurityParameters.IV_size]
```

Note that the PRF is used without a secret in this case: this just means that the secret has a length of zero bytes and contributes nothing to the hashing in the PRF.

6.3.1. Export key generation example

TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 requires five random bytes for each of the two encryption keys and 16 bytes for each of the MAC keys, for a total of 42 bytes of key material. The PRF output is stored in the key_block. The key_block is partitioned, and the write keys are salted because this is an exportable encryption algorithm.

```
    key_block               = PRF(master_secret,
                                  "key expansion",
                                  server_random +
                                  client_random)[0..41]
    client_write_MAC_secret = key_block[0..15]
    server_write_MAC_secret = key_block[16..31]
    client_write_key        = key_block[32..36]
    server_write_key        = key_block[37..41]
```



```
    final_client_write_key  = PRF(client_write_key,
                                  "client write key",
                                  client_random +
                                  server_random)[0..15]
    final_server_write_key  = PRF(server_write_key,
                                  "server write key",
                                  client_random +
                                  server_random)[0..15]

    iv_block                = PRF("", "IV block", client_random +
                                  server_random)[0..15]
    client_write_IV = iv_block[0..7]
    server_write_IV = iv_block[8..15]
```

7. The TLS Handshake Protocol

The TLS Handshake Protocol consists of a suite of three sub-protocols which are used to allow peers to agree upon security parameters for the record layer, authenticate themselves, instantiate negotiated security parameters, and report error conditions to each other.

The Handshake Protocol is responsible for negotiating a session, which consists of the following items:

session identifier
: An arbitrary byte sequence chosen by the server to identify an active or resumable session state.

peer certificate
: X509v3 [X509] certificate of the peer. This element of the state may be null.

compression method
: The algorithm used to compress data prior to encryption.

cipher spec
: Specifies the bulk data encryption algorithm (such as null, DES, etc.) and a MAC algorithm (such as MD5 or SHA). It also defines cryptographic attributes such as the hash_size. (See Appendix A.6 for formal definition)

master secret
: 48-byte secret shared between the client and server.

is resumable
: A flag indicating whether the session can be used to initiate new connections.

These items are then used to create security parameters for use by the Record Layer when protecting application data. Many connections can be instantiated using the same session through the resumption feature of the TLS Handshake Protocol.

7.1. Change cipher spec protocol

The change cipher spec protocol exists to signal transitions in ciphering strategies. The protocol consists of a single message, which is encrypted and compressed under the current (not the pending)

connection state. The message consists of a single byte of value 1.

```
struct {
    enum { change_cipher_spec(1), (255) } type;
} ChangeCipherSpec;
```

The change cipher spec message is sent by both the client and server to notify the receiving party that subsequent records will be protected under the newly negotiated CipherSpec and keys. Reception of this message causes the receiver to instruct the Record Layer to immediately copy the read pending state into the read current state. Immediately after sending this message, the sender should instruct the record layer to make the write pending state the write active state. (See section 6.1.) The change cipher spec message is sent during the handshake after the security parameters have been agreed upon, but before the verifying finished message is sent (see section 7.4.9).

7.2. Alert protocol

One of the content types supported by the TLS Record layer is the alert type. Alert messages convey the severity of the message and a description of the alert. Alert messages with a level of fatal result in the immediate termination of the connection. In this case, other connections corresponding to the session may continue, but the session identifier must be invalidated, preventing the failed session from being used to establish new connections. Like other messages, alert messages are encrypted and compressed, as specified by the current connection state.

```
enum { warning(1), fatal(2), (255) } AlertLevel;

enum {
    close_notify(0),
    unexpected_message(10),
    bad_record_mac(20),
    decryption_failed(21),
    record_overflow(22),
```



```
    decompression_failure(30),
    handshake_failure(40),
    bad_certificate(42),
    unsupported_certificate(43),
    certificate_revoked(44),
    certificate_expired(45),
    certificate_unknown(46),
    illegal_parameter(47),
    unknown_ca(48),
    access_denied(49),
    decode_error(50),
    decrypt_error(51),
    export_restriction(60),
    protocol_version(70),
    insufficient_security(71),
    internal_error(80),
    user_canceled(90),
    no_renegotiation(100),
    (255)
} AlertDescription;

struct {
    AlertLevel level;
    AlertDescription description;
} Alert;
```

7.2.1. Closure alerts

The client and the server must share knowledge that the connection is ending in order to avoid a truncation attack. Either party may initiate the exchange of closing messages.

close_notify
: This message notifies the recipient that the sender will not send any more messages on this connection. The session becomes unresumable if any connection is terminated without proper close_notify messages with level equal to warning.

Either party may initiate a close by sending a close_notify alert. Any data received after a closure alert is ignored.

Each party is required to send a close_notify alert before closing the write side of the connection. It is required that the other party respond with a close_notify alert of its own and close down the connection immediately, discarding any pending writes. It is not required for the initiator of the close to wait for the responding close_notify alert before closing the read side of the connection.

If the application protocol using TLS provides that any data may be carried over the underlying transport after the TLS connection is closed, the TLS implementation must receive the responding close_notify alert before indicating to the application layer that the TLS connection has ended. If the application protocol will not transfer any additional data, but will only close the underlying transport connection, then the implementation may choose to close the

transport without waiting for the responding close_notify. No part of this standard should be taken to dictate the manner in which a usage profile for TLS manages its data transport, including when connections are opened or closed.

NB: It is assumed that closing a connection reliably delivers pending data before destroying the transport.

7.2.2. Error alerts

Error handling in the TLS Handshake protocol is very simple. When an error is detected, the detecting party sends a message to the other party. Upon transmission or receipt of an fatal alert message, both parties immediately close the connection. Servers and clients are required to forget any session-identifiers, keys, and secrets associated with a failed connection. The following error alerts are defined:

unexpected_message
An inappropriate message was received. This alert is always fatal and should never be observed in communication between proper implementations.

bad_record_mac
This alert is returned if a record is received with an incorrect MAC. This message is always fatal.

decryption_failed
A TLSCiphertext decrypted in an invalid way: either it wasn't an even multiple of the block length or its padding values, when checked, weren't correct. This message is always fatal.

record_overflow
A TLSCiphertext record was received which had a length more than 2^14+2048 bytes, or a record decrypted to a TLSCompressed record with more than 2^14+1024 bytes. This message is always fatal.

decompression_failure
The decompression function received improper input (e.g. data that would expand to excessive length). This message is always fatal.

handshake_failure
Reception of a handshake_failure alert message indicates that the sender was unable to negotiate an acceptable set of security parameters given the options available. This is a fatal error.

bad_certificate
A certificate was corrupt, contained signatures that did not verify correctly, etc.

unsupported_certificate
A certificate was of an unsupported type.

certificate_revoked
A certificate was revoked by its signer.

certificate_expired
A certificate has expired or is not currently valid.

certificate_unknown
Some other (unspecified) issue arose in processing the certificate, rendering it unacceptable.

illegal_parameter
A field in the handshake was out of range or inconsistent with other fields. This is always fatal.

unknown_ca
A valid certificate chain or partial chain was received, but the certificate was not accepted because the CA certificate could not be located or couldn't be matched with a known, trusted CA. This message is always fatal.

access_denied
A valid certificate was received, but when access control was applied, the sender decided not to proceed with negotiation. This message is always fatal.

decode_error
A message could not be decoded because some field was out of the specified range or the length of the message was incorrect. This message is always fatal.

decrypt_error
A handshake cryptographic operation failed, including being unable to correctly verify a signature, decrypt a key exchange, or validate a finished message.

export_restriction
A negotiation not in compliance with export restrictions was detected; for example, attempting to transfer a 1024 bit ephemeral RSA key for the RSA_EXPORT handshake method. This

message is always fatal.

protocol_version
The protocol version the client has attempted to negotiate is recognized, but not supported. (For example, old protocol versions might be avoided for security reasons). This message is always fatal.

insufficient_security
Returned instead of handshake_failure when a negotiation has failed specifically because the server requires ciphers more secure than those supported by the client. This message is always fatal.

internal_error
An internal error unrelated to the peer or the correctness of the protocol makes it impossible to continue (such as a memory allocation failure). This message is always fatal.

user_canceled
This handshake is being canceled for some reason unrelated to a protocol failure. If the user cancels an operation after the handshake is complete, just closing the connection by sending a close_notify is more appropriate. This alert should be followed by a close_notify. This message is generally a warning.

no_renegotiation
Sent by the client in response to a hello request or by the server in response to a client hello after initial handshaking. Either of these would normally lead to renegotiation; when that is not appropriate, the recipient should respond with this alert; at that point, the original requester can decide whether to proceed with the connection. One case where this would be appropriate would be where a server has spawned a process to satisfy a request; the process might receive security parameters (key length, authentication, etc.) at startup and it might be difficult to communicate changes to these parameters after that point. This message is always a warning.

For all errors where an alert level is not explicitly specified, the sending party may determine at its discretion whether this is a fatal error or not; if an alert with a level of warning is received, the

receiving party may decide at its discretion whether to treat this as a fatal error or not. However, all messages which are transmitted with a level of fatal must be treated as fatal messages.

7.3. Handshake Protocol overview

The cryptographic parameters of the session state are produced by the TLS Handshake Protocol, which operates on top of the TLS Record Layer. When a TLS client and server first start communicating, they agree on a protocol version, select cryptographic algorithms, optionally authenticate each other, and use public-key encryption techniques to generate shared secrets.

The TLS Handshake Protocol involves the following steps:

- Exchange hello messages to agree on algorithms, exchange random values, and check for session resumption.

- Exchange the necessary cryptographic parameters to allow the client and server to agree on a premaster secret.

- Exchange certificates and cryptographic information to allow the client and server to authenticate themselves.

- Generate a master secret from the premaster secret and exchanged random values.

- Provide security parameters to the record layer.

- Allow the client and server to verify that their peer has calculated the same security parameters and that the handshake occurred without tampering by an attacker.

Note that higher layers should not be overly reliant on TLS always negotiating the strongest possible connection between two peers: there are a number of ways a man in the middle attacker can attempt to make two entities drop down to the least secure method they support. The protocol has been designed to minimize this risk, but there are still attacks available: for example, an attacker could block access to the port a secure service runs on, or attempt to get the peers to negotiate an unauthenticated connection. The fundamental rule is that higher levels must be cognizant of what their security requirements are and never transmit information over a channel less secure than what they require. The TLS protocol is secure, in that any cipher suite offers its promised level of security: if you negotiate 3DES with a 1024 bit RSA key exchange with a host whose certificate you have verified, you can expect to be that secure.

However, you should never send data over a link encrypted with 40 bit

security unless you feel that data is worth no more than the effort required to break that encryption.

These goals are achieved by the handshake protocol, which can be summarized as follows: The client sends a client hello message to which the server must respond with a server hello message, or else a fatal error will occur and the connection will fail. The client hello and server hello are used to establish security enhancement capabilities between client and server. The client hello and server hello establish the following attributes: Protocol Version, Session ID, Cipher Suite, and Compression Method. Additionally, two random values are generated and exchanged: ClientHello.random and ServerHello.random.

The actual key exchange uses up to four messages: the server certificate, the server key exchange, the client certificate, and the client key exchange. New key exchange methods can be created by specifying a format for these messages and defining the use of the messages to allow the client and server to agree upon a shared secret. This secret should be quite long; currently defined key exchange methods exchange secrets which range from 48 to 128 bytes in length.

Following the hello messages, the server will send its certificate, if it is to be authenticated. Additionally, a server key exchange message may be sent, if it is required (e.g. if their server has no certificate, or if its certificate is for signing only). If the server is authenticated, it may request a certificate from the client, if that is appropriate to the cipher suite selected. Now the server will send the server hello done message, indicating that the hello-message phase of the handshake is complete. The server will then wait for a client response. If the server has sent a certificate request message, the client must send the certificate message. The client key exchange message is now sent, and the content of that message will depend on the public key algorithm selected between the client hello and the server hello. If the client has sent a certificate with signing ability, a digitally-signed certificate verify message is sent to explicitly verify the certificate.

At this point, a change cipher spec message is sent by the client, and the client copies the pending Cipher Spec into the current Cipher Spec. The client then immediately sends the finished message under the new algorithms, keys, and secrets. In response, the server will send its own change cipher spec message, transfer the pending to the current Cipher Spec, and send its finished message under the new

Cipher Spec. At this point, the handshake is complete and the client and server may begin to exchange application layer data. (See flow chart below.)



Fig. 1 - Message flow for a full handshake

* Indicates optional or situation-dependent messages that are not always sent.

Note: To help avoid pipeline stalls, ChangeCipherSpec is an independent TLS Protocol content type, and is not actually a TLS handshake message.

When the client and server decide to resume a previous session or duplicate an existing session (instead of negotiating new security parameters) the message flow is as follows:

The client sends a ClientHello using the Session ID of the session to be resumed. The server then checks its session cache for a match. If a match is found, and the server is willing to re-establish the connection under the specified session state, it will send a ServerHello with the same Session ID value. At this point, both client and server must send change cipher spec messages and proceed directly to finished messages. Once the re-establishment is complete, the client and server may begin to exchange application layer data. (See flow chart below.) If a Session ID match is not found, the server generates a new session ID and the TLS client and server perform a full handshake.

```
   Client                                               Server

   ClientHello                  -------->
                                                   ServerHello
                                                  [ChangeCipherSpec]
                                <--------             Finished
   [ChangeCipherSpec]
   Finished                     -------->
   Application Data             <------->     Application Data
```

Fig. 2 - Message flow for an abbreviated handshake

The contents and significance of each message will be presented in detail in the following sections.

7.4. Handshake protocol

The TLS Handshake Protocol is one of the defined higher level clients of the TLS Record Protocol. This protocol is used to negotiate the secure attributes of a session. Handshake messages are supplied to the TLS Record Layer, where they are encapsulated within one or more TLSPlaintext structures, which are processed and transmitted as specified by the current active session state.

```
    enum {
        hello_request(0), client_hello(1), server_hello(2),
        certificate(11), server_key_exchange (12),
        certificate_request(13), server_hello_done(14),
        certificate_verify(15), client_key_exchange(16),
        finished(20), (255)
    } HandshakeType;

    struct {
        HandshakeType msg_type;    /* handshake type */
        uint24 length;             /* bytes in message */
        select (HandshakeType) {
            case hello_request:       HelloRequest;
            case client_hello:        ClientHello;
            case server_hello:        ServerHello;
            case certificate:         Certificate;
            case server_key_exchange: ServerKeyExchange;
            case certificate_request: CertificateRequest;
            case server_hello_done:   ServerHelloDone;
            case certificate_verify:  CertificateVerify;
            case client_key_exchange: ClientKeyExchange;
            case finished:            Finished;
        } body;
    } Handshake;
```


The handshake protocol messages are presented below in the order they must be sent; sending handshake messages in an unexpected order results in a fatal error. Unneeded handshake messages can be omitted, however. Note one exception to the ordering: the Certificate message is used twice in the handshake (from server to client, then from client to server), but described only in its first position. The one message which is not bound by these ordering rules in the Hello Request message, which can be sent at any time, but which should be ignored by the client if it arrives in the middle of a handshake.

7.4.1. Hello messages

The hello phase messages are used to exchange security enhancement capabilities between the client and server. When a new session begins, the Record Layer's connection state encryption, hash, and compression algorithms are initialized to null. The current connection state is used for renegotiation messages.

7.4.1.1. Hello request

When this message will be sent:
The hello request message may be sent by the server at any time.

Meaning of this message:
Hello request is a simple notification that the client should begin the negotiation process anew by sending a client hello message when convenient. This message will be ignored by the client if the client is currently negotiating a session. This message may be ignored by the client if it does not wish to renegotiate a session, or the client may, if it wishes, respond with a no_renegotiation alert. Since handshake messages are intended to have transmission precedence over application data, it is expected that the negotiation will begin before no more than a few records are received from the client. If the server sends a hello request but does not receive a client hello in response, it may close the connection with a fatal alert.

After sending a hello request, servers should not repeat the request until the subsequent handshake negotiation is complete.

Structure of this message:

```
    struct { } HelloRequest;
```

Note: This message should never be included in the message hashes which are maintained throughout the handshake and used in the finished messages and the certificate verify message.

7.4.1.2. Client hello

When this message will be sent:
When a client first connects to a server it is required to send the client hello as its first message. The client can also send a client hello in response to a hello request or on its own initiative in order to renegotiate the security parameters in an existing connection.

Structure of this message:
The client hello message includes a random structure, which is used later in the protocol.

```
struct {
   uint32 gmt_unix_time;
   opaque random_bytes[28];
} Random;
```

gmt_unix_time
The current time and date in standard UNIX 32-bit format (seconds since the midnight starting Jan 1, 1970, GMT) according to the sender's internal clock. Clocks are not required to be set correctly by the basic TLS Protocol; higher level or application protocols may define additional requirements.

random_bytes
28 bytes generated by a secure random number generator.

The client hello message includes a variable length session identifier. If not empty, the value identifies a session between the same client and server whose security parameters the client wishes to reuse. The session identifier may be from an earlier connection, this connection, or another currently active connection. The second option is useful if the client only wishes to update the random structures and derived values of a connection, while the third option makes it possible to establish several independent secure connections without repeating the full handshake protocol. These independent connections may occur sequentially or simultaneously; a SessionID becomes valid when the handshake negotiating it completes with the exchange of Finished messages and persists until removed due to aging or because a fatal error was encountered on a connection associated with the session. The actual contents of the SessionID are defined by the server.

```
opaque SessionID<0..32>;
```


Warning:
Because the SessionID is transmitted without encryption or immediate MAC protection, servers must not place confidential information in session identifiers or let the contents of fake session identifiers cause any breach of security. (Note that the content of the handshake as a whole, including the SessionID, is protected by the Finished messages exchanged at the end of the handshake.)

The CipherSuite list, passed from the client to the server in the client hello message, contains the combinations of cryptographic algorithms supported by the client in order of the client's preference (favorite choice first). Each CipherSuite defines a key exchange algorithm, a bulk encryption algorithm (including secret key length) and a MAC algorithm. The server will select a cipher suite or, if no acceptable choices are presented, return a handshake failure alert and close the connection.

```
uint8 CipherSuite[2];    /* Cryptographic suite selector */
```

The client hello includes a list of compression algorithms supported by the client, ordered according to the client's preference.

```
enum { null(0), (255) } CompressionMethod;

struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites<2..2^16-1>;
    CompressionMethod compression_methods<1..2^8-1>;
} ClientHello;
```

client_version
The version of the TLS protocol by which the client wishes to communicate during this session. This should be the latest (highest valued) version supported by the client. For this version of the specification, the version will be 3.1 (See Appendix E for details about backward compatibility).

random
A client-generated random structure.

session_id
The ID of a session the client wishes to use for this connection. This field should be empty if no session_id is available or the client wishes to generate new security parameters.

cipher_suites
This is a list of the cryptographic options supported by the client, with the client's first preference first. If the session_id field is not empty (implying a session resumption request) this vector must include at least the cipher_suite from that session. Values are defined in Appendix A.5.

compression_methods
This is a list of the compression methods supported by the client, sorted by client preference. If the session_id field is not empty (implying a session resumption request) it must include the compression_method from that session. This vector must contain, and all implementations must support, CompressionMethod.null. Thus, a client and server will always be able to agree on a compression method.

After sending the client hello message, the client waits for a server hello message. Any other handshake message returned by the server except for a hello request is treated as a fatal error.

Forward compatibility note:
In the interests of forward compatibility, it is permitted for a client hello message to include extra data after the compression methods. This data must be included in the handshake hashes, but must otherwise be ignored. This is the only handshake message for which this is legal; for all other messages, the amount of data in the message must match the description of the message precisely.

7.4.1.3. Server hello

When this message will be sent:
The server will send this message in response to a client hello message when it was able to find an acceptable set of algorithms. If it cannot find such a match, it will respond with a handshake failure alert.

Structure of this message:

```
struct {
    ProtocolVersion server_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
} ServerHello;
```


server_version
This field will contain the lower of that suggested by the client in the client hello and the highest supported by the server. For this version of the specification, the version is 3.1 (See Appendix E for details about backward compatibility).

random
This structure is generated by the server and must be different from (and independent of) ClientHello.random.

session_id
This is the identity of the session corresponding to this connection. If the ClientHello.session_id was non-empty, the server will look in its session cache for a match. If a match is found and the server is willing to establish the new connection using the specified session state, the server will respond with the same value as was supplied by the client. This indicates a resumed session and dictates that the parties must proceed directly to the finished messages. Otherwise this field will contain a different value identifying the new session. The server may return an empty session_id to indicate that the session will not be cached and therefore cannot be resumed. If a session is resumed, it must be resumed using the same cipher suite it was originally negotiated with.

cipher_suite
The single cipher suite selected by the server from the list in ClientHello.cipher_suites. For resumed sessions this field is the value from the state of the session being resumed.

compression_method
The single compression algorithm selected by the server from the list in ClientHello.compression_methods. For resumed sessions this field is the value from the resumed session state.

7.4.2. Server certificate

When this message will be sent:
The server must send a certificate whenever the agreed-upon key exchange method is not an anonymous one. This message will always immediately follow the server hello message.

Meaning of this message:
The certificate type must be appropriate for the selected cipher suite's key exchange algorithm, and is generally an X.509v3 certificate. It must contain a key which matches the key exchange method, as follows. Unless otherwise specified, the signing

algorithm for the certificate must be the same as the algorithm for the certificate key. Unless otherwise specified, the public key may be of any length.

Key Exchange Algorithm	Certificate Key Type
RSA	RSA public key; the certificate must allow the key to be used for encryption.
RSA_EXPORT	RSA public key of length greater than 512 bits which can be used for signing, or a key of 512 bits or shorter which can be used for either encryption or signing.
DHE_DSS	DSS public key.
DHE_DSS_EXPORT	DSS public key.
DHE_RSA	RSA public key which can be used for signing.
DHE_RSA_EXPORT	RSA public key which can be used for signing.
DH_DSS	Diffie-Hellman key. The algorithm used to sign the certificate should be DSS.
DH_RSA	Diffie-Hellman key. The algorithm used to sign the certificate should be RSA.

All certificate profiles, key and cryptographic formats are defined by the IETF PKIX working group [PKIX]. When a key usage extension is present, the digitalSignature bit must be set for the key to be eligible for signing, as described above, and the keyEncipherment bit must be present to allow encryption, as described above. The keyAgreement bit must be set on Diffie-Hellman certificates.

As CipherSuites which specify new key exchange methods are specified for the TLS Protocol, they will imply certificate format and the required encoded keying information.

Structure of this message:

```
opaque ASN.1Cert<1..2^24-1>;

struct {
    ASN.1Cert certificate_list<0..2^24-1>;
} Certificate;
```


certificate_list
This is a sequence (chain) of X.509v3 certificates. The sender's certificate must come first in the list. Each following certificate must directly certify the one preceding it. Because certificate validation requires that root keys be distributed independently, the self-signed certificate which specifies the root certificate authority may optionally be omitted from the chain, under the assumption that the remote end must already possess it in order to validate it in any case.

The same message type and structure will be used for the client's response to a certificate request message. Note that a client may send no certificates if it does not have an appropriate certificate to send in response to the server's authentication request.

Note: PKCS #7 [PKCS7] is not used as the format for the certificate vector because PKCS #6 [PKCS6] extended certificates are not used. Also PKCS #7 defines a SET rather than a SEQUENCE, making the task of parsing the list more difficult.

7.4.3. Server key exchange message

When this message will be sent:
This message will be sent immediately after the server certificate message (or the server hello message, if this is an anonymous negotiation).

The server key exchange message is sent by the server only when the server certificate message (if sent) does not contain enough data to allow the client to exchange a premaster secret. This is true for the following key exchange methods:

- RSA_EXPORT (if the public key in the server certificate is longer than 512 bits)
- DHE_DSS
- DHE_DSS_EXPORT
- DHE_RSA
- DHE_RSA_EXPORT
- DH_anon

It is not legal to send the server key exchange message for the following key exchange methods:

- RSA
- RSA_EXPORT (when the public key in the server certificate is

```
          less than or equal to 512 bits in length)
          DH_DSS
          DH_RSA
```


Meaning of this message:
This message conveys cryptographic information to allow the client to communicate the premaster secret: either an RSA public key to encrypt the premaster secret with, or a Diffie-Hellman public key with which the client can complete a key exchange (with the result being the premaster secret.)

As additional CipherSuites are defined for TLS which include new key exchange algorithms, the server key exchange message will be sent if and only if the certificate type associated with the key exchange algorithm does not provide enough information for the client to exchange a premaster secret.

Note: According to current US export law, RSA moduli larger than 512 bits may not be used for key exchange in software exported from the US. With this message, the larger RSA keys encoded in certificates may be used to sign temporary shorter RSA keys for the RSA_EXPORT key exchange method.

Structure of this message:

```
enum { rsa, diffie_hellman } KeyExchangeAlgorithm;

struct {
    opaque rsa_modulus<1..2^16-1>;
    opaque rsa_exponent<1..2^16-1>;
} ServerRSAParams;
```

rsa_modulus
The modulus of the server's temporary RSA key.

rsa_exponent
The public exponent of the server's temporary RSA key.

```
struct {
    opaque dh_p<1..2^16-1>;
    opaque dh_g<1..2^16-1>;
    opaque dh_Ys<1..2^16-1>;
} ServerDHParams;     /* Ephemeral DH parameters */
```

dh_p
The prime modulus used for the Diffie-Hellman operation.

dh_g
The generator used for the Diffie-Hellman operation.

dh_Ys
The server's Diffie-Hellman public value (g^X mod p).


```
struct {
    select (KeyExchangeAlgorithm) {
        case diffie_hellman:
            ServerDHParams params;
            Signature signed_params;
        case rsa:
            ServerRSAParams params;
            Signature signed_params;
    };
} ServerKeyExchange;
```

params
The server's key exchange parameters.

signed_params
For non-anonymous key exchanges, a hash of the corresponding params value, with the signature appropriate to that hash applied.

```
md5_hash
    MD5(ClientHello.random + ServerHello.random + ServerParams);

sha_hash
    SHA(ClientHello.random + ServerHello.random + ServerParams);

enum { anonymous, rsa, dsa } SignatureAlgorithm;

select (SignatureAlgorithm)
{   case anonymous: struct { };
    case rsa:
        digitally-signed struct {
            opaque md5_hash[16];
            opaque sha_hash[20];
        };
    case dsa:
        digitally-signed struct {
            opaque sha_hash[20];
        };
} Signature;
```

7.4.4. Certificate request

When this message will be sent:
A non-anonymous server can optionally request a certificate from the client, if appropriate for the selected cipher suite. This message, if sent, will immediately follow the Server Key Exchange message (if it is sent; otherwise, the Server Certificate message).

Structure of this message:

```
enum {
    rsa_sign(1), dss_sign(2), rsa_fixed_dh(3), dss_fixed_dh(4),
    (255)
} ClientCertificateType;

opaque DistinguishedName<1..2^16-1>;

struct {
    ClientCertificateType certificate_types<1..2^8-1>;
    DistinguishedName certificate_authorities<3..2^16-1>;
} CertificateRequest;
```

certificate_types
This field is a list of the types of certificates requested, sorted in order of the server's preference.

certificate_authorities
A list of the distinguished names of acceptable certificate authorities. These distinguished names may specify a desired distinguished name for a root CA or for a subordinate CA; thus, this message can be used both to describe known roots and a desired authorization space.

Note: DistinguishedName is derived from [X509].

Note: It is a fatal handshake_failure alert for an anonymous server to request client identification.

7.4.5. Server hello done

When this message will be sent:
The server hello done message is sent by the server to indicate the end of the server hello and associated messages. After sending this message the server will wait for a client response.

Meaning of this message:
This message means that the server is done sending messages to support the key exchange, and the client can proceed with its phase of the key exchange.

Upon receipt of the server hello done message the client should verify that the server provided a valid certificate if required and check that the server hello parameters are acceptable.

Structure of this message:

```
struct { } ServerHelloDone;
```


7.4.6. Client certificate

When this message will be sent:
This is the first message the client can send after receiving a server hello done message. This message is only sent if the server requests a certificate. If no suitable certificate is available, the client should send a certificate message containing no certificates. If client authentication is required by the server for the handshake to continue, it may respond with a fatal handshake failure alert. Client certificates are sent using the Certificate structure defined in Section 7.4.2.

Note: When using a static Diffie-Hellman based key exchange method (DH_DSS or DH_RSA), if client authentication is requested, the Diffie-Hellman group and generator encoded in the client's certificate must match the server specified Diffie-Hellman parameters if the client's parameters are to be used for the key exchange.

7.4.7. Client key exchange message

When this message will be sent:
This message is always sent by the client. It will immediately follow the client certificate message, if it is sent. Otherwise it will be the first message sent by the client after it receives the server hello done message.

Meaning of this message:
With this message, the premaster secret is set, either though direct transmission of the RSA-encrypted secret, or by the transmission of Diffie-Hellman parameters which will allow each side to agree upon the same premaster secret. When the key exchange method is DH_RSA or DH_DSS, client certification has been requested, and the client was able to respond with a certificate which contained a Diffie-Hellman public key whose parameters (group and generator) matched those specified by the server in its certificate, this message will not contain any data.

Structure of this message:
The choice of messages depends on which key exchange method has been selected. See Section 7.4.3 for the KeyExchangeAlgorithm definition.

```
struct {
    select (KeyExchangeAlgorithm) {
        case rsa: EncryptedPreMasterSecret;
        case diffie_hellman: ClientDiffieHellmanPublic;
    } exchange_keys;
} ClientKeyExchange;
```


7.4.7.1. RSA encrypted premaster secret message

Meaning of this message:
If RSA is being used for key agreement and authentication, the client generates a 48-byte premaster secret, encrypts it using the public key from the server's certificate or the temporary RSA key provided in a server key exchange message, and sends the result in an encrypted premaster secret message. This structure is a variant of the client key exchange message, not a message in itself.

Structure of this message:

```
struct {
    ProtocolVersion client_version;
    opaque random[46];
} PreMasterSecret;
```

client_version
The latest (newest) version supported by the client. This is used to detect version roll-back attacks. Upon receiving the premaster secret, the server should check that this value matches the value transmitted by the client in the client hello message.

random
46 securely-generated random bytes.

```
struct {
    public-key-encrypted PreMasterSecret pre_master_secret;
} EncryptedPreMasterSecret;
```

Note: An attack discovered by Daniel Bleichenbacher [BLEI] can be used to attack a TLS server which is using PKCS#1 encoded RSA. The attack takes advantage of the fact that by failing in different ways, a TLS server can be coerced into revealing whether a particular message, when decrypted, is properly PKCS#1 formatted or not.

The best way to avoid vulnerability to this attack is to treat incorrectly formatted messages in a manner indistinguishable from correctly formatted RSA blocks. Thus, when it receives an incorrectly formatted RSA block, a server should generate a random 48-byte value and proceed using it as the premaster secret. Thus, the server will act identically whether the received RSA block is correctly encoded or not.

pre_master_secret
This random value is generated by the client and is used to generate the master secret, as specified in Section 8.1.

7.4.7.2. Client Diffie-Hellman public value

Meaning of this message:
This structure conveys the client's Diffie-Hellman public value (Yc) if it was not already included in the client's certificate. The encoding used for Yc is determined by the enumerated PublicValueEncoding. This structure is a variant of the client key exchange message, not a message in itself.

Structure of this message:

```
enum { implicit, explicit } PublicValueEncoding;
```

implicit
If the client certificate already contains a suitable Diffie-Hellman key, then Yc is implicit and does not need to be sent again. In this case, the Client Key Exchange message will be sent, but will be empty.

explicit
Yc needs to be sent.

```
struct {
    select (PublicValueEncoding) {
        case implicit: struct { };
        case explicit: opaque dh_Yc<1..2^16-1>;
    } dh_public;
} ClientDiffieHellmanPublic;
```

dh_Yc
The client's Diffie-Hellman public value (Yc).

7.4.8. Certificate verify

When this message will be sent:
This message is used to provide explicit verification of a client certificate. This message is only sent following a client certificate that has signing capability (i.e. all certificates except those containing fixed Diffie-Hellman parameters). When sent, it will immediately follow the client key exchange message.

Structure of this message:

```
struct {
    Signature signature;
} CertificateVerify;
```


The Signature type is defined in 7.4.3.

```
CertificateVerify.signature.md5_hash
    MD5(handshake_messages);

Certificate.signature.sha_hash
    SHA(handshake_messages);
```

Here handshake_messages refers to all handshake messages sent or received starting at client hello up to but not including this message, including the type and length fields of the handshake messages. This is the concatenation of all the Handshake structures as defined in 7.4 exchanged thus far.

7.4.9. Finished

When this message will be sent:
A finished message is always sent immediately after a change cipher spec message to verify that the key exchange and authentication processes were successful. It is essential that a change cipher spec message be received between the other handshake messages and the Finished message.

Meaning of this message:
The finished message is the first protected with the just-negotiated algorithms, keys, and secrets. Recipients of finished messages must verify that the contents are correct. Once a side has sent its Finished message and received and validated the Finished message from its peer, it may begin to send and receive application data over the connection.

```
struct {
    opaque verify_data[12];
} Finished;

verify_data
    PRF(master_secret, finished_label, MD5(handshake_messages) +
    SHA-1(handshake_messages)) [0..11];
```

finished_label
For Finished messages sent by the client, the string "client finished". For Finished messages sent by the server, the string "server finished".

handshake_messages
All of the data from all handshake messages up to but not including this message. This is only data visible at the handshake layer and does not include record layer headers.

This is the concatenation of all the Handshake structures as defined in 7.4 exchanged thus far.

It is a fatal error if a finished message is not preceded by a change cipher spec message at the appropriate point in the handshake.

The hash contained in finished messages sent by the server incorporate Sender.server; those sent by the client incorporate Sender.client. The value handshake_messages includes all handshake messages starting at client hello up to, but not including, this finished message. This may be different from handshake_messages in Section 7.4.8 because it would include the certificate verify message (if sent). Also, the handshake_messages for the finished message sent by the client will be different from that for the finished message sent by the server, because the one which is sent second will include the prior one.

Note: Change cipher spec messages, alerts and any other record types are not handshake messages and are not included in the hash computations. Also, Hello Request messages are omitted from handshake hashes.

8. Cryptographic computations

In order to begin connection protection, the TLS Record Protocol requires specification of a suite of algorithms, a master secret, and the client and server random values. The authentication, encryption, and MAC algorithms are determined by the cipher_suite selected by the server and revealed in the server hello message. The compression algorithm is negotiated in the hello messages, and the random values are exchanged in the hello messages. All that remains is to calculate the master secret.

8.1. Computing the master secret

For all key exchange methods, the same algorithm is used to convert the pre_master_secret into the master_secret. The pre_master_secret should be deleted from memory once the master_secret has been computed.

```
master_secret = PRF(pre_master_secret, "master secret",
                    ClientHello.random + ServerHello.random)
[0..47];
```

The master secret is always exactly 48 bytes in length. The length of the premaster secret will vary depending on key exchange method.

8.1.1. RSA

When RSA is used for server authentication and key exchange, a 48-byte pre_master_secret is generated by the client, encrypted under the server's public key, and sent to the server. The server uses its private key to decrypt the pre_master_secret. Both parties then convert the pre_master_secret into the master_secret, as specified above.

RSA digital signatures are performed using PKCS #1 [PKCS1] block type 1. RSA public key encryption is performed using PKCS #1 block type 2.

8.1.2. Diffie-Hellman

A conventional Diffie-Hellman computation is performed. The negotiated key (Z) is used as the pre_master_secret, and is converted into the master_secret, as specified above.

Note: Diffie-Hellman parameters are specified by the server, and may be either ephemeral or contained within the server's certificate.

9. Mandatory Cipher Suites

In the absence of an application profile standard specifying otherwise, a TLS compliant application MUST implement the cipher suite TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA.

10. Application data protocol

Application data messages are carried by the Record Layer and are fragmented, compressed and encrypted based on the current connection state. The messages are treated as transparent data to the record layer.

A. Protocol constant values

This section describes protocol types and constants.

A.1. Record layer

```
struct {
    uint8 major, minor;
} ProtocolVersion;

ProtocolVersion version = { 3, 1 };     /* TLS v1.0 */

enum {
    change_cipher_spec(20), alert(21), handshake(22),
    application_data(23), (255)
} ContentType;

struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
    opaque fragment[TLSPlaintext.length];
} TLSPlaintext;

struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
    opaque fragment[TLSCompressed.length];
} TLSCompressed;

struct {
```

```
        ContentType type;
        ProtocolVersion version;
        uint16 length;
        select (CipherSpec.cipher_type) {
            case stream: GenericStreamCipher;
            case block:  GenericBlockCipher;
        } fragment;
    } TLSCiphertext;

    stream-ciphered struct {
        opaque content[TLSCompressed.length];
        opaque MAC[CipherSpec.hash_size];
    } GenericStreamCipher;

    block-ciphered struct {
        opaque content[TLSCompressed.length];
```



```
        opaque MAC[CipherSpec.hash_size];
        uint8 padding[GenericBlockCipher.padding_length];
        uint8 padding_length;
    } GenericBlockCipher;
```

A.2. Change cipher specs message

```
    struct {
        enum { change_cipher_spec(1), (255) } type;
    } ChangeCipherSpec;
```

A.3. Alert messages

```
    enum { warning(1), fatal(2), (255) } AlertLevel;

        enum {
            close_notify(0),
            unexpected_message(10),
            bad_record_mac(20),
            decryption_failed(21),
            record_overflow(22),
            decompression_failure(30),
            handshake_failure(40),
            bad_certificate(42),
            unsupported_certificate(43),
            certificate_revoked(44),
            certificate_expired(45),
            certificate_unknown(46),
            illegal_parameter(47),
            unknown_ca(48),
            access_denied(49),
            decode_error(50),
            decrypt_error(51),
            export_restriction(60),
            protocol_version(70),
            insufficient_security(71),
            internal_error(80),
            user_canceled(90),
            no_renegotiation(100),
            (255)
        } AlertDescription;

    struct {
        AlertLevel level;
        AlertDescription description;
    } Alert;
```


A.4. Handshake protocol

```
    enum {
        hello_request(0), client_hello(1), server_hello(2),
        certificate(11), server_key_exchange (12),
        certificate_request(13), server_hello_done(14),
        certificate_verify(15), client_key_exchange(16),
        finished(20), (255)
    } HandshakeType;

    struct {
        HandshakeType msg_type;
        uint24 length;
        select (HandshakeType) {
            case hello_request:       HelloRequest;
            case client_hello:        ClientHello;
            case server_hello:        ServerHello;
            case certificate:         Certificate;
            case server_key_exchange: ServerKeyExchange;
            case certificate_request: CertificateRequest;
            case server_hello_done:   ServerHelloDone;
            case certificate_verify:  CertificateVerify;
            case client_key_exchange: ClientKeyExchange;
            case finished:            Finished;
        } body;
    } Handshake;
```

A.4.1. Hello messages

```
    struct { } HelloRequest;
```

```
    struct {
        uint32 gmt_unix_time;
        opaque random_bytes[28];
    } Random;

    opaque SessionID<0..32>;

    uint8 CipherSuite[2];

    enum { null(0), (255) } CompressionMethod;

    struct {
        ProtocolVersion client_version;
        Random random;
        SessionID session_id;
        CipherSuite cipher_suites<2..2^16-1>;
        CompressionMethod compression_methods<1..2^8-1>;
```



```
    } ClientHello;

    struct {
        ProtocolVersion server_version;
        Random random;
        SessionID session_id;
        CipherSuite cipher_suite;
        CompressionMethod compression_method;
    } ServerHello;
```

A.4.2. Server authentication and key exchange messages

```
    opaque ASN.1Cert<2^24-1>;

    struct {
        ASN.1Cert certificate_list<1..2^24-1>;
    } Certificate;

    enum { rsa, diffie_hellman } KeyExchangeAlgorithm;

    struct {
        opaque RSA_modulus<1..2^16-1>;
        opaque RSA_exponent<1..2^16-1>;
    } ServerRSAParams;

    struct {
        opaque DH_p<1..2^16-1>;
        opaque DH_g<1..2^16-1>;
        opaque DH_Ys<1..2^16-1>;
    } ServerDHParams;

    struct {
        select (KeyExchangeAlgorithm) {
            case diffie_hellman:
                ServerDHParams params;
                Signature signed_params;
            case rsa:
                ServerRSAParams params;
                Signature signed_params;
        };
    } ServerKeyExchange;

    enum { anonymous, rsa, dsa } SignatureAlgorithm;

    select (SignatureAlgorithm)
    {   case anonymous: struct { };
        case rsa:
            digitally-signed struct {
```



```
                opaque md5_hash[16];
                opaque sha_hash[20];
            };
        case dsa:
            digitally-signed struct {
                opaque sha_hash[20];
            };
    } Signature;

    enum {
        rsa_sign(1), dss_sign(2), rsa_fixed_dh(3), dss_fixed_dh(4),
        (255)
    } ClientCertificateType;

    opaque DistinguishedName<1..2^16-1>;

    struct {
        ClientCertificateType certificate_types<1..2^8-1>;
        DistinguishedName certificate_authorities<3..2^16-1>;
    } CertificateRequest;

    struct { } ServerHelloDone;
```

A.4.3. Client authentication and key exchange messages

```
    struct {
        select (KeyExchangeAlgorithm) {
```

```
            case rsa: EncryptedPreMasterSecret;
            case diffie_hellman: DiffieHellmanClientPublicValue;
        } exchange_keys;
    } ClientKeyExchange;

    struct {
        ProtocolVersion client_version;
        opaque random[46];

    } PreMasterSecret;

    struct {
        public-key-encrypted PreMasterSecret pre_master_secret;
    } EncryptedPreMasterSecret;

    enum { implicit, explicit } PublicValueEncoding;

    struct {
        select (PublicValueEncoding) {
            case implicit: struct {};
            case explicit: opaque DH_Yc<1..2^16-1>;
```



```
        } dh_public;
    } ClientDiffieHellmanPublic;

    struct {
        Signature signature;
    } CertificateVerify;
```

A.4.4. Handshake finalization message

```
    struct {
        opaque verify_data[12];
    } Finished;
```

A.5. The CipherSuite

The following values define the CipherSuite codes used in the client hello and server hello messages.

A CipherSuite defines a cipher specification supported in TLS Version 1.0.

TLS_NULL_WITH_NULL_NULL is specified and is the initial state of a TLS connection during the first handshake on that channel, but must not be negotiated, as it provides no more protection than an unsecured connection.

```
 CipherSuite TLS_NULL_WITH_NULL_NULL               = { 0x00,0x00 };
```

The following CipherSuite definitions require that the server provide an RSA certificate that can be used for key exchange. The server may request either an RSA or a DSS signature-capable certificate in the certificate request message.

```
 CipherSuite TLS_RSA_WITH_NULL_MD5                 = { 0x00,0x01 };
 CipherSuite TLS_RSA_WITH_NULL_SHA                 = { 0x00,0x02 };
 CipherSuite TLS_RSA_EXPORT_WITH_RC4_40_MD5        = { 0x00,0x03 };
 CipherSuite TLS_RSA_WITH_RC4_128_MD5              = { 0x00,0x04 };
 CipherSuite TLS_RSA_WITH_RC4_128_SHA              = { 0x00,0x05 };
 CipherSuite TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5    = { 0x00,0x06 };
 CipherSuite TLS_RSA_WITH_IDEA_CBC_SHA             = { 0x00,0x07 };
 CipherSuite TLS_RSA_EXPORT_WITH_DES40_CBC_SHA     = { 0x00,0x08 };
 CipherSuite TLS_RSA_WITH_DES_CBC_SHA              = { 0x00,0x09 };
 CipherSuite TLS_RSA_WITH_3DES_EDE_CBC_SHA         = { 0x00,0x0A };
```

The following CipherSuite definitions are used for server-authenticated (and optionally client-authenticated) Diffie-Hellman. DH denotes cipher suites in which the server's certificate contains the Diffie-Hellman parameters signed by the certificate authority

(CA). DHE denotes ephemeral Diffie-Hellman, where the Diffie-Hellman parameters are signed by a DSS or RSA certificate, which has been signed by the CA. The signing algorithm used is specified after the DH or DHE parameter. The server can request an RSA or DSS signature-capable certificate from the client for client authentication or it may request a Diffie-Hellman certificate. Any Diffie-Hellman certificate provided by the client must use the parameters (group and generator) described by the server.

```
 CipherSuite TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA  = { 0x00,0x0B };
 CipherSuite TLS_DH_DSS_WITH_DES_CBC_SHA           = { 0x00,0x0C };
 CipherSuite TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA      = { 0x00,0x0D };
 CipherSuite TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA  = { 0x00,0x0E };
 CipherSuite TLS_DH_RSA_WITH_DES_CBC_SHA           = { 0x00,0x0F };
 CipherSuite TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA      = { 0x00,0x10 };
 CipherSuite TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA = { 0x00,0x11 };
 CipherSuite TLS_DHE_DSS_WITH_DES_CBC_SHA          = { 0x00,0x12 };
 CipherSuite TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA     = { 0x00,0x13 };
 CipherSuite TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA = { 0x00,0x14 };
 CipherSuite TLS_DHE_RSA_WITH_DES_CBC_SHA          = { 0x00,0x15 };
 CipherSuite TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA     = { 0x00,0x16 };
```

The following cipher suites are used for completely anonymous Diffie-Hellman communications in which neither party is authenticated. Note that this mode is vulnerable to man-in-the-middle

attacks and is therefore deprecated.

```
   CipherSuite TLS_DH_anon_EXPORT_WITH_RC4_40_MD5     = { 0x00,0x17 };
   CipherSuite TLS_DH_anon_WITH_RC4_128_MD5           = { 0x00,0x18 };
   CipherSuite TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA  = { 0x00,0x19 };
   CipherSuite TLS_DH_anon_WITH_DES_CBC_SHA           = { 0x00,0x1A };
   CipherSuite TLS_DH_anon_WITH_3DES_EDE_CBC_SHA      = { 0x00,0x1B };
```

Note: All cipher suites whose first byte is 0xFF are considered private and can be used for defining local/experimental algorithms. Interoperability of such types is a local matter.

Note: Additional cipher suites can be registered by publishing an RFC which specifies the cipher suites, including the necessary TLS protocol information, including message encoding, premaster secret derivation, symmetric encryption and MAC calculation and appropriate reference information for the algorithms involved. The RFC editor's office may, at its discretion, choose to publish specifications for cipher suites which are not completely described (e.g., for classified algorithms) if it finds the specification to be of technical interest and completely specified.

Note: The cipher suite values { 0x00, 0x1C } and { 0x00, 0x1D } are reserved to avoid collision with Fortezza-based cipher suites in SSL 3.

A.6. The Security Parameters

These security parameters are determined by the TLS Handshake Protocol and provided as parameters to the TLS Record Layer in order to initialize a connection state. SecurityParameters includes:

```
      enum { null(0), (255) } CompressionMethod;

      enum { server, client } ConnectionEnd;

      enum { null, rc4, rc2, des, 3des, des40, idea }
      BulkCipherAlgorithm;

      enum { stream, block } CipherType;

      enum { true, false } IsExportable;

      enum { null, md5, sha } MACAlgorithm;

   /* The algorithms specified in CompressionMethod,
   BulkCipherAlgorithm, and MACAlgorithm may be added to. */

      struct {
          ConnectionEnd entity;
          BulkCipherAlgorithm bulk_cipher_algorithm;
          CipherType cipher_type;
          uint8 key_size;
          uint8 key_material_length;
          IsExportable is_exportable;
          MACAlgorithm mac_algorithm;
          uint8 hash_size;
          CompressionMethod compression_algorithm;
          opaque master_secret[48];
          opaque client_random[32];
          opaque server_random[32];
      } SecurityParameters;
```


B. Glossary

application protocol
An application protocol is a protocol that normally layers directly on top of the transport layer (e.g., TCP/IP). Examples include HTTP, TELNET, FTP, and SMTP.

asymmetric cipher
See public key cryptography.

authentication
Authentication is the ability of one entity to determine the identity of another entity.

block cipher
A block cipher is an algorithm that operates on plaintext in groups of bits, called blocks. 64 bits is a common block size.

bulk cipher
A symmetric encryption algorithm used to encrypt large quantities of data.

cipher block chaining (CBC)
CBC is a mode in which every plaintext block encrypted with a block cipher is first exclusive-ORed with the previous ciphertext block (or, in the case of the first block, with the initialization vector). For decryption, every block is first decrypted, then exclusive-ORed with the previous ciphertext block (or IV).

certificate
As part of the X.509 protocol (a.k.a. ISO Authentication framework), certificates are assigned by a trusted Certificate Authority and provide a strong binding between a party's identity or some other attributes and its public key.

client
The application entity that initiates a TLS connection to a server. This may or may not imply that the client initiated the underlying transport connection. The primary operational difference between the server and client is that the server is generally authenticated, while the client is only optionally authenticated.

client write key
The key used to encrypt data written by the client.

client write MAC secret
The secret data used to authenticate data written by the client.

connection
A connection is a transport (in the OSI layering model definition) that provides a suitable type of service. For TLS, such connections are peer to peer relationships. The connections are transient. Every connection is associated with one session.

Data Encryption Standard
DES is a very widely used symmetric encryption algorithm. DES is a block cipher with a 56 bit key and an 8 byte block size. Note that in TLS, for key generation purposes, DES is treated as having an 8 byte key length (64 bits), but it still only provides 56 bits of protection. (The low bit of each key byte is presumed to be set to produce odd parity in that key byte.) DES can also be operated in a mode where three independent keys and three encryptions are used for each block of data; this uses 168 bits of key (24 bytes in the TLS key generation method) and provides the equivalent of 112 bits of security. [DES], [3DES]

Digital Signature Standard (DSS)
A standard for digital signing, including the Digital Signing Algorithm, approved by the National Institute of Standards and Technology, defined in NIST FIPS PUB 186, "Digital Signature Standard," published May, 1994 by the U.S. Dept. of Commerce. [DSS]

digital signatures
Digital signatures utilize public key cryptography and one-way hash functions to produce a signature of the data that can be authenticated, and is difficult to forge or repudiate.

handshake
An initial negotiation between client and server that establishes the parameters of their transactions.

Initialization Vector (IV)
When a block cipher is used in CBC mode, the initialization vector is exclusive-ORed with the first plaintext block prior to encryption.

IDEA
A 64-bit block cipher designed by Xuejia Lai and James Massey. [IDEA]

Message Authentication Code (MAC)
A Message Authentication Code is a one-way hash computed from a message and some secret data. It is difficult to forge without knowing the secret data. Its purpose is to detect if the message has been altered.

master secret
Secure secret data used for generating encryption keys, MAC secrets, and IVs.

MD5
MD5 is a secure hashing function that converts an arbitrarily long data stream into a digest of fixed size (16 bytes). [MD5]

public key cryptography
A class of cryptographic techniques employing two-key ciphers. Messages encrypted with the public key can only be decrypted with the associated private key. Conversely, messages signed with the private key can be verified with the public key.

one-way hash function
A one-way transformation that converts an arbitrary amount of data into a fixed-length hash. It is computationally hard to reverse the transformation or to find collisions. MD5 and SHA are examples of one-way hash functions.

RC2
A block cipher developed by Ron Rivest at RSA Data Security, Inc. [RSADSI] described in [RC2].

RC4
A stream cipher licensed by RSA Data Security [RSADSI]. A compatible cipher is described in [RC4].

RSA
A very widely used public-key algorithm that can be used for either encryption or digital signing. [RSA]

salt
Non-secret random data used to make export encryption keys resist precomputation attacks.

server
The server is the application entity that responds to requests for connections from clients. See also under client.

session
A TLS session is an association between a client and a server. Sessions are created by the handshake protocol. Sessions define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

session identifier
A session identifier is a value generated by a server that identifies a particular session.

server write key
The key used to encrypt data written by the server.

server write MAC secret
The secret data used to authenticate data written by the server.

SHA
The Secure Hash Algorithm is defined in FIPS PUB 180-1. It produces a 20-byte output. Note that all references to SHA actually use the modified SHA-1 algorithm. [SHA]

SSL
Netscape's Secure Socket Layer protocol [SSL3]. TLS is based on SSL Version 3.0

stream cipher
An encryption algorithm that converts a key into a cryptographically-strong keystream, which is then exclusive-ORed with the plaintext.

symmetric cipher
See bulk cipher.

Transport Layer Security (TLS)
This protocol; also, the Transport Layer Security working group of the Internet Engineering Task Force (IETF). See "Comments" at the end of this document.

C. CipherSuite definitions

CipherSuite	Is Exportable	Key Exchange	Cipher	Hash
TLS_NULL_WITH_NULL_NULL	*	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	*	RSA	NULL	MD5
TLS_RSA_WITH_NULL_SHA	*	RSA	NULL	SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5	*	RSA_EXPORT	RC4_40	MD5
TLS_RSA_WITH_RC4_128_MD5		RSA	RC4_128	MD5
TLS_RSA_WITH_RC4_128_SHA		RSA	RC4_128	SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	*	RSA_EXPORT	RC2_CBC_40	MD5
TLS_RSA_WITH_IDEA_CBC_SHA		RSA	IDEA_CBC	SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA	*	RSA_EXPORT	DES40_CBC	SHA
TLS_RSA_WITH_DES_CBC_SHA		RSA	DES_CBC	SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA		RSA	3DES_EDE_CBC	SHA
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	*	DH_DSS_EXPORT	DES40_CBC	SHA

TLS_DH_DSS_WITH_DES_CBC_SHA		DH_DSS	DES_CBC	SHA
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA		DH_DSS	3DES_EDE_CBC	SHA
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	*	DH_RSA_EXPORT	DES40_CBC	SHA
TLS_DH_RSA_WITH_DES_CBC_SHA		DH_RSA	DES_CBC	SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA		DH_RSA	3DES_EDE_CBC	SHA
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	*	DHE_DSS_EXPORT	DES40_CBC	SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA		DHE_DSS	DES_CBC	SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA		DHE_DSS	3DES_EDE_CBC	SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	*	DHE_RSA_EXPORT	DES40_CBC	SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA		DHE_RSA	DES_CBC	SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA		DHE_RSA	3DES_EDE_CBC	SHA
TLS_DH_anon_EXPORT_WITH_RC4_40_MD5	*	DH_anon_EXPORT	RC4_40	MD5
TLS_DH_anon_WITH_RC4_128_MD5		DH_anon	RC4_128	MD5
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA		DH_anon	DES40_CBC	SHA
TLS_DH_anon_WITH_DES_CBC_SHA		DH_anon	DES_CBC	SHA
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA		DH_anon	3DES_EDE_CBC	SHA

* Indicates IsExportable is True

Key Exchange Algorithm	Description	Key size limit
DHE_DSS	Ephemeral DH with DSS signatures	None
DHE_DSS_EXPORT	Ephemeral DH with DSS signatures	DH = 512 bits
DHE_RSA	Ephemeral DH with RSA signatures	None
DHE_RSA_EXPORT	Ephemeral DH with RSA signatures	DH = 512 bits, RSA = none
DH_anon	Anonymous DH, no signatures	None
DH_anon_EXPORT	Anonymous DH, no signatures	DH = 512 bits

DH_DSS	DH with DSS-based certificates	None
DH_DSS_EXPORT	DH with DSS-based certificates	DH = 512 bits
DH_RSA	DH with RSA-based certificates	None
DH_RSA_EXPORT	DH with RSA-based certificates	DH = 512 bits, RSA = none
NULL	No key exchange	N/A
RSA	RSA key exchange	None
RSA_EXPORT	RSA key exchange	RSA = 512 bits

Key size limit
The key size limit gives the size of the largest public key that can be legally used for encryption in cipher suites that are exportable.

Cipher		Type	Key Material	Expanded Key Material	Effective Key Bits	IV Size	Block Size
NULL	*	Stream	0	0	0	0	N/A
IDEA_CBC		Block	16	16	128	8	8
RC2_CBC_40	*	Block	5	16	40	8	8
RC4_40	*	Stream	5	16	40	0	N/A
RC4_128		Stream	16	16	128	0	N/A
DES40_CBC	*	Block	5	8	40	8	8
DES_CBC		Block	8	8	56	8	8
3DES_EDE_CBC		Block	24	24	168	8	8

* Indicates IsExportable is true.

Type
Indicates whether this is a stream cipher or a block cipher running in CBC mode.

Key Material
The number of bytes from the key_block that are used for generating the write keys.

Expanded Key Material
The number of bytes actually fed into the encryption algorithm

Effective Key Bits
How much entropy material is in the key material being fed into the encryption routines.

IV Size
How much data needs to be generated for the initialization vector. Zero for stream ciphers; equal to the block size for block ciphers.

Block Size
The amount of data a block cipher enciphers in one chunk; a block cipher running in CBC mode can only encrypt an even multiple of its block size.

Hash function	Hash Size	Padding Size
NULL	0	0
MD5	16	48
SHA	20	40

D. Implementation Notes

The TLS protocol cannot prevent many common security mistakes. This section provides several recommendations to assist implementors.

D.1. Temporary RSA keys

US Export restrictions limit RSA keys used for encryption to 512 bits, but do not place any limit on lengths of RSA keys used for signing operations. Certificates often need to be larger than 512 bits, since 512-bit RSA keys are not secure enough for high-value transactions or for applications requiring long-term security. Some certificates are also designated signing-only, in which case they cannot be used for key exchange.

When the public key in the certificate cannot be used for encryption, the server signs a temporary RSA key, which is then exchanged. In exportable applications, the temporary RSA key should be the maximum allowable length (i.e., 512 bits). Because 512-bit RSA keys are relatively insecure, they should be changed often. For typical electronic commerce applications, it is suggested that keys be changed daily or every 500 transactions, and more often if possible. Note that while it is acceptable to use the same temporary key for multiple transactions, it must be signed each time it is used.

RSA key generation is a time-consuming process. In many cases, a low-priority process can be assigned the task of key generation.

Whenever a new key is completed, the existing temporary key can be replaced with the new one.

D.2. Random Number Generation and Seeding

TLS requires a cryptographically-secure pseudorandom number generator (PRNG). Care must be taken in designing and seeding PRNGs. PRNGs based on secure hash operations, most notably MD5 and/or SHA, are acceptable, but cannot provide more security than the size of the random number generator state. (For example, MD5-based PRNGs usually provide 128 bits of state.)

To estimate the amount of seed material being produced, add the number of bits of unpredictable information in each seed byte. For example, keystroke timing values taken from a PC compatible's 18.2 Hz timer provide 1 or 2 secure bits each, even though the total size of the counter value is 16 bits or more. To seed a 128-bit PRNG, one would thus require approximately 100 such timer values.

Warning: The seeding functions in RSAREF and versions of BSAFE prior to 3.0 are order-independent. For example, if 1000 seed bits are supplied, one at a time, in 1000 separate calls to the seed function, the PRNG will end up in a state which depends only on the number of 0 or 1 seed bits in the seed data (i.e., there are 1001 possible final states). Applications using BSAFE or RSAREF must take extra care to ensure proper seeding. This may be accomplished by accumulating seed bits into a buffer and processing them all at once or by processing an incrementing counter with every seed bit; either method will reintroduce order dependence into the seeding process.

D.3. Certificates and authentication

Implementations are responsible for verifying the integrity of certificates and should generally support certificate revocation messages. Certificates should always be verified to ensure proper signing by a trusted Certificate Authority (CA). The selection and addition of trusted CAs should be done very carefully. Users should be able to view information about the certificate and root CA.

D.4. CipherSuites

TLS supports a range of key sizes and security levels, including some which provide no or minimal security. A proper implementation will probably not support many cipher suites. For example, 40-bit encryption is easily broken, so implementations requiring strong security should not allow 40-bit keys. Similarly, anonymous Diffie-Hellman is strongly discouraged because it cannot prevent man-in-the-middle attacks. Applications should also enforce minimum and maximum key sizes. For example, certificate chains containing 512-bit RSA keys or signatures are not appropriate for high-security applications.

E. Backward Compatibility With SSL

For historical reasons and in order to avoid a profligate consumption of reserved port numbers, application protocols which are secured by TLS 1.0, SSL 3.0, and SSL 2.0 all frequently share the same connection port: for example, the https protocol (HTTP secured by SSL or TLS) uses port 443 regardless of which security protocol it is using. Thus, some mechanism must be determined to distinguish and negotiate among the various protocols.

TLS version 1.0 and SSL 3.0 are very similar; thus, supporting both is easy. TLS clients who wish to negotiate with SSL 3.0 servers should send client hello messages using the SSL 3.0 record format and client hello structure, sending {3, 1} for the version field to note that they support TLS 1.0. If the server supports only SSL 3.0, it will respond with an SSL 3.0 server hello; if it supports TLS, with a TLS server hello. The negotiation then proceeds as appropriate for the negotiated protocol.

Similarly, a TLS server which wishes to interoperate with SSL 3.0 clients should accept SSL 3.0 client hello messages and respond with an SSL 3.0 server hello if an SSL 3.0 client hello is received which has a version field of {3, 0}, denoting that this client does not support TLS.

Whenever a client already knows the highest protocol known to a server (for example, when resuming a session), it should initiate the connection in that native protocol.

TLS 1.0 clients that support SSL Version 2.0 servers must send SSL Version 2.0 client hello messages [SSL2]. TLS servers should accept either client hello format if they wish to support SSL 2.0 clients on the same connection port. The only deviations from the Version 2.0 specification are the ability to specify a version with a value of three and the support for more ciphering types in the CipherSpec.

Warning: The ability to send Version 2.0 client hello messages will be phased out with all due haste. Implementors should make every effort to move forward as quickly as possible. Version 3.0 provides better mechanisms for moving to newer versions.

The following cipher specifications are carryovers from SSL Version 2.0. These are assumed to use RSA for key exchange and authentication.

```
V2CipherSpec TLS_RC4_128_WITH_MD5          = { 0x01,0x00,0x80 };
V2CipherSpec TLS_RC4_128_EXPORT40_WITH_MD5 = { 0x02,0x00,0x80 };
V2CipherSpec TLS_RC2_CBC_128_CBC_WITH_MD5  = { 0x03,0x00,0x80 };
```



```
V2CipherSpec TLS_RC2_CBC_128_CBC_EXPORT40_WITH_MD5
                                           = { 0x04,0x00,0x80 };
V2CipherSpec TLS_IDEA_128_CBC_WITH_MD5     = { 0x05,0x00,0x80 };
V2CipherSpec TLS_DES_64_CBC_WITH_MD5       = { 0x06,0x00,0x40 };
V2CipherSpec TLS_DES_192_EDE3_CBC_WITH_MD5 = { 0x07,0x00,0xC0 };
```

Cipher specifications native to TLS can be included in Version 2.0 client hello messages using the syntax below. Any V2CipherSpec element with its first byte equal to zero will be ignored by Version

2.0 servers. Clients sending any of the above V2CipherSpecs should also include the TLS equivalent (see Appendix A.5):

```
V2CipherSpec (see TLS name) = { 0x00, CipherSuite };
```

E.1. Version 2 client hello

The Version 2.0 client hello message is presented below using this document's presentation model. The true definition is still assumed to be the SSL Version 2.0 specification.

```
uint8 V2CipherSpec[3];

struct {
    uint8 msg_type;
    Version version;
    uint16 cipher_spec_length;
    uint16 session_id_length;
    uint16 challenge_length;
    V2CipherSpec cipher_specs[V2ClientHello.cipher_spec_length];
    opaque session_id[V2ClientHello.session_id_length];
    Random challenge;
} V2ClientHello;
```

msg_type
: This field, in conjunction with the Version field, identifies a version 2 client hello message. The value should be one (1).

version
: The highest version of the protocol supported by the client (equals ProtocolVersion.version, see Appendix A.1).

cipher_spec_length
: This field is the total length of the field cipher_specs. It cannot be zero and must be a multiple of the V2CipherSpec length (3).

session_id_length
: This field must have a value of either zero or 16. If zero, the client is creating a new session. If 16, the session_id field will contain the 16 bytes of session identification.

challenge_length
: The length in bytes of the client's challenge to the server to authenticate itself. This value must be 32.

cipher_specs
: This is a list of all CipherSpecs the client is willing and able to use. There must be at least one CipherSpec acceptable to the server.

session_id
: If this field's length is not zero, it will contain the identification for a session that the client wishes to resume.

challenge
: The client challenge to the server for the server to identify itself is a (nearly) arbitrary length random. The TLS server will right justify the challenge data to become the ClientHello.random data (padded with leading zeroes, if necessary), as specified in this protocol specification. If the length of the challenge is greater than 32 bytes, only the last 32 bytes are used. It is legitimate (but not necessary) for a V3 server to reject a V2 ClientHello that has fewer than 16 bytes of challenge data.

Note: Requests to resume a TLS session should use a TLS client hello.

E.2. Avoiding man-in-the-middle version rollback

When TLS clients fall back to Version 2.0 compatibility mode, they should use special PKCS #1 block formatting. This is done so that TLS servers will reject Version 2.0 sessions with TLS-capable clients.

When TLS clients are in Version 2.0 compatibility mode, they set the right-hand (least-significant) 8 random bytes of the PKCS padding (not including the terminal null of the padding) for the RSA encryption of the ENCRYPTED-KEY-DATA field of the CLIENT-MASTER-KEY to 0x03 (the other padding bytes are random). After decrypting the ENCRYPTED-KEY-DATA field, servers that support TLS should issue an error if these eight padding bytes are 0x03. Version 2.0 servers receiving blocks padded in this manner will proceed normally.

F. Security analysis

The TLS protocol is designed to establish a secure connection between a client and a server communicating over an insecure channel. This document makes several traditional assumptions, including that attackers have substantial computational resources and cannot obtain

secret information from sources outside the protocol. Attackers are assumed to have the ability to capture, modify, delete, replay, and otherwise tamper with messages sent over the communication channel. This appendix outlines how TLS has been designed to resist a variety of attacks.

F.1. Handshake protocol

The handshake protocol is responsible for selecting a CipherSpec and generating a Master Secret, which together comprise the primary cryptographic parameters associated with a secure session. The handshake protocol can also optionally authenticate parties who have certificates signed by a trusted certificate authority.

F.1.1. Authentication and key exchange

TLS supports three authentication modes: authentication of both parties, server authentication with an unauthenticated client, and total anonymity. Whenever the server is authenticated, the channel is secure against man-in-the-middle attacks, but completely anonymous sessions are inherently vulnerable to such attacks. Anonymous servers cannot authenticate clients. If the server is authenticated, its certificate message must provide a valid certificate chain leading to an acceptable certificate authority. Similarly, authenticated clients must supply an acceptable certificate to the server. Each party is responsible for verifying that the other's certificate is valid and has not expired or been revoked.

The general goal of the key exchange process is to create a pre_master_secret known to the communicating parties and not to attackers. The pre_master_secret will be used to generate the master_secret (see Section 8.1). The master_secret is required to generate the certificate verify and finished messages, encryption keys, and MAC secrets (see Sections 7.4.8, 7.4.9 and 6.3). By sending a correct finished message, parties thus prove that they know the correct pre_master_secret.

F.1.1.1. Anonymous key exchange

Completely anonymous sessions can be established using RSA or Diffie-Hellman for key exchange. With anonymous RSA, the client encrypts a pre_master_secret with the server's uncertified public key

extracted from the server key exchange message. The result is sent in a client key exchange message. Since eavesdroppers do not know the server's private key, it will be infeasible for them to decode the pre_master_secret. (Note that no anonymous RSA Cipher Suites are defined in this document).

With Diffie-Hellman, the server's public parameters are contained in the server key exchange message and the client's are sent in the client key exchange message. Eavesdroppers who do not know the private values should not be able to find the Diffie-Hellman result (i.e. the pre_master_secret).

Warning: Completely anonymous connections only provide protection against passive eavesdropping. Unless an independent tamper-proof channel is used to verify that the finished messages were not replaced by an attacker, server authentication is required in environments where active man-in-the-middle attacks are a concern.

F.1.1.2. RSA key exchange and authentication

With RSA, key exchange and server authentication are combined. The public key may be either contained in the server's certificate or may be a temporary RSA key sent in a server key exchange message. When temporary RSA keys are used, they are signed by the server's RSA or DSS certificate. The signature includes the current ClientHello.random, so old signatures and temporary keys cannot be replayed. Servers may use a single temporary RSA key for multiple negotiation sessions.

Note: The temporary RSA key option is useful if servers need large certificates but must comply with government-imposed size limits on keys used for key exchange.

After verifying the server's certificate, the client encrypts a pre_master_secret with the server's public key. By successfully decoding the pre_master_secret and producing a correct finished message, the server demonstrates that it knows the private key corresponding to the server certificate.

When RSA is used for key exchange, clients are authenticated using the certificate verify message (see Section 7.4.8). The client signs a value derived from the master_secret and all preceding handshake messages. These handshake messages include the server certificate, which binds the signature to the server, and ServerHello.random, which binds the signature to the current handshake process.

F.1.1.3. Diffie-Hellman key exchange with authentication

When Diffie-Hellman key exchange is used, the server can either

supply a certificate containing fixed Diffie-Hellman parameters or can use the server key exchange message to send a set of temporary Diffie-Hellman parameters signed with a DSS or RSA certificate. Temporary parameters are hashed with the hello.random values before signing to ensure that attackers do not replay old parameters. In either case, the client can verify the certificate or signature to ensure that the parameters belong to the server.

If the client has a certificate containing fixed Diffie-Hellman parameters, its certificate contains the information required to complete the key exchange. Note that in this case the client and server will generate the same Diffie-Hellman result (i.e., pre_master_secret) every time they communicate. To prevent the pre_master_secret from staying in memory any longer than necessary, it should be converted into the master_secret as soon as possible. Client Diffie-Hellman parameters must be compatible with those supplied by the server for the key exchange to work.

If the client has a standard DSS or RSA certificate or is unauthenticated, it sends a set of temporary parameters to the server in the client key exchange message, then optionally uses a certificate verify message to authenticate itself.

F.1.2. Version rollback attacks

Because TLS includes substantial improvements over SSL Version 2.0, attackers may try to make TLS-capable clients and servers fall back to Version 2.0. This attack can occur if (and only if) two TLS-capable parties use an SSL 2.0 handshake.

Although the solution using non-random PKCS #1 block type 2 message padding is inelegant, it provides a reasonably secure way for Version 3.0 servers to detect the attack. This solution is not secure against attackers who can brute force the key and substitute a new ENCRYPTED-KEY-DATA message containing the same key (but with normal padding) before the application specified wait threshold has expired. Parties concerned about attacks of this scale should not be using 40-bit encryption keys anyway. Altering the padding of the least-significant 8 bytes of the PKCS padding does not impact security for the size of the signed hashes and RSA key lengths used in the protocol, since this is essentially equivalent to increasing the input block size by 8 bytes.

F.1.3. Detecting attacks against the handshake protocol

An attacker might try to influence the handshake exchange to make the parties select different encryption algorithms than they would normally choose. Because many implementations will support 40-bit exportable encryption and some may even support null encryption or MAC algorithms, this attack is of particular concern.

For this attack, an attacker must actively change one or more handshake messages. If this occurs, the client and server will compute different values for the handshake message hashes. As a result, the parties will not accept each others' finished messages. Without the master_secret, the attacker cannot repair the finished messages, so the attack will be discovered.

F.1.4. Resuming sessions

When a connection is established by resuming a session, new ClientHello.random and ServerHello.random values are hashed with the session's master_secret. Provided that the master_secret has not been compromised and that the secure hash operations used to produce the encryption keys and MAC secrets are secure, the connection should be secure and effectively independent from previous connections. Attackers cannot use known encryption keys or MAC secrets to compromise the master_secret without breaking the secure hash operations (which use both SHA and MD5).

Sessions cannot be resumed unless both the client and server agree. If either party suspects that the session may have been compromised, or that certificates may have expired or been revoked, it should force a full handshake. An upper limit of 24 hours is suggested for session ID lifetimes, since an attacker who obtains a master_secret may be able to impersonate the compromised party until the corresponding session ID is retired. Applications that may be run in relatively insecure environments should not write session IDs to stable storage.

F.1.5. MD5 and SHA

TLS uses hash functions very conservatively. Where possible, both MD5 and SHA are used in tandem to ensure that non-catastrophic flaws in one algorithm will not break the overall protocol.

F.2. Protecting application data

The master_secret is hashed with the ClientHello.random and ServerHello.random to produce unique data encryption keys and MAC secrets for each connection.

Outgoing data is protected with a MAC before transmission. To prevent message replay or modification attacks, the MAC is computed from the MAC secret, the sequence number, the message length, the message contents, and two fixed character strings. The message type field is necessary to ensure that messages intended for one TLS Record Layer client are not redirected to another. The sequence number ensures that attempts to delete or reorder messages will be detected. Since sequence numbers are 64-bits long, they should never overflow. Messages from one party cannot be inserted into the other's output, since they use independent MAC secrets. Similarly, the server-write and client-write keys are independent so stream cipher keys are used only once.

If an attacker does break an encryption key, all messages encrypted with it can be read. Similarly, compromise of a MAC key can make message modification attacks possible. Because MACs are also encrypted, message-alteration attacks generally require breaking the encryption algorithm as well as the MAC.

Note: MAC secrets may be larger than encryption keys, so messages can remain tamper resistant even if encryption keys are broken.

F.3. Final notes

For TLS to be able to provide a secure connection, both the client and server systems, keys, and applications must be secure. In addition, the implementation must be free of security errors.

The system is only as strong as the weakest key exchange and authentication algorithm supported, and only trustworthy cryptographic functions should be used. Short public keys, 40-bit bulk encryption keys, and anonymous servers should be used with great caution. Implementations and users must be careful when deciding which certificates and certificate authorities are acceptable; a dishonest certificate authority can do tremendous damage.

G. Patent Statement

Some of the cryptographic algorithms proposed for use in this protocol have patent claims on them. In addition Netscape Communications Corporation has a patent claim on the Secure Sockets Layer (SSL) work that this standard is based on. The Internet Standards Process as defined in RFC 2026 requests that a statement be obtained from a Patent holder indicating that a license will be made available to applicants under reasonable terms and conditions.

The Massachusetts Institute of Technology has granted RSA Data Security, Inc., exclusive sub-licensing rights to the following patent issued in the United States:

> Cryptographic Communications System and Method ("RSA"), No. 4,405,829

Netscape Communications Corporation has been issued the following patent in the United States:

> Secure Socket Layer Application Program Apparatus And Method ("SSL"), No. 5,657,390

Netscape Communications has issued the following statement:

> Intellectual Property Rights
>
> Secure Sockets Layer
>
> The United States Patent and Trademark Office ("the PTO") recently issued U.S. Patent No. 5,657,390 ("the SSL Patent") to Netscape for inventions described as Secure Sockets Layers ("SSL"). The IETF is currently considering adopting SSL as a transport protocol with security features. Netscape encourages the royalty-free adoption and use of the SSL protocol upon the following terms and conditions:
>
> * If you already have a valid SSL Ref license today which includes source code from Netscape, an additional patent license under the SSL patent is not required.
>
> * If you don't have an SSL Ref license, you may have a royalty free license to build implementations covered by the SSL Patent Claims or the IETF TLS specification provided that you do not to assert any patent rights against Netscape or other companies for the implementation of SSL or the IETF TLS recommendation.

What are "Patent Claims":

Patent claims are claims in an issued foreign or domestic patent that:

1) must be infringed in order to implement methods or build products according to the IETF TLS specification; or

2) patent claims which require the elements of the SSL patent claims and/or their equivalents to be infringed.

The Internet Society, Internet Architecture Board, Internet Engineering Steering Group and the Corporation for National Research Initiatives take no position on the validity or scope of the patents and patent applications, nor on the appropriateness of the terms of the assurance. The Internet Society and other groups mentioned above have not made any determination as to any other intellectual property rights which may apply to the practice of this standard. Any further consideration of these matters is the user's own responsibility.

Security Considerations

Security issues are discussed throughout this memo.

References

[3DES] W. Tuchman, "Hellman Presents No Shortcut Solutions To DES," IEEE Spectrum, v. 16, n. 7, July 1979, pp40-41.

[BLEI] Bleichenbacher D., "Chosen Ciphertext Attacks against Protocols Based on RSA Encryption Standard PKCS #1" in Advances in Cryptology -- CRYPTO'98, LNCS vol. 1462, pages: 1--12, 1998.

[DES] ANSI X3.106, "American National Standard for Information Systems-Data Link Encryption," American National Standards Institute, 1983.

[DH1] W. Diffie and M. E. Hellman, "New Directions in Cryptography," IEEE Transactions on Information Theory, V. IT-22, n. 6, Jun 1977, pp. 74-84.

[DSS] NIST FIPS PUB 186, "Digital Signature Standard," National Institute of Standards and Technology, U.S. Department of Commerce, May 18, 1994.

[FTP] Postel J., and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985.

[HTTP] Berners-Lee, T., Fielding, R., and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.

[HMAC] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, February 1997.

[IDEA] X. Lai, "On the Design and Security of Block Ciphers," ETH Series in Information Processing, v. 1, Konstanz: Hartung-Gorre Verlag, 1992.

[MD2] Kaliski, B., "The MD2 Message Digest Algorithm", RFC 1319, April 1992.

[MD5] Rivest, R., "The MD5 Message Digest Algorithm", RFC 1321, April 1992.

[PKCS1] RSA Laboratories, "PKCS #1: RSA Encryption Standard," version 1.5, November 1993.

[PKCS6] RSA Laboratories, "PKCS #6: RSA Extended Certificate Syntax Standard," version 1.5, November 1993.

[PKCS7] RSA Laboratories, "PKCS #7: RSA Cryptographic Message Syntax Standard," version 1.5, November 1993.

[PKIX] Housley, R., Ford, W., Polk, W. and D. Solo, "Internet Public Key Infrastructure: Part I: X.509 Certificate and CRL Profile", RFC 2459, January 1999.

[RC2] Rivest, R., "A Description of the RC2(r) Encryption Algorithm", RFC 2268, January 1998.

[RC4] Thayer, R. and K. Kaukonen, A Stream Cipher Encryption Algorithm, Work in Progress.

[RSA] R. Rivest, A. Shamir, and L. M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, v. 21, n. 2, Feb 1978, pp. 120-126.

[RSADSI] Contact RSA Data Security, Inc., Tel: 415-595-8782

[SCH] B. Schneier. Applied Cryptography: Protocols, Algorithms, and Source Code in C, Published by John Wiley & Sons, Inc. 1994.

[SHA] NIST FIPS PUB 180-1, "Secure Hash Standard," National Institute of Standards and Technology, U.S. Department of Commerce, Work in Progress, May 31, 1994.

[SSL2] Hickman, Kipp, "The SSL Protocol", Netscape Communications Corp., Feb 9, 1995.

[SSL3] A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 Protocol", Netscape Communications Corp., Nov 18, 1996.

[TCP] Postel, J., "Transmission Control Protocol," STD 7, RFC 793, September 1981.

[TEL] Postel J., and J. Reynolds, "Telnet Protocol Specifications", STD 8, RFC 854, May 1993.

[TEL] Postel J., and J. Reynolds, "Telnet Option Specifications", STD 8, RFC 855, May 1993.

[X509] CCITT. Recommendation X.509: "The Directory - Authentication Framework". 1988.

[XDR] R. Srinivansan, Sun Microsystems, RFC-1832: XDR: External Data Representation Standard, August 1995.

Credits

Win Treese
Open Market

EMail: treese@openmarket.com

Editors

Christopher Allen
Certicom

EMail: callen@certicom.com

Tim Dierks
Certicom

EMail: tdierks@certicom.com

Authors' Addresses

Tim Dierks
Certicom

EMail: tdierks@certicom.com

Philip L. Karlton
Netscape Communications

Alan O. Freier
Netscape Communications

EMail: freier@netscape.com

Paul C. Kocher
Independent Consultant

EMail: pck@netcom.com

Other contributors

Martin Abadi
Digital Equipment Corporation

EMail: ma@pa.dec.com

Robert Relyea
Netscape Communications

EMail: relyea@netscape.com

Ran Canetti
IBM Watson Research Center

EMail: canetti@watson.ibm.com

Jim Roskind
Netscape Communications

EMail: jar@netscape.com

Taher Elgamal
Securify

EMail: elgamal@securify.com

Micheal J. Sabin, Ph. D.
Consulting Engineer

EMail: msabin@netcom.com

Anil R. Gangolli
Structured Arts Computing Corp.

EMail: gangolli@structuredarts.com

Dan Simon
Microsoft

EMail: dansimon@microsoft.com

Kipp E.B. Hickman
Netscape Communications

EMail: kipp@netscape.com

Tom Weinstein
Netscape Communications

EMail: tomw@netscape.com

Hugo Krawczyk
IBM Watson Research Center

EMail: hugo@watson.ibm.com

Comments

The discussion list for the IETF TLS working group is located at the e-mail address <ietf-tls@lists.consensus.com>. Information on the group and information on how to subscribe to the list is at <http://lists.consensus.com/>.

Archives of the list can be found at:
<http://www.imc.org/ietf-tls/mail-archive/>

Full Copyright Statement

Copyright (C) The Internet Society (1999). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.