Colin Boyd
Anish Mathuria

Protocols for Authentication and Key Establishment



Springer

Petitioner Samsung Ex-1036, 0001

Information Security and Cryptography
Texts and Monographs

Series Editors
Ueli Maurer
Ronald L. Rivest

Associate Editors
Martin Abadi
Ross Anderson
Mihir Bellare
Oded Goldreich
Tatsuaki Okamoto
Paul van Oorschot
Birgit Pfitzmann
Aviel D. Rubin
Jacques Stern

Petitioner Samsung Ex-1036, 0002

Springer-Verlag Berlin Heidelberg GmbH

Petitioner Samsung Ex-1036, 0003

Colin Boyd • Anish Mathuria

Protocols for Authentication and Key Establishment

With 12 Figures and 31 Tables



Springer

Petitioner Samsung Ex-1036, 0004

Authors

Colin Boyd
School of Software Engineering
and Data Communications
Queensland University of Technology
GPO Box 2434, Brisbane Q4001
Australia

c.boyd@qut.edu.au

Anish Mathuria
Dhirubhai Ambani Institute
of Information & Communication
Technology
Gandhinagar-382 009 Gujarat
India

anish_mathuria@da-iict.org

Series Editors

Ueli Maurer
Inst. für Theoretische Informatik
ETH Zürich, 8092 Zürich
Switzerland

Ronald L. Rivest
MIT, Dept. of Electrical Engineering
and Computer Science
545 Technology Square, Room 324
Cambridge, MA 02139
USA

Library of Congress Cataloging-in-Publication Data

Boyd, Colin. Protocols for authentication and key establishment Colin A. Boyd,
Anish Mathuria. p.cm. -- (Information security and cryptography)
Includes bibliographical references and index.
ISBN 978-3-642-07716-6 ISBN 978-3-662-09527-0 (eBook)
DOI 10.1007/978-3-662-09527-0

1. Computer networks--Security measures. 2. Computer network protocols.
3. Data encryption (Computer science) 4. Public key infrastructure
(computer security) I. Mathuria, Anish, 1967- II. Title. III. Series.
TK5105.59.B59 2003 005.8' 2--dc21 2003054427

ACM Subject Classification (1998): K.6.5

ISBN 978-3-642-07716-6

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag Berlin Heidelberg GmbH .
Violations are liable for prosecution under the German Copyright Law.

http://www.springer.de

© Springer-Verlag Berlin Heidelberg 2003
Originally published by Springer-Verlag Berlin Heidelberg New York in 2003
Softcover reprint of the hardcover 1st edition 2003

The use of general descriptive names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typesetting: Camera-ready by the authors
Cover Design: KünkelLopka, Heidelberg
Printed on acid-free paper 06/3142PS – 5 4 3 2 1 0

Petitioner Samsung Ex-1036, 0005

To the memory of my father
Thou thy worldly task hast done

To my wife, Hemal

Petitioner Samsung Ex-1036, 0006

Preface

Authentication and key establishment are fundamental building blocks for securing electronic communications. Cryptographic algorithms for encryption and integrity cannot perform their function unless secure keys have been established and the users know which parties share such keys. It is essential that protocols for providing authentication and key establishment are fit for their purpose.

Although there are certainly earlier publications on techniques for authentication, it is fair to regard the 1978 *Communications of the ACM* paper of Needham and Schroeder as the starting point for the modern study of protocols for authentication and key establishment. There are two main research camps that have advanced understanding of cryptographic protocols since the publication of the Needham–Schroeder paper. These are the *cryptography* community and the *computer security* community. Between them, these communities have produced a vast number of protocols; some would say far too many. This has led to many difficulties for researchers and practitioners who wish to navigate this vast literature. We hope that this book will help them find their way.

We believe that this book is the first comprehensive treatment of protocols for authentication and key establishment. There have, of course, been previous surveys of a more limited nature and these have provided valuable information and examples for us. Particularly useful was the extensive survey of Clark and Jacob [95] which includes a library of some 40 protocols as well as introductory material and an annotated bibliography. Two chapters in the authoritative *Handbook of Applied Cryptography* of Menezes, van Oorschot and Vanstone were also a source of much information. Nevertheless, in virtually all cases we have taken protocol descriptions directly from the original sources.

Our aim has been to provide a comprehensive coverage of our subject matter. Yet, despite restricting ourselves strictly to protocols for authentication and key establishment, we quickly realised that it was unrealistic to be exhaustive. When we started the project in 1999 it seemed to us that there was a certain stability in the field, and this was a strong argument for the time-

Petitioner Samsung Ex-1036, 0007

liness of the work. To an extent this judgement has turned out to be sound, at least for simple protocols of well-established type. But for more complex protocols we were very definitely wrong, and the problem of trying to draw a definitive picture of an ever changing landscape caused many headaches. This is particularly true of the material on group-oriented protocols and that on password-based protocols. Even as we write this preface, we are aware that new protocols and design techniques are being developed in both these areas.

How to Use This Book

We hope that this book will prove useful both to those who wish to learn more about the field and as a reference for those looking for information about a specific protocol or group of protocols. The first two chapters are introductory and may be useful for the graduate student, or anyone coming to the field for the first time. Material in the remaining five chapters has been arranged thematically to help the reader identify connections between different protocols. Between them, these five chapters survey more than 150 protocols from the literature.

Chapter 1 starts with a tutorial aimed at explaining the general methods of how protocols work and the typical capabilities of protocol adversaries. Definitions for the basic protocol components follow, including a quick overview of cryptographic algorithms and their properties, as well as a list of typical protocol attacks.

Chapter 2 is devoted to a study of the different goals that protocols for authentication and key establishment may have. We believe that this is a critical part of understanding protocol analysis, and neglect of this issue has been the source of much error in the past. The chapter develops a hierarchy of different goals, considering only extensional goals. This hierarchy is used in subsequent chapters to evaluate various protocols. We feel the hierarchy provides a simple yet effective tool for describing protocol properties and for evaluating attacks against protocols with unclear goals. The last section contains a brief survey of formal protocol analysis techniques, broadly divided into those using formal specification and those using complexity-theoretic proofs.

Chapter 3 is concerned with protocols that employ symmetric cryptography. Many of these protocols involve an on-line trusted third party, in the tradition established by Needham and Schroeder.

Chapter 4 deals with protocols using public key cryptography, but excluding key agreement protocols. As in Chap. 2 we include some standardised protocols and also some protocols in wide use today, such as the TLS protocol.

Chapter 5 is concerned with key agreement based on public keys. Most of the protocols in this chapter are based on the Diffie–Hellman key exchange. There is a vast range of protocols in this class, and consequently

Petitioner Samsung Ex-1036, 0008

this is the longest chapter. There is also a treatment of identity-based key agreement protocols.

Chapter 6 covers conference key protocols. Much of this chapter concerns generalisations of protocols from Chap. 5 to the multi-party setting. In particular, Diffie–Hellman key agreement with multiple parties is discussed in some detail. A topic that is not treated in any depth in this chapter is that of dynamic conferences.

Chapter 7 deals with password-based protocols. Such protocols were first developed not much over 10 years ago. Recently there have been many new protocols proposed in this area, and we have tried to take into account the most important of these.

Appendix A is a brief overview of published standards for authentication and key establishment protocols. Rather than give the details in this appendix, reference is made to many of the standardised protocols described in different chapters of the book.

We expect that the book may be used by the practitioner for tasks such as finding whether an established protocol exists for a specific application, or whether any attacks are known on a specific protocol or on related protocols. To help with such tasks we have provided a list of protocols and a list of attacks in the front matter. There is also an index of protocol names at the back as well as the more usual general index.

We have used a uniform notation for all protocols in the book, and have aimed to present the protocols in a manner that explains the important details clearly, and yet does not give excessive detail. There are different views of how to best achieve this aim, and we have compromised by using two different main presentation techniques. In Chaps. 3 and 4, and occasionally in later chapters, we have used a notation simply showing the messages exchanged between the protocol principals. We have preferred this notation when there are more than two principals involved, and when the interpretation of protocol actions seems straightforward. In most of the protocols in Chaps. 5, 6 and 7 we have used protocol flow diagrams that include principal actions. This notation seems more useful when there are complex and important actions involved. However, these diagrams can get rather cluttered, particularly for conference key protocols.

Acknowledgements

Many people have given their help and advice generously over the several years that it has taken us to produce the book. We would like particularly to acknowledge Paul van Oorschot who looked at an early draft of the book, provided detailed feedback and sterling advice, and also introduced us to Springer. The series editors, Ueli Maurer and Ron Rivest, have also been supportive throughout.

Different people have reviewed and provided comment on various portions of the book. We thank them all for the generous spirit and are sincerely sorry

Petitioner Samsung Ex-1036, 0009

if we have misinterpreted or forgotten any of their helpful advice. This list includes all those who we can remember; we hope not to have missed anyone.

Ross Anderson	Malcolm Boyd	John Clark
Ed Dawson	Marie Henderson	Yvonne Hitchcock
David Jablon	Yongdae Kim	Philip MacKenzie
Wenbo Mao	Keith Martin	Alfred Menezes
Chris Mitchell	DongGook Park	Josef Pieprzyk
Ron Rivest	Carsten Rudolph	Rei Safavi-Naini
Kapali Viswanathan		

In addition, Daniel Bleichenbacher and Serge Vaudenay provided helpful answers to specific questions we asked of them. We are solely responsible for all errors of fact, presentation style, or just plain typing errors that this book may have.

The team at Springer have been encouraging and helpful throughout. We would particularly like to thank Alfred Hofmann who carefully introduced us to the publication process. Ingeborg Mayer has been an unfailing source of information and advice, and displayed extreme patience as we continually failed to meet our deadlines.

CB would like to record his personal thanks to his family $D + C^3$ who put up with considerable neglect, week after week and month after month, yet remained understanding and supportive. The Information Security Research Centre and the School of Software Engineering and Data Communications at Queensland University of Technology provided time and material support and he would like to acknowledge the encouragement of their respective heads, Ed Dawson and Bill Caelli. He would also like to acknowledge Papa Haydn for sustaining him during many a long working session.

AM would like to thank his wife, Hemal, and daughter, Radhika, for their love and patience throughout this project. His writing of this book was started at the University of Massachusetts at Dartmouth; he would like to acknowledge his colleagues in the Computer and Information Sciences Department at UMASS for their support and encouragement. His current institution, Dhirubhai Ambani Institute of Information and Communication Technology, generously granted him time to complete the book.

This book was typeset in LaTeX on various different platforms. We used Ron Rivest's `windex` package to help with index preparation. The style for the protocols and attacks is adapted from Anselm Lingau's float package.

Brisbane, Gandhinagar
June 2003

Colin Boyd
Anish Mathuria

Petitioner Samsung Ex-1036, 0010

Contents

List of Protocols XVII
List of Attacks XXIII

1 A Tutorial Introduction to Authentication and Key Establishment 1
1.1 Introduction 1
1.2 Building a Key Establishment Protocol 2
1.2.1 Confidentiality 4
1.2.2 Authentication 5
1.2.3 Replay 8
1.3 Protocol Architectures 12
1.3.1 Existing Cryptographic Keys 12
1.3.2 Method of Session Key Generation 13
1.3.3 Number of Users 13
1.3.4 Example 13
1.4 Cryptographic Properties 14
1.4.1 Confidentiality 16
1.4.2 Data Origin Authentication and Data Integrity 17
1.4.3 Non-repudiation 18
1.4.4 Examples of Cryptographic Algorithms 19
1.4.5 Secret Sharing 20
1.5 Freshness 21
1.6 Types of Attack on Protocols 23
1.6.1 Eavesdropping 24
1.6.2 Modification 25
1.6.3 Replay 25
1.6.4 Preplay 25
1.6.5 Reflection 25
1.6.6 Denial of Service 27
1.6.7 Typing Attacks 28
1.6.8 Cryptanalysis 29

Petitioner Samsung Ex-1036, 0011

1.6.9 Certificate Manipulation 30
1.6.10 Protocol Interaction 31
1.7 Design Principles for Cryptographic Protocols 31

2 Goals for Authentication and Key Establishment 33
2.1 Introduction 33
2.2 Basic Goals 35
2.2.1 Models of Security 35
2.2.2 Key Establishment or Authentication? 36
2.2.3 User-Oriented Goals 38
2.2.4 Key-Oriented Goals 40
2.3 Enhanced Goals 41
2.3.1 A Hierarchy of Protocol Goals 41
2.3.2 Example: STS Protocol 44
2.3.3 Intensional and Extensional Goals 46
2.3.4 Protocol Efficiency 47
2.3.5 Responsibility and Credit 48
2.4 Goals Concerning Compromised Keys 49
2.4.1 Forward Secrecy 50
2.4.2 Key Compromise Impersonation 52
2.5 Formal Verification of Protocols 52
2.5.1 FDR 53
2.5.2 Murϕ 56
2.5.3 Brutus 57
2.5.4 NRL Analyzer 58
2.5.5 BAN Logic 59
2.5.6 Strand Space Model 62
2.5.7 The Inductive Model 63
2.5.8 Comparison of Formal Methods Approaches 65
2.6 Complexity-Theoretic Proofs of Security 66
2.6.1 Model of Communication 67
2.6.2 Defining Security 68
2.6.3 Shoup's Simulation Model 69
2.6.4 A Modular Approach to Proofs 71
2.7 Conclusion 71

3 Protocols Using Shared Key Cryptography 73
3.1 Introduction 73
3.2 Entity Authentication Protocols 75
3.2.1 Bird–Gopal–Herzberg–Janson–Kutten–Molva–Yung Protocols 75
3.2.2 Bellare–Rogaway MAP1 Protocol 76
3.2.3 ISO/IEC 9798-2 Protocols 77
3.2.4 Woo–Lam Authentication Protocol 78
3.2.5 Comparison of Entity Authentication Protocols 80

Petitioner Samsung Ex-1036, 0012

3.3 Server-Less Key Establishment 80
3.3.1 Andrew Secure RPC Protocol 81
3.3.2 Janson–Tsudik 2PKDP Protocol 83
3.3.3 Boyd Two-Pass Protocol 83
3.3.4 ISO/IEC 11770-2 Server-Less Protocols 84
3.3.5 Comparison of Server-Less Protocols 86
3.4 Server-Based Key Establishment 87
3.4.1 Needham–Schroeder Shared Key Protocol 87
3.4.2 Otway–Rees Protocol 88
3.4.3 Kerberos Protocol 91
3.4.4 ISO/IEC 11770-2 Server-Based Protocols 93
3.4.5 Wide-Mouthed-Frog Protocol 94
3.4.6 Yahalom Protocol 95
3.4.7 Janson–Tsudik 3PKDP Protocol 97
3.4.8 Bellare–Rogaway 3PKD Protocol 98
3.4.9 Woo–Lam Key Transport Protocol 99
3.4.10 Gong Key Agreement Protocols 100
3.4.11 Boyd Key Agreement Protocol 101
3.4.12 Gong Hybrid Protocol 102
3.4.13 Comparison of Server-Based Protocols 103
3.5 Key Establishment Using Multiple Servers 104
3.5.1 Gong's Multiple Server Protocol 104
3.5.2 Chen–Gollmann–Mitchell Protocol 105
3.6 Conclusion 106

4 Authentication and Key Transport Using Public Key Cryptography 107
4.1 Introduction 107
4.1.1 Notation 108
4.1.2 Design Principles for Public Key Protocols 108
4.2 Entity Authentication Protocols 110
4.2.1 Protocols in ISO/IEC 9798-3 110
4.2.2 Protocols in ISO/IEC 9798-5 113
4.2.3 SPLICE/AS 113
4.2.4 Comparison of Entity Authentication Protocols 115
4.3 Key Transport Protocols 116
4.3.1 Protocols in ISO/IEC 11770-3 116
4.3.2 Blake-Wilson and Menezes Provably Secure Key Transport Protocol 120
4.3.3 Needham–Schroeder Public Key Protocol 121
4.3.4 Protocols in the X.509 Standard 122
4.3.5 TLS Protocol 124
4.3.6 Beller–Chang–Yacobi Protocols 126
4.3.7 TMN Protocol 131
4.3.8 AKA Protocol 132

Petitioner Samsung Ex-1036, 0013

4.3.9 Comparison of Key Transport Protocols 133
4.4 Conclusion 134

5 Key Agreement Protocols 137
5.1 Introduction 137
5.1.1 Key Control 138
5.1.2 Unknown Key-Share Attacks 139
5.1.3 Classes of Key Agreement 140
5.2 Diffie–Hellman Key Agreement 141
5.2.1 Small Subgroup Attacks 144
5.2.2 ElGamal Encryption and One-Pass Key Establishment . 144
5.2.3 Lim–Lee Protocol Using Static Diffie–Hellman 146
5.3 MTI Protocols 147
5.3.1 Small Subgroup Attack on MTI Protocols 149
5.3.2 Unknown Key-Share Attacks on MTI Protocols 150
5.3.3 Lim–Lee Attack on MTI Protocols 151
5.3.4 Impersonation Attack of Just and Vaudenay 152
5.3.5 Triangle Attacks on MTI Protocols 153
5.3.6 Forward Secrecy and Key Compromise Impersonation for MTI Protocols 154
5.4 Diffie–Hellman-Based Protocols with Basic Message Format . . . 155
5.4.1 The Goss Protocol 156
5.4.2 KEA Protocol 157
5.4.3 The Unified Model Protocol 158
5.4.4 MQV Protocol 159
5.4.5 Yacobi's Protocol 160
5.4.6 Ateniese–Steiner–Tsudik Protocol 161
5.4.7 Just–Vaudenay–Song–Kim Protocol 162
5.4.8 Adding Key Confirmation 163
5.4.9 Comparison 164
5.5 Diffie–Hellman-Based Protocols with Enhanced Message Format 165
5.5.1 STS Protocol 166
5.5.2 Oakley Protocol 168
5.5.3 SKEME Protocol 172
5.5.4 Internet Key Exchange 174
5.5.5 Arazi's Protocol 178
5.5.6 Lim–Lee Protocols 179
5.5.7 Hirose–Yoshida Protocol 180
5.5.8 Comparison 181
5.6 Identity-Based Schemes 182
5.6.1 Okamoto's Scheme 184
5.6.2 Günther's Scheme 186
5.6.3 Girault's Scheme 188
5.6.4 Schemes Using Signatures with Message Recovery 190

Petitioner Samsung Ex-1036, 0014

5.7 Protocols Designed for Computationally Limited Devices 190
5.7.1 Yacobi–Shmuely Protocol 191
5.7.2 ASPeCT Protocol 192
5.7.3 Jakobsson–Pointcheval Protocol 193
5.8 Protocols in ISO/IEC 11770-3 195
5.9 Diffie–Hellman Key Agreement in Other Groups 196
5.10 Protocols Not Based on Diffie–Hellman 197
5.10.1 SKEME without Forward Secrecy 197
5.10.2 Key Pre-distribution Schemes 198
5.11 Conclusion 199

6 Conference Key Protocols 201
6.1 Introduction 201
6.1.1 Generalised Security Goals 202
6.1.2 Static and Dynamic Groups 203
6.1.3 Notation 204
6.2 Generalising Diffie–Hellman Key Agreement 204
6.2.1 Ingemarsson–Tang–Wong Key Agreement 204
6.2.2 Steiner–Tsudik–Waidner Key Agreement 206
6.2.3 Steer–Strawczynski–Diffie–Wiener Key Agreement 209
6.2.4 Perrig's Generalised Diffie–Hellman 210
6.2.5 Becker and Wille's Octopus Protocol 212
6.2.6 Burmester–Desmedt Key Agreement 214
6.2.7 Joux's Tripartite Diffie–Hellman 215
6.2.8 Security of Generalised Diffie–Hellman 216
6.2.9 Efficiency of Generalised Diffie–Hellman 217
6.3 Conference Key Agreement Protocols 219
6.3.1 Authenticating Generalised Diffie–Hellman 219
6.3.2 Klein–Otten–Beth Protocol 220
6.3.3 Authenticated GDH Protocols 221
6.4 Identity-Based Conference Key Protocols 225
6.4.1 Koyama and Ohta Protocols 226
6.4.2 Protocols of Saeednia and Safavi-Naini 229
6.5 Conference Key Agreement without Diffie–Hellman 230
6.5.1 Pieprzyk and Li's Key Agreement Protocol 230
6.5.2 Tzeng–Tzeng Protocols 232
6.5.3 Boyd's Conference Key Agreement 233
6.6 Conference Key Transport Protocols 235
6.6.1 Burmester–Desmedt Star and Tree Protocols 235
6.6.2 Mayer and Yung's Protocols 237
6.6.3 Key Hierarchies 239
6.7 Key Broadcasting Protocols 240
6.7.1 Key Broadcasting Using Number Theory 242
6.7.2 Key Broadcasting Using Secret Sharing 244
6.8 Conclusion 245

Petitioner Samsung Ex-1036, 0015

7 Password-Based Protocols 247
7.1 Introduction 247
7.2 Encrypted Key Exchange Using Diffie–Hellman 250
7.2.1 Bellovin–Merritt's Original EKE 250
7.2.2 The PAK Protocol 252
7.2.3 SPEKE 256
7.2.4 Katz–Ostrovsky–Yung Protocol 258
7.3 Augmented EKE 260
7.3.1 B-SPEKE 263
7.3.2 SRP Protocol 264
7.3.3 AMP Protocol 265
7.4 Three-Party EKE 266
7.4.1 GLNS Secret Public Key Protocols 267
7.4.2 Steiner, Tsudik and Waidner Three-Party EKE 271
7.5 RSA-Based Protocols 273
7.5.1 RSA-Based EKE 273
7.5.2 OKE and SNAPI 274
7.6 Protocols Using a Server Public Key 276
7.6.1 GLNS Protocols with Server Public Keys 277
7.6.2 Kwon–Song Protocols 278
7.6.3 Halevi–Krawczyk Protocols 279
7.6.4 Three-Party Protocol of Yen and Liu 280
7.7 Other Protocols 282
7.7.1 Lee–Sohn–Yang–Won Protocol 282
7.7.2 Anderson–Lomas Protocol 283
7.7.3 Strengthening Passwords 284
7.8 Conclusion 285

A Standards for Authentication and Key Establishment 289
A.1 ISO Standards 289
A.1.1 ISO/IEC 9798 289
A.1.2 ISO/IEC 11770 290
A.1.3 ISO 9594-8/ITU X.509 290
A.2 Other Standards 290
A.2.1 IETF Standards 290
A.2.2 IEEE P1363-2000 291
A.2.3 NIST and ANSI Standards 291

B Summary of Notation 293

References 295

Index of Protocols 317

General Index 319

Petitioner Samsung Ex-1036, 0016

List of Protocols

1.1 First protocol attempt in conventional notation 3
1.2 A protocol in an unusual class 14
1.3 Use of a nonce (random challenge) 22
1.4 A protocol vulnerable to reflection attack 26
1.5 Otway–Rees protocol 28
1.6 A protocol vulnerable to certificate manipulation (MTI protocol) 30
2.1 Example protocol 34
2.2 A simple authentication protocol 38
2.3 Another simple authentication protocol 39
2.4 STS protocol 44
2.5 STS protocol modified to include identifiers 45
2.6 A protocol in which A takes 'responsibility' 49
2.7 A protocol in which A takes 'credit' 49
2.8 Key transport protocol providing forward secrecy 51
2.9 Server-based protocol providing forward secrecy 51
3.1 Bird *et al.* canonical protocol 1 75
3.2 Bellare–Rogaway MAP1 protocol 76
3.3 Protocol for attacking MAP1 protocol 76
3.4 ISO/IEC 9798-2 one-pass unilateral authentication protocol 77
3.5 ISO/IEC 9798-2 two-pass unilateral authentication protocol ... 78
3.6 ISO/IEC 9798-2 two-pass mutual authentication protocol 78
3.7 ISO/IEC 9798-2 three-pass mutual authentication protocol 78
3.8 Woo–Lam unilateral authentication protocol 79
3.9 Andrew secure RPC protocol 81
3.10 Revised Andrew protocol of Burrows *et al.* 82
3.11 Janson–Tsudik 2PKDP protocol 83
3.12 Boyd two-pass protocol 84
3.13 ISO/IEC 11770-2 Key Establishment Mechanism 1 84
3.14 ISO/IEC 11770-2 Key Establishment Mechanism 2 84
3.15 ISO/IEC 11770-2 Key Establishment Mechanism 3 85
3.16 ISO/IEC 11770-2 Key Establishment Mechanism 4 85

Petitioner Samsung Ex-1036, 0017

3.17 ISO/IEC 11770-2 Key Establishment Mechanism 5 85
3.18 ISO/IEC 11770-2 Key Establishment Mechanism 6 86
3.19 Needham–Schroeder shared key protocol 88
3.20 Denning–Sacco protocol 88
3.21 Bauer–Berson–Feiertag protocol 88
3.22 Otway–Rees protocol 89
3.23 Otway–Rees protocol modified by Burrows *et al.* 90
3.24 Otway–Rees protocol modified by Abadi and Needham 91
3.25 Basic Kerberos protocol 92
3.26 Optional Kerberos message to complete mutual authentication . 92
3.27 ISO/IEC 11770-2 Key Establishment Mechanism 10 93
3.28 ISO/IEC 11770-2 Key Establishment Mechanism 12 94
3.29 ISO/IEC 11770-2 Key Establishment Mechanism 13 94
3.30 Wide-mouthed-frog protocol 94
3.31 Yahalom protocol 95
3.32 Yahalom protocol modified by Burrows *et al.* 96
3.33 Janson–Tsudik 3PKDP protocol 98
3.34 Janson–Tsudik optimised 3PKDP protocol 98
3.35 Bellare–Rogaway 3PKD protocol 99
3.36 Woo–Lam key transport protocol 99
3.37 Gong's timestamp-based protocol 100
3.38 Gong's nonce-based protocol 100
3.39 Alternative Gong's protocol 101
3.40 Boyd key agreement protocol 101
3.41 Gong's hybrid protocol 102
3.42 Gong's simplified multi-server protocol 104
3.43 Chen–Gollmann–Mitchell multi-server protocol 105
4.1 ISO/IEC 9798-3 one-pass unilateral authentication 110
4.2 ISO/IEC 9798-3 two-pass unilateral authentication 111
4.3 ISO/IEC 9798-3 two-pass mutual authentication 111
4.4 ISO/IEC 9798-3 three-pass mutual authentication 112
4.5 Early version of ISO/IEC 9798-3 three-pass mutual authentication 112
4.6 ISO/IEC 9798-3 two-pass parallel authentication 113
4.7 SPLICE/AS protocol 114
4.8 Clark–Jacob variant of SPLICE/AS 114
4.9 Gray variant of SPLICE/AS 115
4.10 ISO/IEC 11770-3 Key Transport Mechanism 1 116
4.11 ISO/IEC 11770-3 Key Transport Mechanism 2 117
4.12 ISO/IEC 11770-3 Key Transport Mechanism 3 118
4.13 Denning–Sacco public key protocol 118
4.14 ISO/IEC 11770-3 Key Transport Mechanism 4 118
4.15 ISO/IEC 11770-3 Key Transport Mechanism 5 119
4.16 ISO/IEC 11770-3 Key Transport Mechanism 6 119
4.17 Helsinki protocol 120

Petitioner Samsung Ex-1036, 0018

4.18 Blake-Wilson–Menezes provably secure key transport protocol . . 121
4.19 Needham–Schroeder public key protocol 121
4.20 Lowe's variant of Needham–Schroeder public key protocol 122
4.21 X.509 one-pass authentication 123
4.22 X.509 two-pass authentication 123
4.23 X.509 three-pass authentication 124
4.24 Simplified TLS key transport protocol 125
4.25 Simplified TLS key agreement protocol 126
4.26 Basic MSR protocol of Beller, Chang and Yacobi 127
4.27 Improved IMSR protocol of Carlsen 128
4.28 Beller–Yacobi protocol 129
4.29 Improved Beller–Yacobi protocol 130
4.30 Carlsen's improved Beller–Chang–Yacobi MSR + DH protocol . 131
4.31 Simplified TMN protocol (KDP2) 132
4.32 AKA protocol 132
5.1 Diffie–Hellman key agreement 141
5.2 Agnew–Mullin–Vanstone protocol 145
5.3 Original Nyberg–Rueppel protocol 145
5.4 Revised Nyberg–Rueppel protocol 146
5.5 Lim–Lee protocol using static Diffie–Hellman 147
5.6 MTI A(0) protocol 148
5.7 MTI A(k) protocol 148
5.8 Modified MTI B(0) protocol 151
5.9 KEA protocol 157
5.10 Unified Model key agreement 158
5.11 MQV protocol 159
5.12 Ateniese–Steiner–Tsudik key agreement 161
5.13 Just–Vaudenay–Song–Kim protocol 162
5.14 Generic addition of key confirmation to basic DH protocols 164
5.15 STS protocol 166
5.16 Modified STS protocol 167
5.17 STS protocol using MACs 168
5.18 Oakley aggressive mode protocol 169
5.19 Alternative Oakley protocol 170
5.20 Oakley conservative protocol 171
5.21 SKEME protocol basic mode 173
5.22 IKE main protocol using digital signatures 176
5.23 Arazi's key agreement protocol 178
5.24 Lim–Lee Schnorr-based protocol 179
5.25 Lim–Lee Schnorr-based variant 180
5.26 Hirose–Yoshida key agreement protocol 181
5.27 Okamoto's identity-based protocol 184
5.28 Okamoto–Tanaka identity-based protocol 186
5.29 Günther's key agreement protocol 187
5.30 Günther's extended key agreement protocol 188

Petitioner Samsung Ex-1036, 0019

5.31 Saeednia's variant of Günther's key agreement protocol 189
5.32 Girault's identity-based protocol 190
5.33 Yacobi–Shmuely protocol 191
5.34 Park key agreement protocol 192
5.35 ASPeCT protocol 193
5.36 Jakobsson–Pointcheval protocol 194
5.37 Wong–Chan protocol 195
5.38 SKEME protocol without forward secrecy 197
6.1 Ingemarsson–Tang–Wong generalised Diffie–Hellman 206
6.2 Steiner–Tsudik–Waidner GDH.1 207
6.3 Steiner–Tsudik–Waidner GDH.2 207
6.4 Steiner–Tsudik–Waidner GDH.3 209
6.5 Steer–Strawczynski–Diffie–Wiener generalised Diffie–Hellman .. 210
6.6 Perrig's generalised Diffie–Hellman 212
6.7 Four-principal basic Octopus protocol 213
6.8 Burmester–Desmedt generalised Diffie–Hellman with broadcasts 214
6.9 Burmester–Desmedt pairwise generalised Diffie–Hellman 216
6.10 Ateniese–Steiner–Tsudik A-GDH.2 222
6.11 Protocol 6.10 when $m = 4$ 223
6.12 Ateniese–Steiner–Tsudik SA-GDH.2 224
6.13 Koyama–Ohta type 1 identity-based conference key agreement . 227
6.14 Saeednia–Safavi-Naini identity-based conference key agreement . 230
6.15 Tzeng and Tzeng's conference key agreement 233
6.16 Boyd's conference key agreement 234
6.17 Burmester–Desmedt star protocol 236
6.18 Hirose–Yoshida conference key transport protocol 236
6.19 Mayer–Yung conference key transport protocol 238
6.20 Chiou–Chen's key broadcasting 243
6.21 Key broadcasting using secret sharing 244
7.1 Diffie–Hellman-based EKE protocol 250
7.2 PAK protocol 254
7.3 PPK protocol 255
7.4 PAK-R protocol 256
7.5 SPEKE protocol 257
7.6 Katz–Ostrovsky–Yung protocol 259
7.7 Augmented Diffie–Hellman-based EKE protocol 261
7.8 PAK-Y protocol 262
7.9 B-SPEKE protocol 263
7.10 SRP protocol 264
7.11 AMP protocol 266
7.12 GLNS secret public key protocol 267
7.13 Simplified GLNS secret public key protocol 269
7.14 Optimal GLNS secret public key protocol 271
7.15 Steiner, Tsudik and Waidner three-party EKE 271
7.16 SNAPI protocol 275

Petitioner Samsung Ex-1036, 0020

7.17 GLNS compact protocol 277
7.18 Optimal GLNS nonce-based protocol 278
7.19 Kwon–Song basic protocol 279
7.20 Halevi–Krawczyk password-based protocol 280
7.21 Yen–Liu protocol 281
7.22 Lee–Sohn–Yang–Won protocol 282
7.23 Anderson–Lomas protocol 283

Petitioner Samsung Ex-1036, 0021

List of Attacks

1.1 Reflection attack on Protocol 1.4 26
1.2 Typing attack on Otway–Rees protocol 29
1.3 Certificate manipulation attack on MTI protocol 30
2.1 Attack on Protocol 2.1 34
2.2 An attack on Protocol 2.2 39
2.3 Lowe's attack on Protocol 2.5 45
3.1 An oracle attack on Protocol 3.1 76
3.2 Chosen protocol attack on MAP1 77
3.3 Abadi's attack on Protocol 3.8 79
3.4 Clark–Jacob attack on Andrew protocol 81
3.5 Lowe's attack on revised Andrew protocol 82
3.6 Attack on Otway–Rees protocol without plaintext checking 89
3.7 Attack on Otway–Rees protocol modified by Burrows *et al.* ... 90
3.8 Attack on Wide-mouthed-frog protocol 95
3.9 Syverson's attack on modified Yahalom protocol 97
3.10 Alternative Syverson's attack on modified Yahalom protocol ... 97
3.11 Insider attack on Protocol 3.41 102
4.1 Canadian attack on Protocol 4.5 112
4.2 Attack of Clark–Jacob on SPLICE/AS protocol 114
4.3 Attack on Helsinki protocol 120
4.4 Lowe's attack on Needham–Schroeder public key protocol 122
4.5 Attack on Beller–Yacobi protocol 130
4.6 Abadi's attack on AKA protocol 133
5.1 Attack on basic Diffie–Hellman 142
5.2 Small subgroup attack on MTI C(1) protocol 150
5.3 Unknown key-share attack on MTI B(0) protocol 150
5.4 Lim–Lee attack on MTI A(0) 152
5.5 Just–Vaudenay impersonation attack on MTI A(0) protocol ... 153
5.6 Key compromise impersonation on MTI C(0) 155
5.7 Unknown key-share attack on generic protocol 156
5.8 Kaliski's unknown key-share attack on MQV protocol 160

Petitioner Samsung Ex-1036, 0022

5.9 Key compromise impersonation on Just–Vaudenay–Song–Kim protocol 163
5.10 Attack on Park key agreement 192
7.1 Ding and Horster's attack on Protocol 7.15 272
7.2 Lin–Sun–Hwang attack on Protocol 7.15 272
7.3 Attack on Protocol 7.21 281

Petitioner Samsung Ex-1036, 0023

1

A Tutorial Introduction to Authentication and Key Establishment

1.1 Introduction

The newcomer to the subject of cryptographic protocols for authentication and key management is likely to be bemused by the sheer variety of techniques and technical background required. But even before this stage is reached a more fundamental question needs to be faced. What are these protocols there for at all? To answer this question it is necessary to provide an understanding of what sets cryptographic protocols apart from other types of protocols. Above all, what are the 'rules of the game' when designing or attacking a protocol, and why are techniques for design and analysis of ordinary (non-cryptographic) protocols not sufficient? This chapter provides necessary background material for those readers who are not already familiar with the topic of cryptographic protocols.

The first section of this chapter is a tutorial introduction to the whole topic. This is intended to lead the beginner (who may already be familiar with cryptographic algorithms and communications protocols) through the fundamentals of the subject by following common mistakes in a hypothetical protocol design. At the same time it enables us to start establishing some common concepts and notation that will be used throughout the rest of this book.

The remaining sections in this chapter form a more systematic approach to the fundamental concepts required to understand cryptographic protocols. We first present a loose classification of the different architectural settings that are commonly encountered in protocols of this sort, in terms of the participants and their roles. The next two sections outline cryptographic mechanisms and then methods to ensure freshness. Both of these are fundamental tools required in almost all practical protocols. This is followed by a survey of most known attacks on protocols; understanding protocol failures is essential in understanding how to design and assess unfamiliar protocols. Finally we review some general design principles coined by Abadi and Needham [4] which summarise many of the themes of this chapter.

Petitioner Samsung Ex-1036, 0024

1.2 Building a Key Establishment Protocol

We now explicate an attempt to design one good protocol. The procedure we will use to explain the ideas is to try to design a protocol for key establishment from first principles. Problems with the protocol will be revealed in stages through presentation of legitimate attacks so that each may be solved in turn. Eventually a good protocol is achieved.

Before we start designing any protocol the communications architecture must be established. We choose one common scenario, but the reader should be aware that there is a wide variety of alternatives (these are explored in Sect. 1.3). Our scenario has a set of users, any two of whom may wish to establish a new key for use in securing their subsequent communications through cryptography. Such a key is known as a *session key*. It is important to understand that successful completion of key establishment (and entity authentication) is only the beginning of a secure communications session: once an appropriate key has been established its use comes in protecting the real data to be communicated with whatever cryptographic mechanisms are chosen.

In order to achieve their aim the users interact with an entity called the *server* which will also engage in the protocol. All users trust the server to execute the protocol faithfully and not to engage in any other activity that will deliberately compromise their security. Furthermore, the server is trusted to generate the new key and to do so in such a way that it is sufficiently random to prevent an attacker gaining any useful information about it.

Our protocols thus involve three entities (often called *principals* in the literature). These are two users whom we denote A and B (often expanded to *Alice* and *Bob*) and the trusted server S. The aim of the protocol is for A and B to establish a new secret key K_{AB} which they can use for subsequent secure communications. The role of S is to generate K_{AB} and transport it to A and B. The aims of the protocol can be summarised as follows.

- At the end of the protocol the value of K_{AB} should be known to both A and B, but to no other parties with the possible exception of S.
- A and B should know that K_{AB} is newly generated.

Let us begin with a completely naïve outlook. A protocol to achieve transport of a new session key K_{AB} is shown in Fig. 1.1. The protocol consists of three messages. Firstly, user A contacts S by sending the identities of the two parties who are going to share the new session key; secondly, S returns the key K_{AB} to A; finally, A passes K_{AB} on to B. Before we examine the (lack of) security of this protocol we should acknowledge that, as a specification of the protocol, Fig. 1.1 is significantly incomplete. Note the following features.

- Only the messages passed in a successful run of the protocol are specified. In particular there is no description of what happens in the case that a message of the wrong format is received or that no message is received at

Petitioner Samsung Ex-1036, 0025

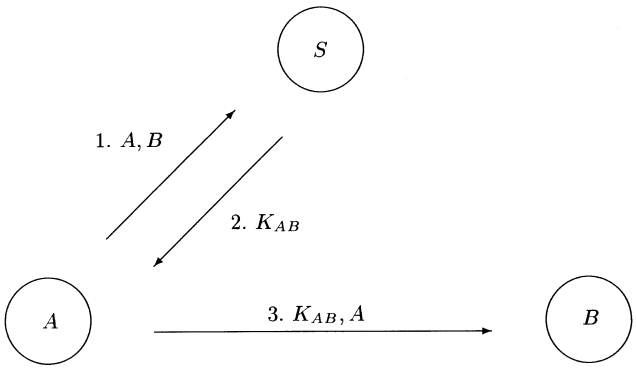


Fig. 1.1. First protocol attempt

all. The inclusion of all such information is standard in specifying ordinary communications protocols, and is essential to prove basic functional properties [278]. It is unfortunate that it is commonplace to omit such information from specifications of cryptographic protocols.

- There is no specification of internal actions of principals. In many protocols the internal actions are fairly obvious; for example, they may be simply to calculate what is required to output the next message. But in others there are numerous alternatives and the choice can have security-critical relevance.
- It is implicitly assumed that A and B 'know' that the received messages are part of the protocol. It is common practice to omit such details which would be required for a networked computer to be able to track the progress of a particular protocol run. This may include details of which key should be used to decrypt a received message which has been encrypted.

Despite the obvious limitations associated with specifying protocols by showing only the messages of a successful run, it remains the most popular method of describing cryptographic protocols in the literature. An equivalent representation of the protocol of Fig. 1.1 is to show the messages sent in a successful run, each preceded by the principals between whom it is intended to pass. For example, the protocol in Fig. 1.1 could be specified as follows.

---

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : K_{AB}$
3. $A \rightarrow B : K_{AB}, A$

Protocol 1.1: First protocol attempt in conventional notation

Petitioner Samsung Ex-1036, 0026

This notation is often preferred when a more compact description is desired. However, it does make the protocol harder to visualise. Furthermore, it may be argued that it tends to reinforce, even more than the diagrammatic version, an assumption that messages automatically reach their destination securely. In the authors' experience it is usually helpful when trying to understand a protocol for the first time to draw a figure showing the message flows between the principals involved.

In this book we will generally use two different formats for protocol descriptions. One is the format of Protocol 1.1, showing only the protocol messages. However, we will usually describe any internal actions explicitly in the commentary. The second format is to show the protocol flows between principals and include, where appropriate, any internal checking that is required. Although the second format is more complete, it is more cumbersome and we will generally use it when the details are less obvious. For both formats we will try to be as precise as possible about the properties required of cryptographic mechanisms. (Such properties are discussed in Sect. 1.4.)

1.2.1 Confidentiality

The reader is probably already aware of the obvious problem with our first attempt. Nevertheless it is our purpose here to be explicit about our assumptions. The problem is that the session key K_{AB} must be transported to A and B but to *no other entities*. It is an assumption that the adversary, against whose attacks we are implementing our security, can eavesdrop on all messages that are sent or received. This is a realistic assumption in typical communications systems such as the Internet and corporate networks. Indeed, if this possibility can be discounted then there is probably no need to apply security at all.

Security Assumption 1 *The adversary is able to eavesdrop on all messages sent in a cryptographic protocol.*

In order to provide confidentiality it is necessary to use a cryptographic algorithm and associated keys. For now we will simply make the assumption that the server S initially shares a secret key with each user of the system. The keys K_{AS} and K_{BS} are shared between A and S, and between B and S, respectively. The need to keep K_{AB} confidential leads immediately to our second protocol attempt shown in Fig. 1.2.

This protocol starts, as our first attempt, with A sending to S the identities of the two parties who are to share the session key. S generates the session key K_{AB}, then encrypts it with each of the keys K_{AS} and K_{BS} and sends the result back to A. Principal A then relays the encrypted key to B along with her identity so that B knows who else has this key.

This protocol is just as insecure in an open environment as our first attempt, but for a completely different reason. A passive eavesdropper cannot

Petitioner Samsung Ex-1036, 0027

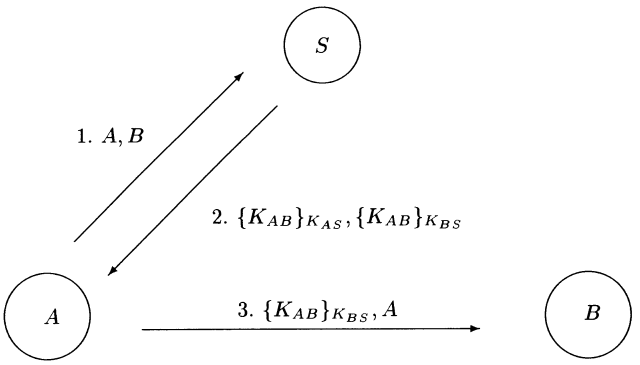


Fig. 1.2. Second protocol attempt

see K_{AB} since encrypted messages may only be read by the legitimate recipients who have the keys required to decrypt. This brings us to the question of cryptographic algorithms. Fundamental cryptographic properties and their use in cryptographic protocols are reviewed in Sect. 1.4. An understanding of cryptographic algorithms is essential for the correct design of protocols. However, the details of the exact cryptographic algorithm used are often irrelevant and such details are frequently avoided during both protocol design and analysis. All the attacks described in this section are independent of the cryptographic algorithms used.

In many analyses an assumption of 'perfect cryptography' is made, which means that there is nothing gained whatsoever by the adversary in observing an encrypted message. Naturally this convenient assumption brings with it certain responsibilities for the protocol designer. In order to make a design practical there must be suitable cryptographic algorithms available that satisfy the security requirements. Furthermore, these requirements must be made explicit as part of the protocol specification. An alternative approach is to include an abstract model of the cryptographic algorithms as part of the protocol specification. The protocol analysis can then aim to verify that any advantage gained by the adversary from eavesdropping on the protocol is sufficiently small. A comparison of these two different views has been made by Abadi and Rogaway [5].

1.2.2 Authentication

The problem with the protocol in Fig. 1.2 is not that it gives away the secret key K_{AB}. The difficulty is that the information about who else has the key is not protected. We need to take into account that the adversary is not only able to eavesdrop on messages sent, but can also capture messages and alter them.

Petitioner Samsung Ex-1036, 0028

Security Assumption 2 *The adversary is able to alter all messages sent in a cryptographic protocol using any information available. In addition the adversary can re-route any message to any other principal. This includes the ability to generate and insert completely new messages.*

We may summarise the situation by saying that the adversary has complete control of the channel over which protocol messages flow. The reason for the difficulty in designing authentication protocols now begins to clarify. In contrast to ordinary communications protocols, there is an unknown and unpredictable principal involved. Although there may be no more than four or five messages involved in a legitimate run of the protocol, there are an infinite number of variations in which the adversary participates. These variations have an unbounded number of messages and each must satisfy the protocol's security requirements. Over the past 10 years there have been various methods devised to gain confidence in the security of protocols; some of these are discussed in Chap. 2.

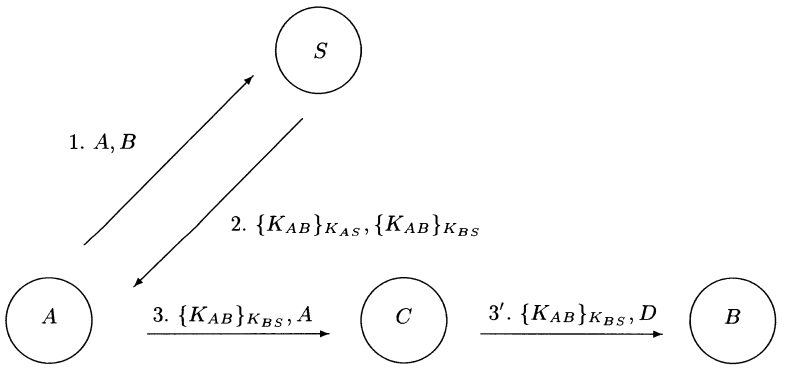


Fig. 1.3. Attack on the second protocol attempt

One attack on our second protocol is shown in Fig. 1.3. The attack is very simple. The adversary C simply intercepts the message from A to B and substitutes D's identity for A's (where D could be any identity including C's own). The consequence is that B believes that he is sharing the key with D whereas he is in fact sharing it with A. The subsequent results of this attack will depend on the scenario in which the protocol is used, but may include such actions as B giving away information to A which should have been shared only with D. Although C does not obtain K_{AB} we still regard the protocol as broken since it does not satisfy our requirement that the users should know who else knows the session key. However, another attack on the protocol does allow C to obtain the session key as shown in Fig. 1.4.

Petitioner Samsung Ex-1036, 0029

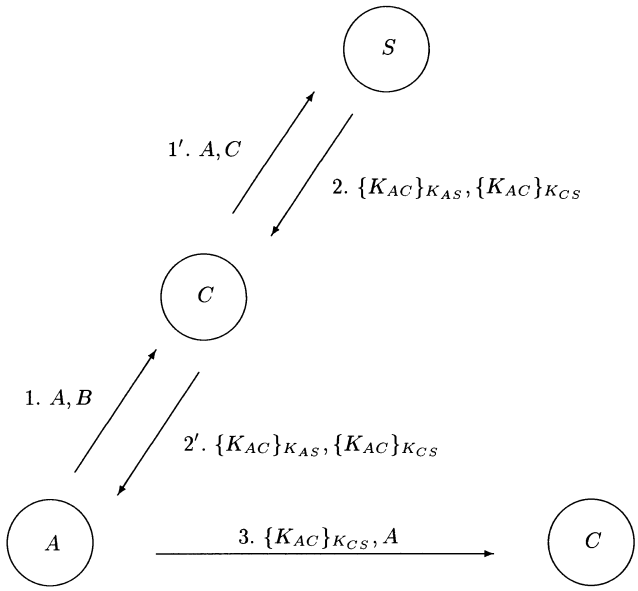


Fig. 1.4. Alternative attack on second protocol attempt

In this second attack C alters the message from A to S so that S encrypts the key K_{AC} with C's key, K_{CS}, instead of with B's key. Since A cannot distinguish between encrypted messages meant for other principals she will not detect the alteration. Notice that K_{AC} is simply a formal name for the bit-string representing the session key so will be accepted by A. Also C collects the message from A intended for B so that B will not detect any anomaly. The result of this attack is that A will believe that the protocol has been successfully completed with B whereas in fact C knows K_{AC} and so can masquerade as B as well as learn all the information that A sends to B. Notice that, in contrast to the previous attack, this one will only succeed if C is a legitimate user known to S. This, again, is a quite realistic assumption – it is widely agreed that insiders are often more of a threat than outsiders.

Security Assumption 3 *The adversary may be a legitimate protocol participant (an insider), or an external party (an outsider), or a combination of both.*

To overcome the attack, the names of the users who are to share K_{AB} need to be bound cryptographically to the value of K_{AB}. This leads to the protocol shown in Fig. 1.5 where the names of A and B are included in the encrypted messages received from S. It can easily be checked that in this

Petitioner Samsung Ex-1036, 0030

protocol neither of the two attacks on the protocol of Fig. 1.2 will succeed. It is a necessary property of the encryption algorithm used by S that it is not possible to alter the value of the encrypted messages. The importance of distinguishing between this *integrity* property and the *confidentiality* property of cryptographic algorithms will be discussed further in Sect. 1.4.

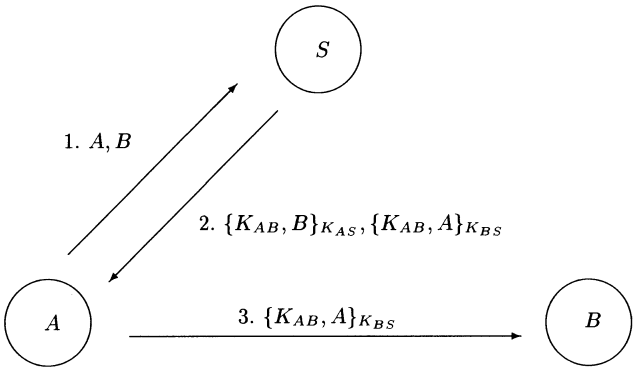


Fig. 1.5. Third protocol attempt

1.2.3 Replay

So far our protocol has improved to the point where an adversary is unable to attack it by either eavesdropping or altering the messages sent between the legitimate users. However, even now the protocol is not good enough to provide security in normal operating conditions. The problem stems from the difference in quality between the long-term key-encrypting keys shared initially with S, and the session keys K_{AB} generated for each protocol run.

One reason that a new key is generated for each session is that session keys are expected to be vulnerable to attack. They are likely to be used with a variety of data of regular formats, making them targets for cryptanalysis; also they may be placed in relatively insecure storage and could easily be discarded carelessly after the session is closed. A second reason for using new session keys is that communications in different sessions should be separated. In particular, it should not be possible to replay messages from previous sessions.

For these reasons a whole class of attacks becomes possible based on the notion that old keys may be *replayed* in a subsequent session. Notice that even if A is careful in the management of session keys used by her, compromise of a session key by B may still allow replay attacks when A communicates with B.

Petitioner Samsung Ex-1036, 0031

Security Assumption 4 *An adversary is able to obtain the value of the session key K_{AB} used in any sufficiently old previous run of the protocol.*

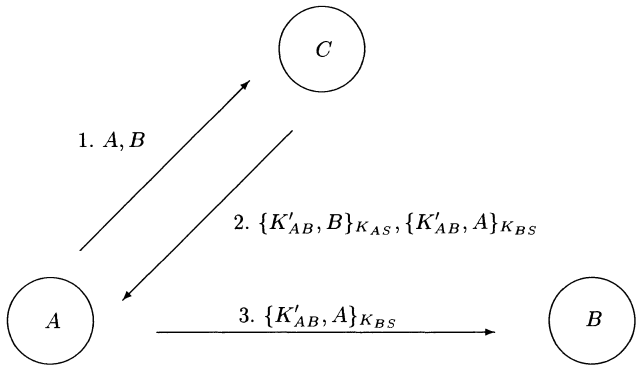


Fig. 1.6. Attack on third protocol attempt

Figure 1.6 shows a replay attack on our third protocol attempt. This time C intercepts the message from A to S – indeed S plays no part in the protocol. The key K'_{AB} is an old session key used by A and B in a previous session; by Security Assumption 1, C can be expected to know the encrypted messages via which K'_{AB} was transported to A and B. By Security Assumption 4, C can be expected to know the value of K'_{AB}. Thus when A completes the protocol with B, C is able to decrypt subsequent information encrypted with K'_{AB} or insert or alter messages whose integrity is protected by K'_{AB}.

Notice that the replay attack in Fig. 1.6 can still be regarded as successful even if C has not obtained the value of K'_{AB}. This is because C has succeeded in making A and B accept an old session key. Such an attack may be useful to C because it allows C to replay messages protected by K'_{AB} which were sent in the previous session. In addition it enables C to obtain more ciphertext with the same key which might aid in cryptanalysis.

There are various mechanisms that may be employed to allow users to check that session keys have not been replayed. These are considered in detail in Sect. 1.5, but for now we will improve our protocol using the popular method called *challenge–response*. In this method, A will generate a new random value N_A commonly known as a *nonce* (a number used only once).

Definition 1.1. *A* nonce *is a random value generated by one party and returned to that party to show that a message is newly generated.*

Principal A sends her nonce N_A to S at the start of the protocol together with the request for a new key. If this same value is received with the session

Petitioner Samsung Ex-1036, 0032

key then A can deduce that the key has not been replayed. This deduction will be valid as long as the session key and nonce are bound together cryptographically in such a way that only S could have formed such a message.

Since B does not directly contact the server S, it is inconvenient for him to send his own nonce to S to be returned with K_{AB}. How, then, is he able to gain the same assurance as A that K_{AB} has not been replayed?

If the encrypted key for B is included in the encrypted part of A's message, then A can gain assurance that it is fresh. It is tempting to believe that A may pass this assurance on to B in an extra handshake: B will generate a nonce N_B and send this to A protected by K_{AB} itself. Then A can use the session key to send a related reply to B. This leads to a fourth protocol attempt shown in Fig. 1.7.

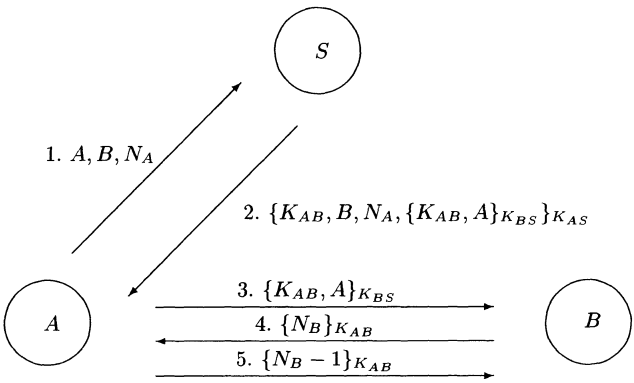


Fig. 1.7. Fourth protocol attempt (Needham–Schroeder)

The protocol in Fig. 1.7, which we have reached by a series of steps, is one of the most celebrated in the subject of cryptographic protocols. It was published by Needham and Schroeder in 1978 [249] and has been the basis for a whole class of related protocols. Unfortunately the original Needham–Schroeder protocol is vulnerable to an almost equally celebrated attack due to Denning and Sacco [99]. Their attack illustrates that there was a flaw in the above argument used to justify the protocol design. This can be pinpointed to an assumption that only A will be able to form a correct reply to message 4 from B. Since the adversary C can be expected to know the value of an old session key, this assumption is unrealistic. In the attack in Fig. 1.8, C masquerades as A and is thus able to persuade B to use the old key K'_{AB}.

As usual, once an attack has been spotted, it is relatively easy to suggest ways of overcoming it. The method we choose here is to throw away the

Petitioner Samsung Ex-1036, 0033

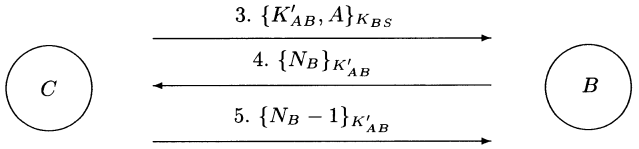


Fig. 1.8. Attack on fourth protocol attempt

assumption that it is inconvenient for both B and A to send their challenges to S. This leads to our final protocol shown in Fig. 1.9.

It would be rash to claim that this protocol is secure before giving a precise meaning to that term. Yet we can say that it avoids all the attacks that we have met so far, as long as the cryptographic algorithm used provides the properties of both confidentiality and integrity, and the server S acts correctly. The security of a protocol must always be considered relative to its goals; the different possible goals are considered in detail in Chap. 2, as well as ways to gain greater assurance that they are met.

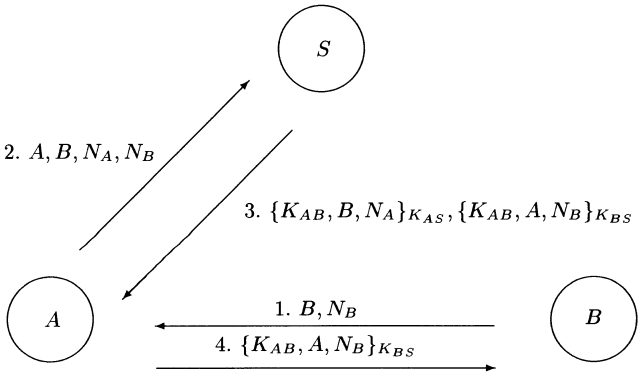


Fig. 1.9. Fifth protocol attempt

To enable both users to send their nonces to S, the protocol is now initiated by B who sends his nonce, N_B, first to A. A adds her nonce, N_A, and sends both to S who is now able to return K_{AB} in separate messages for A and B, which can each be verified as fresh by their respective recipients. Although it may seem that we have achieved more than the protocol in Fig. 1.7 using fewer messages, A has in fact achieved less in this protocol. This is because A in Fig. 1.7 could verify not only that the key is new and known only by A,

Petitioner Samsung Ex-1036, 0034

B and S, but also that B has in fact received the key. This property of *key confirmation* is achieved due to B's use of the key in message 4, assuming that $\{N_B\}_{K_{AB}}$ cannot be formed without knowledge of K_{AB}. We will discuss key confirmation further in Chap. 2. In the protocol of Fig. 1.9, neither A nor B can deduce at the end of a successful protocol run that the other has actually received K_{AB}. We leave it as an exercise for the reader to construct variant protocol runs showing why this is the case.

It is worth noting that it has been a very common pattern for published protocols to be subsequently found to be flawed. Each time a new protocol is designed and an attack is found our understanding of protocol design improves. The frequent occurrence of such attacks should be a caution, particularly for implementers of security protocols. Much of the recent research in cryptographic protocols has been devoted to remedying the situation.

1.3 Protocol Architectures

In Sect. 1.2 the protocol architecture consisted of an on-line trusted server, which generated the session key, and two other principals. Here we consider alternative architectures. There are three features that we regard as architectural criteria to classify different protocols: which keys are already established, how the session key is generated, and how many users a protocol is designed to serve. Note that the second criterion is applicable only to protocols concerned with key establishment.

1.3.1 Existing Cryptographic Keys

As a matter of general principle it is not possible to establish an authenticated session key without existing secure channels already being available. In fact this principle can be stated formally and proven to be correct [55]. Therefore, except for the possibility of secure physical establishment of keys, it is essential either that keys are already shared between different principals or that certified public keys are available. We may regard the use of public key certificates as equivalent to the use of an *off-line* server. Therefore if two principals wish to establish a new session key there are essentially three possibilities.

1. The principals already share a secret key.
2. An off-line server is used. This means that the principals possess certified public keys. In order to verify the authenticity of a public key it may be necessary to verify a chain of certificates.
3. An on-line server is used. This means that each principal shares a key with a trusted server. In order to pass information between parties it may be necessary to pass it via a chain of on-line servers.

Petitioner Samsung Ex-1036, 0035

1.3.2 Method of Session Key Generation

There are various ways that may be employed to generate session keys in a key establishment protocol. In the following, we use the term 'user' to mean an entity who will use the session key for subsequent communication. We continue to use the term 'principal' to mean an entity who will engage in the protocol. For example, in Fig. 1.9, A and B are users, whereas S is not. All three of A, B and S are principals.

Definition 1.2. *A* key transport *protocol is a key establishment protocol in which one of the principals generates the key and this key is then transferred to all protocol users.*

Definition 1.3. *A* key agreement *protocol is a key establishment protocol in which the session key is a function of inputs by all protocol users.*

Definition 1.4. *A* hybrid *protocol is a key establishment protocol in which the session key is a function of inputs by more than one principal, but not by all users. This means that the protocol is a key agreement protocol from the viewpoint of some users, and a key transport protocol from the viewpoint of others.*

In the literature, key transport protocols have typically been associated with protocols using an on-line server, while key agreement has been associated with protocols using an off-line server. However, there is no need for this restriction and there are examples of key agreement protocols in which an on-line server has an input to the session key, and key transport protocols with an off-line server.

1.3.3 Number of Users

A final component of the protocol architecture is the number of principals that take part in the protocol. Most key establishment protocols have concentrated on the case where two users wish to establish a session key for point-to-point communications. To extend these to *conference key* protocols can complicate matters a great deal. These protocols are examined in Chap. 6. It is quite possible for a conference key protocol to look like a key transport protocol to some principals (who receive the session key on a cryptographic channel) and a key agreement protocol to other principals (who have an input to the session key).

1.3.4 Example

The three criteria mentioned above may be used to classify key establishment protocols; for a given number of users the first two criteria (which keys exist and which method of key generation is used) give rise to $3 \times 3 = 9$ different

Petitioner Samsung Ex-1036, 0036

classes of protocols. Lest it be doubted by the reader that all classes of protocols can be populated we give in Protocol 1.2 an example in a very unusual class. This protocol uses an on-line server, uses hybrid key generation and has two users.

The initial state is the familiar one where A and B share long-term keys K_{AS} and K_{BS} respectively with S. The session key is calculated as $K_{AB} = f(N_B, N_S)$ for a suitable function f, where N_B and N_S are random values generated by B and S respectively. From B's viewpoint this is like a key agreement protocol because B has input to the key. From A's viewpoint it looks like a key transport protocol.

Goal: Hybrid key establishment

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow S : \{N_B, A, B\}_{K_{BS}}, N_A$
3. $S \rightarrow A : \{K_{AB}, A, B, N_A\}_{K_{AS}}, N_S$
4. $A \rightarrow B : N_S, \{A, B\}_{K_{AB}}$
5. $B \rightarrow A : \{B, A\}_{K_{AB}}$

Protocol 1.2: A protocol in an unusual class

Upon receiving message 2, S must check that the value obtained by decrypting the field B is the same as the identity of the principal whose key is used to decrypt the message. If the same identity is received with A's identity, then S can deduce that A and B are to share the session key.

1.4 Cryptographic Properties

In this section we highlight the importance of distinguishing the possible different properties that may be provided by cryptographic algorithms. A good understanding of the algorithms and methods of cryptography is highly beneficial in assessing cryptographic protocols, but it is not the purpose of this section to develop such an understanding, only to give a brief overview.

Much of the material in this section is summarised from the book of Menezes, van Oorschot and Vanstone [234]. They identify four fundamental objectives that may be achieved by cryptographic algorithms.

Confidentiality ensures that data is only available to those authorised to obtain it. This is usually achieved through encryption of the data so that only those with the correct decryption key can recover it. In cryptographic protocols confidentiality is essential to ensure that keys and other data are available only as intended.

Petitioner Samsung Ex-1036, 0037

Data Integrity ensures that data has not been altered by unauthorised entities. This can be achieved through use of hash functions in combination with encryption, or by use of a message authentication code to create a separate check field. Data integrity is essential in most cryptographic protocols to protect elements such as identity fields and nonces.

Data Origin Authentication guarantees the origin of data. It is a fundamental step in achieving entity authentication in protocols as well as establishing keys. Since altering the data must alter its origin, we may say that data origin authentication implies data integrity. Although it is in principle possible to achieve data integrity without origin authentication, they are normally achieved by the same mechanisms.

Non-repudiation ensures that entities cannot deny sending data that they have committed to. This is typically provided using a digital signature mechanism. Non-repudiation is rarely a requirement in protocols for authentication and key establishment, but it automatically provides the important data integrity and data origin authentication services.

Some cryptographic transformations can provide more than one of these properties. It is often noted that if the message source has sufficient redundancy, for example a natural language source like English, then encryption for confidentiality automatically provides some degree of data integrity. However, it is a serious error to believe that just because a decrypted message makes sense it must be the same message that was sent. Before looking at the properties of cryptographic algorithms meeting the different objectives, let us consider an example illustrating the importance of identifying which properties are provided.

The *one-time pad* is one of the simplest cryptosystems and it is also provably secure in a very strong sense. If we assume that the message is a string of n bits $b_1, b_2, \ldots, b_n$ then the key is a random string of bits $k_1, k_2, \ldots, k_n$. Encryption takes place one bit at a time and the ciphertext string $c_1, c_2, \ldots, c_n$ is found by adding, modulo 2, each message bit with the corresponding key bit.

$$c_i = b_i \oplus k_i \text{ for } 1 \leq i \leq n.$$

Decryption is the same as encryption, since adding modulo 2 is a self-inverse operation. Now, suppose that we know that the one-time pad is used to secure the amount sent in a funds transfer. Then we may alter any of the bits of ciphertext and change the amount sent even though we cannot find what the original amount was. If we know that amounts are usually small then we could alter the most significant bit and expect to increase substantially the amount sent. This simple example shows that the one-time pad on its own provides no data integrity even though it provides perfect secrecy.

We now consider definitions for the cryptographic mechanisms that are typically used to provide the main cryptographic services. Table 1.1 summarises the notation which we will use for these mechanisms throughout this book.

Petitioner Samsung Ex-1036, 0038

Table 1.1. Summary of notation for cryptographic algorithms

$E_A(M)$	Public key encryption of message M with public key of entity A.
$\{M\}_K$	Symmetric encryption of message M with shared key K.
$MAC_K(M)$	Message authentication code of M using shared key K.
$Sig_A(M)$	Digital signature of message M generated by entity A.

The definitions we will give are informal, but rely on an intuitive understanding of what it means for a computation to be easy or difficult. In complexity theory a computation is said to be easy (or feasible) if the time it takes to complete increases as a polynomial (which could be a constant) of the input length. If the time required increases faster than any polynomial we may say that the computation is hard (or infeasible). Sometimes it is more meaningful to have a specific number of operations in mind. Nowadays it is accepted that performing up to 2^{40} fundamental computational operations is 'easy' while performing 2^{80} such operations is 'hard', with something of a grey area in between.

1.4.1 Confidentiality

Definition 1.5. *An* encryption scheme *consists of three sets: a key set $\mathcal{K}$, a message set $\mathcal{M}$, and a ciphertext set $\mathcal{C}$ together with three algorithms.*

1. *A key generation algorithm, which outputs a valid encryption key $K \in \mathcal{K}$ and a valid decryption key $K^{-1} \in \mathcal{K}$.*
2. *An encryption algorithm, which takes an element $m \in \mathcal{M}$ and an encryption key $K \in \mathcal{K}$ and outputs an element $c \in \mathcal{C}$ defined as $c = E_K\{m\}$. The encryption algorithm may be randomised so that a different c will result given the same m.*
3. *A decryption function, which takes an element $c \in \mathcal{C}$ and a decryption key $K^{-1} \in \mathcal{K}$ and outputs an element $m \in \mathcal{M}$ defined as $m = D_{K^{-1}}\{c\}$. We require that $D_{K^{-1}}\{E_K\{m\}\} = m$.*

An encryption scheme is a *symmetric* algorithm if $K = K^{-1}$. In contrast, in an *asymmetric* or *public key* encryption algorithm K and K^{-1} are different and it is computationally hard to obtain the private key K^{-1} from the public key K. As shown in Table 1.1 we use different notation to distinguish between symmetric and asymmetric encryption.

Although all encryption algorithms are intended to hide information from the adversary, different algorithms can provide different properties. The following definition tries to capture the idea that the ciphertext should not be any help to the adversary in learning anything new, including the plaintext.

Petitioner Samsung Ex-1036, 0039

Definition 1.6. *An encryption scheme provides* semantic security *if anything that can be efficiently computed given the ciphertext can also be efficiently computed without the ciphertext.*

A useful equivalent characterisation of semantic security is *indistinguishability.* This means that given the ciphertext corresponding to one out of two known messages the adversary cannot guess with probability greater than 1/2 which one is the plaintext. In this challenge the adversary is able to obtain the encryption of any chosen messages; in other words the adversary is allowed a *chosen plaintext* attack.

Definition 1.7. *An encryption scheme provides* non-malleability *if it is infeasible to take an existing ciphertext and transform it into a related ciphertext without knowledge of the input plaintext.*

The property of non-malleability is strictly stronger than semantic security. Indeed the definition is known to be equivalent to the indistinguishability property mentioned above when the adversary is additionally given the decryption of any chosen ciphertexts; an algorithm with non-malleability is secure against a *chosen ciphertext* attack.

Since non-malleability is a stronger property than semantic security it is not surprising that algorithms providing non-malleability in general have greater computational requirements and message expansion than those providing only semantic security. The extent of the overhead varies from algorithm to algorithm, but where computation and bandwidth are at a premium it is important to know whether an algorithm used in a protocol must provide non-malleability. There are many key establishment protocols in which non-malleability is required, although some protocol designers have only implicitly recognised this.

1.4.2 Data Origin Authentication and Data Integrity

Data authentication and integrity are essential in most protocols for authentication and key establishment. These two cryptographic services are strongly connected and are typically both provided together using the same mechanism. This is because the origin of a message can only be guaranteed if the message has not changed since it was formed.

There are two standard mechanisms. One is to use a hash function that has the property of being *one-way.*

Definition 1.8. *A function $f : X \to Y$ is* one-way *if it is computationally easy to calculate $y = f(x)$ given $x \in X$, but computationally difficult to find any x with $f(x) = y$ for almost all values of $y \in Y$.*

A one-way function that is appended to a message before encryption is known as a *manipulation detection code* (MDC). The idea is that without

Petitioner Samsung Ex-1036, 0040

knowledge of the message or the MDC the adversary is unable to alter the message without destroying the correctness of the MDC. More generally, the adversary cannot create any encrypted message which, when decrypted, would have a correct MDC.

Definition 1.9. *A* message authentication code *(MAC) is a family of functions parametrised by a key K such that $MAC_K(m)$ takes a message m of arbitrary length and outputs a fixed length value and satisfying:*

1. *it is computationally easy to calculate $MAC_K(m)$ given K and m;*
2. *given any number of MAC values for a given K, it is computationally hard to find any valid MAC value for any new message.*

The second mechanism for providing data origin authentication and data integrity is to append a MAC to a message which may be either in plaintext or encrypted. On receipt of the MAC, the recipient who has the correct key is able to recompute the MAC from the message and verify that it is the same as that received.

1.4.3 Non-repudiation

Non-repudiation is usually provided through a digital signature mechanism. Although non-repudiation is not a property that is typically required for authentication or key establishment protocols, nevertheless digital signatures are a common element in their construction. This is because digital signatures also provide authentication and data integrity services; their implementation through public keys makes them useful to provide these essential services.

Definition 1.10. *A* digital signature algorithm *consists of three sets: a key set $\mathcal{K}$, a message set $\mathcal{M}$, and a signature set $\mathcal{S}$ together with three algorithms.*

1. *A key generation algorithm, which outputs a valid signature key $K \in \mathcal{K}$ and a valid verification key $K^{-1} \in \mathcal{K}$.*
2. *A signature generation algorithm, which takes an element $m \in \mathcal{M}$ and a signature key $K \in \mathcal{K}$ and outputs an element $s \in \mathcal{S}$. We will write $s = Sig_A\{m\}$ where K is the signature generation key of entity A. The signature generation algorithm may be randomised so that a different output will result given the same m.*
3. *A verification function, which takes a signature $s \in \mathcal{S}$, a message $m \in \mathcal{M}$, and a verification key $K^{-1} \in \mathcal{K}$ and outputs an element $v \in \{0, 1\}$. If $v = 1$ then we say the signature is* valid *or if $v = 0$ we say that the signature is* invalid.

A digital signature algorithm is regarded as secure if it is computationally hard for the adversary to find a valid signature of any message that has not been previously signed, even given many previously signed messages.

Petitioner Samsung Ex-1036, 0041

In the above definition we have stated that it is necessary to have both the signature s and the message m in order to verify the signature. This is sometimes called a signature *with appendix*. In contrast, a signature that can be verified without separate knowledge of the message is called a signature with *message recovery*. Throughout the book we will use the notation $Sig_A(m)$ to denote a signature with appendix of message m by entity A. Notice that even though we assume that m must be available in order to verify $Sig_A(m)$, it does not follow that an adversary cannot obtain information about m from $Sig_A(m)$.

1.4.4 Examples of Cryptographic Algorithms

Table 1.2 lists some of the best known cryptographic algorithms which provide the different types of cryptographic properties discussed earlier in this section. Some of these algorithms have been proven secure in the sense that there is a proven reduction to some well-known difficult problem. However, the reader should be aware that these reductions have varying complexities and the underlying problems have no proof regarding their absolute difficulty. The IEEE P1363 standard [154] covers a few public key algorithms and how to implement them. The recent book by Mao [221] gives a detailed description of such algorithms and explains their security proofs.

Table 1.2. Some well-known cryptographic algorithms and their properties

Algorithm	Type	Cryptographic service
AES [98]	Block Cipher	Confidentiality
ElGamal encryption [106]	Public Key Cipher	Semantic Security
Cramer–Shoup [97]	Public Key Cipher	Non-malleability
RSA-OAEP [32]	Public Key Cipher	Non-malleability
RSA Signature [274, 154]	Digital Signature	Non-repudiation
DSS [247]	Digital Signature	Non-repudiation
SHA-1 [245]	Hash Function	One-way Function
HMAC [26]	MAC	Data Integrity

The Advanced Encryption Standard (AES) is given as an example of a block cipher. The security properties of any block cipher depend critically on the *mode of operation* of the cipher, and so we have just given the generic term 'confidentiality' as its provided security service. The reader who is unfamiliar with these algorithms is encouraged to refer to the cited literature for further details.

Modern complexity-theoretic definitions are nowadays in frequent use in the literature of cryptography. The definitions given above are informal ver-

Petitioner Samsung Ex-1036, 0042

sions of these. Study of the formal definitions is helpful in gaining a deeper understanding of what algorithms are appropriate to use in a particular protocol, but is outside the scope of this book. Relationships exist between the different formal definitions of confidentiality [28]. Similarly formal definitions of security for digital signatures are available. Certain algorithms have been proven to possess the different cryptographic properties defined above, given certain reasonable assumptions on the underlying mathematical problems, and sometimes on the existence of functions with random-like properties. In many protocols the cryptographic algorithms used are not specified. However, it is helpful to know that appropriate algorithms do actually exist.

1.4.5 Secret Sharing

Secret sharing is a mechanism that allows the owner of a secret to distribute *shares* amongst a group. The owner of the secret is often called the *dealer*. Individual shares are of no help in recovering the secret, but if all shares in some pre-defined *access sets* are available then the secret can be collectively found. A (t, n) *threshold scheme* is a secret sharing scheme for which n shares are distributed, such that any set of t (or more) shares is sufficient to obtain the secret, while any set of $t-1$ (or fewer) shares is of no help in recovering the secret.

There are some similarities between secret sharing and key establishment for groups since, for both, a group of users cooperates to derive a secret value. However, a secret sharing scheme on its own lacks the means to provide fresh keys, to distribute keys to principals, and to provide key authentication. We look at some specific schemes based on secret sharing in Chap. 6.

The most well-known threshold scheme is due to Shamir [294] and is based on the use of polynomial interpolation. This allows any polynomial of degree d to be completely recovered once any $d+1$ points on it are known. Polynomial interpolation works over any field, but in cryptographic applications the field is typically $\mathbb{Z}_p$, the field of integers modulo p, for some prime p.

In order to share a secret $s \in \mathbb{Z}_p$ in Shamir's (t, n) threshold scheme, the dealer generates a polynomial of degree $t-1$:

$$f(z) = a_0 + a_1 z + \ldots + a_{t-1} z^{t-1}$$

with coefficients randomly chosen in $\mathbb{Z}_p$ except for $a_0 = s$. The shares are values $f(x)$ with $1 \leq x \leq n$. If any t shares are known then s can be recovered. For example, if $f(1), f(2), \ldots, f(t)$ are known then:

$$s = \sum_{i=1}^{t} f(i) \prod_{1 \leq j \leq t, j \neq i} \frac{j}{j-i}.$$

Given any $t-1$ points on the polynomial (excluding the value at 0), all possible values for $f(0)$ can be obtained given one extra point. Consequently,

Petitioner Samsung Ex-1036, 0043

absolutely no information about the secret can be obtained if $t-1$ or fewer shares are known.

1.5 Freshness

We have already seen that for protocols used to establish a session key, one of the requirements is that each user of the key should be able to verify that it is new and not replayed from an old session. This property extends to a variety of other protocol types. For example, protocols designed to achieve authentication in real time also need to ensure that messages sent are not replays. Thus we see that the need to ensure that message elements are new, or *fresh*, is a very common protocol requirement. It is thus worthwhile to look at the typical ways that freshness is achieved in existing protocols.

Before considering detailed mechanisms we note that there are essentially two different ways that freshness of a value may be guaranteed to a particular user. The first is where the user has a part in choosing the value (which will often be a new session key), while the second is where the user has to rely on something received with the value that is known to be fresh itself. A typical instance of the first case is in a key agreement protocol. Here two users A and B both choose an input, N_A and N_B respectively, to a new session key K_{AB}. The session key will be formed by choosing some function of the inputs:

$$K_{AB} = f(N_A, N_B).$$

A desirable property of the function f is that it should not be possible for A or B to force an old value of K_{AB} even if the other's input is known. This means that each user has independent assurance that K_{AB} is fresh. This property is achieved for A if, once N_A is chosen, B is unable to choose N_B in such a way that K_{AB} is an old value. If we define the function $g(.) = f(N_A, .)$ then this means that g must be a one-way function. In practice it may be necessary to add other conditions to disallow exceptional values. A symmetrical condition must also hold to provide the same property for B. We will consider the desirable properties of such combining functions in more detail in Chap. 5.

Let us turn now to the second case, where freshness depends on something received with the message. Suppose, as is most common, that a principal A wishes to verify the freshness of a session key K_{AB} that has been generated by some principal S (which may be a server or perhaps is the principal B that shares K_{AB}). Principal A must trust S to freshly generate K_{AB} but needs to be sure that the message received is not an old message that has been replayed by the adversary. Assume that A receives the message field $F(K_{AB}, N)$ which is a function of K_{AB} and a freshness value N. What are the properties required of F to allow the recipient to be sure that the composite message is fresh? F must provide data origin authentication and data integrity

Petitioner Samsung Ex-1036, 0044

so that the recipient can deduce that $F(K_{AB}, N)$ was generated by S and has not been altered. If A can be sure that N is fresh then she can also be sure that $F(K_{AB}, N)$ is fresh. Since S is trusted to generate and authenticate only fresh keys, A can therefore deduce that K_{AB} is fresh. We next consider the different forms that the freshness value N can take.

A freshness value must have the property that it can be guaranteed not to have been used before. (In some protocols values used for freshness also are required to have other properties, but this is because they are used for other purposes as well as to ensure freshness.) There are three common types of freshness value used: timestamps, nonces and counters. Gong [126] has classified the various ways that these types may be used in a protocol.

Timestamps The sender of the message adds the current time to the message when it is sent. This is checked by the recipient when the message is received by comparing with the local time. If the received timestamp is within an acceptable window of the current time then the message is regarded as fresh. The difficulty of using timestamps is that synchronised time clocks are required and must be maintained securely. Gong [123] has pointed out that if a principal's clock is *advanced* beyond the time in the rest of the system, a vulnerability can exist even after the clock has been corrected. This is because an adversary could have captured, and suppressed, a message that will become fresh in the future. Gong calls this a *suppress relay* attack.

Nonces (Random challenges) The recipient, A, of the message generates a nonce, N_A, and passes it to the sender of the message, B. The nonce N_A is then returned with the message after processing with some cryptographic function f as shown in Protocol 1.3.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : f(N_A, \ldots)$

Protocol 1.3: Use of a nonce (random challenge)

A checks the nonce on receipt and deduces that the message is fresh because the message cannot have been formed before the nonce was generated. A disadvantage of using a challenge is that it requires an interactive protocol which may add to both the number of messages and the number of message exchanges required. Attention must also be paid to the quality of random numbers produced, since if the nonce to be used is predictable a valid reply can be obtained in advance and later replayed (a preplay attack).

Counters The sender and recipient maintain a synchronised counter whose value is sent with the message and then incremented. A disadvantage of counters is that state information must be maintained for each potential

Petitioner Samsung Ex-1036, 0045

communication partner. Management of counters can also cause problems in the presence of channel errors. Gong [126] pointed out that if a counter is not synchronised with the receiver then preplay attacks are possible.

Mitchell [235] has suggested a hybrid between counters and timestamps. His idea is to use a counter based on real time. For example, the counter may be the time in seconds from some starting point: this allows a 32-bit counter to be used for over 136 years before repeating. The motivation for such a suggestion is that a server authenticating multiple clients can easily recover from loss of the counter states by checking its real-time clock. Mitchell discusses an application in mobile telephony in which the mobile handset may not have a reliable clock but is still required to verify freshness of session keys generated by a server.

A different way of obtaining freshness, which is less common, is if an element is transformed with a cryptographic key that is known to be fresh. For this method to work the cryptographic transformation must provide data integrity. Then the recipient can be sure that the fresh key has been used to form the message, and so the message must have been formed after the key was formed.

1.6 Types of Attack on Protocols

In our move towards a secure protocol in Sect. 1.2 a number of attacks were encountered. The purpose of this section is to summarise them, together with many others that were not encountered. Before doing so we should sound a couple of notes of caution.

Firstly, for many protocols the list will not be complete. The ways in which the adversary may interact with one or more protocol runs are infinite. There are almost bound to be attacking possibilities that we have omitted. Indeed, it could be argued that it is not tremendously helpful to know that a protocol is not vulnerable to a certain list of threats; what is really required is confidence that it meets its security objectives given a known list of assumptions. On the other hand, we should not underestimate the usefulness of a list of typical weaknesses to check against.

Secondly, different protocols have different objectives. In our examples above the stated aim was to derive securely a new session key. However, some protocols may have no material to convey confidentially, being concerned solely with real-time authentication. Others may have additional goals, such as confirmation that the session key was correctly received by the protocol users. The difficult issue of defining appropriate goals for cryptographic protocols is the subject of the next chapter and so we will not address it here. Naturally, whether or not a protocol achieves particular goals depends on what attacks are deemed possible.

Bearing in mind these caveats, we now consider the most commonly encountered threats to cryptographic protocols. Table 1.3 lists and defines the

Petitioner Samsung Ex-1036, 0046

attacks and they are considered in turn in more detail below. It should be noted that there are other ways to classify attacks; an alternative list, with examples, is given by Carlsen [83].

Table 1.3. Types of protocol attack

Eavesdropping	The adversary captures the information sent in the protocol.
Modification	The adversary alters the information sent in the protocol.
Replay	The adversary records information seen in the protocol and then sends it to the same, or a different, principal, possibly during a later protocol run.
Preplay	The adversary engages in a run of the protocol prior to a run by the legitimate principals.
Reflection	The adversary sends protocol messages back to the principal who sent them.
Denial of Service	The adversary prevents or hinders legitimate principals from completing the protocol.
Typing Attacks	The adversary replaces a (normally encrypted) protocol message field of one type with a (normally encrypted) message field of another type.
Cryptanalysis	The adversary gains some useful leverage from the protocol to help in cryptanalysis.
Certificate Manipulation	The adversary chooses or modifies certificate information to attack one or more protocol runs.
Protocol Interaction	The adversary chooses a new protocol to interact with a known protocol.

1.6.1 Eavesdropping

Eavesdropping is perhaps the most basic attack on a protocol. Nearly all protocols address eavesdropping by using encryption. It is obvious that encryption must be used to protect confidential information such as session keys. In certain protocols there may be other information that also needs to be protected. An interesting example is that protocols for key establishment in mobile communications usually demand that the identity of the mobile station remain confidential. Eavesdropping is sometimes distinguished as being a *passive* attack since it does not require the adversary to disturb the communications of legitimate principals. The other attacks we consider all require

Petitioner Samsung Ex-1036, 0047

the adversary to be *active*. It should be remembered that many sophisticated attacks include eavesdropping of protocol runs as an essential part.

1.6.2 Modification

If any protocol message field is not redundant then modification of it is a potential attack. Use of cryptographic integrity mechanisms is therefore pervasive in protocols for authentication and key establishment.

Whole messages, as well as individual message fields, are vulnerable to modification. Many attacks do not alter any known message field at all, but split and re-assemble fields from different messages. This means the integrity measures must cover all parts of the message that must be kept together; encryption of these fields is not enough. Examples of attacks on protocols in which encryption does not provide the required integrity properties are given by Stubblebine and Gligor [311] and by Mao and Boyd [223].

1.6.3 Replay

Replay attacks include any situation where the adversary interferes with a protocol run by insertion of a message, or part of a message, that has been sent previously in any protocol run. We may regard replay as another fundamental type of attack which is often used in combination with other attack elements. Just as almost all protocols address eavesdropping and modification attacks by using cryptography, almost all protocols include elements to address possible replay attacks. Various means to combat replay were discussed in Sect. 1.5.

It is possible for the replayed message in an attack to have been originally part of a protocol run that happened in the past (an example of such an attack was given in Sect. 1.2.3). Alternatively the replayed material may be from a protocol run that takes place at the same time as the attacking run. Syverson [313] has produced a taxonomy of replay attacks based upon this distinction.

1.6.4 Preplay

Preplay might be regarded as a natural extension of replay, although it is not clear that this is really an attack that can be useful on its own. An interesting example of an attack that employs preplay is the so-called *triangle attack* of Burmester [74] which will be presented in Chap. 5.

1.6.5 Reflection

Reflection is really an important special case of replay. A typical scenario is where two principals engage in a shared key protocol and one simply returns a challenge that is intended for itself. This attack may only be possible if parallel runs of the same protocol are allowed but this is often a realistic assumption.

Petitioner Samsung Ex-1036, 0048

For example, if one principal is an Internet host, it may accept sessions from multiple principals while using the same identity and set of cryptographic keys. The possibility of instigating several protocol runs simultaneously is another common and realistic strategy for the adversary.

Security Assumption 5 *The adversary may start any number of parallel protocol runs between any principals including different runs involving the same principals and with principals taking the same or different protocol roles.*

Consider Protocol 1.4, which gives a very basic example. Suppose A and B already share a secret key K and choose respective nonces N_A and N_B for use in the protocol. The protocol is intended to mutually authenticate both parties by demonstrating knowledge of K.

1. $A \rightarrow B : \{N_A\}_K$
2. $B \rightarrow A : \{N_B\}_K, N_A$
3. $A \rightarrow B : N_B$

Protocol 1.4: A protocol vulnerable to reflection attack

On receipt of message 2, A deduces that it must have been sent by B since only B has K. However, if A is willing to engage in parallel protocol runs then there is another possibility, namely that message 2 was originally formed by A. An adversary C can successfully complete two runs of the protocol, as shown in Attack 1.1.

1. $A \rightarrow C : \{N_A\}_K$
1′. $C \rightarrow A : \{N_A\}_K$
2′. $A \rightarrow C : \{N'_A\}_K, N_A$
2. $C \rightarrow A : \{N'_A\}_K, N_A$
3. $A \rightarrow C : N'_A$
3′. $C \rightarrow A : N'_A$

Attack 1.1: Reflection attack on Protocol 1.4

Immediately after receiving the first message, C starts another run of the protocol with A, and reflects back the message received from A. The reply allows C to respond to the first message and then both runs of the protocol can be completed. In fact all cryptographic processing has been performed by A, while A believes two protocol runs have been completed with B.

These sorts of attacks are sometimes called 'oracle attacks' since A acts as an oracle to C by presenting the required decryption. An extensive treatment of reflection attacks is given by Bird *et al.* [41].

Petitioner Samsung Ex-1036, 0049

1.6.6 Denial of Service

In a denial of service attack (often contracted to *DoS attack*) the adversary prevents legitimate users from completing the protocol. Denial of service attacks in practice take place against servers who are required to interact with many clients. Attacks can be divided into those that aim to use up the computational resources of the server (*resource depletion attacks*) and those that aim to exhaust the number of allowed connections to the server (*connection depletion attacks*).

As a matter of principle it seems that it is impossible to prevent denial of service attacks completely. Any attempt to establish a connection must either result in allocation of a connection or use some computational work to establish that the attempt is invalid. Nevertheless there are certain measures that may be taken to reduce the impact of denial of service attacks and some protocols are much more vulnerable to this sort of attack than others, so it is important not to ignore this issue.

- Aura and Nikander [21] suggested using *stateless connections* to protect against connection depletion attacks. Their idea is to make the client store all the state information required by the server and return it to the server as necessary with each message sent. In this way the server need not store any state information. Of course it is necessary that the state information returned to the server can be verified by the server to be authentic and it may also need to be confidential. Therefore there is an overhead in both communication and computation incurred by transforming protocols to provide stateless connections.
  A practical mechanism for delaying the need for state at the server is the use of *cookies*, which first seems to have been suggested by Karn and Simpson for their Photuris protocols (most recent version 1999 [177] but originally published in 1995). When a client attempts to make a connection the server sends back a cookie. This procedure is similar to the familiar use of cookies by World Wide Web servers, but here the cookies take a special form: they are a function of a secret known only to the server and other information unique to the particular connection. At this stage the server stores no state for this request. The client needs to return the cookie in the next message and its validity can be checked by the server from the information sent and its secret. The idea is to ensure, before investing significant resources, that the client is making a unique request for connection. This technique prevents denial of service attacks in which the adversary sends random connection requests. By changing the server secret on a regular basis (typically every 60 seconds) even the same client cannot make unlimited connection requests.
- Meadows [231] suggests that in order to protect against connection depletion each message in a protocol must be authenticated. However, to minimise on possibly wasted computation the authentication can be weak

Petitioner Samsung Ex-1036, 0050

at the start of the protocol and increase in strength with subsequent messages. Cookies formed by the server, and which must be returned by the client, may form the weak authentication. Meadows develops a formal framework based on the idea of *fail–stop protocols* introduced by Gong and Syverson [132] which abort as soon as a bogus message is discovered.

- Juels and Brainard [172] proposed a mechanism that they call *client puzzles* to form stronger authentication than that provided by cookies. Their idea is that when load on a server becomes high (possibly as a result of a denial of service attack) the server will send a 'puzzle' of moderate computational difficulty to each client which must be solved before a new connection is made. Genuine clients will be only mildly inconvenienced by this demand but an adversary trying to make multiple connections will have to solve many puzzles.

1.6.7 Typing Attacks

When a protocol is written on the page its elements are clearly distinct. But in practice a principal receiving a message, whether encrypted or not, simply sees a string of bits which have to be interpreted. Typing attacks exploit this by making a recipient misinterpret a message, accepting one protocol element as another one (that is, a message element of a different type). For example, an element which was intended as a principal identifier could be accepted as a key. Such an attack typically works with replay of a previous message.

An example can be seen in the well-known protocol of Otway and Rees [259] shown in Protocol 1.5 (see also Sect. 3.4.2). A and B share long-term keys, K_{AS} and K_{BS} respectively, with the server S. S generates a new session key K_{AB} and passes it to both A and B. M and N_A are nonces chosen by A and N_B is a nonce chosen by B.

Goal: Key transport

1. $A \rightarrow B : M, A, B, \{N_A, M, A, B\}_{K_{AS}}$
2. $B \rightarrow S : M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}}$
3. $S \rightarrow B : M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
4. $B \rightarrow A : M, \{N_A, K_{AB}\}_{K_{AS}}$

Protocol 1.5: Otway–Rees protocol

The typing attack works because of the similarity in the encrypted parts of the first and last messages – they start with the same message field and are encrypted with the same key. As usual for this kind of attack we need to make some extra assumptions if the attack is to succeed. The attack depends on the length of the composite field M, A, B being the same as that expected for the

Petitioner Samsung Ex-1036, 0051

key K_{AB}. This may be a quite reasonable assumption; for example, M may be 64 bits, A and B could be 32 bits, so that K_{AB} would have to be of length 128 bits, which is a popular choice of symmetric key size today. With these assumptions, an adversary C is able to execute Attack 1.2. Here we introduce the notation C_B to indicate that the adversary C is masquerading as principal B.

1. $A \rightarrow C_B : M, A, B, \{N_A, M, A, B\}_{K_{AS}}$
4. $C_B \rightarrow A : \{N_A, M, A, B\}_{K_{AS}}$

Attack 1.2: Typing attack on Otway–Rees protocol

C masquerades as B and intercepts the message from A. C then returns the encrypted part of this message to A, which is interpreted by A as message 4 of the protocol. With the assumptions mentioned above, A will accept the composite field M, A, B as the shared key K_{AB}. Of course C knows the values of M, A and B from message 1, and so is able to continue masquerading as B for the duration of the session.

Typing attacks can be countered by various measures. Ad hoc precautions include changing the order of message elements each time they are used, and ensuring that each encryption key is used only once. More systematic methods are to include an authenticated message number in each message or an authenticated type field with each field. Naturally these come at a cost in computation and bandwidth. Aura [20] has considered systematic methods to avoid typing-based replay attacks.

1.6.8 Cryptanalysis

It was already stated in Sect. 1.2.2 that cryptographic algorithms used in protocols are often treated abstractly and considered immune to cryptanalysis. However, there are some exceptions that should be mentioned. The most important exception is when it is known that a key is weak and is (relatively) easy to guess once sufficient evidence in available. This 'evidence' will normally be a pair of values, one of which is a function of the key; examples are a plaintext value and the corresponding ciphertext, or a plaintext value and its MAC.

The most common example of use of a weak key is when the key is formed from a password that needs to be remembered by a human. In this situation the effective key length can be estimated from the set of values that are practically used as passwords, and is certainly much smaller than would be acceptable as the key length of any modern cryptosystem. A number of protocols have been designed specifically to hide the evidence needed to guess at weak keys. These are examined in some detail in Chap. 7.

Petitioner Samsung Ex-1036, 0052

1.6.9 Certificate Manipulation

In public key protocols the *certificate* of a principal acts as an off-line assurance from a trusted authority that the principal's public key really does belong to that principal. Other principals who make use of a certificate are trusting that the authority has correctly identified the owner of the public key at the time that the certificate was issued. However, it is not necessarily expected that the authority is provided with evidence that the corresponding private key is actually held by the principal claiming ownership of the key pair. This leads to potential attacks in which the adversary gains a certificate that a public key is its own, even though it does not know the corresponding private key. By choosing the public key to be a function of an existing public key some undesirable consequences may arise.

An example of a certificate manipulation attack is given by Menezes *et al.* [233] on a key agreement protocol of Matsumoto *et al.* [224]. (This protocol, and related ones, will be examined in some detail in Chap. 5.) Principals A and B possess public keys g^a and g^b respectively, and corresponding private keys a and b. Here g generates a suitable group in which the discrete logarithm problem is hard. Each public key is certified and so A and B possess certificates $Cert(A)$ and $Cert(B)$ respectively which contain copies of their public keys. A normal protocol run proceeds as shown in Protocol 1.6, where x and y are random values chosen by A and B respectively.

Goal: Key agreement

1. $A \rightarrow B : g^x, Cert(A)$
2. $B \rightarrow A : g^y, Cert(B)$

Protocol 1.6: A protocol vulnerable to certificate manipulation (MTI protocol)

The shared key is $K_{AB} = g^{ay+bx}$, calculated by A as $(g^y)^a(g^b)^x$ and by B as $(g^a)^y(g^x)^b$. The adversary C engineers an attack by choosing a random value c, claiming that g^{ac} is its public key, and obtaining a certificate for this public key. (Notice that C cannot obtain the corresponding private key ac.) C then masquerades as B in Protocol 1.6, and completes two runs of the protocol, one with A and one with B, as shown in Attack 1.3.

---

1. $A \rightarrow C_B : g^x, Cert(A)$
1'. $C \rightarrow B : \ g^x, Cert(C)$
2'. $B \rightarrow C : \ g^y, Cert(B)$
2. $C_B \rightarrow A : g^{yc}, Cert(B)$

Attack 1.3: Certificate manipulation attack on MTI protocol

Petitioner Samsung Ex-1036, 0053

After the attacking run is complete, A will calculate the key $K_{AB} = (g^{yc})^a(g^b)^x = g^{acy+bx}$, and B will calculate the key $K_{CB} = (g^{ac})^y(g^x)^b = g^{acy+bx}$. Thus A and B have found the same key, but A believes this key is known only to A and B while B believes it is known only to C and B. This misunderstanding can lead to problems in the subsequent use of the session key.

Attacks of this sort can be avoided by demanding that every principal demonstrates knowledge of the private key before a certificate is issued for any public key. Such a demonstration is ideally achieved using zero knowledge techniques so that the trusted authority gains nothing useful in finding the private key. A more convenient method may be to have the private key owner sign a specific message or a challenge. More generally this process is part of *public key validation* which provides assurance that the public and corresponding private key have been properly generated as specified in the protocol.

1.6.10 Protocol Interaction

Most long-term keys are intended to be used for a single protocol. However, it could be the case that keys are used in multiple protocols. This could be due to careless design, but may be deliberate in cases where devices with small storage capability are used for multiple applications (smart cards are the obvious example).

It is easy to see that protocols designed independently may interact badly. For example, a protocol that uses decryption to prove possession of an authenticating key may be used by an adversary to decrypt messages from another protocol if the same key is used. Kelsey *et al.* [180] give several examples of how things can go wrong, and discuss the *chosen protocol attack* in which a new protocol is designed by the adversary to attack an existing protocol. Apart from limiting keys to be used in unique protocols, one method to prevent such attacks is to include the protocol details (such as unique identifier and version number) in an authenticated part of the protocol messages.

1.7 Design Principles for Cryptographic Protocols

Abadi and Needham [4] have proposed a set of principles intended to act as 'rules of thumb' for protocol designers. They were derived from observation of the most common errors that have been found in published protocols. By following these principles designers are less likely to make errors, but it must be emphasised that there can be no guarantee that this will result in a good protocol. Furthermore, there are many examples of protocols that ignore one or more of the principles and yet are (believed to be) secure.

The principles are paraphrased in Table 1.4. Many of them can be related to discussions and examples earlier in this chapter. For example, Principle 4 is

Petitioner Samsung Ex-1036, 0054

about ensuring that the correct cryptographic properties have been used, as discussed in Sect. 1.4, while Principle 9 could refer directly to the attack shown in Fig. 1.8. We believe that most of the principles are self-explanatory. Abadi and Needham discuss each principle in detail with examples to illustrate their application.

Table 1.4. Abadi and Needham's principles for design of cryptographic protocols

1. Every message should say what it means: the interpretation of the message should depend only on its content.
2. The conditions for a message to be acted upon should be clearly set out so that someone reviewing a design may see whether they are acceptable or not.
3. If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message.
4. Be clear about why encryption is being done.
5. When a principal signs material that has already been encrypted, it should not be inferred that the principal knows the content of the message.
6. Be clear about what properties you are assuming about nonces.
7. If a predictable quantity is to be effective, it should be protected so that an intruder cannot simulate a challenge and later replay a response.
8. If timestamps are used as freshness guarantees, then the difference between local clocks at various machines must be much less than the allowable age of a message.
9. A key may have been used recently, for example to encrypt a nonce, and yet be old and possibly compromised.
10. It should be possible to deduce which protocol, and which run of that protocol, a message belongs to, and to know its number in the protocol.
11. The trust relations in a protocol should be explicit and there should be good reasons for the necessity of these relations.

Petitioner Samsung Ex-1036, 0055

2

Goals for Authentication and Key Establishment

2.1 Introduction

Any attack on a protocol is only valid if it violates some property that the protocol was intended to achieve. In other words all attacks must be considered relative to the protocol goals. Experience has proven that many protocol problems result when designers are unclear about the protocol goals they are trying to achieve. This in turn leads to disputes about whether protocol attacks are valid, since designers may regard the goals differently from analysers. Gollmann [118] has recognised that it is a difficult matter to decide exactly what is meant by commonly used words such as 'authentication'; even though everyone has a general idea of the meaning of such a word, the actual interpretation may vary with the protocol. It turns out that although most authors can agree on general definitions, their ideas diverge when precision is required.

Clarity in describing protocol goals is desirable for all parties concerned.

Designers should make use of the protocol goals to justify each message field and all cryptographic processing. Experience shows that protocols with well-defined goals are streamlined and transparent to analyse.

Analysers should make use of protocol goals to direct their attempts to find attacks or prove they do not exist.

Implementers must be clear on exactly what a protocol is intended to achieve so that they protect the user from the correct threats. It is not the responsibility of the implementer to work out what a protocol achieves.

Many authors have considered the question of what are the appropriate goals for cryptographic protocols, mainly in the context of protocol analysis. Definitions of various goals also now appear in a number of standards for cryptographic protocols. However, there is a lack of agreement as to what are the desirable goals for authentication and key establishment, as well as what the definitions for these goals should be.

Let us consider an initial example to illustrate the divergent paths that may be taken in assessing protocol goals and attacks. In Protocol 2.1 principals

Petitioner Samsung Ex-1036, 0056

A and B wish to authenticate each other, using an initially shared key K_{AB}. We will discuss below some possible meanings of *authenticate*, but for now we will assume that both users wish to know that their communicating peer is in possession of K_{AB}.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : MAC_{K_{AB}}(B, A, N_A), N_B$
3. $A \rightarrow B : MAC_{K_{AB}}(A, B, N_B)$

Protocol 2.1: Example protocol

Both A and B choose nonces, N_A and N_B respectively, and generate a message authentication code (MAC) using the shared key. Protocols similar to this one have been published in the literature (although we do not believe this exact one has been suggested before). An attack on Protocol 2.1 is possible which is very similar to some previously published attacks [41, 103]. In Attack 2.1, principal A is used as an 'oracle' by the adversary C.

1. $C_A \rightarrow B : N_C$
2. $B \rightarrow C_A : MAC_{K_{AB}}(B, A, N_C), N_B$
1'. $C_B \rightarrow A : N_B$
2'. $A \rightarrow C_B : MAC_{K_{AB}}(A, B, N_B), N_A$
3. $C_A \rightarrow B : MAC_{K_{AB}}(A, B, N_B)$

Attack 2.1: Attack on Protocol 2.1

From the view of B the protocol has completed normally. However, the protocol has not run correctly and it is certainly not the case that A is the communication peer of B. On the other hand, if the protocol goal for B was to establish that A is ready and willing to communicate with him then the protocol has not failed. Indeed we may note that A was sent a challenge by someone purporting to be B and replied with a message to the effect that she was prepared to communicate with B. This example illustrates how careful we must be to evaluate attacks against definitions of protocol goals.

One of the most important properties of any security protocol, and one that we may regard as a goal in itself, is that the protocol should satisfy its security goals. Much research has been devoted in recent years to ways of gaining assurance that the security goals are satisfied and many different approaches have emerged. Most approaches use a formal model of one sort or another. It is beyond the scope of this book to cover each of these techniques in detail – as a contrast the recent book of Ryan and Schneider [281] is devoted to a single formal approach. However, we believe it is worthwhile to give a

Petitioner Samsung Ex-1036, 0057

flavour of some of the main techniques. We aim to make a practitioner aware of what it means to conduct an analysis or obtain a proof with these approaches.

In the next few sections we will consider various possible definitions for the fundamental goals of authentication and key establishment protocols and explain the reasons for our choice of definitions that we will use in subsequent chapters. This will lead to enhanced goals which may be desirable in key establishment. We then look at special goals that cover the situation when long-term keys are compromised, and consider some alternative authentication properties which may be useful in some applications. After this, the remainder of the chapter is devoted to an overview of the main formal approaches to protocol analysis. We look first at some of the techniques using formal methods for model checking or theorem proving. Finally we briefly look at the complexity theory approach to providing protocol proofs. This chapter concentrates on goals for protocols with two principals; goals for multi-party protocols are discussed further in Chap. 6.

2.2 Basic Goals

In this section we will examine the fundamental goals that are typically expected of authentication and key establishment protocols. After looking at the different kinds of adversary, we consider to what extent key establishment and entity authentication are connected. This leads us to consider separately first *user-oriented* goals and then *key-oriented* goals.

2.2.1 Models of Security

As well as deciding what are the required goals of a protocol, we also need to agree on what is the attack model that will be used to decide whether a goal is achieved. In Chap. 1 we have already seen the kinds of actions that we expect the adversary to be capable of. In particular the adversary:

- controls the communications between all principals, which means that the adversary can observe all messages sent, alter messages, insert new messages, delay messages or delete messages;
- can obtain any session keys used in previous protocol runs.

Some models (a particular example is that used by Burrows *et al.* for their logic [76]) have assumed that legitimate principals will act honestly. This means that an entity that has been authenticated will follow the protocol specification, which corresponds to a situation in which a closed group of users trust each other to act correctly. This may be regarded as a model of a weak adversary that cannot corrupt insiders. A more powerful attacking model is that in which (authenticated) principals may act maliciously. This model has been used by other researchers such as Bellare and Rogaway [30]

Petitioner Samsung Ex-1036, 0058

who assume that the adversary is able to corrupt any principal at any time. It is pointed out by Gollmann [119] that this corresponds to the most realistic situation in commercial applications where the majority of real-world attacks are said to come from insiders. In order to allow as rich an exploration of protocol properties as possible we will not assume either of these situations in general, but try to point out where the difference in these models is important for different types of protocol.

Definition 2.1. *A model for the the adversary allows:*

- malicious insiders *if the adversary may be a legitimate protocol principal;*
- honest insiders *if principals will only act according to the protocol.*

2.2.2 Key Establishment or Authentication?

In the early literature on cryptographic protocols it was common to refer to all protocols concerned with setting up session keys as 'authentication protocols'. This is not entirely satisfactory because some protocols that set up session keys provide no authentication of one party to the other, while other protocols designed to provide entity authentication involve no session key. Therefore it has become usual to distinguish between protocols that provide only authentication, and call these *entity authentication protocols*, while using the term *key establishment protocol* for one that involves setting up a new key, typically for a communications session.

Gollmann [118] has put forward a number of different options for what could be meant by authentication. The first one is as follows.

Gol1 The protocol shall establish a fresh session key, known only to the participants in the session and possibly some trusted third parties.

This goal may be achieved even though each party knows nothing about even the existence of the other party, let alone whether the other party is willing to engage in a session. Thus this is a goal about key establishment rather than entity authentication.

The second goal suggested by Gollmann is as follows, in which A and B are the protocol principals.

Gol2 A cryptographic key associated with B was used in a message received by A during the protocol run. The protocol run is defined by A's challenge or a current timestamp.

This is a goal concerning entity authentication. It says nothing about a new session key and can be satisfied by a protocol that is not concerned with key establishment.

There appears to be more dissent in the literature regarding the nature of entity authentication than there is with regard to key establishment. One reason for this may be that it is difficult to be clear on the purpose of entity authentication in the absence of key establishment. Diffie *et al.* [103] say

Petitioner Samsung Ex-1036, 0059

that it is 'accepted that these topics should be considered jointly rather than separately', while Bellare and Rogaway [33] go further in stating:

> ... entity authentication is rarely useful in the absence of an associated key distribution, while key distribution, all by itself, is not only useful, but it is not appreciably more so when an entity authentication occurs along side. Most of the time entity authentication is *irrelevant*: it doesn't matter if you have been speaking to a given communication partner, in that by the time you become aware of [an authenticated entity] there will be no particular reason to believe that the partner is still "out there" anyway.

In our view there are situations when entity authentication by itself may be useful, such as when using a physically secured communication channel. But it is important to appreciate exactly what it provides.

Syverson and van Oorschot [315] identify what they term six 'Generic Formal Goals'. These are expressed in English in Table 2.1; for formal statements readers should refer to their paper. There are clearly dependencies between various of these goals. For example, **SVO2** is a stronger property than **SVO1**. Furthermore, it is not clear why these particular goals are important; for example, it might be questioned whether Secure Key Establishment is useful without Key Freshness. To be fair to the authors they state that it is *not* intended as a 'definitive list of *the* goals that a key agreement or key distribution protocol should meet'.

Table 2.1. Syverson and van Oorschot's generic formal goals for protocols

SVO1	**Far-end Operative** A believes B recently 'said' something.
SVO2	**Entity Authentication** A believes B recently replied to a specific challenge.
SVO3	**Secure Key Establishment** A has a certain key K that A believes is good for communication with B.
SVO4	**Key Confirmation** In addition to SVO3, A has received evidence confirming that B knows K.
SVO5	**Key Freshness** A believes a certain key K is fresh.
SVO6	**Mutual Understanding of Shared Key** A believes that B has recently confirmed that B has a certain key K that B believes is good for communication with A.

So far we hope to have convinced the reader that there is no unanimity either on what the goals of authentication and key establishment protocols should be or on how to define those goals. We will now focus on three different classes of goals which can be thought of as those concerning entity authentication (which we term *user-oriented goals*), those concerning key establishment

Petitioner Samsung Ex-1036, 0060

(which we term *key-oriented goals*) and those which are optional additions to key establishment (which we term *enhanced goals*). We will also show how all these goals may be fitted into a hierarchy so that it is often possible to say that satisfaction of one goal automatically implies another. This is achieved by considering goals in terms of the functional elements of the protocols and what assurances can be gained from them.

2.2.3 User-Oriented Goals

The ISO Security Architecture [157] defines entity authentication as 'the corroboration that an entity is the one claimed'. This is not as precise a definition as one might like since it does not explain which entity is the subject. Menezes *et al.* [234] give a more comprehensive definition as follows.

Definition 2.2. Entity authentication *is the process whereby one party is assured (through acquisition of corroborative evidence) of the identity of a second party involved in a protocol, and that the second has actually participated (i.e., is active at, or immediately prior to, the time the evidence is acquired).*

Protocol 2.2 is an example that seems to provide entity authentication of B to A satisfying this definition. A sends her nonce to B who replies by signing it. It seems clear that A knows that B must have engaged in this protocol and that the signature is fresh.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : Sig_B(N_A)$

Protocol 2.2: A simple authentication protocol

Definition 2.2 is a clear explanation, but does not go as far as is possible or perhaps even desirable. Imagine user A having received some messages in an entity authentication protocol. What is it that she can hope to have learned from those messages? One aspect is that user B is really out there now, somewhere on the network. This is the far-end operative property (**SVO1**) that we have already seen. The only other assurance that seems relevant is to know that B is ready to engage in communication with A.

Attack 2.2 on Protocol 2.2 shows why this extra assurance may be desirable. In this protocol run A can verify that the signature received was formed by B, yet B has not indicated that he is aware of A. In some sense we may even accept that the adversary C has provided assurance that he is B. On the other hand C does not appear to be doing anything other than faithfully replaying messages between A and B. The next definition tries to capture the notion of one principal being prepared to communicate with another principal.

Petitioner Samsung Ex-1036, 0061

1. $A \rightarrow C_B : N_A$
1'. $C \rightarrow B : \ N_A$
2'. $B \rightarrow C : \ Sig_B(N_A)$
2. $C_B \rightarrow A : Sig_B(N_A)$

Attack 2.2: An attack on Protocol 2.2

Definition 2.3. *A principal A is said to have* knowledge of B as her peer entity *if A is aware of B as her claimed peer entity in the protocol.*

Considering again the fundamental elements used in authentication protocols this seems to be all that can be achieved. Messages can convey either freshness, or principals with which communication is desired. Combining these leads to a strong definition of entity authentication. (There are several alternative ways of expressing this property which all indicate that A is authenticated to B only if A is prepared to engage in communications with B.)

Definition 2.4. Strong entity authentication *of A to B is provided if B has a fresh assurance that A has knowledge of B as her peer entity.*

An enhanced version of Protocol 2.2 can provide this stronger assurance. Protocol 2.3 provides strong entity authentication of B to A. It may be checked that an adversary C cannot use Attack 2.2 to convince A that B is aware of A as his peer entity.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : Sig_B(A, N_A)$

Protocol 2.3: Another simple authentication protocol

According to Defn. 2.4, there are two subgoals of entity authentication:

- A (once) has had knowledge of B as her peer entity;
- A is operative.

The latter of these is the far-end operative property discussed before in goal **SVO1** of Table 2.1. Notice that it is straightforward to extend Defn. 2.4 to a multi-party goal of entity authentication of a group of users $\mathcal{U}$ to A: the principal A is freshly aware of the principals in $\mathcal{U}$ as her peer entities.

Entity authentication is a service that is provided by one entity to one or more other parties. Most often we are concerned with the interaction between two entities and then it is common to differentiate between two situations.

Definition 2.5. Mutual authentication *occurs if both entities are authenticated to each other in the same protocol.* Unilateral authentication *occurs if only one entity is authenticated to the other.*

Petitioner Samsung Ex-1036, 0062

When multiple entities are involved there are many possibilities for different combinations of entity authentication; in principle a protocol can authenticate any subset of the entities to any other subset. In practice there seem to be few situations where a complex rule for who should be authenticated to whom is useful.

2.2.4 Key-Oriented Goals

Menezes *et al.* [234] give the following definition for key establishment.

> *Key establishment* is a process or protocol whereby a shared secret becomes available to two or more parties, for subsequent cryptographic use.

This definition can be extended and made more specific. One way to understand the possible goals for key establishment is to consider what may be achieved with typical message components. There are three types of message components that are conventionally used in cryptographic protocols for key establishment and entity authentication. These are:

1. **keys** which may be long-term keys or session keys;
2. **identifiers** for protocol principals;
3. **nonces** which may be random values, timestamps or counters.

These components are combined and processed with cryptographic mechanisms to provide confidentiality and/or authentication. For key establishment a new session key may be associated with a nonce, or with identifiers of protocol principals. In practice a session key is not of any use unless it is known to be fresh and it is known which other entities may possess it. Most authors agree that secure key establishment should require the two goals that the key is known to be fresh and is known only to the other protocol participant(s), possibly including trusted third parties. This is often referred to as establishing a *good key.*

Definition 2.6. *The shared session key is a* good key *for A to use with B only if A has assurance that:*

- *the key is fresh (***key freshness***);*
- *the key is known only to A and B and any mutually trusted parties (***key authentication***).*

The second of these properties is often also called *implicit key authentication.* (As pointed out by Gollmann [121], this property may equally be regarded as being about confidentiality of the key.) It can be argued that key authentication must imply key freshness, since a key that is not fresh cannot be guaranteed to be kept confidential. From this viewpoint a separate requirement for key freshness is not required.

Although not a common requirement, public session keys are certainly possible. The above definition is easily extended to this case.

Petitioner Samsung Ex-1036, 0063

Definition 2.7. *The public session key is a* good key *for A to use with B only if:*

- *the key is fresh (***key freshness***);*
- *the corresponding private key is known only to B (***key authentication***).*

An interesting additional key-oriented goal has been considered by some authors, including Janson and Tsudik [169].

Definition 2.8. Key integrity *is the property that the key has not been modified by the adversary, or equivalently only has inputs from legitimate principals.*

- *For a key transport protocol, key integrity means that if the key is accepted by any principal it must be the same key as chosen by the key originator.*
- *For a key agreement protocol, key integrity means that if a key is accepted by any principal it must be a known function of only the inputs of the protocol principals.*

Note that there is no contradiction if a key establishment protocol provides the good key property and yet fails to provide key integrity. It is quite conceivable that an adversary may be able to disturb a protocol, whether it is a key transport or a key agreement protocol, in such a way that the key has been changed from its 'correct' value and yet the key is still fresh and unknown to the adversary.

2.3 Enhanced Goals

In this section we first examine how to combine and extend the basic goals introduced in Sect. 2.2 and then give a detailed example to clarify the definitions. Following this we explain how to classify protocol goals into either intensional or extensional types. We also look briefly at protocol efficiency and the notions of responsibility and credit.

2.3.1 A Hierarchy of Protocol Goals

It is possible to consider any combination of the goals or subgoals from key establishment and from entity authentication. There are also goals that go beyond both good key and entity authentication which are termed here *enhanced goals*. Which are the useful goals to aim for? To answer this question we must examine what is the purpose to go beyond key establishment. A protocol that provides only key establishment gives no assurance that the partner with whom communication is desired even exists. Thus key establishment only provides the *ability* to engage in secure communication. Enhanced goals seek to establish the *readiness* of the partner to engage in secure communication. Since entity authentication, as in Defn. 2.4, deals with exactly this concern it

Petitioner Samsung Ex-1036, 0064

is natural that enhanced goals should include entity authentication together with key establishment.

Definition 2.9. Key confirmation *of A to B is provided if B has assurance that key K is a good key to communicate with A, and that principal A has possession of K.*

Key confirmation provides evidence that the partner has the same key but leaves open the possibility that the key is intended by the partner for a different communication session (with the assumption that the partner may be engaged in several conversations). Key confirmation provides evidence that the partner wishes to communicate with some entity, so implies far-end operative, but may not imply entity authentication. Key confirmation is typically achieved by having both parties send each other some fresh data using a cryptographic function depending on the key; this is often referred to as a *handshake.*

Shoup [299] has put forward the idea that key confirmation is not a valuable security property. His point is that it is not really useful for a principal to know that the partner has, or can obtain, possession of the session key, but rather that the partner has *accepted* the session key. It is never possible to guarantee this for both parties, since one party must always finish, and therefore accept, without the other party knowing. Nevertheless, the property as stated may or may not be achieved, even mutually. As with entity authentication, we prefer not to judge whether this property is a useful one.

It can be seen that Defn. 2.9 requires that the identified other party has received the session key. Not all authors use this definition of key confirmation. For example, the following definition is given in the *Handbook of Applied Cryptography* [234].

> *Key confirmation* is the property whereby one party is assured that a second (possibly unidentified) party actually has possession of a particular secret key.

This contrasts with the definition in the ISO/IEC 11770-2 key management standard [158].

> *Key confirmation:* the assurance for one entity that another identified entity is in possession of the correct key.

The following definition is also taken from the *Handbook of Applied Cryptography.*

Definition 2.10. Explicit key authentication *is the property obtained when both (implicit) key authentication and key confirmation hold.*

Notice that it does not matter for this definition whether or not key confirmation includes identification of the other party in possession of the key.

Petitioner Samsung Ex-1036, 0065

This is because implicit key authentication assures B that only A may have the key, so any party that shows possession of the key must be entity A.

Mutual belief in the key, following **SVO6** in Table 2.1, adds to key confirmation that the key is known by the partner to be good. (Actually, **SVO6** does not require the good key property, but seems of little value if it does not also hold.) It provides both key confirmation and entity authentication since if the partner has acknowledged that the key is good for the communication this can be taken as a confirmation that the partner is willing to communicate.

Definition 2.11. Mutual belief in the key K *is provided for* B *only if* K *is a good key for use with* A*, and* A *wishes to communicate with* B *using key* K *which* A *believes is good for that purpose.*

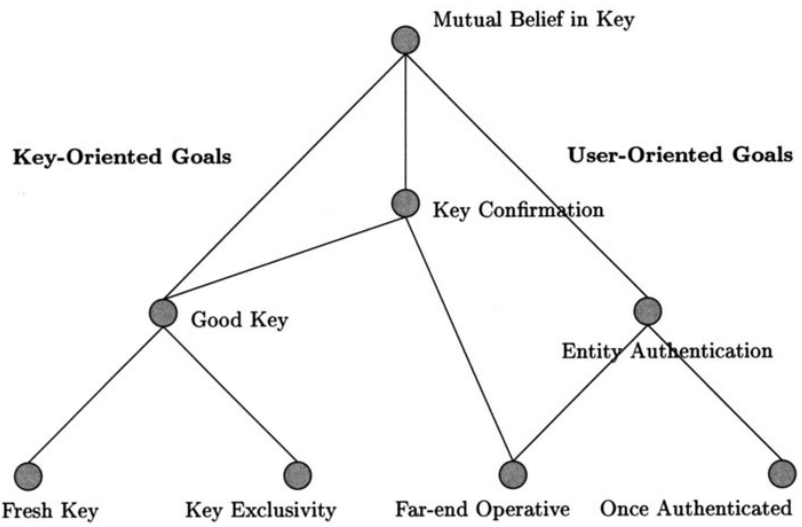


Fig. 2.1. Hierarchy of authentication and key establishment goals

The hierarchy of goals is shown in Fig. 2.1 as a lattice. Entity authentication and its two subgoals are classed as user-oriented goals, while good key and its subgoals are key-oriented goals. Mutual belief and key confirmation are classed as extended goals which concern both keys and users. Of course this hierarchy does not show all possible goals. The ones shown appear to be some of the most important ones considered in the literature. As an example of a goal that is not included, there is an enhanced goal that lies between key confirmation and mutual belief, which provides key confirmation and entity authentication but does not provide assurance that the key is known by the partner to be good.

Petitioner Samsung Ex-1036, 0066

2.3.2 Example: STS Protocol

We turn to an example to focus discussion on the subtleties of assessing protocol attacks against published goals. The station-to-station (STS) protocol [103] uses a digital signature in the exchanged messages to add authentication to the well-known Diffie–Hellman protocol [102]. This uses arithmetic in a multiplicative group with generator g. Exponents x and y are chosen randomly by A and B respectively and are used to form the session key $K_{AB} = g^{xy}$. Protocol 2.4 shows the messages in a successful protocol run.

1. $A \rightarrow B : g^x$
2. $B \rightarrow A : g^y, \{Sig_B(g^y, g^x)\}_{K_{AB}}$
3. $A \rightarrow B : \{Sig_A(g^x, g^y)\}_{K_{AB}}$

Protocol 2.4: STS protocol

Here $Sig_X(.)$ represents the signature by the principal X on the string in the brackets, while $\{M\}_K$ denotes symmetric encryption of message M using key K. The particular signature algorithm chosen does not matter for the protocol. Consider how the good key goal is achieved for A.

1. The signature in message 2 can only be formed by B.
2. It is not a replay from an old protocol run since A knows that g^x was fresh.
3. The signature alone does not imply that B knows K_{AB}. Therefore the encryption with K_{AB} is necessary to provide assurance that B really knows K_{AB}.

Thus it appears that A gains key confirmation, as well as a good key with B, from message 2. With regard to user-oriented goals, it seems clear that both users achieve liveness of the other, since each receives a signed message containing a value they know to be fresh. Strong entity authentication, in the sense of Defn. 2.4, is more problematic since there is no explicit inclusion of identifiers in the signed messages which could be used to deduce the identity of the desired communications partner.

Lowe [212] has proposed an attack on the STS protocol. To be quite precise, the protocol analysed by Lowe is slightly different in that principal identifiers are added to each message to give the modified version shown in Protocol 2.5. The addition of the identifiers appears to have no material difference on the protocol since they are attached as plaintext and so are vulnerable to both eavesdropping and modification. However, their addition is critical to the interpretation of the attack. Lowe [212] states that the identifiers were 'included to make the subsequent explanations clearer'.

Lowe's attack does not affect the key establishment properties but is addressed at whether entity authentication is achieved. Suppose that C is an

Petitioner Samsung Ex-1036, 0067

1. $A \rightarrow B : A, B, g^x$
2. $B \rightarrow A : B, A, g^y, \{Sig_B(g^y, g^x)\}_{K_{AB}}$
3. $A \rightarrow B : A, B, \{Sig_A(g^x, g^y)\}_{K_{AB}}$

Protocol 2.5: STS protocol modified to include identifiers

adversary who wishes to attack the protocol. C intercepts a protocol run started by A and masquerades as B. In parallel, C starts a protocol run with B. Attack 2.3 shows an attacking run, where C_B denotes C masquerading as principal B.

1. $A \rightarrow C_B : A, B, g^x$
1'. $C \rightarrow B : \ C, B, g^x$
2'. $B \rightarrow C : \ B, C, g^y, \{Sig_B(g^y, g^x)\}_{K_{AB}}$
2. $C_B \rightarrow A : B, A, g^y, \{Sig_B(g^y, g^x)\}_{K_{AB}}$
3. $A \rightarrow C_B : A, B, \{Sig_A(g^x, g^y)\}_{K_{AB}}$

Attack 2.3: Lowe's attack on Protocol 2.5

The attack is very simple; C is doing little more than relaying each message that passes between A and B. What is the result? In the attacking run B has no indication that A has engaged in the protocol and yet A has completed a successful run and accepted that her partner is B. Is this a successful attack on the STS protocol? If we apply the same attack to the original STS protocol (2.4) without identifiers, we see that C does nothing more than relay messages between A and B, so how can this constitute an attack?

Diffie *et al.* [103] define security based on matching conversations. Consequently, when applying the attack to their original protocol the conversation of A does indeed match that of B and so the attack does not violate their definition of security. A reasonable conclusion then may be that the attack is invalid on the STS protocol as specified by the authors and in accordance with their definition of security. But what about the modified protocol? Is it really different from the original and is the attack valid in that case? The answer must depend on what are the intended goals.

- After the attacking run it is clear that the good key goal has not been broken.
- Key confirmation has indeed been achieved: A can be sure that B knows the shared key.
- A does not know that B knows the key is good for use with A. In other words the *mutual belief in key* goal is not achieved for A.

Petitioner Samsung Ex-1036, 0068

- The attack shows that A would be wrong to conclude, after a successful run, that B wishes to communicate with her. Thus strong entity authentication (using Defn. 2.4) is not achieved.

We conclude that the attack would be valid if either mutual belief in the key or strong entity authentication were protocol goals. However, it is clear from the paper of Diffie *et al.* that they did not regard these as goals of their protocol. The insight gained from the attack is therefore that the protocol does not meet extended goals that could be desired by some users.

Lowe proposes [212] that the identity of the other party be included in the signatures in order to prevent the attack. This also allows an informal argument that strong entity authentication is achieved, if the included identifier is interpreted as the name of the entity with which communication is desired.

2.3.3 Intensional and Extensional Goals

Roscoe [277] considered the difference between *intensional* and *extensional* specifications of cryptographic protocols. Although no formal definition of these terms is given, the general principle is whether specifications take into account the details of how the protocol operates rather than what each protocol principal gains from the protocol. An *extensional* property is defined by Roscoe as one that is:

> ... independent of the details of the protocol and would apply to any other protocol designed to achieve the same effect.

Thus extensional properties cannot refer to any specific protocol messages. An example of an extensional property would be that user A wishes to communicate with a user B using a shared key.

In contrast, Roscoe defines an *intensional* property as one:

> ... whose primary purpose is to assert a property of the way, in terms of communications within the protocol, a particular state is reached.

Thus an example of an intensional property would be that user A has responded to a specific message from user B using a shared key. This intensional property could be used to provide the extensional property mentioned above. However, there are other methods that can be used to achieve the extensional property; for example, a trusted server might inform B that user A has the key and wishes to use it to communicate with B.

Roscoe showed with examples that analysis of a protocol from an intensional specification is often more useful in finding protocol attacks than one from an extensional specification. He defines the *canonical intensional specification* as:

> ... an entity will believe a protocol run has completed only if a correct series of messages has occurred up to and including the last message that entity communicates.

Petitioner Samsung Ex-1036, 0069

The canonical specification is a very strong requirement which fails for many protocols that are believed secure by different measures. It essentially says that if the protocol completes in the view of its participants then the protocol must have run as specified. Notice that this specification says nothing about what the protocol achieves, and in particular is satisfied by the (admittedly very secure) null protocol.

While intensional properties may be important for analysis, it seems that it is extensional properties that are appropriate to be considered by the protocol designer. In other words the protocol designer should be ensuring that the extensional protocol goals are achieved, by whatever means. This is because the protocol designer should be concerned with what the protocol achieves for the principals. It is our view that an attack on a protocol must be measured against whether it defeats the extensional goals of the protocol. The definitions of goals that we have presented in Sects. 2.2.3, 2.2.4 and 2.3.1 are all extensional.

Various researchers have included the notion of 'matching conversations' in their definitions of security. This means that the protocol only succeeds if both principals agree on the messages that were sent and received. We may regard such issues as being concerned with intensional goals. Our example in Sect. 2.3.2 shows that an attack may violate the matching conversations goal and yet fail to violate important extensional goals. Thus an attack that violates the intensional goals of the protocol does not give an indication of which extensional goals fail.

An interesting context in which to consider the goal of matching conversations is mutual authentication. Is there a sense in which a mutual authentication protocol can achieve more than two symmetrical runs of a unilateral authentication protocol? If the mutual authentication protocol can be proven to complete only with matching conversations then both parties have assurance that the other has been engaged in the protocol. On the one hand this seems like a natural property to expect of an authentication protocol, while on the other hand it is difficult to see what extensional goal this benefits. Gollmann [120] has discussed this problem at some length. He points out that matters look different depending on whether the protocol is implemented in a circuit-switched or packet-switched network (or more to the point whether or not the communications are connection oriented). The point is that if there are no connections involved then it makes no sense to talk about which partner an entity is 'talking to'. In our view it is not very satisfying to make an extensional definition of entity authentication depend on whether or not communications are connection oriented.

2.3.4 Protocol Efficiency

It is usually important to make protocols as efficient as possible. There are two concerns with regard to efficiency: *computational efficiency* and *communications efficiency.*

Petitioner Samsung Ex-1036, 0070

Computational efficiency is concerned with the computations that the protocol principals need to engage in to complete the protocol. The amount of computation required in cryptographic protocols will depend on the algorithms used to provide the cryptographic services, such as encryption and decryption functions, generation and verification of digital signatures and message authentication codes, and calculation of hash functions. These can vary considerably between specific algorithms and, in particular, computations required for public key algorithms are much greater than those for symmetric algorithms. Furthermore, there has been, and continues to be, considerable research into improvements of implementations of different fundamental cryptographic operations, such as multiplication in finite fields. Advances in implementations can make a significant difference to the relative merits of different cryptographic algorithms.

Communications efficiency is concerned with the number and length of messages that need to be sent and received during the protocol. As well as minimising the number of individual messages, it can be important to have as few *rounds* as possible in the protocol. Gong [125] gives the following definition.

Definition 2.12. *One protocol* round *includes all messages that can be sent in parallel from any point in the protocol.*

Protocols where the messages are independent of each other require fewer rounds than those where messages include fields received in previous protocol messages. Protocol messages can often be rearranged to optimise either the number of individual message exchanges or the number of protocol rounds.

2.3.5 Responsibility and Credit

An interesting dichotomy in the way that session keys can be used has been considered by Abadi [2]. He used the terms *responsibility* and *credit* to describe these two different uses.

- A key can provide *responsibility* if the use of that key implies that the message sender can later be held accountable for the message contents.
- A key can provide *credit* if the use of that key implies that the message sender can later claim ownership of the message contents.

Although the notion of responsibility seems to be related to *non-repudiation*, it does not require that a third party is able to judge whether the holder of the key is authentic, but only that the recipient has assurance that the holder of the key is authorised. Abadi gives several examples to clarify this difference. The first (Protocol 2.6) entails establishment of a new public session key, K, sent by A to B signed with A's private key, with freshness ensured by a timestamp T_A.

Petitioner Samsung Ex-1036, 0071

1. $A \rightarrow B : A, B, Sig_A(K, A, B, T_A)$

Protocol 2.6: A protocol in which A takes 'responsibility'

The key K will subsequently be used by B to verify signed messages during the session. However, as Abadi points out, there is no assurance to B that A in fact knows the private key K^{-1}. Therefore it would be wrong for B to assign credit to A for messages signed with K^{-1}. However, since A has committed to the key K as her short-term public key by signing it, it seems reasonable to accept that A should take responsibility for messages signed with K^{-1}. When we compare this example with Defn. 2.7 we see that K should only be regarded by B as a good public key if B gains assurance that K^{-1} is known only to A. If B trusts A for generating new public keys for herself then it can be a good key, otherwise it is not.

In another example (Protocol 2.7) K_{AB} is a shared session key sent by A to B encrypted with B's public key. The key K_{AB} will subsequently be used to encrypt messages sent between A and B during the session.

1. $A \rightarrow B : E_B(A, K_{AB})$

Protocol 2.7: A protocol in which A takes 'credit'

In this case B has no assurance of the origin of K_{AB} and so cannot be sure who it is shared with. This means that B should not send secrets using K_{AB} and so it seems inappropriate for B to use the key to claim credit. However, the inclusion of A's identity in the encrypted message can be interpreted so as to give A credit for information later sent protected with the session key K_{AB}. A possible scenario is where the protocol is used to deliver the answer in a competition of some sort. From A's perspective only B can obtain K_{AB} and so also B may be assigned responsibility by A for messages authenticated with K_{AB}. As already noted, since B has no assurance as to the other entities who possess K_{AB}, this protocol does not provide a good key according to Defn. 2.6.

2.4 Goals Concerning Compromised Keys

The goals considered in this section are different from the goals discussed previously in this chapter because they only apply in situations that we normally assume cannot happen: when a long-term key is compromised. A user's long-term key is either a private key corresponding to a certified public key, or a key shared with a trusted server. If such a key is known to an adversary, then it is

Petitioner Samsung Ex-1036, 0072

clear that the adversary can masquerade as that user at any time. So what is there left to salvage in such a situation? Two useful properties are discussed in this section. *Forward secrecy* is concerned with protecting information that was not compromised before the long-term key was lost. Preventing *key compromise impersonation* reduces the sorts of attack that can be mounted before the compromise of the long-term key becomes known. Both of these goals are most often discussed with regard to key agreement protocols, and indeed we shall discuss them in this context in Chap. 5. However, they may both apply generally to many different sorts of protocols.

2.4.1 Forward Secrecy

The idea of forward secrecy is that when a long-term key is compromised, session keys that were previously established using that long-term key should not be compromised too. Key agreement protocols in which the long-term key is used only to authenticate the exchange provide typical examples of protocols with forward secrecy. Key transport protocols in which the long-term key is used to encrypt the session key cannot provide forward secrecy.

Definition 2.13. *A key establishment protocol provides* forward secrecy *if compromise of the long-term keys of a set of principals does not compromise the session keys established in previous protocol runs involving those principals.*

Definition 2.14. *A protocol provides* partial forward secrecy *if compromise of the long-term keys of one or more specific principals does not compromise the session keys established in previous protocol runs involving those principals.*

If a protocol does not provide (full) forward secrecy then partial forward secrecy may still be useful if there is an asymmetry in the roles of the principals involved. For example, in a client–server protocol it may be deemed more likely that a client long-term key will be compromised than that the server key will be. In this situation partial forward secrecy in which compromise of client long-term keys does not compromise old session keys is a useful property.

The term forward secrecy seems to have been coined by Günther [138]. In fact he used the term *perfect forward secrecy* but, in common with other authors, we have dropped the word 'perfect'; this is not only for the sake of brevity, but also because it gives connotations with the term 'perfect secrecy' which refers to unconditional (information-theoretic) security which is not relevant here.

The critical concept in providing forward secrecy seems to be the *ephemeral public key*; this is a public key that is used only for the duration of the key establishment protocol and is then destroyed along with the corresponding private key. If the long-term public key is used only for authenticating the session key then the session key cannot be recovered without the ephemeral private key. The most commonly used ephemeral keys are of the type needed

Petitioner Samsung Ex-1036, 0073

in the Diffie–Hellman protocol which is examined in detail in Chap. 5. It should be appreciated that this is not the only type of ephemeral public key; ephemeral keys can be used for any public key cryptosystem. Furthermore, it is a common misconception that forward secrecy can be achieved *only* with key agreement.

As an example consider Protocol 2.8 which provides key transport between A and B. Here K_T is an ephemeral public key chosen by A uniquely for this session. This key is sent to B and signed by A together with a nonce N_A chosen by A. B then uses this ephemeral key to transport the session key K_{AB} confidentially back to A. Here $E_T(.)$ denotes encryption with K_T and h is a one-way hash function.

1. $A \rightarrow B : K_T, N_A, Sig_A(K_T, B)$
2. $B \rightarrow A : E_T(K_{AB}), Sig_B(h(K_{AB}), A, N_A)$

Protocol 2.8: Key transport protocol providing forward secrecy

The private key corresponding to the ephemeral public key should be destroyed by A immediately after the session key is recovered. It can be seen that compromise of the long-term signature keys will not help an adversary in obtaining the session key.

The long-term keys used in a protocol providing forward secrecy may be either shared or public. Consider Protocol 2.9, in which A and B share long-term keys K_{AS} and K_{BS} with server S. Random values N_A, N_B and K_S are chosen by A, B and S respectively. The protocol includes Diffie–Hellman-like key agreement, using the generator g of some multiplicative group, together with encryption using the long-term keys.

1. $A \rightarrow S : A, B$
2. $A \rightarrow B : A, g^{N_A}$
3. $S \rightarrow B : \{A, B, K_S\}_{K_{BS}}$
4. $S \rightarrow A : \{A, B, K_S\}_{K_{AS}}$
5. $B \rightarrow A : B, g^{N_B}$

Protocol 2.9: Server-based protocol providing forward secrecy

The session key K_{AB} is calculated by A as $K_{AB} = (g^{N_B})^{N_A K_S}$ and by B as $K_{AB} = (g^{N_A})^{N_B K_S}$. Once the ephemeral values N_A and N_B are destroyed the session key is protected against compromise of the long-term keys shared with S.

Petitioner Samsung Ex-1036, 0074

2.4.2 Key Compromise Impersonation

When an adversary learns the long-term key of Alice the adversary can impersonate Alice until the compromise is detected and the long-term key is revoked[1]. Key compromise impersonation refers to an attack in which the adversary uses Alice's compromised long-term key to masquerade *to Alice* as another user.

Definition 2.15. *A protocol provides* resistance to key compromise impersonation *if compromise of a long-term key of a principal A does not allow the adversary to masquerade to A as a different principal.*

The typical situation in which key compromise impersonation is possible is when the protocol gives assurance only that each entity has any one of a pair of long-term keys. This situation is commonly found in key agreement protocols where the security is often based on the property that a particular value can be calculated by either of the two principals. It is also common in server-based protocols. For example, the protocol in Fig. 1.9 in Chap. 1 is vulnerable to key compromise impersonation. To see why, suppose that the long-term key K_{AS} of user A becomes known to the adversary C. Then C can generate the message that A is expecting from the server and insert the name of any principal that C wishes to impersonate.

Protection against key compromise impersonation seems to require use of asymmetric cryptography. If each party can verify that the correct private key was used then the other party must be present. For example, if each party receives a digital signature from the other, the adversary cannot forge the signature from B if it only has A's private key. Examples of protocols providing key compromise impersonation are presented in Chap. 5.

2.5 Formal Verification of Protocols

Verifying a protocol means proving that the protocol is correct. The sheer complexity of behaviour that security protocols can exhibit makes their verification no small task. It has long been recognised that informal arguments about protocol correctness are not reliable. Many researchers have developed formal methods to increase assurance in the correctness of security protocols; this topic remains highly active in the research community. The purpose of this section is to give an overview of some of the methods that have received most attention to date.

Formal methods for analysing security protocols can be divided into two major categories.

[1] However, there are measures that can be taken to protect compromised signature keys against abuse, as discussed by Just and van Oorschot [174].

Petitioner Samsung Ex-1036, 0075

Model checking methods consider a relatively large, but finite, number of possible protocol behaviours, and allow checking that they satisfy a set of correctness conditions. These methods are generally more suitable for finding attacks on protocols, rather than proving their correctness.

Theorem proving methods consider *all* possible protocol behaviours, and allow checking that they satisfy a set of correctness conditions. These methods are generally more suitable for proving protocols correct, rather than finding attacks on them.

Both model checking and theorem proving methods require computer assistance to aid with the analysis. However, methods based on theorem proving are less automated than those based on model checking.

Model checking methods for verifying security protocols include the use of general-purpose model checkers, as was done by Lowe [211] and by Mitchell *et al.* [241], as well as the development of special-purpose model checkers for security protocols, as was done by Clarke *et al.* [96]. An appealing feature of model checking methods is that they can provide a counterexample (an attack) when a protocol is found not to satisfy a correctness condition. However, model checkers can be uninformative when a protocol is in fact correct. They do not yield a symbolic proof that can provide insight into the reasons why a protocol is correct. The failure to find an attack simply indicates that the protocol is correct. A more important limitation of model checking methods is that they generally guarantee only correctness of a scaled down version of the protocol. An appealing feature of theorem proving methods is that they can provide a symbolic proof when a protocol is found to be correct. Their main limitation is that they generally require more expert human guidance than methods based on model checking.

2.5.1 FDR

Lowe [211] has developed a method for verifying security protocols using FDR, a model checker for the process algebra CSP [278]. This method has been used to find a previously unknown attack on the Needham–Schroeder public key protocol (see Sect. 4.3.3). A comprehensive introduction to the method, including background on CSP, is contained in the book of Ryan and Schneider [281].

Each principal taking part in the protocol is modelled as a CSP process representing the protocol steps performed by the principal. In CSP, communication is modelled by the notion of *channels*. In Lowe's formulation of the Needham–Schroeder public key protocol the following channels are defined:

- *comm*, which carries messages sent by honest principals;
- *fake*, which carries messages introduced by the intruder;
- *intercept*, which carries messages that are killed by the intruder.

In CSP, a communication is an event of the form $c.v$ where c is the name of the channel on which the communication takes place and v is the value of the

Petitioner Samsung Ex-1036, 0076

message that passes along the channel. Message components are put together using a dot. For example, message 1 of the Needham–Schroeder public key protocol is expressed by the event $comm.Msg1.a.b.Encrypt.kb.na.a$. Here the letters a and b stand for variables denoting principal identities; na stands for a variable denoting a nonce; and kb stands for a variable denoting a public key. The simplest way of constructing processes in CSP is by prefixing. A process that performs an action x and then behaves like process P is denoted $x \rightarrow P$ (pronounced 'x then P'). The $\rightarrow$ operator is right associative, so $x \rightarrow y \rightarrow P = x \rightarrow (y \rightarrow P)$. The initiator in the Needham–Schroeder public key protocol is defined by the CSP process $INITIATOR(a, na)$ below.

$$
\begin{aligned}
&INITIATOR(a, na) = \\
&\quad user.a?b \rightarrow I_running.a.b \rightarrow \\
&\quad comm!Msg1.a.b.Encrypt.kb.na.a \rightarrow \\
&\quad comm.Msg2.b.a.Encrypt.ka?na'.nb \rightarrow \\
&\quad \text{if } na = na' \\
&\quad \text{then } comm!Msg3.a.b.Encrypt.kb.nb \rightarrow \\
&\quad I_commit.a.b \rightarrow session.a.b \rightarrow Skip \\
&\quad \text{else} STOP
\end{aligned}
$$

The question marks model inputting of data. The initiator waits for the value for b from the channel *user* and then sends message 1. The event $I_running.a.b$ indicates that a is taking part in a protocol run with b. The initiator then waits for a corresponding message 2, decrypts this message and checks that the value for na' matches the same value sent in message 1 (the initiator will accept any value for nb). If the nonce matches, then the initiator sends message 3 and commits to the session. The event $I_commit.a.b$ represents the fact that the initiator is committing to a session with b; the event $session.a.b$ represents the fact that a carries out a session with b; and *Skip* represents a process that completes its task. If the nonce does not match then the initiator halts; this is represented by the CSP process $STOP$.

A CSP process $RESPONDER(b, nb)$ that captures the steps performed by the responder is defined similarly.

The intruder is modelled using the CSP choice operator $\Box$. The choice operator can be applied to any number of processes. The resulting process can choose to act like any one of the processes. For the Needham–Schroeder public key protocol the intruder is modelled with the choice operator applied to 12 processes: three for overhearing the messages sent by the honest principals, three for intercepting messages by the honest principals, three for replaying overheard messages, and three for generating messages using the known nonces and injecting them. The intruder is defined by the process $INTRUDER(m1s, m2s, m3s, ns)$, where the sets $m1s$, $m2s$, $m3s$ collect the undecrypted messages 1, 2 and 3 the intruder has overheard so far and ns

Petitioner Samsung Ex-1036, 0077

is the set of nonces the intruder has learnt. The part of the intruder process involving message 1 only is as follows; the parts involving messages 2 and 3 are similar.

$$
\begin{aligned}
&INTRUDER(m1s, m2s, m3s, ns) = \\
&\quad comm.Msg1?a.b.Encrypt.k.n.a' \rightarrow \\
&\quad \text{if } k = Ki \text{ then } I(m1s, m2s, m3s, ns \cup \{n\}) \\
&\quad \text{else } I(m1s \cup \{Encrypt.k.n.a'\}, m2s, m3s, ns) \\
&\quad \Box\ intercept.Msg1?a.b.Encrypt.k.n.a' \rightarrow \\
&\quad \text{if } k = Ki \text{ then } I(m1s, m2s, m3s, ns \cup \{n\}) \\
&\quad \Box\ fake.Msg1?a.b?m : m1s \rightarrow I(m1s, m2s, m3s, ns) \\
&\quad \Box\ fake.Msg1?a.b!Encrypt?k?n : ns?a' \rightarrow I(m1s, m2s, m3s, ns)
\end{aligned}
$$

To analyse the protocol, we need to restrict the CSP specification of the protocol and intruder so that the resulting system is finite. For example, Lowe's CSP model of the Needham–Schroeder public key protocol is limited to a single initiator A, a single responder B, and a single intruder. Despite this restriction, Lowe found an attack on this protocol. This finite CSP model of the protocol, represented by the process $SYSTEM$, is compared against other CSP processes that capture the desired security properties. Lowe formalises two authentication properties as the following CSP processes:

$$
\begin{aligned}
AI &= I_running.A.B \rightarrow R_commit.A.B \rightarrow AI \\
AR &= R_running.A.B \rightarrow I_commit.A.B \rightarrow AR
\end{aligned}
$$

The first process, AI, has the behaviour that the responder B commits to a session with initiator A only if A took part in the protocol run. The second process, AR, has the behaviour that the initiator A commits to a session with responder B only if B took part in the protocol run.

To check whether a given property holds for a protocol, one tests for refinement between the CSP processes representing the protocol and the property in question. In CSP, a process P refines process Q if every trace of P is also a trace of Q. Intuitively, a trace is a sequence of events. The first authentication property amounts to checking that $SYSTEM$ refines AI; the second property amounts to checking that $SYSTEM$ refines AR. The FDR tool is used for testing these refinements. It turns out that the refinement check involving AR succeeds, while the one involving AI does not. FDR produces a trace in the latter case, which shows that B commits to a session with A, although A never attempted to interact with B. This trace is the famous attack published by Lowe.

Lowe proposed a correction to the protocol to prevent the attack found on the original protocol. No attack on the corrected protocol is found using the model checking technique. However, since the model checking is carried out

Petitioner Samsung Ex-1036, 0078

on a small system running the protocol, the question remains whether there is an attack on an arbitrary system (with an arbitrary number of initiators and responders). To answer this question, Lowe offers a proof that any attack on the general protocol implies an attack on the smaller protocol. The proof is by hand and runs into several pages. Since no attack is found on the smaller system, he concludes that there can be no attack on an arbitrarily sized system.

Lowe [213] has written a program called CASPER to make his model checking technique accessible to protocol designers and implementers lacking specialist skills in CSP. CASPER automatically produces a CSP model of a protocol as well as the intruder using a more abstract description of the protocol, similar to the conventional description. The CASPER input language is machine-readable and provides special syntax to make explicit certain aspects of protocols. For example, the first message of the Needham–Schroeder public key protocol is represented as:

```
<pkb := PK(B)>
1. A -> B: {na, A}{pkb}
```

The first line represents that principal A uses a function PK to look up B's public key. As well as the definition of the sequence of messages passed between the principals, CASPER requires the definition of the actual system to be checked. To define the actual system, one has to instantiate the parameters appearing in the protocol definition to actual values. Consider a system with a single initiator, Alice, and a single responder, Bob, each of whom has a public key, a secret key and a nonce. This is defined by writing the following lines within the ♯ system heading in the CASPER input file.

```
INITIATOR(Alice, Na, SKa)
RESPONDER(Bob, Nb, SKb)
```

The size of the system to be model checked can be changed very easily.

CASPER provides a concise notation for specifying a range of security properties, including a number of authentication properties described in another paper by Lowe [214].

2.5.2 Murϕ

Murϕ (or just Murphi) is a general-purpose model checker with its own language. It has been used to verify several different kinds of protocols and algorithms over several years. Mitchell *et al.* [241] used Murϕ to analyse three key establishment protocols. They describe the general methodology and the results of their analysis.

As with other model checkers, the idea is to specify certain desired security properties and then search all reachable states and check whether the properties are always satisfied. The reachable states are limited by the number of

Petitioner Samsung Ex-1036, 0079

instances of each principal (including the adversary) that are modelled and the number of protocol messages. Mitchell *et al.* [241] reported that protocols with three to five messages and four or five principal instances could be analysed in a few minutes, but doubling these numbers could lead to an analysis requiring several hours and possible memory problems.

The methodology of an analysis with Murϕ is similar to that using FDR. Mitchell *et al.* [241] identify two main differences between the two tools.

1. FDR has an explicit notion of channels for communication between processes, whereas Murϕ models communication through shared variables.
2. Murϕ has a richer set of methods to reduce the searching time required for a given choice of protocol parameters.

The three protocols analysed by Mitchell *et al.* [241] are the Needham–Schroeder public key protocol, the TMN protocol and the Kerberos protocol. In each case the tool was able to quickly find previously known problems. Later, Mitchell *et al.* [242] used Murϕ to analyse the widely used SSL protocol. Although no attacks were found which revealed keys or violated authentication, some anomalies were identified in parts of the protocol involved with negotiation of the protocol version and parameters. One of these had not been previously recognised.

In common with other formal methods analyses, the model used with Murϕ has usually taken a simple 'perfect black-box' view of cryptography which does not distinguish authentication from encryption or recognise different types of confidentiality. However, Mitchell *et al.* [241] did model the multiplicative property of plain RSA encryption in their analysis of the TMN protocol. This means that the adversary is able to derive the ciphertext of multiples of a message m from the ciphertext of m. This is a particular type of malleability but does not capture the whole concept.

2.5.3 Brutus

Brutus is a model checking tool designed especially for analysis of cryptographic protocols. According to Clarke *et al.* [96] it was developed to allow 'push-button' checking of protocols so that designers can easily obtain the assurance of formal analysis. If a problem is found with the protocol then Brutus will provide an explicit attack.

As well as application to key establishment protocols, Brutus has been used to analyse protocols for electronic payments. Clarke *et al.* [96] give sample analyses of three protocols, two of which are for key establishment. One of these is the Needham–Schroeder public key protocol, for which they were able to rediscover the flaw previously found by Lowe using FDR. Another is the Wide-mouthed-frog protocol, which is discussed in Chap. 3. However, because their model does not include a notion of time they were unable to find known replay attacks (indeed they remove the timestamps from their analysis).

Petitioner Samsung Ex-1036, 0080

Messages are formed from keys, principal names and nonces, as well as any data to be conveyed. An assumption of *perfect encryption* is made, which requires that a ciphertext $\{m\}_k$ can only be formed with knowledge of key k and message m. Notice that this means that any symmetric encryption algorithm also provides data integrity, while asymmetric encryption must be non-malleable. The set of messages known to any principal is recorded.

A flexible logic allows many protocol properties to be specified suitable for different requirements. For example, for the Needham–Schroeder protocol three properties are checked. One is an *authentication* property which demands that if an instance of A has completed a protocol run apparently with B, then an instance of B must have at least started a protocol run with A. The second requirement is a *secrecy* property which requires that if the adversary knows a nonce of principal A then A must be running the protocol with the adversary. The final requirement is a 'weak' *non-repudiation* property which ensures that if an instance of A has finished a protocol run with B then B must know the nonce of A. The analysis finds that the protocol satisfies only the last of these properties.

Clarke *et al.* [96] provide an extended comparison between Brutus and other formal approaches to analysis of cryptographic protocols. They stress that the main benefit of their tool is the ease of operation for the practitioner, while the special-purpose language is more expressive than some general-purpose approaches. They also describe extensive measures to reduce the complexity of the search process.

2.5.4 NRL Analyzer

As part of a long-term project on cryptographic protocol analysis, the US Naval Research Laboratory (NRL) developed a special-purpose software tool known as the Analyzer [229]. It was one of the first tools developed for this purpose and has been continuously improved over many years to provide increased automatic support for the user. The tool is a Prolog program with several thousand lines of code.

The Analyzer is a hybrid that possesses features of both a model checker and a theorem prover. Searching begins from an insecure state looking to see if that state can be reached from the initial state. If so then an explicit attack has been found. Lemmas may be proven to show that infinite classes of states are unreachable. These may lead to a proof that all paths to the insecure state start in unreachable states.

The specification of a protocol consists of several elements such as:

System state which includes the values known by the adversary and the protocol principals, as well as event sequences that have occurred.

Protocol rules which state how honest principals behave and what is learnt by the adversary after each protocol step. The adversary may encrypt or decrypt with known keys and concatenate known values. It is assumed

Petitioner Samsung Ex-1036, 0081

that the adversary is able to recognise which key was used to encrypt any ciphertext.

Rewrite rules which define the cryptographic properties, such as that encryption and decryption are inverse operations.

Protocol goals are defined by the insecure states. As well as the values known to the adversary (such as keys) the state can include a notion of local variables (such as the value that some principal believes is the key) or conditions on sequences of events.

In common with other tools, cryptographic properties are defined implicitly by what values may be found by the adversary using known keys. However, there have been some extensions that take into account the specific properties of certain algorithms, in particular cipher-block chaining for block ciphers [312].

The Analyzer has been used to analyse a large number of protocols and to reproduce many known faults and find new ones. Initially the subjects were restricted to relatively simple key establishment protocols, but more recently the tool has been used to analyse complex protocols such as the Internet Key Exchange (IKE) protocol [230] (see Chap. 5 for a description of IKE).

2.5.5 BAN Logic

The logic of Burrows, Abadi and Needham (BAN) [76, 77] is one of the significant early formal methods for reasoning about authentication and key establishment protocols. The BAN logic stimulated widespread interest in the field. Proofs constructed in the BAN logic tend to be short and can be obtained quite easily by hand.

The syntax of the BAN logic distinguishes three types of primitive objects: *principals*, *keys* and *nonces*. A protocol message is expressed as a formula of the logic. Let P, Q range over principals; let K range over keys; let X, Y range over formulae. The formulae most frequently used in analysing shared key protocols along with their informal semantics are shown in Table 2.2.

The inference rules of the logic reflect intuitive consequences of the semantics of the logical constructs. Typically, an inference rule is read, 'if formulae $X_1, \ldots, X_n$ hold then formula Y holds', written more concisely as:

$$\frac{X_1, \ldots, X_n}{Y}$$

The following are some specific inference rules provided in the BAN logic [76]:

- *Message-meaning rule:*

$$\frac{P \models P \overset{K}{\leftrightarrow} Q,\ P \triangleleft \{X\}_K}{P \models Q \mid\sim X}$$

Petitioner Samsung Ex-1036, 0082

Table 2.2. Common BAN logic formulae

$P \models X$	P *believes* X; P believes that X is true.
$P \triangleleft X$	P *sees* X; P has received a message from which X can be read.
$P \mid\sim X$	P *once said* X; P has sent (or uttered) a message containing X.
$P \Rightarrow X$	P has *jurisdiction* over X; P is trusted on the truth of X.
$\sharp(X)$	X is *fresh*; X has not been sent previous to the current protocol run.
$P \overset{K}{\leftrightarrow} Q$	P and Q share key K which is *good* in the sense that it remains confidential to P, Q and principals trusted by either P or Q.
$\{X\}_K$	X encrypted with key K.

- *Freshness rule:*

$$\frac{P \models \sharp(X)}{P \models \sharp(X, Y)}$$

- *Nonce-verification rule:*

$$\frac{P \models \sharp(X),\ P \models Q \mid\sim X}{P \models Q \models X}$$

- *Jurisdiction rule:*

$$\frac{P \models Q \Rightarrow X,\ P \models Q \models X}{P \models X}$$

To analyse a protocol using the BAN logic, one has to transform the actual or concrete protocol into an idealised protocol. An idealised protocol is a set of logical formulae that express the intended meaning of the protocol messages. An example helps convey the basic idea behind idealisation. Consider the message exchange $S \rightarrow A : \{N_A, B, K_{AB}\}_{K_{AB}}$, in which a trusted server S distributes a session key K_{AB} to be shared between A and B. Here K_{AB} is a key shared by A and S, and N_A is A's nonce, which is used by A to verify that the message is not a replay. Suppose the above message exchange is part of some protocol, and that this message exchange implies that S asserts K_{AB} to be a good session key for A and B. Then this assertion is typically reflected in the protocol idealisation by the formula: (1) $A \triangleleft \{N_A, A \overset{K_{AB}}{\leftrightarrow} B\}_{K_{AS}}$. To proceed with the analysis, some formulae that express initial assumptions about the protocol in question are also asserted, and the inference rules are then applied to determine whether the formulae expressing the goal of the protocol are derivable using the logic. For example, the above idealisation can be accompanied by the following assumptions: (2) $A \models \sharp(N_A)$; and (3) $A \models S \Rightarrow A \overset{K_{AB}}{\leftrightarrow} B$. We can derive the formula $A \models A \overset{K_{AB}}{\leftrightarrow} B$ from (1), (2) and (3) using the logic. The import of the logical analysis is that it forces us to make explicit the various assumptions needed to obtain the desired goals. If the desired goals do not follow from an application of the logic, the preconditions to the inference rules often provide a hint to further assumptions

Petitioner Samsung Ex-1036, 0083

that might be needed. Intuitively, if an unreasonable assumption is found in the process, it suggests the presence of a protocol flaw. For example, the BAN logic has been used to verify a flaw in the Needham–Schroeder protocol pointed out by Denning and Sacco [99]. In analysing the protocol using their logic, Burrows *et al.* [76] show that this flaw manifests as a dubious statement amounting to the *assumption* that one of the parties believes that the session key distributed via the protocol is fresh.

The BAN logic has been successfully used in this way to demonstrate serious flaws in several published protocols from the literature. However, it has a rather limited ability to detect protocol flaws, as many flaws do not manifest themselves in the logical manipulation needed to construct a formal proof of a protocol's correctness using the logic. Rather they manifest themselves only indirectly, either in one's inability to verify the validity of the particular idealisation of the protocol chosen or in one's inability to verify that the various assumptions made in the logic always hold for the duration of a protocol run. This difficulty in the use of the BAN logic is well known but is easily overlooked in the practical use of the logic by protocol designers, thereby allowing incorrect protocols to be reasoned as correct. Boyd and Mao [60] have demonstrated two examples of flawed protocols that can be proven correct using the BAN logic; van Oorschot [323] has pointed out that the proof of one protocol relied on an incorrect idealisation and the proof of the other relied on an incorrect assumption. A more controversial example is provided by a protocol due to Nessett [250] (we omit Message 2 from the original protocol since it is not relevant to our discussion):

$$\text{Message 1. } A \rightarrow B : \{N_A, K_{AB}\}_{K_A^{-1}}$$

Here B trusts A to generate a session key K_{AB} to be shared between them. The nonce N_A is used by A to convince B that Message 1 is fresh. K_A^{-1} is A's private key for use in a suitable public key system. The corresponding public key K_A is publicly known. Nessett [250] asserts some formulae meant to reflect the initial assumptions, which include amongst others the formula $A \models A \overset{K_{AB}}{\leftrightarrow} B$, and shows that the BAN logic can be used to sanction the protocol in the sense that the formula $B \models A \overset{K_{AB}}{\leftrightarrow} B$ can be derived from Message 1. The protocol of course is insecure, nevertheless, since everyone knows K_A and can thus decrypt Message 1 to obtain K_{AB}. Based on his analysis of the protocol, Nessett claims that the BAN logic is *flawed.* However, Burrows *et al.* [78] have argued that Nessett's claim is invalid. They pointed out that the assumption $A \models A \overset{K_{AB}}{\leftrightarrow} B$ is unjustified since Message 1 contradicts this assumption. Informal guidelines to avoid incorrect idealisations have been proposed by the BAN authors [79] (see also [323]), but as Mao and Boyd [222] have argued, applying them correctly may be no less difficult than finding a protocol error by informal means.

Many logics have been proposed by others as extensions or improvements over the BAN logic. Prominent proposals include the logics suggested by Gong

Petitioner Samsung Ex-1036, 0084

et al. [131], Abadi and Tuttle [6], and Syverson and van Oorschot [315]. However, a positive result of an analysis using any of these logics still leads to a limited assurance in protocol correctness, like the original BAN logic.

2.5.6 Strand Space Model

The strand space model is a framework for producing rigorous yet mathematically straightforward proofs of protocol correctness. According to Fabrega *et al.* [108], proofs in the strand space model can be more convincing than more 'conceptual' proofs such as proofs in belief logics, while being simple enough to be carried out by the practitioner. As with proofs in belief logics, strand space proofs of correctness of typical protocols can be obtained without automated support.

A strand represents the behaviour of a principal or the adversary in terms of message sends or receives. The elements of a strand are called nodes and the set of all possible strands is called a strand space. In the strand space model, the sending of a message m is represented by $+m$ and the receiving is represented by $-m$. A strand is therefore a sequence of sign prefixed messages. For example, the sequence $\langle +\{N_A, A\}_{K_B}, -\{N_A, N_B, B\}_{K_A}, +\{N_B\}_{K_B} \rangle$ represents an initiator strand in the Lowe–Needham–Schroeder protocol for each particular choice of parameters A, B, N_A and N_B. (This protocol is different from the original Needham–Schroeder public key protocol only in that B's name is included in the second message.) The abilities of the adversary are also represented by a set of adversary strands. A peculiarity of this approach is that the messages generated by the adversary are transmitted publicly. For example, K-type adversary strands, $\langle +K \rangle$, emit a key K from a set $\mathcal{K}_{\mathcal{P}}$ of keys initially known to the adversary, which includes all private keys of the adversary.

The main idea of proving protocol properties using strand spaces is to use the induction principle on well-founded sets. Proofs in the strand space model are carried out with respect to structures called bundles. A bundle is a well-founded set of strands satisfying the following two conditions: (1) every message received by a strand is actually sent previously in some strand present in the bundle and (2) if a node occurs on a strand then all nodes that would have preceded itself on the same strand also occur on the strand. Bundles thus represent individual runs of a protocol.

Different kinds of authentication properties can be proved using the strand space framework. For example, the proofs obtained by Fabrega *et al.* for the the Lowe–Needham–Schroeder protocol include assurances that the protocol achieves injective agreement on N_A and N_B for both A and B. The injective agreement for the initiator demands that if principal A completes a run of the protocol as initiator with B, then there exists a unique run of the protocol with the principal B as responder, and both agree on the specific values of the nonces N_A and N_B. The injective agreement for the responder demands that if principal B completes a run of the protocol as responder with A, then there

Petitioner Samsung Ex-1036, 0085

exists a unique run of the protocol with principal A as initiator, and both agree on the specific values of the nonces N_A and N_B. The above agreement properties are proved by establishing that whenever an initiator or responder strand belongs to a bundle, then a matching strand exists uniquely in the bundle. The analysis finds that the original Needham–Schroeder protocol does not guarantee injective agreement for the responder. The strand space proofs for the Lowe–Needham–Schroeder protocol also include guarantees that the initiator's nonce N_A and the responder's nonce N_B remain secret in the protocol. The secrecy properties are proved by establishing that no node ever has the nonce values unprotected.

Song [304] has designed a model checking algorithm, Athena, based on the strand space model. One of the main benefits of Athena, as compared to other model checking approaches, is that it does not require a large number of states to be checked unnecessarily. For the Needham–Schroeder public key protocol with one initiator and one responder, Murϕ explores over 1500 times as many states as Athena, whereas Brutus with partial order and symmetry reductions explores over 150 times as many states as Athena.

2.5.7 The Inductive Model

The idea of using induction is to prove results about *all* of the infinite possible protocol states without having to explicitly examine each one. The use of induction as a technique for verification of cryptographic protocols was pioneered by Paulson [263] using the proof tool called Isabelle with the formalism known as higher order logic (HOL).

Induction is a familiar proof technique in mathematics. If $P(n)$ is a proposition indexed by an integer n then an induction proof requires a base case, $P(0)$, to be established along with a general result that $P(i) \Rightarrow P(i+1)$. It can then be concluded that $P(n)$ is true for any positive integer n. The idea of an inductive proof is to identify all desirable properties P of a protocol and show that P is preserved under all the allowed extensions of any given set of observed events. Properties for different principals can be proven by including an assumption that the final message received by that principal was properly delivered.

An inductive proof can be more convincing than the results of a model checker that finds no attack, but there are drawbacks too. One is the lack of an explicit attack in the case that the proof does not work. Another is that the process tends to be significantly longer and requires more effort. Paulson reports that proof construction for a typical key establishment protocol takes many days to complete, although this can be reduced to hours when an existing proof needs to be modified due to a change in the protocol.

As with most other formal specifications, messages are made up of principal names, nonces and keys. Messages can also be concatenated to form compound messages. Cryptography is limited to two functions: hashing and

Petitioner Samsung Ex-1036, 0086

encryption. Encryption follows the usual 'perfect' model in which messages cannot be either read or written without the appropriate key.

Three operators are defined on any set of messages H.

- `parts` H represents the set of components that can be recovered from H, which may require knowledge of certain keys.
- `analz` H represents the set of components that can be recovered from H without knowing any secrets (apart from what may already be in H).
- `synth` H is the set of messages that can be formed from H by adding principal names, concatenating message components, or encrypting with keys in H.

The adversary observes the set H of messages sent on the network and can send messages from the set `synth(analz` H). Because the adversary is modelled in the same way for all protocols, general lemmas and proof procedures that have been derived can be used in any protocol proof. The adversary is allowed to start or continue any run of the protocol between any principals. A protocol specification describes how the set of observed messages may be extended by adding messages that would be output by principals who received messages which the adversary can read or synthesise from H. The adversary may also be allowed to obtain old session keys.

Paulson [263] describes proofs for three protocols. For the Otway–Rees protocol he was able to discover an attack on a variant proposed by Burrows *et al.* [76], guided by the failure of the proof. In the Needham–Schroeder public key protocol, secrecy proofs for the initiator can be derived but not the responder, due to the attack previously found by Lowe. When fixed in the manner suggested by Lowe, a corresponding proof for the responder is provided. The third protocol analysed is a 'recursive' version of the Otway–Rees protocol involving multiple entities, which will be discussed further below. In later papers Paulson provided an extended analysis of the Yahalom protocol [266] and proofs for a simplified version of the TLS protocol [264].

The proofs obtained by Paulson for the recursive Otway–Rees protocol include assurances that the adversary cannot obtain session keys. Such proofs are interesting to consider in the light of an attack found by Ryan and Schneider [280] which does result in compromise of session keys. The attack was described by Ryan and Schneider as 'easy to spot and ... easy to mount'. How then could the proof of security be formed? The answer is that the concrete encryption scheme used in the recursive protocol does not provide the properties assumed for 'perfect encryption'. This provides an instructive lesson that proofs are only valid if their assumptions are satisfied.

The proofs constructed by the inductive approach using formal specification are very different from the complexity-theoretic proofs that we will examine in Sect. 2.6. In particular:

- the proofs are machine checkable, which means that there is no possibility for human error in the formal arguments;

Petitioner Samsung Ex-1036, 0087

- in common with other formal methods approaches, the cryptographic definitions are very simplified in comparison to those in the complexity-theoretic proofs. Therefore cryptographic attacks cannot be ruled out.

2.5.8 Comparison of Formal Methods Approaches

In this section we have briefly introduced some of the most prominent formal approaches to analysis of authentication and key establishment protocols. Although there are a number of related ideas, each method seems to have both strengths and weaknesses. Our list has certainly not been exhaustive and research in this area continues to be active. The majority of the formal methods analysis techniques rely on computer assistance, although there are exceptions such as the BAN logic and its derivatives, and the strand space model. Table 2.3 summarises some of the properties of several of the analysis tools.

Table 2.3. Summary of some tools used for protocol analysis

Properties → ↓ *Tool*	Type	Language	Usage
FDR	Model checker	CASPER	Automatic
Murϕ	Model checker	Special	Automatic
Brutus	Model checker	Special	Automatic
NRL Analyzer	Hybrid	Special	User-guided
Athena	Model checker	Special	Automatic
AAPA	Theorem prover	HOL	Automatic
Isabelle	Theorem prover	HOL	User-guided

The properties mentioned in Table 2.3 do not include any measure of how useful each tool is in providing assurance about protocol security. This is very difficult to compare, especially when tools use different techniques. However, roughly we can divide the tools into those that are automatic and those which require user intervention. It seems reasonable to suggest that an automatic tool should be used in the early protocol design stage to filter out any simple errors while the more complex tools, which generally provide proofs of some sort, should be used at a later stage.

All of the tools in Table 2.3 have been mentioned in previous subsections except for the Automated Authentication Protocol Analyzer (AAPA) [66]. AAPA (and its successor AAPA2) is an automatic tool that provides proofs in a modified version of the Gong, Needham and Yahalom (GNY) logic [131]. Brackin, the developer of AAPA, has stated that the aim of AAPA is to be fast and automatic while discovering the majority of protocol flaws. Using the

Petitioner Samsung Ex-1036, 0088

protocols in the Clark–Jacob survey [95], Brackin [67] found that AAPA2 was able to correctly classify 88% of the protocols as flawed or unflawed, using an average time of 2.6 minutes on a modest computer.

2.6 Complexity-Theoretic Proofs of Security

An important direction in cryptographic protocol research was pioneered by Bellare and Rogaway in 1993 [30] when they published the first mathematical proof that a simple entity authentication protocol was secure. This work, which covered only the two-party case, was followed up with a paper on server-based protocols [33] and various authors have extended the same idea to include public-key-based key transport [44], key agreement protocols [45], password-based protocols [29, 65], and conference key protocols [70, 68].

The general approach is the same as that used in the cryptographic research community to prove security of cryptographic algorithms. These are complexity-theoretic reduction proofs; the security of the subject algorithm or protocol $\mathcal{S}$ is *reduced* to the security of another better understood problem $\mathcal{P}$ in the sense that if there is an efficient algorithm that can break $\mathcal{S}$ then there is an efficient algorithm to solve $\mathcal{P}$. One limitation of this approach is that we usually have no guarantee that $\mathcal{P}$ really is hard to solve; for example, $\mathcal{P}$ might be the integer factorisation problem.

The approach uses a mathematical model that defines a protocol in which a powerful adversary plays a central role. The adversary essentially controls all the principals and can initiate protocol runs between any principals at any time. Insider attacks are modelled by allowing the adversary to corrupt any principals, and the adversary can also obtain previously used keys. Cryptographic algorithms may be modelled either with generic properties or as specific transformations. Security of protocols is defined in terms of matching conversations (for authentication) and indistinguishability (for confidentiality of keys).

The significance of a proof of security remains somewhat controversial. Some of the potential limitations are the following.

- Current provable security techniques do not help in protocol design. A small change in the protocol will require a new proof to be constructed.
- Security proofs tend to be difficult to understand for the average practitioner. They typically run to several pages of mathematical reasoning.
- Although there are relatively few protocols with security proofs available, a number of protocols for which proofs were claimed have turned out to have significant security weaknesses. (Examples are discussed in Sect. 5.7.3.) The lack of accessibility of the proofs means that there are few people who check the proofs in any detail.

Even if you agree with these points, they do not imply that a proof of security is not worthwhile. Our view is that a security proof is a very valuable

Petitioner Samsung Ex-1036, 0089

property of a protocol but that it should be supplemented by informal scrutiny, machine analysis, and any other approaches available to gain assurance.

Many complexity-theoretic proofs for protocols rely on the so-called *random oracle model* [31]. This model assumes that a hash function works in an idealised manner: it returns truly random values each time it is used, except that it always returns the same output when the input is the same. No such function exists in reality, but a random oracle seems to be representative of the way that we want hash functions to work. The research community has differing views on the reasonableness of the random oracle model, but everyone agrees that a proof that does not rely on it is preferable to one that does.

In the remainder of this section we outline the model proposed by Bellare and Rogaway to give an idea of the way that the proofs work. We will then explain their definition of security for protocols. Following this we outline an alternative model of Shoup, which uses a different security definition. Finally, we mention briefly a newer modular approach which is in some ways a hybrid of the previous two.

2.6.1 Model of Communication

The model of communication used is independent of the details of the protocol and is the same for all protocols with the same set of principals and the same protocol goals. The adversary controls all the communications that take place and does this by interacting with a set of *oracles*, each of which represents an instance of a principal in a specific protocol run. The principals are defined by an identifier U from a finite set and an oracle Π_U^s represents the actions of principal U in the protocol run indexed by integer s. Interactions with the adversary are called oracle *queries* and the list of allowed queries is summarised in Table 2.4. This list applies to the model appropriate for server-based key transport protocols, as described in Bellare and Rogaway's 1995 paper [33]; additional queries are appropriate for other protocol types. We now describe each one informally.

Table 2.4. Queries available to the adversary in Bellare–Rogaway model

$\mathsf{Send}(U, s, M)$	Send message M to oracle Π_U^s
$\mathsf{Reveal}(U, s)$	Reveal session key (if any) accepted by Π_U^s
$\mathsf{Corrupt}(U, K)$	Reveal state of U and set long-term key of U to K
$\mathsf{Test}(U, s)$	Attempt to distinguish session key accepted by oracle Π_U^s

$\mathsf{Send}(U, s, M)$ This query allows the adversary to make the principals run the protocol normally. The oracle Π_U^s will return to the adversary the next

Petitioner Samsung Ex-1036, 0090

message that an honest principal U would do if sent message M according to the conversation so far. (This includes the possibility that M is just a random string in which case Π_U^s may simply halt.) If Π_U^s accepts the session key or halts this is included in the response. The adversary can also use this query to start a new protocol instance by sending an empty message M in which case U will start a protocol run with a new index s.

Reveal(U, s) This query models the adversary's ability to find old session keys. If a session key has previously been accepted by Π_U^s then it is returned to the adversary. An oracle can only accept a key once. (Of course a principal can accept many keys modelled in different oracles.)

Corrupt(U, K) This query models insider attacks by the adversary. The query returns the oracle's internal state and sets the long-term key of U to be the value K chosen by the adversary. The adversary can then control the behaviour of U with Send queries.

Test(U, s) Once the oracle Π_U^s has accepted a session key K_s the adversary can attempt to distinguish it from a random key as the basis of determining security of the protocol. A random bit b is chosen; if $b = 0$ then K_s is returned while if $b = 1$ a random string is returned from the same distribution as session keys.

2.6.2 Defining Security

Success of the adversary is measured in terms of its *advantage* in distinguishing the session key from a random key after running the Test query. If we define Good-Guess to be the event that the adversary guesses correctly whether $b = 0$ or $b = 1$ then:

$$\text{Advantage} = 2 \cdot \Pr[\text{Good-Guess}] - 1.$$

The Test query may only be used for an oracle that has not been corrupted and has accepted a session key that has not been revealed in a Reveal query. In addition the *partner* of the oracle to be tested must not have had a Reveal query. The way of defining partner oracles has varied in different papers using the technique. In the more recent research, partners have been defined by having the same session identifier (SID) which consists of a concatenation of the messages exchanged between the two. Partners must both have accepted the same session key and recognise each other as partners.

Bellare and Rogaway define a protocol in this model to be secure if:

1. when the protocol is run without any intervention from the adversary then both principals will accept the same session key;
2. Advantage is negligible for any efficient adversary.

The first condition is a completeness criterion that guarantees that the protocol will complete as expected in normal circumstances. The second condition says that the adversary is unable to find anything useful about the session key after interacting in the specified way. One may ask how this condition relates

Petitioner Samsung Ex-1036, 0091

to more conventional protocol goals such as Defn. 2.6. Although it appears to be concerned only with key confidentiality it does imply key authentication. For suppose that the session key is known to an oracle Π_U^s different from that recorded in an oracle $\Pi_{U'}^t$ to be tested. Then Π_U^s is not the partner of $\Pi_{U'}^t$ and so it can be opened by the adversary and so the protocol cannot be secure. A key is defined as fresh in the Bellare–Rogaway model if it has been accepted by an oracle that has not been opened, its partner (if any) has not been opened, and the user it represents has not been corrupted. Since the Test query can only be performed on oracles with fresh keys, all keys accepted in a secure protocol must be fresh. We conclude that the conditions of Defn. 2.6 have indeed been met.

In their 1995 paper Bellare and Rogaway proved the security of a server-based protocol (Protocol 3.35). The idea to prove security is to assume that the adversary can obtain a significant advantage and use this to show that the encryption mechanism does not provide indistinguishability. Then, as long as the encryption mechanism is assumed to be secure, we deduce that the adversary cannot obtain this significant advantage. In order to achieve this goal a *simulator* of the oracles is defined which is efficiently constructed from public information assuming that the usual attacks are possible (which essentially means that anything can be known apart from the long-term key of the oracle whose key is to be guessed and that of its partner). The simulator is fed two random ciphertexts and asked to distinguish which of two plaintexts these correspond to. The ciphertexts are used in randomly chosen oracle queries. If these oracles are chosen by the adversary for the Test query it will return the correct guess with the assumed significant advantage unless it fails. The authenticated messages are used to show that the simulator succeeds with significant probability.

2.6.3 Shoup's Simulation Model

In the same tradition as the Bellare–Rogaway model, Shoup [299] developed a formal model in which to prove security of protocols. We call this Shoup's *simulation model.* There are many similarities with the Bellare–Rogaway model, in particular the adversary runs the network to set up all connections, and has various attacking options. We may regard Shoup's model as being more abstract than Bellare–Rogaway's, and indeed it is formally shown to be a generalisation. The major novelty is that Shoup defines two systems, an *ideal* system and a *real* system.

In the ideal system the adversary controls the network and also chooses all the keys and random values used by the protocol principals. The adversary can run the protocol by initialising users, starting and aborting sessions, and interacting with applications. This last option points out a difference from the Bellare–Rogaway model, since Shoup's model includes the possibility of using the agreed key in an *application.* Use of the session key is ruled out in the

Petitioner Samsung Ex-1036, 0092

Bellare–Rogaway model since it allows the adversary to distinguish between the session key and a random value.

In the real system the adversary still controls the network but is not given the keys and random values chosen by the principals. The adversary can initialise principals and can invoke instances of principals to respond to messages as in the Bellare–Rogaway model. In addition, a trusted third party, TTP, is defined, and principals can interact with TTP to obtain long-term keys. This allows explicit inclusion of certificate information for public keys in the model, another distinction from that of Bellare–Rogaway. Security is defined in a simple manner.

1. Each principal must terminate after a small (polynomial) number of message interactions.
2. If the adversary simply relays messages faithfully between principals then both accept and share the same session key.
3. For any efficient adversary in the real world there must exist an efficient adversary in the ideal world such that it is computationally infeasible to differentiate between the behaviour of the two adversaries and the information gained by them.

This notion of *simulatability* is familiar in cryptography literature as the basis for zero knowledge proofs. The motivation behind it is that whatever the adversary could gain by interacting with the real system, could have been gained in the ideal system. But by construction the ideal system gains nothing for the adversary. An attractive feature of this security definition is that it is independent of the protocol goals. Shoup defines three shades of security by allowing the adversary different powers to corrupt principals.

Static Corruptions cover the case in which the adversary does not use interactions with the principals to decide which principals to corrupt. In this case corruption is not explicitly modelled but any real system adversary is able to *register* principals for which the secrets and random values are known. Shoup proved that security with static corruptions in his simulation model is equivalent to security in the Bellare–Rogaway model. This seems paradoxical since Bellare and Rogaway explicitly allow the adversary to corrupt principals at any time. However, Shoup points out that there is an efficient reduction from the Bellare–Rogaway model to the same model in which only static corruptions are allowed.

Adaptive Corruptions are those in which the adversary can choose which principals to corrupt at any time. Protocols that are secure against an adversary who can adaptively corrupt must provide forward secrecy. The reason is that if old session keys could be found from a corrupted long-term key then it must be possible to simulate this in the ideal world where no such compromise is available. Shoup shows that security against adaptive corruptions in the simulation model is equivalent to security with forward secrecy in the Bellare–Rogaway model.

Petitioner Samsung Ex-1036, 0093

Strong Adaptive Corruptions are adaptive corruptions in which the adversary obtains not only the long-term key of corrupted principals, but also any short-term secrets that have not been explicitly erased. Shoup has explained how to maintain a secure session that is compatible with security of such a protocol.

2.6.4 A Modular Approach to Proofs

A limitation of the models of both Bellare–Rogaway and Shoup is that new protocols need to be proven secure afresh when any change is made. A step towards a reusable framework for design of proven secure protocols was taken by Bellare *et al.* in 1998 [27]. They called their method a *modular* approach. The basic idea is to prove simple protocols secure in a setting in which ideal authentication is provided. This means that the adversary is limited to attacks that use only existing messages. The simple protocol can then be proven secure in the 'real' world by applying a general transformation called an *authenticator*. Since there can be many simple protocols and many different authenticators this approach can result in multiple protocols being proven secure together.

The 1998 paper of Bellare *et al.* used a definition of security based on the simulation approach. However, this turned out to be too restrictive to be very useful so in 2001 Canetti and Krawczyk [82] re-worked the model to use a security definition based on indistinguishability similar to the older Bellare–Rogaway definition. An important additional step in this work was a proof that following the key establishment phase with a secure communication phase still remains secure. The communications phase is limited to use of the session key for encryption and authentication of messages in a well-defined way.

At the time of writing there is still relatively little experience with the formal proof models outlined in this section. Currently the Canetti–Krawczyk approach seems the most promising. The security community would benefit from efforts to make these models more accessible.

2.7 Conclusion

Protocol goals are not simple to define, even for key establishment and authentication. However, a clear understanding of what a protocol is intended to achieve is a prerequisite for deciding whether its goals can be achieved.

The first half of this chapter was concerned with establishing informal definitions of various protocol goals. As well as the basic goals for user- and key-oriented concerns we also examined more advanced goals that can sometimes be achieved. These included goals that may allow something to be salvaged when the disaster of a long-term key compromise occurs.

The second half of the chapter looked briefly at different methods and techniques that have been used for protocol analysis. This is still an active

Petitioner Samsung Ex-1036, 0094

research area and we cannot say that there is yet a foolproof method to ensure that a particular protocol will have no flaws, especially if that protocol is relatively complex or has unusual goals. Meadows [232] discusses various promising research directions which the formal methods community could follow to extend existing approaches.

Petitioner Samsung Ex-1036, 0095

3

Protocols Using Shared Key Cryptography

3.1 Introduction

The majority of protocols for key establishment and entity authentication that have been proposed in the literature concentrate on the case where there are exactly two users who wish to communicate or establish a session key. This is commonly referred to as the two-party case. In this chapter we discuss two-party key establishment and authentication protocols based on symmetric algorithms. The next chapter discusses two-party protocols using public key algorithms, while the multi-party case is covered in Chap. 6.

We can classify two-party key establishment protocols using the following two criteria discussed in Chap. 1:

1. The cryptographic keys available *a priori*.
2. The method of session key generation.

If only shared keys are available to establish a new session key, there are essentially two cases to consider with respect to criterion 1:

a. The two principals already share a secret key.
b. Each principal shares a key with a trusted server.

Criterion 2 is concerned with the method of session key generation, for which there are three different possibilities in general: key transport, key agreement and hybrid. If a protocol has only two principals and is server-less, as in case (a) above, one cannot distinguish between key transport and hybrid key generation. The criteria mentioned above lead to the classification of $1 \times 2 + 1 \times 3 = 5$ different classes of protocols. The recognition of the criteria allows two-party shared key protocols found in the literature to be classified into one of the above five classes. However, in this chapter we mainly emphasise the division between server-less and server-based protocols.

In the remainder of this section, we explain the notation used to describe protocols in this chapter. Section 3.2 discusses protocols aimed at providing

Petitioner Samsung Ex-1036, 0096

entity authentication without key establishment. Section 3.3 discusses protocols aimed at providing key establishment without a server, including key transport protocols and key agreement protocols. Section 3.4 discusses protocols aimed at providing key establishment using a server, including key transport protocols, key agreement protocols and hybrid protocols. For each class of protocols, we provide published protocols from the literature.

Notation

A widely used notation for denoting a message part as a ciphertext is $\{M\}_K$ where M is the input data to a symmetric encryption algorithm that is parametrised by the secret key K. There are other familiar notations for indicating the use of encryption when specifying protocols but the above notation remains the most popular one. Frequently, protocol designers use the above-mentioned notation to imply that encryption provides both confidentiality and integrity properties. Recall from Chap. 1 that the confidentiality property is fundamental to encryption algorithms whereas the integrity property is not. The traditional notation ignores the fact that not all encryption algorithms possess the second property and encourages designers to assume that both these properties should be applied together. It also brings with it additional responsibility on the protocol implementer when choosing a specific encryption algorithm.

If the protocol designer makes use of encryption for achieving both confidentiality and integrity, we use the traditional notation for encryption when presenting the protocol. Some designers use one notation for cryptographic transformations that provide message integrity and another notation for cryptographic transformations that provide confidentiality. In such cases, we use the notation $[\![\cdot]\!]_K$ to denote a ciphertext obtained from an encryption algorithm that provides the confidentiality property alone and the notation $[\cdot]_K$ to denote a ciphertext obtained from a one-way cryptographic transformation such as a MAC which provides the integrity property alone.

The notation used in this chapter is summarised in Table 3.1.

Table 3.1. Notation used throughout Chap. 3

A and B	The two users who wish to share a new session key
S	A trusted server
$\{M\}_K$	Encryption of message M with key K to provide confidentiality and integrity
$[\![M]\!]_K$	Encryption of message M with key K to provide confidentiality
$[M]_K$	One-way transformation of message M with key K to provide integrity

Petitioner Samsung Ex-1036, 0097

3.2 Entity Authentication Protocols

Protocols that aim at providing entity authentication without key establishment are relatively scarce in the literature. Perhaps this is because the variety of approaches is quite limited, or maybe because the usefulness of such protocols is questionable as discussed in Chap. 2. In this section we examine the prominent examples and discuss whether they achieve the extensional definition of entity authentication introduced in Chap. 2, or the simpler liveness property.

3.2.1 Bird–Gopal–Herzberg–Janson–Kutten–Molva–Yung Protocols

A paper by IBM researchers in 1993 [41] was one of the first to demonstrate a wide class of attacks on several authentication protocols, including a draft protocol proposed by ISO. Based on the attacks on these protocols, the authors develop a set of security criteria to avoid such attacks and propose protocols that meet their criteria. They start with a basic protocol which they did not regard as secure, and improve it in a series of steps through consideration of various attacks and other design requirements. Eventually, a good protocol is achieved.

Protocol 3.1 is the basic protocol from their paper. Here u and v are two functions such that K_{AB} is needed to calculate them, and they do not give away K_{AB}.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : N_B, u(K_{AB}, N_A, \ldots)$
3. $A \rightarrow B : v(K_{AB}, N_B, \ldots)$

Protocol 3.1: Bird *et al.* canonical protocol 1

Initially we will assume that the functions u and v are the same. Under this assumption the protocol does not achieve the matching conversation goal, which Bird *et al.* regarded as important for the security of any authentication protocol. In Attack 3.1, A is used as an oracle against B. Suppose I is an adversary who wishes to attack the protocol.

- I starts a protocol run with B while masquerading as A.
- In parallel, I starts a protocol run with A while masquerading as B.

Attack 3.1 certainly violates intensional goals since B accepts even though the conversations do not match. In light of this attack, Bird *et al.* conclude that the functions u and v must be different from one another so that A's reply in the parallel session cannot be used by the adversary to complete the

Petitioner Samsung Ex-1036, 0098

1. $I_A \rightarrow B : N_I$
2. $B \rightarrow I_A : N_B, u(K_{AB}, N_I, \ldots)$
1'. $I_B \rightarrow A : N_B$
2'. $A \rightarrow I_B : N_A, u(K_{AB}, N_B, \ldots)$
3. $I_A \rightarrow B : u(K_{AB}, N_B, \ldots)$

Attack 3.1: An oracle attack on Protocol 3.1

first run. While the condition that u and v be different is adequate to prevent the attack, it is not necessary to achieve the extensional specification. As long as the identities of the sender and the intended partner are included inside the functions u and v, the extensional goal can be achieved even if the functions u and v are the same. Of course, a similar attack on such a protocol is still possible but it would not violate the extensional definition.

3.2.2 Bellare–Rogaway MAP1 Protocol

The MAP1 mutual authentication protocol was proposed in a landmark paper of Bellare and Rogaway [30]. They provide a formal definition of matching conversations and showed that MAP1 is provably secure. The messages are shown in Protocol 3.2.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : N_B, [B, A, N_A, N_B]_{K_{AB}}$
3. $A \rightarrow B : [A, N_B]_{K_{AB}}$

Protocol 3.2: Bellare–Rogaway MAP1 protocol

In the Bellare–Rogaway model of security (see Sect. 2.6), an adversary may only interact with sessions of the same protocol. An attack of Alves-Foss [11] shows why this assumption is important in practice. He designed Protocol 3.3, known as EVE1, which can also be shown to be provably secure.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : N_B, [A, B, N_A, N_B]_{K_{AB}}$
3. $A \rightarrow B : [A, N_B]_{K_{AB}}$

Protocol 3.3: Protocol for attacking MAP1 protocol

The only difference between MAP1 and EVE1 is that the identities of A and B are swapped in message 2. A chosen protocol attack on the MAP1

Petitioner Samsung Ex-1036, 0099

protocol is now possible. Suppose I is an adversary who wishes to attack the protocol. In Attack 3.2 A is used as an oracle against herself.

- I masquerades as B in a run of the MAP1 protocol started by A.
- In parallel, I starts a run of the EVE1 protocol with A while masquerading as B.

---

1. $A \rightarrow I_B : N_A$
1′. $I_B \rightarrow A : N_A$
2′. $A \rightarrow I_B : N'_A, [B, A, N_A, N'_A]_{K_{AB}}$
2. $I_B \rightarrow A : N'_A, [B, A, N_A, N'_A]_{K_{AB}}$
3. $A \rightarrow I_B : [A, N'_A]_{K_{AB}}$

Attack 3.2: Chosen protocol attack on MAP1

Is Attack 3.2 on the MAP1 protocol valid? A reasonable conclusion may be that the attack is invalid since it violates an assumption of the model used in proving the protocol secure. On the other hand, the attack is a reminder that provable security does not guarantee security against chosen protocol attacks.

3.2.3 ISO/IEC 9798-2 Protocols

The international standard ISO/IEC 9798 Part 2 [160] specifies six protocols using symmetric encryption algorithms. Four of these protocols are intended to provide entity authentication alone, while two are intended to provide key establishment as well as entity authentication. These last two are essentially identical to two protocols in the ISO/IEC 11770-2 standard, which are described in Sect. 3.4.4. Two of the four protocols aimed solely at entity authentication are concerned with unilateral authentication, while the other two are concerned with mutual authentication. Below we examine these protocols with some optional text fields omitted.

The first protocol, shown as Protocol 3.4, consists of a single message from a claimant A to a verifier B. It provides unilateral entity authentication of A to B. The timestamp T_A allows B to deduce that A is live, while inclusion of the identity B ensures that A has knowledge of B are her peer entity.

$A \rightarrow B : \{T_A, B\}_{K_{AB}}$

Protocol 3.4: ISO/IEC 9798-2 one-pass unilateral authentication protocol

The second protocol (3.5) is similar to the first, but uses a nonce instead of a timestamp. It provides the same properties as Protocol 3.4.

Petitioner Samsung Ex-1036, 0100

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : \{N_B, B\}_{K_{AB}}$

Protocol 3.5: ISO/IEC 9798-2 two-pass unilateral authentication protocol

The third protocol (3.6) is constructed from two instances of Protocol 3.4. It provides mutual entity authentication between A and B.

1. $A \rightarrow B : \{T_A, B\}_{K_{AB}}$
2. $B \rightarrow A : \{T_B, A\}_{K_{AB}}$

Protocol 3.6: ISO/IEC 9798-2 two-pass mutual authentication protocol

The fourth protocol (3.7), like Protocol 3.6, is aimed at providing mutual authentication but uses nonces instead of timestamps.

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : \{N_A, N_B, B\}_{K_{AB}}$
3. $B \rightarrow A : \{N_B, N_A\}_{K_{AB}}$

Protocol 3.7: ISO/IEC 9798-2 three-pass mutual authentication protocol

Notice that Protocol 3.7 is not simply a combination of two instances of the nonce-based unilateral authentication protocol (3.5) in which the number of messages has been reduced from four to three. Instead, the second message includes both nonces which binds them together. This design may be chosen because the standard implicitly regards the matching conversation goal as important for security.

In all of the above four protocols, the inclusion of the identity of B in the encrypted message from A is optional. Furthermore, in Protocol 3.6, the inclusion of the identity of A in the encrypted message from B is also optional. The standard comments that these fields are included to prevent reflection attacks. Their inclusion is made optional so that the protocols can be 'optimised' for networking environments where such attacks are precluded by other means.

3.2.4 Woo–Lam Authentication Protocol

All the entity authentication protocols we have looked at so far assume a shared key already exists between the two principals involved. Woo and Lam [329] devised several protocols for authentication. One of these was a unilateral

Petitioner Samsung Ex-1036, 0101

authentication protocol using a trusted server as a *key translation centre* which will convert messages encrypted with one key that it knows into messages encrypted with a different key. Protocol 3.8 shows the message flows.

1. $A \rightarrow B : A$
2. $B \rightarrow A : N_B$
3. $A \rightarrow B : \{N_B\}_{K_{AS}}$
4. $B \rightarrow S : \{A, \{N_B\}_{K_{AS}}\}_{K_{BS}}$
5. $S \rightarrow B : \{N_B\}_{K_{BS}}$

Protocol 3.8: Woo–Lam unilateral authentication protocol

The idea is that B chooses his nonce N_B and challenges A to encrypt it with K_{AS}. On receipt of the purported encryption, B asks S to translate it into an encryption with K_{BS} which B can decrypt and check. There are several attacks known on Protocol 3.8, the first of which was found by Abadi (as attributed by Woo and Lam [330]).

As shown in Attack 3.3, the adversary I starts two runs with B, in one of which I claims to be A. The two protocol runs continue in parallel but I simply sends a random value R when asked to respond to the challenge intended for A. Furthermore, I uses the challenge intended for A in the encryption for the legitimate run. The server S can only successfully translate the properly encrypted ciphertext but the returned value is correct only for the run where I masquerades as A. The result is that B accepts the run in which I is masquerading as A and rejects the other run.

1. $I_A \rightarrow B : A$
1′. $I \rightarrow B : I$
2. $B \rightarrow I_A : N_B$
2′. $B \rightarrow I : N'_B$
3. $I_A \rightarrow B : R$
3′. $I \rightarrow B : \{N_B\}_{K_{IS}}$
4. $B \rightarrow S : \{A, R\}_{K_{BS}}$
4′. $B \rightarrow S : \{I, \{N_B\}_{K_{IS}}\}_{K_{BS}}$
5. $S \rightarrow B : \{N_B\}_{K_{BS}}$

Attack 3.3: Abadi's attack on Protocol 3.8

Woo and Lam [330] consider a number of variants of Protocol 3.8 in an effort to identify the precise source of the problem. Clark and Jacob [95] showed that most of these are still vulnerable to typing attacks. However,

Petitioner Samsung Ex-1036, 0102

such attacks are prevented if principals can detect replay of messages they have created, which is an assumption of Woo and Lam.

3.2.5 Comparison of Entity Authentication Protocols

Table 3.2 summarises major properties of the seven entity authentication protocols described earlier. For the goals, an entry (*) indicates that the goal is claimed for the protocol but fails due to attack. In the second column, we indicate whether the extensional definition of entity authentication is achieved. From the table, we see that all the protocols meet this definition. In many protocols, knowledge of the peer entity is conveyed implicitly in the authentication message. For example, in the Bellare–Rogaway MAP1 protocol even though the identity of B is not included in the authentication field of message 3, it is possible to infer it through the use of K_{AB}. In the final column, we indicate whether specific attacks have been proposed in the literature. As discussed in Sect. 3.2.1, the attack on the Bird *et al.* canonical protocol fails to violate the extensional goal. The chosen protocol attack on the Bellare–Rogaway MAP1 protocol does violate the extensional goal, but the attack is aimed at showing a limitation of security proofs in practice rather than showing a weakness of the protocol.

Table 3.2. Summary of major properties of specific entity authentication protocols

Properties → ↓ *Protocol*	Liveness	Entity authentication	Security proof?	Attack
Bird *et al.* canonical 1 (3.1)	$A+B$	$A+B$	No	Yes
Bellare–Rogaway MAP1 (3.2)	$A+B$	$A+B$	Yes	Yes
9798-2 one-pass unilateral (3.4)	B	B	No	No
9798-2 two-pass unilateral (3.5)	B	B	No	No
9798-2 two-pass mutual (3.6)	$A+B$	$A+B$	No	No
9798-2 three-pass mutual (3.7)	$A+B$	$A+B$	No	No
Woo–Lam (3.8)	B (*)	B (*)	No	Yes

3.3 Server-Less Key Establishment

This section discusses protocols that allow keys to be established directly between two users without the use of a server. The protocols considered require that the two users already share a long-term secret key and may either require that one user generate the established key (key transport) or both users contribute part of the established key (key agreement).

Petitioner Samsung Ex-1036, 0103

Table 3.3 gives additional notation used in this section. In the remainder of this section, we examine server-less key transport protocols followed by server-less key agreement protocols.

Table 3.3. Additional notation used for server-less protocols

K_{AB}	The long-term key initially shared by A and B
K'_{AB}	The value of the new session key

3.3.1 Andrew Secure RPC Protocol

Although more than 10 years old, the Andrew secure RPC protocol [289], shown in Protocol 3.9, is still a widely used example in the literature. The protocol has two rather independent components. In the first three messages, A and B perform a handshake using a key they already share, K_{AB}. In the final message, B sends a new session key K'_{AB} to A. Nonce N_A is chosen by A and nonces N_B, N'_B are chosen by B.

1. $A \rightarrow B : \{N_A\}_{K_{AB}}$
2. $B \rightarrow A : \{N_A + 1, N_B\}_{K_{AB}}$
3. $A \rightarrow B : \{N_B + 1\}_{K_{AB}}$
4. $B \rightarrow A : \{K'_{AB}, N'_B\}_{K_{AB}}$

Protocol 3.9: Andrew secure RPC protocol

Burrows *et al.* [76] have pointed out a major problem with Protocol 3.9: A has no assurance that K'_{AB} is fresh. An intruder could substitute a previously recorded message 4 (from B to A) and force A to accept an old, possibly compromised, session key. Another problem was pointed out by Clark and Jacob [94]. They proposed Attack 3.4, a typing attack in which an intruder records message 2 and substitutes it in place of message 4.

1. $A \rightarrow B : \{N_A\}_{K_{AB}}$
2. $B \rightarrow A : \{N_A + 1, N_B\}_{K_{AB}}$
3. $A \rightarrow B : \{N_B + 1\}_{K_{AB}}$
4. $I_B \rightarrow A : \{N_A + 1, N_B\}_{K_{AB}}$

Attack 3.4: Clark–Jacob attack on Andrew protocol

Petitioner Samsung Ex-1036, 0104

The result of the attack is that A accepts the value $N_A + 1$ as a session key with B. Clark and Jacob point out that the potential damage of Attack 3.4 depends on what property the nonce N_A is assumed to have. If N_A is a predictable nonce such as a counter value, then an attacker could force A into accepting a bogus quantity as a session key, whose value could be known to the attacker. If N_A were random, however, then the potential damage of the attack is not so immediate since there is no release of the session key.

It is interesting to consider a revised version of the protocol suggested by Burrows *et al.* shown as Protocol 3.10. Their idea was to change the treatment of the nonces used in the protocol. The nonce N_A need not be secret; when sent by A in plaintext it still forms a typical usage of the challenge–response mechanism. The nonce N_B could be omitted altogether. Instead of sending N_B, B could send a key K'_{AB} along with A's nonce in message 2. Further differences can be found in the last two messages of Protocol 3.10. In the second last message, the encryption with K'_{AB} is intended to assure B that A knows the key. In the last message, B sends N'_B in plaintext since its purpose is not connected with protocol, but with the subsequent communications session.

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : \{N_A, K'_{AB}\}_{K_{AB}}$
3. $A \rightarrow B : \{N_A\}_{K'_{AB}}$
4. $B \rightarrow A : N'_B$

Protocol 3.10: Revised Andrew protocol of Burrows *et al.*

Lowe [212] has published Attack 3.5 on Protocol 3.10. An intruder I engages in two protocol runs with A while masquerading as B. In one of these runs I is the initiator of the protocol, while in the other A is induced to act as initiator.

1. $A \rightarrow I_B : A, N_A$
1′. $I_B \rightarrow A : B, N_A$
2′. $A \rightarrow I_B : \{N_A, K'_{AB}\}_{K_{AB}}$
2. $I_B \rightarrow A : \{N_A, K'_{AB}\}_{K_{AB}}$
3. $A \rightarrow I_B : \{N_A\}_{K'_{AB}}$
3′. $I_B \rightarrow A : \{N_A\}_{K'_{AB}}$
4. $I_B \rightarrow A : N_I$
4′. $A \rightarrow I_B : N'_A$

Attack 3.5: Lowe's attack on revised Andrew protocol

Petitioner Samsung Ex-1036, 0105

The result of the attack is that A has completed two successful runs of the protocol, apparently with B, although B has not engaged in the protocol. More importantly, the attack defeats goals from entity authentication rather than from key establishment. The attack is valid if either entity authentication or key confirmation were protocol goals.

3.3.2 Janson–Tsudik 2PKDP Protocol

Janson and Tsudik [169] proposed Protocol 3.11 which extends a two-party authentication protocol of Bird *et al.* [41] to provide key establishment. One distinctive aspect of 2PKDP is that it employs two separate cryptographic algorithms: one algorithm that provides confidentiality and another algorithm that provides authentication. The algorithm used for confidentiality purposes is bitwise exclusive-or, while that for authentication is a MAC algorithm. The protocol design allows an encryption-based (CBC-MAC) or hash-based MAC algorithm to be used. In the first message, A sends her nonce N_A to B. In the second message, B generates a new session key K'_{AB} and computes two values, $AUTH$ and $MASK$, using K_{AB}.

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : AUTH, MASK \oplus K'_{AB}$
3. $A \rightarrow B : [N_A, K'_{AB}, A]_{K_{AB}}$

Protocol 3.11: Janson–Tsudik 2PKDP protocol

The quantities $AUTH$ and $MASK$ are defined as follows:

$$AUTH = [N_A, K'_{AB}, B]_{K_{AB}}$$
$$MASK = [\![AUTH]\!]_{K_{AB}}.$$

The second part of message 2 can be viewed as analogous to an encryption of K'_{AB} using a one-time pad. Upon receiving message 2, A computes a MAC under K_{AB} of the received $AUTH$ value to allow decryption of the session key from the second part of message 2, then verifies that the first part of message 2 agrees with the MAC under K_{AB} of the nonce sent earlier, received session key, and B's identity. Verification of the first part of message 2 implies key freshness as well as key integrity. Upon receiving message 3, B verifies that the received value is correct, which implies key confirmation of A to B.

3.3.3 Boyd Two-Pass Protocol

Protocol 3.12 by Boyd [56] allows both A and B to contribute part of the established key. The messages sent are simply the random numbers chosen by

Petitioner Samsung Ex-1036, 0106

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : N_B$

Protocol 3.12: Boyd two-pass protocol

A and B. The new key is $K'_{AB} = f(N_A, N_B, K_{AB})$, where f is a combining function such that it must be infeasible to find $f(.,.,K_{AB})$ without knowledge of K_{AB}, even after repeated use. This property is necessary to ensure key authentication (that is, the secrecy of K'_{AB}). Another property required of f is that it is one-way in the first two inputs. This property is necessary to ensure key freshness. For a concrete example, any practical secure MAC algorithm may be chosen for f.

3.3.4 ISO/IEC 11770-2 Server-Less Protocols

The international standard ISO/IEC 11770 Part 2 [158] specifies 13 protocols using symmetric encryption algorithms. Six of these protocols are server-less, while the other seven rely on a trusted server. The server-based protocols are described in Sect. 3.4.4. Some of the protocols make use of a *key derivation function* $f(\cdot)$ in forming the new session key from two or more inputs. Two examples of f are given in the appendix of the standard: one is bitwise XOR of the inputs and the other is application of a hash function to the concatenation of the inputs.

The first two of the six server-less protocols each use only one message pass and provide only implicit key authentication. Mechanism 1, shown as Protocol 3.13, consists only of the encrypted timestamp of A. The new session key is derived as $K'_{AB} = f(K_{AB}, T_A)$ where f is the key derivation function.

$A \rightarrow B : \{T_A\}_{K_{AB}}$

Protocol 3.13: ISO/IEC 11770-2 Key Establishment Mechanism 1

The single message in mechanism 2, shown as Protocol 3.14, consists only of the new encrypted session key. This means that B gains no assurance on its freshness.

$A \rightarrow B : \{K'_{AB}\}_{K_{AB}}$

Protocol 3.14: ISO/IEC 11770-2 Key Establishment Mechanism 2

The other four server-less protocols are derived from each of the four two-party entity authentication protocols that were described in Sect. 3.2.3 by

Petitioner Samsung Ex-1036, 0107

adding a key (or more generally *keying material*) to each encrypted message. The next four protocols, Protocols 3.15 to 3.18, may be compared with the four entity authentication protocols, Protocols 3.4 to 3.7. The entity authentication properties of each corresponding pair are the same.

Mechanism 3, shown as Protocol 3.15, consists of a single message from a claimant A to a verifier B. The new session key, K'_{AB}, is chosen by A and encrypted for B. Both A and B achieve the good key property, but only B achieves key confirmation.

$A \rightarrow B : \{T_A, B, K'_{AB}\}_{K_{AB}}$

Protocol 3.15: ISO/IEC 11770-2 Key Establishment Mechanism 3

Mechanism 4, shown as Protocol 3.16, uses a nonce instead of a timestamp. The properties achieved are the same as for mechanism 3.

---

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : \{N_B, B, K'_{AB}\}_{K_{AB}}$

Protocol 3.16: ISO/IEC 11770-2 Key Establishment Mechanism 4

Mechanism 5, shown as Protocol 3.17, is constructed from two instances of Protocol 3.15. Now both A and B choose keying material, F_{AB} and F_{BA} respectively. (Optionally either one of these may be omitted.) The session key is derived as $K'_{AB} = f(F_{AB}, F_{BA})$ where f is the key derivation function. Both A and B obtain the good key property, but if F_{BA} is included in the session key derivation only A will achieve key confirmation.

---

1. $A \rightarrow B : \{T_A, B, F_{AB}\}_{K_{AB}}$
2. $B \rightarrow A : \{T_B, A, F_{BA}\}_{K_{AB}}$

Protocol 3.17: ISO/IEC 11770-2 Key Establishment Mechanism 5

Mechanism 6, shown as Protocol 3.18, is similar to Protocol 3.17. Keying material is provided from both parties and the session key is calculated in the same way. A major difference is that it uses nonces instead of timestamps. The extensional properties achieved are the same as for mechanism 5. However, like Protocol 3.7, inclusion of both nonces in messages 2 and 3 binds the protocol messages together. In contrast, an adversary could interleave two runs of Protocol 3.17 so that A and B do not see matching conversations.

Petitioner Samsung Ex-1036, 0108

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : \{N_A, N_B, B, F_{AB}\}_{K_{AB}}$
3. $B \rightarrow A : \{N_B, N_A, F_{BA}\}_{K_{AB}}$

Protocol 3.18: ISO/IEC 11770-2 Key Establishment Mechanism 6

As with the authentication protocols in Sect. 3.2.3, in all of the above four protocols, the inclusion of the identity of B in the encrypted message from A is optional. Furthermore, in Protocol 3.17, the inclusion of the identity of A in the encrypted message from B is also optional.

3.3.5 Comparison of Server-Less Protocols

Table 3.4 summarises major properties of the 10 server-less protocols described earlier. The last five columns summarise the goals that these protocols meet. For these goals, an entry (*) indicates that the goal is claimed for the protocol but fails due to an attack. From the table, we see that all the protocols except the Andrew secure RPC protocol and ISO/IEC 11770-2 Mechanism 2 meet the good key goal. Further, the revised version of the Andrew protocol suggested by Burrows *et al.* achieves the good key goal but fails to meet the key confirmation goal.

Table 3.4. Summary of major properties of specific server-less protocols

Properties → ↓ *Protocol*	No. of passes	Key control	Key freshness	Key auth.	Key conf.	Attack
Andrew (3.9)	4	B	No (*)	Yes	No	Yes
BAN–Andrew (3.10)	4	B	Yes	Yes	No (*)	Yes
Janson–Tsudik (3.11)	3	B	Yes	Yes	Yes	No
Boyd (3.12)	2	A/B	Yes	Yes	No	No
11770-2 Mech. 1 (3.13)	1	A	Yes	Yes	B	No
11770-2 Mech. 2 (3.14)	1	A	No	Yes	No	No
11770-2 Mech. 3 (3.15)	1	A	Yes	Yes	B	No
11770-2 Mech. 4 (3.16)	2	B	Yes	Yes	B	No
11770-2 Mech. 5 (3.17)	2	A/B	Yes	Yes	A	No
11770-2 Mech. 6 (3.18)	3	A/B	Yes	Yes	A	No

Table 3.4 indicates that key confirmation may be obtained by one principal in most of the ISO mechanisms. This is because when a key is received from

Petitioner Samsung Ex-1036, 0109

the partner, the recipient knows that the sender is in possession of the key. It is interesting to note that the ISO/IEC 11770-2 standard [158] indicates that none of these protocols provides key confirmation. This may be because mutual key confirmation is expected. The standard indicates that key confirmation may be achieved by sending a time varying parameter encrypted with the session key. Note that mechanism 1 allows B to know that A sent the timestamp (assuming that reflection attacks are prevented) and so B has assurance that A has the ability to calculate the session key.

3.4 Server-Based Key Establishment

There exist numerous examples of server-based protocols in the literature. Most of these are key transport or key agreement protocols in which the server, or one or both users, have the responsibility for key generation. Hybrid protocols in which all three of them share the responsibility for key generation are less common and not as well known as the other two classes of protocols.

An important consideration in the design of server-based key transport protocols is who generates the session key. Many protocol designers implicitly assume that the users are not capable of generating good quality session keys, leaving this task for the server. However, this dependence is not always necessary, as may be seen in published protocols where users, not the server, choose a session key.

Table 3.5 gives additional notation used in this section.

Table 3.5. Additional notation used for server-based protocols

A and B	Two users wishing to establish a session key
S	The server
K_{AS}, K_{BS}	Long-term keys initially shared by A and S, and by B and S
K_{AB}	Session key to be shared by A and B

3.4.1 Needham–Schroeder Shared Key Protocol

The famous protocol proposed by Needham and Schroeder [249] in 1978 is shown as Protocol 3.19. As discussed in Chap. 1, this protocol achieves the good key property with respect to A but not B. The second message encrypted with A's shared key K_{AS} includes both A's nonce and B's identity, assuring A of session key freshness and key authentication, respectively.

For B the situation is slightly different: he decrypts the encrypted message relayed by A to learn the session key value and then carries out a nonce handshake with A to be sure that the message is not a replay. However, the

Petitioner Samsung Ex-1036, 0110

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $A \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$
4. $B \rightarrow A : \{N_B\}_{K_{AB}}$
5. $A \rightarrow B : \{N_B - 1\}_{K_{AB}}$

Protocol 3.19: Needham–Schroeder shared key protocol

handshake can be easily subverted since an adversary can be expected to know the value of an old session key. This weakness of the protocol was originally pointed out by Denning and Sacco [99]. In their attack an intruder uses a compromised session key to masquerade as A to B. To overcome the attack, Denning and Sacco suggested Protocol 3.20 as a solution, using timestamps to allow verification of key freshness.

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{B, K_{AB}, T_S, \{A, K_{AB}, T_S\}_{K_{BS}}\}_{K_{AS}}$
3. $A \rightarrow B : \{A, K_{AB}, T_S\}_{K_{BS}}$

Protocol 3.20: Denning–Sacco protocol

Bauer *et al.* [24] discuss the vulnerability of the Needham–Schroeder protocol to the compromise of A's long-term key: an intruder who learns the long-term key of A can impersonate A (as in the Denning–Sacco attack) even after the compromise is detected and the long-term key replaced. They suggested a solution without using timestamps shown as Protocol 3.21. The protocol they proposed is essentially symmetric with respect to A and B: each of them sends a nonce to S in plaintext, and S returns the nonces in separate messages for A and B.

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow S : A, N_A, B, N_B$
3. $S \rightarrow B : \{K_{AB}, A, N_B\}_{K_{BS}}, \{K_{AB}, B, N_A\}_{K_{AS}}$
4. $B \rightarrow A : \{K_{AB}, B, N_A\}_{K_{AS}}$

Protocol 3.21: Bauer–Berson–Feiertag protocol

3.4.2 Otway–Rees Protocol

The Otway–Rees protocol [259], like Protocol 3.21, provides symmetric assurances of freshness. The message flows are shown as Protocol 3.22, where M is

Petitioner Samsung Ex-1036, 0111

a second nonce generated by A. This protocol is susceptible to a typing attack, as described in Sect. 1.6.7. The attack described there is due to Boyd [54], subsequently rediscovered by Clark and Jacob [94].

1. $A \rightarrow B : M, A, B, \{N_A, M, A, B\}_{K_{AS}}$
2. $B \rightarrow S : M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}}$
3. $S \rightarrow B : M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
4. $B \rightarrow A : M, \{N_A, K_{AB}\}_{K_{AS}}$

Protocol 3.22: Otway–Rees protocol

Consider exactly what actions are required of S upon receiving message 2 in the protocol. There are essentially two possibilities:

A1 S checks that the values obtained by decrypting the fields M, A and B in the two encrypted parts match.

A2 S checks that the plaintext versions of (M, A, B) match the values obtained by decrypting the fields M, A and B in the two encrypted parts.

An attack of Boyd and Mao [60] shows that it is essential that the plaintext versions of M, A and B are checked. Without this, the session key can turn out to be shared between A and an intruder, allowing the intruder to masquerade as B as shown in Attack 3.6. Suppose A starts a protocol run with B. An intruder C who wishes to impersonate B chooses a nonce N_C and substitutes the following message in place of the original one.

$2'. C_B \rightarrow S : M, A, C, \{N_A, M, A, B\}_{K_{AS}}, \{N_C, M, A, B\}_{K_{CS}}$

Attack 3.6: Attack on Otway–Rees protocol without plaintext checking

If we assume that S does only the checking specified by (A1), the message will be found correct by S and so S will encrypt K_{AB} with the key K_{CS} in message $3'$. It is clear that this attack violates the goal of implicit key authentication: A believes the key is shared with B, whereas in fact it is shared with C. Note that the attack is easily prevented if S does the checking specified by (A2). Whether this is a valid attack on the protocol depends on the action taken by S, something (unfortunately) not clearly specified in the protocol description.

In Protocol 3.22 the encrypted message received by a user from the server does not include a concrete field for the identity of the other user to whom S intends to make the key known. What indication, then, does a user have about who else the session key is shared with? The answer to this question can be found by examining a simplified version of the protocol due to Burrows

Petitioner Samsung Ex-1036, 0112

et al. shown as Protocol 3.23. In the simplified version of the protocol, the nonce N_B is sent unencrypted in message 2.

1. $A \rightarrow B : M, A, B, \{N_A, M, A, B\}_{K_{AS}}$
2. $B \rightarrow S : M, A, B, \{N_A, M, A, B\}_{K_{AS}}, N_B, \{M, A, B\}_{K_{BS}}$
3. $S \rightarrow B : M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
4. $B \rightarrow A : M, \{N_A, K_{AB}\}_{K_{AS}}$

Protocol 3.23: Otway–Rees protocol modified by Burrows *et al.*

Protocol 3.23 was shown to be flawed by Boyd and Mao. Consider an attacker C who has obtained the encrypted message $\{M, C, B\}_{K_{BS}}$ by engaging in a previous legitimate run of the protocol with B. To attack the protocol, C starts a run with B and masquerades as A by capturing message 2 and modifying it, replacing cleartext identifier A by C and $\{M, A, B\}_{K_{BS}}$ by $\{M, C, B\}_{K_{BS}}$. The attacking run proceeds as shown in Attack 3.7.

1. $C_A \rightarrow B : M, A, B, \{N_C, M, C, B\}_{K_{CS}}$
2. $B \rightarrow C_S : M, A, B, \{N_C, M, C, B\}_{K_{CS}}, N_B, \{M, A, B\}_{K_{BS}}$
2′. $C_B \rightarrow S : M, C, B, \{N_C, M, C, B\}_{K_{CS}}, N_B, \{M, C, B\}_{K_{BS}}$
3. $S \rightarrow B : \quad M, \{N_C, K_{CB}\}_{K_{CS}}, \{N_B, K_{CB}\}_{K_{BS}}$
4. $B \rightarrow C_A : M, \{N_C, K_{CB}\}_{K_{CS}}$

Attack 3.7: Attack on Otway–Rees protocol modified by Burrows *et al.*

The result of the attack is that B believes that the session key is shared with A, whereas in fact it is shared with C. Notice that the above attack is not applicable to the original Otway–Rees protocol where the nonce N_B is cryptographically bound to the identity A by encryption in message 2. As a result of this binding, B can rely on the nonce N_B in message 3 to infer that the key is shared with A. The attack is possible on the simplified protocol because it removes the binding. The treatment of nonces in the Otway–Rees protocol was discussed by van Oorschot [323], subsequently by Abadi and Needham [4]. The latter authors suggested an alternative protocol where the nonce N_B (as well as the nonce N_A) is sent unencrypted in message 2 and yet the above attack is prevented. The main idea is that in Protocol 3.24 the encrypted message received by each user includes the identity of the other user to whom S makes the session key known.

This idea was first discussed in a preliminary draft of Boyd and Mao's paper [59], subsequently by van Oorschot [323]. With this modification, it is easy to check that the previous attack no longer works. Note that Protocol 3.24 is similar in structure and goal to the protocol of Bauer *et al.* (3.21). Like the

Petitioner Samsung Ex-1036, 0113

1. $A \rightarrow B : A, B, N_A$
2. $B \rightarrow S : A, B, N_A, N_B$
3. $S \rightarrow B : \{N_A, A, B, K_{AB}\}_{K_{AS}}, \{N_B, A, B, K_{AB}\}_{K_{BS}}$
4. $B \rightarrow A : \{N_A, A, B, K_{AB}\}_{K_{AS}}$

Protocol 3.24: Otway–Rees protocol modified by Abadi and Needham

original Otway–Rees protocol it provides key authentication and key freshness assurances, although it differs with respect to goals for entity authentication. In Protocol 3.22, A has assurance that B is alive: when A receives message 4, she knows that B must have sent message 2 recently. In Protocol 3.24, A does not achieve liveness of B.

3.4.3 Kerberos Protocol

The Kerberos software system was developed at MIT to protect the network services provided as part of project Athena. It is one of the *de facto* standards for authentication on computer networks. Kerberos uses as its building block a key establishment protocol based on the Needham–Schroeder protocol but with timestamps instead of challenge–response, following Denning and Sacco's suggestion. A good overview of the current version of the Kerberos protocol, known as Version 5, can be found in a paper by Neuman and Ts'o [251]. The Version 5 protocol has evolved from the Version 4 protocol; it addresses several shortcomings of the Version 4 protocol including some potential security weaknesses.

Differences between Kerberos Versions 4 and 5 are described by Kohl *et al.* [189]. One weakness of the Version 4 protocol is that the encryption method used does not provide adequate integrity protection for encrypted messages, even though the protocol was specifically designed with this requirement in mind. Version 4 makes use of a non-standard mode of DES known as *Plaintext Cipher-Block Chaining* (PCBC) mode with the property that errors in the decrypted ciphertext propagate to all successive blocks of plaintext. However, as pointed out by Kohl [187], PCBC encryption is susceptible to a block-swapping attack which allows a partially garbled message to be accepted by the receiver. Kerberos Version 5 uses standard CBC encryption and embeds a checksum in the message before encryption for providing sufficient integrity protection.

The basic Kerberos protocol involves three parties: a client which desires to use some service, an application server which provides a service, and an authentication server (AS) which is contacted by the client before attempting to access the application server. We will use the notations A, B and S below to denote the Kerberos terms client, application server and AS, respectively. The client and the server do not initially share a key between themselves, but they do share a key with S. For the sake of clarity, Protocol 3.25 shows only

Petitioner Samsung Ex-1036, 0114

those message fields that are critical to security; details of other message fields can be found in RFC 1510 [188].

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : \{K_{AB}, B, L, N_A, \ldots\}_{K_{AS}}, \{K_{AB}, A, L, \ldots\}_{K_{BS}}$
3. $A \rightarrow B : \{A, T_A\}_{K_{AB}}, \{K_{AB}, A, L, \ldots\}_{K_{BS}}$

Protocol 3.25: Basic Kerberos protocol

In the Kerberos papers, the protocol message fields that are encrypted with K_{BS} and K_{AB} are referred to as *ticket* and *authenticator*, respectively. Among other data, the ticket contains the session key generated by S that will be used by the client and server, the client's identity, and an expiration time L after which the session key is no longer valid. The authenticator contains the client's identity and a timestamp from the client's clock. If the timestamp is successfully verified by the application server, then the server obtains assurance that the client possesses the session key contained in the ticket.

The basic Kerberos protocol allows an optional fourth message shown in Protocol 3.26, in which the server returns the client's timestamp along with other optional information, all encrypted using the session key.

4. $B \rightarrow A : \{T_A, \ldots\}_{K_{AB}}$

Protocol 3.26: Optional Kerberos message to complete mutual authentication

The Kerberos protocol has key establishment as well as entity authentication as its goal. From A's viewpoint the protocol provides the good key property since A can be confident that the key is fresh and known only to herself and B. From B's viewpoint the protocol provides only key authentication since the ticket does not contain any information from which B can be confident that the key is fresh. However, the freshness of the key is judged by a different measure. To ensure key freshness from B's viewpoint, the protocol relies on the expiration time contained in the ticket, rather than sending with the key a quantity known to be new. As long as the ticket has not expired, B can still be confident that the session key is safe, even if it was used in a previous session with A. The use of a ticket without an absolute freshness indicator has a useful aspect. It makes it possible for a client to cache the ticket from the server so that a session key may be *re-established* directly with B without the intervention of S. (Protocols that use tickets to re-establish session keys are often known as repeated authentication protocols in the literature.)

As for the authentication properties achieved by the protocol, the combination of authenticator and ticket in the third message provides strong entity

Petitioner Samsung Ex-1036, 0115

authentication of A to B. The fourth message, if present, provides entity authentication of B to A.

3.4.4 ISO/IEC 11770-2 Server-Based Protocols

The server-less protocols in the international standard ISO/IEC 11770 Part 2 [158] were described in Sect. 3.3.4. Here we discuss the seven server-based protocols in that standard. In four of these the server chooses the session key and acts as a key distribution centre. In the other three the session key is chosen by either A or B and the server acts as a *key translation centre* to make that key available to the other party.

The first three of the protocols using a key distribution centre are very similar to protocols we have already examined, therefore we do not look at the details.

Key Establishment Mechanism 7 The server simply encrypts and sends K_{AB} to both parties including the identity of the peer entity. This is similar to the Bauer-Berson-Feiertag protocol (3.21) but the lack of nonces means that neither A nor B gains key freshness.

Key Establishment Mechanism 8 Principal A sends a nonce to S and the key is returned encrypted for A together with A's nonce and the identity of B. In addition the key is encrypted for B together with a timestamp or counter and the identity of principal A. This mechanism is essentially identical to the basic Kerberos protocol (3.25).

Key Establishment Mechanism 9 Both principals send their nonces to S and they are returned with the encrypted key and the identity of the peer entity. This protocol is identical to Protocol 3.21.

In Key Establishment Mechanism 10, shown in Protocol 3.27, principal A first sends an encrypted request message to S which checks its authenticity (including checking the freshness of the timestamp or sequence number). This provides the rather unusual feature that only authentic parties are able to request new keys. S replies by sending the key to both principals after encrypting it together with the identity of the peer entity and a timestamp or counter.

1. $A \rightarrow S : \{T_A, B\}_{K_{AS}}$
2. $S \rightarrow A : \{T_S, K_{AB}, B\}_{K_{AS}}$
3. $S \rightarrow B : \{T'_S, K_{AB}, A\}_{K_{BS}}$

Protocol 3.27: ISO/IEC 11770-2 Key Establishment Mechanism 10

The first of the protocols that uses S as a key translation centre, called Key Establishment Mechanism 11, does not provide key freshness. Mechanism

Petitioner Samsung Ex-1036, 0116

12, shown as Protocol 3.28, is similar but now adds a nonce for A to check that the key was received by S and a timestamp for B to check freshness. The session key, K_{AB}, is chosen by A.

1. $A \rightarrow S : \{N_A, B, K_{AB}\}_{K_{AS}}$
2. $S \rightarrow A : \{N_A, B\}_{K_{AS}}, \{T_S, K_{AB}, A\}_{K_{BS}}$
3. $A \rightarrow B : \{T_S, K_{AB}, A\}_{K_{BS}}$

Protocol 3.28: ISO/IEC 11770-2 Key Establishment Mechanism 12

An optional handshake for entity authentication and key confirmation is also specified. It is interesting to compare this protocol with the Wide-mouthed-frog protocol (3.30) below. Although there are distinct similarities, the asymmetry in Protocol 3.28 prevents the reflection attack described on Protocol 3.30.

The final protocol in ISO/IEC 11770-2, Key Establishment Mechanism 13, is shown as Protocol 3.29. This time both parties send their nonces to S, while K_{AB} is chosen by B.

1. $A \rightarrow B : N_A$
2. $B \rightarrow S : \{N_B, N_A, A, K_{AB}\}_{K_{BS}}$
3. $S \rightarrow B : \{N_B, A\}_{K_{BS}}, \{N_A, K_{AB}, B\}_{K_{AS}}$
4. $B \rightarrow A : \{N_A, K_{AB}, B\}_{K_{AS}}$

Protocol 3.29: ISO/IEC 11770-2 Key Establishment Mechanism 13

3.4.5 Wide-Mouthed-Frog Protocol

Numerous server-based key transport protocols assume that users trust only a server to choose the key for a session. The Wide-mouthed-frog protocol, due to Burrows [76], is intended for environments where one user trusts the other to choose the key. The server simply makes the key chosen by one user available to the other. The message flows are shown in Protocol 3.30, where T_A and T_S denote timestamps from the local clocks of A and S, respectively.

1. $A \rightarrow S : A, \{T_A, B, K_{AB}\}_{K_{AS}}$
2. $S \rightarrow B : \{T_S, A, K_{AB}\}_{K_{BS}}$

Protocol 3.30: Wide-mouthed-frog protocol

Petitioner Samsung Ex-1036, 0117

The intention is that the timestamps in messages 1 and 2 provide freshness assurances to S and B, respectively. However, this may not provide adequate protection as shown by Attack 3.8. Suppose I is an intruder who has recorded one run of the protocol. I replays the message sent out by S in the first run back to S to start a second run of the protocol while masquerading as B. This would cause S to send $\{T'_S, B, K_{AB}\}_{K_{AS}}$, where T'_S is a new timestamp. Again I replays this message to S to start a third run of the protocol, this time masquerading as A. The intruder continues to execute new runs of the protocol, long after the session key is discarded. Therefore I can force B to re-accept the key again simply by allowing him to receive a sufficiently up-to-date message from S containing the key.

$1'.\ I_B \rightarrow S : B, \{T_S, A, K_{AB}\}_{K_{BS}}$
$2'.\ S \rightarrow I_A : \{T'_S, B, K_{AB}\}_{K_{AS}}$
$1''.\ I_A \rightarrow S : A, \{T'_S, B, K_{AB}\}_{K_{AS}}$
$2''.\ S \rightarrow B : \{T''_S, A, K_{AB}\}_{K_{BS}}$

Attack 3.8: Attack on Wide-mouthed-frog protocol

Attack 3.8 is discussed by Anderson and Needham [14] and by Clark and Jacob [94]. However, it should be mentioned that in the BAN logic paper [76], where Protocol 3.30 first appeared, it is assumed that each principal will recognise and reject their own messages. This will prevent these attacks.

3.4.6 Yahalom Protocol

The Yahalom protocol first appeared in the BAN logic paper [76]. It is frequently used as a benchmark protocol by researchers using formal methods for protocol verification (for example, see Paulson [266]). One reason for this may be that it has a rather unusual structure. More importantly, it is subject to some subtle attacks, which make it a challenging subject for testing a new technique.

---

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow S : B, \{A, N_A, N_B\}_{K_{BS}}$
3. $S \rightarrow A : \{B, K_{AB}, N_A, N_B\}_{K_{AS}}, \{A, K_{AB}\}_{K_{BS}}$
4. $A \rightarrow B : \{A, K_{AB}\}_{K_{BS}}, \{N_B\}_{K_{AB}}$

Protocol 3.31: Yahalom protocol

Protocol 3.31 provides key authentication to both A and B. Key freshness is more problematic because the positions of A and B differ. Note that A gains

Petitioner Samsung Ex-1036, 0118

key freshness from the first part of message 3. A knows that this part is not a replay from an old protocol run since she knows the message was sent recently. The same cannot be said of the message received by B. In message 4, B has no direct indication of the freshness of the key from the server but infers that the key K_{AB} was used recently by A. Since the server makes N_B available only to the party requested by B, B can be assured that the encryption with K_{AB} must have been formed by A recently. If A acts properly in relaying the message that was encrypted for B by S as part of the current run, then B can be assured of session key freshness. If A misbehaves by relaying a similar encrypted message she has from an old run, then B cannot determine if the key is fresh. It seems clear that both users gain liveness of the other. A knows that B is active as verification of the first part of message 3 implies B sent message 2 recently. Because of the encryption of N_B with K_{AB} in message 4, B is assured that A really knows K_{AB}. Thus it appears that B gains key confirmation and liveness from message 4.

Burrows *et al.* suggested a modified form of the Yahalom protocol, shown in Protocol 3.32. The nonce N_B is sent unencrypted by B in message 2 and is returned by S in the message part encrypted with K_{BS}. The modified form constitutes a typical usage of the challenge–response mechanism.

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow S : B, N_B, \{A, N_A\}_{K_{BS}}$
3. $S \rightarrow A : N_B, \{B, K_{AB}, N_A\}_{K_{AS}}, \{A, K_{AB}, N_B\}_{K_{BS}}$
4. $A \rightarrow B : \{A, K_{AB}, N_B\}_{K_{BS}}, \{N_B\}_{K_{AB}}$

Protocol 3.32: Yahalom protocol modified by Burrows *et al.*

Protocol 3.32 was shown to be flawed by Syverson [313] who pointed out two attacks. Suppose I is an intruder who wishes to attack the protocol. In Attack 3.9 the intruder I starts a protocol run with B while masquerading as A. After receiving the second message, I starts another run with B by sending an arbitrary value N_I concatenated with N_B from the first run. The intruder I is then able to play the role of S in message 4 of the first run by replaying the encrypted component sent out by B in the second run back to B.

Attack 3.9 is an example of a typing attack. The result of the attack is that B will interpret the value N_I as a session key with A. The attack relies on the assumption that nonces can be of arbitrary length (this is termed substituting 'doubled' nonces for nonces by Syverson).

Attack 3.10 shows Syverson's second attack on Protocol 3.32. It begins with the intruder I intercepting an initial message from A to B. On receiving this message, I initiates a new protocol run with A using N_A as challenge, while masquerading as B. The second protocol run proceeds as follows. I modifies the message from A to S, replacing the new nonce N'_A by the old

Petitioner Samsung Ex-1036, 0119

1. $I_A \rightarrow B : A, N_I$
2. $B \rightarrow I_S : B, N_B, \{A, N_I\}_{K_{BS}}$
1'. $I_A \rightarrow B : A, (N_I, N_B)$
2'. $B \rightarrow I_S : B, N'_B, \{A, N_I, N_B\}_{K_{BS}}$
3. *Omitted.*
4. $I_S \rightarrow B : \{A, N_I, N_B\}_{K_{BS}}, \{N_B\}_{N_I}$

Attack 3.9: Syverson's attack on modified Yahalom protocol

nonce N_A. I also intercepts the message from S to B and simply replays the encrypted components of this message back as encrypted components of message 3 in the first run but with the order of components switched.

1. $A \rightarrow I_B : A, N_A$
1'. $I_B \rightarrow A : B, N_A$
2'. $A \rightarrow I_S : A, N'_A, \{B, N_A\}_{K_{AS}}$
2''. $I_A \rightarrow S : A, N_A, \{B, N_A\}_{K_{AS}}$
3'. $S \rightarrow I_A : N_A, \{A, K_{AB}, N_A\}_{K_{BS}}, \{B, K_{AB}, N_A\}_{K_{AS}}$
2. *Omitted.*
3. $I_S \rightarrow A : N_I, \{B, K_{AB}, N_A\}_{K_{AS}}, \{A, K_{AB}, N_A\}_{K_{BS}}$

Attack 3.10: Alternative Syverson's attack on modified Yahalom protocol

Attack 3.10 shows that A would be wrong to conclude, after a successful run, that B is active. Note that these two attacks are different not only in structure but also in aim. The first attack violates key establishment properties whereas the second attack violates entity authentication properties.

3.4.7 Janson–Tsudik 3PKDP Protocol

The 3PKDP protocol proposed by Janson and Tsudik [169] is a server-based protocol that uses 2PKDP (discussed in Sect. 3.3.2) as a building block. This protocol has two executions of 2PKDP: firstly between A and S (messages 1 through 3), and then between B and S (messages 4 through 6). The final three messages are intended for entity authentication. In Protocol 3.33 the quantities $AUTH$ and $MASK$ are defined as follows.

$$
\begin{aligned}
AUTH_A &= [N_A, K_{AB}, B]_{K_{AS}} \\
MASK_A &= [\![AUTH_A]\!]_{K_{AS}} \\
AUTH_B &= [N_B, K_{AB}, A]_{K_{BS}} \\
MASK_B &= [\![AUTH_B]\!]_{K_{BS}}.
\end{aligned}
$$

Petitioner Samsung Ex-1036, 0120

1. $A \rightarrow S: A, B, N_A$
2. $S \rightarrow A: AUTH_A, MASK_A \oplus K_{AB}$
3. $A \rightarrow S: [N_A, K_{AB}, A]_{K_{AS}}$
4. $B \rightarrow S: B, A, N_B$
5. $S \rightarrow B: AUTH_B, MASK_B \oplus K_{AB}$
6. $B \rightarrow S: [N_B, K_{AB}, B]_{K_{BS}}$
7. $A \rightarrow B: A, N'_A$
8. $B \rightarrow A: [N'_A, N'_B, B]_{K_{AB}}, N'_B$
9. $A \rightarrow B: [N'_A, N'_B, A]_{K_{AB}}$

Protocol 3.33: Janson–Tsudik 3PKDP protocol

Protocol 3.33 achieves goals from both key establishment and entity authentication. It achieves the good key goal and the mutual authentication goal. Considering that the entity authentication goal implies the far-end operative property, it is easily seen that the protocol also provides the enhanced goals of mutual belief in key and key confirmation. One oddity of this protocol is that messages 3 and 6 do not seem to have any useful purpose. All S can hope to learn from these messages is that A and B are really out there, a property that is not crucial to the service provided by the protocol. Protocol 3.34 is a modified version proposed by Janson and Tsudik which omits the above-mentioned messages and routes all communication with S via B. It achieves the same goals as 3PKDP using five messages rather than nine messages.

1. $A \rightarrow B: A, N_A, N'_A$
2. $B \rightarrow S: A, B, N_A, N_B$
3. $S \rightarrow B: AUTH_A, MASK_A \oplus K_{AB}, AUTH_B, MASK_B \oplus K_{AB}$
4. $B \rightarrow A: AUTH_A, MASK_A \oplus K_{AB}, [N'_A, N'_B, B]_{K_{AB}}, N'_B$
5. $A \rightarrow B: [N'_A, N'_B, A]_{K_{AB}}$

Protocol 3.34: Janson–Tsudik optimised 3PKDP protocol

3.4.8 Bellare–Rogaway 3PKD Protocol

Protocol 3.35 was proposed by Bellare and Rogaway [33]. It has key establishment as its goal and is provably secure in the Bellare–Rogaway model. The 3PKD protocol uses two distinct cryptographic transformations: a symmetric encryption algorithm and a MAC. As one possibility, the encryption function can be constructed from a keyed pseudorandom function f_K. Specifically, the encryption of message m using f under a shared key K is computed as the quantity $(r, m \oplus f_K(r))$, where r is a random number.

Petitioner Samsung Ex-1036, 0121

1. $A \rightarrow B : N_A$
2. $B \rightarrow S : N_A, N_B$
3. $S \rightarrow A : [\![K_{AB}]\!]_{K_{AS}}, [A, B, N_A, [\![K_{AB}]\!]_{K_{AS}}]_{K_{AS}}$
4. $S \rightarrow B : [\![K_{AB}]\!]_{K_{BS}}, [A, B, N_B, [\![K_{AB}]\!]_{K_{BS}}]_{K_{BS}}$

Protocol 3.35: Bellare–Rogaway 3PKD protocol

Protocol 3.35 provides both A and B with assurances of key authentication and key freshness. It is not designed to provide entity authentication or key confirmation. The provably secure style definition has the property that if the session key itself is used to cryptographically protect messages within the protocol, the resulting protocol cannot be considered secure. Thus standard techniques for key confirmation, such as encrypting something using the session key, are not compatible with the security proof of the 3PKD protocol.

3.4.9 Woo–Lam Key Transport Protocol

Woo and Lam [330] proposed a protocol intended to achieve mutual entity authentication as well as key establishment. As shown in Protocol 3.36, principals A and B exchange nonces before contacting the server. This allows them to include both nonces in the encrypted messages sent to S.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : N_B$
3. $A \rightarrow B : \{A, B, N_A, N_B\}_{K_{AS}}$
4. $B \rightarrow S : \{A, B, N_A, N_B\}_{K_{AS}}, \{A, B, N_A, N_B\}_{K_{BS}}$
5. $S \rightarrow B : \{B, N_A, N_B, K_{AB}\}_{K_{AS}}, \{A, N_A, N_B, K_{AB}\}_{K_{BS}}$
6. $B \rightarrow A : \{B, N_A, N_B, K_{AB}\}_{K_{AS}}, \{N_A, N_B\}_{K_{AB}}$
7. $A \rightarrow B : \{N_B\}_{K_{AB}}$

Protocol 3.36: Woo–Lam key transport protocol

Clark and Jacob [95] found an attack on Protocol 3.36 in which a malicious principal B can force A to accept two copies of the same session key as new keys. Lowe [212] also found a typing attack which allows the adversary to masquerade as A and obtain the session key accepted by B. Even without these attacks, it is difficult to recommend Protocol 3.36 as a practical protocol in view of the large number of message flows and encryptions required in comparison with alternatives.

Petitioner Samsung Ex-1036, 0122

3.4.10 Gong Key Agreement Protocols

Gong [125] designed several protocols to illustrate lower bounds he derived on the numbers of messages and rounds in server-based protocols for key establishment. Because he was not concerned with minimising message lengths, he took a conservative approach to message formats. In the protocols in this section, messages encrypted with key K are of the format $\{Sender, Recipient, Client1, Key, Client2, Freshness\ ID\}_K$ where *Client1* and *Client2* will share *Key*, and *Freshness ID* may be a nonce or timestamp. Since some of these fields may be identical, there is some redundancy in many of the messages.

Protocol 3.37 is a key agreement protocol since the session key is derived from the information contributed by both A and B. The server makes each party's contribution available to the other but does not itself contribute any information to the session key. The encrypted portions of the protocol messages include timestamps to provide freshness guarantees.

1. $A \rightarrow S : A, B, \{A, S, A, K_1, B, T_A\}_{K_{AS}}$
2. $S \rightarrow B : \{S, B, A, K_1, B, T_S\}_{K_{BS}}$
3. $B \rightarrow S : \{B, S, B, K_2, A, T_B\}_{K_{BS}}$
4. $S \rightarrow A : \{S, A, B, K_2, A, T_S\}_{K_{AS}}$

Protocol 3.37: Gong's timestamp-based protocol

The values K_1 and K_2 are random numbers serving as contributions of A and B respectively to the session key. Both A and B compute the session key as $f(K_1, K_2)$ where f is a one-way function. This protocol provides to both A and B implicit key authentication and key freshness. The timestamps may be replaced by random nonces to provide freshness guarantees. The resulting protocol has one more message as shown in Protocol 3.38.

1. $A \rightarrow B : A, B, N_A$
2. $B \rightarrow S : A, B, N_B, \{B, S, B, K_2, A, N_A\}_{K_{BS}}$
3. $S \rightarrow A : \{S, A, B, K_2, A, N_A\}_{K_{AS}}, N_B$
4. $A \rightarrow S : \{A, S, A, K_1, B, N_B\}_{K_{AS}}$
5. $S \rightarrow B : \{S, B, A, K_1, B, N_B\}_{K_{BS}}$

Protocol 3.38: Gong's nonce-based protocol

Protocols 3.37 and 3.38 may be extended to provide key confirmation; the cost is an extra message. In the first protocol, B could make use of K_{AB} as an encryption key in message 3 to assure A that he really knows K_{AB}. A new fifth

Petitioner Samsung Ex-1036, 0123

message could provide B similar assurance. In the second protocol, A could make use of K_{AB} as an encryption key in message 4 to assure B that she really knows K_{AB}. A new sixth message could provide A similar assurance.

In Protocols 3.37 and 3.38 both A and B check the freshness of an element received with the keying material. While this is the usual method for ensuring session key freshness in key transport protocols, it is generally redundant in key agreement protocols. Each user can ensure that the key is fresh by simply ensuring that its keying material is fresh; there is no need for a user to be able to verify that the input received from the other user is fresh. Thus, the timestamps in Protocol 3.37 and the nonces in Protocol 3.38 could be eliminated. Protocol 3.39, due to Gong, makes use of this optimisation. It provides key authentication, key freshness and key confirmation in only five messages. As before, both A and B compute the session key as $f(K_1, K_2)$.

1. $A \rightarrow S: A, B, \{A, S, A, K_1, B\}_{K_{AS}}, N_A$
2. $S \rightarrow B: A, B, \{S, B, A, K_1, B\}_{K_{BS}}, N_A$
3. $B \rightarrow S: \{B, S, B, K_2, A\}_{K_{BS}}, \{B, A, N_A\}_{K_{AB}}, N_B$
4. $S \rightarrow A: \{S, A, B, K_2, A\}_{K_{AS}}, \{B, A, N_A\}_{K_{AB}}, N_B$
5. $A \rightarrow B: \{A, B, N_B\}_{K_{AB}}$

Protocol 3.39: Alternative Gong's protocol

3.4.11 Boyd Key Agreement Protocol

Protocol 3.40, proposed by Boyd [57], provides key authentication, key freshness and key confirmation in only four messages. It is a server-based protocol in which both users as well as the server contribute to the key value. The values N_A and N_B are generated by A and B respectively as input to the MAC function determining the session key. Additionally, S generates a value K_S which serves as the MAC key. Both A and B compute the session key as $K_{AB} = MAC_{K_S}(N_A, N_B)$.

1. $A \rightarrow S: A, B, N_A$
2. $S \rightarrow B: \{A, B, K_S\}_{K_{AS}}, \{A, B, K_S\}_{K_{BS}}, N_A$
3. $B \rightarrow A: \{A, B, K_S\}_{K_{AS}}, [N_A]_{K_{AB}}, N_B$
4. $A \rightarrow B: [N_B]_{K_{AB}}$

Protocol 3.40: Boyd key agreement protocol

Petitioner Samsung Ex-1036, 0124

3.4.12 Gong Hybrid Protocol

Protocol 3.41, due to Gong [122], is an example of a hybrid protocol in which only A and S have an input to the key derivation function. It employs two one-way functions: a function f used for key derivation and a function g used for authentication. These functions need not be distinct. The output of the key derivation function f is divided into three components:

$$f(N_S, N_A, B, K_{BS}) = (K_{AB}, H_A, H_B).$$

The first component is the session key value itself. The second component is sent from A to B to assure the latter that A has the key. The third component provides A with reciprocal assurance. In this protocol B derives freshness by checking an element received with the key, while A derives freshness by generating a fresh input to the session key generation process.

1. $A \rightarrow B : A, B, N_A$
2. $B \rightarrow S : A, B, N_A, N_B$
3. $S \rightarrow B : N_S, f(N_S, N_B, A, K_{BS}) \oplus (K_{AB}, H_A, H_B), g(K_{AB}, H_A, H_B, K_{BS})$
4. $B \rightarrow A : N_S, H_B$
5. $A \rightarrow B : H_A$

Protocol 3.41: Gong's hybrid protocol

Boyd and Mathuria [61] demonstrated an unusual feature of Protocol 3.41. Suppose A has executed a normal run of the protocol with B, with the derived session key being K'_{AB} and the other two quantities being H'_A and H'_B. Furthermore, suppose that A has also recorded the reply from the server to B and thus is in possession of the value $g(K'_{AB}, H'_A, H'_B, K_{BS})$. Now, A is able to complete the protocol with B as shown in Attack 3.11.

3. $S \rightarrow A_B : N_S, f(N_S, N_B, A, K_{BS}) \oplus (K_{AB}, H_A, H_B),$
   $g(K_{AB}, H_A, H_B, K_{BS})$
3'. $A_S \rightarrow B : N_S, f(N_S, N_B, A, K_{BS}) \oplus (K'_{AB}, H'_A, H'_B),$
   $g(K'_{AB}, H'_A, H'_B, K_{BS})$
4. $B \rightarrow A : N_S, H'_B$
5. $A \rightarrow B : H'_A$

Attack 3.11: Insider attack on Protocol 3.41

The insight gained from the attack is that a malicious principal A can force B into accepting an old session key as new. It highlights an assumption that was probably not obvious when the protocol was designed.

Petitioner Samsung Ex-1036, 0125

3.4.13 Comparison of Server-Based Protocols

Table 3.6 summarises major properties of the 22 server-based protocols described earlier. For the goals an entry (*) indicates that the goal is claimed but fails due to attack. From the table, we see that all of the protocols except the Needham–Schroeder protocol and Wide-mouthed-frog protocol achieve the key freshness goal. Further, all of the protocols except the BAN Otway–Rees protocol and the BAN Yahalom protocol achieve the key authentication goal.

Table 3.6. Summary of major properties of specific server-based protocols

Properties → ↓ *Protocol*	No. of passes	Key control	Fresh key	Key auth.	Key conf.	Attack
Needham–Schroeder (3.19)	5	S	A (*)	A+B	A	Yes
Denning–Sacco (3.20)	3	S	A+B	A+B	No	No
Bauer–Berson–Feiertag (3.21)	4	S	A+B	A+B	No	No
Otway–Rees (3.22)	4	S	A+B	A+B	No	No
BAN Otway–Rees (3.23)	4	S	A+B	A (*)	No	Yes
AN Otway–Rees (3.24)	4	S	A+B	A+B	No	No
Kerberos (3.25)	3	S	A+B	A+B	B	No
11770-2 Mech. 10 (3.27)	3	S	A+B	A+B	No	No
11770-2 Mech. 12 (3.28)	3	A	A+B	A+B	No	No
11770-2 Mech. 13 (3.29)	4	B	A+B	A+B	No	No
Wide-mouthed-frog (3.30)	2	A	No (*)	Yes	No	Yes
Yahalom (3.31)	4	S	A+B	A+B	B	No
BAN Yahalom (3.32)	4	S	A+B	A (*)	No (*)	Yes
3PKDP (3.33)	9	S	A+B	A+B	A+B	No
Optimised 3PKDP (3.34)	5	S	A+B	A+B	A+B	No
Bellare–Rogaway (3.35)	4	S	A+B	A+B	No	No
Woo–Lam (3.36)	7	S	No (*)	B (*)	A+B	Yes
Gong timestamp (3.37)	4	A/B	A+B	A+B	No	No
Gong nonce-based (3.38)	5	A/B	A+B	A+B	No	No
Gong alternative (3.39)	5	A/B	A+B	A+B	A+B	No
Boyd four-pass (3.40)	4	$S/A/B$	A+B	A+B	A+B	No
Gong hybrid (3.41)	5	S/A	A+B	A+B	A+B	No

Petitioner Samsung Ex-1036, 0126

3.5 Key Establishment Using Multiple Servers

Each of the server-based protocols we have examined so far in this chapter has involved three principals: one server and two users. One natural way to generalise this situation is to allow more than two users. Key establishment with multiple users is the subject of Chap. 6. A different generalisation is to use more than one server. There are at least two potential benefits of such an architecture.

- If one or more servers become unavailable, it may still be possible for the users to establish a session key.
- If one or more servers are untrustworthy, users may still be able to establish a good key.

There have been a few concrete proposals for protocols using multiple servers. We examine two of these in this section.

3.5.1 Gong's Multiple Server Protocol

Gong [124] proposed a number of variant protocols all with the same basic structure. A feature of all these protocols is that the users, A and B, choose the keying material while the n servers, $S_1, S_2, \ldots, S_n$, act as key translation centres to allow keying material from one user to be made available to the other. Initially A shares a long-term key $K_{A,i}$ with each server S_i, and similarly B shares $K_{B,i}$ with S_i.

In order to ensure that the correct key can be recovered even if some servers become unavailable, A and B both split up their secrets using a threshold scheme (see Sect. 1.4.5). Specifically, A chooses a secret x and splits it into shares $x_1, x_2, \ldots, x_n$ so that x can be recovered from any t shares. Similarly B chooses a secret y and divides it into shares $y_1, y_2, \ldots, y_n$. Protocol 3.42 shows a simplified version of Gong's main protocol. Messages 2 and 3 are repeated for each of the n servers so there are $2n+3$ messages sent in total. On receipt of the translated shares from each server, A is able to recover the secret y of B, and similarly B recovers x. The session key is defined as $K_{AB} = h(x, y)$.

1. $A \rightarrow B: \; A, B, N_A, \{A, B, x_i, cc(x)\}_{K_{A,i}}$
2. $B \rightarrow S_i: A, B, N_A, N_B, \{A, B, x_i, cc(x)\}_{K_{A,i}}, \{B, A, y_i, cc(y)\}_{K_{B,i}}$
3. $S_i \rightarrow B: \{B, N_A, y_i, cc_i(y)\}_{K_{A,i}}, \{A, N_B, x_i, cc_i(x)\}_{K_{B,i}}$
4. $B \rightarrow A: \; \{B, N_A, y_1, cc_1(y)\}_{K_{A,1}}, \ldots, \{B, N_A, y_n, cc_n(y)\}_{K_{A,n}}, \{N_A\}_{K_{AB}}, N_B$
5. $A \rightarrow B: \; \{N_B\}_{K_{AB}}$

Protocol 3.42: Gong's simplified multi-server protocol

In order to prevent malicious rogue servers from disrupting the protocol, A and B form a *cross-checksum* for all the shares. The cross-checksum for

Petitioner Samsung Ex-1036, 0127

x is $cc(x) = (h(x_1), h(x_2), \ldots, h(x_n))$ where h is a one-way function. The cross-checksum $cc_i(x)$ received from server S_i may or may not be equal to the correct value $cc(x)$. When B receives a checksum $cc_i(x)$ from server S_i he calculates $h(x_j)$ for every x_j received from any other server S_j, and compares the result with that in $cc_i(x)$. If they are the same then S_j is allocated one credit point. When all the checks are complete B retains those shares from servers with the most credit points. The users require t shares in order to recover the secret. The process will ensure that the correct secret is recovered as long as half of the responding servers are honest and t of them are available.

In Protocol 3.42 an adversary could replay the messages of A or B (or even both) since the server cannot detect replayed requests. However, this does not seem to be a major problem since both parties have assurance that K_{AB} is fresh, from the freshness of their own input. Gong's main protocol [124] differs from Protocol 3.42 by including an initial exchange between A and each server, in which each server delivers a nonce that is returned with the encrypted messages intended for translation. A consequence of this is that servers can detect replayed requests and ignore them, but the cost of this is that the number of messages is increased to $4n + 3$.

3.5.2 Chen–Gollmann–Mitchell Protocol

Chen *et al.* [87] designed two protocols using multiple servers. A significant difference from Protocol 3.42 is that in both of their protocols the servers, instead of the users, choose the keying material. Furthermore, instead of sharing one secret, each server chooses an independent secret value. Both users employ a cross-checksum to decide which servers have given valid inputs and all of these are used in the definition of the session key. The parallel protocol of Chen *et al.* is shown as Protocol 3.43. Messages 2 and 3 are repeated n times between B and each server S_i so there are $2n + 4$ messages in total.

---

1. $A \rightarrow B: \ A, B, N_A$
2. $B \rightarrow S_i: \ A, B, N_A, N_B$
3. $S_i \rightarrow B: \ \{B, N_A, K_i\}_{K_{A,i}}, \{A, N_B, K_i\}_{K_{B,i}}$
4. $B \rightarrow A: \ \{B, N_A, K_1\}_{K_{A,1}}, \ldots, \{B, N_A, K_n\}_{K_{A,n}}, cc_B(1), \ldots, cc_B(n)$
5. $A \rightarrow B: \ cc_A(1), \ldots, cc_A(n), \{B, N_B, N'_A\}$
6. $B \rightarrow A: \ \{A, N'_A, N_B\}$

Protocol 3.43: Chen–Gollmann–Mitchell multi-server protocol

The cross-checksum used in Protocol 3.43 is quite different from that used in Protocol 3.42. If B has apparently received all n keys, then $cc_B(i) = \{h(K_1), h(K_2), \ldots, h(K_n)\}_{K_i}$, for all i with $1 \leq i \leq n$, where h is a one-way function. However, if B has not received any message from server S_j then

Petitioner Samsung Ex-1036, 0128

$cc_B(j)$ is simply an error message, and also $h(K_j)$ is replaced by an error message in the calculation of the other $cc_B(i)$ values. On receipt of the checksums $cc_B(1), \ldots, cc_B(n)$ from B, A first decrypts the values and compares them. Some of these may be different if A and B have received some different K_i values, but as long the majority of the servers are honest and operational, the majority of the decrypted values will be the same. The K_i secrets are retained for this majority of i values and the others discarded. The cross-checksums $cc_A(i)$ are then defined in a symmetrical way, except that error messages are inserted either if A did not receive a K_i value, or if B has indicated that he did not receive it.

Once the good keys have been identified, the session key K_{AB} is defined to be the hash of all the good K_i values concatenated. Chen *et al.* show that as long as at least half of the servers are honest and operational, then an honest A and B pair will accept the same K_i values and hence the same K_{AB}.

Chen *et al.* also proposed a variant of Protocol 3.43, which they called a 'cascade protocol'. The difference is that instead of B making a request to each server by repeating message 2, the request is passed on from server S_1 to server S_2 and so on. The response from each server is also sent on to the next server. Finally server S_n returns all the server responses to B, and the last three messages are the same as in Protocol 3.43. The advantage of the cascade protocol is that the number of messages sent is reduced to $n + 5$ since all but one of the responses in message 3 of Protocol 3.43 are no longer required. However, the protocol will only work if all the servers are operational; if only one server fails to cooperate, either maliciously or due to an error, then the protocol will fail.

3.6 Conclusion

To a large extent the problem of key establishment between two parties using symmetric cryptography seems to have been solved. There have been no significant new protocols in recent years. Efficient solutions have evolved which have resisted attacks or even have security proofs. Tables 3.2, 3.4 and 3.6 show that there exist a variety of protocols which should be suitable for most applications requiring these sorts of protocols.

Petitioner Samsung Ex-1036, 0129

4

Authentication and Key Transport Using Public Key Cryptography

4.1 Introduction

It is generally regarded that there are two main potential advantages of public key techniques over symmetric cryptography. The first is that public key systems allow the straightforward definition of digital signatures, thereby enabling the service of non-repudiation which is so useful in commercial applications. The second is the simplification of key management, because there is no requirement for the on-line third party that is part of typical protocols based on symmetric cryptography. The first of these advantages is not really our concern in this book since non-repudiation is of limited value in authentication and key establishment. However, the second advantage has led to a great variety of new key establishment protocols since the invention of public key cryptography. In the modern distributed communications environments exemplified by the Internet, public-key-based protocols have become far more important than protocols based on symmetric cryptography.

There are two costs that must be paid in exchange for these benefits. The first one is the high computational cost that comes with all known public key cryptosystems. Despite the advances made in public key cryptosystems and the advantages of elliptic curve cryptology [42], public key algorithms require two or three orders of magnitude more computation than symmetric algorithms. Although computing power on a typical desktop has increased to the extent where a handful of public key operations will not cause a delay of more than a fraction of a second, the overhead for servers of multiple clients and for low-power computing devices, such as smart cards, is still significant. At the same time increased computing power means that longer key sizes are required so that the cost of public key operations increases. It is therefore essential that designers of public-key-based protocols should minimise the number of public key operations wherever possible. Another issue to be considered is whether the protocol requires more private key operations (signature generations and decryptions) or more public key operations (signature verifications and encryptions). RSA and related algorithms are much more efficient for public key

Petitioner Samsung Ex-1036, 0130

operations than for private key operations, while for most algorithms based on discrete logarithms the opposite is true. Furthermore, although algorithms based on discrete logarithms (including elliptic curve algorithms) are more efficient than RSA overall, a protocol that requires mainly public key operations may be much more efficiently implemented using RSA with a small public exponent. Generally it is not a simple task to compare the efficiency of different protocols when implemented using different algorithms.

The second cost of public key cryptography is that public keys still need to be managed. The best solution for a public key infrastructure is still the subject of considerable research. Although public keys need not (indeed usually should not) be kept confidential their integrity must be maintained, normally through use of *certificates* signed by reputable third parties. The question of how to deal with compromised private keys is a tricky one, but the most established solution is to use certificate revocation lists to check whether a public key is still valid, much as a blacklist is checked before accepting a credit card. In the descriptions of protocols given in this chapter we shall ignore the certificates, or any other means, used to ensure that a public key is valid, and simply assume that public keys are available to the parties that need them and are guaranteed to be correct. What *correct* means here is that the claimed owner of the public key is the only entity who is able to use the corresponding private key. Designers and users of public key protocols need to be aware that this is an important simplification that must be addressed in any implementation.

In this chapter we shall not cover the important class of public-key-based key agreement protocols which are the subject of Chap. 5. In the remainder of this section we explain our notation in this chapter and discuss general design principles for public key protocols. Section 4.2 examines public key protocols for entity authentication and Sect. 4.3 covers public key protocols providing key transport.

4.1.1 Notation

The notation used in this chapter is summarised in Table 4.1. In all our protocol descriptions generic algorithms for public key encryption and digital signatures are shown, although in some cases we will note that certain protocols were designed with specific algorithms in mind. We assume that all encryption algorithms provide semantic security and sometimes comment if non-malleability is also required.

4.1.2 Design Principles for Public Key Protocols

Anderson and Needham [15] have proposed a set of what they call *robustness principles* for public-key-based protocols. These can be considered as more specific instances of the general principles for protocol design proposed earlier by Abadi and Needham and which were discussed in Sect. 1.7. They form a

Petitioner Samsung Ex-1036, 0131

Table 4.1. Notation used throughout Chap. 4

$E_X(M)$	Encryption of message M using the public key of principal X.
$Sig_X(M)$	Signature with appendix of message M by principal X.
N_X	Random nonce value chosen by principal X.
T_X	Timestamp chosen by principal X.
$\{M\}_K$	Symmetric encryption of message M with key K.

checklist that can be used by protocol designers to avoid the most common errors. A summary of these principles is shown in Table 4.2. Anderson and Needham give several examples of potential attacks that can result by ignoring these principles.

Table 4.2. Anderson and Needham's robust principles for public key protocols

1. Sign before encrypting. If a signature is affixed to encrypted data then one cannot assume that the signer has any knowledge of the data.
2. Be careful how entities are distinguished. If possible avoid using the same key for two different purposes (such as signing and decryption) and be sure to distinguish different runs of the same protocol from each other.
3. Be careful when signing or decrypting data that you never let yourself be used as an oracle by your opponent.
4. Account for all the bits – how many provide equivocation, redundancy, computational complexity, and so on.
5. Do not assume the secrecy of anybody else's 'secrets' (except possibly those of a certification authority).
6. Do not assume that a message you receive has a particular form (such as g^r for known r) unless you can check this.
7. Be explicit about the security parameters of cryptographic primitives.

Later Syverson [314] questioned the applicability of some of the principles proposed by Anderson and Needham by showing examples when they are not appropriate. Nevertheless, Syverson concludes that the principles are still useful but should be used intelligently and critically by the protocol designer. In this chapter we will see that several protocols ignore the first Anderson–Needham principle, including protocols that are proven secure. This is discussed further in Sect. 4.3.1 and Sect. 4.3.2.

Petitioner Samsung Ex-1036, 0132

4.2 Entity Authentication Protocols

Before looking at protocols for key establishment we examine some protocols that achieve only authentication. In this section we examine some prominent examples, in particular those in the international standard ISO/IEC 9798 Part 3 [159]. We discuss whether they achieve the extensional definition of entity authentication introduced in Chap. 2, or the simpler liveness property.

4.2.1 Protocols in ISO/IEC 9798-3

Five protocols are included in ISO/IEC 9798-3 for authentication between a pair of principals. Two of these are for unilateral authentication of one party to another, and three are for mutual authentication of both parties to each other. Two of the protocols (Protocols 4.2 and 4.4) are also included in the US standard FIPS 196 [246] which includes some further guidance on the use of optional fields.

Each protocol includes options for various 'text' fields to be included in each message for application-dependent data. According to the standard, a text field may be included in a signature for various reasons including:

- to authenticate any information;
- to add extra redundancy to the signature;
- to provide additional time variant parameters such as timestamps;
- to provide validity information for the protocol in use.

Unsigned text fields may be used for the claimed identity of the message sender which is not otherwise explicitly stated in the protocols. Since they are not part of the basic protocols we shall ignore the optional text fields in the following descriptions.

The first protocol in the standard, shown as Protocol 4.1, consists of a single message from a claimant A to a verifier B. The timestamp T_A is used to provide freshness, or alternatively it may be replaced by a counter. The protocol assures B that A is alive, as well as providing assurance that A is aware of B as her peer entity.

1. $A \rightarrow B : T_A, B, Sig_A(T_A, B)$

Protocol 4.1: ISO/IEC 9798-3 one-pass unilateral authentication

The second protocol (4.2) uses a nonce instead of a timestamp, which is the most obvious difference from Protocol 4.1. The other difference is the inclusion of the random value N_A chosen by A. This field has nothing to do with authentication but is included to ensure that A is not signing a message that has been chosen by B; this could cause problems for A if the signature

Petitioner Samsung Ex-1036, 0133

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : N_A, N_B, B, Sig_A(N_A, N_B, B)$

Protocol 4.2: ISO/IEC 9798-3 two-pass unilateral authentication

scheme and signing key used in the protocol are also used in other applications. This protocol provides entity authentication of A to B.

It is interesting that the standard allows the field containing the identity B in message 2 to be omitted, stating that its inclusion 'depends on the environment in which this authentication mechanism is used'. With a weaker definition of entity authentication, having no requirement for knowledge of the peer entity, the omission of this field is acceptable. However, if knowledge of the peer entity is required then this field must be included in the signature of message 2. The standard specifically forbids exclusion of the corresponding field in the one-pass Protocol 4.1. We can speculate that the reasoning behind this is that when a timestamp is used B has nothing to connect him to the protocol instance, whereas when B's challenge is returned he can reason that he is 'connected' with A even if A has not indicated this. We refer the reader to Chap. 2 for more discussion on the possible shades of entity authentication.

The third protocol (4.3) is simply the combination of two instances of Protocol 4.1 for one-pass unilateral authentication and as in that protocol the timestamps T_A and T_B may be replaced by counters. This protocol provides mutual entity authentication. Because the messages sent are independent of each other, this protocol can be executed in one round.

1. $A \rightarrow B : T_A, B, Sig_A(T_A, B)$
2. $B \rightarrow A : T_B, A, Sig_B(T_B, A)$

Protocol 4.3: ISO/IEC 9798-3 two-pass mutual authentication

The fourth protocol (4.4) is an extension of Protocol 4.2, allowing both A and B to use respective nonces, N_A and N_B. The standard again allows the field B in message 2 and the field A in message 3 to be omitted; but the field must be included at least in message 2 if it is desired for B to gain assurance that A is aware of B as her peer entity. The standard specifically forbids exclusion of the corresponding fields in Protocol 4.3 for two-pass mutual authentication.

Blake-Wilson and Menezes [44] have proven the security of Protocol 4.4 using the Bellare–Rogaway model described in Chap. 2. This means that the protocol satisfies the matching conversations property. By providing signatures of the other party's identity in messages 2 and 3, the protocol also provides knowledge of the peer entity.

Petitioner Samsung Ex-1036, 0134

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : N_A, N_B, B, Sig_A(N_A, N_B, B)$
3. $B \rightarrow A : N_B, N_A, A, Sig_B(N_B, N_A, A)$

Protocol 4.4: ISO/IEC 9798-3 three-pass mutual authentication

Protocol 4.5 shows an earlier version of Protocol 4.4 that was proposed during the standardisation process. The only difference from Protocol 4.4 is that B chooses and signs a nonce N'_B in the final message, which is different from the nonce N_B used in the first two messages. A probable reason for this choice is that it ensures that B does not have to sign a message that is predictable by A.

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : N_A, N_B, B, Sig_A(N_A, N_B, B)$
3. $B \rightarrow A : N'_B, N_A, A, Sig_B(N'_B, N_A, A)$

Protocol 4.5: Early version of ISO/IEC 9798-3 three-pass mutual authentication

Protocol 4.5 is subject to Attack 4.1, which has become known as the 'Canadian attack' because it was publicised by the Canadian team taking part in the standards process. In the attack the adversary C sets up two protocol runs, masquerading as B to A and as A to B. The response from A in the second run can be used by C to complete the first run with B.

1. $C_B \rightarrow A : N_C$
2. $A \rightarrow C_B : N_A, N_C, B, Sig_A(N_A, N_C, B)$
1'. $C_A \rightarrow B : N_A$
2'. $B \rightarrow C_A : N_B, N_A, A, Sig_B(N_B, N_A, A)$
3. $C_B \rightarrow A : N_B, N_A, A, Sig_B(N_B, N_A, A)$

Attack 4.1: Canadian attack on Protocol 4.5

The result of the attack is that A completes the protocol apparently with B, whereas in fact the protocol was run with C. In terms of matching conversations this is a valid attack, since A and B disagree on the second message. However, A is correct to conclude that B is alive and has indicated awareness of A as his peer entity. Therefore the extensional protocol goals do not seem to be violated. This attack is closely related to Attack 2.1 discussed in Chap. 2. Mitchell and Thomas [238] discuss Attack 4.1. They also consider methods to prevent signatures obtained during similar protocols from being abused by an

Petitioner Samsung Ex-1036, 0135

adversary. These methods include use of a protocol identifier in every signed message.

Protocol 4.6 is the final protocol in the standard; it allows authentication to be conducted in parallel between A and B. Thus messages 1 and $1'$ can be sent together, as can 2 and $2'$. As with Protocol 4.4 the standard allows the identity fields B and A, in messages 2 and $2'$ respectively, to be omitted. Once again, omission of these fields means that knowledge of the peer entity cannot be provided, this time in either direction.

---

1. $A \rightarrow B : N_A$
$1'$. $B \rightarrow A : N_B$
2. $A \rightarrow B : N_A, N_B, B, Sig_A(N_A, N_B, B)$
$2'$. $B \rightarrow A : N_B, N_A, A, Sig_B(N_B, N_A, A)$

Protocol 4.6: ISO/IEC 9798-3 two-pass parallel authentication

We note that Protocols 4.4 and 4.6 both link together previous protocol messages in their signed messages; this means that they provide the 'matching conversations' property described in Chap. 2. In this sense it can be argued that they provide more than Protocol 4.3 even though all three protocols provide mutual entity authentication.

4.2.2 Protocols in ISO/IEC 9798-5

The international standard ISO/IEC 9798-5 [161] is devoted to entity authentication mechanisms using zero knowledge techniques. Three protocols are specified. The first two cover essentially the well-known zero knowledge identification protocols of Fiat–Shamir [111], Guillou–Quisquater [137] and Schnorr [291]. These protocols are strongly dependent on specific cryptographic mechanisms and are designed to prove knowledge of private keys corresponding to known public keys. The third protocol also provides unilateral authentication without knowledge of the peer entity.

4.2.3 SPLICE/AS

The protocol known as SPLICE/AS was proposed in 1990 by Yamaguchi *et al.* [337] to provide mutual authentication between a client A and server B. A number of different papers have progressively found attacks and proposed improvements. The full protocol includes retrieval of certified public keys from an authentication server. Hwang and Chen [152] showed that the original certificate format was flawed, allowing an adversary to alter the apparent public key of either client or server. The authentication part is shown in

Petitioner Samsung Ex-1036, 0136

1. $A \rightarrow B : A, B, T_A, L, E_B(N_A), Sig_A(A, T_A, L, E_B(N_A))$
2. $B \rightarrow A : B, A, E_A(B, N_A + 1)$

Protocol 4.7: SPLICE/AS protocol

Protocol 4.7, where L is a lifetime of the message, purported to be used to prevent replay.

The protocol is intended to provide mutual entity authentication. The server B checks the signature and timestamp on receipt of the first message in order to authenticate A. When A receives the returned message she checks her nonce in order to authenticate B. However, Protocol 4.7 was attacked by Clark and Jacob [93] who noted that an adversary C is able to intercept the message from A to B and replace the signature of A with C's own signature. Consequently A believes the protocol has been run with B while B believes it has been run with C.

1. $A \rightarrow C_B : A, B, T_A, L, E_B(N_A), Sig_A(A, T_A, L, E_B(N_A))$
1'. $C \rightarrow B : \quad C, B, T_A, L, E_B(N_A), Sig_C(C, T_A, L, E_B(N_A))$
2'. $B \rightarrow C : \quad B, C, E_C(B, N_A + 1)$
2. $C_B \rightarrow A : B, A, E_A(B, N_A + 1)$

Attack 4.2: Attack of Clark–Jacob on SPLICE/AS protocol

Attack 4.2 shows that neither party can be aware of their peer entity and that matching conversations cannot be guaranteed. However, the responder B can be sure that the signer of the first message is live, while A also gets this assurance, since only B is able to decrypt N_A. Clark and Jacob proposed Protocol 4.8 as an alternative to prevent their attack.

1. $A \rightarrow B : A, B, T_A, L, E_B(A, N_A), Sig_A(A, T_A, L, E_B(A, N_A))$
2. $B \rightarrow A : B, A, E_A(B, N_A + 1)$

Protocol 4.8: Clark–Jacob variant of SPLICE/AS

The protocol is again intended to provide mutual entity authentication. A look at the message sent from A to B shows that A has given no indication that she wishes to communicate with B so that this protocol does not provide knowledge of the peer entity. The only difference from Protocol 4.7 is that the identity of A is included in the encrypted field of message 1 which, it is assumed, cannot be altered by C. This assumption is only reasonable if the public key encryption algorithm used is non-malleable. Gray [136] pointed out

Petitioner Samsung Ex-1036, 0137

that Attack 4.2 still works if the encrypted identity of A can be changed to another identity. He therefore proposed the simplified Protocol 4.9 which does provide knowledge of the peer entity. This can be seen as a hybrid of Protocol 4.3 and Protocol 4.4 since B relies on a timestamp to ensure liveness while A uses a nonce.

1. $A \rightarrow B : A, B, T_A, L, N_A, Sig_A(B, T_A, L, N_A)$
2. $B \rightarrow A : B, A, N_A, Sig_B(A, N_A)$

Protocol 4.9: Gray variant of SPLICE/AS

4.2.4 Comparison of Entity Authentication Protocols

Table 4.3 compares some of the main features of the various entity authentication protocols explored in this section. As with the authentication protocols based on symmetric encryption examined in Chap. 3, the protocols that have resisted attack all have similar basic format. Therefore it seems that the basic two-party authentication protocols are quite well understood and the protocols in the ISO/IEC 9798-3 standard all seem satisfactory. It would be preferable if the security proof for the three-pass mutual authentication protocol were extended to cover the other cases too.

Table 4.3. Summary of properties of public key entity authentication protocols

Properties → ↓ *Protocol*	Liveness	Entity authentication	Security proof	Attack
9798-3 One-Pass Unilateral (4.1)	B	B	No	No
9798-3 Two-Pass Unilateral (4.2)	B	B	No	No
9798-3 Two-Pass Mutual (4.3)	A+B	A+B	No	No
9798-3 Three-Pass Mutual (4.4)	A+B	A+B	Yes	No
9798-3 Two-Pass Parallel (4.6)	A+B	A+B	No	No
SPLICE/AS (4.7)	A+B	No	No	Yes
Clark–Jacob SPLICE (4.8)	A+B	No	No	Yes
Gray SPLICE (4.9)	A+B	A+B	No	No

Petitioner Samsung Ex-1036, 0138

4.3 Key Transport Protocols

As discussed in Chap. 1, key transport refers to protocols in which one principal chooses a session key and securely transports it to the other principal, or principals. Sometimes it can be difficult to classify protocols as key transport or key agreement. Some protocol specifications allow each of two principals to choose and transport their own key but leave open whether these two will be combined to form an agreed key, or used separately. We have included in this chapter those protocols that can potentially be used for key transport, but may be implemented to provide key agreement.

4.3.1 Protocols in ISO/IEC 11770-3

In this section we shall examine protocols specified in the international standard ISO/IEC 11770 Part 3 [162]. The standard specifies six key transport protocols in a generic fashion and with some optional items. We shall see that many of these standardised protocols are related to some other previously published protocols, particularly those in ISO/IEC 9798-3.

Protocols in the standard are presented using generic encryption and signature functions, but specific examples are included in an annex (which is not a formal part of the standard). As with the protocols in ISO/IEC 9798-3 discussed in Sect. 4.2.1, there are various optional text fields included in all the standardised protocols which are ignored in our descriptions here. The standard does not distinguish between signatures with message recovery and those with appendix by assuming that if a signature with appendix is used then the message signed is sent together with the signature. We show all protocols using signatures with appendix.

Protocol 4.10 shows mechanism 1, the simplest in the standard. The session key K_{AB} is chosen by A and sent to B encrypted with B's public key. Also encrypted are the identity of A and the timestamp T_A (or alternatively a counter). In this protocol, as with all the key transport protocols in this standard, it is essential that the public key encryption used provides non-malleability as well as semantic security. If this were not the case then the adversary may be able to change the value of the fields A and T_A included with the encrypted session key. Notice that these fields are not generally required to be confidential, so it may be inferred that the designers intended to use the non-malleability to bind them to the session key. However, the properties of the encryption algorithm used are not explicitly stated in the standard.

1. $A \rightarrow B : E_B(A, K_{AB}, T_A)$

Protocol 4.10: ISO/IEC 11770-3 Key Transport Mechanism 1

Petitioner Samsung Ex-1036, 0139

From A's viewpoint Protocol 4.10 provides a good key since A can choose the key to be fresh and the encryption provides confidence that it is known only to herself and to B. However, A achieves no assurance with regard to key confirmation, or even that B is operative. Since there is no authentication at all of the origin of the key this protocol gives no assurance to B as to who this key is shared with. As long as B can rely on the freshness of T_A, he knows that the message received is fresh; but this does not seem useful since without authentication of the sending party B cannot deduce that K_{AB} itself is fresh. Indeed the standard states that the inclusion of T_A is optional.

Protocol 4.11 shows mechanism 2, which extends mechanism 1 by adding a signature of the whole message by A. The timestamp T_A may again be replaced by a counter. As before, the protocol provides a good key for A but provides no key confirmation from B. However, in contrast to Protocol 4.10, the signature of A allows B to achieve key confirmation. As long as B trusts A to generate the key faithfully, B also achieves the good key property. Protocol 4.11 is similar to Protocol 4.1 for entity authentication given in Sect. 4.2. Indeed, through use of the optional text fields in that protocol, Protocol 4.11 conforms to the 9798-3 standard as well as to 11770-3.

1. $A \rightarrow B : B, T_A, E_B(A, K_{AB}), Sig_A(B, T_A, E_B(A, K_{AB}))$

Protocol 4.11: ISO/IEC 11770-3 Key Transport Mechanism 2

It is interesting to note that this protocol violates the first principle of Anderson and Needham (see Table 4.2). The motivation behind this principle is that the signature does not provide assurance that the signer knows the plaintext in the encrypted message. For example, in Protocol 4.11 an adversary C could remove the signature of A and replace it with C's own signature. This illustrates that it is essential that B trusts the signer of the message only to sign keys that it has generated and encrypted. However, as long as the encryption used is non-malleable an adversary cannot change the identity of A in the encrypted part of the message and so this does not result in a valid attack.

Anderson and Needham give details of an attack on both RSA and discrete logarithm encryption algorithms which allows a malicious principal B to gain a signature on an encrypted message of his choice by registering a new public encryption key matched to the signed encrypted message. However, these attacks do not apply if non-malleable versions of encryption are used; also Syverson [314] discussed a number of practical difficulties with the attack. Nevertheless it is prudent for users of this protocol to consider such possibilities.

Mechanism 3, shown in Protocol 4.12, swaps around the order in which the signature and encryption are applied in mechanism 2. The intention of each of the fields is the same so that the protocol achieves the same goals as

Petitioner Samsung Ex-1036, 0140

mechanism 2 as long as B trusts A to generate a good key. The inclusion of the timestamp T_A (or counter) is again optional in the standard but without it B cannot gain key freshness or key confirmation.

1. $A \rightarrow B : E_B(B, K_{AB}, T_A, Sig_A(B, K_{AB}, T_A))$

Protocol 4.12: ISO/IEC 11770-3 Key Transport Mechanism 3

Protocol 4.13 is a closely related protocol proposed by Denning and Sacco [99]. Two preliminary messages, which allow principal A to obtain public key certificates for both A and B, are omitted here. The only difference from Protocol 4.12 is the omission of the identity of B.

1. $A \rightarrow B : E_B(K_{AB}, T_A, Sig_A(K_{AB}, T_A))$

Protocol 4.13: Denning–Sacco public key protocol

This omission allows an attack, discussed by Abadi and Needham [4], in which a malicious B can engage in a protocol run with A as initiator, and then send message 1 to C re-encrypted with C's public key. As a result C believes the key to be shared with A but it is also known to B. Abadi and Needham suggest including the identities of both A and B in the signature of message 1 to prevent this attack, but including only B's identity, as in Protocol 4.12, seems sufficient.

Mechanism 4, shown in Protocol 4.14, is a two-pass protocol very similar to mechanism 2 (with roles reversed), the main difference being that A now uses a nonce N_A to achieve key freshness and entity authentication of B. As long as B is trusted to generate the key, A achieves the good key and key confirmation properties. B achieves the good key property but no authentication of A.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : A, N_A, N_B, E_A(B, K_{AB}), Sig_B(A, N_A, N_B, E_A(B, K_{AB}))$

Protocol 4.14: ISO/IEC 11770-3 Key Transport Mechanism 4

The random number N_B is optional in the standard, and apparently used only to maintain consistency with Protocol 4.2 which is the corresponding entity authentication protocol in the 9798-3 standard. Protocol 4.14, with N_B omitted, was proven secure by Shoup in his simulation model [299] under the assumption of only static corruptions by the adversary (recall that this is

Petitioner Samsung Ex-1036, 0141

equivalent to a proof of security in the Bellare–Rogaway model as discussed in Chap. 2).

Protocol 4.15 shows mechanism 5, which is a mutual version of mechanism 4 for which two session keys K_{AB} and K_{BA} are chosen by A and B respectively. The standard suggests that the two session keys may be combined using a one-way hash function, in which case this protocol is strictly a key agreement protocol rather than a key transport protocol. It also suggests that either $E_B(A, K_{AB})$ could be omitted from message 3 so that K_{BA} becomes the session key, or $E_A(B, K_{BA})$ could be omitted from message 2 so that K_{AB} is the session key. If K_{AB} is used to define the session key, then only B obtains key confirmation. Mechanism 5 conforms to the 9798-3 standard by adding optional text fields to Protocol 4.4, the corresponding entity authentication protocol.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : N_B, N_A, A, E_A(B, K_{BA}), Sig_B(N_B, N_A, A, E_A(B, K_{BA}))$
3. $A \rightarrow B : N_A, N_B, B, E_B(A, K_{AB}), Sig_A(N_A, N_B, B, E_B(A, K_{AB}))$

Protocol 4.15: ISO/IEC 11770-3 Key Transport Mechanism 5

The final key transport protocol in the ISO/IEC 11770-3 standard is mechanism 6, shown in Protocol 4.16. In contrast to all the other protocols in the standard it uses only encryption and no signatures. Perhaps it is most clear here that the encryption algorithm requires non-malleability since otherwise all the fields used for authentication could be potentially altered by the adversary.

1. $A \rightarrow B : E_B(A, K_{AB}, N_A)$
2. $B \rightarrow A : E_A(B, K_{BA}, N_A, N_B)$
3. $A \rightarrow B : N_B$

Protocol 4.16: ISO/IEC 11770-3 Key Transport Mechanism 6

The standard states that K_{AB} and K_{BA} may be combined using a one-way function to form a single session key. It is also stated that K_{AB} may be used by A to encipher messages for B and to authenticate messages from B, and K_{BA} may be used in an analogous way by B. The protocol achieves mutual entity authentication and mutual key confirmation.

An earlier draft version of the ISO/IEC 11770-3 standard included Protocol 4.17 instead of Protocol 4.16; the former has become known as the Helsinki protocol due to the location of a particular meeting of the relevant standards

Petitioner Samsung Ex-1036, 0142

committee. The only difference from the final standardised Protocol 4.16 is that in message 2 the identity field of B is missing.

1. $A \rightarrow B : E_B(A, K_{AB}, N_A)$
2. $B \rightarrow A : E_A(K_{BA}, N_A, N_B)$
3. $A \rightarrow B : N_B$

Protocol 4.17: Helsinki protocol

Attack 4.3 on the Helsinki protocol was published by Horng and Hsu [148] in 1998. There is a strong similarity between this attack and Lowe's earlier attack on the Needham–Schroeder public key protocol discussed in Sect. 4.3.3 below. The adversary, C, induces A to commence the protocol with C, and then starts a protocol run with B while masquerading as A.

1. $A \rightarrow C : \quad E_C(A, K_{AB}, N_A)$
1'. $C_A \rightarrow B : E_B(A, K'_{AB}, N_A)$
2'. $B \rightarrow C_A : E_A(K_{BA}, N_A, N_B)$
2. $C \rightarrow A : \quad E_A(K_{BA}, N_A, N_B)$
3. $A \rightarrow C : \quad N_B$
3'. $C_A \rightarrow B : N_B$

Attack 4.3: Attack on Helsinki protocol

Both A and B have the view of a successful protocol run. However, if the session key is $f(K_{AB}, K_{BA})$ for some one-way function f, then A 'believes' it shares this key with C, while B 'believes' it shares $f(K'_{AB}, K_{BA})$ with A. Notice that the goal of implicit key authentication has not been violated in this attack, because C does not know K_{BA} and therefore cannot compute either of the session keys accepted by A and B. However, entity authentication is not achieved in that B has incorrect knowledge of his peer entity. Mitchell and Yeun [240] proposed to fix the protocol by adding B's identity to message 2 which, as we have seen, was the solution adopted in the final ISO/IEC 11770 standard.

4.3.2 Blake-Wilson and Menezes Provably Secure Key Transport Protocol

Blake-Wilson and Menezes [44] have proven the security of Protocol 4.18, which is a simplified version of Protocol 4.15 (indeed it is an optional variant conforming to the standard). Here the session key K_{AB} is chosen by B. The protocol is strongly related to Protocol 4.4 for entity authentication, and like

Petitioner Samsung Ex-1036, 0143

that is proven secure in the Bellare–Rogaway model. The proof incorporates the assumption that the encryption algorithm provides non-malleability by assuming that the adversary is able to conduct a chosen-ciphertext attack.

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : B, A, N_B, N_A, E_A(B, K_{AB}), Sig_B(B, A, N_B, N_A, E_A(B, K_{AB}))$
3. $A \rightarrow B : A, B, N_B, Sig_A(A, B, N_B)$

Protocol 4.18: Blake-Wilson–Menezes provably secure key transport protocol

We note that again this provably secure protocol ignores the first principle of Anderson and Needham (see Table 4.2) not to sign encrypted data. In this case the order of encryption and signature is not chosen by chance; it *has* to be this way around in order for the proof to work. The reason for this is that it is necessary in the proof to be able to simulate how the principals in the protocol behave when the plaintext is not known. If the plaintext is included in a signature then it is not possible to determine if a received message has been properly signed without knowing if the plaintext in the signature is the same as in the ciphertext. Anderson and Needham [15] commented that protocol logics, such as the BAN logic, on the contrary cannot be used when encrypted messages are signed. This illustrates an interesting dichotomy between different methods of protocol validation.

4.3.3 Needham–Schroeder Public Key Protocol

The Needham–Schroeder public key protocol [249] was one of the earliest published key establishment protocols along with its well-known companion using symmetric encryption (see Sect. 3.4.1). Protocol 4.19 shows the messages exchanged. There is a strong similarity with the Helsinki protocol (4.17). The protocol was designed to provide mutual entity authentication but with the option of using the exchanged nonces, N_A and N_B, as shared secrets for key establishment.

1. $A \rightarrow B : E_B(N_A, A)$
2. $B \rightarrow A : E_A(N_A, N_B)$
3. $A \rightarrow B : E_B(N_B)$

Protocol 4.19: Needham–Schroeder public key protocol

Although this protocol is nearly 20 years old it has aroused quite some interest recently. Lowe [211] discovered Attack 4.4 which shows that B cannot be sure that the final message came from A. Notice that A has never explicitly

Petitioner Samsung Ex-1036, 0144

declared her intention to converse with B so this protocol cannot provide any assurance to B that A has knowledge of B as the peer entity.

---

1. $A \rightarrow C: \quad E_C(N_A, A)$
1'. $C_A \rightarrow B: E_B(N_A, A)$
2'. $B \rightarrow C_A: E_A(N_A, N_B)$
2. $C \rightarrow A: \quad E_A(N_A, N_B)$
3. $A \rightarrow C: \quad E_C(N_B)$
3'. $C_A \rightarrow B: E_B(N_B)$

Attack 4.4: Lowe's attack on Needham–Schroeder public key protocol

Attack 4.4 is similar to Attack 4.3 on the Helsinki protocol. In order to fix the protocol against his attack, Lowe proposed the variant Protocol 4.20 which simply includes the identifier of B in the second message.

---

1. $A \rightarrow B: E_B(N_A, A)$
2. $B \rightarrow A: E_A(N_A, N_B, B)$
3. $A \rightarrow B: E_B(N_B)$

Protocol 4.20: Lowe's variant of Needham–Schroeder public key protocol

Lowe was able to prove secure a finite model of Protocol 4.20 using the model checker FDR (see Sect. 2.5.1) and extended the proof to the infinite version using several pages of mathematical reasoning. It is again important to notice that this revised protocol is only secure as long as the encryption algorithm used provides non-malleability. Otherwise it cannot be guaranteed that an adversary will not be able to alter the value of the identifier B in message 2 even without knowing the values of N_A and N_B. Protocol 4.20 has a lot in common with Protocol 4.16 and the properties it achieves are the same. Instead of encrypting an explicit session key, the nonces of A and B can act as a shared secret; this is the reason why the third message needs to be encrypted.

4.3.4 Protocols in the X.509 Standard

The X.500 series of recommendations was standardised by the ITU (formerly CCITT) in parallel with ISO to provide directory services for communications. One purpose of the directory is to store certificates for public keys and Part 8 of the standard [163] uses such public keys as the basis for the Authentication Framework. This framework specification includes a number of protocols for authentication and key establishment which can be used for access control to

Petitioner Samsung Ex-1036, 0145

the directory or for other purposes. The protocols are classified as either *simple* or *strong*. Simple authentication uses passwords sent either in cleartext or as input to a one-way function; we shall consider only the strong authentication protocols that use public key cryptographic methods.

There are three protocols specified, with one, two and three message flows respectively. Each protocol extends the previous one by adding an extra message. The goal of each protocol is transport of a session key from A to B and, for the two and three flow protocols, transport of a session key from B to A.

In the simplest protocol there is only one message which is sent from A to B. The certificates (or more generally a chain of certificates) are omitted in the following descriptions, as well as optional signed data. The standard allows for simplified versions in which the session key is omitted, intended to provide entity authentication only. Protocol 4.21 shows the one-pass version with the encrypted data forming the session key K_{AB}; this is suggested as only one possibility by the standard.

1. $A \rightarrow B : T_A, N_A, B, E_B(K_{AB}), Sig_A(T_A, N_A, B, E_B(K_{AB}))$

Protocol 4.21: X.509 one-pass authentication

There is a strong similarity between Protocol 4.21 and Protocol 4.11. The main difference is that here the identity of A is missing from the encrypted data, which makes any potential attacks on Protocol 4.11 easier to mount. In particular, an adversary may remove the signature on the message and replace it with a new signature on the identical message which leads to B believing that the key was sent by the adversary. This problem has been pointed out by Burrows *et al.* [76] while I'Anson and Mitchell [153] also discuss the consequences of such an attack when the encrypted portion of the message acts as a confidential request for information. Both sets of authors suggest that to fix this problem the signature should include the unencrypted key (hashed to protect its confidentiality) with the encrypted key sent separately. An alternative is to use Protocol 4.11 instead.

Protocol 4.22 is the two-pass X.509 protocol; the same first message is sent from A to B as in Protocol 4.21 and a reply is sent from B to A, which is symmetrical except that both nonces are included in the signed part of this message.

1. $A \rightarrow B : T_A, N_A, B, E_B(K_{AB}), Sig_A(T_A, N_A, B, E_B(K_{AB}))$
2. $B \rightarrow A : T_B, N_B, A, N_A, E_A(K_{BA}), Sig_B(T_B, N_B, A, N_A, E_A(K_{BA}))$

Protocol 4.22: X.509 two-pass authentication

Petitioner Samsung Ex-1036, 0146

Similar remarks to those concerning Protocol 4.21 apply. The protocol may be fixed in different ways such as by adding the sender's name to the encrypted blocks to form a mutual version of Protocol 4.11. The encrypted data sent from B to A is shown as a session key K_{BA}; although the standard suggests this data may be used as a session key there is no recommendation on whether this should be used separately or combined with K_{AB}. Protocol 4.23 is the final X.509 protocol and includes a third message intended to provide acknowledgement of message 2 by A.

1. $A \rightarrow B : T_A, N_A, B, E_B(K_{AB}), Sig_A(T_A, N_A, B, E_B(K_{AB}))$
2. $B \rightarrow A : T_B, N_B, A, N_A, E_A(K_{BA}), Sig_B(T_B, N_B, A, N_A, E_A(K_{BA}))$
3. $A \rightarrow B : N_B, B, Sig_A(N_B, B)$

Protocol 4.23: X.509 three-pass authentication

There is no need in either this protocol or the two-pass protocol for both T_B and N_A to be included since either of them is enough for A to acquire freshness of K_{BA}. Indeed the standard states that T_B may be set to zero which, as pointed out by Burrows *et al.* [76], makes T_B completely redundant. The standard also states that T_A need not be checked in message 1 either.

In the first (1988) version of the X.509 standard, the field B was absent from the third message of Protocol 4.23. A consequence of this was that if T_A was not used for freshness then the protocol could be attacked, since B is not able to check that message 3 is part of the same protocol run. Specifically C can replay an old first message from A to B and, in order to complete the protocol, need only obtain A's signature on the challenge received in message 2. C can now obtain such a signature by engaging in a protocol run with A as the initiator. This attack is detailed by both Burrows *et al.* [76] and I'Anson and Mitchell [153].

4.3.5 TLS Protocol

The Transport Layer Security (TLS) protocol [101] is standardised by the Internet Engineering Task Force to provide a complete security service for applications running over a reliable communications infrastructure such as the TCP/IP set of communications protocols. Unlike many of the examples in this book, the TLS protocol is a 'real-world' protocol that includes extensive practical details and is widely implemented. TLS is the next generation of the well-known Secure Sockets Layer (SSL) protocol originally developed by Netscape Communications and implemented today in most popular web browsers. In practice TLS or SSL is often employed using only unilateral entity authentication due to the lack of a universal public key infrastructure; certificates for server public keys are usually available but at present clients

Petitioner Samsung Ex-1036, 0147

rarely possess one. Therefore many applications rely on a shared secret (or password) to supplement the server side authentication.

It is instructive to examine a specification such as the TLS protocol to see all the details necessary to describe a practical protocol suitable for a variety of applications. The TLS protocol describes how to secure application data in its *Record Protocol* as well as specifying key establishment in its *Handshake Protocol*. Some of the issues that are covered in the more than 75 pages of specification include:

- negotiation of cryptographic algorithms to be used;
- negotiation of protocol version to be used;
- optional use of compression for both the Record and Handshake Protocols;
- specific hash function constructions;
- generation of random values for key generation;
- resumption of sessions with session key renewal.

Key establishment with the TLS Handshake Protocol itself allows a number of options which may be regarded as quite different protocols. Protocol 4.24 is a simplified version of one possible protocol which transports the 'pre-master secret', PMK, from the client A to the server B. The TLS standard specifies use of RSA public key encryption for this protocol, although other algorithms could be added to the available protocols. The digital signature algorithm can be either RSA or DSS. As usual, we omit mention of certificates in our description.

---

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : N_B$
3. $A \rightarrow B : E_B(PMK), Sig_A(Mess_Seq_1), \{Mess_Seq_2\}_{K_{AB}}$
4. $B \rightarrow A : \{Mess_Seq_3\}_{K_{AB}}$

Protocol 4.24: Simplified TLS key transport protocol

The session key K_{AB} is calculated as $K_{AB} = MAC_{PMK}(N_A, N_B)$. Notice that although PMK is transported from A to B, the calculation of K_{AB} turns the protocol into key agreement. In fact the TLS protocol specifies that four different session keys must be derived from the pre-master secret: one for encryption and one for data integrity (MAC calculation) in each direction.

The message sequences $Mess_Seq_1$, $Mess_Seq_2$ and $Mess_Seq_3$ consist of hashes of the sequence of all previous exchanged messages up to that point using a hash function H. For example,

$$Mess_Seq_1 = H(N_A, N_B, E_B(PMK)).$$

Protocol 4.25 is a key agreement variant of TLS based on certified Diffie–Hellman keys in a suitable group generated by g. (Strictly this protocol should be in Chap. 5 but we include it here for continuity.)

Petitioner Samsung Ex-1036, 0148

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : N_B$
3. $A \rightarrow B : Sig_A(Mess_Seq_1), \{Mess_Seq_2\}_{K_{AB}}$
4. $B \rightarrow A : \{Mess_Seq_3\}_{K_{AB}}$

Protocol 4.25: Simplified TLS key agreement protocol

This time the pre-master secret is calculated as $PMK = g^{ab}$ where a and b are private Diffie–Hellman keys for which the corresponding public keys are g^a and g^b, respectively. As before $K_{AB} = MAC_{PMK}(N_A, N_B)$.

The key transport version of the TLS protocol was formally analysed in a simplified version by Paulson [265, 264] using an interactive theorem prover for higher order logic (see Sect. 2.5.7). He was able to prove that this abstract version of the protocol is secure within the formal model. However, Paulson commented that the analysis would be more straightforward if the identity of A was included in the public-key-encrypted part of message 3 in Protocol 4.24, so that $E_B(PMK)$ is replaced by $E_B(A, PMK)$.

An informal analysis of the predecessor of TLS, SSL 3.0, was made by Wagner and Schneier [325]. There were two issues that they raised as being of concern, but neither of these relates directly to the key establishment protocol itself but rather to its interaction with the Record Protocol. Both of these problems have, in any case, been avoided in the TLS protocol specification.

Another formal analysis by Mitchell *et al.* [242], using the tool Murϕ (see Sect. 2.5.2), found anomalies in the negotiation process of SSL 3.0, similar to some found by Wagner and Schneier. As in Paulson's study, Mitchell *et al.* examined the key transport version of the protocol but also included in their study the SSL protocol variant in which a session is resumed using the same pre-master secret but with new random values so that the session key is new. An interesting approach taken by Mitchell *et al.* was to strip down the Handshake Protocol and examine how it fails when different parts are missing.

Steiner *et al.* [306] have shown how to integrate password-based key establishment into TLS. This allows users who possess only a short secret (a password) with the server to set up a secure connection without allowing an eavesdropper the opportunity to mount a brute force guessing attack to find the password. Such protocols are considered in detail in Chap. 7.

4.3.6 Beller–Chang–Yacobi Protocols

Beller, Chang and Yacobi (BCY) [34, 35, 36], and Beller and Yacobi (BY) [37] proposed hybrid protocols using a combination of asymmetric and symmetric cryptographic algorithms. These protocols are designed to satisfy the requirements of the mobile communications environment. They are intended to pro-

Petitioner Samsung Ex-1036, 0149

vide security between a mobile station and a base station of the fixed network, rather than to provide end-to-end security between mobile users.

There are at least two requirements in addition to those usually needed for authentication and key establishment protocols.

- The computational load on the mobile station must be minimised, even at the expense of increased load on the base station.
- The identity of the mobile station must remain hidden from the adversary.

The protocols of Beller *et al.* were critically examined by Carlsen [84], who identified some possible attacks and suggested protocol modifications to avoid them. He also pointed out an inherent shortcoming of their protocols. Although these protocols hide the identity of an initiating mobile station, the dual requirement of hiding the identity of the responding station remained unsolved.

The protocols of Beller *et al.* rely on a public key cryptosystem for which encryption is particularly efficient, at least in comparison to other public key cryptosystems. The specific public key cryptosystem employed is due to Rabin [272], in which encryption and decryption are tantamount, respectively, to modulo squaring and extracting a modulo square root (MSR). Instead of showing the mathematical details of the MSR algorithms, we shall continue to use our more general notation in describing the protocols of Beller *et al.* (hereafter referred to as the MSR protocols). The MSR protocols consist of three variants with different complexity and security features.

MSR Protocol

In the following, the notation SC_B is a structure known as the *secret certificate* of the mobile station, B, which is issued by a trusted central authority. This certificate can be checked by anyone using the public key of the central authority in order to verify the mobile's identity. Unlike a usual public key certificate, this certificate must be kept secret from all other mobile users and eavesdroppers, because it is all that is required to masquerade as B. Protocol 4.26 shows the basic MSR protocol [36].

1. $A \rightarrow B : A, K_A$
2. $B \rightarrow A : E_A(K_{AB}), \{B, SC_B\}_{K_{AB}}$

Protocol 4.26: Basic MSR protocol of Beller, Chang and Yacobi

Upon receiving the base A's public key K_A, the mobile uses it to encrypt the session key K_{AB}, and sends the encrypted message to A. The mobile also sends its identity and secret certificate encrypted under K_{AB} to authenticate K_{AB} to the base. The symmetric encryption with K_{AB} in message 2 is of

Petitioner Samsung Ex-1036, 0150

negligible computational effort compared to the public key encryption in the same message; therefore the computational effort at the mobile is effectively limited to that of modulo squaring of the session key. Carlsen [84] identified two security weaknesses in Protocol 4.26. The first of these weaknesses appears to have been recognised as early as 1993 by Beller *et al.* [36] themselves.

- The public key of A is uncertified, thereby allowing anyone to masquerade as A.
- It is not possible for A to differentiate between a new run of the protocol and one where messages from an old run are replayed by a malicious adversary. At the least this may allow the adversary to transfer connection charges to B. In addition replay of an old compromised session key then allows the adversary to masquerade as B.

Improved MSR (IMSR) Protocol

The improved MSR protocol of Beller *et al.* [36], IMSR, overcomes a major weakness of MSR by using a public key certificate of the base station. This results in a twofold increase in the computational complexity as compared to Protocol 4.26 since the mobile now calculates an additional modulo square to verify the base's certificate on receiving message 1.

Apart from this feature it is identical to the basic MSR protocol, and therefore does not address the problem of replay. Carlsen [84] recognised this and suggested an 'improved IMSR' protocol which includes a challenge–response mechanism to allow B to detect a session key replay as shown in Protocol 4.27. (He also adds an expiration time to the public key certificate of A, to allow for checks on the certificate's validity, while at the same time deleting A's identity from the certificate for purposes of anonymity. The effect of this latter change is that base station 'impersonation attacks' become possible, as pointed out by Mu and Varadharajan [243]. As usual, this public key certificate is omitted from our description.)

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : E_A(K_{AB}), \{N_A, B, SC_B\}_{K_{AB}}$

Protocol 4.27: Improved IMSR protocol of Carlsen

Upon receiving the final message, A decrypts it using the session key K_{AB}, and checks that the value N_A is the same as the nonce sent in message 1. Curiously, although Carlsen clearly identifies the problem of replay, his suggested improvement does not really overcome it. In the above protocol, if K_{AB} is compromised an adversary can obtain SC_B, and thus freely masquerade as B.

Petitioner Samsung Ex-1036, 0151

Beller *et al.* [35] have mentioned the security threat posed to the IMSR protocol by an adversary who obtains the session key; however, they do not treat the problem of replay as such. They suggested two methods to protect the certificate of B against a compromised session key. One of these, called the *split-certificate method*, is to have the mobile send at least half of its certificate encrypted together with K_{AB} under the base's public key. The other, called the *split-key method*, is to divide K_{AB} into two sub-keys, one of which is used to encrypt the protocol message that authenticates B to A, and the other as the session key proper. Both these methods can also be used to overcome the weakness of Protocol 4.27.

Beller–Yacobi Protocol

In a separate publication, Beller and Yacobi [37] suggest a further variation on the IMSR protocol. Like the MSR+DH protocol discussed below, the Beller–Yacobi (BY) protocol employs a public key for the mobile as well as the base. The mobile's private key is used to implement digital signatures using the ElGamal algorithm [106]. The main reason for choosing this algorithm is that most of the computations required for signature generation can be executed prior to choosing the message to be signed. This means that it is easy for the mobile processor to do most of its work off-line, during idle time between calls. The basic structure of Protocol 4.28 is similar to Protocol 4.26. The main difference is in the last two messages which implement a challenge–response mechanism based on digital signatures.

1. $A \rightarrow B : A, K_A$
2. $B \rightarrow A : E_A(K_{AB})$
3. $A \rightarrow B : \{N_A\}_{K_{AB}}$
4. $B \rightarrow A : \{B, K_B, Cert(B), Sig_B(N_A)\}_{K_{AB}}$

Protocol 4.28: Beller–Yacobi protocol

In the third message, A sends a nonce N_A encrypted using K_{AB}. B then returns N_A signed using his private key together with his identity, public key and certificate $Cert(B)$, all encrypted under K_{AB}. Finally, A decrypts this message and verifies the signature on N_A.

We now present a potential attack on Protocol 4.28 [62]. Although the attack makes quite strong assumptions, it may be taken seriously because it indicates a flaw in the protocol design. We understand that the same attack was found independently by the original authors subsequent to the protocol's publication. The adversary, C, must be a legitimate user known to A. Further, C needs to be able to set up simultaneous sessions with both A and B. (C could be a rogue mobile and base station in collusion.) In Attack 4.5, C is able to convince B that C is A.

Petitioner Samsung Ex-1036, 0152

1. $A \rightarrow C_B : A, K_A$
2. $C_B \rightarrow A : E_A(K_{AB})$
3. $A \rightarrow C_B : \{N_A\}_{K_{AB}}$
1'. $C \rightarrow B : C, K_C$
2'. $B \rightarrow C : E_C(K'_{AB})$
3'. $C \rightarrow B : \{N_A\}_{K'_{AB}}$
4'. $B \rightarrow C : \{B, K_B, Cert(B), Sig_B(N_A)\}_{K'_{AB}}$
4. $C_B \rightarrow A : \{B, K_B, Cert(B), Sig_B(N_A)\}_{K_{AB}}$

Attack 4.5: Attack on Beller–Yacobi protocol

The essence of the attack is that C starts a parallel session with B in order to obtain B's signature on A's challenge N_A. At the end of the attack, A accepts K_{AB} as a session key with B, whereas in fact it is shared with C. The session started between C and B can be dropped after the receipt of message 4'. Note that message 3 must precede message 3', and message 4' must precede message 4; the remaining messages may overlap each other.

There is a simple way to alter the protocol so as to avoid the attack [62]. Essentially the change is to have B sign the new session key K_{AB} when it is first sent to A, in message 2, together with the challenge N_A, which guarantees its freshness. The key must have its confidentiality protected by a suitable one-way hash function h, but the use of such a function is a standard practice in most digital signature schemes. Since K_{AB} is now authenticated in message 2, message 4 is redundant and message 3 is used simply for B to verify that A has received the key. Protocol 4.29 shows the revised version.

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : E_A(K_{AB}), \{B, K_B, Cert(B)\}_{K_{AB}}, Sig_B(h(A, B, N_A, K_{AB}))$
3. $A \rightarrow B : \{N_A\}_{K_{AB}}$

Protocol 4.29: Improved Beller–Yacobi protocol

Comparison with Protocol 4.28 shows that Protocol 4.29 is no more costly in either computational or communications requirements than the original. Therefore it appears to be just as suitable as the original for the situation where B has limited computing power.

Beller–Yacobi MSR+DH Protocol

Beller and Yacobi also proposed an extended version of the IMSR protocol which incorporates the Diffie–Hellman key exchange [102]. (The Diffie–Hellman and many related protocols are discussed in detail in Chap. 5.) A

Petitioner Samsung Ex-1036, 0153

major improvement is that now the mobile terminal has a public key which means that it no longer needs to reveal its permanent secret to the base.

In this protocol the base station A has two public keys: the Diffie–Hellman key and a public key used for encryption in the modular square root system. The mobile station B has one public key. Carlsen [84] has also suggested an 'improved MSR+DH' protocol by making similar modifications to those carried out in the improved MSR protocol. Protocol 4.30 shows the improved MSR+DH protocol [84].

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : E_A(x), \{N_A, B\}_x$

Protocol 4.30: Carlsen's improved Beller–Chang–Yacobi MSR + DH protocol

The static Diffie–Hellman key S_{AB} (see Sect. 5.2) is used to derive the session key as $K_{AB} = \{x\}_{S_{AB}}$. Although the security of the MSR+DH protocol appears far improved over the other MSR variants, it carries a computational price. Now both parties need to calculate a full modular exponentiation at session set-up leading, as per the calculations of Beller *et al.*, to a 100 times increase in the required computing power.

4.3.7 TMN Protocol

One of the earliest key establishment protocols designed for use in a mobile environment was that of Tatebayashi, Matsuzaki and Newman [317], which has widely become known as the TMN protocol. A favourite with protocol analysts due to its many vulnerabilities, we include it for its historical importance. The principals are two mobile stations A and B who wish to exchange a session key to provide end-to-end security, and a server S with whom they share distinct long-term secrets. The design takes account of the limitations in mobile station computational ability by requiring the mobile stations only to encrypt with short RSA [274] public exponents. A number of attacks have been published on the TMN protocol, some of which rely on the specific cryptographic algorithms used, and others exploiting problems in the message structures [181].

The TMN paper [317] includes two protocols. The first protocol (called KDP1) contains no authentication information and was found by the designers to be vulnerable to certain attacks. As an improvement, Protocol 4.31 (called KDP2) was proposed. The field s_A is a shared secret value between S and A while T_A is a timestamp generated by A. The fields s_B and T_B are defined analogously. The session key K_{AB} is generated by B, while A generates a one-time key-encrypting key KEK. The second protocol message is simply a request from S for B to respond.

Petitioner Samsung Ex-1036, 0154

1. $A \rightarrow S : E_S(T_A, s_A, KEK)$
2. $S \rightarrow B : RSVP$
3. $B \rightarrow S : E_S(T_B, s_B, K_{AB})$
4. $S \rightarrow A : \{K_{AB}\}_{KEK}$

Protocol 4.31: Simplified TMN protocol (KDP2)

The encryption in message 4 is carried out using a symmetric cryptosystem. The identities of A and B are relevant to the intended meaning of messages 3 and 4, respectively, although they are not included within the encrypted fields of these messages. As a result, neither A nor B has any assurance of who else has the session key K_{AB}. Since S has a shared secret with both A and B, it is questionable whether the use of public key cryptography in the TMN protocol is justified.

A different attack, based on the algebraic properties of the encryption algorithms, has been found by Park *et al.* [260]. They propose a variant protocol with different algorithms, but its general structure is identical to Protocol 4.31, and consequently it suffers from the same weaknesses.

4.3.8 AKA Protocol

The Texas A&M University Anarchistic Key Authorization (AKA) protocol was proposed by Safford *et al.* [286]. The name of the protocol reflects the use of informally certified public keys in the style of PGP [344] although that does not seem to influence the design in any particular way. AKA employs an unusual mechanism to provide forward secrecy; instead of Diffie–Hellman key agreement, short-term RSA keys are used.

A number of variations on the basic idea are given by Safford *et al.* providing greater efficiency and flexibility. Protocol 4.32 is one simple version in which only B chooses a short-term public key; in other variants both parties choose and exchange a short-term public key. The public keys K_A and K_B are long-term public keys of A and B while K'_B is a short-term public key chosen by B for this session; encryption of M using K'_B is denoted by $E'_B(M)$.

1. $A \rightarrow B : K_A$
2. $B \rightarrow A : K_B, K'_B$
3. $A \rightarrow B : E'_B(N_A, A)$
4. $B \rightarrow A : E_A(N_B)$
5. $B \rightarrow A : E_A(Sig_B(N_A))$
6. $A \rightarrow B : E'_B(H(N_B))$

Protocol 4.32: AKA protocol

Petitioner Samsung Ex-1036, 0155

The session key is defined as a function of the shared secret $N_A \oplus N_B$. The point of using the temporary public key can be seen if we consider the result of a compromise of either A's or B's long-term private key. Since the private key corresponding to K'_B is not compromised then N_A cannot be recovered and so forward secrecy is provided.

Abadi [1] has shown that this protocol is vulnerable to an attack due to the lack of sufficient authenticating information inside the signature of B in message 5. This means that an adversary C can interleave a run of the protocol with B with another run with A in which C masquerades as B. Attack 4.6 shows a specific attacking run: C replaces B's short-term public key with a new short-term key K'_C. C can use the signature of message 5 to convince A that C is in fact B. C simply aborts the run with B after capturing message 6 from A.

1. $A \rightarrow C_B : K_A$
1'. $C_A \rightarrow B : K_A$
2'. $B \rightarrow C_A : K_B, K'_B$
2. $C_B \rightarrow A : K_B, K'_C$
3. $A \rightarrow C_B : E'_C(N_A, A)$
3'. $C_A \rightarrow B : E'_B(N_A, A)$
4'. $B \rightarrow C_A : E_A(N_B)$
4. $C_B \rightarrow A : E_A(N_C)$
5'. $B \rightarrow C_A : E_A(Sig_B(N_A))$
5. $C_B \rightarrow A : E_A(Sig_B(N_A))$
6. $A \rightarrow C_B : E'_C(H(N_C))$

Attack 4.6: Abadi's attack on AKA protocol

The result of the attack is that A and C share the secret $N_A \oplus N_C$ but A believes that this secret is shared with B. Abadi points out that this attack can be prevented by including more fields in the signature in message 5; specifically the nonce N_B and the identities of A and B should be included.

A similar attack applies to the other AKA variants. In the versions where both A and B choose a short-term key, the adversary can replace them both with new short-term keys and obtain all encrypted information before sending it on re-encrypted with the expected key.

4.3.9 Comparison of Key Transport Protocols

Table 4.4 summarises the main features of the key transport protocols examined in this chapter. The protocols in the ISO/IEC 11770-3 standard have stood up well to scrutiny over a number of years, and the troubled history of the earlier X.509 protocols seems to have been avoided. The formal proofs

Petitioner Samsung Ex-1036, 0156

of mechanisms 1 and 4 is a good indication that these seem well-designed protocols that cater for a variety of requirements.

Table 4.4. Summary of major properties of key transport protocols using public key

Properties →	Key	Key	Key	Key	Attack	Security
↓ *Protocol*	control	freshness	auth.	conf.		proof
11770-3 Mechanism 1 (4.10)	*A*	*A*	*A*	No	No	No
11770-3 Mechanism 2 (4.11)	*A*	*A*+*B*	*A*+*B*	*B*	No	No
11770-3 Mechanism 3 (4.12)	*A*	*A*+*B*	*A*+*B*	*B*	No	No
11770-3 Mechanism 4 (4.14)	*B*	*A*+*B*	*A*+*B*	*A*	No	Yes
11770-3 Mechanism 5 (4.15)	*A*+*B*	*A*+*B*	*A*+*B*	*A*	No	No
11770-3 Mechanism 6 (4.16)	*A*+*B*	*A*+*B*	*A*+*B*	*A*+*B*	No	No
Blake-Wilson–Menezes (4.18)	*B*	*A*+*B*	*A*+*B*	No	No	Yes
Needham–Schroeder (4.19)	*A*+*B*	*A*+*B*	*A*+*B*	*A*+*B*	Yes	Yes
X.509 One-Pass (4.21)	*A*	*A*	*A*	No	Yes	No
X.509 Two-Pass (4.22)	*A*	*A*	*A*	No	Yes	No
X.509 Three-Pass (4.23)	*A*+*B*	*A*+*B*	No	No	No	No
TLS Key Transport (4.24)	*A*	*A*+*B*	*A*+*B*	*A*+*B*	No	Yes
(I)MSR (4.27)	*B*	*B*	*A*+*B*	*A*	Yes	No
TMN (4.31)	*B*	*B*	No	No	Yes	No
AKA (4.32)	*A*+*B*	*A*+*B*	No	No	Yes	No

The TLS protocol, together with its predecessor SSL, is undoubtedly the most widely implemented of the protocols examined in this chapter. It has received the benefit of more than one formal analysis although, as Paulson has remarked [265], the complete protocol is so complex that it is beyond the scope of existing methods to encompass all its variations. It is interesting that use of the session key within the TLS protocol means that it cannot be proven secure in the Bellare–Rogaway model.

4.4 Conclusion

The ISO/IEC 11770-3 standard specifies a variety of key transport protocols using asymmetric cryptography. Blake-Wilson and Menezes provided a proof for a simplified version of one of these which provides extra confidence in their security. It would be useful to have security proofs for each of the protocols.

Petitioner Samsung Ex-1036, 0157

The TLS protocol has been widely scrutinised and provides an alternative to the ISO/IEC protocols.

The other key transport protocols examined in this chapter all seem to have some problems, and they cannot be recommended over the ISO standard solutions. We think the study of these protocols can nevertheless be instructive in understanding typical mistakes in protocol design.

Recent attention in the research community has focused on key agreement rather than key transport. One reason for this is that key transport protocols do not usually provide forward secrecy, which is often possible with key agreement. Key agreement is the topic of the following chapter.

Petitioner Samsung Ex-1036, 0158

5

Key Agreement Protocols

5.1 Introduction

Key agreement, as the name implies, is a process in which principals cooperate in order to establish a session key. Amongst the class of public key protocols for key establishment without a server, key agreement has become much more popular than key transport in recent years. There is an intuitive feeling that key agreement is 'fairer' than key transport and can result in higher quality random keys than key transport. In addition, by basing key agreement on the Diffie–Hellman protocol, forward secrecy can often be achieved. We will consider these points further below. Notice that key agreement does not have to use public key cryptography but most examples do so. In this chapter we look only at key agreement based on public key cryptography; some examples of key agreement using symmetric cryptography are discussed in Chap. 3.

The definition of key agreement given by Menezes *et al.* [234] is as follows:

> A key agreement protocol or mechanism is a key establishment technique in which a shared secret is derived by two (or more) parties as a function of information contributed by, or associated with, each of these (ideally) such that no party can predetermine the resulting value.

A similar definition is given in international standard ISO/IEC 11770-3 [162], although it insists that neither party can predetermine the shared secret. In this chapter we are concerned with key agreement between any two principals A and B. The general format of such protocols requires each principal to select an independent input to the key. For our two principals these will be denoted r_A and r_B respectively. The principals will then send each other messages depending on r_A and r_B, and possibly depending on other values too. There are usually two stages to forming the session key.

1. The random inputs r_A and r_B, and possibly the long-term public/private keys, are combined to form a *shared secret*, Z_{AB}.

Petitioner Samsung Ex-1036, 0159

2. The session key K_{AB} is formed from Z_{AB}, and possibly other inputs, using a *key derivation function.*

In different protocols the key derivation function will generally be different. A typical key derivation function will be a one-way hash function of the shared secret and other data such as an algorithm identifier for the session key, a counter and public information about A and B. A generic key derivation function, known as KDF1, is specified in the IEEE P1363-2000 standard [154]; although KDF1 does specify the hash function to be used, the inputs to the function are left open.

The plan for the rest of the chapter is as follows. Before looking at specific key agreement protocols we consider some of the special properties and attacks that can apply to them. Of course all the general attacks on key establishment protocols discussed in Chap. 1 are relevant too.

Section 5.2 looks at the basic Diffie–Hellman protocol and emphasises its properties and limitations. Then in Sect. 5.3 we examine in detail a set of protocols based on Diffie–Hellman which are known as the MTI protocols. This set of protocols was designed relatively early and serves to illustrate many of the properties and potential attacks on key agreement protocols. Section 5.4 includes a number of more recent protocols whose exchanged messages are identical to those of Diffie–Hellman. Extra information, particularly public and private keys, is used in the calculation of the shared secret. Section 5.5 is devoted to a set of protocols that add extra information into the exchanged messages rather than into the definition of the shared secret. Identity-based key agreement is the subject of Sect. 5.6. These protocols are designed to replace a certified public key with information about the identity of the principals involved. Some key agreement protocols have been designed to make the computational effort required of one principal much less than that of the other. These protocols, suitable for devices with limited computational power, are covered in Sect. 5.7. International standard ISO/IEC 11770-3 specifies key agreement protocols in an abstract format. These are summarised in Sect. 5.8. Up to this point all the detailed protocols have been described in the setting of the multiplicative group of integers modulo a prime p. In Sect. 5.9 we list some alternative groups that have been proposed for the Diffie–Hellman protocol. Finally we look at some key agreement protocols that do not use the Diffie–Hellman technique in Sect. 5.10

5.1.1 Key Control

One potential benefit of key agreement is that each principal does not have to rely on any other party to generate appropriate keys. As long as neither party is malicious, it can often be guaranteed that the session key is sufficiently random if at least one of the principals is able to generate sufficiently random inputs. A related benefit is that principals can often be sure that the session key is fresh by ensuring that their own input is fresh. For this to be true

Petitioner Samsung Ex-1036, 0160

neither of the principals must be able to force the key to be any chosen value, otherwise one party could force use of an old key.

Key control is a term used to describe the extent to which principals have the ability to choose or influence the value of the shared key (or session key). As expressed in the definitions of key agreement given above, it is usually desired that neither principal can control the shared secret value.

In most practical situations one party will receive the random input of the other party before it has had to reveal anything about its own random input. This gives that party an 'advantage' in that it can effectively choose a number of bits of the session key. Mitchell *et al.* [239] have pointed out that, by choosing about 2^s random values, the party with the advantage can effectively choose any s bits of the key by generating new keys until the desired bits occur. Although s will typically be much shorter than the total key length, it is important to be aware of this possibility in assessing the properties of a key agreement protocol.

Mitchell *et al.* point out that such an attempt to control the key can be prevented by ensuring that both parties fix their random input before information about the other party's input is known. Ways to achieve this include strict use of timeouts and use of a third party, but most reasonable seems to be to have the first party send a hash of its random input as a *commitment* which will be opened in a later message after the second party's random element is received. A major drawback of this approach is that an extra message is required in the protocol.

A lack of strict key control applies to most published key agreement protocols. At present designers do not seem to be concerned enough about it to propose countermeasures.

5.1.2 Unknown Key-Share Attacks

Unknown key-share attacks are applicable in a security model that allows malicious insiders. The aim of the adversary C is to make one principal, say A, believe that the session key is shared with C when it is in fact shared with a different principal, B. The adversary need not, and usually does not, obtain the session key. Nevertheless, such an attack is profitable to the adversary in an application where B will deliver some information of value (such as electronic cash) to principal A. Since A believes the session key is shared with C, credit for this deposit will rest with C. (This means that this key is not good to establish credit in the sense defined in Sect. 2.3.5.)

Unknown key-share attacks seem to have first been described by Diffie *et al.* [103]. Their importance is somewhat controversial, since there are some general methods that can be used to avoid them. Furthermore, the assumptions made in many of the proposed attacks are rather unusual. A common scenario is for the adversary to obtain a public key certificate that has the same public key value as another principal. (The adversary will not know the corresponding secret key.)

Petitioner Samsung Ex-1036, 0161

Many protocols have been shown to be vulnerable to this type of attack (specific examples will be mentioned below). There are a number of methods that can help to defeat the attack.

- Certifiers of public keys should ensure that each entity is in possession of the corresponding private key before a certificate is issued. Some authors assume that this precaution is always taken and do not regard an unknown key-share attack as valid if the adversary does not know the private key corresponding to the certified public key.
- Key confirmation can often defeat the attack. The confirmation messages should include the identities of both principals so that the key is confirmed to be held by specific claimed entities, rather than known only to some unidentified party.
- A general method to ensure that unknown key-share attacks do not apply is to include both principal identities within the key derivation function. As long as the function used is collision resistant then A, if she believes the key is shared with C, will not derive the same session key as B, who believes the session key is shared with A. Somewhat surprisingly, Blake-Wilson and Menezes [46] deprecate this method of avoiding unknown key-share attacks on the grounds that the requirements of key derivation functions have not been widely studied.

5.1.3 Classes of Key Agreement

The most well-known technique used in key agreement protocols is Diffie–Hellman key exchange [102]; indeed sometimes key agreement is even used synonymously with the Diffie–Hellman technique. There are some special advantages of basing key agreement on Diffie–Hellman.

- Most of the protocols can be generalised to work in any Abelian (commutative) group. This allows a flexible choice of groups, including some that are particularly efficient in terms of computational and storage requirements.
- Many protocols based on Diffie–Hellman have the forward secrecy property, which is costly to achieve any other way. (However, several examples in this chapter show that basing a protocol on Diffie–Hellman does not guarantee that forward secrecy is achieved.)

It should be remembered that there are other ways of designing key agreement protocols apart from using the Diffie–Hellman primitive. A widely used alternative is to use a one-way function of user inputs to derive the session key. There can be advantages in this approach too.

- There can be computational savings over Diffie–Hellman by reducing (or eliminating) the number of expensive exponentiations.
- Keys can be guaranteed to be random even if one input becomes known – a property lacking in Diffie–Hellman-based protocols.

Petitioner Samsung Ex-1036, 0162

5.2 Diffie–Hellman Key Agreement

Diffie–Hellman key agreement was published in 1976 [102][1]. The elegant and simple construction has been the basis for a vast range of protocols, but on its own lacks any authentication. Later in the chapter we examine different ways to add authentication to produce more effective key agreement protocols. In this section we will introduce Diffie–Hellman and mention some protocols in which one or both of the Diffie–Hellman inputs are fixed, and which therefore lack some of the benefits of the technique.

In the basic Diffie–Hellman protocol, two principals A and B agree publicly on an element g that generates a multiplicative group G. They then select random values, r_A and r_B respectively, in the range between 1 and the order of G. A calculates $t_A = g^{r_A}$ and B calculates $t_B = g^{r_B}$ and they exchange these values as shown in Protocol 5.1. The shared secret is $Z_{AB} = g^{r_A r_B}$. This value can be calculated by both A and B due to the homomorphic property of exponentiation: $Z_{AB} = t_A^{r_B} = t_B^{r_A}$.

Shared information: Generator g of G.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{t_B}$	$Z_{AB} = t_A^{r_B}$

Protocol 5.1: Diffie–Hellman key agreement

Diffie–Hellman key agreement was originally described in the multiplicative group $\mathbb{Z}_p^*$ of non-zero integers modulo a large prime p. It has become more usual to define the group G in which the protocol takes place to be a subgroup of $\mathbb{Z}_p^*$ of prime order q. (Note that the order of $\mathbb{Z}_p^*$ is $p-1$ which cannot be prime.) There are two potential advantages of this. Firstly, several attacks (to be described shortly) can be avoided. Secondly, the size of the group G can usually be made much smaller than $\mathbb{Z}_p^*$, which results in computational savings. Typical sizes in use today are 1024 bits for the length of p and 160 bits for the length of q. Several other algebraic groups have been proposed as the setting for Diffie–Hellman key exchange. Examples are given in Sect. 5.9. In particular, elliptic curve groups are popular today. To enable

[1] It seems that the idea was invented previously in 1974 but was not made public at that time [107].

Petitioner Samsung Ex-1036, 0163

a uniform presentation all the Diffie–Hellman-based protocols in this chapter are described with G as a subgroup of $\mathbb{Z}_p^*$, even in cases when the protocol designers have used other groups.

Diffie–Hellman is secure against passive eavesdroppers on the widely accepted assumption that it is infeasible to recover Z_{AB} from the values of g^{r_A} and g^{r_B}. This is often referred to as the *Diffie–Hellman assumption*. Breaking Diffie–Hellman is clearly no harder than solving the discrete logarithm problem, since by finding the discrete logarithm of either of the exchanged values the Diffie–Hellman key can be found. It is a long standing open question as to whether the Diffie–Hellman problem is really as hard as the discrete logarithm problem despite considerable research on the topic [225]. In formal analysis of Diffie–Hellman-based protocols it is usually possible only to obtain a proof based on the generally stronger *Decision Diffie–Hellman* (DDH) assumption. This states that it is a hard problem to distinguish between a genuine Diffie–Hellman triple (g^x, g^y, g^{xy}) and a triple (g^x, g^y, g^z) for random exponents x, y and z.

The fundamental limitation of the basic Diffie–Hellman protocol is that there is no authentication of the messages sent. This is illustrated by the well-known 'man-in-the-middle' attack in which the adversary C masquerades as B to A and masquerades as A to B. Attack 5.1 shows how both A and B complete a normal run, but both share keys with C, namely $g^{r_A r_C}$ and $g^{r_C r_B}$ respectively.

A		C		B
$r_A \in_R \mathbb{Z}_q$				
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_C \in_R \mathbb{Z}_q$		
		$t_C = g^{r_C}$	$\xrightarrow{t_C}$	$r_B \in_R \mathbb{Z}_q$
	$\xleftarrow{t_C}$	$Z_{AC} = t_A^{r_C}$	$\xleftarrow{t_B}$	$t_B = g^{r_B}$
$Z_{AB} = t_C^{r_A}$		$Z_{CB} = t_B^{r_C}$		$Z_{AB} = t_C^{r_B}$

Attack 5.1: Attack on basic Diffie–Hellman

The shared secret $g^{r_A r_B}$ derived in basic Diffie–Hellman is called an *ephemeral* Diffie–Hellman key since it depends only on randomly chosen values and lasts only until the session key is derived. In contrast, if A and B exchange their respective public keys $y_A = g^{x_A}$ and $y_B = g^{x_B}$ then both can calculate the value $S_{AB} = g^{x_A x_B}$. This is often called a *static* Diffie–Hellman key, since it does not depend on any random input. Use of a static Diffie–Hellman key on its own to derive session keys is not satisfactory since it does not allow generation of new session keys. One method for design of robust

Petitioner Samsung Ex-1036, 0164

protocols based on Diffie–Hellman has revolved around how best to mix the ephemeral and static values in such a way as to obtain the desired properties.

The static Diffie–Hellman key can be used in a very simple protocol by incorporating it with a fresh value in a one-way function. Suppose that K is a fixed key derived from S_{AB}. Rueppel and van Oorschot [279] note that K could be used to transport a random session key by sending $\{K_{AB}\}_K$, or alternatively the session key could be defined as $K_{AB} = MAC_K(r)$ where r is a sequence number or nonce sent in cleartext. Such protocols could provide implicit key authentication, but fail to provide some of the typical advantages of key agreement. In particular there is no joint key control and neither forward secrecy nor resistance to key compromise impersonation are provided.

The notation used in this section is included in Table 5.1 and will be used throughout this chapter for describing Diffie–Hellman-based protocols. We will also continue to use notation introduced in Table 4.1.

Table 5.1. Notation used throughout Chap. 5

p	A large prime (usually at least 1024 bits).
q	A prime (typically of 160 bits) with $q \mid p-1$.
G	A subgroup of $\mathbb{Z}_p^*$. G is often a subgroup of order q, but sometimes is equal to $\mathbb{Z}_p^*$.
g	A generator of G.
r_A, r_B	Random integers, typically of the same size as the order of G, chosen by A and B respectively.
t_A, t_B	Ephemeral public keys: $t_A = g^{r_A}$ and $t_B = g^{r_B}$. All computations take place in $\mathbb{Z}_p$.
x_A, x_B	The private long-term keys of A and B respectively.
y_A, y_B	The public keys of A and B: $y_A = g^{x_A}$ and $y_B = g^{x_B}$. These public keys will have to be certified in some standard way that we do not consider here.
Z_{AB}	The shared secret calculated by the principals.
K_{AB}	The derived session key.
S_{AB}	The static Diffie–Hellman key of A and B: $S_{AB} = g^{x_A x_B}$.
N_A, N_B	Nonces chosen by A and B respectively.
$H(.)$	A one-way hash function. Certain protocols may require specific properties and may specify particular functions.
$x \in_R X$	The element x is chosen uniformly at random from the set X.
$F \stackrel{?}{=} G$	Verify that F and G evaluate to the same value.

Petitioner Samsung Ex-1036, 0165

5.2.1 Small Subgroup Attacks

Small subgroup attacks seem to have been first recognised by Vanstone (attributed in later descriptions [324, 201]). The idea of small subgroup attacks is to exploit the structure of the group G in which Diffie–Hellman key agreement takes place. If the order of G is composite then G will have subgroups; furthermore, if g^{r_A} lies in some subgroup then so does $g^{r_A r_B}$. The idea of the small subgroup attack is to force the shared secret to lie in a small set. Then there will be relatively few possible values available for the session key which will help the adversary and possibly allow exhaustive search.

One way to avoid small subgroup attacks is to make G of prime order. This is frequently done by choosing g to have prime order q where $q|p-1$. In this case the only proper subgroup of G consists of the single identity element. In order to avoid attacks it may still be necessary to check that received elements do lie in the correct group (and are not equal to the identity). See Sect. 5.3.1 for a specific example of a small subgroup attack.

A related type of attack, first proposed by Lim and Lee [206], exploits small subgroups that are outside the subgroup generated by G. In these attacks the adversary, who is an insider, sends the recipient a value that should be in the group G, but is mixed with a value outside G. This allows the adversary to gain information about the value of the victim's private key. This type of attack can still work when G is a prime order subgroup of $\mathbb{Z}_p^*$. It can be prevented if the principals ensure that all received elements are inside G. Further details, including a specific example, are given in Sect. 5.3.3.

Zuccherato [345] has summarised the situations in which small subgroup attacks are a threat, and proposed different ways that they can be avoided.

5.2.2 ElGamal Encryption and One-Pass Key Establishment

Before considering protocols in which both principals contribute a random value, we first look at the situation where only one principal does so. These protocols can be considered as half way between using static Diffie–Hellman keys and the inclusion of ephemeral keys. These protocols are useful when it is possible only to have communications in one direction; a typical application scenario would be secure electronic mail.

ElGamal encryption [106] was not conceived as a key establishment protocol, and yet we can view it in this manner. The sender A forms a shared secret using her random input r_A in combination with B's long-term y_B by calculating $Z_{AB} = y_B^{r_A}$. On receipt of an encrypted message and the ephemeral public key $t_A = g^{r_A}$, principal B is able to reconstruct $Z_{AB} = t_A^{x_B}$ and so decrypt the message. Evidently B receives no authentication regarding the session key and cannot even check the freshness of Z_{AB}; however, A does obtain implicit key authentication.

Protocol 5.2 shows one-pass key establishment as proposed by Agnew *et al.* [8]. The static Diffie–Hellman key is used together with a nonce k_A to

Petitioner Samsung Ex-1036, 0166

form the shared secret. The nonce k_A is sent encrypted from A to B using B's public key y_B: the ElGamal encryption algorithm is used for this purpose.

Shared information: Static Diffie–Hellman key, S_{AB}.

A		B
$r_A, k_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}, s_A = y_B^{r_A} \cdot k_A$		
	$\xrightarrow{t_A, s_A}$	$k_A = s_A / t_A^{x_B}$
$Z_{AB} = S_{AB}^{k_A}$		$Z_{AB} = S_{AB}^{k_A}$

Protocol 5.2: Agnew–Mullin–Vanstone protocol

The shared secret is $Z_{AB} = S_{AB}^{k_A}$. Implicit key authentication is provided for both parties since knowledge of either x_A or x_B is required to form Z_{AB}. However, B has no way of knowing that the shared secret is fresh. Entity authentication is not achieved for either party and neither is forward secrecy nor protection against key compromise impersonation. The encryption of k_A prevents the adversary from using a known Z_{AB} value to obtain S_{AB} and mounting a future active attack.

Nyberg and Rueppel [253, 255] investigated ways of incorporating message recovery into signatures based on the discrete logarithm. As an application of this technique they suggested to use ElGamal encryption of a signed session key as a one-pass key establishment protocol. Protocol 5.3 is their initial protocol [253]. If the protocol runs correctly then both A and B calculate the shared secret $Z_{AB} = g^{r_A x_B}$.

A		B
$r_A, k \in_R \mathbb{Z}_q$		
$Z_{AB} = y_B^{r_A}$		
$r = g^{r_A - k}, s = k - x_A r \bmod q$	$\xrightarrow{r, s}$	$Z_{AB} = (g^s y_A^r r)^{x_B}$

Protocol 5.3: Original Nyberg–Rueppel protocol

Later, Nyberg [252] pointed out that it is possible for an adversary who finds an old Z_{AB} value to replay the old r value and, by replacing the corresponding s with $s + u$, to establish a new key $K'_{AB} = K_{AB} \cdot y_B^u$. She therefore designed the enhanced Protocol 5.4. This employs a time varying parameter

Petitioner Samsung Ex-1036, 0167

which could be a counter. Neither of these protocols provides forward secrecy since knowledge of x_B allows the adversary to obtain Z_{AB} in the same way as B.

Shared information: Time varying parameter t.

A		B
$r_A \in_R \mathbb{Z}_q$		
$Z_{AB} = y_B^{r_A}$		
$r = g^{r_A}, r' = H(Z_{AB}, t)$		
$s = r_A - x_A r' \bmod q$	$\xrightarrow{r, s}$	$Z_{AB} = r^{x_B}$
		$r' = H(Z_{AB}, t)$
		$r \stackrel{?}{=} g^s y_A^{r'}$

Protocol 5.4: Revised Nyberg–Rueppel protocol

Other schemes for authenticated message exchange that are suitable for one-pass key establishment have been proposed by Horster *et al.* [150]. (The revised version of this paper [149] includes an attack on some of the schemes which was found by C.H. Lim.) Schemes for combined encryption and signature known as *signcryption*, due originally to Zheng [341], can also be used for this purpose.

In Sect. 5.4 we will examine several two-pass key agreement protocols. Most of these can be converted into one-pass key establishment protocols by replacing the random input of the receiving party with that principal's long-term secret. Blake-Wilson and Menezes [45] discuss this procedure and illustrate its application.

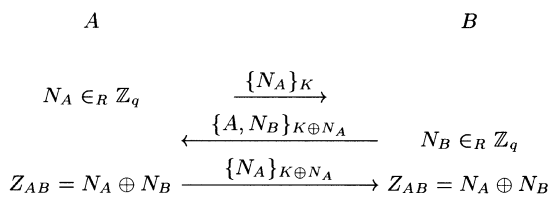
5.2.3 Lim–Lee Protocol Using Static Diffie–Hellman

Most key agreement protocols using Diffie–Hellman make use of ephemeral keys. We have already seen a one-pass protocol that uses the static key in Protocol 5.2 above. Lim and Lee [205] proposed a three-pass protocol using the static Diffie–Hellman key together with random inputs from both principals. This allows both parties to be sure that the key is fresh. In Protocol 5.5 the symmetric key K is a static key derived in some (unspecified) way from the static Diffie–Hellman key S_{AB}.

Lim and Lee suggest that the session key may be defined either as $K_{AB} = N_A \oplus N_B$ or as $K_{AB} = K \oplus N_A \oplus N_B$. As long as A and B both use random inputs this ensures the freshness of the key. However, with either of these choices for K_{AB} it is possible for B to completely control the session key

Petitioner Samsung Ex-1036, 0168

Shared information: Shared key K derived from S_{AB}.



Protocol 5.5: Lim–Lee protocol using static Diffie–Hellman

value. As is usual with protocols using only static keys, Protocol 5.5 does not provide even partial forward secrecy, since knowledge of one of the long-term keys is sufficient to find S_{AB}. Key compromise impersonation is possible too.

5.3 MTI Protocols

Matsumoto, Takashima and Imai (MTI) [224] showed in 1986 how to define three classes of authenticated key agreement protocols. They incorporate authentication into the Diffie–Hellman exchange in a very elegant manner by combining the long-term and ephemeral inputs into a single equation. Although many protocols have been designed using the same ideas as the MTI protocols they have a number of shortcomings in their original form. Nevertheless we look at them in some detail in this section for two reasons. Firstly, a detailed knowledge of these protocols will be very helpful in understanding the many protocols based on them. Secondly, they form a useful vehicle to explain many types of attacks on key agreement protocols.

MTI protocols are divided into three families: A, B and C. Protocol 5.6 shows the basic protocol of type A, denoted A(0). In the original specifications the subgroup G in which Diffie–Hellman exchange takes place is equal to the whole of $\mathbb{Z}_p^*$; as we will see below, a better choice is to make G a subgroup of prime order q. When both principals follow the protocol, the shared secret is $Z_{AB} = g^{x_A r_B + x_B r_A}$.

If we accept that either x_A or x_B is required to form Z_{AB} then implicit key authentication follows. Below we will mention a number of potential attacks on the scheme that show that this assumption is not always valid. As with any two-pass key agreement protocol, key confirmation is not achieved in the basic protocol.

Matsumoto *et al.* [224] showed that there is an infinite sequence of protocols with a related format. Protocol 5.7 shows the A(k) class of protocols, defined for any integer k. When both principals follow the protocol, the shared secret is $Z_{AB} = g^{x_A r_B x_B^k + x_B r_A x_A^k}$.

Petitioner Samsung Ex-1036, 0169

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$Z_{AB} = t_B^{x_A} y_B^{r_A}$	$\xleftarrow{t_B}$	$Z_{AB} = t_A^{x_B} y_A^{r_B}$

Protocol 5.6: MTI A(0) protocol

A		B
$r_A \in_R \mathbb{Z}_q$		
$z_A = g^{x_A^k r_A}$	$\xrightarrow{z_A}$	$r_B \in_R \mathbb{Z}_q$
		$z_B = g^{x_B^k r_B}$
$Z_{AB} = z_B^{x_A} y_B^{x_A^k r_A}$	$\xleftarrow{z_B}$	$Z_{AB} = z_A^{x_B} y_A^{x_B^k r_B}$

Protocol 5.7: MTI A(k) protocol

This sequence of protocols constitutes one of three classes of MTI protocols. They are all of the same basic format; they involve only two messages and achieve implicit key authentication but no key confirmation. Table 5.2 summarises the base ($k = 0$) protocols for each of the classes A, B and C; in this table z_A and z_B are the messages sent from A to B, and from B to A, respectively. The computational effort for protocols A(0) and B(0) is the same and consists of one exponentiation before message exchange and one multi-exponentiation to obtain the key. Protocol C(0) is slightly less complex, since an ordinary exponentiation only is required to obtain the key.

Table 5.2. Base protocols for each class of MTI key exchange

Type	z_A	z_B	Z_{AB}	Computed by A	Computed by B
A(0)	g^{r_A}	g^{r_B}	$g^{x_A r_B + x_B r_A}$	$z_B^{x_A} y_B^{r_A}$	$z_A^{x_B} y_A^{r_B}$
B(0)	$y_B^{r_A}$	$y_A^{r_B}$	$g^{r_A + r_B}$	$z_B^{x_A^{-1}} g^{r_A}$	$z_A^{x_B^{-1}} g^{r_B}$
C(0)	$y_B^{r_A}$	$y_A^{r_B}$	$g^{r_A r_B}$	$z_B^{x_A^{-1} r_A}$	$z_A^{x_B^{-1} r_B}$

For each sequence, to obtain the k'th protocol from the base protocol the exponent in z_A must be multiplied by x_A^k and that of z_B by x_B^k. The equation to calculate Z_{AB} by A must use $r_A x_A^k$ in place of r_A, and the equation for

Petitioner Samsung Ex-1036, 0170

B must use $r_B x_B^k$ in place of r_B. The exponents needed to calculate the shared secret for each protocol are shown in Table 5.3. Note that the protocols for negative k values are only well defined in the case that x_B is chosen to be invertible in G (for example, x_B must be prime to $p-1$ in the case that $G = \mathbb{Z}_p^*$). The extra computational effort required by the variants with parameter k is one exponentiation with exponent of size $|k|$.

Table 5.3. Exponent of shared secret for each MTI protocol

k	A(k)	B(k)	C(k)
-1	$x_A x_B^{-1} r_B + x_B x_A^{-1} r_A$	$x_A^{-1} r_A + x_B^{-1} r_B$	$x_A^{-1} r_A x_B^{-1} r_B$
0	$x_A r_B + x_B r_A$	$r_A + r_B$	$r_A r_B$
1	$x_A x_B r_B + x_B x_A r_A$	$x_A r_A + x_B r_B$	$x_A r_A x_B r_B$
2	$x_A x_B^2 r_B + x_B x_A^2 r_A$	$x_A^2 r_A + x_B^2 r_B$	$x_A^2 r_A x_B^2 r_B$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
k	$x_A x_B^k r_B + x_B x_A^k r_A$	$x_A^k r_A + x_B^k r_B$	$x_A^k r_A x_B^k r_B$

The MTI protocols provide an elegant and flexible approach to authenticated key agreement and have been the subject of considerable scrutiny in the research community. A number of attacks have been proposed which we examine now. It is worth noting that all known attacks can be avoided by use of suitable countermeasures.

5.3.1 Small Subgroup Attack on MTI Protocols

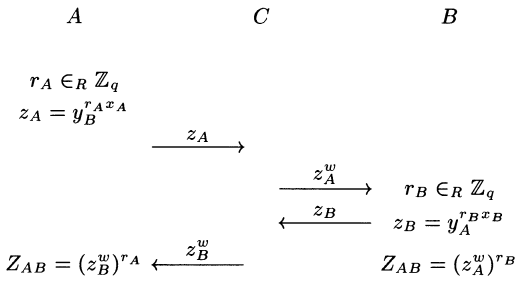
A small subgroup attack (see Sect. 5.2.1) applies to the MTI protocol sequence C(k) in the situation that the group G is the whole of $\mathbb{Z}_p^*$ as originally proposed. We suppose that the factorisation of $p-1$, which is the order of G, is known to the adversary. The attack is easiest in the case that $p-1$ has a very small factor r; let us write $w = (p-1)/r$. The attack works by raising the exchanged messages to the power w which moves these elements into the small subgroup of G of order r. Attack 5.2 shows a small subgroup attack on MTI protocol C(1). The adversary C plays in the middle between A and B. The shared secret calculated by A and B is:

$$Z_{AB} = g^{x_A r_B x_B r_A w}.$$

Since this is an element in the small subgroup, C can easily find the shared secret by exhaustive search and verify it from subsequent use in communications between A and B. Notice that in the extreme case when $r = 1$ it follows that $w = p-1$ and so the element received by both A and B is 1.

Petitioner Samsung Ex-1036, 0171

Shared information: Generator g of $\mathbb{Z}_p^*$. Small factor r of $p-1$. $w = (p-1)/r$.



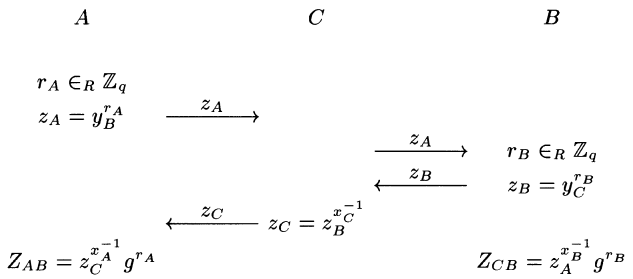
Attack 5.2: Small subgroup attack on MTI C(1) protocol

The small subgroup attack can be prevented by making G a subgroup of prime order q. In addition it is necessary to check that the received elements really are in the group G and are not equal to the identity.

5.3.2 Unknown Key-Share Attacks on MTI Protocols

Menezes *et al.* [233] discovered unknown key-share attacks on all the classes of MTI protocols. All the attacks require the adversary C to obtain a certificate for a long-term key y_C which is related to the public key of A by the equation $y_C = y_A^{x_C} = g^{x_A x_C}$. This means that C cannot know the private key $x_A x_C$ corresponding to the public key y_C. Attack 5.3 shows an unknown key-share attack on the MTI protocol B(0).

Shared information: Public key of C is $y_C = y_A^{x_C}$.



Attack 5.3: Unknown key-share attack on MTI B(0) protocol

Petitioner Samsung Ex-1036, 0172

The shared secret calculated by A is $(y_C^{r_B x_C^{-1}})^{x_A^{-1}} g^{r_A} = g^{r_B + r_A}$, while B calculates $(y_B^{r_A})^{x_B^{-1}} g^{r_B} = g^{r_A + r_B}$ to get the same value. Although A and B both have the same session key, A believes it to be shared with B, while B believes it to be shared with C. There are several ways to avoid the attack including:

- having certification authorities check that principals know the corresponding private key before issuing a public key certificate;
- including the principal identities in the key derivation function.

Menezes *et al.* [233] suggested to add to the message returned by B a hash of the key generated and B's randomised reply. A must check this on receipt of message 2 and abort the protocol if the check fails. Protocol 5.8 shows MTI B(0) modified in this way. Just and Vaudenay [175] pointed out that the protocol is still insecure if degenerate values such as 0 or 1 are accepted by A. For example, in Protocol 5.8 the adversary C can masquerade as B if A will accept the response $(u_B, h) = (0, H(0,0))$; the shared secret is calculated by A as 0 and is known to C.

Shared information: Hash function H.

A		B
$r_A \in_R \mathbb{Z}_q$		
$z_A = y_B^{r_A}$	$\xrightarrow{z_A}$	$r_B \in_R \mathbb{Z}_q$
		$z_B = y_A^{r_B}, Z_{AB} = z_A^{x_B^{-1}} g^{r_B}$
$Z_{AB} = z_B^{x_A^{-1}} g^{r_A}$	$\xleftarrow{z_B, h}$	$h = H(z_B, Z_{AB})$
$H(z_B, Z_{AB}) \stackrel{?}{=} h$		

Protocol 5.8: Modified MTI B(0) protocol

5.3.3 Lim–Lee Attack on MTI Protocols

Lim and Lee [206] devised ingenious attacks on interactive protocols that work in prime order subgroups. Their attack is applicable to MTI variants in which G is a prime order subgroup. As already mentioned, this is desirable in order to avoid small subgroup attacks on C(k) protocols and is also very beneficial in terms of the savings in computational requirements due to smaller exponent sizes.

The idea of the attack is that the adversary C will engage in a run of the protocol with the victim B. For B the run will seem normal, but in the first message C sends a value that is not in the group G; consequently the key

Petitioner Samsung Ex-1036, 0173

calculated by B will give away information about B's long-term secret key x_B. In an interesting echo of the prime order subgroup attack discussed in Sect. 5.3.1, this requires that $(p-1)/q$, rather than q itself, contains many small factors. Attack 5.4 illustrates the procedure on MTI protocol A(0).

Information known to A: β of small order r with $r|(p-1)/q$.

C		B
$r_C \in_R \mathbb{Z}_q$		
$t_C = g^{r_C}$	$\xrightarrow{\beta t_C}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$Z_{CB} = t_B^{x_C} y_B^{r_C} = Z_{BC}/\beta^{x_B}$	$\xleftarrow{t_B}$	$Z_{BC} = (\beta t_C)^{x_B} y_C^{r_B}$
Test each value of $x_B \bmod r$		

Attack 5.4: Lim–Lee attack on MTI A(0)

Suppose that β is an element whose order r is a small factor of $(p-1)/q$. The shared secret is calculated by B as $Z_{BC} = (\beta t_C)^{x_B} y_C^{r_B}$. Now since $t_C^{x_B} = y_B^{r_C}$ and $y_C^{r_B} = t_B^{x_C}$, C can calculate Z_{BC}/β^{x_B} and, since there are only r possible values for β^{x_B}, C can try out each of these in turn. There are a number of ways that C could verify whether the correct value has been found. One is if a check function is returned by B as in the modified MTI protocol of Menezes *et al.* described in Protocol 5.8 above. Another possibility is if A waits for an authenticated message sent by B following the protocol. Whatever the method used, having identified the value of Z_{BC}, C can obtain β^{x_B} which reveals the value of $x_B \bmod r$. To complete the attack C repeats this procedure with new factors of $(p-1)/q$ in place of r until the value of x_B is obtained. A very similar attack applies to all the A(k) and B(k) MTI protocols.

Lim and Lee suggested two ways to avoid this attack. The first is that each recipient of a protocol message must check that the received value lies in G. The cost of this is an exponentiation, which is a significant extra computational burden. The second method, which they favour, is to choose the prime p so that $(p-1)/q$ has no small factors apart from 2. In this case the attack will give away one bit of information about the principal's secret: the adversary chooses $\beta = -1$, with order $r = 2$, to obtain $x_A \bmod 2$.

5.3.4 Impersonation Attack of Just and Vaudenay

Just and Vaudenay [175] found an impersonation attack on MTI A(0) protocol in the case that the adversary can claim to A also to have identity A. This may be possible in a poor implementation. Notice that A is usually expected

Petitioner Samsung Ex-1036, 0174

to make multiple connections. Attack 5.5 shows how C can impersonate A to herself by choosing t_C based on both the received message from A and a random input. At the end of the attack both principals calculate the shared secret as $y_A^{r_C}$.

A		C_A
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_C \in_R \mathbb{Z}_q$
		$t_C = t_A^{-1} g^{r_C}$
$Z_{AA} = t_C^{x_A} y_A^{r_A}$	$\xleftarrow{t_C}$	$Z_{AC} = y_A^{r_C}$

Attack 5.5: Just–Vaudenay impersonation attack on MTI A(0) protocol

The attack also extends to the whole of the MTI A(k) class of protocols. The adversary simply returns $z_C = z_A^{-1} g^{r_C}$ and A will calculate the shared secret as $Z_{AA} = y_A^{r_C}$. A similar attack is also applicable to the MTI B(0) protocol, although it does not seem to be quite so strong in this case. If C masquerades as A to A and returns $z_C = z_A^{-1} \cdot g^{r_C}$ then A will calculate $Z_{AA} = g^{x_A^{-1} r_C}$. Although C cannot calculate this value directly, C could replay it to get the same key any number of times. Also by choosing $r_C = 0$ the shared secret becomes $Z_{AA} = 1$. The obvious way to avoid all these attacks is for both A and B to ensure that the other has a different identity.

5.3.5 Triangle Attacks on MTI Protocols

Burmester [74] has shown that a 'triangle' attack can be mounted on MTI A(0), given certain assumptions about the release of session keys. The attack also applies to the MTI B(0) protocol, as well as to several protocols related to MTI A(0) which will be mentioned later. The format of the attack is as follows.

1. The adversary C eavesdrops on a session between A and B.
2. C starts separate sessions with A and B in which C uses information gained during step 1. (As a result C does not obtain the session key used in these sessions.)
3. C now induces A and B to reveal the keys used in the sessions between them. Because A and B believe that the session key should be known to C this may be a reasonable assumption in certain application scenarios.
4. With this information C can recover the key used in step 1.

It can be seen that the attack requires more assumptions than usual. It illustrates how difficult it is to consider all possible attacks on cryptographic

Petitioner Samsung Ex-1036, 0175

protocols. A specific example of the triangle attack on the MTI protocol A(0) is as follows.

1. C records the values t_A and t_B used by A and B to form $Z_{AB} = g^{r_B x_A + r_A x_B}$.
2. C uses t_A as its input in a run with B. The agreed key calculated by B is $Z'_{AB} = g^{\tilde{r}_B x_C + r_A x_B}$ where $\tilde{t}_B = g^{\tilde{r}_B}$ is the value sent by B in this run. Similarly C uses t_B in a session with A, which A uses together with its new value $\tilde{t}_A = g^{\tilde{r}_A}$ to generate a key $Z''_{AB} = g^{\tilde{r}_A x_C + r_B x_A}$.
3. C somehow obtains Z'_{AB} and Z''_{AB}.
4. C can now calculate $Z_{AB} = Z'_{AB} \tilde{t}_A^{-x_C} Z''_{AB} \tilde{t}_B^{-x_C}$.

Burmester discusses ways to prevent the attack. Perhaps the simplest is the sensible precaution never to reveal previous session keys; good practice is to destroy session keys immediately after use. Another way is to insist on key confirmation before using a session key. Protocols that incorporate key confirmation will be discussed below.

5.3.6 Forward Secrecy and Key Compromise Impersonation for MTI Protocols

Since Diffie–Hellman key exchange is known to have the attractive property of forward secrecy it is natural to expect that MTI protocols will have this property. However, this turns out not to be always the case.

The shared secret in MTI protocol A(0) is $g^{x_A r_B + x_B r_A}$, which can be found from knowledge of the long-term keys x_A and x_B and the exchanged messages. The same is true for all protocols in the sequence A(k). Again, for the protocols in sequence B(k) it is not necessary to know either r_A or r_B to find the shared secret when both x_A and x_B are known; therefore these protocols do not provide forward secrecy either. (Both the sequences A(k) and B(k) do provide partial forward secrecy since compromise of only one of x_A and x_B does not reveal past session keys.)

The shared secret in the MTI protocol C(0) is $g^{r_A r_B}$, the ephemeral Diffie–Hellman key. The Diffie–Hellman assumption asserts that this secret can be found only with knowledge of either r_A or r_B. Noting that neither r_A nor r_B can be found from the exchanged messages we conclude that C(0) does provide forward secrecy. A similar argument applies to all the protocols in the protocol sequence C(k).

We now turn our attention to key compromise impersonation. Consider again the MTI protocol A(0) and suppose that an adversary C has obtained the long-term secret x_A of A. In order to masquerade as B to A, C must send some value X in message 2 in such a way that C can find the value $X^{x_A} y_B^{r_A}$ calculated by A as the session key. If we assume that A will reject degenerate values such as 0 or 1 then the value cannot be found without knowledge of either x_B or r_A, neither of which are available to C. Therefore we deduce that

Petitioner Samsung Ex-1036, 0176

protocol A(0) is not vulnerable to key compromise impersonation[2]. Similarly, in order to attack any protocol in the sequence A(k) the adversary must find an X such that $X^{x_A} y_B^{r_A x_A^k}$ can be found. Again this requires knowledge of either r_A or x_B and so key compromise impersonation seems impossible.

A similar property holds in the protocol sequence B(k), the difference being that now the adversary cannot find g^{r_A} without knowledge of either x_B or r_A. Therefore we conclude that the MTI B(k) family is also not vulnerable to key compromise impersonation.

However, the situation with C(0) is different. Here C needs to find an X such that $X^{x_A^{-1} r_A}$ may be calculated. With knowledge of r_A, C can use the first message sent by A to construct an appropriate value for X. This observation leads to Attack 5.6. After receiving C's reply, A calculates the shared secret as $y_B^{r_A r_C}$ which can also be calculated by C. Therefore protocol C(0) is vulnerable to key compromise impersonation. A similar attack also applies to any protocol in the sequence C(k).

Information known to C: Private key of A, x_A.

A		C
$r_A \in_R \mathbb{Z}_q$		
$z_A = y_B^{r_A}$	$\xrightarrow{z_A}$	$r_C \in_R \mathbb{Z}_q$
		$z_B = y_B^{x_A r_C}$
$Z_{AB} = z_B^{x_A^{-1} r_A}$	$\xleftarrow{z_B}$	$Z_{AB} = z_A^{r_C}$

Attack 5.6: Key compromise impersonation on MTI C(0)

In summary we find that sequences A(k) and B(k) provide protection against key compromise impersonation but do not provide forward secrecy, while the situation for the sequence C(k) is exactly the opposite. A natural question that arises is whether it is possible to achieve both properties at the same time. In Sect. 5.4.4 we will examine the MQV protocol which seems to achieve this.

5.4 Diffie–Hellman-Based Protocols with Basic Message Format

In this section a number of protocols are examined for which the messages exchanged are the same as the basic Diffie–Hellman protocol. Only two mes-

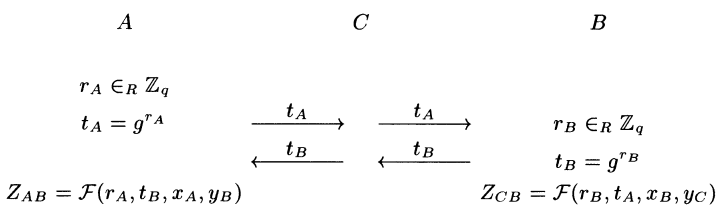
[2] Although Just and Vaudenay [175] stated that A(0) is vulnerable to key compromise impersonation this statement was later retracted.

Petitioner Samsung Ex-1036, 0177

sage passes are involved: A sends $t_A = g^{r_A}$ and B sends $t_B = g^{r_B}$. In contrast to the Diffie–Hellman protocol, the calculation of the shared key involves other components in order to achieve authentication of the key. This other information typically includes the public and private keys of the involved parties, so there is some function $\mathcal{F}$ such that A calculates the shared secret as $Z_{AB} = \mathcal{F}(r_A, t_B, x_A, y_B)$ while B calculates in the symmetrical fashion $Z_{AB} = \mathcal{F}(r_B, t_A, x_B, y_A)$. The MTI A(0) protocol discussed above fits into this class but has a number of potential weaknesses.

One general property is that *all* protocols in this class are vulnerable to the basic unknown key-share attack. For suppose that the adversary C can obtain a certificate for the public key used by B. Then C may sit between A and B and masquerade as B to A as shown in Attack 5.7.

Shared information: $y_C = y_B$.



Attack 5.7: Unknown key-share attack on generic protocol

Principal A calculates the shared secret as $Z_{AB} = \mathcal{F}(r_A, t_B, x_A, y_B)$ while B calculates $Z_{CB} = \mathcal{F}(r_B, t_A, x_B, y_C) = \mathcal{F}(r_B, t_A, x_B, y_B) = Z_{AB}$ since $y_C = y_B$. This attack is prevented if certificates are only issued to users who have shown that they know the private key corresponding to their public key. However, even this is not sufficient to prevent unknown key-share attacks in all cases. Key confirmation can also prevent the generic attack; in Sect. 5.4.8 we will see how to add messages to achieve this.

5.4.1 The Goss Protocol

Goss [135] was awarded a US patent covering a protocol that is extremely similar to the MTI protocol A(0). The only difference is that the shared secret is defined as $Z_{AB} = g^{x_A r_B} \oplus g^{x_B r_A}$ instead of $Z_{AB} = g^{x_A r_B} \cdot g^{x_B r_A}$ in MTI A(0). The similarity carries over into many of the protocol properties: forward secrecy is not provided but key compromise impersonation seems impossible. It is also vulnerable to Burmester's triangle attack (see Sect. 5.3.5).

Petitioner Samsung Ex-1036, 0178

5.4.2 KEA Protocol

The Key Exchange Algorithm (KEA) protocol was designed by the US National Security Agency for use with the SKIPJACK algorithm in key escrow implementations [244]. Originally classified, the protocol was released in June 1998 [248].

The KEA protocol is, like the Goss protocol, a variant of the MTI protocol A(0) as shown in Protocol 5.9. The differences are that the shared secret is defined as $Z_{AB} = g^{x_A r_B} + g^{x_B r_A} \bmod p$ and that there are extra checks in place. As with the Goss protocol, KEA inherits resistance to key compromise impersonation, but does not provide forward secrecy.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
	$\xleftarrow{t_B}$	
$Z_{AB} = t_B^{x_A} + y_B^{r_A} \bmod p$		$Z_{AB} = t_A^{x_B} + y_A^{r_B} \bmod p$

Protocol 5.9: KEA protocol

The KEA protocol specification includes exact parameter sizes: p is a 1024-bit prime and G is a subgroup of order q for a 160-bit prime q with $q|p-1$. There is also a particular key derivation function specified, which makes use of the SKIPJACK algorithm itself to form the 80-bit session key. As usual we have omitted processing of the certificates which is an essential part of the protocol.

Before calculating the shared secret, several checks should be made. A checks that:

1. t_B and y_B are integers greater than 1 and less than p;
2. $t_B^q \bmod p = 1$ and $y_B^q \bmod p = 1$ which ensures that the t_B and y_B are both in G.

B makes the analogous checks. If any check fails then the checking party halts. These checks prevent most of the attacks described for the MTI protocols. The specification also requires each party to check that Z_{AB} does not equal 0 before accepting the key. However, Blake-Wilson and Menezes [45] have pointed out that this check seems unnecessary. If t_B and y_B are in the prime order subgroup G then $t_B^{x_A} = -y_B^{r_A}$ would imply that -1 is an element of order 2 in G. The checks already made before calculating Z_{AB} ensure that t_B is in G and as long as y_B is a genuine public key it must also be in G.

Petitioner Samsung Ex-1036, 0179

5.4.3 The Unified Model Protocol

The Unified Model is a protocol in the IEEE P1363-2000 standard [154] and apparently originates in a standards committee document due to Ankney, Johnson and Matyas (cited by Blake-Wilson and Menezes [45]). It has a very simple design and attractive security properties. As shown in Protocol 5.10 the shared secret is the concatenation of the static and ephemeral Diffie–Hellman keys: $Z_{AB} = g^{r_A r_B}, g^{x_A x_B}$.

Shared information: Static Diffie–Hellman key S_{AB}.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$Z_{AB} = t_B^{r_A}, S_{AB}$	$\xleftarrow{t_B}$	$Z_{AB} = t_A^{r_B}, S_{AB}$

Protocol 5.10: Unified Model key agreement

Before accepting the shared key, A must make the following checks. B makes the analogous checks.

1. $1 < t_B < p$. In particular, degenerate values such as 0 and p should not be allowed.
2. $t_B^q \bmod p = 1$. This ensures that both components of Z_{AB} are in G as long as A has chosen t_A correctly.

The Unified Model provides forward secrecy since it is necessary to know one of the ephemeral private keys to find Z_{AB}. The direct inclusion of the static Diffie–Hellman key prevents unknown key-share attacks if the principals have shown knowledge of the private keys corresponding to their public keys. This is because derivation of the shared secret requires knowledge of one of the long-term private keys. However, the protocol does not prevent key compromise impersonation since knowledge of either of the long-term keys is sufficient to calculate Z_{AB}.

Security proofs for the Unified Model were provided by Blake-Wilson *et al.* [43] in the Bellare–Rogaway model. The basic version shown in Protocol 5.10 was proven secure against active attacks as long as the Diffie–Hellman assumption holds. However, the proof only works with a weakened model of the adversary, whereas the version with key confirmation added (see Sect. 5.4.8) can be proven secure with a strong model of the adversary. The technical point hinges on whether the adversary is able to obtain session keys that have not been confirmed. Certainly in this model key confirmation seems to provide

Petitioner Samsung Ex-1036, 0180

a useful security function, rather than simply being a convenient hint to the partner that the key is ready to use.

5.4.4 MQV Protocol

The MQV protocol was originally due to Menezes *et al.* [233]. It was later improved by these authors plus Law and Solinas [201] and standardised in the IEEE P1363-2000 standard [154].

A special operation is defined on any element t of $\mathbb{Z}_p$, which results in output $\bar{t} = t \bmod 2^w + 2^w$. Outputs of the operation are of fixed size w, which must be large enough to prevent exhaustive search of 2^w elements. Typically w would be 80. Protocol 5.11 shows the message exchange in our usual discrete log setting; the protocol is often described in an elliptic curve setting, where by convention the group is written additively.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$S_A = r_A + \bar{t}_A x_A \bmod q$	$\xleftarrow{t_B}$	$S_B = r_B + \bar{t}_B x_B \bmod q$
$Z_{AB} = (t_B y_B^{\bar{t}_B})^{S_A}$		$Z_{AB} = (t_A y_A^{\bar{t}_A})^{S_B}$

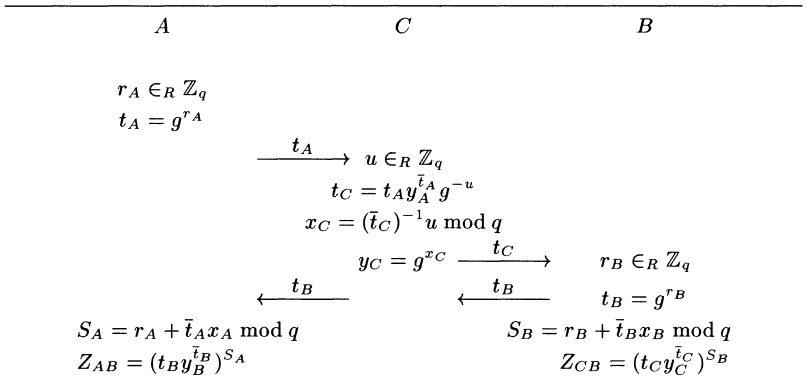
Protocol 5.11: MQV protocol

The shared key is $Z_{AB} = g^{(r_A + \bar{t}_A x_A)(r_B + \bar{t}_B x_B)}$. Before accepting the shared key A must make the following checks. B makes the analogous checks.

1. $1 < t_B < p$. In particular, degenerate values such as 0 and p should not be allowed.
2. $t_B^q \bmod p = 1$. This ensures that $Z_{AB} \in G$ as long as A has chosen t_A correctly.
3. $Z_{AB} \neq 1$. Together with the previous check this ensures that Z_{AB} has order q so preventing any small subgroup attacks.

MQV provides forward secrecy and also protects against key compromise impersonation. The special operation on the exponents destroys the algebraic structure; this may have benefits for practical security but at the same time it obstructs a security proof in terms of any established hard problems. So far there is no formal proof of security for MQV. It was shown by Kaliski [176] that MQV is vulnerable to unknown key-share attacks even in the case that users have shown possession of their private keys. Attack 5.8 shows Kaliski's

Petitioner Samsung Ex-1036, 0181



Attack 5.8: Kaliski's unknown key-share attack on MQV protocol

attack in which the adversary C intercepts and modifies the message sent by A to B.

Although C is able to find the private key x_C corresponding to y_C, it can only be calculated once the first message t_A from A has been seen. An implementation of the attack would therefore require C to get y_C certified before sending on the message t_C to B, a scenario that sounds slightly far-fetched but should not be ruled out without justification. Kaliski suggests that the attack provides a lesson that an active certification authority should be considered in any protocol description. As usual, the unknown key-share attack may be prevented by including the identities in the key derivation function. It may also be prevented in MQV by addition of key confirmation.

The point of using the reduced size exponents $\bar{t}_A$ and $\bar{t}_B$ is that the total calculations required by each principal are reduced, by half of one exponentiation, when compared with most other protocols of this type, such as the Unified Model. On the other hand it may be possible to perform some of the computation *off-line* by choosing random values in advance and assuming the partner's public key is available. In such a case two of the required exponentiations may be performed off-line for the Unified Model but only one for MQV, so that MQV requires half of one exponentiation more on-line computation. A detailed comparison of the Unified Model and MQV is presented by Blake-Wilson and Menezes [45].

5.4.5 Yacobi's Protocol

Yacobi [335] proposed a protocol identical to the MTI protocol A(0) except that a composite modulus is used. He supplied a proof of security of this protocol based on the idea that the exchanged messages are independent of the private keys and hence protocol runs may be perfectly simulated by anyone;

Petitioner Samsung Ex-1036, 0182

therefore a passive attacker should gain nothing from observing a previous protocol run. Yacobi also claimed that this argument extends to the case of an active attacker who is able to obtain old session keys.

Subsequently Desmedt and Burmester [100] pointed out that it cannot be assumed that a malicious protocol partner will act according to the protocol. They showed that the protocol must leak information unless the Diffie–Hellman assumption is false. This theoretical result shows that the proof is flawed, but does not result in any practical attack on the Yacobi or MTI protocols. Desmedt and Burmester proved that a modified protocol is indeed secure in this model. The messages exchanged are identical, but after sending $t_A = g^{r_A}$ and $t_B = g^{r_B}$ both principals engage in a zero knowledge protocol to show that they know r_A and r_B respectively. Now the protocol will only complete if the principals act 'correctly' and so it can always be simulated.

As well as the drawback of the extra interaction required, the proof of security of Burmester and Desmedt's variant does not encompass all the possible actions of an adversary and also partial information on the secret is not accounted for. Therefore it is difficult to say how useful the proof of security is. Indeed, the generic unknown key-share attack applies, as does Burmester's triangle attack (see Sect. 5.3.5).

5.4.6 Ateniese–Steiner–Tsudik Protocol

Ateniese *et al.* [18, 19] examined key agreement for groups. Their general protocol will be examined in Chap. 6, but here we consider their two-party key agreement protocol which was used as a building block. Protocol 5.12 gives a slightly rearranged, but equivalent, description.

Shared information: Static Diffie–Hellman key S_{AB}.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$K = H(S_{AB})$	$\xleftarrow{t_B}$	$K = H(S_{AB})$
$Z_{AB} = t_B^{r_A \cdot K}$		$Z_{AB} = t_A^{r_B \cdot K}$

Protocol 5.12: Ateniese–Steiner–Tsudik key agreement

The shared secret is $Z_{AB} = g^{r_A r_B K} \bmod p$, where $K = H(S_{AB})$. In the initial paper [18] the function H is specified as either a hash function with range in $\mathbb{Z}_q^*$ or simply reduction modulo q. However, the later version of the

Petitioner Samsung Ex-1036, 0183

paper [19] specifies that H should be a bijection from G to $\mathbb{Z}_q$ in the case that $p = 2q + 1$. This latter choice ensures that all possible chosen secrets will be equally likely.

There are many similarities with the Unified Model protocol (5.10) and the security properties are similar. Specifically, forward secrecy is provided and unknown key-share attacks are avoided if the principals are guaranteed to know the private keys corresponding to their public keys. Key compromise impersonation attacks are possible since knowledge of either long-term private key is sufficient to complete the protocol as either initiator or responder.

5.4.7 Just–Vaudenay–Song–Kim Protocol

Recognising that the MTI A(k) protocols do not provide forward secrecy, Just and Vaudenay [175] proposed a variant of MTI A(0) with this property. Their protocol also includes a key confirmation handshake. Later, Song and Kim [303] proposed a very similar protocol designed for use on elliptic curves and with optimised computation, but the shared key is the same in both protocols. The protocol has some enhanced properties over MTI A(0) but unfortunately a security weakness is also introduced, as explained below.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$Z_{AB} = y_B^{r_A} \cdot t_B^{x_A + r_A}$	$\xleftarrow{t_B}$	$Z_{AB} = y_A^{r_B} \cdot t_A^{x_B + r_B}$

Protocol 5.13: Just–Vaudenay–Song–Kim protocol

Protocol 5.13 shows a combined version of the two protocols that we call the Just–Vaudenay–Song–Kim (JVSK) protocol. In keeping with our custom in this section we omit the handshake for key confirmation included in the Just–Vaudenay protocol (Song and Kim also provide variants with key confirmation). Just and Vaudenay also proposed a variant of the MTI C(0) protocol.

The shared secret is $Z_{AB} = g^{x_A r_B + x_B r_A + r_A r_B}$. Note that Z_{AB} is the key of the MTI A(0) protocol multiplied by the ephemeral Diffie–Hellman key. This change allows the protocol to provide forward secrecy. However, contrary to a claim of Song and Kim [303], the protocol becomes vulnerable to key compromise impersonation. Attack 5.9 shows how this can be implemented with adversary C masquerading as B, using knowledge of x_A. The attack still applies if key confirmation messages are included, as in the Just–Vaudenay version of the protocol.

Petitioner Samsung Ex-1036, 0184

Information known to C: Private key of A, x_A.

$$
\begin{array}{lcl}
A & & C_B \\
r_A \in_R \mathbb{Z}_q & & \\
t_A = g^{r_A} & \xrightarrow{\quad t_A \quad} & r_C \in_R \mathbb{Z}_q \\
 & & t_C = g^{r_C} y_B^{-1} \\
Z_{AB} = y_B^{r_A} \cdot t_C^{x_A + r_A} & \xleftarrow{\quad t_C \quad} & Z_{AC} = t_A^{r_C} \cdot t_C^{x_A}
\end{array}
$$

Attack 5.9: Key compromise impersonation on Just–Vaudenay–Song–Kim protocol

There is another flaw in Protocol 5.13 in which an active adversary can replace t_A with y_A^{-1} and t_B with y_B^{-1}. In this case B will calculate $Z_{AB} = y_A^{r_B} \cdot y_A^{-x_B - r_B} = y_A^{-x_B} = S_{AB}^{-1}$. Similarly A calculates $Z_{AB} = y_B^{r_A} \cdot t_B^{x_A + r_A} = S_{AB}^{-1}$ and so both principals have agreed the same key which is the inverse of their static Diffie–Hellman key. If, as we usually assume, the agreed key becomes known to the adversary then the attack can be run again to compromise new sessions. The attack can be avoided if B checks that $t_A \neq y_A^{-1}$ and A does similarly, which adds to the required computation. However, there is no guarantee that other attacks are not possible even if these checks are included. This attack will not work in the version of the protocol that includes key confirmation as long at the values of t_A and t_B are included in the confirmation messages. Such messages were included in the Just–Vaudenay version of the protocol.

5.4.8 Adding Key Confirmation

In this section we have so far only considered key agreement protocols with two message flows without any key confirmation. All the protocols we have described can be extended to provide key confirmation in a generic way, and indeed this technique was explicitly used by many of the protocol authors.

Protocol 5.14 shows the general procedure. In the first two messages A and B exchange their ephemeral Diffie–Hellman keys, as usual, while in the second and third they exchange MACs using the derived key K'. To ensure that the protocol does not give away material to help the adversary, two different hash functions are used, H_1 and H_2, which are assumed to be independent. The key K' is calculated using hash function H_1 while H_2 is used to derive the shared session key $K_{AB} = H_2(Z_{AB})$. This ensures that there is no obstacle to a Bellare–Rogaway-style proof of security as a result of this procedure. Exactly this procedure was used by Blake-Wilson *et al.* [43, 45] to provide key confirmation to the Unified Model and prove its security.

Although the MACs exchanged contain essentially the same fields, the ordering is different. This ensures that no MAC may be replayed to the opposite

Petitioner Samsung Ex-1036, 0185

Shared information: Two independent hash functions, H_1 and H_2.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
		Calculate Z_{AB}
Calculate Z_{AB}	$\xleftarrow{t_B, MAC_{K'}(2, B, A, t_B, t_A)}$	$K' = H_1(Z_{AB})$
$K' = H_1(Z_{AB})$		
Verify MAC	$\xrightarrow{MAC_{K'}(3, A, B, t_A, t_B)}$	Verify MAC

Protocol 5.14: Generic addition of key confirmation to basic DH protocols

role, either to the initiator or responder of the protocol; in particular it prevents a simple reflection of the MAC value in the third message. The inclusion of the message flow numbers also ensures this.

5.4.9 Comparison

Table 5.4 compares the main features of some of the protocols we have examined in this section. We refer the reader again to the survey of Blake-Wilson and Menezes [45] for a detailed comparison between the Unified Model and MQV protocol.

The comparison in Table 5.4 includes only the basic two-pass protocols. Concerning potential attacks, the table indicates whether each protocol provides forward secrecy and protects against key compromise impersonation attacks. We have chosen not to include an indication concerning unknown key-share attacks in Table 5.4 even though at least one protocol (MQV) is known to be vulnerable in its basic form. Our reasoning for this is that such attacks can always be prevented, and also that the vulnerability of a protocol depends on the assumptions about the capabilities of the adversary. We refer the reader to the detailed description of the protocols for discussions on this matter. We have also noted that a security proof exists only for the Unified Model.

The computational requirements are indicated in Table 5.4 by counting the number of exponentiations computed by each principal in a protocol run; this is divided into those that must be on-line (after the protocol starts) and those that may be off-line assuming that the partner's public key is available and the ephemeral Diffie–Hellman private key is chosen in advance. It is important to note that, in order to make a uniform comparison, we have counted only

Petitioner Samsung Ex-1036, 0186

Table 5.4. Summary of major properties of key agreement protocols using basic message format

Properties →	Fwd.	Resists	Exponentiations		Security
↓ *Protocol*	secrecy	KCI	off-line	on-line	proof
MTI A(0)	No	Yes	2	1	No
MTI B(0)	No	Yes	2	1	No
MTI C(0)	Yes	No	1	1	No
KEA/Goss	No	Yes	2	1	No
UM (5.10)	Yes	No	2	1	Yes
MQV (5.11)	Yes	Yes	1	1.5	No
AST (5.12)	Yes	No	2	1	No
JVSK (5.13)	Yes	No	2	1	No

those computations required to calculate the shared secret. For all protocols it is necessary to perform an extra on-line exponentiation to prevent small subgroup attacks in the case that G is a subgroup of much smaller size than $\mathbb{Z}_p^*$ (see Sect. 5.3.1). For MQV we assumed that the value of w used is half the size of other exponents, which accounts for the half exponentiation shown in the table. We also point out that different optimisations are possible. For example, in MTI A(0) and MTI B(0) one of the off-line exponentiations can be removed by calculating an on-line multi-exponentiation in place of the on-line basic exponentiation, which only slightly increases the on-line calculation.

Overall there seems to be no clear winner amongst this class of protocols. But if forward secrecy is required then either MQV or the Unified Model could be good choices. The advantage of MQV is its reduced overall computation and resistance to key compromise impersonation. On the other hand the Unified Model enjoys smaller on-line computation and has a valuable proof of security.

5.5 Diffie–Hellman-Based Protocols with Enhanced Message Format

In this section we look at protocols that add information to authenticate the Diffie–Hellman messages exchanged between the parties. In many cases the additional algorithms used for authentication are made deliberately independent of the Diffie–Hellman exchange, which reduces the chance of 'unfortunate' interaction between different protocol fields. Two features differentiate the protocols in this section from those in Sect. 5.4.

- The shared secret is usually the ephemeral Diffie–Hellman key. In this case the protocol will usually possess forward secrecy as long as the shared

Petitioner Samsung Ex-1036, 0187

secret and the long-term keys of the principals are completely independent. There are exceptions in which protocol designers have reused the ephemeral Diffie–Hellman inputs in the authentication information.

- The ephemeral public keys are usually signed by the principal generating them. In this case key compromise impersonation is always avoided since knowing A's private key does not help the adversary to forge B's signature. A similar effect occurs if the ephemeral public keys are encrypted with the partner's public key.

5.5.1 STS Protocol

The Station-to-Station (STS) protocol, due to Diffie *et al.* [103], adds a digital signature to the exchanged messages to provide authentication for the Diffie–Hellman protocol. In addition the shared secret is used to provide further assurances. Protocol 5.15 shows the main version of the STS protocol; a variant in which the encryption is replaced by the use of a MAC is discussed below. The shared secret is $Z_{AB} = g^{r_A r_B}$ and the session key K_{AB} is derived from Z_{AB} in some unspecified way.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{t_B, \{Sig_B(t_B, t_A)\}_{K_{AB}}}$	$Z_{AB} = t_A^{r_B}$
Decrypt/verify signature		
	$\xrightarrow{\{Sig_A(t_A, t_B)\}_{K_{AB}}}$	Decrypt/verify signature

Protocol 5.15: STS protocol

Because the shared secret is the ephemeral Diffie–Hellman key, forward secrecy is provided by the STS protocol. Also the signatures provide protection against key compromise impersonation since if a long-term key is lost this does not help an adversary to forge a signature of a different entity. Lowe's attack [212] on the STS protocol was discussed in Sect. 2.3.2. This shows that the protocol does not provide mutual belief in the key, or strong entity authentication.

Diffie *et al.* [103] explain that the symmetric encryption is essential in Protocol 5.15 in order to prevent unknown key-share attacks. Specifically, without the encryption the adversary C could remove the signature of A in the final message and replace it with C's own. The result is that A and

Petitioner Samsung Ex-1036, 0188

B both complete the protocol but B believes that the key is shared with C, while A believes that it is shared with B. An unknown key-share attack remains possible if the adversary can obtain a certificate for a public key that is identical to that of the victim (which can be either A or B). In a similar way to Attack 5.7 the adversary simply relays messages between A and B.

Unknown key-share attacks can be prevented by including the name of the partner entity in the signatures exchanged. Moreover, this change provides an explicit indication of the peer entity so that the stronger form of entity authentication is achieved (in particular Lowe's attack no longer applies). In addition there no longer seems to be a need for the symmetric encryption in the protocol. This leads to the STS variant shown in Protocol 5.16. An essentially identical protocol has been proven secure using the model of Bellare *et al.* [27, 82] (see also the description of Blake-Wilson and Menezes [45]).

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
Verify signature	$\xleftarrow{t_B, Sig_B(t_B, t_A, A)}$	$Z_{AB} = t_A^{r_B}$
$Z_{AB} = t_B^{r_A}$	$\xrightarrow{Sig_A(t_A, t_B, B)}$	Verify signature

Protocol 5.16: Modified STS protocol

The role of the encryption mechanism in the STS protocol can be taken by a MAC since its purpose is to ensure that the signing party has possession of the session key and not to provide confidentiality. This leads to the STS variant in Protocol 5.17. A potential disadvantage is that it increases the lengths of the second and third messages.

Blake-Wilson and Menezes [46] published an alternative unknown key-share attack in which the adversary can choose the correct private key, and so the certification process cannot be used to prevent the attack. Given a signature $Sig_A(t_A, t_B)$ used in the protocol run, the adversary C must find a new public key such that $Sig_C(t_A, t_B) = Sig_A(t_A, t_B)$. There are two problems that the adversary faces in engineering this situation. The first is to find such a key that satisfies this *duplicate signature* property, and the second is to have the new key certified in real time.

Blake-Wilson and Menezes show that the first problem can be solved for many popular signature schemes. The second problem is common with the Attack 5.8 on the MQV protocol. Once this substitution is achieved the attack proceeds again with the adversary simply relaying messages between the

Petitioner Samsung Ex-1036, 0189

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{t_B, Sig_B(t_B, t_A), MAC_{K_{AB}}(t_B, t_A)}$	$Z_{AB} = t_A^{r_B}$
Verify signature and MAC	$\xrightarrow{Sig_A(t_A, t_B), MAC_{K_{AB}}(t_A, t_B)}$	Verify signature and MAC

Protocol 5.17: STS protocol using MACs

parties. However, this attack can work only against the version of STS using MACs (5.17), and not against Protocol 5.15, because the adversary needs to see the signature in order to calculate the new public key.

Since the adversary knows the private key of the new public key in this second attack, it is not sufficient to prevent it by asking certifiers to check this knowledge. Instead Blake-Wilson and Menezes suggest a number of preventative measures. The generic solution to include the principal identities in the key derivation function still works. Blake-Wilson and Menezes suggest including the identities within the signature as a preventative measure. However, Baek and Kim [22] show that this is still not sufficient, but recommend including identities explicitly within the MAC as well as in the signature.

5.5.2 Oakley Protocol

In this section, and the following two, we consider three sets of protocols designed for practical use in the Internet. In contrast to the separate key establishment protocols that we mainly focus on in this book, these sets of protocols can be customised by negotiation of various protocol features during the protocol run itself. This flexibility can result in new security threats. Oakley is defined in Internet RFC 2412 [258]. The basic design is similar to the STS protocol but the specification emphasises the following differences.

- As part of the protocol, principals A and B choose cookies, CK_A and CK_B respectively, which are used to mitigate denial of service attacks. The correct cookie must be returned in subsequent messages. The exact format of cookies is not specified and it is optional whether cookies are exchanged before the key exchange begins.
- The protocol includes negotiation of cryptographic algorithms that are used to support the protocol, including the encryption algorithm, key derivation function and authentication method. The group used for Diffie–Hellman itself can also be negotiated.

Petitioner Samsung Ex-1036, 0190

- There need be no use of encryption (or a MAC) for authentication of the session key. In some versions this allows the session key to be derived after completion of the protocol if desired.

Several specific protocols are given in the Oakley documentation. We examine simplified versions of three of these, ignoring some plaintext fields such as a string indicating the name of the protocol itself. A further option does not use Diffie–Hellman and therefore fails to provide forward secrecy.

Protocol 5.18 is named 'aggressive' because it anticipates that certain possible problems will not arise. In particular:

- it only works when B can use the same group as initially proposed and used by A in the first message;
- it does not force either principal to respond with a cookie before the other principal stores state information regarding the connection.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$CK_A, t_A, \texttt{list}, A, B, N_A,$	
	$Sig_A(A, B, N_A, t_A, \texttt{list}) \longrightarrow$	
		Verify signature
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
	$CK_B, CK_A, t_B, \texttt{algo}, B, A, N_B, N_A,$	
	$Sig_B(B, A, N_B, N_A, t_A, t_B, \texttt{algo}) \longleftarrow$	
		$Z_{AB} = t_A^{r_B}$
Verify signature		
$Z_{AB} = t_B^{r_A}$	$CK_A, CK_B, t_A, \texttt{algo}, A, B, N_A, N_B,$	
	$Sig_A(A, B, N_A, N_B, t_A, t_B, \texttt{algo}) \longrightarrow$	
		Verify signature
$K_{AB} = MAC_{N_A, N_B}(Z_{AB}, CK_A, CK_B)$		

Protocol 5.18: Oakley aggressive mode protocol

In the first message of Protocol 5.18, principal A offers a list of possible algorithms, `list`, for encryption, hashing and authentication, which A is prepared to use for this session. B responds with a particular algorithm set, `algo`, in message 2. The nonces N_A and N_B are chosen by A and B respectively and are of unspecified size. Protocol 5.18 has a similar basic structure to Protocol 5.16, but here A needs to calculate two signatures instead of one.

As well as checking all received signatures, both parties must ensure that received ephemeral values are not the degenerate values 1 or $p - 1$. A specific

Petitioner Samsung Ex-1036, 0191

key derivation function is used to calculate the session key from the shared secret $Z_{AB} = g^{r_A r_B}$ as follows.

$$K_{AB} = MAC_{N_A,N_B}(Z_{AB}, CK_A, CK_B). \tag{5.1}$$

Oakley was specifically designed to provide forward secrecy, which follows from the use of the ephemeral Diffie–Hellman key as the shared secret. In addition, the signatures of both parties prevent key compromise impersonation. Unknown key-share attacks are prevented by inclusion of the principal identities in the signatures, similar to Protocol 5.16.

Protocol 5.19 is an Oakley variant designed to prevent disclosure of user identities, and for this reason the user identities are encrypted instead of being signed. The identity B' can be thought of as a domain name for B and may be a generic identity at the location of B; for example, if B is an entity in a corporate environment, B' may be the public key of the corporation with a known public key. The reason for using the generic name is so that the recipient node is able to decrypt without receiving the identity of the recipient in plaintext. An interim key $K = H(N_A, N_B)$ is used with the hash function H in order to authenticate the exchange.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$CK_A, t_A, \texttt{list}, B',$ $E_{B'}(A, B, E_B(N_A)) \rightarrow$	
		$r_B \in_R \mathbb{Z}_q$
		$K = H(N_A, N_B)$
		$t_B = g^{r_B}$
	$CK_B, CK_A, t_B, \texttt{algo},$ $E_A(B, A, N_B), MAC_K(B, A, t_B, t_A, \texttt{algo}) \leftarrow$	
$K = H(N_A, N_B)$		$Z_{AB} = t_A^{r_B}$
Verify MAC		
$Z_{AB} = t_B^{r_A}$	$CK_A, CK_B, MAC_K(A, B, t_A, t_B, \texttt{algo}) \rightarrow$	Verify MAC

Protocol 5.19: Alternative Oakley protocol

In this variant the shared secret is again $Z_{AB} = g^{r_A r_B}$ and the session key is defined using Eqn. 5.1. Forward secrecy and protection against unknown key-share attacks are both provided as in Protocol 5.18. Unknown key-share attacks are prevented because of the encryption of the N_A and N_B values used in the MAC calculation.

Protocol 5.20 is a 'conservative' Oakley variant which exchanges cookies before the key exchange itself commences. This is the way that cookies were

Petitioner Samsung Ex-1036, 0192

originally designed to work, so that some assurance about the origin of the request is obtained before the computationally expensive part of the protocol begins and without requiring the responder to store any state. The first message is simply an indication to B that the protocol should commence. The consequence of the use of cookies in this manner is the increased number of messages and rounds required in the protocol.

A		B
	$\xrightarrow{OK}$	
	$\xleftarrow{CK_B}$	
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{CK_A, CK_B, t_A, \texttt{list}}$	$r_B \in_R \mathbb{Z}_q$
	$\xleftarrow{CK_B, CK_A, t_B, \texttt{algo}}$	$t_B = g^{r_B}$
$Z_{AB} = t_B^{r_A}$		
$K' = H(Z_{AB})$	$\xrightarrow{CK_A, CK_B, t_A, \{A, B, E_B(N_A)\}_{K'}}$	
		$K = H(N_A, N_B)$
		$Z_{AB} = t_A^{r_B}$
	$\xleftarrow{CK_B, CK_A, \{E_A(N_B, N_A), B, A, MAC_K(B, A, t_B, t_A, \texttt{algo})\}_{K'}}$	$K' = H(Z_{AB})$
$K = H(N_A, N_B)$		
Verify MAC		
	$\xrightarrow{CK_A, CK_B, \{MAC_K(A, B, t_A, t_B, \texttt{algo})\}_{K'}}$	Verify MAC

Protocol 5.20: Oakley conservative protocol

As in Protocol 5.19, the principal identities are protected in Protocol 5.20 but here a temporary key K', derived from the shared secret, is used for this purpose. The first four messages exchanged are therefore anonymous. Once again the session key is defined using Eqn. 5.1. Forward secrecy and protection against key compromise impersonation and unknown key-share attacks appear to be provided as in Protocol 5.19. However, there is an important difference in this protocol which could cause problems as we now explain.

The return of the encrypted nonce sent from B to A in Protocol 5.20 opens up the possibility of an interesting attack in which the adversary masquerades as A, and subsequently uses the real A as a decryption oracle. The attack depends on an assumption that the message field $E_A(N_B, N_C)$ sent by B will be interpreted by A as the encryption of a single nonce when replayed by C in a second protocol run. This need not be unreasonable since the length of nonces is variable in the Oakley specification. In a correct implementation

Petitioner Samsung Ex-1036, 0193

each nonce should have its length specified and therefore A should detect the problem and abort the protocol. On the other hand the attack would still work if A ignores the second nonce and simply decrypts N_B, discarding the second nonce. Therefore we believe that a careless implementation could be vulnerable to the attack. In more detail the attack proceeds as follows.

1. C masquerades as A and sends $t_C = g^{r_C}$ to B so that B accepts the temporary key $K' = H(Z_{CB})$ as shared with A.
2. C sends $\{A, B, E_B(N_C)\}_{K'}$ to B in the fifth message flow. B will choose a nonce N_B and calculate $K = H(N_C, N_B)$.
3. B sends $\{E_A(N_B, N_C), \ldots\}_{K'}$ to C as part of the sixth message flow. The adversary can remove the symmetric encryption to obtain $E_A(N_B, N_C)$.
4. C starts a second protocol run with A. This proceeds normally until the fifth message flow when C sends $\{C, A, E_A(N_B, N_C)\}_{K''}$ where K'' is a temporary key shared between C and A.
5. As long as A interprets the pair (N_B, N_C) as a single nonce from C, she will reply with $\{E_C(N_A, N_B, N_C)\}_{K''}$. C can extract N_B and so complete the first protocol run with B.

5.5.3 SKEME Protocol

SKEME is due to Krawczyk [193]. Like Oakley, it is a set of protocols suitable for negotiation of services in a general networked environment. In addition to providing establishment of a good key Krawczyk [193] states additional requirements for SKEME as follows.

Periodic Key Refreshment It should be possible to update the session key used at low computational cost. A special mode of the protocol is included for this purpose.

Forward Secrecy This property should be available as an option. In some cases it may be sacrificed for efficiency gains.

Key Separation This is a principle governing how the shared secret is used to obtain session keys. Different cryptographic functions are provided with independent keys.

Privacy and Anonymity As well as providing confidentiality of user data it may be desirable to hide the identities of the communicating parties. For this reason the SKEME protocol avoids digital signatures, which can always be used at least to confirm the identity of the signer.

Denial of Service Protection SKEME was designed to employ Photuris-style cookies [300] in the same way as Oakley. This requires an additional initial message exchange that is omitted from the descriptions below (as well as in those of Krawczyk [193]).

Efficiency and Simplicity The design tries to reduce the options and use compact and uniform message formats. In addition, efficient algorithms such as hash functions are preferred over the use of digital signatures.

Petitioner Samsung Ex-1036, 0194

Support for Multiple Models and Algorithms Certificate and shared key models are supported. Cryptographic algorithms are specified in a generic sense so as to avoid dependence on particular primitives.

There are a number of possible protocols.

1. The basic mode, shown as Protocol 5.21, which uses public keys of both parties to provide forward secrecy and anonymity of the principals.
2. A version using public keys but without forward secrecy, which is described as Protocol 5.38.
3. A version using an existing shared secret and providing forward secrecy. This is similar to Protocol 5.21 but the encrypted field in messages 1 and 2 is omitted and instead the previously shared key is used as the key K_0 for the MAC.
4. A re-keying mechanism.

Shared information: Security parameter L.

A		B
$N_A \in_R [0, 2^L]$ $r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$	$\xrightarrow{E_B(A, N_A), t_A}$	$N_B \in_R [0, 2^L]$ $r_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}$
Verify MAC $K_0 = H(N_A, N_B)$ $Z_{AB} = t_B^{r_A}$	$\xleftarrow{E_A(N_B), t_B, MAC_{K_0}(t_A, t_B, B, A)}$ $\xrightarrow{MAC_{K_0}(t_B, t_A, A, B)}$	$K_0 = H(N_A, N_B)$ Verify MAC $Z_{AB} = t_A^{r_B}$

Protocol 5.21: SKEME protocol basic mode

In Protocol 5.21 the nonces N_A and N_B are shown as being chosen in $[0, 2^L]$ for some security parameter; the size of L is not specified by Krawczyk, who says only that they should be 'chosen as (pseudo-)random values'. The shared secret is $Z_{AB} = g^{r_A r_B}$ with session key $K_{AB} = H(Z_{AB})$.

Krawczyk presents arguments that the simple modes of SKEME, in which K_0 is previously shared, can be proven secure in the Bellare–Rogaway model. Moreover, Bellare *et al.* [27] show that Protocol 5.21 can be derived as an output of their modular design method, and therefore the protocol inherits a formal proof of security in their model.

Petitioner Samsung Ex-1036, 0195

5.5.4 Internet Key Exchange

As part of an initiative to secure the foundations of the Internet, the Internet Engineering Task Force has been working to develop an IP security protocol, known simply as IPSec. The concerns of IPSec go considerably beyond key establishment so here we examine only a very small part of the development. The reader interested in understanding the complexity of designing an overall system to secure Internet communications is encouraged to consult the considerable literature on IPSec. We refer to a number of useful sources in this section.

During the 1990s a number of different key establishment protocols were proposed for inclusion within the IPSec solution. These included Oakley (see Sect. 5.5.2), SKEME (see Sect. 5.5.3) and Photuris [300]. Through a long and complex evolution a single proposed Internet standard emerged in 1998, known as Internet Key Exchange, or IKE [142]. At the time of writing, this document remains a proposed standard. However, many criticisms of the protocols have been made [110, 269, 342, 343] and it seems inevitable that a new version of IKE will soon emerge with significant differences. Detailed descriptions of the current IKE protocols are given by both Borella [52] and Cheng [88] which are useful supplements to the description in the draft standard itself [142].

The IKE protocol is strongly related to the Oakley and SKEME protocols. It has two phases. This first phase is concerned with establishing a secure channel by key exchange. The second phase is concerned with using the secure channel to set up sessions known as *Security Associations* which can be used to protect the confidentiality and/or integrity of exchanged data. However, the need for this split has been questioned [269] and the indications are that the revised version of IKE will not have two phases. We will not consider the second phase further here.

The IKE protocols employ cookies similar to those of the Oakley protocol. However, the protocol specification requires some state to be recorded even in the first message exchange in order to check a returned cookie. Therefore, although there is some mitigation of denial of service attacks, the result is not as effective as when stateless cookies are used. In addition it is optional whether cookies are exchanged at all before the key exchange begins. This weakening of the cookie mechanism has received considerable criticism [269]. The protocol also includes negotiation of cryptographic algorithms that are used to support the protocol, including encryption algorithm, key derivation function and authentication method. The group used for Diffie–Hellman itself can also be negotiated.

Because IKE emerged as a compromise solution it is not surprising to learn that it contains a lot of different options. It contains no less than eight key establishment protocols which can be divided into two sets of four according to whether *main mode* or *aggressive mode* is used. The protocols in main mode use six messages, which can be divided into three pairs with the following aims.

Petitioner Samsung Ex-1036, 0196

Stage 1: Exchange cookies and agree the algorithms to be used.
Stage 2: Exchange ephemeral Diffie–Hellman keys and nonces.
Stage 3: Perform entity authentication and key confirmation.

This division presents an attractive framework, but the large number of exchanges is another point of significant criticism. The aggressive mode collapses most of the functionality into only three message exchanges in a more conventional manner. In general the protocols in aggressive mode suffer from two limitations in comparison with the main mode.

- Use of cookies is of limited value since the responder needs to start the computationally expensive part of the protocol before the cookie exchange is complete.
- Because the Diffie–Hellman exchange begins in the first message the responding party can only complete the protocol if the group used is supported by the responding party. (This can be negotiated in the main mode.)

Protocol 5.22 shows one of the four main versions of the IKE protocol; this one uses digital signatures for authentication. In Stage 1, principal A offers a list of acceptable algorithms, `list`, that A is prepared to use, and B responds with a particular algorithm set, `algo`. Once the ephemeral Diffie–Hellman keys have been exchanged in the second pair of messages, both parties can derive the shared secret and form the temporary key $KM = MAC_{N_A,N_B}(Z_{AB})$, which is used to form the MAC of the protocol parameters in the final message exchange. We have used the suggestion of Ferguson and Schneier [110] in replacing the more general keyed hash function used in the IKE specification with a MAC, since it seems that a MAC is what is required here. Indeed HMAC [26] is the only explicit suggestion in the standard for this function. In addition to KM, several other keys are derived from Z_{AB}. One of these is used to encrypt the final message exchange and is denoted by K' in Protocol 5.22. We have not specified the function used to derive K', but the IKE specification does so in terms of a generic keyed hash function. Although examples of possible functions are given, Ferguson and Schneier [110] criticised the lack of any description in the specification of the properties that such a function should have.

Since the principal identities are hidden in all messages, the protocol provides anonymity against a passive eavesdropper. However, Perlman and Kaufman [269] pointed out that an active adversary can discover the identity of A by masquerading as B during the first two phases, and thereby obtain the correct value of K'. Of course the adversary cannot cannot forge B's signature and so is unable to complete the final message of the protocol. But if A were a mobile station who wished to protect her location then the attack could be deemed successful. Perlman and Kaufman suggest to protect A's identity by moving the contents of the final message back to the fourth message. Although this means that a similar active attack is now possible against B's identity,

Petitioner Samsung Ex-1036, 0197

Shared information: Symmetric key derivation function $SKDF$. Security parameter L with $64 \leq |L| \leq 2048$.

A		B
	Stage 1	
	$CK_A, \texttt{list} \longrightarrow$	
	$\longleftarrow CK_B, \texttt{algo}$	
	Stage 2	
	$CK = CK_A, CK_B$	
$N_A \in_R [0, L], r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{CK, t_A, N_A}$	$N_B \in_R [0, L], r_B \in_R \mathbb{Z}_q$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{CK, t_B, N_B}$	$t_B = g^{r_B}$
		$Z_{AB} = t_A^{r_B}$
	Stage 3	
	$KM = MAC_{N_A, N_B}(Z_{AB})$	
	$K' = SKDF(KM)$	
	$M_{AB} = MAC_{KM}(t_A, t_B, CK_A, CK_B, \texttt{list}, A)$	
	$M_{BA} = MAC_{KM}(t_B, t_A, CK_B, CK_A, \texttt{list}, B)$	
	$\xrightarrow{CK, \{A, Sig_A(M_{AB})\}_{K'}}$	
		Decrypt and verify MAC and signature
	$\xleftarrow{CK, \{B, Sig_B(M_{BA})\}_{K'}}$	
Decrypt and verify MAC and signature		

Protocol 5.22: IKE main protocol using digital signatures

Petitioner Samsung Ex-1036, 0198

they argue that the initiator is more vulnerable to such an attack in a typical application involving a mobile client connecting to a server. Moreover this suggestion reduces the number of messages by one.

Ferguson and Schneier [110] pointed out that a reflection attack is possible on Protocol 5.22. The adversary masquerades as B and simply copies and returns the values CK_A and t_A in place of CK_B and t_B in the first two stages. Then A will also accept her own signature reflected back in the final message. Of course the attack is only meaningful if A will accept her own identity as the identity of her peer principal. Since IKE may be configured only to authenticate IP (machine) addresses this may not be an unreasonable assumption. This is an attack against the key confirmation and entity authentication properties; the adversary does not obtain the shared secret. The attack also applies to the main mode of IKE, when authentication is based on a previously shared secret.

The signatures in the final two messages are intended to provide authentication and integrity of the parameters that have been agreed earlier in the protocol. However, it is an oversight that the list of algorithms that was offered by A is included in the MAC, while the set, `algos`, accepted by B is not part of the calculation. Potential problems with this omission were described by both Zhou [342] and Ferguson and Schneier [110]. The adversary can cause A and B to believe that different sets of algorithms were accepted, or make A accept a weak set and attack the algorithms in a subsequent session.

Meadows [230] performed a formal analysis of IKE using the NRL Analyzer. She identified a number of ambiguities in the specification which could lead to insecure implementations. Indeed, the incompleteness of the specification has been a recurring theme in the several analyses of IKE that have been published. Meadows also pointed out that several of the IKE protocols do not provide strong entity authentication in the sense discussed in Chap. 2. (Meadows refers instead to 'penultimate authentication' as the property that, when an entity accepts a key, the peer entity should have taken part in the earlier part of the protocol. Here we re-interpret this to mean that B does not achieve assurance that A has knowledge of B as the peer entity.) In Protocol 5.22 the adversary C can sit between A and B masquerading as A to B while A believes the protocol is being executed with C. The protocol messages are relayed unchanged by C, but the final message from B is simply deleted. Consequently A will not accept but B completes the protocol properly. The effect is the same as in Lowe's attack on STS discussed in Chap. 2 and cannot be accepted as an attack unless strong entity authentication is a protocol goal. Nevertheless, this extra property could easily be achieved by including the peer identity in the signatures of the last two messages of the protocol.

Our brief look at the IKE protocols in this section has highlighted a number of serious deficiencies. A proposal for a new version of IKE has recently been introduced into the standards discussion. It contains a simpler and more efficient protocol which avoids the problems that have been found in the cur-

Petitioner Samsung Ex-1036, 0199

rent version. It seems certain that a better version of IKE will soon be agreed and published.

5.5.5 Arazi's Protocol

Arazi's protocol [17] was designed to integrate Diffie–Hellman with the Digital Signature Standard (DSS) [247] in such a way that computation is saved. This is done by overloading the random value chosen by each party to serve two roles: one as the ephemeral Diffie–Hellman private key and one as part of the signature. The protocol has an elegant design and is efficient, but pays for these with some security weaknesses.

Shared information: Hash function H for DSS signature.

A		B
$r_A \in_R \mathbb{Z}_q, t_A = g^{r_A}$		
$s_A = r_A^{-1}(H(t_A) + x_A t_A) \bmod q$	$\xrightarrow{t_A, s_A}$	Verify signature $(t_A \bmod q, s_A)$
		$r_B \in_R \mathbb{Z}_q, t_B = g^{r_B}$
Verify signature $(t_B \bmod q, s_B)$	$\xleftarrow{t_B, s_B}$	$s_B = r_B^{-1}(H(t_B) + x_B t_B) \bmod q$
$Z_{AB} = t_B^{r_A}$		$Z_{AB} = t_A^{r_B}$

Protocol 5.23: Arazi's key agreement protocol

In Protocol 5.23, the value t_A is the ephemeral Diffie–Hellman public value, while s_A is calculated so as to make the pair $(t_A \bmod q, s_A)$ the DSS signature of t_A. The values t_B and s_B are calculated in a symmetrical fashion. Both A and B are able to check the DSS signatures from the received messages and, if they are correct, they calculate the shared secret as $Z_{AB} = g^{r_A r_B}$.

Forward security is not provided since if x_B, say, becomes known then r_B may be found from s_B and t_B using the same equation used by B to calculate s_B. Furthermore, Nyberg and Rueppel [254] showed that knowledge of one shared secret allows all subsequent shared secrets to be found by a passive eavesdropper. This may be seen from the following equality.

$$Z_{AB}^{s_A s_B} = g^{H(t_A)H(t_B)+x_A x_B t_A t_B} y_B^{H(t_A)t_B} y_A^{H(t_B)t_A}.$$

If Z_{AB} is known then $g^{x_A x_B}$ (which is the static Diffie–Hellman key) may be found since all other values are known. When a new instance of the protocol is run between A and B the equation can be used again to find the new Z_{AB} value. Nyberg and Rueppel suggest that the overloading of the random values r_A and r_B is the root of the problem and compare the problem with an attack on Protocol 5.33 below, which can be said to make the same mistake.

Petitioner Samsung Ex-1036, 0200

Later, Brown and Menezes [72] showed that a key recovery attack is also possible on Protocol 5.23. This attack follows the same basic idea as the attacks of Lim and Lee discussed in Sect. 5.3.3, but in addition uses recent lattice-based attacks on digital signatures to recover the key when only the lower significant bits of r_B are known in several runs.

5.5.6 Lim–Lee Protocols

Lim and Lee [205] proposed five key agreement protocols based on Diffie–Hellman. One of these that uses only static Diffie–Hellman was described above as Protocol 5.5. A second is essentially the MTI A(0) key agreement with a special key confirmation mechanism. The remaining three protocols are similar to Arazi's protocol in that they include a signature by each party that reuses the Diffie–Hellman parameters.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
		$E = H(t_A, t_B)$
$t_B = g^{s_B} y_B^E$	$\xleftarrow{s_B, E}$	$s_B = r_B - x_B E \bmod q$
$E \stackrel{?}{=} H(t_A, t_B)$		$Z_{AB} = t_A^{r_B}$
$s_A = r_A - x_A E \bmod q$		
$Z_{AB} = t_B^{r_A}$	$\xrightarrow{s_A}$	$g^{s_A} y_A^E \bmod p \stackrel{?}{=} t_A$

Protocol 5.24: Lim–Lee Schnorr-based protocol

Protocol 5.24 uses the signature scheme of Schnorr [291] to authenticate the messages. The shared secret is the ephemeral Diffie–Hellman key but in this case forward secrecy is not provided. This is because if x_B becomes known then r_B can be recovered from the public values s_B and E and then $Z_{AB} = t_A^{r_B}$ can be calculated.

Lim and Lee [205] showed that if an adversary obtains one Z_{AB} value then all subsequent values may be found in a similar way to the attack of Nyberg and Rueppel described in Sect. 5.5.5. They point out that this attack may be avoided if a one-way key derivation function is used, and the adversary is assumed to be able only to obtain K_{AB} rather than Z_{AB}.

A variant of Protocol 5.24 redefines the value E as $E = ((Z_{AB} \oplus t_A) \bmod q) \bmod 2^t$ for a security parameter t and adjusts the checks made by A and B accordingly. This has the advantage that there is no extra exponentiation required to find the shared secret, and so computation is saved.

Petitioner Samsung Ex-1036, 0201

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$Z_{AB} = (y_A t_A)^{x_B r_B}$
		$E = ((Z_{AB} \oplus r_B) \bmod q) \bmod 2^t$
$Z_{AB} = y_B^{(x_A + r_A) r_B}$	$\xleftarrow{r_B, E}$	
$E \stackrel{?}{=} ((Z_{AB} \oplus r_B) \bmod q) \bmod 2^t$		
$s_A = r_A - x_A E \bmod q$	$\xrightarrow{s_A}$	$g^{s_A} y_A^E \stackrel{?}{=} t_A$

Protocol 5.25: Lim–Lee Schnorr-based variant

Protocol 5.25 is a final variant. The shared secret is $Z_{AB} = g^{(r_A + x_A) x_B r_B}$ which is unusual in that it is asymmetric with respect to A and B. Notice that q and t are both much smaller than p, so knowledge of E and r_B does not give a simple way to recover Z_{AB}. Unfortunately this protocol also fails to provide forward secrecy since knowledge of either x_A or x_B alone allows the calculation of Z_{AB}. If x_A becomes known then r_A can be found from s_A and E and then $Z_{AB} = y_B^{(x_A + r_A) r_B}$ can be re-calculated. If x_B becomes known then $Z_{AB} = (y_A t_A)^{x_B r_B}$ can be re-calculated. Since $Z_{AB} = S_{AB}^{r_B (E+1)} \cdot y_B^{r_B s_A}$ this protocol shares the property with Protocol 5.23 that knowledge of one Z_{AB} value is sufficient to find all subsequent values.

5.5.7 Hirose–Yoshida Protocol

The protocol of Hirose and Yoshida [144] uses a novel signature scheme designed to include elements from the ephemeral keys of both entities. Although the signatures seem similar to those used in the protocols in Sects. 5.5.5 and 5.5.6, they use an extra random parameter and this allows the weaknesses of the former protocols to be avoided. The description in Protocol 5.26 includes some changes of sign in order to maintain our standard notation for public keys.

The signature parameters formed by B are defined as $e_B = H(g^{s_B}, t_B, t_A)$ and $w_B = s_B - e_B r_B - e_B^2 x_B \bmod q$, where s_B is randomly chosen for this signature. The values e_A and w_A are formed by A in an analogous fashion. On receipt of message 2, A must verify the signature. A will respond with message 3 only if this signature is correct. On receipt of message 3, B must verify the signature from A. If the protocol completes successfully then both A and B calculate the shared secret as the ephemeral Diffie–Hellman key: $Z_{AB} = g^{r_A r_B}$. The extra random values, s_A and s_B, prevent r_A and r_B becoming known even if x_A and x_B are known, and therefore forward secrecy is provided. As long as

Petitioner Samsung Ex-1036, 0202

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A, A}$	$r_B, s_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
		$e_B = H(g^{s_B}, t_B, t_A)$
		$w_B = s_B - e_B r_B - e_B^2 x_B \bmod q$
$e_B \stackrel{?}{=} H(g^{w_B}(t_B y_B^{e_B})^{e_B}, t_B, t_A)$	$\xleftarrow{t_B, e_B, w_B, B}$	
$s_A \in_R \mathbb{Z}_q$		
$e_A = H(g^{s_A}, t_A, t_B)$		
$w_A = s_A - e_A r_A - e_A^2 x_A \bmod q$	$\xrightarrow{e_A, w_A}$	$e_A \stackrel{?}{=} H(g^{w_A}(t_A y_A^{e_A})^{e_A}, t_A, t_B)$
$Z_{AB} = t_B^{r_A}$		$Z_{AB} = t_A^{r_B}$

Protocol 5.26: Hirose–Yoshida key agreement protocol

the signatures cannot be forged, resistance to key compromise impersonation is provided too.

Unknown key-share attacks are possible on Protocol 5.26 if an adversary can arrange a certified public key equal to that of a victim. The adversary can then simply change either of the identifiers in the first two messages and make the other party believe that the secret is shared with the adversary. Baek and Kim [22] have shown that the more sophisticated unknown key-share attacks constructed by Blake-Wilson and Menezes [46] apply to this protocol too by showing that duplicate signatures can be found for the signatures used.

5.5.8 Comparison

Table 5.5 compares some of the major properties of the protocols of this section. A feature of these protocols is that either a signature is required from each principal, or each principal is required to decrypt some protocol parameters with their private key. This means that key compromise impersonation will always be resisted by this kind of protocol. Therefore we omit this attack from consideration in the table but we do consider vulnerability to unknown key-share (UKS) attacks.

Although the IKE main protocol satisfies most of the desirable properties listed in Table 5.5, its large number of message passes is a drawback. In addition, the complexity of the protocol has been heavily criticised. The SKEME protocol is an attractive choice amongst this set of protocols. It is simple and carries a proof of security. As with all these protocols mitigation of denial of service can be added by use of an initial state-free exchange of cookies at the expense of two additional message passes.

Petitioner Samsung Ex-1036, 0203

Table 5.5. Summary of major properties of key agreement protocols using enhanced message format

Properties →	No. of	DoS	Resists	Forward Security	
↓ *Protocol*	passes	mitigation	UKS	secrecy	proof
STS (5.15, 5.17)	3	No	No	Yes	No
Oakley aggressive (5.18)	3	No	Yes	Yes	No
Oakley conservative (5.20)	7	Yes	Yes	Yes	No
SKEME (5.21)	3	No	Yes	Yes	Yes
IKE main (5.22)	6	Yes	Yes	Yes	Yes
Arazi (5.23)	2	No	No	No	Broken
Lim–Lee (5.24, 5.25)	3	No	Yes	No	No
Hirose–Yoshida (5.26)	3	No	No	Yes	No

5.6 Identity-Based Schemes

Identity-based public key cryptography was first proposed by Shamir in 1984 [295]. The idea is to avoid the use of public key certificates by embedding the owner's identity in the public key. The identification information can include any desired information such as real name, physical description and identification numbers. In the following we will use the notation ID_I to denote the identifying string of entity I. In most identity-based protocols it is not acceptable for a random string to be used as an identity, and therefore ID_I is typically calculated using a hash function. We will often write $ID_I = H(I)$ for a suitable hash function H, but in practice the information that is included in the hash may be much more extensive.

Identity-based schemes can be classified into three *levels* using a categorisation of Girault's [115]. In the schemes at level 1 the public key of the entity is exactly the identity string ID_I. This means that once the description of a principal is obtained, its public key can be correctly calculated without recourse to any other information. This is a very attractive way of obtaining keying material but unfortunately there is a significant drawback. No principal can be allowed to generate the corresponding private key. If this were the case then *any* entity could do the same; at the least this would mean that any entity could masquerade as any other. Therefore, in all identity-based schemes of this basic type the private key of each principal is calculated by a trusted third party. Such complete trust in the third party is not reasonable in many situations, although it may be acceptable in a corporate environment. Another important matter, which often goes unmentioned in descriptions of identity-based schemes, is that some mechanism to deal with compromised private keys still needs to be provided, just as with certificate-based schemes.

Petitioner Samsung Ex-1036, 0204

A number of level 1 key establishment protocols were proposed during the 1980s. Later it was realised that similar benefits can be achieved in key establishment protocols even when it is not possible to explicitly extract the public key from the identity. A value obtained from the trusted third party may be used in combination with the partner's identity to derive a key that can only be calculated by a principal with the correct private key. We call such a value an *implicit certificate*. This allows the private keys to be kept secret from the authority and can be regarded as a compromise between the basic identity-based scheme and certificate-based schemes. These schemes reach level 2 in Girault's classification. However, the first proposals in this category suffered from a limitation that we now explain.

A certificate of a public key is no more than a signature by a third party on certain information that includes the public key value. Any principal who receives such a certificate, including the owner of the private key, cannot use it to form another contradictory certificate (for example, one that shows a different public key). This is simply because only the third party can form new signatures. Certainly a malicious certification authority can generate its own private key and produce a certificate that shows that this private key belongs to any victim principal. This would allow the authority to masquerade as the principal. However, this certificate will contradict the real certificate of the principal. Because only the authority is able to produce a certificate the two contradictory certificates can be used to show that the authority has cheated. The problem with the early identity-based protocols, in which principals could choose their own private key, was that the principals could use their private information to form new implicit certificates. Since both the authority and the principal can form contradictory certificates it is impossible to tell which of them has cheated when two certificates appear. This situation is arguably very little better than when the authority has access to the private key, since it is able to masquerade as any principal without being caught.

Finally in 1991 Girault [115] solved this problem by defining *self-certified* public keys. These are keys that have an implicit certificate that can only be generated by the trusted authority. These protocols reach level 3 of Girault's classification. The three different levels of security are summarised in Table 5.6.

Table 5.6. Girault's levels of identity-based schemes

Level	Properties	Example
1	Third party knows secret key	Okamoto
2	Third part cannot find private key. Principal can generate contradictory public key	Girault and Paillès
3	Only third party can generate valid public key	Girault

Petitioner Samsung Ex-1036, 0205

Recently there has been a revival of interest in identity-based cryptography due to the discovery of several new applications of *pairings* on elliptic curves. One of the applications of these techniques is to identity-based key agreement, first recognised by Sakai *et al.* [287]. Dupont and Enge [105] have analysed a non-interactive protocol using pairings; this is analogous to static Diffie–Hellman but does not require certificates. Smart [301] defined an identity-based two-pass authenticated key agreement protocol using pairings. Both these examples are at level 1 in Girault's classification.

We now look at two prominent schemes at level 1 and then compare them with Girault's level 3 scheme.

5.6.1 Okamoto's Scheme

Okamoto's scheme [256, 257] was the first published identity-based key agreement protocol. It uses a composite modulus $n = pq$ whose factorisation is known only to a trusted authority. The authority chooses values e and d as in the RSA algorithm, so that $ed \bmod \phi(n) = 1$, and an element g that is primitive in both the integers mod p and the integers mod q. The values g and e are made public.

Before engaging in the key agreement protocol each user must register with the authority to obtain a private key. User I's identification string, ID_I, is treated as an integer modulo n. The authority calculates the value $s_I = ID_I^{-d} \bmod n$ and distributes s_I securely to user I.

Protocol 5.27 shows the key agreement message flows. The shared secret is defined as $Z_{AB} = g^{er_Ar_B}$. On the assumption that it is necessary to know either s_A or s_B in order to find Z_{AB}, the scheme provides implicit key authentication.

Shared information: Public modulus n and exponent e. Element g of high order in $\mathbb{Z}_n^*$.
Information known to A: s_A such that $s_A^e = ID_A^{-1} \bmod n$.
Information known to B: s_B such that $s_B^e = ID_B^{-1} \bmod n$.

A		B
$r_A \in_R \mathbb{Z}_n$		
$t_A = g^{r_A}$	$\xrightarrow{s_At_A}$	$r_B \in_R \mathbb{Z}_n$
	$\xleftarrow{s_Bt_B}$	$t_B = g^{r_B}$
$Z_{AB} = ((s_Bt_B)^e ID_B)^{r_A}$		$Z_{AB} = ((s_At_A)^e ID_A)^{r_B}$

Protocol 5.27: Okamoto's identity-based protocol

Petitioner Samsung Ex-1036, 0206

It can be seen that the scheme has a close similarity with MTI protocol C(0) (see Table 5.2) despite the different algebraic setting. Like that protocol, the Okamoto protocol is vulnerable to key compromise impersonation. Unknown key-share attacks seem to be prevented on the assumption that users cannot choose their own identities. We need to be careful with regard to whether forward secrecy is provided since compromise of the long-term private key of the trusted authority may allow the adversary to compute discrete logarithms in the group $\mathbb{Z}_n^*$ and thereby find old session keys. Therefore, forward secrecy is only provided if n is large enough that taking discrete logarithms in (at least one of) $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$ is hard.

Mambo and Shizuya [220] conducted a formal proof of security of Protocol 5.27 against a passive eavesdropper. They showed that any efficient algorithm that can find the shared secret can also break the Diffie–Hellman problem in $\mathbb{Z}_n^*$. Later this analysis was extended by Kim *et al.* [182] to show a security proof against active attacks. They provide a reduction of attacks on the protocol to the Diffie–Hellman problem or to the RSA problem. Their model is similar to the Bellare–Rogaway security model in that they define a powerful adversary who can interact with several oracles running protocol instances. However, success of the adversary requires that the complete key is returned, rather than being defined in terms of any partial information on the key being recovered. Therefore it seems that these results are not as strong as they might be.

Protocol 5.28 is a variation of Protocol 5.27 proposed by Okamoto and Tanaka [257] which includes a hashed value to allow explicit authentication. Timestamps, T_A and T_B, are included inside the hashes, c_A and c_B respectively, to ensure freshness. The shared secret is again $Z_{AB} = g^{er_Ar_B}$ but this value is calculated in a different way in this variant.

A similar variant was later published by Shieh *et al.* [296] with claimed computational advantages, but Yen [338] showed that explicit authentication fails. A further variation is to provide one-pass key establishment (previously discussed in Section 5.2.2) suitable for applications such as electronic mail. A scheme originally proposed by Okamoto and Tanaka [257] was shown by Tsai and Hwang [318] to be vulnerable to attacks by insiders, and they proposed new schemes of their own. Later, Tanaka and Okamoto [316] designed another scheme to prevent the trusted authority from obtaining session keys (although the centre can always masquerade as any user). A drawback of all these one-pass schemes is that a file containing auxiliary public information for each user is required and so it seems simpler to use a conventional public key scheme instead of an identity-based one.

Girault and Paillès' protocol [116] has a strong connection with Protocol 5.27. Although the former lies in level 2 of Girault's classification, a comparison of the two is instructive in understanding why level 2 is of marginal improvement over level 1. In Girault and Paillès protocol the public key, y_A, of A satisfies $y_A^e ID_A = g^{-ex_A} \bmod n$, where g^{x_A} is given to the server but x_A is kept secret by A. But if we rearrange this we find that $y_A g^{x_A} = ID^{-d}$

Petitioner Samsung Ex-1036, 0207

Shared information: Public modulus n and exponent e. Element g of high order in $\mathbb{Z}_n^*$.
Information known to A: s_A such that $s_A^e = ID_A^{-1} \bmod n$.
Information known to B: s_B such that $s_B^e = ID_B^{-1} \bmod n$.

A		B
$r_A \in_R \mathbb{Z}_n, u_A = g^{er_A}$		
$c_A = H(u_A, ID_A, ID_B, T_A)$		
$v_A = s_A g^{c_A r_A}$	$\xrightarrow{u_A, v_A}$	$c_A = H(u_A, ID_A, ID_B, T_A)$
		$ID_A \stackrel{?}{=} u_A^{c_A}/v_A^e$
		$r_B \in_R \mathbb{Z}_n, u_B = g^{er_B}$
		$c_B = H(u_B, ID_B, ID_A, T_B)$
	$\xleftarrow{u_B, v_B}$	$v_B = s_A g^{c_B r_B}$
$c_B = H(u_B, ID_B, ID_A, T_B)$		
$ID_B \stackrel{?}{=} u_B^{c_B}/v_B^e$		
$Z_{AB} = u_B^{r_A}$		$Z_{AB} = u_A^{r_B}$

Protocol 5.28: Okamoto–Tanaka identity-based protocol

so that A can calculate y_A herself when the same registration process as that in Okamoto's scheme is used. In the Girault and Paillès protocol A sends the message $y_A g^{x_A - r_A}$ to B. But when we rearrange this message it becomes $ID^{-d} g^{-r_A}$, which is the same as the corresponding message sent in Protocol 5.27 except for a change of sign. Similarly the shared secrets are identical in the two protocols. We conclude that there is essentially no difference between the two even though the former is classified as level 1 and the latter as level 2.

5.6.2 Günther's Scheme

Günther [138] proposed an identity-based scheme in the familiar setting of $\mathbb{Z}_p^*$. The trusted authority S has private and public key pair x_S and $y_S = g^{x_S}$. Again ID_I is a description of the identity of user I, which Günther suggests should be a one-way function of any relevant characteristics of I. We have denoted this in the protocol description as $ID_I = H(I)$. At the registration phase user I obtains an ElGamal signature (u_i, v_i) generated by S by choosing k_i randomly in $\mathbb{Z}_p^*$ (but prime to $p-1$) and setting $u_i = g^{k_i}$, $v_i = (ID_I - x_S u_i)/k_i \bmod p-1$. The signature pair (u_i, v_i) is given to I. The verification equation of the ElGamal signature scheme can be written as:

$$u_i^{v_i} = g^{ID_I} y_S^{-u_i}.$$

Petitioner Samsung Ex-1036, 0208

Although any party can verify this equation, the idea is not to reveal v_i but rather that I is the only party who is in possession of the discrete logarithm of $g^{ID_I} y_S^{-u_i}$ to the base u_i. Protocol 5.29 shows a successful protocol run. The shared secret is $Z_{AB} = u_A^{r_B v_A} u_B^{v_B r_A}$.

Shared information: Public key y_S of S.
Information known to A: ElGamal signature of S on ID_A, (u_A, v_A), so that $g^{ID_A} = y_S^{u_A} u_A^{v_A}$.
Information known to B: ElGamal signature of S on ID_B, (u_B, v_B), so that $g^{ID_B} = y_S^{u_B} u_B^{v_B}$.

A		B
	$\xrightarrow{A, u_A}$	$ID_A = H(A)$
$ID_B = H(B)$	$\xleftarrow{B, u_B}$	$X_B = g^{ID_A} y_S^{-u_A}$
$X_A = g^{ID_B} y_S^{-u_B}$		
$r_A \in_R \mathbb{Z}_q$		
$w_A = u_B^{r_A}$	$\xrightarrow{w_A}$	$r_B \in_R \mathbb{Z}_q$
	$\xleftarrow{w_B}$	$w_B = u_A^{r_B}$
$Z_{AB} = w_B^{v_A} X_A^{r_A}$		$Z_{AB} = w_A^{v_B} X_B^{r_B}$

Protocol 5.29: Günther's key agreement protocol

There is a similarity with MTI protocol A(0) and consequently it seems reasonable to assume that one of v_A or r_B, and one of v_B or r_A, are required to find Z_{AB}. With this assumption it follows that resistance to key compromise impersonation is provided. Like Okamoto's protocol, if users cannot choose their own identities then the unknown key-share attacks on the MTI protocols do not carry over. It can be seen that compromise of the long-term secrets v_A and v_B enables the adversary to find old session keys, so that forward secrecy is not provided. Therefore Günther also proposed Protocol 5.30 as an extension of the basic protocol that provides forward secrecy at the cost of an extra exponentiation on each side. A and B choose additional random values, r'_A and r'_B respectively. This time the shared secret is the same as in the basic protocol multiplied by the ephemeral Diffie–Hellman key: $Z_{AB} = u_A^{r_B v_A} u_B^{v_B r_A} g^{r'_A r'_B}$. Burmester [74] showed that his triangle attack is applicable to both versions of the protocol.

Like Okamoto's scheme, a drawback is that the trusted authority has access to users' private keys so that the protocol only reaches level 1. Another potential disadvantage is that four messages are required, although it is worth noting that messages 2 and 4 can be combined in both protocol versions. It should also be remembered that no conventional certificates are required, so

Petitioner Samsung Ex-1036, 0209

Shared information: Public key y_S of S.
Information known to A: ElGamal signature of S on ID_A, (u_A, v_A), so that $g^{ID_A} = y_S^{u_A} u_A^{v_A}$.
Information known to B: ElGamal signature of S on ID_B, (u_B, v_B), so that $g^{ID_B} = y_S^{u_B} u_B^{v_B}$.

A		B
	$\xrightarrow{A, u_A}$	$ID_A = H(A)$
$ID_B = H(B)$	$\xleftarrow{B, u_B}$	$X_B = g^{ID_A} y_S^{-u_A}$
$X_A = g^{ID_B} y_S^{-u_B}$		
$r_A, r'_A \in_R \mathbb{Z}_q$		
$w_A = u_B^{r_A}$		
$t_A = g^{r'_A}$	$\xrightarrow{w_A, t_A}$	$r_B, r'_B \in_R \mathbb{Z}_q$
		$w_B = u_A^{r_B}$
	$\xleftarrow{w_B, t_B}$	$t_B = g^{r'_B}$
$Z_{AB} = w_B^{v_A} X_A^{r_A} t_B^{r'_A}$		$Z_{AB} = w_A^{v_B} X_B^{r_B} t_A^{r'_B}$

Protocol 5.30: Günther's extended key agreement protocol

in practice there may be savings in comparison with other protocols that may require additional messages to obtain the necessary public key certificates.

Saeednia [282] has proposed Protocol 5.31 as a variant of Günther's scheme that reduces the number of messages to only two. The idea is to replace the equation for the user's secret v_i with $v_i = ID_I k_i + x_S u_i \bmod (p-1)$ while, as before, the public key is $u_i = g^{k_i}$. Consequently A can generate her random value, $t_A = g^{r_A}$, for the protocol without waiting to receive B's public value u_B. The shared secret becomes $Z_{AB} = g^{v_A r_B + v_B r_A}$. The change effectively replaces ElGamal's signature in Günther's protocol with a variant signature (in fact it is variant 3 in Table 11.5 of the *Handbook of Applied Cryptography* [234]). Saeednia also showed how to add forward secrecy to Protocol 5.31 without further message exchanges.

5.6.3 Girault's Scheme

The algebraic setting for Girault's key agreement scheme [115] using self-certified keys is the same as that used for Okamoto's protocol (5.27). An RSA modulus n and key pair e, d are chosen by the server, together with an element g of high order in $\mathbb{Z}_n^*$. When A registers to use the scheme she chooses her private key x_A and provides g^{x_A} to S, who calculates the self-certified public key $y_A = (g^{x_A} - ID_A)^d \bmod n$. (We have changed the sign of x_A from Girault's original in order to maintain our usual notation.) In order for the scheme to

Petitioner Samsung Ex-1036, 0210

Shared information: Public key y_S of S.
Information known to A: Signature of S on ID_A, (u_A, v_A), so that $g^{v_A} = y_S^{u_A} u_A^{ID_A}$.
Information known to B: Signature of S on ID_B, (u_B, v_B), so that $g^{v_B} = y_S^{u_B} u_B^{ID_B}$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{A, u_A, t_A}$	$r_B \in_R \mathbb{Z}_q$
	$\xleftarrow{B, u_B, t_B}$	$t_B = g^{r_B}$
$ID_B = H(B)$		$ID_A = H(A)$
$X_A = u_B^{ID_B} y_S^{u_B}$		$X_B = g^{ID_A} y_S^{u_A}$
$Z_{AB} = t_B^{v_A} X_A^{r_A}$		$Z_{AB} = t_A^{v_B} X_B^{r_B}$

Protocol 5.31: Saeednia's variant of Günther's key agreement protocol

achieve level 3 it is essential that S is unable to find x_A. Saeednia [283] has pointed out that a malicious server could choose n so that finding discrete logarithms is easy, and for this reason the size of n should preferably be 2048 bits. In addition S should provide a proof that n is the product of two safe primes; Camenisch and Michels [80] provide a method to achieve such a proof.

In Girault's original short paper he only mentioned the possibility of using the self-certified keys to produce an authenticated static shared key. If the public keys are already available this can be achieved with no message exchanges: A calculates $(y_B^e + ID_B)^{x_A}$ in order to produce $Z_{AB} = g^{x_A x_B}$ and B calculates the analogous value. However, since the public key is simply a way to find an implicitly certified key for each party, any of the MTI protocols can be modified to provide a dynamic protocol. For example, Protocol 5.32 shows a version modified from MTI A(0), as suggested by Rueppel and van Oorschot [279]. The shared key is $Z_{AB} = g^{r_A x_B + r_B x_A}$. Of course the usual extra checks should be made in order to avoid the potential attacks, as discussed in Sect. 5.3. Analogues of the other MTI protocols can be made similarly, replacing y_A in the MTI original with $y_A^e + ID_A$ in the Girault version, and similarly for y_B.

Later Saeednia [284] described two identity-based schemes, but there seems little to recommend these over the previously published protocols. Saeednia's first protocol is at level 1 on Girault's scale, but it is not as efficient as Okamoto's. His second protocol was claimed to achieve level 3, but Kim *et al.* [183] showed that only level 2 was achieved and proposed a small change to remedy this. In any case, although there may be a very small reduction in storage requirements, the protocol computational requirements are higher than in Girault's.

Petitioner Samsung Ex-1036, 0211

Shared information: Public modulus n and exponent e. Element g of high order in $\mathbb{Z}_n^*$.
Information known to A: x_A with $y_A^e + ID_A = g^{x_A} \bmod n$.
Information known to B: x_B with $y_B^e + ID_B = g^{x_B} \bmod n$.

A		B
$r_A \in_R \mathbb{Z}_n$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_n$
	$\xleftarrow{t_B}$	$t_B = g^{r_B}$
$Z_{AB} = t_B^{x_A}(y_B^e + ID_B)^{r_A}$		$Z_{AB} = t_A^{x_B}(y_A^e + ID_A)^{r_B}$

Protocol 5.32: Girault's identity-based protocol

5.6.4 Schemes Using Signatures with Message Recovery

Nyberg and Rueppel [253] suggested using their signature with message recovery, already discussed in Sect. 5.2.2, as the basis of identity-based key agreement. The idea is to make the public key of each principal the signature of $g^{x_A} ID_A^{-1}$. After recovery of the message from the signature the implicitly certified public key can be recovered by multiplying by ID_A. With this basis, any of the MTI-style protocols can be applied, in the same manner as explained in the previous section. Sakazaki *et al.* [288] explored use of elliptic curves and other variations in order to make the basic idea more efficient.

5.7 Protocols Designed for Computationally Limited Devices

Despite the inexorable increase in the availability of computing resources there has been considerable interest in protocols that can be implemented on devices with limited computational power. Typical examples of such devices are mobile terminals and embedded hardware. Very often the low-power device is a client required to establish a key with a computationally powerful server and so an acceptable solution may have *unbalanced* computational requirements: the server end can bear an increased computational load in order to ease the burden on the client side. Another technique that is effective is to allow the client side to pre-compute values which can be used during the protocol execution; mobile terminals typically have the opportunity to make off-line computations during the idle time between awaiting calls. Horn *et al.* [145] provide a survey and comparison of several protocols proposed for use by mobile devices.

Petitioner Samsung Ex-1036, 0212

5.7.1 Yacobi–Shmuely Protocol

Although not itself designed for low-power computational devices, the Yacobi–Shmuely protocol [336] has been adapted for this purpose as described below. Protocol 5.33 differs from most other Diffie–Hellman-based protocols in that the exchanged values are not exponentiations, but rather the random exponents themselves. The shared secret is $Z_{AB} = g^{r_A r_B}$.

Shared information: Modulus n. Element g of high order in $\mathbb{Z}_n^*$.

A		B
$r_A \in_R \mathbb{Z}_n$		
$u_A = r_A + x_A$	$\xrightarrow{u_A}$	$r_B \in_R \mathbb{Z}_n$
	$\xleftarrow{u_B}$	$u_B = r_B + x_B$
$Z_{AB} = (g^{u_B} y_B^{-1})^{r_A}$		$Z_{AB} = (g^{u_A} y_A^{-1})^{r_B}$

Protocol 5.33: Yacobi–Shmuely protocol

Yacobi and Shmuely proved that their protocol is secure against passive eavesdroppers but there is an active attack on the protocol found by Yacobi [335]. It applies if the adversary C is able to obtain a shared secret Z_{AB} from an old protocol run, for which the protocol messages have been previously recorded. Thus C has the old values $Z_{AB} = g^{r_A r_B}$, $r_A + x_A$ and $r_B + x_B$. The adversary C now starts a new run of the protocol with the entity A and replays the message $r_B + x_B$ sent by B in the old protocol run. Entity A will send a new message $r'_A + x_A$, but by subtracting the old message $r_A + x_A$ from this, C obtains $r'_A - r_A$. The adversary is now able to find the new shared secret $Z'_{AB} = g^{r'_A r_B}$ by the calculation $Z'_{AB} = (g^{x_B + r_B} y_B^{-1})^{(r'_A - r_A)} \cdot Z_{AB}$.

An essentially identical protocol was subsequently published by Alexandris *et al.* [10] and it was pointed out by Nyberg and Rueppel [254] that a similar attack applies.

Park [261] proposed a protocol for use in a wireless environment that is based on the Yacobi–Shmuely protocol, but with a small and significant difference. In Protocol 5.34 the server or base station, A, carries out the exponentiation using u_A on behalf of the mobile station, B, with the aim of reducing the computational load on B. Apart from this change in where the computation takes place, the protocol (including the shared secret) is the same as Protocol 5.33. (We have omitted certain extra fields used for key confirmation which have no bearing on this discussion.)

The replay attack on Protocol 5.33 still applies, but Boyd and Park [64] showed that an even stronger attack is possible. Any entity C can execute this protocol successfully with the entity B without the knowledge of the private

Petitioner Samsung Ex-1036, 0213

Shared information: Modulus n. Element g of high order in $\mathbb{Z}_n^*$.

A		B
$r_A \in_R \mathbb{Z}_n$		
$u_A = g^{r_A + x_A}$	$\xrightarrow{u_A}$	$r_B \in_R \mathbb{Z}_n$
	$\xleftarrow{u_B}$	$u_B = r_B + x_B$
$Z_{AB} = (g^{u_B} y_B^{-1})^{r_A}$		$Z_{AB} = (u_A y_A^{-1})^{r_B}$

Protocol 5.34: Park key agreement protocol

key of A. This can be seen in Attack 5.10. The adversary C calculates the session key in exactly the same way as A does in a normal protocol run, since it is not necessary to know the value of x_A to calculate $Z_{AB} = (y_B g^{x_B + r_B})^{r_A} = g^{r_A r_B}$. Principal B believes that Z_{AB} is shared with A whereas it is actually shared with C.

Shared information: Modulus n. Element g of high order in $\mathbb{Z}_n^*$.

C		B
$r_C \in_R \mathbb{Z}_n$		
$u_C = y_A g^{r_C}$	$\xrightarrow{u_C}$	$r_B \in_R \mathbb{Z}_n$
	$\xleftarrow{u_B}$	$u_B = r_B + x_B$
$Z_{CB} = (g^{u_B} y_B^{-1})^{r_C}$		$Z_{AB} = (u_C y_A^{-1})^{r_B}$

Attack 5.10: Attack on Park key agreement

5.7.2 ASPeCT Protocol

ASPeCT was a European collaborative project concerned with future mobile communications. The work included specification of a candidate protocol for authentication and key establishment based on asymmetric cryptography [146, 145]. The protocol is intended for use by mobile users to establish a secure session with either a fixed network or a value-added service provider (VASP). There are a number of slightly different versions; Protocol 5.35 describes what seems to be the most recent [145]. As usual we omit the certificates from our description, as well as other parameters used for charging and payment data.

Protocol 5.35 does not provide forward secrecy since knowledge of the long-term key x_B is sufficient to calculate K_{AB} from the protocol messages.

Petitioner Samsung Ex-1036, 0214

Shared information: Three hash functions h_1, h_2 and h_3.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
$K_{AB} = h_1(r_B, y_B^{r_A})$	$\xleftarrow{r_B, h_2(K_{AB}, r_B, B)}$	$K_{AB} = h_1(r_B, t_A^{x_B})$
$H = h_3(t_A, y_B, r_B, B)$	$\xrightarrow{\{Sig_A(H)\}_{K_{AB}}}$	Decrypt and verify

Protocol 5.35: ASPeCT protocol

However, since either x_B or r_A is required to find K_{AB} and also A provides a signature, key compromise impersonation seems impossible. Notice that principal A, which represents the mobile user, can calculate all exponentials before the protocol is run, as long as the public key of B is known in advance. The signature generation algorithm can also be mostly performed off-line if a suitable algorithm, such as DSA, is used. It is for this reason that a protocol of this sort may be preferred for a resource-limited device, even though forward secrecy is not provided.

5.7.3 Jakobsson–Pointcheval Protocol

Jakobsson and Pointcheval's protocol [167] is related to Protocol 5.35 and calculates the shared secret in almost the same way. In Protocol 5.36 the computationally limited entity A is able to complete all the computations required to send message 1 off-line as long as the public key of the server, B, is previously available. This means that only simple calculations need to be performed by A during the protocol run itself. As with Protocol 5.35 a by-product of the improved efficiency is that forward secrecy is not provided, since once x_B is compromised any previously agreed session keys become available.

A difference between Protocols 5.35 and 5.36 is that the shared key K_{AB} in Protocol 5.36 has no fresh input from B, whereas in Protocol 5.35 both principals have a random input to K_{AB}. This means that in Protocol 5.36, B must trust A to generate a new session key. The pair (d, r) forms a signature on the shared secret K. Since B has a fresh input, e, to the signature, he can be sure that A chose the input t_A for this protocol run, so a simple replay attack is not possible.

Protocol 5.36 shows a revised version of the protocol. In the original version the message field r was defined by $r = h_2(g^{r'_A})$; this means that there is nothing to connect the signature (d, r) to the session key which allows an impersonation attack as shown by Wong and Chan [328]. An interesting aspect of this attack is that the original protocol already carried a proof of security in

Petitioner Samsung Ex-1036, 0215

Shared information: Three hash functions h_1, h_2, h_3 with outputs of lengths l_1, l_2, l_3 respectively. Security parameter k (suggested value $k = 64$).

A		B
$r_A, r'_A \in_R \mathbb{Z}_q$		
$K = y_B^{r_A}$		
$r = h_2(g^{r'_A}, y_B, t_A, K)$		
$t_A = g^{r_A}$	$\xrightarrow{B, t_A, r}$	$K = t_A^{x_B}$
		$X = h_3(y_B, t_A, K)$
	$\xleftarrow{X, e}$	$e \in_R \{0,1\}^k$
$X \stackrel{?}{=} h_3(y_B, t_A, K)$		
$d = r'_A - ex_A \bmod q$	$\xrightarrow{A, d}$	$r \stackrel{?}{=} h_2(g^d y_A^e, y_B, t_A, K)$
	$K_{AB} = h_1(y_A, t_A, K)$	

Protocol 5.36: Jakobsson–Pointcheval protocol

the Bellare–Rogaway model, bringing into question the value of the security proof in helping to avoid basic errors.

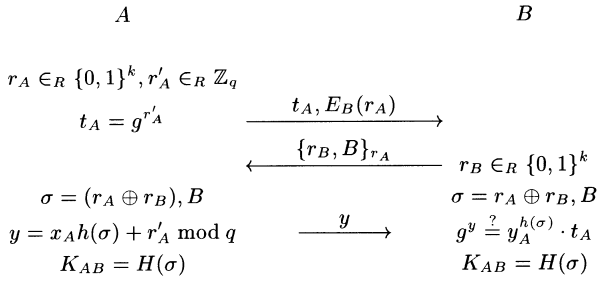
Wong and Chan [328] proposed Protocol 5.37 as an improvement to Protocol 5.36. The general design is similar but it uses a generic encryption function to send the keying material from A to B. Again, most of the computation for A can be carried out off-line if the public key of B is known. The symmetric encryption using r_A in the second message is not clearly defined, but evidently must provide both integrity of the identity B in addition to confidentiality of r_B. As with the previous protocols in this section, forward secrecy is not provided since if B's private key is compromised then both r_A and r_B may be found, revealing the session key.

Protocol 5.37 is proven secure in the Bellare–Rogaway model, but in an interesting echo of the problem with Protocol 5.36 the published version of the protocol [328] is missing the identity B from the definition of field σ. Without this there is again a straightforward impersonation attack, in which the adversary masquerades as a different server to A, but completes the protocol with B impersonating as A. It seems difficult to pinpoint how this difference affects the proof given in the paper of Wong and Chan.

A related protocol was previously published by Boyd and Park [64]. It has the same basic structure, but uses a generic encryption algorithm in the first message and a generic digital signature in the third message.

Petitioner Samsung Ex-1036, 0216

Shared information: Three hash functions h_1, h_2, h_3. Security parameter k.



Protocol 5.37: Wong–Chan protocol

5.8 Protocols in ISO/IEC 11770-3

The international standard ISO/IEC 11770-3 [162] is devoted to key management techniques using public key (asymmetric) techniques. The key transport protocols in this standard are discussed in Chap. 4. There are seven key agreement mechanisms in the standard; these may be regarded as frameworks for detailed protocols into which fall many of the protocols we have examined. All the key agreement protocols in the standard are based on the Diffie–Hellman exchange.

Key Agreement Mechanism 1

This is a no-message protocol. It is simply the static Diffie–Hellman protocol.

Key Agreement Mechanisms 2 and 3

These have one message exchange. Mechanism 2 involves a random input from A to form an ElGamal encryption key. Mechanism 3 includes authentication information from A in addition. The Nyberg–Rueppel protocol (5.3) fits into this category.

Key Agreement Mechanisms 4 and 5

These are two message exchanges in which both principals send an ephemeral public key. Mechanism 4 is simply ephemeral Diffie–Hellman. Mechanism 5 uses the long-term and ephemeral keys in the calculation of the shared secret. The MTI protocols (see Sect. 5.3) fit into mechanism 5.

Petitioner Samsung Ex-1036, 0217

Key Agreement Mechanism 6

This unusual mechanism is related to Protocol 4.12 examined in Chap. 4 but instead of transporting a generated key, the signature used to respond to a challenge is used as the session key. Without any special properties of the signature this is a questionable approach. The protocol cannot provide forward secrecy. The standard claims that an example of this mechanism is the Beller–Yacobi protocol [37] but it is difficult to recognise any close similarity.

Key Agreement Mechanism 7

This three-message protocol is the same as Protocol 5.16, with the exception that a MAC is added into the last two messages to provide explicit key confirmation in the manner described in Sect. 5.4.8.

5.9 Diffie–Hellman Key Agreement in Other Groups

Diffie–Hellman key agreement was originally proposed in the algebraic setting of the multiplicative group $\mathbb{Z}_p^*$ and we have used this setting in all our descriptions so far. It has long been known that the basic structure can be generalised to any commutative group. In this section we mention some of the most prominent alternative groups that have been proposed.

Elliptic curve groups have significant potential advantages over using $\mathbb{Z}_p^*$ because of their greater efficiency and compact representation. Many recent protocols have been specially designed with elliptic curve implementation in mind rather than using prime fields. Examples include the MQV (5.11) and Oakley (5.18) protocols. The Oakley specification includes a number of candidate elliptic curves and also provides for negotiation of new curves during the protocol. It is sometimes possible to avoid certain attacks because of the structure of the curve used. For example, elliptic curve groups can be chosen to have prime order so that there is no need to check whether elements are in a particular subgroup.

Hyperelliptic curves are generalisations of elliptic curves. The *jacobian group* of a hyperelliptic curve is another setting that has been suggested for the Diffie–Hellman technique [186]. It seems that there are many open questions with respect to the security and best implementations in this setting. Smart and Siksek [302] have proposed to use a different structure on hyperelliptic curves.

Extension fields have been used by several authors. For example, Scheidler *et al.* [290] give details of Diffie–Hellman in real quadratic fields. Brouwer *et al.* [71] designed a Diffie–Hellman version using a subgroup of the extension field $GF(p^6)$ which has attractive performance properties. This is related to Lenstra and Verheul's XTR group [204] which can also be used for Diffie–Hellman.

Petitioner Samsung Ex-1036, 0218

$\mathbb{Z}_n^*$ McCurley [227] designed a special version of Diffie–Hellman key agreement in $\mathbb{Z}_n^*$ when n is the product of two primes: $n = pq$. He showed that breaking this version is equivalent to both factorising n and breaking Diffie–Hellman in the two subgroups $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$. Scott [292] extended Protocol 5.27 to this setting to obtain an identity-based key agreement protocol for which it is proven that obtaining the shared secret from the messages exchanged (and any other public information) is equivalent to factorising the modulus.

5.10 Protocols Not Based on Diffie–Hellman

Key agreement protocols without Diffie–Hellman usually possess a computational advantage, but lack forward secrecy. Protocol 4.16 can also be regarded as a key agreement protocol in this class.

5.10.1 SKEME without Forward Secrecy

The SKEME protocol described in Sect. 5.5.3 has versions with and without forward secrecy. Protocol 5.38 shows the mode without forward secrecy; it replaces the Diffie–Hellman exchange of Protocol 5.21 with a simple exchange of nonces chosen by A and B respectively.

Shared information: Security parameter L.

A		B
$N_A \in_R [0, 2^L]$		
$r_A \in_R \mathbb{Z}_q$	$\xrightarrow{E_B(A, N_A), r_A}$	$N_B \in_R [0, 2^L]$
		$r_B \in_R \mathbb{Z}_q$
Verify MAC	$\xleftarrow{E_A(N_B), r_B, MAC_{K_0}(r_A, r_B, B, A)}$	$K_0 = H(N_A, N_B)$
$K_0 = H(N_A, N_B)$	$\xrightarrow{MAC_{K_0}(r_B, r_A, A, B)}$	Verify MAC

$$K_{AB} = MAC_{K_0}(MAC_{K_0}(r_B, r_A, A, B))$$

Protocol 5.38: SKEME protocol without forward secrecy

The temporary shared secret K_0 is calculated as $K_0 = H(N_A, N_B)$. The session key is calculated by applying the MAC function to the final message.

$$K_{AB} = MAC_{K_0}(MAC_{K_0}(r_B, r_A, A, B)).$$

Petitioner Samsung Ex-1036, 0219

Evidently N_A and N_B, and consequently K_{AB}, are revealed if the long-term decryption keys of A and B become known, so forward secrecy is not obtained. However, use of the public key encryption prevents key compromise impersonation.

The SKEME re-keying protocol is identical to Protocol 5.38 with the encrypted values in messages 1 and 2 omitted. Instead the previous K_0 value is reused to find a new K_{AB} value based on the newly chosen nonces. There is also a version of the Oakley protocol without Diffie–Hellman, whose structure is very similar to Protocol 5.38.

5.10.2 Key Pre-distribution Schemes

Over the years a number of key establishment protocols have been proposed in which each user stores a relatively large amount of keying material locally. In order to agree a static key with another user a shared subset of the keying material is selected, and used as a shared secret. Although these schemes may not be strictly regarded as key agreement protocols, they can be seen as analogous to static Diffie–Hellman and therefore we mention them here.

The general idea of such schemes is that a trusted server will generate a set of secret elements and distribute a subset of these to each user. Each pair of users must be able to generate a shared secret, so the intersection of their secret element sets must be non-empty. Furthermore, this intersection must be unique to those two users to ensure that no other user can derive the same secret. Further properties may be desirable. In particular, it is often required that any (small) subset of users should be unable to derive the keys of other users, in order to prevent coalitions of users from attacking other users.

Blom [48, 49] seems to have been the first to address this problem. He presented solutions based on coding theory which can be guaranteed to provide an optimal solution in terms of the storage required by each principal. Jansen [168] proposed a potentially more efficient scheme but Mitchell and Piper [237] showed that it can be insecure. Gong and Wheeler [133] considered a geometrically-based scheme. Users are given sets of keys that define lines in the integers modulo a suitable number. The shared secret then consists of the keys that lie at the intersection of the lines they hold.

Later Leighton and Micali [203] proposed a simple scheme in which users' keys are stored in tamper-resistant devices. This assumption greatly eases the problem of avoiding coalition attacks. Here we outline the simplest form of one of their schemes.

1. The trusted authority T chooses M k-bit secret elements $K_1, \ldots, K_M$.
2. An identity bit-string $(\alpha_1, \ldots, \alpha_M)$ is generated as a public function of the user's identity.
3. The user with identity string $(\alpha_1, \ldots, \alpha_M)$ is allocated a secret sequence $(S_1, \ldots, S_M)$ where

$$S_j = \begin{cases} K_j & \text{if } \alpha_j = 0 \\ h(K_j) & \text{if } \alpha_j = 1. \end{cases}$$

Petitioner Samsung Ex-1036, 0220

When principals A and B, with respective identity strings $(\alpha_1, \ldots, \alpha_M)$ and $(\beta_1, \ldots, \beta_M)$, wish to generate a shared secret they simply form the string $X_{AB} = X_1, X_2, \ldots, X_M$ where

$$X_j = \begin{cases} K_j & \text{if } \alpha_j = 0 \text{ and } \beta_j = 0 \\ h(K_j) & \text{otherwise.} \end{cases}$$

Leighton and Micali omitted to include the user identities when defining how to obtain the key from the shared secret. Zheng [340] pointed out that this omission can allow an attack in which the adversary with identity string $(\gamma_1, \ldots, \gamma_M)$ can find the shared key between A and B as long as $\gamma_j = 0$ when both $\alpha_j = 0$ and $\beta_j = 0$. Therefore he suggests to derive the shared key by hashing the shared secret together with the identities of the principals.

All these schemes are suitable only for environments in which a small number of principals are fixed in advance. A typical application is for encryption between nodes in a network. Two limitations are that users need to store a relatively large amount of keying material and that the keys agreed are static. This latter problem may be addressed by including a timestamp to vary the session key agreed each time.

5.11 Conclusion

The number of key agreement protocols that we have examined in this chapter gives some indication of their importance, both in the research literature and in practical applications. Protocols based on Diffie–Hellman key exchange are by far the most common, but come in various different types. In particular we distinguished between those in which the exchanged information consists only of the Diffie–Hellman messages, those which exchange additional authenticating information, and those which are identity-based. This last class of key agreement protocols has been of increasing interest lately due to new constructions using pairings on elliptic curves.

The number of proposed key agreement protocols continues to increase. Recent proposals for an IKE replacement have shown that new ideas are still possible and the range of additional security requirements is very large, with anonymity and protection against denial of service becoming increasingly seen as part of a standard list. Certainly understanding of how to properly design such protocols has increased immensely in recent years, but there are still some aspects that have not been covered in formal analyses. It seems likely that there will continue to be some interesting developments in the design of key agreement protocols in the coming few years.

Petitioner Samsung Ex-1036, 0221

6

Conference Key Protocols

6.1 Introduction

As electronic communications and information services become more sophisticated, many applications involving multiple entities become necessary. Since these applications will generally require secure communications it is necessary to design protocols that establish keys for groups of principals. There is a great variety of different practical requirements that may be appropriate in different applications, and the number of protocols is very large. In this chapter we will mainly restrict attention to ways in which the two-party protocols that have been explored in previous chapters can be generalised to the multi-party situation.

Just as with two-party key establishment, different types of protocol are appropriate depending on the application. Applications such as video and audio conferencing may have very different requirements from other applications such as satellite TV or Internet video broadcasting. Hardjono and Tsudik [141] discuss the following four factors that influence the requirements for such protocols.

Application Type A fundamental feature is how many of the parties must be able to send information. In a corporate teleconference all parties may wish to transmit. In a satellite broadcast to a large group there is only one sender. There may be many intermediate cases too.

Group Size and Dynamics For small groups it is feasible for all principals to take part in interactive key establishment; for very large groups it becomes impractical. Some protocols can easily accommodate addition and deletion of members or subsets, while for others there may be a significant computation or communication cost.

Scalability The efficiency of protocols may vary as the size of the group of users changes.

Trust Model It is important to define which principals are trusted to generate and authenticate keys.

Petitioner Samsung Ex-1036, 0222

Similar properties, as well as a number of typical application scenarios, are considered by Canetti *et al.* [81]. Detailed study of these issues is outside the scope of this book. However, in certain cases we are able to make some comments regarding these matters.

The rest of this section discusses the generalisations of the security goals of two-party key establishment that are relevant for multi-party key establishment. In the next section many generalisations of Diffie–Hellman key agreement are described and analysed. Section 6.3 describes protocols that provide authenticated conference key agreement by adding authentication to the generalisations of Diffie-Hellman key agreement. Section 6.4 looks at identity-based multi-party protocols while Sect. 6.5 is concerned with some proposals for conference key agreement that do not use Diffie–Hellman as a basis. We then look at multi-party key transport protocols in Sect. 6.6, including the important idea known as *logical key hierarchy.* The final main section introduces protocols for broadcasting keys.

6.1.1 Generalised Security Goals

Before discussing some of the different types of conference key protocols we first consider how the basic definitions for key establishment can be extended to multi-party protocols. The definition of a good shared key given in Chap. 2 can be generalised to a conference key as follows.

Definition 6.1. *The shared session key is a* good key *for principal U_i to use with the set of principals $\mathcal{U}$ only if U_i has assurance that:*

- *the key is fresh (***key freshness***);*
- *the key is known only to principals in $\mathcal{U}$ (***key authentication***).*

If the conference is large it may be too expensive, in terms of both communications and computation, for each principal to receive and verify authentication information from all other group members. Instead many protocols simply allow each group member to implicitly authenticate the key with respect to only one other group member. In a multi-party key transport protocol it is natural for the principal distributing the key to provide key authentication. In a multi-party key agreement protocol group members communicate with each other in some systematic way. For example, members are often arranged in a logical ring and each principal's key input can be authenticated to the 'next' member. This may be regarded as a weaker form of authentication since each principal is relying on every other conference member to check the authenticity for the whole group.

Generalisations of enhanced protocol properties can be tricky. We consider in particular the important property of *key confirmation.* The potential problems are illustrated by considering the definition given in the *Handbook of Applied Cryptography* [234] which was discussed in Chap. 2.

Petitioner Samsung Ex-1036, 0223

> *Key confirmation* is the property whereby one party is assured that a second (possibly unidentified) party actually has possession of a particular secret key.

For a two-party protocol this definition is useful: if A knows that the key is good for use with B, and that a different entity has the key, then A knows that B must have the key. But in a conference it may be of quite limited use to know only that some other member of the set of principals has the key.

Saeednia and Safavi-Naini [285] suggest that every principal should be sure that either the same key is shared with *all* other principals or that no two principals share the same key. In particular they consider that a situation in which a session key is established by a subset of the intended set of principals, but which is not known to other members of the set, is a major threat. In contrast Ateniese *et al.* [18, 19] note that key confirmation for all users requires 'at the very least, one round of simultaneous broadcasts', implying that it may be too costly to justify.

Instead of making a judgement on whether or not conference key protocols ought to provide key confirmation we simply generalise our definition from Chap. 2 so that we can examine which protocols provide the property.

Definition 6.2. *Let $\mathcal{U}$ be a set of principals with $U_i, U_j \in \mathcal{U}$.* Key confirmation *of U_j to U_i is provided if U_i has assurance that key K is a good key to communicate with every principal in $\mathcal{U}$, and principal U_j has received K.* Complete key confirmation *is provided to U_i if key confirmation of U_j is provided to U_i for all $U_j \in \mathcal{U} \setminus \{U_i\}$.*

6.1.2 Static and Dynamic Groups

After a multi-party key has been established between a group of principals it may be desired either to add principals to the group or to remove principals from it. Of course this may be done by simply re-starting the protocol to establish a new key for the modified group. However, this may be inconvenient, especially if the group is large or the full protocol is computationally expensive. Therefore many conference key protocols have been designed, or optimised, in order to allow members to be easily added or removed.

Notice that this issue does not apply to key establishment protocols between two principals and so we have a new way to classify protocols as catering for either *static groups* in which the principals are fixed, or *dynamic groups* in which the principals can change. Protocols for dynamic groups include sub-protocols for joining and removing of principals. They may also include sub-protocols for *merging* two groups together or *partitioning* a group into subgroups.

New security goals may also be required for dynamic groups. Kim *et al.* [184] define three new properties that may be applicable.

Petitioner Samsung Ex-1036, 0224

Forward secrecy An adversary who knows a set of group keys cannot derive any subsequent group key. (This is an unfortunate conflict with the more usual meaning of the term forward secrecy.)

Backward secrecy An adversary who knows a set of group keys cannot find any earlier group key.

Key independence An adversary who knows any set of group keys cannot find any other group key.

The motivation for forward secrecy (in this sense) is that any group member who leaves the group should not be able to learn any new group keys after leaving. Similarly, backward secrecy ensures that new group members cannot learn old group keys. Key independence is the strongest of the three properties, and implies the other two.

In this chapter we will concentrate mainly on static groups, generalising the protocols in earlier chapters. However, we will mention extensions to dynamic group protocols in many cases.

6.1.3 Notation

The notation used in this chapter is shown in Table 6.1. Because many conference key protocols are based on generalisations of Diffie–Hellman key exchange there is much similarity with the notation used in Chap. 5.

6.2 Generalising Diffie–Hellman Key Agreement

There are different ways that the basic Diffie–Hellman two-party key agreement can be generalised to a multi-party protocol. In this section we will look at several of the natural ways to do this without considering how to use them to provide authenticated key agreement. In Sect. 6.3.1 we outline how these protocols can be incorporated into conference key protocols to provide establishment of good keys.

6.2.1 Ingemarsson–Tang–Wong Key Agreement

Ingemarsson *et al.* [156] considered a model in which principals are connected in a ring, so that principal U_i receives messages only from U_{i-1} and sends messages only to U_{i+1}. To enable a general description we allow any index i but U_i and U_j are the same principal when $i \equiv j \bmod m$. Ingemarsson *et al.* described a number of Diffie–Hellman generalisations based on the idea of *symmetric functions.*

Definition 6.3. *The j'th symmetric function on the set $S = \{r_1, \ldots, r_m\}$ is denoted $S^{(j)}$ and consists of the sum of all possible products of j distinct elements of S.*

Petitioner Samsung Ex-1036, 0225

Table 6.1. Notation for conference key protocols

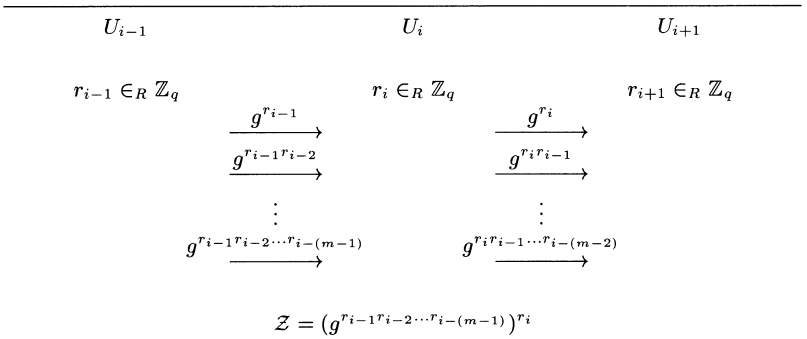
p	A large prime (usually at least 1024 bits).
q	A prime with $q \mid p-1$.
G	A subgroup of $\mathbb{Z}_p^*$. G is often a subgroup of order q, but sometimes is equal to $\mathbb{Z}_p^*$.
g	A generator of G.
$\mathcal{U}$	The set of principals intended to share the session key.
m	The size (cardinality) of $\mathcal{U}$.
U_i	The i'th principal in $\mathcal{U}$, where $1 \leq i \leq m$.
r_i	Random integer, typically of the same size as the order of G, chosen by U_i.
t_i	The value g^{r_i}. All computations take place in $\mathbb{Z}_p$.
x_i	The private long-term keys of U_i.
y_i	The public key of U_i, the value g^{x_i}. These public keys will have to be certified in some standard way which we do not consider here.
$\mathcal{Z}$	The shared secret calculated by the principals.
$\mathcal{K}$	The shared session key.
$H(.)$	A one-way hash function. Certain protocols may require specific properties and may specify particular functions.

For example, if $m = 3$ then $S^{(1)} = r_1 + r_2 + r_3$, $S^{(2)} = r_1r_2 + r_1r_3 + r_2r_3$ and $S^{(3)} = r_1r_2r_3$. Ingemarsson *et al.* showed that protocols exist in which the shared key is $g^{S^{(j)}}$ for any j with $1 \leq j \leq m$. However, they pointed out that all their protocols are insecure against a passive adversary who can tap the communications channels between the principals, except for the case $j = m$, which also turns out to be the case that minimises the computations required for each principal. (More precisely they considered how many of the communications channels need to be tapped in order for the eavesdropper to find the shared key in each case.)

Protocol 6.1 shows the message flows between principals U_{i-1}, U_i and U_{i+1} for the case $j = m$. In the first round, principal U_1 sends g^{r_1} to U_2, principal U_2 sends g^{r_2} to U_3, and so on. In the second round, principal U_1 sends $g^{r_1r_m}$ to U_2, U_2 sends $g^{r_1r_2}$ to U_3, and so on. At the end of $m-1$ rounds, principal U_i is able to calculate the shared secret $\mathcal{Z} = g^{r_1r_2\cdots r_m}$ by raising the final received message to its exponent r_i.

Of interest in all such protocols is the amount of communication and computation required for each principal, as well as the number of rounds required. In this case each principal must calculate m exponentiations, and send and receive $m-1$ messages. As explained above, $m-1$ rounds are required.

Petitioner Samsung Ex-1036, 0226



Protocol 6.1: Ingemarsson–Tang–Wong generalised Diffie–Hellman

6.2.2 Steiner–Tsudik–Waidner Key Agreement

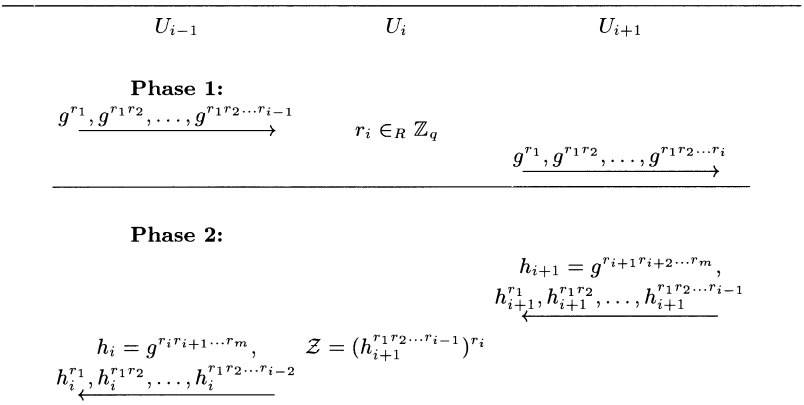
Steiner *et al.* [308] proposed three protocols, named GDH.1, GDH.2 and GDH.3, which can be regarded as variations of Protocol 6.1. Indeed, in each of their protocols the same key as that of Protocol 6.1 is derived by the principals from the same set of messages. The differences between all these protocols lie in where the computation is done and which messages are communicated. This leads to a flexible set of protocols that can be adapted to a variety of applications depending on the priorities in optimising communications or computational requirements.

The two phases of GDH.1 are shown in Protocol 6.2; messages travel in opposite directions along the line in the two phases. During the first phase values are collected up by the principals. Principal U_1 initially sends g^{r_1} to U_2. Then U_2 raises the received value to its secret r_2 and adds this to the received message to form the message sent to U_3. A similar pattern is followed by each other principal. At the end of the first phase principal U_m is able to calculate the shared secret $\mathcal{Z} = g^{r_1 r_2 \cdots r_m}$. In the second phase principal U_m starts sending messages in the opposite direction. The message sent by U_m to U_{m-1} is $g^{r_m}, g^{r_1 r_m}, g^{r_1 r_2 r_m}, \ldots, g^{r_1 r_2 \cdots r_{m-2} r_m}$. Principal U_{m-1} now starts the general pattern by using the last part of the message to form $\mathcal{Z}$ by raising it to its own random value r_{m-1}, removing it from the message, then raising the remaining $m-2$ parts of the message to its secret r_{m-1} before sending it to U_{m-2}.

In comparison with Protocol 6.1, GDH.1 reduces the average amount of computation required per principal. At the same time it takes twice the number of rounds; in GDH.1 no message can be sent until the previous message has been received, whereas in Protocol 6.1 many of the messages can be sent in parallel. In the next version of their protocol[1], GHD.2, Steiner *et al.* reduced

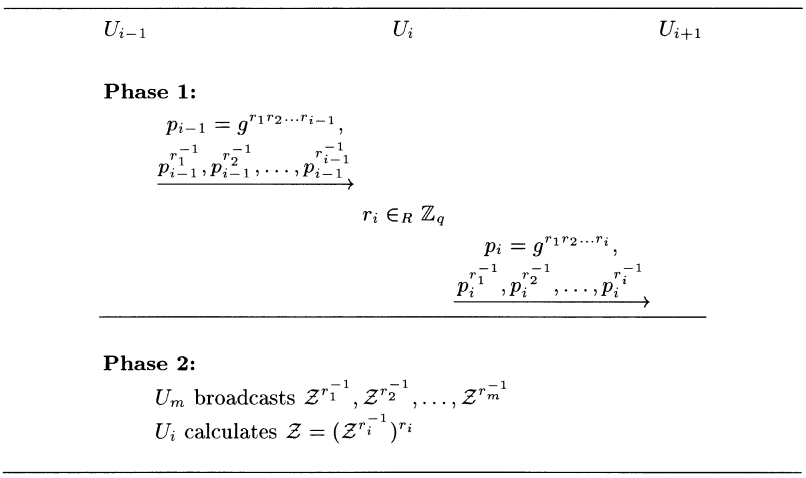
[1] This protocol is also known as IKA.1 in later papers of these authors [309].

Petitioner Samsung Ex-1036, 0227



Protocol 6.2: Steiner–Tsudik–Waidner GDH.1

the number of rounds required by gathering more partial calculations in the first phase and replacing the second phase with a single broadcast message from principal U_m.



Protocol 6.3: Steiner–Tsudik–Waidner GDH.2

Protocol 6.3 shows the messages in the GDH.2 protocol. In the first phase U_i receives i values from U_{i-1}; one of these values is the *principal value* p_{i-1}, while the remaining $i-1$ values consist of p_{i-1} with one of the exponents

Petitioner Samsung Ex-1036, 0228

$r_1, r_2, \ldots, r_{i-1}$ 'missing'. Initially U_1 starts Phase 1 by sending g^{r_1} and g to U_2. The notation used in Protocol 6.3 is not intended to indicate how these values are calculated: the inverted exponents are written only to conveniently summarise the values sent. On receiving this message U_i raises all received values to its exponent r_i to form i new message components and also includes the principal value p_{i-1} in the message sent on to U_{i+1}.

The second phase of GDH.2 consists of a single message broadcast by U_m, which includes all the partial calculations necessary for each other U_i to find $\mathcal{Z}$ with a single exponentiation using r_i. On receiving the final message in the first phase, U_m can calculate the shared secret from the principal value p_{m-1} as $\mathcal{Z} = p_{m-1}^{r_m}$. The final broadcast message can be calculated by U_m by raising each of the other $m-1$ components of its received message to its secret exponent r_m.

Steiner *et al.* proposed another variant protocol designed to minimise the average computation required for each principal. This protocol[2], GDH.3, has four phases.

Phase 1: Partial information is generated by the first $m-1$ principals.
Phase 2: Principal U_{m-1} broadcasts $g^{r_1 r_2 \ldots r_{m-1}} = \mathcal{Z}^{r_m^{-1}}$.
Phase 3: Each of the principals $U_1, U_2, \ldots, U_{m-1}$ 'removes' its exponent from the broadcast information and sends the result to principal U_m to add the final exponent to these partial values.
Phase 4: Principal U_m applies its exponent r_m to all the received partial calculations and broadcasts the results. This allows each principal to find $\mathcal{Z}$ by applying its exponent to the correct partial value.

Protocol 6.4 summarises the message flows for GDH.3. Again, the notation in Protocol 6.4 is not intended to imply that any inversion of exponents is actually calculated. In Sect. 6.2.9 below the relative features of each of the GDH variants is compared, together with other generalised Diffie–Hellman protocols.

Protocols GDH.2 and GDH.3 are well suited for dynamic groups. Steiner *et al.* provided explicit protocols for adding and deleting group members after the initial key has been agreed. The idea is to reuse most of the keying material from the initial protocol run, but one principal must renew its random value. The introduction of a fresh random exponent makes the new and old keys independent so that any added principal cannot find the initial key and a removed principal cannot find the new key. Steiner *et al.* also provide a protocol for many principals to join the group together.

In the original paper [308] the protocols for addition and deletion of group members required principal U_m to choose a new random input and update the previous values, but later Steiner *et al.* [309] pointed out that any principal can act as a *group controller* responsible for this. The protocols for adding and deleting principals are essentially identical for both GDH.2 and GDH.3.

[2] Also known as IKA.2 [309].

Petitioner Samsung Ex-1036, 0229

Phase 1: (for $i < m - 1$)

U_{i-1} $\xrightarrow{g^{r_1 r_2 \cdots r_{i-1}}}$ U_i $r_i \in_R \mathbb{Z}_q$ $\xrightarrow{g^{r_1 r_2 \cdots r_i}}$ U_{i+1}

Phase 2:

U_{m-1} broadcasts $g^{r_1 r_2 \cdots r_{m-1}}$

Phase 3:

U_i $\xrightarrow{(g^{r_1 r_2 \cdots r_{m-1}})^{r_i^{-1}}}$ U_m

Phase 4:

U_m broadcasts $\mathcal{Z}^{r_1^{-1}}, \mathcal{Z}^{r_2^{-1}}, \ldots, \mathcal{Z}^{r_{m-1}^{-1}}$

U_i calculates $\mathcal{Z} = (\mathcal{Z}^{r_i^{-1}})^{r_i}$

Protocol 6.4: Steiner–Tsudik–Waidner GDH.3

In order to add a member the group controller must refresh the message just before the final broadcast message using a new random exponent. Then the added principal sends a new broadcast message and all other principals calculate the new key the same way as in the final phase of the protocols. In the protocol for removal of a member the group controller broadcasts a new message for the final phase which excludes the component intended for the member to be removed.

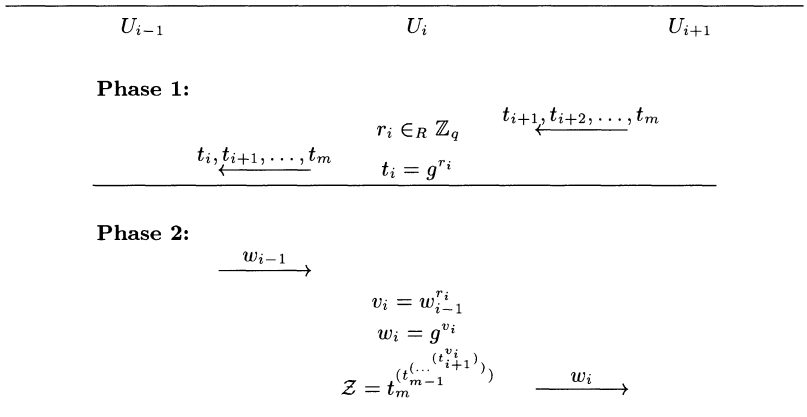
6.2.3 Steer–Strawczynski–Diffie–Wiener Key Agreement

The generalised Diffie–Hellman key agreement proposed by Steer *et al.* [305] was designed for use in a secure audio teleconference. In their description the messages between the principals are broadcast to all principals. However, not all messages are required by all principals and in our description principals are arranged in a linear fashion. Following the pattern of Protocol 6.2 there are two phases in which messages flow in opposite directions. The shared secret $\mathcal{Z}$ has the following value, but is calculated in a different way by each principal.

$$\mathcal{Z} = g^{r_m(g^{r_{m-1}(g^{\cdots(g^{r_1 r_2})})})}.$$

Protocol 6.5 shows the messages flowing to and from a typical principal U_i together with the computations made by U_i. Principal U_1 is able to calculate $\mathcal{Z}$ immediately once Phase 1 is complete as follows.

Petitioner Samsung Ex-1036, 0230



Protocol 6.5: Steer–Strawczynski–Diffie–Wiener generalised Diffie–Hellman

$$\mathcal{Z} = t_m^{(t_{m-1}^{(\ldots^{(t_2^{r_1})})})}.$$

To start Phase 2 principal U_1 sets $w_1 = t_1$ and sends this to U_2 who can now calculate $\mathcal{Z}$. Notice that the principals at the right hand end of the sequence use fewer computations than those at the left hand end. Principals U_1 and U_2 both use m exponentiations, but the number of exponentiations required decreases by one as i increases by one. Principal U_m requires only two exponentiations: one in Phase 1 and one in Phase 2 to find $\mathcal{Z} = v_{m-1}^{r_m}$. As compared with Protocol 6.1 the increase in the average number of computations is accompanied by a large decrease in the number of messages that are sent. In Protocol 6.5 each principal sends and receives only two messages (except for the end points which send and receive only one).

6.2.4 Perrig's Generalised Diffie–Hellman

Perrig [270] designed a key agreement protocol in which principals are considered as leaves in a binary tree structure. At stage j of the protocol a secret is agreed between each pair of sibling nodes of depth j. This secret is known to all principals at leaves that are children of those nodes. This continues until a secret is agreed between the two nodes that are the children of the root node, and this key is the group shared secret. In comparison with the previous protocols we have looked at, an advantage of Perrig's protocol is that the number of rounds required is only logarithmic in the number of principals.

Figure 6.1 illustrates the protocol for the case of four principals. We assume that there are $m = 2^d$ principals involved. The description below works equally well in general with $d = \lceil \log_2 m \rceil$ and letting $r_i = 1$ for $m < i \leq 2^d$. We denote by $Z_{j,k}$ the k'th key shared during round j.

Petitioner Samsung Ex-1036, 0231

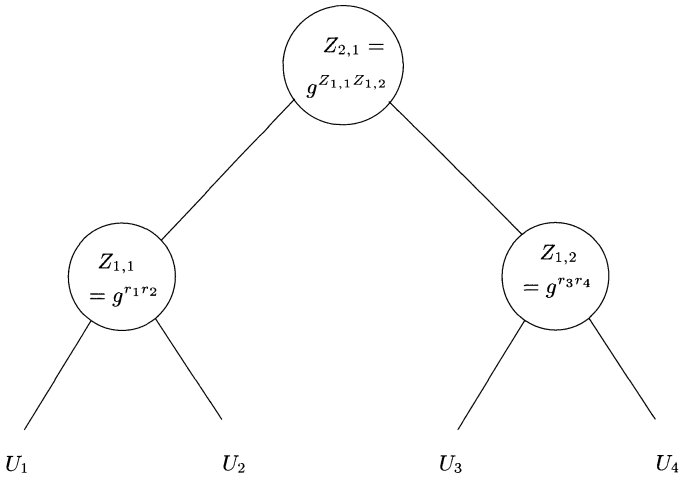


Fig. 6.1. Perrig's key agreement protocol with four principals

There are d rounds as shown in Protocol 6.6. During the first round each pair of sibling leaves shares a standard Diffie–Hellman secret. Conceptually we place each shared secret at the node of depth $d-1$ that is the parent node of the two leaves (principals) who share it. During round 2 one of the principals from the pair that share $Z_{1,k}$ calculates and broadcasts $g^{Z_{1,k}}$; both principals also receive $g^{Z_{1,k'}}$ where $Z_{1,k'}$ is the sibling node of $Z_{1,k}$ and thus both can calculate $Z_{2,k''}$ at the parent node of $Z_{1,k}$ and $Z_{1,k'}$. During round i the two sets of principals who share keys at sibling nodes of depth $d-i+1$ generate a Diffie–Hellman key by using these keys as the input to the Diffie–Hellman protocol.

At the end of round d the shared secret $\mathcal{Z} = Z_{d,1}$ is calculated by each of the principals as the root of the tree. Each principal can calculate the key for every node between its leaf and the root of the tree. In round i one principal must be nominated to broadcast g^K, for each key K at depth $d-i$, to ensure that all principals can calculate the key for the next round. We may regard this large number of broadcast messages as the price to be paid in Perrig's scheme for the reduced number of rounds.

Kim *et al.* [184] have considered this form of tree-based key agreement in some detail. In particular they have shown that this protocol is very suitable when it is desirable to alter the composition of the group without restarting the whole protocol. They show that protocols designed to add or remove either single or groups of principals can be performed efficiently by restructuring the tree of keys.

Petitioner Samsung Ex-1036, 0232

Round 1

$$Z_{1,1} = g^{r_1 r_2}$$
$$Z_{1,2} = g^{r_3 r_4}$$
$$\vdots$$
$$Z_{1,2^{d-1}} = g^{r_{m-1} r_m}$$

Round 2

$$Z_{2,1} = g^{Z_{1,1} Z_{1,2}}$$
$$Z_{2,2} = g^{Z_{1,3} Z_{1,4}}$$
$$\vdots$$
$$Z_{2,2^{d-2}} = g^{Z_{1,2^{d-1}-1} Z_{1,2^{d-1}}}$$

Round d

$$Z_{d,1} = g^{Z_{d-1,1} Z_{d-1,2}}$$

Protocol 6.6: Perrig's generalised Diffie–Hellman

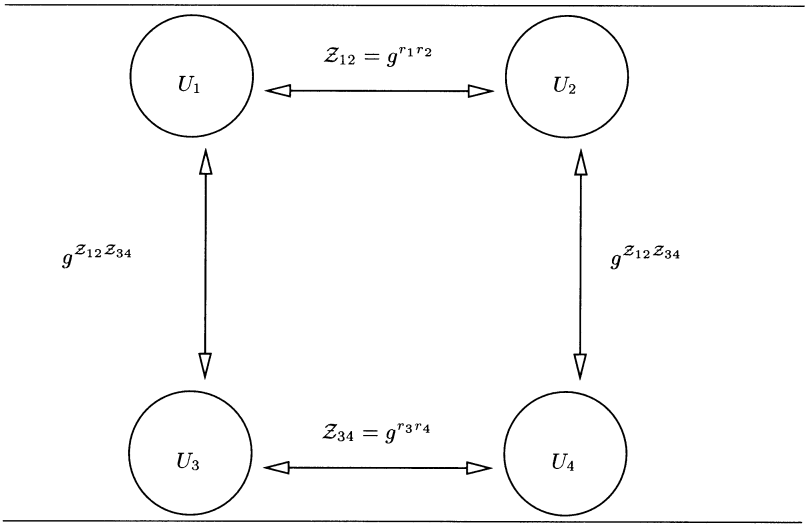
6.2.5 Becker and Wille's Octopus Protocol

Becker and Wille [25] proposed their 'Octopus' protocol in order to provide an example that minimises the number of simultaneous message exchanges between principals. An exchange between two principals can include a pair of messages, one in either direction. We will discuss this matter further in Sect. 6.2.9.

A building block for the Octopus protocol is the four-party key agreement shown as Protocol 6.7. The resulting shared secret is the same as in Perrig's four-party protocol in Fig. 6.1. However, the number of exchanges required is reduced (to four) due to the manner in which the key is derived. Firstly U_1 and U_2 exchange Diffie–Hellman messages to obtain the shared key $\mathcal{Z}_{12}$, while U_3 and U_4 similarly obtain $\mathcal{Z}_{34}$. Then U_1 and U_3 use these keys as inputs to a further Diffie–Hellman exchange so that both can obtain the shared secret $g^{\mathcal{Z}_{12}\mathcal{Z}_{34}}$, while U_2 and U_4 do the same. (We should strictly include a mapping to take elements $\mathcal{Z}_{12}$ and $\mathcal{Z}_{34}$, which are elements of G, to elements of $\mathbb{Z}_q$, which are required for the Diffie–Hellman inputs. Such a mapping is easily defined if $G = Z_p^*$, but in general there is no efficient mapping known. Becker and Wille include such a mapping, but we omit it from our descriptions.)

To set up the Octopus protocol the principals are partitioned into four subgroups of as near equal size as possible (this is a four-legged Octopus, but Becker and Wille also proposed a version with 2^d legs). Each group has a distinguished member, the *subgroup controller*, who manages the protocol for that subgroup. There are three phases in the Octopus protocol.

Petitioner Samsung Ex-1036, 0233



Protocol 6.7: Four-principal basic Octopus protocol

Phase 1: Each subgroup controller runs a Diffie–Hellman exchange individually with each of its group members. Let us denote the secret shared with the j'th member of the i'th subgroup as $Z_{i,j}$.

Phase 2: The four subgroup controllers run the basic protocol (6.7) with each controller using as input the product of all the keys agreed with members of its subgroup: $r_i = \prod_j Z_{i,j}$. Thus controller U_1 first agrees $\mathcal{Z}_{12} = g^{\prod Z_{1,j} \prod Z_{2,j}}$ with controller U_2. Then U_1 sends $g^{\mathcal{Z}_{12}}$ to controller U_3 and receives $g^{\mathcal{Z}_{34}}$ from U_3. This process allows each subgroup controller to calculate the group shared secret:

$$\mathcal{Z} = g^{(g^{\prod Z_{1,j} \prod Z_{2,j}})(g^{\prod Z_{3,j} \prod Z_{4,j}})}. \tag{6.1}$$

Phase 3: Each subgroup controller sends each member of its subgroup two values, which consist of its two inputs to the last step in Phase 2 except that the secret it shared with that principal during Phase 1 is missing from the exponent. For example, suppose subgroup controller U_1 shared secret $Z_{1,k}$ with one of its subgroup members in Phase 1. Then U_1 sends to that principal the two values $g^{\prod_{j \neq k} Z_{1,j} \prod Z_{2,j}}$ and $g^{(g^{\prod Z_{3,j} \prod Z_{4,j}})}$. From these values the principal can first reconstruct $g^{\prod Z_{1,j} \prod Z_{2,j}}$ using $Z_{1,k}$ and then the group shared secret $\mathcal{Z}$ shown in Eqn. 6.1.

The number of message exchanges required in the Octopus protocol is $m - 4$ during each of Phases 1 and 3, and four during Phase 2, making a total of $2m - 4$. The exchanges in Phases 1 and 2 include two messages each, while

Petitioner Samsung Ex-1036, 0234

those in Phase 3 consist of only one message. Thus in total there are $3m - 4$ messages sent.

6.2.6 Burmester–Desmedt Key Agreement

The protocol of Burmester and Desmedt [75] uses an ingenious method to obtain efficiency both in the number of messages sent per user and in the amount of computation required. Instead of using a symmetric function to define the shared key, as in Protocol 6.1, Burmester–Desmedt key agreement uses a *cyclic* function. Protocol principals are arranged in a ring so that $U_1 = U_{m+1}$. The protocol is simplest to understand in the version that allows broadcast communications; below we also describe how it can be implemented using communication only between adjacent users in the ring. There are two phases in this broadcast version as shown in Protocol 6.8.

U_{i-1}	U_i	U_{i+1}
Phase 1:		
	$r_i \in_R \mathbb{Z}_q$	
$\xleftarrow{t_i}$	$t_i = g^{r_i}$	$\xrightarrow{t_i}$
	$X_i = (t_{i+1}/t_{i-1})^{r_i}$	
	$Z_{i-1,i} = t_{i-1}^{r_i}$	
Phase 2:		
	Principal U_i broadcasts X_i to all other parties.	
	U_i calculates $\mathcal{Z} = (Z_{i-1,i})^m X_i^{m-1} X_{i+1}^{m-2} \ldots X_{i-2}$.	

Protocol 6.8: Burmester–Desmedt generalised Diffie–Hellman with broadcasts

During Phase 1 each adjacent pair of users, U_i and U_{i+1}, performs a basic Diffie–Hellman key exchange. However, instead of calculating the individual secrets, principal U_i calculates the ratio, X_i, of its two secrets with adjacent principals. In Phase 2 each principal broadcasts its X_i value. Once all principals have broadcast their values, every principal can calculate the shared secret. The following calculation shows that all principals calculate the same shared secret if all principals act correctly.

$$\begin{aligned}
\mathcal{Z} &= (Z_{i-1,i})^m \cdot X_i^{m-1} \cdot X_{i+1}^{m-2} \cdot \ldots \cdot X_{i-2} \\
&= (Z_{i-1,i})^m \left(\frac{Z_{i,i+1}}{Z_{i-1,i}}\right)^{m-1} \left(\frac{Z_{i+1,i+2}}{Z_{i,i+1}}\right)^{m-2} \cdots \left(\frac{Z_{i-2,i-1}}{Z_{i-3,i-2}}\right) \\
&= Z_{i-1,i} \cdot Z_{i,i+1} \cdot Z_{i+1,i+2} \cdot \ldots \cdot Z_{i-2,i-1} \\
&= g^{r_1 r_2 + r_2 r_3 + \ldots + r_m r_1}.
\end{aligned}$$

Petitioner Samsung Ex-1036, 0235

Just and Vaudenay [175] have pointed out that the Burmester–Desmedt protocol can be generalised in two ways. Firstly, the keys shared between the adjacent principals, the $Z_{i,i+1}$ values, can be found using any key agreement protocol. Secondly, there are alternative ways to combine these shared keys. One example is to choose $X_i = Z_{i,i+1} - Z_{i-1,i}$ and then define the shared secret as follows.

$$\begin{aligned}\mathcal{Z} &= mZ_{i-1,i} + (m-1)X_i + (m-2)X_{i+1} + \ldots + X_{i-2} \\ &= Z_{1,2} + Z_{2,3} + \ldots + Z_{m,1}.\end{aligned}$$

Horng [147] designed a protocol closely related to Protocol 6.8 but with the principals arranged in a logical line rather than in a ring. The shared secret is also subtly different from that of Protocol 6.8, including only the adjacent exponent pairs on the line: $\mathcal{Z} = g^{r_1r_2+r_2r_3+\ldots+r_{m-1}r_m}$. The method for calculating the shared secret depends on where the principal stands in the line, but is more efficient than Protocol 6.8, especially if the number of principals is large. Horng shows that any passive attack on the protocol that can distinguish $\mathcal{Z}$ from a random string is as hard as the decision Diffie–Hellman problem.

Since broadcast messages can be expensive in some communications networks, Burmester and Desmedt also proposed a protocol version that uses only communication between adjacent principals. The basic idea of the protocol is unchanged, but the calculations are distributed between different principals, reducing the computation required for each principal. There are three phases in this pairwise version as shown in Protocol 6.9.

Phase 1 in Protocol 6.9 is the same as in the broadcast version of the protocol described above. After this phase each principal knows the shared secrets with each adjacent principal. The purpose of Phase 2 is to distribute the computation of the shared secret amongst all principals. This phase requires m rounds starting from U_1 with $b_0 = c_0 = 1$. Alternatively, any one principal could gather all the X_i values and do all the computation for this phase on behalf of the other principals.

When Phase 2 is complete, U_1 has received the value $c_m = X_1^{m-1} \cdot X_2^{m-2} \cdot \ldots \cdot X_{m-1}$ from U_m and sets this value to d_0. In the final phase each principal can calculate the shared secret $\mathcal{Z}$ and also calculates the d_i value to be sent on to the next principal. For example, U_1 calculates $\mathcal{Z} = d_0 \cdot Z_{0,1}^m$ and sends $d_1 = d_0/X_1^m$ on to U_2. It follows from the equality $X_1 = X_2 \cdot X_3 \cdot \ldots \cdot X_m$ that $d_1 = X_2^{m-1} \cdot X_3^{m-2} \cdot \ldots \cdot X_m$ so that U_2 can calculate $\mathcal{Z}$ in a similar way. Notice that all the messages passed between the principals can be calculated from the public X_i values, which are available to an eavesdropper in the broadcast protocol version.

6.2.7 Joux's Tripartite Diffie–Hellman

All the Diffie–Hellman generalisations described so far require at least two rounds to complete. Later we will see that there are conference key agreement

Petitioner Samsung Ex-1036, 0236

U_{i-1}	U_i	U_{i+1}
Phase 1:		
	$r_i \in_R \mathbb{Z}_q$	
$\xleftarrow{t_i}$	$t_i = g^{r_i}$	$\xrightarrow{t_i}$
	$X_i = (t_{i+1}/t_{i-1})^{r_i}$	
	$Z_{i-1,i} = t_{i-1}^{r_i}$	
Phase 2:		
$\xrightarrow{b_{i-1}, c_{i-1}}$	$b_i = X_i b_{i-1}$	
	$c_i = b_{i-1} c_{i-1}$	$\xrightarrow{b_i, c_i}$
Phase 3:		
$\xrightarrow{d_{i-1}}$	$d_i = d_{i-1}/X_i^m$	
	$\mathcal{Z} = d_{i-1} \cdot Z_{i-1,i}^m$	$\xrightarrow{d_i}$

Protocol 6.9: Burmester–Desmedt pairwise generalised Diffie–Hellman

protocols that can be run in one round, but they do not achieve forward secrecy. The protocol of Joux [171] is the only example currently known of a group key agreement protocol that can be run in a single round and still provide forward secrecy; however, the protocol can only work with three parties.

Joux's protocol works in elliptic curve groups and exploits properties known as *pairings* of group points. These pairings are bi-linear mappings, which means that it is possible for principal A to calculate $g^{x_A x_B x_C}$ knowing just x_A and the ephemeral Diffie–Hellman keys of B and C on the elliptic curve; here g is a point in a different group defined by the elliptic curve base point and the pairing.

The security of Joux's protocol is based on a different problem from the standard Diffie–Hellman problem on elliptic curves. Al-Riyami and Paterson [9] discuss several authenticated versions of Joux's protocol.

6.2.8 Security of Generalised Diffie–Hellman

Because we have considered only protocols without authentication in this section, none of these protocols can provide security against an active adversary. However, for many of the protocols in this section it is possible to relate their security to that of the two-party Diffie–Hellman key exchange. Since the Diffie–Hellman problem has resisted all challenges for over 25 years this is as good a security property as we can hope for.

Petitioner Samsung Ex-1036, 0237

Definition 6.4. *Suppose that n principals each choose respective random inputs $r_1, r_2, \ldots, r_n$ and derive the shared secret $\mathcal{Z} = g^{r_1 r_2 \cdots r_n}$. The n-party Diffie–Hellman problem is to find $\mathcal{Z}$ given all elements of the set $\{g^{\Pi(S)} | S \subset \{r_1, r_2, \ldots, r_n\}\}$.*

Steiner *et al.* [308] give a formal proof that the n-party Decision Diffie–Hellman (DDH) problem is hard if the original two-party version is hard. This gives assurance that if the adversary is able to distinguish between a random value and the shared secret, given the public values, then the adversary can also solve the two-party DDH. Although it is not known that the DDH is as hard as the basic Diffie–Hellman problem, it is widely believed to be hard, and many security proofs assume this to be the case. For example, the problem of finding the cyclic key used in Burmester and Desmedt's Protocol 6.9 is claimed by its authors [75] to be as hard as breaking the two-party Diffie–Hellman protocol, although no proof is given.

Proofs of security seem problematic when the shared secret is defined through 'recursive' use of Diffie–Hellman, because the output does not lie in the same algebraic group as the input. Thus no security proof is available for the Steer–Strawczynski–Diffie–Wiener key agreement (6.5) or for Perrig's protocol (6.6). However, Becker and Wille, in their description of the Octopus protocol (6.7), assume the existence of an efficient bijection from the group G generated by g to $\mathbb{Z}_q$; this mapping is invoked when moving the outputs of a Diffie–Hellman protocol in G to the inputs in $\mathbb{Z}_q$. Under this assumption they prove that if ordinary Diffie–Hellman is secure then so is the Octopus protocol.

6.2.9 Efficiency of Generalised Diffie–Hellman

When comparing the relative efficiencies of the protocols in this section there are a number of different measures that may be relevant depending on the implementation and application scenario. Table 6.2 summarises the most important performance features of each of the protocols. The purpose of this table is only to be indicative of the comparative costs in each protocol and therefore we have taken licence in some of the figures where the exact values differ only slightly from those indicated. For example, in GDH.1 the end principals are required to send only one message, while in GDH.2, U_m sends only the broadcast message. Similarly the number of exponentiations for U_m in GDH.1 and GDH.2 is m.

In Table 6.2 the number of messages refers to the number of point-to-point messages sent, while the total number of broadcast messages is in addition to the point-to-point messages. Any message sent to more than one principal is counted as a broadcast. A number of different papers also include tables of comparison [308, 270, 12] but the reader should be aware that these do not all count items in the same way. Amir *et al.* [12] have reported on extensive practical tests of the performance of several of these protocols.

Petitioner Samsung Ex-1036, 0238

Table 6.2. Computational requirements in generalised Diffie–Hellman protocols

	Exponentiations per U_i	Messages per U_i	Messages in total	Broadcasts	Rounds
ITW (6.1)	m	$m-1$	$m(m-1)$	0	$m-1$
GDH.1 (6.2)	$i+1$	2	$2(m-1)$	0	$2(m-1)$
GDH.2 (6.3)	$i+1$	1	$m-1$	1	m
GDH.3 (6.4)	$3^{\dagger}$ (m for U_m)	2	$2m-3$	2	$m+1$
SSDW (6.5)	$m-i+2$	2	$2(m-1)$	0	$2(m-1)$
Perrig (6.6)	$\lceil \log_2 m \rceil + 1^{\ddagger}$	1	m	$m-2$	$\lceil \log_2 m \rceil$
Octopus (6.7)	$4^{\ddagger}$	3	$3m-4$	0	4
BDB (6.8)	$3^{\star}$	2	$2m$	m	2
BD (6.9)	$3^{\star}$	4	$4m-1$	0	$2m$

† Calculation of an inverse also required.
‡ Minimum value. Some principals must do more.
* Average of two exponentiations with exponent m also required.

Choosing a suitable protocol for an application may depend on many factors and it may not be sufficient to optimise any one particular performance parameter. There is no single generalised Diffie–Hellman protocol that is the best for all the criteria used in Table 6.2. If the lowest computation per principal is desired then the Burmester–Desmedt (BD) or GDH.3 protocols are attractive. (Note, however, that one principal has a high computational load in GDH.3.) If the number of rounds is to be minimised then the Burmester–Desmedt broadcast (BDB) protocol looks attractive but the communications network used must support simultaneous broadcasts if the figure of only two rounds is to be achieved. If broadcast communications are to be avoided then GDH.1 or BD are both worth considering, although ITW may be the best if the number of rounds should be minimised at the same time.

Becker and Wille [25] have provided lower bounds on several of the desirable performance parameters. These bounds are derived from an analysis using basic counting and inductive proofs. Their results are summarised in Table 6.3, which is set out to emphasise the clear division between protocols with and without broadcast, as made by Becker and Wille. As well as the number of messages used and the number of (synchronous) rounds, Becker and Wille also consider the number of *exchanges* (connections between pairs of users) and the number of *simple rounds* (in which every party sends and receives at most one message). Notice that synchronous rounds are only relevant when broadcast messages are used.

Comparison between Tables 6.2 and 6.3 shows that in many cases the bounds derived by Becker and Wille can be met by practical protocols. Specif-

Petitioner Samsung Ex-1036, 0239

Table 6.3. Performance bounds for conference key agreement (Becker and Wille)

	Messages	Exchanges	Simple rounds	Synchronous rounds
Without broadcasts	$2(m-1)$	$2(m-2)$	$\lceil \log_2 m \rceil$	-
With broadcasts	m	m	$\lceil \log_2 m \rceil$	1

ically GDH.1 and GDH.2 meet the bounds on the minimum number of messages for the two cases without and with broadcast messages. In the case of GDH.2, adding the single broadcast message to the $m-1$ point-to-point messages gives the bound of m messages indicated in Table 6.3. When it comes to (simple) rounds, we have seen that Perrig's protocol meets this bound. The Octopus protocol was specifically designed to be optimal in terms of number of exchanges and meets the lower bound of $2m-4$. Although Becker and Wille left it as an open question whether a protocol with only one synchronous round is possible in the case without broadcasts, we will show in Sect. 6.5 examples of protocols that meet this bound.

In the next section we will examine how the protocols introduced in this section can be enhanced by adding authentication to the messages sent. We will also see that there are other features that may prove important in comparing the suitability of protocols for different applications.

6.3 Conference Key Agreement Protocols

When examining two-party key agreement based on Diffie–Hellman in Chap. 5, two classes of protocol were evident: those in which the long-term public keys were included in the calculation of the shared secret and those in which long-term keys were used only for message authentication. For conference key agreement the former class of protocols is completely missing. The reason for this is that there seems to be no symmetrical way that long-term keying information from multiple users can be incorporated into the shared key. Therefore all the published authenticated conference key agreement protocols make use of either public key encryption or digital signatures to provide authentication of the messages sent in the various generalised Diffie–Hellman key agreement protocols.

6.3.1 Authenticating Generalised Diffie–Hellman

Designers of many of the multi-party Diffie–Hellman generalisations that were examined in Sect. 6.2 have ignored the issue of key authentication, or have simply stated that authentication using digital signatures may be added. When

Petitioner Samsung Ex-1036, 0240

there are no long-term keys providing authentication, forward secrecy is meaningless. If key authentication is added by, for example, signing protocol messages with a long-term key, then forward secrecy will usually be provided automatically as long as the signature mechanism and ephemeral keys are independent.

Burmester and Desmedt proposed that their generalised Diffie–Hellman key exchange (Protocol 6.9) could provide key authentication if each t_i value was authenticated by any chosen means. They suggested that this means could be any good digital signature scheme but also provide an explicit interactive mechanism. However, Just and Vaudenay [175] pointed out that authenticating the messages is not sufficient, since it does not show that the party authenticating knows the random input r_i and consequently unknown key-share attacks are possible. In any case, this simple authentication can only provide a weak form of key authentication in that each party authenticates the participation of only one other party. There is no direct assurance of which other parties may have the shared secret.

Just and Vaudenay [175] also propose a generalised form of the Burmester–Desmedt protocol, using their two-party key agreement protocol examined in Chap. 5 (Protocol 5.13) as the basic building block. They prove the security of their protocol against a passive adversary. Although it avoids the unknown key-share attack against the Burmester–Desmedt protocol it still only provides weak key authentication. Just and Vaudenay suggest that such problems may be avoided by making each party broadcast a signed hash of all messages sent in the protocol. This seems to be an expensive way to provide implicit key authentication.

6.3.2 Klein–Otten–Beth Protocol

Klein *et al.* [185] proposed a protocol that is related to the protocol of Ingemarsson *et al.* (Protocol 6.1) but with two distinct enhancements designed to provide authentication and robustness.

- Each message is protected by a digital signature of the sender, which also includes a unique identifier for the protocol run.
- Each intermediate value is calculated multiple times with the aim of detecting and recovering from errors caused by principals deviating from the protocols.

More precisely there are $m-1$ rounds to the protocol during which messages are broadcast, via a write-only bulletin board, to all other principals. Messages all consist of triples of the form $(U_i, \mathcal{A}, Sig_{U_i}(\texttt{PID}, g^A))$, where PID is an identifier, $\mathcal{A}$ is a set of users in $\mathcal{U}$, and A is the set of exponents input by the users in $\mathcal{A}$. For example, a particular message sent by U_2 might be $(U_2, \{U_3, U_2\}, Sig_{U_2}(\texttt{PID}, g^{r_3 r_2}))$.

Round 1: U_i chooses random ephemeral input x_i and broadcasts the triple $(U_i, \{U_i\}, Sig_{U_i}(\texttt{PID}, g^{x_i}))$.

Petitioner Samsung Ex-1036, 0241

Round j $(2 \leq j \leq m-1)$**:** The following steps are taken.

1. U_i collects all messages from round $j-1$ that exclude x_i in the exponents. For each of these U_i raises the g^A value to the exponent x_i, and forms the new message triple (adding its identity to the set $\mathcal{A}$) and broadcasts the result.
2. Each message with the same user set $\mathcal{A}$ is then compared. If there are any differences then the recovery process is invoked.
3. Once these are resolved, all duplicates are deleted and j is incremented.

The recovery protocol has as input a set of principals $\mathcal{A}$ and a set of message triples using set $\mathcal{A}$ but with differing values of g^A. All principals belonging to $\mathcal{A}$ are required to reveal their secret input x_i which allows checking against the signed inputs from the previous round. Principals found to have been cheated by a majority of the other principals are expelled from the conference. Those principals who have not cheated choose new x_i values and reconstruct their inputs for the current round.

Despite the attraction of having a protocol robust to disruption, this is an expensive protocol. Even without any recovery, each principal is required to perform a total of $2 + \binom{m-1}{1} + \binom{m-1}{2} + \ldots + \binom{m-1}{m-2} = 2^{m-1}$ exponentiations, which becomes impractical for large values of m.

6.3.3 Authenticated GDH Protocols

Ateniese *et al.* [18, 19] propose two methods to extend their GDH.2 protocol (6.3) to provide authenticated group key agreement. They remark that the same extensions can be applied to GDH.3 too, but imply that GDH.1 is less interesting in practice, since the alternatives have smaller computation and communications requirements (see Table 6.2).

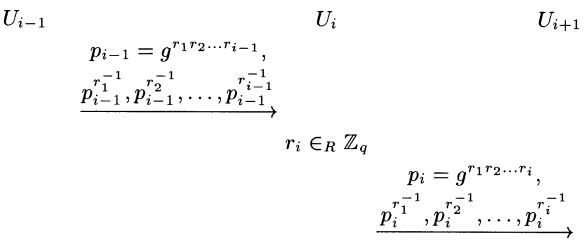
The simplest method of providing authentication changes only the final broadcast message of Protocol 6.3. In order to achieve this extension it is assumed that the distinguished principal U_m shares a secret K_i with each U_i. It is suggested by Ateniese *et al.* that this secret should be calculated from the static Diffie–Hellman key, $S_{i,m} = g^{x_i x_m}$, shared between U_i and U_m; for example, $K_i = F(S_{i,m})$ where F is a function taking elements of G to $\mathbb{Z}_q$. Then the first phase of the protocol is the same as Protocol 6.3. In the second phase U_m sends $\mathcal{Z}^{r_i^{-1} K_i}$ to U_i. On receipt of this message U_i can find the shared secret $\mathcal{Z} = g^{r_1 r_2 \ldots r_m}$ by raising the received message to the power $r_i K_i^{-1}$. The message flows are shown in Protocol 6.10 which Ateniese *et al.* called A-GDH.2.

The use of the shared secret K_i values means that principals can be sure that the value they calculate for $\mathcal{Z}$ will be known only to those that actually participate in the protocol with U_m. This is on the assumption that U_m follows the protocol faithfully. In this sense the protocol provides implicit key

Petitioner Samsung Ex-1036, 0242

Information shared by U_i and U_m: Key K_i.

Phase 1:



Phase 2:

U_m broadcasts $\mathcal{Z}^{r_1^{-1} K_1}, \mathcal{Z}^{r_2^{-1} K_2}, \ldots, \mathcal{Z}^{r_m^{-1} K_m}$

U_i calculates $\mathcal{Z} = (\mathcal{Z}^{r_i^{-1} K_i})^{r_i K_i^{-1}}$

Protocol 6.10: Ateniese–Steiner–Tsudik A-GDH.2

authentication. However, in common with many multi-party protocols, principals have no direct assurance of which other principals are participating in the protocol. This means that the principals really know which other parties have the shared secret only if the group is fixed or assured by some other means. Ateniese *et al.* claim a proof of security against passive attacks but not against an active adversary. Indeed they point out that an attack similar to Burmester's triangle attack is possible, although they claim that the unusual circumstances involved make such attacks 'very unlikely in practice'.

Pereira and Quisquater [267] conducted an analysis of the security of Protocol 6.10 using a systematic technique for deciding which powers of g can be obtained by an active adversary. They showed that strong key authentication can indeed fail if the group membership varies. They demonstrated an explicit attack in which the adversary takes part in one protocol run and can find the key in a subsequent run in which the adversary is not intended to be involved. To see how this attack can work consider the particular case of A-GDH.2 when $m = 4$ shown in Protocol 6.11. The attack proceeds as follows.

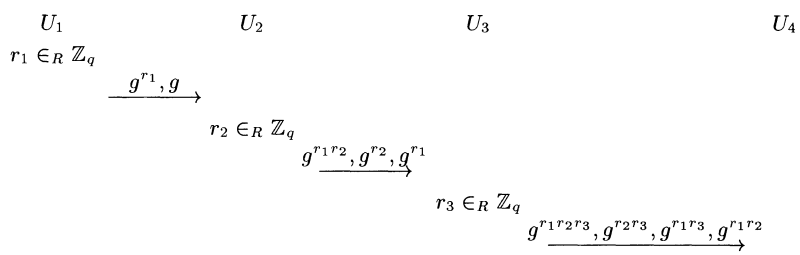
1. The adversary takes part in Protocol 6.11 as U_3 but alters the message sent from U_3 to U_4 by replacing $g^{r_1 r_3}$ with $g^{r_1 r_2}$. As a consequence, U_4 will include $g^{r_1 r_2 r_4 K_2}$ in the broadcast part intended for U_2, instead of the correct value $g^{r_1 r_3 r_4 K_2}$. The adversary also obtains $g^{r_1 r_2 r_4}$ from the last broadcast part, since it knows K_3. The adversary records the values exchanged in this run.
2. The adversary observes a new protocol run involving only the other three principals from the first run. In the new run suppose that new values r_1', r_2'

Petitioner Samsung Ex-1036, 0243

and r_3' are chosen by U_1, U_2 and U_3. The adversary replaces the message from U_1 by $g^{r_1 r_2 r_4}$ obtained from the first run. Then U_2 will send $g^{r_1 r_2 r_4 r_2'}$ as part of its message to U_4. Finally the adversary can replace the part of the broadcast message to U_2 by $g^{r_1 r_2 r_4 K_2}$ recorded from the first run. This means that U_2 will calculate $\mathcal{Z} = g^{r_1 r_2 r_4 r_2'}$ which was sent by U_2 in the second message and so is known to the adversary.

Information shared by U_j and U_4: Key K_j.

Phase 1:



Phase 2:

U_4 chooses $r_4 \in_R \mathbb{Z}_q$ and broadcasts $g^{r_2 r_3 r_4 K_1}, g^{r_1 r_3 r_4 K_2}, g^{r_1 r_2 r_4 K_3}$

U_i calculates $\mathcal{Z} = g^{r_1 r_2 r_3 r_4}$

Protocol 6.11: Protocol 6.10 when $m = 4$

Pereira and Quisquater [267] also found a more convincing attack than that of Ateniese *et al.* in which old keying material is replayed by an active adversary. This attack could be prevented in the case that a one-way key derivation function is used and only the derived session key becomes compromised.

Because the long-term keying material (the K_i values) is used only for the authentication, it follows that forward secrecy from a passive adversary is provided in Protocol 6.10, as long as the multi-party Diffie–Hellman assumption holds. However, Pereira and Quisquater [267] also demonstrated that an active adversary can alter the protocol in such a way that all but one of the principals computes the same key but forward secrecy will subsequently fail.

If principal U_i's long-term key becomes compromised then K_i is also compromised. In this situation the adversary can forge the messages intended for U_i in both phases and hence can find the key that U_i calculates as $\mathcal{Z}$. Therefore resistance to key compromise impersonation is not provided. Key confirmation is not provided either since U_i obtains no assurance that any other party has obtained the same key. However, Ateniese *et al.* suggest that U_m may

Petitioner Samsung Ex-1036, 0244

include $g^{H(\mathcal{Z})}$ in the broadcast message. This gives each U_i key confirmation with respect to U_m only.

In comparison with the unauthenticated GDH.2 protocol, Protocol 6.10 adds only a small overhead. The computations required for U_m and for each U_i remain essentially unchanged except for the K_i values. If the protocol is run frequently the K_i values may already be available.

In order to reduce the reliance on principal U_m, Ateniese *et al.* designed an alternative method to provide authentication in GDH.2, resulting in a protocol that they called SA-GDH.2. The general idea is to allow each principal to mutually authenticate each other principal through use of a long-term shared secret. Each pair of principals U_i and U_j shares two keys $K_{i,j}$ and $K_{j,i}$. Ateniese *et al.* do not define how these are calculated, but an obvious way is to use two different derivations of the long-term static Diffie–Hellman key $S_{i,j}$. Protocol 6.12 shows the message flows.

Information shared by U_i and U_j: Keys $K_{i,j}$ and $K_{j,i}$.

Phase 1:

U_{i-1} U_i U_{i+1}

$V_{i-1,1}, V_{i-1,2}, \ldots, V_{i-1,m}$ $\longrightarrow$

$r_i \in_R \mathbb{Z}_q$

$V_{i-1,1}^{r_i K_{i,1}}, \underrightarrow{V_{i-1,2}^{r_i K_{i,2}}, \ldots, V_{i-1,i}, \ldots}, V_{i-1,m}^{r_i K_{i,m}}$

Phase 2:

U_m broadcasts $V_{m,i} = \mathcal{Z}^{r_i^{-1} K_{1,i} \cdots K_{i-1,i} K_{i+1,i} \cdots K_{m,i}}$ for each i

U_i calculates $\mathcal{Z} = V_{m,i}^{r_i K_{1,i}^{-1} \cdots K_{i-1,i}^{-1} K_{i+1,i}^{-1} \cdots K_{m,i}^{-1}}$

Protocol 6.12: Ateniese–Steiner–Tsudik SA-GDH.2

As in Protocols 6.3 and 6.10 there are two phases. During the first phase each principal sends and receives a message with m ordered fields (for U_1 the fields $V_{0,1}, V_{0,2}, \ldots, V_{0,m}$ should all be taken to equal g). U_i raises the j'th received field to the power $r_i K_{j,i}$ with the exception that the i'th field is left unchanged. The updated fields are then sent on to the next principal. On completion of the first phase, U_m is able to calculate the shared secret $\mathcal{Z} = g^{r_1 r_2 \cdots r_m}$ from the last field of its received message. This last field will be $g^{r_1 r_2 \cdots r_{m-1} K_{1,m} K_{2,m} \cdots K_{m-1,m}}$ and so U_m needs to remove the $K_{j,m}$ exponents for each $0 \leq j \leq m-1$ and add the r_m exponent.

Petitioner Samsung Ex-1036, 0245

In the second phase U_m, after performing the same transformations as done by each U_i in Phase 1, broadcasts the results. This allows each other principal to find $\mathcal{Z}$ in the symmetrical way by removing the relevant $K_{i,j}$ exponents and adding the exponent r_i.

The advantage of Protocol 6.12 over Protocol 6.10 is that a stronger form of implicit key authentication is achieved. Each principal U_i knows that only principals that possess one of the shared keys $K_{i,j}$ are able to calculate the same value of $\mathcal{Z}$ calculated by U_i. There is still no confirmation that any other principal does actually possess that key and again U_i must know the other group members' identities by some external means. Furthermore, the computational cost of Protocol 6.12 is greatly increased for each principal, with the exception of U_m: now each principal needs to perform m exponentiations. Ateniese *et al.* [19] provide a more detailed analysis of the communication and computational cost of the A-GDH.2 and SA-GDH.2 protocols.

Bresson *et al.* [70], and later Bresson *et al.* [68, 69], have proposed a different authenticated version of GDH.2 and provided proofs in the Bellare–Rogaway model. Their protocol is the same as Protocol 6.3 with two additions.

- The group identity (the names of all principals involved in the protocol) is added to each message flow.
- Each message flow is signed with a long-term key of the sender.

The proof in their first paper [70] reduces the security of the protocol to that of the m-party Diffie–Hellman problem, when the set of exponents is restricted to those exchanged in the protocol. The reduction is not very tight, however. Their next paper [68] extends their results in two way. Firstly, additional protocols to allow principals to join and leave an existing conference are added and proven secure. Secondly, the security reduction is made tighter, at least for a fixed size group. These first two papers assumed random oracles in the proofs; the latest paper in the series [69] was able to remove this assumption, but replaces the computational m-party Diffie–Hellman assumption with a stronger n-party decision problem.

6.4 Identity-Based Conference Key Protocols

In Chap. 5 we examined identity-based key agreement for two parties. This idea can be generalised to multi-party protocols. Of course identity-based conference key protocols do not have to use key agreement, but published protocols seem mainly to have considered this option. As is usual with identity-based protocols, the convenience of using only identity information in place of public keys comes at the price of requiring a trusted authority to generate secret keys on behalf of principals. In the following, ID_i denotes the identifying string of entity U_i.

Petitioner Samsung Ex-1036, 0246

6.4.1 Koyama and Ohta Protocols

Koyama and Ohta [191] designed three protocols using different physical architectures. A succession of attacks and consequent countermeasures have been published, illustrating how difficult it is to consider all possibilities without a formal security statement. Indeed, it may be suggested that problems with these early protocols arise because at the time there was no clear notion of what are the desired security properties of such protocols.

Koyama and Ohta's first protocol type uses Diffie–Hellman in $\mathbb{Z}_n^*$ where $n = pq$ is an RSA modulus whose factorisation is known only to a trusted authority T. The authority chooses public and private keys e and d, with $ed \equiv 1 \bmod \phi(n)$. The principals are arranged in a ring where, as usual, we take U_{m+1} to mean U_1. The protocol can be viewed as a variant of Protocol 6.1 where authentication information is added to each message.

Each principal is issued with a secret key by the authority. Before issuing any secrets, the authority needs to choose a value M which will be the maximum number of users who can participate in a conference (with the current parameter values). In our notation m is the size of the conference so this means that $m \leq M$. Because the computational requirements for each principal increase with M it should not be chosen larger than necessary. Once M is fixed the authority can calculate the secret value S_i, using the following equation, and transfer it securely to principal U_i.

$$S_i = ID_i^{d^{M-1} \bmod \phi(n)}.$$

A successful protocol run is shown as Protocol 6.13. There are m rounds in a run. During each round, except for the last, each principal sends three values to the next principal in the ring. On receipt of a triple in round j the principal performs a check designed to verify that the triple has been processed by the previous j principals in the ring. If the check is successful the principal constructs and sends on its own triple. The first value in a triple, X_j, is a partial version of the final shared secret and the principal adds its random exponent to the received X_j value. The final shared secret $\mathcal{Z}$ is the following value.

$$\mathcal{Z} = g^{e^{m-1} r_1 r_2 \ldots r_m}.$$

Although not explicitly stated by Koyama and Ohta, it seems reasonable to say that Protocol 6.13 is designed to provide a strong form of key authentication. It is apparent that some group entity authentication is also intended. Indeed, Koyama and Ohta [191] state that if the check in round m succeeds then U_i can verify that the received messages came via U_{i-1}, U_{i-2}, ..., and U_{i-m+1} successively. However, there is nothing that any principal can use to verify that any received message is not replayed from an old protocol run. Therefore it is clear that no other principal need be present in any specific run.

Petitioner Samsung Ex-1036, 0247

Shared information: Identity information ID_i of principal U_i, public RSA key (n, e), element g of maximal order in $\mathbb{Z}_n^*$, prime c.
Information known only to U_i: Secret key S_i with $S_i^{e^{M-1}} \bmod n = ID_i$.

U_{i-1}	U_i	U_{i+1}
Round 1:		
	$r_i \in_R \mathbb{Z}_q$	$\xrightarrow{g^{er_i}, S_i g^{cr_i}, 1}$
Round 2:		
$\xrightarrow{X_1, Y_1, Z_1}$	$T_1 = X_1 Z_1^e$	
	$(Y_1^e / T_1^c)^{e^{M-2}} \stackrel{?}{=} ID_{i-1}$	$\xrightarrow{X_1^{er_i}, Y_1^e S_i^e X_1^{cr_i}, T_1}$
Round j:		
$\xrightarrow{X_{j-1}, Y_{j-1}, Z_{j-1}}$	$T_{j-1} = X_{j-1} Z_{j-1}^e$	
	$(Y_{j-1}^e / T_{j-1}^c)^{e^{M-j}} \stackrel{?}{=} ID_{i-1} ID_{i-2} \ldots ID_{i-j+1}$	
		$\xrightarrow{X_{j-1}^{er_i}, Y_{j-1}^e S_i^{e^{j-1}} X_{j-1}^{cr_i}, T_{j-1}}$
Round m:		
$\xrightarrow{X_{m-1}, Y_{m-1}, Z_{m-1}}$	$T_m = X_m Z_m^e$	
	$(Y_{m-1}^e / T_{m-1}^c)^{e^{M-m}} \stackrel{?}{=} ID_{i-1} ID_{i-2} \ldots ID_{i-m+1}$	
	$\mathcal{Z} = X_{m-1}^{r_i}$	

Protocol 6.13: Koyama–Ohta type 1 identity-based conference key agreement

Koyama and Ohta's second protocol type [191] allows every principal to communicate with each other principal, which they call a *complete graph* architecture. The original version of these protocols was attacked by Yacobi [334] and a new version was published by Koyama and Ohta the following year [192]. Koyama later refined these protocols again [190] due to a conspiracy attack by Shimbo and Kawamura [298]. We shall look only at the latest versions, which are anyway simpler than the originals.

Initialisation phase The trusted authority chooses a modulus N to be the product of three distinct large primes: $N = pqr$. These primes must be large enough so that pq cannot be factorised. A random value e is chosen so that $N/2 < e < N$ and then d is calculated so that $de \bmod \phi(N) = 1$. An element g is chosen by the authority that is of maximal order in $\mathbb{Z}_N^*$. The values N, r, g, e are all made public, together with a hash function h.

Registration phase Each principal U_i wishing to participate in a conference must obtain a secret key S_i. This is generated by the authority from U_i's identifying information ID_i as $S_i = ID_i^d \bmod N$. Notice that $ID_i =$

Petitioner Samsung Ex-1036, 0248

$S_i^e \bmod N$. S_i must be transferred securely to U_i. If desired, the values d, p and q may be destroyed after all principals are registered.

Round 1 Each principal U_i chooses a value $P_i \in_R Z_{r-1}$ and generates a timestamp T_i. Principal U_i then calculates $X_i = g^{eP_i} \bmod N$ and $Y_i = S_i g^{h(X_i,T_i)P_i} \bmod N$. U_i broadcasts the triple (X_i, Y_i, T_i). On receipt of the triples (X_j, Y_j, T_j), for all $j \neq i$, U_i should verify that each timestamp T_j is fresh and, if so, check the following equation.

$$\frac{Y_j^e}{X_j^{h(X_j,T_j)}} \stackrel{?}{=} ID_j \bmod N.$$

If all $m-1$ checks are verified then U_i proceeds to round 2.

Round 2 Principal U_i chooses two values $r_i \in_R \mathbb{Z}_{r-1}$ and $Q_{i,j} \in_R \mathbb{Z}_{n-1}$ and generates a new timestamp T_i'. Then U_i calculates $A_{i,j} = (X_j + Q_{i,j}r)^{er_i} \bmod N$ and $B_{i,j} = S_i(X_j + Q_{i,j}r)^{h(A_{i,j},T_i')r_i} \bmod N$ for all $j \neq i$. U_i sends the triple $(A_{i,j}, B_{i,j}, T_i')$ to U_j. On receipt of its own $m-1$ triples, U_i verifies that the timestamps T_j' are fresh and if so performs the following check, for all $j \neq i$.

$$\frac{B_{j,i}^e}{A_{j,i}^{h(A_{j,i},T_j')}} \stackrel{?}{=} ID_j \bmod N.$$

If all checks pass then U_i calculates the shared secret $\mathcal{Z}$ as follows.

$$\mathcal{Z} = \left(\prod_{j=1}^{m} A_{j,i}\right)^{P_i^{-1} \bmod r-1} \bmod r.$$

If the protocol is run correctly then all principals obtain the same shared secret $\mathcal{Z} = g^{e^2(r_1+r_2+\ldots+r_m)}$. On the assumption that S_i is required in order to form (X_i, Y_i) pairs and $(A_{i,j}, B_{i,j})$ pairs, each principal does obtain implicit authentication of the shared key. There is no key confirmation provided. Forward secrecy is provided since knowledge of the S_i values does not appear to help in obtaining $\mathcal{Z}$. The computational cost per principal is, depending on the implementation, around $4m$ exponentiations, which is high compared with alternatives.

The third protocol type proposed by Koyama and Ohta uses a *star architecture*: messages are exchanged only between one principal, say U_1, and all other principals. The protocol is related to the previous example, but now the shared secret collapses to $\mathcal{Z} = g^{e^2 r_1}$ and so this becomes a key transport protocol in the sense that only U_1 contributes to the shared secret. Considerable computational savings result for all principals except U_1. Furthermore, forward secrecy of the key is still maintained. However, apart from U_1 no other principal now has any explicit assurance of the other members of the conference.

Petitioner Samsung Ex-1036, 0249

The Koyama and Ohta protocols have evolved to take into account a number of attacks. However, there is no formal basis for their security and the problems in earlier versions do not inspire confidence. They also have high computational costs in the versions that provide implicit key authentication. An implementation of these protocols is reported by Goscinski and Wang [134].

Chikazawa and Inoue [89] devised an identity-based conference key protocol that has similarities with Koyama and Ohta's type 3 protocol. One distinguished principal, say U_1, determines the key. The only messages sent are $m-1$ messages from U_1 to each other principal. Shimbo and Kawamura [298] found an attack on this protocol and it was then repaired by Chikazawa and Yamagishi [90]. Their protocol was recently found to be vulnerable to attacks by Shim [297], who proposed further modifications. Similar protocols were also published by Chen and Hwang [151, 86]. None of these protocols provides forward secrecy.

6.4.2 Protocols of Saeednia and Safavi-Naini

Saeednia and Safavi-Naini [285] proposed two identity-based conference key protocols based on the generalised Diffie–Hellman key agreement protocol of Burmester and Desmedt which was examined in Sect. 6.2.6. Their idea is to use identity information of adjacent principals to gain authentication of keying material. A composite modulus $n = pq$ is used to provide a trapdoor so that only the trusted authority is able to calculate user secrets related to identity information. Two bases are used, g and h, which both have order $p'q'$ where $p' = (p-1)/2$ and $q' = (q-1)/2$ are also prime. In an initialisation stage, all principals are issued with two secrets by the authority; these can only be found with knowledge of the $p'q'$ (equivalent to knowing the factorisation of n).

Both of the proposed protocols use this same initialisation. The first protocol is computationally simpler but each party only uses the identity information of its adjacent principals to gain assurance that these are the correct entities. Thus there is no assurance as to who are the other protocol principals involved so this constitutes only a weak form of key authentication. In their second proposal a form of signature is broadcast by all principals, thereby allowing stronger authentication. This is shown in Protocol 6.14.

A close look at Protocol 6.14 reveals that it includes exactly Protocol 6.8 of Burmester and Desmedt. Of course this means that the shared secret,

$$\mathcal{Z} = g^{r_1 r_2 + r_2 r_3 + \ldots + r_m r_1}.$$

is exactly the same, and is calculated by each principal in the same way, as in that protocol. The additional fields used in Protocol 6.14 constitute the signature of each principal on the exchanged t_i and X_i values. However, it is also necessary for the t_i values to be broadcast so that all principals can

Petitioner Samsung Ex-1036, 0250

Information held by trusted authority: Primes p', q' with $p = 2p'+1$ and $q = 2q'+1$ both prime and modulus $n = pq$.
Shared information: Modulus n, elements g and h of order $p'q'$, integer $u < \min(p', q')$ and hash function f.
Information held by U_i: $S_i = g^{ID_i^{-1} \bmod p'q'}$, $T_i = h^{-ID_i^{-1} \bmod p'q'}$.

Phase 1:

U_i chooses $r_i \in_R \mathbb{Z}_{p'q'}$ and calculates $t_i = g^{r_i}$
U_i broadcasts t_i
U_i calculates $c = f(t_1, t_2, \ldots, t_m), X_i = (t_{i+1}/t_{i-1})^{r_i}, w_i = S_i^{f(X_i)r_i} T_i^c$

Phase 2:

U_i broadcasts X_i, w_i
U_i checks $w_j^{ID_j} h^c \stackrel{?}{=} t_j^{f(X_j)}$ for $j \neq i$
U_i calculates $\mathcal{Z} = t_{i-1}^{mr_i} X_i^{m-1} X_{i+1}^{m-2} \ldots X_{i-2}$

Protocol 6.14: Saeednia–Safavi-Naini identity-based conference key agreement

sign them; in the Burmester–Desmedt protocol these need only be passed to adjacent principals.

The signature generation and verification requires $2m$ exponentiations for each principal; this dominates the computational requirements of the protocol except when m is very large. Knowledge of the long-term secrets S_i and T_i does not help in finding r_i and so forward secrecy is provided.

6.5 Conference Key Agreement without Diffie–Hellman

Just as in the two-party case, almost all multi-party key agreement protocols are based on Diffie–Hellman key agreement. However, it is possible to use alternative techniques which can lead to advantages in efficiency. We will look at several examples in this section.

6.5.1 Pieprzyk and Li's Key Agreement Protocol

Secret sharing allows a group of users to cooperate to derive a secret value. Some researchers have shown how this can be exploited as a building block in conference key establishment. The protocol described below employs the threshold scheme of Shamir [294] introduced in Sect. 1.4.5. We will not present

Petitioner Samsung Ex-1036, 0251

the full detail, but note that recovery of secrets from a set of shares can be achieved using only additions and multiplications of the share values.

In Sect. 6.7 we will examine the use of secret sharing for key broadcasting. The idea is to use an $(m+1, 2m)$ threshold scheme for which each receiving principal possesses one share; m further shares are broadcast when the key is established. Pieprzyk and Li [271] used the same basic idea to design a conference key agreement protocol but now each principal sets up its own threshold scheme. Instead of recovering each principal's secret during a protocol run, an image of the secret is found. This allows the same secret sharing scheme to be used multiple times.

Initialisation phase This phase must be run before any conference key is established. Each principal U_i engages in an $(m+1, 2m)$ secret sharing scheme in $\mathbb{Z}_q$ with every other principal. Using Shamir's scheme this means that U_i chooses a polynomial:

$$f_i(z) = a_{i,0} + a_{i,1}z + \ldots + a_{i,m}z^m$$

with coefficients randomly chosen in $\mathbb{Z}_q$. The shares are values $f_i(x)$ with $1 \leq x \leq 2m$. There are twice as many shares as principals, and they are divided into two sets: $s_{i,j}(1) = f_i(2j-1)$ and $s_{i,j}(2) = f_i(2j)$ with $1 \leq j \leq m$. The first set is kept secretly by U_i, while principal U_j is given one share, $s_{i,j}(2)$, from the second set. The share must be sent in a secure way.

The secret associated with U_i is $f_i(0)$ which could be recovered from any $m+1$ shares. The shared secret for the whole group will be $F(0)$ where $F(x) = f_1(x) + f_2(x) + \ldots + f_m(x)$. But neither $F(0)$ nor any $f_i(0)$ will be recovered during any protocol run. This allows multiple conference keys to be found following one initialisation phase.

Key agreement This phase is run each time a new conference key is required. The same group may run this phase multiple times following one initialisation phase.

1. A new base value α is chosen for this session. A third party chooses this by taking a random value r and broadcasting the value $\alpha = g^r$. It is not necessary to keep the value of α (or even r) secret.
2. Principal U_i broadcasts the m values $\beta_{i,j} = \alpha^{s_{i,j}(1)}$, for $1 \leq j \leq m$, which are images of its first set of shares.
3. Each principal (indeed any eavesdropper too) can find the image of m shares of the sum of the secret shares by calculating $\alpha^{F(2j-1)} = \prod_{i=1}^{m} \beta_{i,j}$ for $1 \leq j \leq m$.
4. Principal U_j can also obtain $\alpha^{F(2j)} = \alpha^{\sum_{i=1}^{m} f_i(2j)}$. Thus U_i has the image of $m+1$ shares which allows calculation of the shared secret $\mathcal{Z} = \alpha^{F(0)}$ since addition and multiplication of the exponents can be done by multiplication and exponentiation of α.

Petitioner Samsung Ex-1036, 0252

Pieprzyk and Li provide an informal proof that the protocol maintains the confidentiality of the session key against an adversary who observes multiple runs of the protocol (with the same initialisation phase). They also propose a final step in the key agreement protocol in which each principal broadcasts a hash of three values: the principal's identity, the shared secret $\mathcal{Z}$ and the value α. This gives a weak form of key confirmation; notice that any principal in the protocol can form all such values.

The computational requirements for each principal are mainly due to working with the exponents of α. Calculation of the $\beta_{i,j}$ values requires m exponentiations by each principal, and there is also a multi-exponentiation required to calculate the shared secret. A useful feature of the protocol is that the key agreement phase (excluding key confirmation) can be completed in one round. However, this is only true following some earlier initialisation, while there are a number of other drawbacks of this protocol.

- Confidence in freshness of the key depends on a random α being available. This means that the third party supplying this value must be trusted for this purpose.
- The initialisation phase must be run when a different group wishes to establish a key. Also there are $m-1$ secrets that need to be stored by each principal for each group that it belongs to.
- The secret shares are long-term keys whose compromise leads to compromise of any session key. Therefore forward secrecy is not provided.

Pieprzyk and Li also propose an extended protocol that allows any subgroup to form a session key. A limitation of this extension is that users are not able to verify which members of the group take part in any particular protocol run. A related protocol has been proposed by Anzai *et al.* [16].

6.5.2 Tzeng–Tzeng Protocols

The protocols of Tzeng and Tzeng [322] have the same algebraic setting as Diffie–Hellman but the shared secret is $\mathcal{Z} = g^{r_1+r_2+\ldots+r_m}$ where r_i is the ephemeral private input of principal U_i. Calculation of $\mathcal{Z}$ uses public information together with the long-term private key of any principal, and consequently the protocols do not provide forward secrecy.

In order to provide some element of fault tolerance, use is made of a *proof of knowledge* which allows any entity to check that the message elements sent by each party are properly constructed. We denote by $\texttt{PEDL}(X_1, X_2, \ldots, X_m)$ a proof that the discrete logarithm of X_i to the base y_i is the same for each i with $1 \leq i \leq m$. The computational cost of generating each proof is m exponentiations with a similar cost to verify it.

Tzeng and Tzeng propose two protocols, the first one of which is shown as Protocol 6.15. This protocol makes use of a signature scheme; they specify an ElGamal signature variant, but it is shown as a generic signature in Protocol

Petitioner Samsung Ex-1036, 0253

6.15. Their second protocol incorporates private key information into the proof instead of using a conventional signature.

Shared information: Session identifier SID.

Phase 1:

U_i chooses $r_i \in_R \mathbb{Z}_q$.

U_i calculates and broadcasts the following:

- $t_i = g^{r_i}$.
- $u_{i,j} = y_j^{r_i}$ for $1 \leq j \leq m$.
- $\texttt{PEDL}(u_{i,1}, u_{i,2}, \ldots, u_{i,m})$.
- $Sig_{U_i}(t_i, SID)$.

Phase 2:

U_i verifies proof of knowledge and signature.

U_i calculates $\mathcal{Z} = (u_{1,i} \cdot u_{2,i} \cdot \ldots \cdot u_{m,i})^{x_i^{-1}}$.

Protocol 6.15: Tzeng and Tzeng's conference key agreement

Protocol 6.15 has some interesting properties. The fault tolerance property ensures that if one or more of the proof verifications fails then the party concerned can be eliminated from the calculation of the shared secret. However, this mechanism only works on the strong assumption that the broadcast channel provides integrity of all messages; otherwise a malicious insider can send different proofs to different principals. The cost of verifying each of the proofs from the other $m - 1$ principals is also high; about $m(m - 1)$ exponentiations for each principal.

Tzeng and Tzeng [322] point out that their protocols can be completed in one (synchronous) round, which means that they meet the bound of Becker and Wille for contributory key agreement with broadcasts discussed in Sect. 6.2.9. However, their protocols require the session identifier SID to be known by all participating principals. Unless this session identifier is agreed beforehand their protocols cannot be completed in one round. Tzeng and Tzeng also claim a proof of security, but they provide no reduction proof for a powerful adversary, concentrating instead on the properties of the proof of knowledge. Boyd and González Nieto [63] have shown an explicit attack on the second of the protocols.

6.5.3 Boyd's Conference Key Agreement

Boyd [58] proposed a protocol using generic encryption and signature techniques together with a hash function possessing special properties. Its main

Petitioner Samsung Ex-1036, 0254

virtue is its simplicity and efficiency; perhaps its main drawback is a lack of forward secrecy.

Protocol 6.16 shows the message flows. One principal, say U_1, will be distinguished and will send its random value r_1 to each other user in an authenticated and confidential way. The other users only have to broadcast their messages so that all principals in $\mathcal{U}$ receive all the random r_i values. U_1 will sign the value r_1 together with the identities of all principals in the conference. Since this message is the same for every principal it only needs to be formed and sent once in a broadcast to all users. The value of r_1 is sent to user U_i encrypted with that principal's public key, K_i. The protocol then has two stages. In the first stage U_1 broadcasts one signature and $m-1$ encryptions. In the second stage each U_i broadcasts its r_i value.

Shared information: Hash function h. Signature verification with respect to U_1.
Information held by U_1: Public encryption key with respect to U_i for $2 \leq i \leq m$.

Phase 1:

U_1 broadcasts $\mathcal{U}, Sig_{U_1}(\mathcal{U}, h(r_1))$.
U_1 broadcasts $E_{U_2}(r_1), E_{U_3}(r_1), \ldots, E_{U_m}(r_1)$.

Phase 2:

U_i broadcasts r_i, for $2 \leq i \leq m$.
Each U_i calculates
$\mathcal{Z} = MAC_{r_1}(h(r_2) \oplus h(r_3) \oplus \ldots \oplus h(r_m))$.

Protocol 6.16: Boyd's conference key agreement

The security is based on the properties of the MAC function and h. It must not be possible to calculate $MAC_{r_1}(.)$ without knowledge of r_1; since r_1 is sent confidentially to conference members, no outsider can calculate $\mathcal{Z}$. The key freshness is provided by the one-wayness of h; no party is able to force an old value if at least one principal chooses its r_i freshly. Each principal gets an assurance from U_1 regarding the members of the conference, and so U_1 must be trusted for this purpose. A generalisation of the protocol allows any principals to choose and encrypt a value for all other principals. This can reduce trust but increases the computation and communication.

The protocol can be completed with m broadcast messages. The computation required for U_1 is one signature and $m-1$ public key encryptions. Other principals use only one signature verification and one public key decryption. Compromise of any principal's decryption key results in compromise of $\mathcal{Z}$ so, as mentioned above, forward secrecy is not provided.

Petitioner Samsung Ex-1036, 0255

Protocol 6.16 is a contributory key agreement protocol in the sense of Becker and Wille [25]. Each of the broadcast messages can be sent simultaneously in one synchronous round. Therefore this protocol satisfies Becker and Wille's bound of one synchronous round for key agreement with broadcasts, mentioned in Sect. 6.2.9. Boyd and González Nieto [63] have proven the security of an almost identical protocol in the Bellare–Rogaway model. The differences are that the MAC is replaced by a function f that is assumed to act as a random oracle, and the shared secret is calculated as $\mathcal{Z} = f(r_1, r_2, \ldots, r_m)$.

6.6 Conference Key Transport Protocols

The majority of published multi-party key establishment protocols rely on key agreement. In this section we look at a few conference key protocols using key transport. Many key transport protocols designed for two parties can be extended to the multi-party case in a straightforward fashion. We will first look at some simple examples based on different logical architectures. Then we look at a method of Mayer and Yung for systematically and formally extending two-party protocols for multiple parties. Following this we look briefly at proposals for key transport for dynamic groups using key hierarchies. Key transport protocols require a distinguished principal who generates the keys; we will sometimes call this principal the *group manager*.

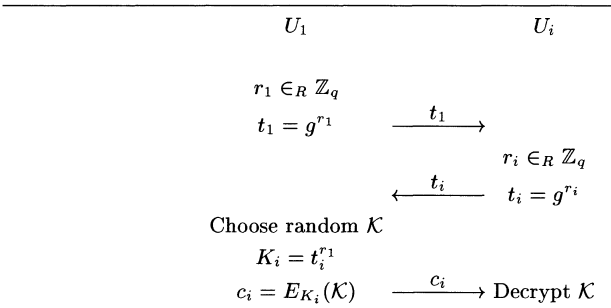
Before looking at the specific protocols, it should also be mentioned that key pre-distribution schemes, which were introduced in Sect. 5.10.2, have been generalised to the multi-party case. The idea is that a trusted centre will distribute secrets to each member of a community in such a way that different subgroups can derive a shared secret known only to themselves. Such schemes may be regarded as 'no-message' protocols since each party is able to derive the correct key for a given group without any interaction. This means that additional measures must be taken in order to obtain a fresh session key. Stinson [310] has made a survey of such methods.

6.6.1 Burmester–Desmedt Star and Tree Protocols

As well as their key agreement protocol (Protocol 6.9) Burmester and Desmedt [75] proposed two simpler key transport protocols based on star and tree configurations. As with their key agreement protocol, these are initially proposed in unauthenticated versions.

In the star protocol a distinguished principal U_1 acts as a manager and interacts with every other principal. First U_1 agrees a Diffie–Hellman key K_i with U_i and then sends the chosen session key $\mathcal{K}$ to U_i encrypted with K_i. The message flows are shown in Protocol 6.17. Note that only messages between U_1 and a single other principal, U_i, are shown; a similar exchange takes place between U_1 and all other principals.

Petitioner Samsung Ex-1036, 0256

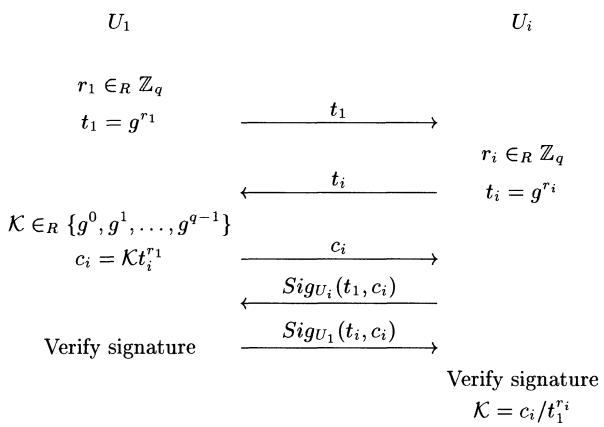


Protocol 6.17: Burmester–Desmedt star protocol

Burmester and Desmedt suggested that Protocol 6.17 may be extended to form an authenticated key transport protocol. To do so each principal U_i should authenticate the value t_i when sent to other principals. This authentication can be achieved using a generic method such as a digital signature.

Hirose and Yoshida [144] proposed a very similar conference key protocol employing their own signature scheme, which was also used in their key agreement scheme (Protocol 5.26 in Chap. 5). We show the message flows in Protocol 6.18 with a generic signature. Only messages between U_1 and a single other principal, U_i, are shown since messages with all other principals are similar.

Shared information: Signature verification information for all principals.



Protocol 6.18: Hirose–Yoshida conference key transport protocol

Petitioner Samsung Ex-1036, 0257

The confidentiality of $\mathcal{K}$ is protected only by the ephemeral keys and so forward secrecy is provided by this protocol. Apart from U_1, no principal gains explicit assurance of the other parties who obtain $\mathcal{K}$. The trust required in U_1 and the high computational load for this principal are the main drawbacks of this protocol. In an earlier protocol, Hirose and Ikeda [143] used a novel approach to key confirmation by allowing all users to contribute to a *multisignature* on the session key, using principal U_1 as a combiner. This multisignature can be verified by all principals using all of their public keys.

Burmester and Desmedt's tree protocol distributes the computational load more evenly amongst principals. Principals are arranged in a binary tree, each principal representing a node in the tree. (Leaf nodes may be left empty.) During the protocol each principal first agrees a Diffie–Hellman key with its parent node and its two child nodes. (The root of the tree has no parent so agrees a key only with its children, and the leaves of the tree have no children, so they agree a key only with their parent.) Once all keys are agreed, the root principal generates the session key $\mathcal{K}$ which is sent recursively by every parent to its children protected, by multiplication, with the shared key.

As with the star protocol, Burmester and Desmedt suggest to add authentication by having each party sign its t_i value for each of its recipients. The advantage of the tree protocol over the star protocol is that the computation is fairly even across all nodes: leaves require two exponentiations, internal nodes four exponentiations, and the root three exponentiations. The drawback is that the number of rounds required increases to $1 + \lceil \log_2 m \rceil$ because $\mathcal{K}$ can only be propagated one level per round. In addition each principal needs to trust its parent node to correctly propagate the key. Once again there is no explicit assurance of other key recipients. Forward secrecy is provided because only ephemeral keys are used to protect $\mathcal{K}$.

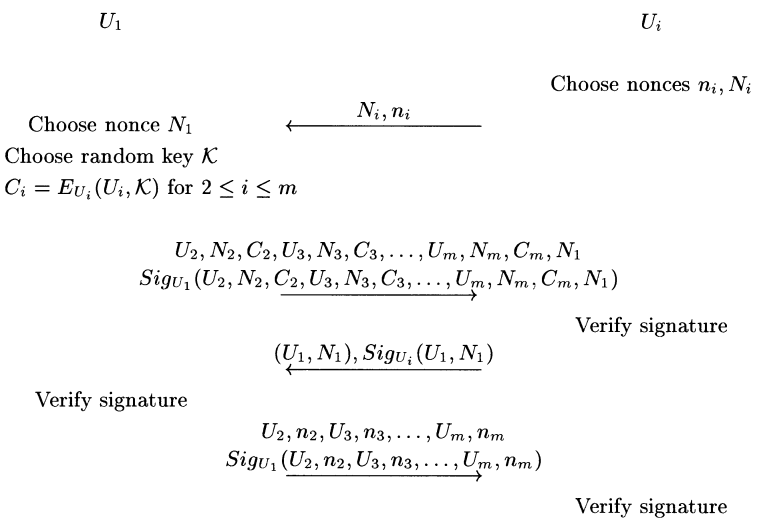
6.6.2 Mayer and Yung's Protocols

Mayer and Yung [226] have made one of the few studies of provable security for multi-party key establishment. They proposed generic transformations that allow two-party key transport protocols to be extended to protocols for multi-party key transport in such a way that a proof for the two-party case automatically extends to the multi-party case. The transformations are quite straightforward, consisting basically of individual runs of the protocol between the group manager U_1 and the key recipients $U_2, \ldots, U_m$ but with the runs authenticated simultaneously. They use a formal model based closely on that of Bellare and Rogaway.

Mayer and Yung include two example protocols for which their method works. One is based on the shared key two-party protocol of Bellare and Rogaway (3.34). This requires that each party already shares a secret with the key sender, which may not be realistic in many applications for authenticated key transport. Their second example is based on Blake-Wilson and Menezes' provable secure protocol (4.18) examined in Chap. 4.

Petitioner Samsung Ex-1036, 0258

Shared information: Public encryption key and signature verification information for all principals.



Protocol 6.19: Mayer–Yung conference key transport protocol

The message flows for the Mayer–Yung generalisation of the Blake-Wilson and Menezes protocol are shown in Protocol 6.19. Only flows between the server U_1 and U_i are shown, although in fact the messages from U_1 are broadcast to all other principals, and the messages from U_i to U_1 are gathered together for all i. The protocol is proven secure in the sense that if all parties accept the key then they must all agree on the messages exchanged and an adversary cannot obtain $\mathcal{K}$.

It may be observed that Protocol 6.19 uses one round more than Blake-Wilson and Menezes' Protocol 4.18 on which it is based. This is due to a deliberate reorganisation of the protocol for reasons that we now explain. In any protocol the adversary can always make one or more principals accept and complete the protocol, while others do not accept. This can be achieved by deleting the final protocol message to any recipient. Mayer and Yung call this situation an *inconsistency* and discuss two alternative consistency requirements.

Consistency type 1 If the key recipients $U_2, \ldots, U_m$ accept the session key then the group manager U_1 has also accepted it.

Consistency type 2 If the group manager U_1 accepts the session key then the recipients $U_2, \ldots, U_m$ have accepted it.

Petitioner Samsung Ex-1036, 0259

A protocol cannot provide both forms of consistency. Mayer and Yung argue that for conference key transport protocols the first type of consistency is more useful than the second type. This is because denial of service attacks may become possible if U_1 is left in a state where it has not yet accepted, but the other principals believe that the protocol is complete. Since U_1 is usually a server, it is more vulnerable to this kind of problem. Mayer and Yung also provide a generic transformation that allows a protocol with one kind of consistency to be transformed into a protocol with the other type of consistency. Protocol 6.19 is the result of this transformation applied to the Blake-Wilson and Menezes protocol, which results in the additional message flow.

6.6.3 Key Hierarchies

Any conference key transport protocol seems to require secure communication between the initiator and all other principals in order to initially establish the key. Solutions in which a single key is securely sent to each member, such as Protocols 6.17, 6.18 and 6.19, are appropriate in static groups. However, in dynamic groups it turns out that much may be gained from using a hierarchy of keys. When the conference key is changed, in particular due to group members being added or deleted, different keys in the hierarchy can be used to save on both communications and computation.

To see the idea, consider the key hierarchy illustrated in Fig. 6.2 which is based on a binary tree. Each principal in the group shares a set of keys with the group manager. Each principal knows the key at one leaf of the tree and also every key that is the parent of a key that it knows. For example, in Fig. 6.2 principal U_5 knows the set of keys that are shaded, namely $\{k_5, k_{56}, k_{5678}, \mathcal{K}\}$. The key $\mathcal{K}$ at the root of the tree is the key shared by the group which can be used for secure communications in the conference.

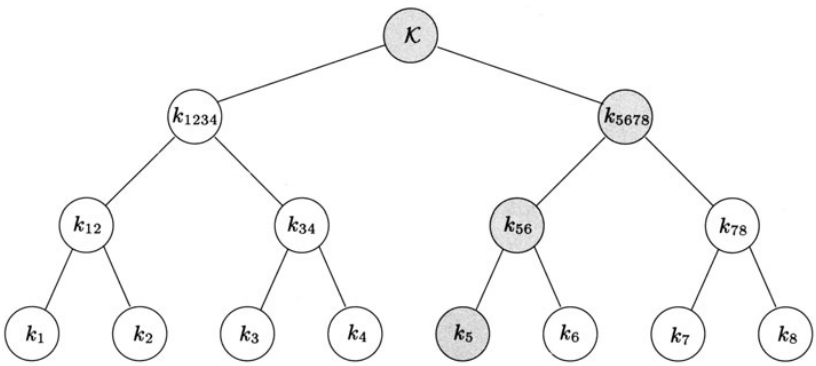


Fig. 6.2. Key hierarchy using binary tree

Petitioner Samsung Ex-1036, 0260

In this example each principal needs to store multiple keys, and indeed this is typical in key hierarchies. However, the potential advantage can immediately be seen if we consider what happens if it is required to remove user U_5 from the conference. The set of keys known to U_5 needs to be changed, but this does not require the group manager to communicate directly with every principal. All principals on the left side of the tree can know the new shared key $\mathcal{K}'$ if the manager encrypts $\mathcal{K}'$ with k_{1234} and sends it to these users (possibly by broadcasting it). Key k_{56} must be replaced by k'_{56} and can be sent to U_6 encrypted with k_6 (and k'_{56} could become the new individual key of U_6). Then key k_{5678} can be replaced by k'_{5678} and sent encrypted with k_{78} and with k'_{56}. In total the manager has to encrypt and send five new keys rather than having to encrypt a new key for all seven remaining principals which would occur with a flat hierarchy.

In general the manager needs to generate a new key for each level of the tree and encrypt each new key with the keys of the two child nodes (except for the deleted node). If $h = \lceil \log_2 n \rceil$ denotes the height of the tree then this means that the manager need only encrypt and send $2h - 1$ keys, which is a considerable saving over $n - 1$ when n is large. A similar saving can be made when adding new group members. (Recall that it is usually required that new members cannot know old group keys, so that $\mathcal{K}$ and all keys on the path from the new member to the root must be changed.)

The idea for using key hierarchies seems to have been published independently at about the same time by both Wong *et al.* [327] and Wallner *et al.* [326]. Both of these papers contain a detailed analysis of the various different related schemes. Wong *et al.* include experimental results on the comparative practical performance of different dynamic group operations depending on parameters such as group size and specific topology of the hierarchy. McGrew and Sherman [228] proposed key hierarchies based on one-way function trees in which the key at each node is defined by the keys at each of its child nodes. This can result in smaller message sizes. All these papers concentrate only on the dynamic operations and say little about establishing the initial group key.

6.7 Key Broadcasting Protocols

Some multi-user applications may have thousands of principals who should receive the same broadcast information that needs to be protected from outsiders. In such applications, interactive protocols involving every principal are quite impractical. The session key must be sent to users in the most efficient manner possible.

The general idea of key broadcasting is for a session key, $\mathcal{K}$, to be chosen by the chair or broadcaster. The key $\mathcal{K}$ must be sent securely to all the relevant principals. The broadcast message itself is then protected by $\mathcal{K}$. A header for the broadcast message will consist of the session key encrypted

Petitioner Samsung Ex-1036, 0261

for all recipients. This encryption can use either symmetric or asymmetric cryptography.

There are a number of features that distinguish key broadcasting protocols from, not only key agreement, but also conference key protocols in which all principals need to gain key authentication.

- There is only one key broadcasting message: all principals receive identical protocol messages.
- The receiving principals do not send protocol messages to each other, nor to the trusted authority.
- Strong key authentication, concerning which other principals may obtain the broadcast key, is usually not important. In some applications it may not even be important to authenticate the trusted authority.
- It may be desirable to provide principal anonymity, in the sense that an eavesdropper (who may be one of the receivers) is unable to find which principals can obtain the session key.

At the time of writing, key broadcasting protocols are still the subject of considerable research. There are a number of issues that go beyond key establishment, which influence the design of such protocols. Some of the widely used terminology is the following.

Broadcast encryption Introduced by Fiat and Naor [112], these protocols are concerned with preventing users outside the group from collaborating to try to find the key. At the same time it is desirable to minimise the number of keys transmitted and stored by each user. Blundo *et al.* [50, 51] found information-theoretic bounds on the various parameters, while other researchers have investigated different aspects [7, 114].

Traitor tracing In some applications it is important to be able to identify users (traitors) who deliberately reveal their secrets in order to allow users outside the group to obtain the broadcast. An example is tracing of the source of pirate decoders for subscription television. Chor *et al.* [92] introduced the study of this problem.

Reliability of protocols The study of reliable networks is a whole subject of its own. A number of researchers [273, 275] have investigated both theoretical and practical performance of reliable multicast networking.

These are all important issues in different application scenarios. However, they extend considerably beyond the concerns of key establishment itself and therefore we do not examine these topics in this chapter. Instead we will simply examine the basic techniques used to establish keys by broadcasting. We divide these into two classes:

1. those using number theory; and
2. those using secret sharing.

In the descriptions below there is a trusted authority T who will be responsible for generating and broadcasting the conference key $\mathcal{K}$. T will share a

Petitioner Samsung Ex-1036, 0262

secret initially with all principals who may be part of a broadcast. In any particular broadcast there will be m principals, $U_1, U_2, \ldots, U_m$ who will receive the broadcast key.

6.7.1 Key Broadcasting Using Number Theory

Chiou and Chen [91] were perhaps the first to propose a protocol for key broadcasting. The authority chooses a new session key $\mathcal{K}$ and encrypts it for each principal intended to receive it, using the secret shared with that principal. The set of encrypted versions of $\mathcal{K}$ is sent as a header for the encrypted message. There are a number of ways that this could be done.

- Each principal can be given a sequence number and the set of all encryptions of the session key is sent sequentially. This simple solution has the drawback that if there are many principals missing from the set to receive a particular message then there can be significant waste in communication. Anonymity for recipients can be maintained if a special string (perhaps all zeros) is (probabilistically) encrypted for those who are not included.
- In order to overcome the drawback mentioned above, an encrypted $\mathcal{K}$ can be included only for the chosen recipients and a label used to indicate the recipients involved. This saves on communications but it is no longer possible to preserve anonymity of recipients. Notice that if no label is used then all users need to attempt to decrypt all fields, resulting in a large increase in computation for recipients.
- Chiou and Chen proposed to wrap all the encrypted versions of $\mathcal{K}$ into a single element using the Chinese Remainder Theorem. Each principal U_i is issued with a modulus p_i larger than the maximum length of the encrypted $\mathcal{K}$. If C_i is the ciphertext intended for U_i then, using the Chinese Reminder Theorem, the authority can find a unique value $B < \prod p_i$ such that $B \bmod p_i = C_i$. A possible drawback, compared with the above solutions, is the extra computation required for each recipient in view of the large modulus that may result. The details are presented in Protocol 6.20.

The CK value allows each recipient to check on the integrity of the key. However, since the modulus values are public, any insider or outsider can verify whether U_i is an intended recipient by checking whether $B \bmod p_i = 0$. Therefore Protocol 6.20 does not provide anonymity of recipients. There is no way for principals to check on freshness of the key so replay attacks are possible. Gong [127] discussed this problem and suggested ways to overcome it. Gong's solutions add a timestamp to the encrypted key and alter Protocol 6.20 to prevent insiders from replaying the same key.

We may regard the Chiou and Chen protocol as using two steps.

1. T encrypts the shared session key $\mathcal{K}$ separately for each user.
2. T combines the encrypted blocks into a single broadcast message.

Petitioner Samsung Ex-1036, 0263

Shared information: Public encryption key for all principals. Modulus p_i corresponding to each principal U_i.

Encryption Phase:

1. T chooses a random session key $\mathcal{K}$ and encrypts the broadcast message M to form $\mathcal{C} = E_{\mathcal{K}}(M)$.
2. T encrypts $\mathcal{K}$ for each principal U_i who is intended to receive the message: $C_i = E_{U_i}(\mathcal{K})$. T then uses the Chinese Remainder Theorem to form $B < \prod p_i$ such that $B \bmod p_i = C_i$.
3. T forms a check value $CK = E_{\mathcal{K}}(\mathcal{K})$.
4. T broadcasts $(B, CK, \mathcal{C})$.

Decryption Phase:

1. Recipient U_i calculates $C_i = B \bmod p_i$.
2. U_i decrypts C_i to obtain $\mathcal{K}'$ and checks whether decryption of CK reveals the same value as $\mathcal{K}'$. If not then U_i halts.
3. U_i decrypts $\mathcal{C}$ to obtain M.

Protocol 6.20: Chiou–Chen's key broadcasting

Just *et al.* [173] pointed out that when it is necessary to update the session key $\mathcal{K}$ (with the group of recipients remaining the same) a new encrypted block must be broadcast to all users. They also proposed a general model which, in a sense, turns around the order of this process.

1. T combines the keys shared with each principal into a single public encryption key K_P.
2. T encrypts $\mathcal{K}$ with K_P and broadcasts both the encrypted $\mathcal{K}$ and K_P.

With this model only the second step needs to be repeated in the case that the conference key for the same principals needs to be renewed. However, to make this model work the keys initially shared between T and other principals must be suitable to be combined into a single public key which allows all principals to decrypt. Just *et al.* proposed two ways to implement their model. One of these uses the secret sharing technique examined below; they also proposed the following number-theoretic protocol.

1. Each principal U_i shares a secret prime p_i with T. T calculates $N = \prod_{i=1}^{m} p_i$ and finds an element y that is a quadratic non-residue modulo p_i for $i = 1, 2, \ldots, m$.
2. T encrypts each bit of $\mathcal{K}$ separately. To encrypt a bit b, T chooses a random $x \in \mathbb{Z}_N^*$ and broadcasts $x^2 y^b \bmod N$. Each U_i can efficiently determine whether the received value is a quadratic residue modulo p_i. If so then the encrypted bit was 0, if not it was a 1.

Petitioner Samsung Ex-1036, 0264

The security of the protocol is based on the proven secure encryption algorithm of Goldwasser and Micali [117]. Its major drawback, which Just *et al.* acknowledged, is the massive data expansion caused by the encryption of the bits of $\mathcal{K}$ separately.

A simple protocol due to Chang *et al.* [85] allows the trusted server to generate a key based on the public keys of all users. However, both Zuk [346] and Burmester [73] showed that a passive eavesdropper is able to obtain the shared key. Although it may be possible to fix that problem, Burmester showed further that any insider in one protocol run can masquerade as the trusted centre in subsequent runs.

6.7.2 Key Broadcasting Using Secret Sharing

The idea to adapt secret sharing for key broadcasting seems to have been first proposed by Laih *et al.* [199] in a special case, and more generally by Berkovits [40]. The basic idea it to use an $(m+1, 2m)$ threshold scheme for which one share is pre-distributed to each principal (this is the secret shared with T). During a protocol run, m shares of $\mathcal{K}$ are broadcast and each user can obtain the new secret using its own share. Protocol 6.21 shows the method using Shamir's secret sharing scheme [294] (see Sect. 1.4.5). The prime p must be chosen to be at least twice the total number of principals.

Initialisation Phase. The secret shared between T and principal U_i is a pair $(s_i, t_i) \in \mathbb{Z}_p \times \mathbb{Z}_p$.

Key Distribution Phase. The authority T chooses the new session key $\mathcal{K}$ and constructs a polynomial $f(z) = \mathcal{K} + a_1 z + \ldots + a_m z^m$ in $\mathbb{Z}_p$ that passes through the m points $(s_1, t_1), (s_2, t_2), \ldots, (s_m, t_m)$. Finally m values $z_1, z_2, \ldots, z_m$ are chosen by T, which are all different from $s_1, s_2, \ldots, s_m$ and the following values are broadcast:

$$(z_1, f(z_1)), (z_2, f(z_2)), \ldots, (z_m, f(z_m)).$$

The m broadcast pairs are *public* shares for the secret sharing of $\mathcal{K}$. Each principal U_i has one extra share (s_i, t_i) and therefore can recover the session key $\mathcal{K}$ using the normal secret recovery protocol.

Protocol 6.21: Key broadcasting using secret sharing

Berkovits suggested that, more generally, an $(m+1+j, 2m+j)$ threshold could be used, for any $j \geq 0$, and $m+j$ public shares revealed. Possible advantages of this are to include more randomisation in the protocol and to hide the exact number of intended recipients.

Just *et al.* [173] refined the basic idea by making the shares broadcast by T simply the coefficients $a_1, a_2, \ldots, a_m$ of $f(z)$. This reduces the amount of

Petitioner Samsung Ex-1036, 0265

communication by half. Since principal U_i knows the value of $f(s_i) = t_i$ the key can be recovered simply. Just *et al.* also showed that in an information-theoretic model, in which the broadcast shares give no information about $\mathcal{K}$, their refined scheme is optimal in the amount of communication.

Using this protocol naturally allows anonymity of receivers. However, when we look at the question of sending different session keys over time there are some problems. A malicious principal who obtains one key gains information regarding the shares of other principals: for each possible s_i value the corresponding t_i value is fixed. Subsequent runs reveal further information since any two polynomials for a conference including U_i must both pass through U_i's share. An outside eavesdropper can also gain this information if the old $\mathcal{K}$ values are revealed.

Laih and Yen [200] extended the basic idea in two different ways.

- They used identity-based public keys to allow any two parties to set up a shared key; in turn this enables any principal to take the role of T.
- They adapted the protocol so that T generates a random value r for each protocol run. The shares of each party are then generated dynamically as (ID_i, r^{s_i}) where s_i is the secret shared between T and U_i, and ID_i is identity information about principal U_i. The broadcast shares are calculated as above and the session key recovered in the same way. The advantage of this technique is seen in repeated protocol runs. If a different r is used for each conference then it does not seem possible for the adversary to gain useful information about s_i values.

A closely related protocol, although expressed somewhat differently, was later published by Wu [331]. Tseng and Jan [320] also published a similar protocol in a slightly different setting. Since knowledge of any of the shared secrets is sufficient to construct the session key, none of these protocols provides forward secrecy.

6.8 Conclusion

In contrast to two-party protocols, current knowledge about protocols for establishment of multi-party keys seems somewhat immature. Few protocols come with any formal proof of security, and even clear definitions of protocol goals for multi-party key establishment are hard to come by. Many protocols have been proposed, but without precise definitions of protocol goals it can be difficult to decide whether they meet their security requirements.

Most multi-party key establishment protocols fall into one of two classes: multi-party key agreement and key broadcasting. They are designed for different applications and have distinct requirements. Generalisations of Diffie–Hellman are many, and the theoretical bounds of Becker and Wille show that we are able to identify protocols that are optimal, at least in certain features.

Petitioner Samsung Ex-1036, 0266

However, when designing authenticated multi-party key agreement with varying efficiency priorities it becomes difficult to assess which are the best current solutions, or whether better ones are possible.

As well as the pressing need for more protocols with proven security, advances in understanding how to integrate security into group management and reliability would be very valuable.

Petitioner Samsung Ex-1036, 0267

7

Password-Based Protocols

7.1 Introduction

Cryptographic authentication relies on possession of a key by the party to be authenticated. Such a key is usually chosen randomly within its domain and can be of lengths from around 100 bits up to many thousands of bits, depending on the algorithm used and security level desired. Experience has shown [109, 333] that humans find it difficult to remember secrets in the form of passwords of even seven or eight characters. But if all upper and lower case letters are used together with the digits 0 to 9 then a random eight-character password represents less than 48 bits of randomness. Therefore we can conclude that even short random keys for cryptographic algorithms cannot be reliably remembered by humans. Another way to express this is that it can be assumed that a computer is able to search through all possible passwords in a short time.

Cryptographic keys are often stored in secure memory in computers or using special devices such as tamper-resistant cryptographic servers or in smart cards. However, there are situations where this is inconvenient or expensive. Not all devices are tamper resistant, and the memory required for public keys can be scarce. Therefore it is desirable to be able to set up secure communications relying only on a short secret that can be remembered by humans.

This chapter examines a number of key establishment protocols that have been designed to be secure in the situation that the principals share only a password of small entropy. At first thought it might seem impossible to achieve key establishment using only a short secret in such a way that brute force searching to find the secret is not possible. This intuition may account for why it was not until 1989 that the first password-based protocols appeared in the literature. These first protocols, due to Lomas *et al.* [210], used the additional assumption that the client (in a client–server application) has knowledge of the server's public key, in addition to sharing the password with the server. Later Bellovin and Merritt [38] introduced a class of protocols that does not require this assumption.

Petitioner Samsung Ex-1036, 0268

The idea of Bellovin and Merritt's Encrypted Key Exchange (EKE) protocols is that the protocol initiator will choose an ephemeral public key and use the shared password to *encrypt* this key. The responder can decrypt the public key and use it to send the session key securely back to the initiator. On the assumption (not always reasonable) that public keys are random strings, an adversary who tries a brute force search of passwords will not be able to distinguish which ephemeral public key was used; furthermore even when the correct public key has been found it cannot be used to discover the session key since it is not possible to obtain the private key from the public key. It is interesting to compare password-based protocols with protocols providing forward secrecy. Both seem to be possible only through use of ephemeral public keys[1] and, for both, a typical approach is to use long-term keys (which may be passwords) only for authentication of the message exchanges. Many password-based protocols do provide forward secrecy.

We may summarise the special properties and requirements for password-based key establishment protocols as follows. Additional properties, such as forward secrecy, are often seen as desirable.

- Users only possess a secret of small entropy. Specifically, it is possible for the adversary to search through all possible secrets in a reasonable time.
- Off-line dictionary attacks should not be feasible. This means that a passive eavesdropper who records the transcript of one or more sessions cannot eliminate a significant number of possible passwords.
- On-line dictionary attacks should not be feasible. This means that an active adversary cannot abuse the protocol so as to eliminate a significant number of possible passwords. An active adversary can always eliminate at least one password per protocol run by attempting to masquerade using that password. Ideally this should be all that the adversary can gain.

There have been many variants of Bellovin and Merritt's EKE protocol, as well as many alternative protocols. Recently these have included some with proven security properties. The original EKE protocol does not specify the encryption algorithm to be used or how to transform the password to a relevant key. Some later protocols have used specific algorithms which typically require a means to map the shared password to an element of the group in which a Diffie–Hellman exchange takes place. Most protocols have variants in which the server stores only an image of the client password rather than the password itself. This means that compromise of the server does not automatically compromise the password; however, it will allow a brute force searching attack so this is not necessarily a significant advantage. Some authors have explored further the situation in which the client has access to a public key of the server and there are now protocols with provable security in this situation too.

[1] Halevi and Krawczyk [140] provide formal arguments in support of this view.

Petitioner Samsung Ex-1036, 0269

The majority of the proposed protocols use Diffie–Hellman-based key agreement with the shared password used for authentication purposes. We therefore require notation similar to that used in Chap. 5. The notation used in this chapter is shown in Table 7.1. Many of the protocols examined in this chapter are client–server protocols for which the actions of the client are different from those of the server. Specifically, the client always possesses the plain password π while the server sometimes stores only an image of π under some one-way function. In order to avoid confusion we always assume that principal A is the client and principal B is the server.

Table 7.1. Notation for password-based protocols

π	A key of short length, such as a password, owned by A.
p	A large prime (usually at least 1024 bits).
q	A prime with $q \mid p-1$.
G	A subgroup of $\mathbb{Z}_p^*$. G is often a subgroup of order q, but sometimes is equal to $\mathbb{Z}_p^*$.
g	A generator of G.
r_A, r_B	Random integers, typically of the same size as the order of G, chosen by A and B respectively.
t_A, t_B	g^{r_A} and g^{r_B} respectively. All computations take place in $\mathbb{Z}_p$.
x_A, x_B	The private long-term keys of A and B respectively.
y_A, y_B	The public keys of A and B, g^{x_A} and g^{x_B} respectively. These public keys will have to be certified in some standard way which we do not consider here.
Z_{AB}	The shared secret calculated by the principals.
K_{AB}	The derived session key.
S_{AB}	The static Diffie–Hellman key of A and B.
$H_i(.)$	One-way hash functions. Functions $H_1, H_2, \ldots$ are often assumed to be independent random functions. Certain protocols may require specific properties and may specify particular functions.

In the next section we examine in some detail the version of EKE based on Diffie–Hellman key exchange and later protocols derived from it. Section 7.3 is concerned with EKE variants in which the server stores only an image of the password; these are often called *augmented* password-based protocols. Then in Sect. 7.4 we look at protocols in which a server shares passwords with two users and establishes a key between them. Most protocols in this chapter are based on Diffie–Hellman, but in Sect. 7.5 some protocols based on RSA are described. Section 7.6 is concerned with protocols that make use of a server public key and we also look at some protocols that do not fit into

Petitioner Samsung Ex-1036, 0270

any of the above sections. In the final section we make a comparison of the properties achieved and resources used in a selection of the protocols.

7.2 Encrypted Key Exchange Using Diffie–Hellman

In this section we examine the Encrypted Key Exchange (EKE) protocol of Bellovin and Merritt [38] as applied to Diffie–Hellman key exchange. Although the original protocol has potential weaknesses and lacks a proof of security, it is instructive to understand what may go wrong with such protocols: completely different attacks are possible from those applicable to protocols with strong keys. We later examine the many variants and extensions that have subsequently been developed from the original idea. Steiner *et al.* [306] give a specification of how to integrate EKE into the TLS protocol including details of how to use the symmetric encryption algorithm.

7.2.1 Bellovin–Merritt's Original EKE

The general idea of EKE is to transmit ephemeral public keys encrypted using the password as a shared key. Only parties that know the password should be able to complete the protocol. This idea can be applied to ephemeral keys from different public key schemes. Here we consider only Diffie–Hellman key exchange in which both principals choose an ephemeral public key; the password is used to encrypt the ephemeral keys as shown in Protocol 7.1.

Shared information: Shared password π. Security parameter L.

A		B
$r_A \in_R \mathbb{Z}_p$		
$t_A = g^{r_A}$	$\xrightarrow{A, \{t_A\}_\pi}$	$r_B \in_R \mathbb{Z}_p$
		$Z_{AB} = t_A^{r_B}$
		$t_B = g^{r_B}$
		$n_B \in_R \{1, \ldots, 2^L\}$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{\{t_B\}_\pi, \{n_B\}_{K_{AB}}}$	
$n_A \in_R \{1, \ldots, 2^L\}$		
	$\xrightarrow{\{n_A, n_B\}_{K_{AB}}}$	Verify n_B
Verify n_A	$\xleftarrow{\{n_A\}_{K_{AB}}}$	

Protocol 7.1: Diffie–Hellman-based EKE protocol

Petitioner Samsung Ex-1036, 0271

As in basic Diffie–Hellman key agreement, the shared secret is $Z_{AB} = g^{r_A r_B}$ although the key derivation function to obtain the session key K_{AB} from Z_{AB} is not specified. Protocol 7.1 requires two exponentiations by each party, which is the same as ordinary Diffie–Hellman. We will see below that there are versions with fewer message passes.

Symmetric Encryption Algorithm

The choice of symmetric encryption algorithm using π was left flexible by Bellovin and Merritt. They suggested that many choices are acceptable, even ones that are 'quite weak'. However, it now seems clear that use of encryption functions without special properties prevents security proofs being obtained and allows attacks in certain cases.

Bellovin and Merritt introduced the notion of *partition attacks* against EKE. The idea is that an adversary guessing the password can attempt to decrypt $\{t_A\}_\pi$ and $\{t_B\}_\pi$ and examine whether the resulting plaintext is a valid Diffie–Hellman ephemeral value. If not then the guessed password is incorrect and can be discarded. Given several runs of the protocol, successive 'partitions' of the passwords into valid and invalid sets may be obtained. The success of partition attacks can depend on two factors: the symmetric encryption used, and the parameters defining the group G in which the protocol works.

Consider an example in which the symmetric encryption algorithm is a conventional block cipher (the key can be derived from the password using a suitable hash function). If $G = \mathbb{Z}_p^*$ then decryption of $\{t_A\}_\pi$ using a candidate password can be assumed to be a random string of the appropriate block length. This string must be interpreted as an element of $\mathbb{Z}_p^*$. During encryption any padding that must be included due to the algorithm block length can be chosen randomly (as suggested by Bellovin and Merritt). Even so, there will be some bit-strings that cannot occur as plaintexts and these allow some passwords to be discarded. This is because if decryption with a candidate password results in a string whose value is greater than p then that candidate is invalid.

In order to make partition attacks harder, Bellovin and Merritt suggest to choose p to be slightly less than a power of 2. In this way very few candidate passwords will be invalid. Jaspan [170] conducted an analysis which concludes that it is adequate in practice to ensure that $1 - (p/2^n) < 10^{-4}$, where 2^n is the smallest power of 2 greater than p. However, it seems preferable to choose the symmetric encryption algorithm to be matched to the group so as to completely eliminate such attacks; we will explain below how this may be done.

Omitting Symmetric Encryption

Bellovin and Merritt suggested that either of the two encryptions using π in Protocol 7.1 could be omitted. Subsequent authors have shown that this can

Petitioner Samsung Ex-1036, 0272

result in weaknesses depending on the precise format of the messages used for authentication.

First suppose that the encryption using π is omitted from the first message so that this message becomes: A, t_A. This change does not appear to help a passive adversary and still prevents K_{AB} from being found without knowledge of π. However, Patel [262] shows how an active adversary masquerading as A may be able to exploit this change. The adversary can send the first message by choosing r_C and setting $t_A = g^{r_C}$. Now if the authenticator $\{n_B\}_{K_{AB}}$ returned in the second message contains redundancy then the adversary can use it to eliminate any potential password $\tilde{\pi}$ by decrypting $\{t_B\}_{\tilde{\pi}}$ with $\tilde{\pi}$ to obtain $\tilde{t}_B$, finding the corresponding value of the session key $\tilde{K}_{AB}$ from $\tilde{Z}_{AB} = (\tilde{t}_B)^{r_C}$, and checking for the redundancy after decrypting $\{n_B\}_{K_{AB}}$ with $\tilde{K}_{AB}$. Jablon [164] noted that small subgroup attacks (see Sect. 5.2.1) may also become possible in this situation; the adversary can move t_A into any small subgroup and use the authenticator in the third message to identify the correct K_{AB} in a brute force attack. Standard precautions against small subgroup attacks can prevent this.

If encryption using π is omitted from the second message then the adversary can attempt to masquerade as B. Steiner *et al.* [307] showed that this allows an attack if the authenticators *do not* contain redundancy. The adversary chooses a random X and r_C, sets $t_B = g^{r_C}$, and sends t_B, X as the second message, which will be accepted by A. Now the adversary can decrypt $\{t_A\}_{\pi}$ with a candidate password $\tilde{\pi}$ to obtain $\tilde{t}_A$ and the corresponding session key $\tilde{K}_{AB} = (\tilde{t}_A)^{r_C}$. The adversary can then decrypt message 3 using $\tilde{K}_{AB}$ and check whether the second field equals the decryption of X using $\tilde{K}_{AB}$, in order to confirm whether $\tilde{\pi}$ is the correct password. Patel [262] also noted an attack here when there is limited redundancy in the authenticator, but this time it is necessary for the adversary to wait to see if a random authenticator is accepted by A: the adversary sends t_B, X as the second message for a random value X. If this message is accepted then the adversary knows that the redundancy is present, but if it is not then all passwords that result in the desired redundancy can be eliminated.

It may seem reasonable to conclude that it is safest not to omit either of the symmetric encryptions using π. However, a better solution is to be more careful in the use of authenticators. We now examine variants in which this is done; not only is the result a protocol with fewer rounds, but also a proof of security becomes feasible.

7.2.2 The PAK Protocol

Bellare *et al.* [29] have provided proofs that the exchange of the password-encrypted Diffie–Hellman messages at the core of EKE is a secure protocol using the Bellare–Rogaway model discussed in Chap. 2. Although they specify the key derivation function to find the session key the symmetric encryption algorithm used in the protocol is not concretely defined; it is assumed to be

Petitioner Samsung Ex-1036, 0273

an 'ideal cipher' that can be regarded as a random function (analogous to the functions used in the random oracle model). Furthermore, the decryption function must take strings of a fixed size and map them to elements of G. This may leave the implementer in some difficulty in choosing an appropriate concrete algorithm. Later MacKenzie [218] proposed concrete ways to achieve a suitable algorithm along with complete proofs.

In parallel work, Boyko *et al.* [65] specified a slightly different version of Diffie–Hellman-based EKE called PAK (Password Authenticated Key exchange). They provided a proof of security in Shoup's simulation model, and later MacKenzie provided proofs in the Bellare–Rogaway model too. The encryption function used is simply multiplication in the Diffie–Hellman group.

PAK uses a prime p of the form $p = rq + 1$, with r and q relatively prime. The Diffie–Hellman key exchange takes place in the subgroup G of order q. The original PAK protocol uses four hash functions. It is important that these functions are different and the formal proof assumes they are independent random functions. In practice we may define the different hash functions by prepending a different fixed string to the input and then using a standard hash function. For example, we may define $H_i(x) = H(\mathrm{ASCII}(i), x)$, where H is SHA-1 [245] and $\mathrm{ASCII}(i)$ is the ASCII representation of i. Protocol 7.2 shows the basic version of PAK. Output from H_1 is treated as an element of $\mathbb{Z}_p^*$ and is used to map π to an element of G which is an important part of defining the symmetric encryption algorithm.

A key feature of Protocol 7.2 is that the element P in the group G is derived from the password π using the formula $P = H_1(A, B, \pi)^r$. In typical applications the computing device of A will not have the value π available beforehand and so this calculation is part of the computational requirements for A. (In contrast we may assume that B is a server which can store the derived value of P.) Since typical lengths of q and r may be 160 bits and 864 bits respectively, this calculation is more expensive than the subsequent Diffie–Hellman exchange. The computational requirements may be optimised for different applications by choosing a larger size for q so that the size of r becomes smaller.

The PAK protocol was proven by Boyko *et al.* [65] to be secure in Shoup's simulation model assuming the hash functions act as random oracles. Originally this result relied on the Decision Diffie–Hellman assumption but later MacKenzie [218] showed that the ordinary (computational) Diffie–Hellman assumption is sufficient. We may regard the PAK protocol as a variant of the original Diffie–Hellman-based EKE protocol with the following instantiations and modifications:

- the key derivation function is specified;
- symmetric encryption is defined as multiplication in G by the image P of π;
- symmetric encryption of the ephemeral Diffie–Hellman key in the second message is omitted;

Petitioner Samsung Ex-1036, 0274

Shared information: Generator g of G where $p - 1 = qr$. Hash functions H_1, H_2, H_3, H_4.
Information held by A: Password π.
Information held by B: Derived password image $P = H_1(A, B, \pi)^r$.

A		B
$P = H_1(A, B, \pi)^r$		
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
$m = t_A P$	$\xrightarrow{m}$	$m \bmod p \stackrel{?}{\neq} 0$
		$t_A = m/P$
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
		$Z_{AB} = t_A^{r_B}$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{t_B, k}$	$k = H_2(A, B, m, t_B, Z_{AB}, P)$
$k \stackrel{?}{=} H_2(A, B, m, t_B, Z_{AB}, P)$		
$k' = H_3(A, B, m, t_B, Z_{AB}, P)$	$\xrightarrow{k'}$	$k' \stackrel{?}{=} H_3(A, B, m, t_B, Z_{AB}, P)$
$K_{AB} = H_4(A, B, m, t_B, Z_{AB}, P)$		$K_{AB} = H_4(A, B, m, t_B, Z_{AB}, P)$

Protocol 7.2: PAK protocol

- a simplified authentication mechanism is used which allows a more efficient protocol.

It is interesting to consider how these changes ensure that the potential weaknesses in Diffie–Hellman EKE are avoided. The choice of symmetric encryption is very simple and matched to the group G. (Although a modular multiplication may be computationally more expensive than encryption with a block cipher, in practice the additional effort over the required exponentiations is very small.) In the partition attack described above the adversary attempts decryption with candidate passwords π'. With this natural choice of encryption, every π' maps to an image $P' \in G$ and attempted decryption of $t_A P$ results in $t_A P / P'$ which will always lie in G. Therefore the partition attack cannot even discount a single candidate π'. If the adversary tries to exploit the omission of the symmetric encryption with π in the second message by sending t_B, X for a random X then with overwhelming probability A will reject the message. Instead the adversary can calculate a correct value k for any candidate password and this would allow checking of that one password.

Boyko *et al.* [65] also specify a variant protocol of PAK without explicit authentication. The result is shown as Protocol 7.3 and is known as PPK (Password Protected Key exchange). Apart from omitting the authenticators

Petitioner Samsung Ex-1036, 0275

the difference is that both messages, instead of just the first, must now use the symmetric encryption with π. A consequence of using two encryptions is that A requires an extra exponentiation, in comparison with Protocol 7.2, to derive the extra password image P_2.

Shared information: Generator g of G where $p-1=qr$. Hash functions H_1, H_2, H_3.
Information held by A: Password π.
Information held by B: $P_1 = H_1(A,B,\pi)^r$ and $P_2 = H_2(A,B,\pi)^r$.

A		B
$P_1 = H_1(A,B,\pi)^r$		
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
$m = t_A P_1$	$\xrightarrow{m}$	$m \bmod p \overset{?}{\neq} 0$
		$t_A = m/P_1$
		$r_B \in_R \mathbb{Z}_q$
$P_2 = H_2(A,B,\pi)^r$		$t_B = g^{r_B}$
$m' \bmod p \overset{?}{\neq} 0$	$\xleftarrow{m'}$	$m' = t_B P_2$
$t_B = m'/P_2$		
$Z_{AB} = t_B^{r_A}$		$Z_{AB} = t_A^{r_B}$

$$K_{AB} = H_3(A,B,m,m',Z_{AB},P_1)$$

Protocol 7.3: PPK protocol

If the first message alone was protected by the password, as in Protocol 7.2, then an adversary could masquerade as B and use knowledge of the derived session key to mount a brute force searching attack. For example, the adversary could choose random $r_C \in \mathbb{Z}_q$ and send g^{r_C} to A as the second message. Then the shared secret accepted by A is $Z_{AB} = g^{r_C r_A}$. The adversary can check any potential password $\tilde{\pi}$ by calculating $\tilde{P}_1 = H_1(A,B,\tilde{\pi})^r$, $\tilde{t}_A = m/\tilde{P}_1$ and the corresponding secret $\tilde{Z}_{AB} = (\tilde{t}_A)^{r_C}$. If $\tilde{\pi}$ is the correct password then $\tilde{Z}_{AB} = Z_{AB}$, which can be checked against subsequent messages from A protected with K_{AB}.

MacKenzie [216] later designed several other variants of the PAK protocol and proved their security. These included versions of the protocol for elliptic curve groups and the XTR group [204]. Protocol 7.4 is an ingenious variant which transfers computational load from the client A to the server B. This is very useful in applications in which the client is a device, such as a smart card, with reduced computational power. The idea is that instead of moving the password image into the group G, the password is simply regarded as a

Petitioner Samsung Ex-1036, 0276

value in $\mathbb{Z}_p^*$ and is masked by a random value of order r. The result is that A no longer needs to compute the exponentiation $H_1(A, B, \pi)^r$ with the large exponent r. However, in order to remove the element of order r, B now has to compute a costly exponentiation instead.

Shared information: Generator g of G where $p - 1 = qr$. Hash functions H_1, H_2, H_3, H_4.
Information held by A: Password π.
Information held by B: Derived password image $P = H_1(A, B, \pi)$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
$h \in_R \mathbb{Z}_p^*$		
$m = t_A h^q \cdot H_1(A, B, \pi)$	$\xrightarrow{m}$	$m \bmod p \stackrel{?}{\neq} 0$
		$t_A = ((m/P)^r)^{r^{-1} \bmod q}$
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
		$Z_{AB} = t_A^{r_B}$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{t_B, k}$	$k = H_2(A, B, m, t_B, Z_{AB}, P)$
$k \stackrel{?}{=} H_2(A, B, m, t_B, Z_{AB}, P)$		
$k' = H_3(A, B, m, t_B, Z_{AB}, P)$	$\xrightarrow{k'}$	$k' \stackrel{?}{=} H_3(A, B, m, t_B, Z_{AB}, P)$
$K_{AB} = H_4(A, B, m, t_B, Z_{AB}, P)$		$K_{AB} = H_4(A, B, m, t_B, Z_{AB}, P)$

Protocol 7.4: PAK-R protocol

7.2.3 SPEKE

The SPEKE (Secure Password Exponential Key Exchange) protocol was designed by Jablon [164]. Although SPEKE, like EKE, is based on Diffie–Hellman, it has a distinctive feature that the password is used to define the base to be used for the Diffie–Hellman exchange. Jablon presents both basic and 'fully constrained' versions, the latter designed to prevent a variety of different attacks.

Protocol 7.5 describes the fully constrained SPEKE protocol, but we have used the verifiers, V_A and V_B, as defined in the version analysed by MacKenzie [217]. Jablon's original verifiers included only Z_{AB} in the hash, which allows a potential attack in which the adversary masquerades as A and makes a guess $\tilde{\pi}$ for the password π. One verifier then allows the adversary to check if $\tilde{\pi}$

Petitioner Samsung Ex-1036, 0277

is any value in the set $\{\pi, \pi^2, \pi^3, \ldots\}$. The seriousness of such an attack can be debated since there may be few potential passwords in this sequence, but MacKenzie's verifiers rule it out altogether.

Shared information: Subgroup G of $\mathbb{Z}_p^*$ of prime order $q = (p-1)/2$. Shared derived password image $P = \pi^2$ where π is interpreted as an element of $\mathbb{Z}_p^*$. Security parameter L.

A		B
$r_A \in_R \{1, \ldots, 2^L\}$		
$t_A = P^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \{1, \ldots, 2^L\}$
		$Z_{AB} = t_A^{r_B}$
		Check $Z_{AB} \notin \{-1, 0, 1\}$
		$t_B = P^{r_B}$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{t_B, V_B}$	$V_B = H_1(t_A, t_B, Z_{AB}, \pi)$
Check $Z_{AB} \notin \{-1, 0, 1\}$		
Verify V_B		
$V_A = H_2(t_A, t_B, Z_{AB}, \pi)$	$\xrightarrow{V_A}$	Verify V_A
$K_{AB} = H_3(Z_{AB})$		$K_{AB} = H_3(Z_{AB})$

Protocol 7.5: SPEKE protocol

The prime p and subgroup G are chosen so that $q = (p-1)/2$ is prime and G is of order q. The password π is regarded as a number in $\mathbb{Z}_p^*$ and then $P = \pi^2$ is guaranteed to lie in G and have order q (assuming that π is not equal to 1, -1, or 0). The value P is used as the generator of G for protocol runs involving π. Apart from this special way of defining the group generator, the protocol is exactly the basic Diffie–Hellman key exchange with key confirmation. The shared secret is therefore $Z_{AB} = P^{r_A r_B}$ and forward secrecy is provided. The check that Z_{AB} is not in the set $\{-1, 0, 1\}$ ensures that small subgroup attacks are avoided. In comparison with Protocol 7.2, calculation of P from π is much simpler so that we can almost ignore it in assessing A's computational requirements. The cost of the Diffie–Hellman exchange calculations depends on the size of the exponents r_A and r_B. Typical lengths might be 160 bits.

The original version of SPEKE does not have a proof of security, but later MacKenzie [217] did provide a proof of a slight variant using Shoup's simulation model. This proof relies on the difficulty of a non-standard decision problem (called the 'Inverted-Additive Diffie–Hellman problem'). It shows that at most two passwords can be tested per login attempt, whereas it is proven that only one password test is possible for the PAK protocol. The differences between the protocol used by MacKenzie and Protocol 7.5 are:

Petitioner Samsung Ex-1036, 0278

- P is defined to include the identities of A and B: $P = (H(A, B, \pi))^2$;
- the hashes used to form the session key include the identities of A and B, the exchanged messages t_A and t_B, the password π, as well as Z_{AB};
- the exponents are chosen randomly in $\mathbb{Z}_q$.

An important practical consequence of the last difference is that exponents are the same size as q. Since $q = (p-1)/2$ this would typically make the exponents around 1024 bits, a significant overhead compared with exponents of size 160 bits suggested by Jablon. Versions of SPEKE suitable for use on elliptic curves, which are potentially much more efficient, are specified in the emerging IEEE P1363.2 standard [155].

A flawed variant of SPEKE appeared in the original paper of Jablon [164]. In this variant the derived password is $P = g^{\pi}$ which, as in Protocol 7.5, is used as generator of the group G in a Diffie–Hellman exchange. The shared secret is again $Z_{AB} = P^{r_A r_B}$. A password guessing attack was found by Jablon and others and resulted in the deletion of this option from later papers on SPEKE [165]. An essentially identical protocol also appears in the paper of Bakhtiari *et al.* [23]. A protocol designed by Seo and Sweeney [293] known as SAKA (Simple Authenticated Key Agreement) was an unfortunate re-discovery of almost the same protocol. The main difference is that the shared secret is $Z_{AB} = g^{r_A r_B}$ but this seems to have little influence. Mitchell [236] showed that a password guessing attack is still possible. A number of other authors [319, 194, 209] have also found various attacks concerned with the explicit authentication exchange phase of SAKA.

A protocol by Kaufman and Perlman [179] is connected with SPEKE only in that the password is used to generate the parameters for the Diffie–Hellman exchange. However, the difference is that in this case the password is used as a seed to generate the prime modulus p, and this is done in such a way that 2 is a generator (at least with high probability) of $\mathbb{Z}_p^*$. The protocol is known as PDM (for Password Derived Moduli). Like SPEKE, it is essentially basic Diffie–Hellman with key confirmation. A computational overhead is the time taken to reconstruct p at both ends, which entails testing for primality (indeed testing for a strong prime) starting from the seed value. In addition, no formal security proof is provided.

7.2.4 Katz–Ostrovsky–Yung Protocol

The protocol of Katz *et al.* [178] does not strictly belong in this section, since it is not really an EKE variant. However, the algebraic setting is the same as other protocols in this section, and there are other similarities too. The protocol has a strong theoretical significance because of its security proof. The proof of security for all the PAK protocol variants uses the random oracle model. From a theoretical viewpoint, eliminating such an assumption is very desirable. The protocol of Katz *et al.* is proven secure in the Bellare–Rogaway model without using random oracles, assuming the Decision Diffie–Hellman

Petitioner Samsung Ex-1036, 0279

problem is hard. The encryption algorithm used in the protocol is a variant of the Cramer–Shoup algorithm [97], chosen because it is proven to provide non-malleability without itself using random oracles. As shown in Protocol 7.6, there are a number of public parameters.

Shared information: Generators g_1, g_2, h, c, d of G where $p - 1 = qr$. Hash function H. Password π.

A		B
Generate ephemeral signature keys PK, SK		
$r_A \in_R \mathbb{Z}_q$		
$t_A = g_1^{r_A}$		
$u_A = g_2^{r_A}$		
$v_A = h^{r_A} g_1^{\pi}$		
$\alpha = H(A, PK, t_A, u_A, v_A)$		
$w_A = (cd^{\alpha})^{r_A}$		
	$\xrightarrow{A, PK, t_A, u_A, v_A, w_A}$	
		$x_B, y_B, z_B, z'_B, r_B \in_R \mathbb{Z}_q$
		$\alpha = H(A, PK, t_A, u_A, v_A)$
		$E = g_1^{x_B} g_2^{y_B} h^{z_B} (cd^{\alpha})^{z'_B}$
		$t_B = g_1^{r_B}$
		$u_B = g_2^{r_B}$
		$v_B = h^{r_B} g_1^{\pi}$
		$\beta = H(B, E, t_B, u_B, v_B)$
		$w_B = (cd^{\beta})^{r_B}$
	$\xleftarrow{B, E, t_B, u_B, v_B, w_B}$	
$\beta = H(B, E, t_B, u_B, v_B)$		
$x_A, y_A, z_A, z'_A \in_R \mathbb{Z}_q$		
$K = g_1^{x_A} g_2^{y_A} h^{z_A} (cd^{\beta})^{z'_A}$		
	$\xrightarrow{K, Sig_{SK}(\beta, K)}$	
		Verify signature
$Z_{AB} = E^{r_A} t_B^{x_A} u_B^{y_A} (v_B / g_1^{\pi})^{z_A} w_B^{z'_A}$		$Z_{AB} = K^{r_B} t_A^{x_B} u_A^{y_B} (v_A / g_1^{\pi})^{z_B} w_A^{z'_B}$

Protocol 7.6: Katz–Ostrovsky–Yung protocol

The first message sent from A to B consists of the encryption of the plaintext message g_1^{π} in the Cramer–Shoup variant. However, an amusing aspect of the protocol is that the encryption is turned around; the public parame-

Petitioner Samsung Ex-1036, 0280

ters g_1, g_2, h, c, d form a public key for which no corresponding private key is known. This ensures that information on the password is not leaked. But since B already knows the value g_1^π, he can obtain the correct parameters to form the session key. The second message returned from B is also an encryption of g_1^π using a different Cramer–Shoup variant but with the same public parameters. Notice that the signature sent in the third message uses a public key generated randomly by A during the protocol, and is not intended to provide entity authentication. Indeed, neither party obtains key confirmation or entity authentication. Katz *et al.* note that the usual additional checks are required, in particular that received values lie in the group G.

Although the protocol of Katz *et al.* is certainly practical, its computational requirements are significantly greater than any of the PAK variants, around 5–10 times as great depending on which variant is compared and how the computation is measured.

7.3 Augmented EKE

For many years it has been standard practice for passwords to be stored on servers under the image of a one-way function H, so that $H(\pi)$ is stored rather than π itself. In this way compromise of the password file will not compromise the passwords directly, and when a claimed password π' is submitted to the server it can be checked whether $H(\pi') = H(\pi)$. However, compromise of the password file will nevertheless allow an off-line dictionary attack to be mounted since the adversary can calculate $H(\pi')$ for any password guess π' and compare with the stored value. Therefore the benefits of this approach will depend on the application. Many designers have provided password-based key establishment protocols that require the server to store only an image of each password.

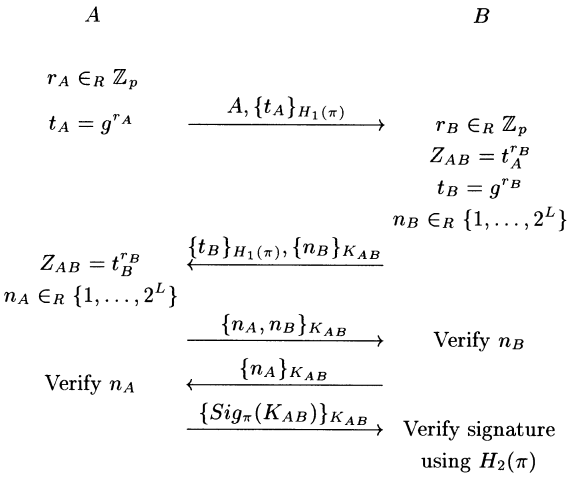
A number of protocols explicitly include the use of *salt* in the calculation of the password image. Salt is another commonly used mechanism for protecting passwords on a server. For each password π_i, a random salt value s_i is chosen and the pair $(s_i, H(\pi_i, s_i))$ can be stored on the server. The purpose of salt is to frustrate bulk guessing attacks if the password file stored on the server becomes compromised. Even though the salt becomes known to the adversary, any password guess can only be verified against one password image at a time.

The original EKE protocol requires that the server stores the plaintext password in order to be able to decrypt the protocol exchanges. Subsequently Bellovin and Merritt [39] designed an 'augmented' version, shown in Protocol 7.7, which requires the server to store only an image of the password.

The first four messages of this protocol are the same as Protocol 7.1 with the plain password π replaced by the hashed password $H_1(\pi)$. These messages are insufficient on their own since the user would only require $H_1(\pi)$ in order to complete the protocol so there is no effective difference from the original EKE. Therefore the fifth message is added which consists of a signature of the

Petitioner Samsung Ex-1036, 0281

Information held by A: Password π.
Information held by B: Two images of password, $H_1(\pi)$ and $H_2(\pi)$. $H_2(\pi)$ is suitable for signature verification.



Protocol 7.7: Augmented Diffie–Hellman-based EKE protocol

shared secret constructed using π as the secret key and so that B can hold the corresponding public key. Bellovin and Merritt did not make an explicit choice for the signature, but an example could be an ElGamal signature [106] with private key π and public verification key $H_2(\pi) = g^\pi$. (It is possible to allow $H_1 = H_2$ but if, as in this example, H_2 requires an exponentiation, computation is reduced for A if H_1 is a simple hash function.)

Steiner *et al.* [307] have pointed out that an adversary who is able to obtain an old K_{AB} can decrypt the final message flow of Protocol 7.7 and attempt to verify the signature using a guessed π' value and therefore mount a dictionary attack. This protocol is therefore arguably weaker than the original in the usual scenario where old session keys may be compromised. However, this problem could be avoided by using a different key derived from Z_{AB} to protect the protocol messages, instead of using the session key K_{AB} for this purpose.

A variant of PAK, called PAK-X, was designed by Boyko *et al.* [65] and can also be considered a variant of augmented EKE. The protocol is very similar to Protocol 7.2 but incorporates an additional verifier. Later MacKenzie [216] devised a conceptually simpler version called PAK-Y shown in Protocol 7.8. The first two messages are essentially identical to those of Protocol 7.2, but in the last message A sends the Schnorr signature [291] of the fields

Petitioner Samsung Ex-1036, 0282

A, B, m, t_B, Z_{AB}, V using the derived password v as the private key. This allows B to verify that A possesses v using the corresponding public key V.

Information held by A: Password π from which is obtained $v = H_0(A, B, \pi)$. (In order to ensure a unique verifier per pair, identities may be ordered and then $v = H_0(\min(A, B), \max(A, B), \pi)$.) Hash functions H_1, H_2, H_3, H_4.
Information held by B: Derived password image $V = g^v$.
Shared information: Group G of prime order $q = (p-1)/r$ and generator g of G.

A		B
$V = g^v$		
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		$P = H_1(A, B, V)^r$
$m = t_A \cdot H_1(A, B, V)^r$	$\xrightarrow{m}$	$m \bmod p \stackrel{?}{\neq} 0$
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
		$Z_{AB} = (m/P)^{r_B}$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{t_B, k}$	$k = H_2(A, B, m, t_B, Z_{AB}, V)$
$k \stackrel{?}{=} H_2(A, B, m, t_B, Z_{AB}, V)$		
$c \in_R \mathbb{Z}_q$		
$a = g^c$		
$e = H_3(A, B, m, t_B, Z_{AB}, a, V)$		
$s = c - ev \bmod q$	$\xrightarrow{e, s}$	$e \stackrel{?}{=} H_3(A, B, m, t_B, Z_{AB}, g^s V^e, V)$
$K_{AB} = H_4(A, B, m, t_B, Z_{AB}, V)$		$K_{AB} = H_4(A, B, m, t_B, Z_{AB}, V)$

Protocol 7.8: PAK-Y protocol

The augmented versions of PAK can also be combined with the 'R' version shown in Protocol 7.4 in order to shift computational requirements from A to B. The only change required in Protocol 7.8 is to replace the first message with that of Protocol 7.4 and adjust the corresponding computation by B.

Recently MacKenzie [218] proposed a generalisation of PAK-Y, called PAK-Z, which allows any digital signature to take the place of the Schnorr signature used in PAK-Y. The server B stores the public and private keys for the signature algorithm and sends the private signing key to A masked with the shared secret. Then A returns a signature on t_B as the third message. As well as allowing any signature to be used, PAK-Z uses exactly the same verifier k as used in Protocol 7.2 so that the result is a more modular approach to the set of PAK protocols.

Petitioner Samsung Ex-1036, 0283

7.3.1 B-SPEKE

Jablon [165] discussed the method used by Bellovin and Merritt to augment EKE and proposed an alternative. He pointed out that Bellovin and Merritt's technique could be applied generally and in particular to his own SPEKE protocol leading to a variant he called A-SPEKE. His own augmentation technique he called the 'B' extension.

Instead of employing a signature to prove knowledge of the password, as used in Bellovin and Merritt's 'A' extension, the 'B' variant uses an additional Diffie–Hellman exchange. The server B stores two password images as before: one is the square of the hash, $P = H(\pi)^2$, and the other is $V = g^{\pi}$. During the protocol B sends an additional challenge $g^{\tilde{r}_B}$ to A and A must respond by showing knowledge of the Diffie–Hellman key $\tilde{Z}_{AB} = g^{\pi\tilde{r}_B}$. A cannot simply send $\tilde{Z}_{AB}$ to B since this value could be used together with the challenge to form a dictionary attack on π. However, if A returns a hash containing both Z_{AB} and $\tilde{Z}_{AB}$, the hash can be used both to check knowledge of π and for confirmation of Z_{AB}. Employing this technique to the SPEKE protocol leads to B-SPEKE, Protocol 7.9.

Information held by A: Password π, interpreted as an element of $\mathbb{Z}_q$.
Information held by B: Derived password image $V = g^{\pi}$. Derived password image $P = H(\pi)^2$ where $H(\pi)$ is interpreted as an element of $\mathbb{Z}_p^*$.
Shared information: Subgroup G of $\mathbb{Z}_p^*$ of prime order $q = (p-1)/2$. Hash functions H, H_0, H_1, H_2. Security parameter L.

A		B
$P = H(\pi)^2$		
$r_A \in_R \{1, \ldots, 2^L\}$		
$t_A = P^{r_A}$	$\xrightarrow{t_A}$	$r_B, \tilde{r}_B \in_R \{1, \ldots, 2^L\}$
		$Z_{AB} = t_A^{r_B}$
		Check $Z_{AB} \notin \{-1, 0, 1\}$
		$t_B = P^{r_B}, \tilde{t}_B = g^{\tilde{r}_B}$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{t_B, V_B, \tilde{t}_B}$	$V_B = H_1(t_A, t_B, Z_{AB}, P)$
Check $Z_{AB} \notin \{-1, 0, 1\}$		$\tilde{Z}_{AB} = V^{\tilde{r}_B}$
Verify V_B		
$\tilde{Z}_{AB} = (\tilde{t}_B)^{\pi}$		
$V_A = H_2(t_A, t_B, Z_{AB}, \tilde{Z}_{AB}, P)$	$\xrightarrow{V_A}$	Verify V_A
$K_{AB} = H_0(Z_{AB})$		$K_{AB} = H_0(Z_{AB})$

Protocol 7.9: B-SPEKE protocol

Petitioner Samsung Ex-1036, 0284

Jablon is not explicit about the construction of the verifiers V_A and V_B in Protocol 7.9. We have adapted the verifier format used in Protocol 7.5. In any case, the proof of MacKenzie for Protocol 7.5 does not cover Protocol 7.9. Jablon [165] provides a number of variants of Protocol 7.9 to optimise various features; for example, he presents a version designed to minimise the total time by allowing various calculations to take place in parallel.

7.3.2 SRP Protocol

The SRP (Secure Remote Password) protocol was designed by Wu [332] as a more efficient alternative to the original augmented EKE. The server B holds a salt value s and the password image $V = g^{H(s,\pi)}$. Protocol 7.10 shows an optimised version of SRP, minimising the number of messages.

Shared information: Subgroup G of $\mathbb{Z}_p^*$ of prime order $q = (p-1)/2$. Hash function H.
Information held by A: Password π.
Information held by B: Password image $V = g^{H(s,\pi)}$. Salt value s.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{A, t_A}$	$u, r_B \in_R \mathbb{Z}_q$
		$Z_{AB} = (t_A V^u)^{r_B}$
		$K_{AB} = H(Z_{AB})$
$V = g^{H(s,\pi)}$	$\xleftarrow{S, s, u}$	$S = V + g^{r_B}$
$Z_{AB} = (S - V)^{r_A + uH(s,\pi)}$		
$K_{AB} = H(Z_{AB})$		
$M_1 = H(A, B, K_{AB})$	$\xrightarrow{M_1}$	Verify M_1
		$M_2 = H(A, M_1, K_{AB})$
Verify M_2	$\xleftarrow{M_2}$	

Protocol 7.10: SRP protocol

The shared secret is $Z_{AB} = g^{r_A r_B} \cdot V^{u r_B}$ which is unusual in that it is asymmetric with regard to the inputs of A and B. Another unusual feature is the public random value u chosen by B and included in the derivation of Z_{AB}. The purpose of u is to ensure that A knows the value of π and not just its image V. If A could know u in advance then she could send $g^{r_A} V^{-u}$ in the first message instead of t_A and then B would calculate $Z_{AB} = g^{r_A r_B}$. This means that A could complete the protocol only with knowledge of V. In order

Petitioner Samsung Ex-1036, 0285

to avoid this possibility it is evident that u cannot be a fixed value. However, for efficiency reasons, Wu suggests that u may be defined as a public function of S which would allow it to be omitted from the exchanged messages.

The symmetric encryption algorithm used with the password is addition modulo p. Forward secrecy is provided since the ephemeral Diffie–Hellman key, $g^{r_A r_B}$, is needed in order to find Z_{AB}. The main drawback of SRP is a lack of a formal security proof. However, Wu did point out that security against passive eavesdroppers is provided in the sense that an oracle that can find the session key from the exchanged messages t_A, S and u, is able to solve the Diffie–Hellman problem.

A minor weakness in Protocol 7.10, whose discovery is attributed by MacKenzie [217] to Bleichenbacher, allows the adversary to eliminate two passwords, π_1 and π_2, with each protocol run. An adversary masquerading as B can exploit the symmetry of the value S by choosing $V_1 = g^{H(s,\pi_1)}$ and $V_2 = g^{H(s,\pi_2)}$ and sending $S = V_1 + V_2$ instead of a correctly formed S in the second message. Then if π_1 is the correct password the adversary can check that $Z_{AB} = V_2^{r_A + uH(s,\pi_1)}$ while a symmetric test can be used for π_2.

The calculation of V must be explicitly made by A during the protocol, but Wu implies that $H(s,\pi)$ can be a 'tiny' exponent since the entropy of π is small. Wu regards as optional whether the values p and g are fixed or are sent as part of the protocol itself (the former situation is intended in Protocol 7.10). If p and g are sent as parameters by B at the start of the protocol it is essential that A checks the primality of $q = (p-1)/2$ and that g has order q.

Kwon and Song [198] have proposed a similar protocol to SRP in which the image of the password is similarly added to one of the ephemeral public keys in order to encrypt it.

7.3.3 AMP Protocol

The AMP (Authentication via Memorable Password) protocol of Kwon [195] is a variant of Diffie–Hellman-based EKE that employs a number of different features found in other protocols. Like the PAK family, Protocol 7.11 masks one of the Diffie–Hellman exchanges by multiplying it with a derived version of the password. However, there is a different method to calculate the shared secret which depends in an unusual way on the exchanged values. The shared secret is $Z_{AB} = g^{r_B(r_A+e)}$ where e is a hash of the exchanged values t_A and T together with the principal identities.

Protocol 7.11 lacks a formal proof of security. It turns out that, like Protocol 7.10, an adversary can eliminate two passwords in one protocol run[2]. To do this the adversary chooses $t_A = g^{-H_1(s,\pi_1)}$ in the first message where π_1 is the first password guess. If this guess is correct then the value T returned

[2] This attack was described by Michael Scott in a message on the IEEE P1363 mailing list in July 2001.

Petitioner Samsung Ex-1036, 0286

Shared information: Subgroup G of $\mathbb{Z}_p^*$ of prime order $q = (p-1)/2r$. Four hash functions H, H_1, H_2 and H_3.
Information held by A: Password π.
Information held by B: Password image $V = g^{H_1(s,\pi)}$. Salt value s.

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$		
	$\xrightarrow{A, t_A}$	$r_B \in_R \mathbb{Z}_q$ $T = (t_A V)^{r_B}$
	$\xleftarrow{s, T}$	
$x = H_1(s, \pi)$ $e = H_2(t_A, T, A, B)$ $Z_{AB} = T^{(r_A+e)/(r_A+x)}$ $K_{AB} = H_3(Z_{AB})$		$e = H_2(t_A, T, A, B)$ $Z_{AB} = (t_A g^e)^{r_B}$ $K_{AB} = H_3(Z_{AB})$
	$\xrightarrow{H(t_A, K_{AB})}$	Verify hash
Verify hash	$\xleftarrow{H(T, K_{AB})}$	

Protocol 7.11: AMP protocol

by B will be equal to 1. If the guess is incorrect then the adversary can still continue the protocol and include a different guess in the calculation of Z_{AB}.

The AMP protocol requires only two exponentiations for both client and server. On the client side this is two exponentiations fewer than PAK-Y, which has a formal proof of security. Both B-SPEKE and SRP also use two main exponentiations but have additional exponentiations with 'small' exponents. In contrast AMP requires an inversion modulo q. Notice that AMP can use a value of q of 160 bits so that exponents can be efficiently reduced before exponentiation. Forward secrecy is provided since the password does not influence the value of the session key.

7.4 Three-Party EKE

The protocols we have examined so far in this chapter are appropriate for the situation when a client wishes to connect securely to a server with whom the password (or an image of the password) is shared. We now consider some protocols designed to allow two users to establish a new key through the cooperation of a mutually trusted server with whom they both share a short key or password. This class of password-based protocols has been largely neglected recently, probably due to the requirement for the on-line server. However, as well as their historical interest, it seems worthwhile to consider such protocols

Petitioner Samsung Ex-1036, 0287

due to potential efficiency gains. An alternative to using a special-purpose protocol is to run two two-party protocols between the server and each of the two principals; the two secure channels established can be then used to distribute a session key.

In this section we examine three-party EKE-type protocols; in Sect. 7.6.1 below we consider three-party protocols in which the users have knowledge of the server's public key.

7.4.1 GLNS Secret Public Key Protocols

Gong, Lomas, Needham and Saltzer (GLNS) [130] (and the same authors earlier with names permuted [210]) published the first password-based protocols for a variety of authentication and key establishment scenarios. We call their set of proposals the GLNS protocols. The majority of their protocols assume that users have knowledge of the server public key – these protocols are examined in Sect. 7.6.1 below. Here we examine their protocols which use 'secret public keys'; these are ephemeral asymmetric keys which might normally be made public but, in order to prevent guessing attacks, are kept secret from eavesdroppers on the protocol. Gong *et al.* [130] provided variants of their protocols for both the three-party and the two-party situations.

The GLNS three-party secret public key protocol is shown in Protocol 7.12. Their two-party 'Direct Authentication Protocol' uses similar ideas by essentially combining the roles of A and S. In the initial two messages the server S sends encrypted ephemeral public keys K_S for A and K'_S for B. Subsequently these are used by A and B to encrypt the request for a session key in messages 3 and 4; in these messages $E_S(.)$ denotes encryption with K_S and $E'_S(.)$ denotes encryption with K'_S.

S has passwords π_A and π_B. S chooses random value n_S and ephemeral encryption keys K_S and K'_S.
A has password π_A. A chooses random values n_A, n'_A, c_A, r_A.
B has password π_B. B chooses random values n_B, n'_B, c_B, r_B.

1. $A \rightarrow S: A, B$
2. $S \rightarrow A: A, B, n_S, \{K_S\}_{\pi_A}, \{K'_S\}_{\pi_B}$
3. $A \rightarrow B: E_S(A, B, n_A, n'_A, c_A, \{n_S\}_{\pi_A}), n_S, r_A, \{K'_S\}_{\pi_B}$
4. $B \rightarrow S: E_S(A, B, n_A, n'_A, c_A, \{n_S\}_{\pi_A}), E'_S(B, A, n_B, n'_B, c_B, \{n_S\}_{\pi_B})$
5. $S \rightarrow B: \{n_A, K_{AB} \oplus n'_A\}_{\pi_A}, \{n_B, K_{AB} \oplus n'_B\}_{\pi_B}$
6. $B \rightarrow A: \{n_A, K_{AB} \oplus n'_A\}_{\pi_A}, \{H_1(r_A), r_B\}_{K_{AB}}$
7. $A \rightarrow B: \{H_2(r_B)\}_{K_{AB}}$

Protocol 7.12: GLNS secret public key protocol

Petitioner Samsung Ex-1036, 0288

Gong *et al.* remark on the similarity with the Otway–Rees protocol (3.22), at least in terms of the general structure. But there are a number of additional fields included in Protocol 7.12 which are worth examining. A (and B similarly) chooses two nonces n_A and n'_A which are transmitted to the server. The first of these is returned with K_{AB} with the usual purpose of allowing A to verify the freshness of this key. The second nonce is used to mask K_{AB}; its purpose is to prevent the adversary from performing a password guessing attack. If n'_A is omitted from message 5 then the adversary can decrypt the message with a candidate password $\tilde{\pi}_A$ and obtain a candidate session key $\tilde{K}_{AB}$. The adversary can then decrypt the second encrypted field in message 6 using $\tilde{K}_{AB}$ and check for the correct $H_1(r_A)$ value, in order to confirm whether $\tilde{\pi}_A$ is the correct password of A. The encrypted fields using K_{AB} in the final two messages are intended to provide key confirmation.

The value c_A (and c_B symmetrically) is a random 'confounder' used to ensure that encryption with K_S in message 4 is randomised. If c_A were not used, and the public key encryption were deterministic, then an adversary who obtained an old K_{AB} value could test a guess $\tilde{\pi}$ for π_A as follows. First the field $\{K_S\}_{\pi_A}$ in message 2 is decrypted with $\tilde{\pi}$ to obtain a possible public key $\tilde{K}_S$. Candidate values for n_A and n'_A can also be obtained by decrypting the first field in message 6 with $\tilde{\pi}$ (remember that we assume the adversary knows K_{AB}). Then all the required inputs are available to re-encrypt the first field in message 3, and check this against the actual value sent in order to verify whether $\tilde{\pi}$ is correct. The 'confounders' c_A and c_B can be omitted by requiring the asymmetric encryption algorithm to provide semantic security, since this ensures that the encryption is randomised. Furthermore, the asymmetric encryption algorithm must provide non-malleability; otherwise the protocol is easily defeated by an adversary C who can change the encrypted name of B to C in message 4 and hence masquerade as B to A.

Just as we have seen previously with Diffie–Hellman-based EKE, choice of the correct encryption algorithms is a crucial matter. Patel [262] demonstrates that use of RSA as the asymmetric encryption algorithm in Protocol 7.12 allows an active attack similar to the one against OKE in Sect. 7.5.2 below. However, Gong *et al.* did specify that the encryption algorithm must have the property that any random number could be a public key, a property not enjoyed by RSA. A protocol using an algorithm that matches the password representation to the encryption algorithm, such as used in PAK, appears more promising.

Although Protocol 7.12 seems to be regarded in the literature as a fundamentally sound one, we note that an attack is possible unless specific precautions are taken by users in an implementation. To perpetrate the attack the adversary masquerades as the server in order to obtain encrypted messages from A and/or B which can be used to verify password guesses. Specifically the adversary can guess values for π_A and π_B, generate new ephemeral keys K_S and K'_S, and use the guessed passwords to form a possibly correct message 2. This message can be sent back to A following a protocol initiation by A.

Petitioner Samsung Ex-1036, 0289

The adversary can collect the subsequent message 4 intended by B for the server. Now the adversary can detect if either of the password guesses is correct, since if so the identifiers A and B will be present in the messages when decrypted with the generated private keys. Further guesses can be verified or discarded in the same way. Notice that A must accept a random value for the field $\{K_S\}_{\pi_A}$ in message 2; otherwise an off-line guessing attack is possible.

If the above attack is to be prevented two precautions must be taken in any implementation. Firstly, failed protocol attempts must be logged by users and the password disabled after a small number of failures. Although such precautions are usually taken for failed logins at a server, it is less usual for this matter to be considered for users. Secondly, it must not be possible for the adversary to start multiple parallel runs of the protocol, since otherwise the correct passwords can be found before any protocol run has failed. It seems likely that in some applications users would be required to enter the password for each protocol run, but in others the user's computing device might cache the password and reuse it for multiple runs.

Tsudik and van Herreweghen [321], and later Gong [129], have proposed variants of Protocol 7.12 aimed at simplifying it and improving its efficiency. Protocol 7.13 shows the version of Tsudik and van Herreweghen which follows the same basic design but reduces the amount of material that needs to be encrypted with the ephemeral public keys. In distinction to Protocol 7.12 there is no attempt to provide key confirmation.

S has passwords π_A and π_B. S chooses ephemeral encryption keys K_S and K'_S.
A has password π_A. A chooses random value n_A.
B has password π_B. B chooses random value n_B.

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{K_S \oplus B\}_{\pi_A}, \{K'_S \oplus A\}_{\pi_B}$
3. $A \rightarrow B : A, E_S(n_A), \{K'_S \oplus A\}_{\pi_B}$
4. $B \rightarrow S : A, B, E'_S(n_B), E_S(n_A)$
5. $S \rightarrow B : n_B \oplus \{n_B \oplus K_{AB}\}_{\pi_B}, n_A \oplus \{n_A \oplus K_{AB}\}_{\pi_A}$
6. $B \rightarrow A : n_A \oplus \{n_A \oplus K_{AB}\}_{\pi_A}$

Protocol 7.13: Simplified GLNS secret public key protocol

Inclusion of the identities of A and B in the fields encrypted with π_A and π_B in message 2 is critical to the security of Protocol 7.13. This is because there is nothing else that allows the principals to know which party the session key is shared with. The nonces n_B and n_A, when XOR'd with the encrypted fields using π_B and π_A in message 5, act as 'confounders'. The outer XOR prevents B (and A similarly), having also obtained K_{AB}, from mounting a brute force guessing attack on π_A.

Petitioner Samsung Ex-1036, 0290

Ding and Horster [104] found an on-line attack on Protocol 7.13, one of several *undetectable* on-line attacks that they discovered. The important feature of such attacks is that an insider can perpetrate them in such a way that the server cannot detect that they have taken place and so cannot restrict the number of repetitions. Therefore the adversary may be able to conduct an on-line exhaustive key search. In Ding and Horster's attack on Protocol 7.13, the insider adversary B masquerades as A to initiate the protocol (the real A does not take part in the attack). The adversary collects the response from S in message 2 and makes a guess at π_A. This allows the adversary to forge a corresponding value for message 3 and hence send a trial value to S in message 4. On receipt of message 5 the adversary can verify whether the guess for π_A was correct since this means that the same K_{AB} value will be found. This attack is undetectable by S as long as there is no redundancy in the encrypted fields in message 4.

A closer look at the attack of Ding and Horster reveals that there may be no need for the adversary to be an insider in order to mount the attack. Instead the adversary can masquerade as both A and B by guessing both passwords π_A and π_B. After initiating the protocol by masquerading as A, the adversary finds candidate values for K_S and K'_S and uses these to construct message 4. On receipt of message 5 incorrect candidate passwords can be eliminated if the decrypted values of K_{AB} are different. In this version of the attack both passwords must be found and so the maximum number of trials required before success is the square of the number of passwords. Depending on the size of the password space this may still be a feasible attack.

In Gong's variant [129], the ephemeral secret public keys are generated by A and B instead of S and consequently the number of protocol messages is reduced to five. This variant is shown in Protocol 7.14, where K_A and K_B are the ephemeral public keys chosen by A and B respectively and $E_A(.)$ and $E_B(.)$ denote encryption with these keys. The confounders c_S and c'_S are chosen to prevent guessing of encrypted contents and, as elsewhere, can be omitted if a public key encryption algorithm with semantic security is used.

Ding and Horster [104] show that there is an undetectable on-line attack on Protocol 7.14 too. Again, the insider adversary B masquerades as A in a protocol run that looks normal to S. Specifically the adversary guesses π_A and chooses a value for K_A (as well as for K_B). On receipt of message 3 the adversary can decrypt both encrypted messages and see if they give the same value for K_{AB}: if so then the guess for π_A was probably correct. Notice that S must accept random values in message 2; otherwise an off-line guessing attack is possible. However, no explicit encryption algorithm for use with π_A and π_B was specified by Gong.

As with the insider attack on Protocol 7.13, this attack may be extended to an outsider attack by guessing both candidate passwords together. Indeed an outsider need guess only one of the passwords. For example, the adversary can guess π_A, choose a value for K_A, and send a message to B as if it were from A. On receipt of message 3, the adversary can detect if this guess is

Petitioner Samsung Ex-1036, 0291

S has passwords π_A and π_B. S chooses random values c_S and c'_S.
A has password π_A. A chooses random value n_A and ephemeral encryption key K_A.
B has password π_B. B chooses random value n_B and ephemeral encryption key K_B.

1. $A \rightarrow B : n_A, \{K_A\}_{\pi_A}$
2. $B \rightarrow S : n_A, \{K_A\}_{\pi_A}, n_B, \{K_B\}_{\pi_B}$
3. $S \rightarrow B : E_A(A, B, c_S, K_{AB}, \{n_A\}_{\pi_A}), E_B(B, A, c'_S, K_{AB}, \{n_B\}_{\pi_B})$
4. $B \rightarrow A : E_A(A, B, c_S, K_{AB}, \{n_A\}_{\pi_A}), \{n_A\}_{K_{AB}}, n_B$
5. $A \rightarrow B : \{n_B\}_{K_{AB}}$

Protocol 7.14: Optimal GLNS secret public key protocol

correct, since if so the identifiers A and B will be present in the encrypted field using K_A in this message, when decrypted with the generated private key. This attack is not detectable by the server S, but B will detect a failed password guess since the adversary is not able to find the correct K_{AB} value and so cannot form the correct message 5. Therefore similar remarks to those regarding the related attack on Protocol 7.12 apply.

7.4.2 Steiner, Tsudik and Waidner Three-Party EKE

Steiner *et al.* [307] proposed Protocol 7.15 as a direct generalisation of Diffie–Hellman-based EKE. They remark that, in distinction to typical three-party protocols, the server does not generate, and indeed cannot obtain, the session key. Each of A, B and S generates an ephemeral private key, r_A, r_B and r_S respectively, and the shared secret for A and B is $Z_{AB} = g^{r_A r_B r_S}$. The encrypted fields using K_{AB} in the final two messages are intended as 'authenticators' that can be used for key confirmation of Z_{AB}.

S has passwords π_A and π_B. S chooses random value r_S.
A has password π_A. A chooses random value r_A and calculates $t_A = g^{r_A}$.
B has password π_B. B chooses random value r_B and calculates $t_B = g^{r_B}$.

1. $A \rightarrow B : \{t_A \oplus B\}_{\pi_A}$
2. $B \rightarrow S : A, \{t_A \oplus B\}_{\pi_A}, \{t_B \oplus A\}_{\pi_B}$
3. $S \rightarrow B : t_A^{r_S}, t_B^{r_S}$
4. $B \rightarrow A : t_B^{r_S}, \{\text{Message } 1\}_{K_{AB}}$
5. $A \rightarrow B : \{\{\text{Message } 1\}_{K_{AB}}\}_{K_{AB}}$

Protocol 7.15: Steiner, Tsudik and Waidner three-party EKE

Petitioner Samsung Ex-1036, 0292

Protocol 7.15 provides forward secrecy against eavesdroppers. However, Ding and Horster [104] showed that it is vulnerable to on-line undetectable guessing as shown in Attack 7.1. The insider adversary C records an old protocol run with A and replays A's input in message 2. Only the interaction with S in messages 2 and 3 is relevant for the attack. By guessing π_A the adversary can find the corresponding value $\tilde{t}_A$ that A would have sent and use this value as C's input to message 2. On receipt of message 3, C can check if both returned values are the same in order to confirm a correct guess of π_A.

C has password π_C and recorded message $\{t_A \oplus C\}_{\pi_A}$. C guesses A's password is $\tilde{\pi}$ and chooses $\tilde{t}_A$ such that $\{t_A \oplus C\}_{\pi_A} = \{\tilde{t}_A \oplus C\}_{\tilde{\pi}}$.

2. $C \rightarrow S : A, \{t_A \oplus C\}_{\pi_A}, \{\tilde{t}_A \oplus A\}_{\pi_C}$
3. $S \rightarrow C : t_A^{r_S}, \tilde{t}_A^{r_S}$

C checks if $t_A^{r_S} = \tilde{t}_A^{r_S}$. If so $\tilde{\pi} = \pi_A$.

Attack 7.1: Ding and Horster's attack on Protocol 7.15

Lin *et al.* [207] also discovered an off-line guessing attack on Protocol 7.15, as shown in Attack 7.2. The adversary C masquerades as both A and S in order to find the password of B. The values X and Y are chosen randomly by C and are simply place-holders for missing values. The value r_C is chosen by C in place of $r_A r_S$ in a real protocol run. C knows that B will calculate $Z_{AB} = (g^{r_C})^{r_B} = t_B^{r_C}$ so messages 2 and 4 can be used to check a guess for π_B.

C chooses random values X, Y and r_C.

1. $C_A \rightarrow B : X$
2. $B \rightarrow C_S : A, X, \{t_B \oplus A\}_{\pi_B}$
3. $C_S \rightarrow B : g^{r_C}, Y$
4. $B \rightarrow C_A : Y, \{\text{Message 1}\}_{K_{AB}}$

C guesses $\pi_B = \tilde{\pi}$ and decrypts $\{t_B \oplus A\}_{\pi_B}$ with $\tilde{\pi}$ to obtain $\tilde{t}_B$ and calculates $\tilde{Z}_{AB} = \tilde{t}_B^{r_C}$. Message 4 is used to check if guess is correct.

Attack 7.2: Lin–Sun–Hwang attack on Protocol 7.15

Lin *et al.* [207] proposed an alternative version of Protocol 7.15 which requires the client to know the correct public key of the server. Recently Lin *et al.* [208] have proposed another variant which avoids the known attacks without the need for a server public key.

Petitioner Samsung Ex-1036, 0293

7.5 RSA-Based Protocols

Although most password-based protocols rely on the Diffie–Hellman key exchange, it is not surprising that the widespread popularity of the RSA algorithm [274] has been the basis of some alternatives. We examine such protocols in this section.

7.5.1 RSA-Based EKE

Bellovin and Merritt's original EKE protocol is applicable to a number of public key algorithms and they included a version for RSA in their paper [38]. The basic model for EKE dictates that A should choose an ephemeral RSA public key and send it to B encrypted with π. B will then choose the session key and return it encrypted with the RSA key and, optionally, with π.

Bellovin and Merritt recognise a number of potential problems with this approach and discuss some possible remedies. The main problem is how to make an RSA key appear 'random' in order to avoid a partition attack which might eliminate candidate passwords that do not provide a valid RSA key pair when applied to decrypt the first message. They point out that sending the RSA key pair encrypted with the password is dangerous since most integer values do not make a good RSA modulus. Therefore a partition attack is possible whereby the adversary decrypts the ciphertext with a candidate password and can eliminate this candidate if the plaintext modulus has small factors. This will allow an adversary to eliminate a large proportion of passwords.

In view of the above problem Bellovin and Merritt propose that only the RSA exponent e be encrypted with π, while the modulus n is sent as cleartext. They further suggested that, since e must always be odd in the RSA algorithm, either e or $e+1$ be sent randomly to ensure that almost all integers less than n could occur as the encrypted value (they suggest to make n the product of two safe primes[3] to enhance this property). The first two messages in the protocol are then as follows.

$$
\begin{array}{ll}
1. & A \rightarrow B : A, n, \{e\}_{\pi} \\
2. & B \rightarrow A : \{K_{AB}^{e} \bmod n\}_{\pi} \\
\ldots .
\end{array}
$$

Notice that an eavesdropper who obtains an old session key K'_{AB} can decrypt the first message with a candidate $\tilde{\pi}$ to obtain a candidate $\tilde{e}$ and then calculate $\{(K'_{AB})^{\tilde{e}} \bmod n\}_{\tilde{\pi}}$. If the RSA encryption is deterministic, this can be used to verify whether $\tilde{\pi} = \pi$ by comparing with the second message in the old protocol run which used K'_{AB}. Therefore it is important that randomness is included with the session key as part of the RSA encryption.

Another necessary measure to prevent partition attacks is to ensure that candidate-decrypted values in the second message are smaller than n; as with

[3] A prime p is a *safe prime* if $(p-1)/2$ is also prime.

Petitioner Samsung Ex-1036, 0294

Diffie–Hellman-based EKE this may be addressed by ensuring that the modulus is slightly less than a power of 2. However, Patel [262] showed that these precautions are still not sufficient to protect the protocol. The attack of Patel requires the adversary to replace the first message with A, n', X where X is random and n' is carefully chosen so that $n' = pq$ with $3|(p-1)$ and $3|(q-1)$. The choice of n' ensures that the mapping $x \mapsto x^3 \bmod n$ is a 9-to-1 mapping in $\mathbb{Z}^*_{n'}$. On receipt of this message B will 'decrypt' X with π and use the resulting value e' to encrypt the random value K_{AB}. There is a probability of $1/3$ that e' is divisible by 3. Although the adversary cannot know if this is the case, the attack can be repeated if it fails initially. Assuming it is the case, the adversary can partition the candidate passwords into those that decrypt the second message into a cubic residue and those that do not. The latter case occurs with probability about $8/9$ and these candidates can be discarded since the value K^e_{AB} is a cubic residue. Cubic residues can easily be recognised by the adversary by raising to the power $(p-1)/3 \bmod p$ and also raising to the power $(q-1)/3 \bmod q$ and checking that both results equal 1. By repeating the attack with the same first message the adversary can obtain a new reply and use it in the same way to discard a proportion $8/9$ of the remaining candidate passwords. The attack can easily be generalised to use any small value in place of 3 as the divisor. An attempt to avoid the attack by having the receiver check that the decrypted e has no small factors fails because the adversary can now discard passwords that result in such e values.

7.5.2 OKE and SNAPI

Although the attack of Patel seems fatal for RSA-based EKE, a new approach was taken by Lucks [215] to revive the protocol. The protocol of Lucks was called OKE (Open Key Exchange) which emphasises that the public ephemeral RSA key is sent 'in the open' (as plaintext). Lucks provides a proof in a Bellare–Rogaway-style model that OKE is secure in a general setting. However, there are problems in satisfying the assumptions of the proof in the RSA setting; the difficulties revolve around the ability of the adversary to select the public RSA key in such a way as to obtain information about the password, much the same as in Patel's attack outlined above. Indeed MacKenzie *et al.* [219] subsequently did find an attack on the RSA version of OKE along these lines. They also proposed a variant of RSA-based OKE called SNAPI (Secure Network Authentication with Password Information) which avoids the attack and is described in Protocol 7.16.

A critical factor in avoiding the attack on the OKE protocol is the choice of RSA public key. The RSA modulus N is chosen using a security parameter L_1 so that $2^{L_1-2} \leq N \leq 2^{L_1}$. The exponent e must be in the range $2^{L_1} \leq e \leq 2^{L_1+1}$. A consequence is that exponentiation with e is more than a full length exponentiation; with typical values e will be 1025 bits long.

An important check is that the parameter p must lie in the set $S_N = \{x : x \leq 2^\eta - (2^\eta \bmod N) \wedge (x, N) = 1\}$. Since p is an output of H which has

Petitioner Samsung Ex-1036, 0295

Shared information: Password π. Security parameters L_1 and L_2. A 'long' hash function H with output of length η with $\eta \geq L_1 + L_2$ and three 'short' hash functions H_1, H_2 and H_3 with outputs of length L_2. A set $S_N = \{x : x \leq 2^\eta - (2^\eta \bmod N) \wedge (x, N) = 1\}$.

A		B
$n_A \in_R \{0,1\}^{L_2}$		
Choose RSA public key N, e		
	$\xrightarrow{A, n_A, N, e}$	
		Check range of N, e
		Check e is prime
		Check length of n_A
		$n_B \in_R \{0,1\}^{L_2}$
		$p = H(N, e, n_A, n_B, A, B, \pi)$
		$a \in_R \mathbb{Z}_N^*$
		If $p \notin S_N$ then $q = a$
		else $q = pa^e \bmod N$
	$\xleftarrow{n_B, q}$	
Check length of n_B		
Check $(q, N) = 1$		
$p = H(N, e, n_A, n_B, A, B, \pi)$		
Check $p \in S_N$		
$a = (q/p)^d \bmod N$		
$r = H_1(N, e, n_A, n_B, A, B, q, a)$	$\xrightarrow{r}$	If $p \notin S_N$ then halt
		Verify r
		$t = H_2(N, e, n_A, n_B, A, B, q, a)$
		$K_{AB} = H_3(N, e, n_A, n_B, A, B, q, a)$
	$\xleftarrow{t}$	
Verify t		
$K_{AB} = H_3(N, e, n_A, n_B, A, B, q, a)$		

Protocol 7.16: SNAPI protocol

length at least $L_1 + L_2$ and N has length not more than L_1, this will fail with tiny probability if N is chosen honestly, but an adversary could choose N with small factors to make the probability high that $(x, N) > 1$. If the condition is not satisfied by the value of p generated then q is set to the random value a so that in any event q appears to A to be random.

MacKenzie *et al.* [219] provide a proof of security of SNAPI in Shoup's simulation model. As well as possessing a security proof, a potential benefit of SNAPI is that the ephemeral RSA key can be reused in several protocol runs. Since RSA key generation is a computationally costly process this is a useful advantage and means that SNAPI can be more efficient than PAK. A

Petitioner Samsung Ex-1036, 0296

drawback of this approach is that storing the ephemeral key makes it part of the long-term key for A and then compromise of the private key means that old session keys can be recovered for all sessions in which that RSA key was used: in other words, this computational advantage comes at the cost of a loss in forward secrecy. MacKenzie *et al.* [219] also proposed an extension to SNAPI, called SNAPI-X, which allows the server side to store only an image of π under a hash function, similar to protocols in Sect. 7.3.

Another variant of OKE was proposed by Roe *et al.* [276]. A novel feature of their protocol is that A sends only the RSA modulus N while the exponent e is defined as a deterministic function of π; this makes the protocol potentially more efficient than SNAPI (or original OKE). B returns $z^{e(\pi)} \bmod N$ where z is a random number of appropriate length used to define various values including the session key and a nonce n_A. On receipt of this value, A should calculate the decryption key $e(\pi)^{-1} \bmod \phi(N)$ and decrypt z as in normal RSA. Then A returns the value n_A to B. Unfortunately, Bleichenbacher [47] has shown that multiple passwords can be checked by B for each protocol run by sending $z^{e(\pi_1)e(\pi_2)\ldots e(\pi_n)}$ and using the returned n_A value to check which, if any, of the $e(\pi_i)$ values were removed by A.

7.6 Protocols Using a Server Public Key

Some of the earliest proposed password-based protocols assume that users not only share passwords with a server but are also in possession of, or at least can obtain, an authentic server public key. This is a significant additional assumption, but it may be realistic in certain applications; a user may trust a workstation to obtain the server's public key or there may be a means to allow the user to check the public key. Since the user's password must be entrusted to some computing device, it does not seem unreasonable to give such a device some trust in obtaining the server public key. Halevi and Krawczyk [139, 140] have proposed a method whereby users can verify the hash of the server public key using sequences of (English) words.

In contrast to EKE and its variants, password-based protocols employing a server key have no need to use the password as a key for encryption, but instead the password can be encrypted by the user with the server public key. A critical issue with this approach is whether the adversary is able to gain any useful information from the ciphertext containing the password. One immediate objection may be that the adversary can make trial encryptions of candidate passwords and check if the same ciphertext is obtained. Such a possibility illustrates that it is usually necessary for the encryption to be probabilistic so that each time a different ciphertext is obtained. We will make the requirement more precise later.

In this section we include both two-party and three-party protocols. Public key computations on the client side are typically restricted to a single encryption in this class of protocols. The two-party protocols tend to be more

Petitioner Samsung Ex-1036, 0297

efficient for the client than the two-party protocols using ephemeral public keys. However, in contrast to the EKE-based Diffie–Hellman protocols examined in Sects. 7.2 and 7.3, forward secrecy is often sacrificed.

7.6.1 GLNS Protocols with Server Public Keys

In addition to their 'secret public key' protocol examined in Sect. 7.4.1, Gong *et al.* [130] published several protocols assuming server public keys are available to clients. We examine some of these and subsequent variants from other authors. Unfortunately none of these protocols carries a security proof.

Protocol 7.17 shows the 'compact' version of the GLNS protocol for establishing a new session key between A and B who initially share passwords π_A and π_B with the server S. There is a strong similarity with messages 3 to 7 of Protocol 7.12.

S has passwords π_A and π_B.
A has password π_A. A chooses random values n_A, n'_A, c_A, r_A and timestamp T_A.
B has password π_B. B chooses random values n_B, n'_B, c_B, r_B and timestamp T_B.

1. $A \rightarrow B : E_S(A, B, n_A, n'_A, c_A, \{T_A\}_{\pi_A}), r_A$
2. $B \rightarrow S : E_S(A, B, n_A, n'_A, c_A, \{T_A\}_{\pi_A}), E_S(B, A, n_B, n'_B, c_B, \{T_B\}_{\pi_B})$
3. $S \rightarrow B : \{n_A, K_{AB} \oplus n'_A\}_{\pi_A}, \{n_B, K_{AB} \oplus n'_B\}_{\pi_B}$
4. $B \rightarrow A : \{n_A, K_{AB} \oplus n'_A\}_{\pi_A}, \{H_1(r_A), r_B\}_{K_{AB}}$
5. $A \rightarrow B : \{H_2(r_B)\}_{K_{AB}}$

Protocol 7.17: GLNS compact protocol

As in Protocol 7.12, the 'confounders' c_A and c_B may be omitted by requiring that the asymmetric encryption algorithm provides semantic security. Also the asymmetric encryption algorithm must provide non-malleability. If the long-term decryption key of S is compromised then n_A and n'_A can be found, allowing a brute force search for π_A, and consequently revealing the session key. Therefore we conclude that forward secrecy is not provided.

The encrypted timestamp, $\{T_A\}_{\pi_A}$, included in the first message, is used by S to ensure that the message is freshly generated by A. Without this timestamp the adversary could replay message 1 and obtain two messages $\{n_A, K_{AB} \oplus n'_A\}_{\pi_A}$ and $\{n_A, K'_{AB} \oplus n'_A\}_{\pi_A}$, the only difference being in the encrypted session keys. Then the adversary could again mount a brute force attack on π_A since a correct guess can be identified when the first components of the two decrypted messages are identical. (This attack was observed by Tsudik and van Herreweghen [321] as well as by Gong *et al.* [130].) The use of timestamps requires the server to record all messages received for a period of the maximum time window allowed for clock differences and message delay.

Petitioner Samsung Ex-1036, 0298

In order to remove this drawback Gong *et al.* also provided a version of the protocol in which the server generates a nonce that must be sent with the client requests; its drawback in turn is the addition of two messages to the protocol. Tsudik and van Herreweghen [321], and later Gong [129], have proposed a variant of this nonce-based protocol, aimed at simplifying it and reducing computational requirements. Protocol 7.18 shows Gong's optimal version, in which the number of messages exchanged is reduced to five. Gong [129] observes that this is the same number of messages used in the timestamp-based version.

S has passwords π_A and π_B.
A has password π_A. A chooses random values $n_A, n'_A, n''_A, c_A, r_A$.
B has password π_B. B chooses random values $n_B, n'_B, n''_B, c_B, r_B$.

1. $A \rightarrow B : E_S(A, B, n_A, n'_A, c_A, n''_A, \{n''_A\}_{\pi_A}), r_A$
2. $B \rightarrow S : E_S(A, B, n_A, n'_A, c_A, n''_A, \{n''_A\}_{\pi_A}),$
   $E_S(B, A, n_B, n'_B, c_B, n''_B, \{n''_B\}_{\pi_B})$
3. $S \rightarrow B : \{n_A, K_{AB} \oplus n'_A\}_{n''_A}, \{n_B, K_{AB} \oplus n'_B\}_{n''_B}$
4. $B \rightarrow A : \{n_A, K_{AB} \oplus n'_A\}_{n''_A}, \{H_1(r_A), r_B\}_{K_{AB}}$
5. $A \rightarrow B : \{H_2(r_B)\}_{K_{AB}}$

Protocol 7.18: Optimal GLNS nonce-based protocol

The main difference between the nonce-based and timestamp-based protocols is that in the former A and B send a third nonce to S, which is used both to authenticate them to S and also as a shared secret to encrypt the session key in the reply from S.

7.6.2 Kwon–Song Protocols

Kwon and Song [196, 197] proposed a set of password-based protocols. Protocol 7.19 is their basic protocol. The session key is generated by B and masked using nonces sent by A in the first message. The protocol is simple and the computation required for both parties is small. A drawback is that this protocol does not provide forward secrecy.

The final message in Protocol 7.19 prevents a replay attack. An adversary who obtains an old K_{AB} value can replay the first message and obtain the new session key since B will reuse the same r_A and r'_A values. However, the adversary cannot complete the protocol since r_A and r'_A are still unknown so the correct final message cannot be formed.

Kwon and Song provide a GNY logic analysis of their basic protocol. However, there is no specification of the requirements for the public key encryption algorithm. There does not seem to be any specific requirement for

Petitioner Samsung Ex-1036, 0299

Shared information: Hash function H. Password π. Authentic public key of B.

A		B
Choose random r_A, r'_A		
	$\xrightarrow{E_B(r_A, r'_A, \pi \oplus r'_A)}$	Decrypt and verify
		Choose K_{AB}
		$C = K_{AB} \oplus r_A \oplus r'_A$
Decrypt C and verify hash	$\xleftarrow{C, H(\pi \oplus r_A, K_{AB}, r'_A)}$	
	$\xrightarrow{H(\pi \oplus r'_A, K_{AB}, r_A)}$	Verify hash

Protocol 7.19: Kwon–Song basic protocol

non-malleability since an adversary should not be able to obtain any of the encrypted values, even if an old session key is obtained.

Kwon and Song [196, 197] also provide some variant protocols. Their 'Challenger's Public Key Protocol' requires the responder to know the authentic public key of the initiator but includes an unspecified symmetric encryption algorithm using the password. They also present two key agreement protocols, the first of which does not provide forward secrecy while the second is essentially the same as Diffie–Hellman-based EKE.

7.6.3 Halevi–Krawczyk Protocols

Halevi and Krawczyk [139, 140] designed password-based protocols in a modular way and conducted a formal analysis. They provided a formal proof of security for the challenge–response protocol that forms the basis of their other protocols, but only an outline argument of how the proof can be extended to key establishment. Boyarsky [53] argued that the protocols in the original paper [139] cannot be secure without a stronger assumption on the asymmetric encryption algorithm used unless only one user is involved. A stronger assumption is used in the later paper of Halevi and Krawczyk [140] but still not as strong as suggested by Boyarsky. However, all parties agree that a cryptosystem that provides non-malleability is sufficient for security.

In all their protocols Halevi and Krawczyk assume that the user A already has possession of a 'public password' which is a hashed version of the public key of the server B. This value can be used to verify the server's full public key which is sent to the user as part of the protocol. We omit this in our descriptions. Protocols for entity authentication only are proposed in addition to protocols for key exchange with and without forward secrecy.

The key establishment protocols use a similar design to the SKEME protocols defined in Protocols 5.21 and 5.38 in Chap. 5. We present the version

Petitioner Samsung Ex-1036, 0300

without forward secrecy in Protocol 7.20. In order to preserve our usual convention that A is the client and B the server, the protocol is shown with the first message originating with B. In practice A would often need to start the protocol with a request to B to connect, which would precede the protocol as shown.

Shared information: Password π. Security parameter L.

A		B
		$r_B \in_R \{0,1\}^L$
Choose random K_0	$\xleftarrow{r_B}$	
	$\xrightarrow{E_B(K_0, \pi, r_B, A, B)}$	Decrypt and verify
		$y = MAC_{K_0}(r_B, B, A)$
Verify y	$\xleftarrow{y}$	$K_{AB} = MAC_{K_0}(y)$
$K_{AB} = MAC_{K_0}(y)$		

Protocol 7.20: Halevi–Krawczyk password-based protocol

Protocol 7.20 is a simple version of the protocol of Halevi and Krawczyk. More generally the encrypted message sent from A to B can be of the form $E_B(K_0, f_\pi(r_B, K_0, A, B))$ for a function $f_\pi(X)$ used to combine the password with the other protocol fields requiring verification. It is required that f is one-to-one (injective) when either π or X is fixed. Therefore a suitable instantiation of $f_\pi(X)$ is the concatenation of π and X as used in Protocol 7.20. Note that although K_0 is transported from A to B, the session key K_{AB} is formed by key agreement since it includes inputs from both parties.

The properties and computational requirements are similar to Protocol 7.19 but a difference is that in Protocol 7.20 the server and client both contribute to the session key, whereas in Protocol 7.19 it is the server that generates the key. Like the SKEME variant Protocol 5.38, Protocol 7.20 does not provide forward secrecy. Another protocol with the same basic design as Protocol 7.20, but incorporating a Diffie–Hellman exchange, was proposed by Halevi and Krawczyk to provide this property. As usual there is a computational cost in so doing, which amounts to two extra exponentiations per user.

7.6.4 Three-Party Protocol of Yen and Liu

The protocols of Yen and Liu [339] use ideas from the simplified GLNS three-party EKE of Tsudik and van Herreweghen (Protocol 7.13). By making use of a server public key they avoid the need for ephemeral public keys. Protocol

Petitioner Samsung Ex-1036, 0301

7.21 shows their main protocol, in which the server generates the session key K_{AB}. They also propose variants in which either the initiator or responder can generate K_{AB}.

S has passwords π_A and π_B. S chooses K_{AB}.
A has password π_A. A chooses random values n_A, n'_A.
B has password π_B. B chooses random value n_B.

1. $A \rightarrow B : A, n'_A, E_S(n_A \oplus \pi_A, n_A \oplus B \oplus n'_A)$
2. $B \rightarrow S : A, B, E_S(n_A \oplus \pi_A, n_A \oplus B \oplus n'_A), E_S(n_B \oplus \pi_B, n_B \oplus A \oplus n'_A)$
3. $S \rightarrow A : n_A \oplus \{n_A \oplus K_{AB}\}_{\pi_A}, \{n'_A\}_{K_{AB}}, n_B \oplus \{n_B \oplus K_{AB}\}_{\pi_B}$
4. $A \rightarrow B : n_B \oplus \{n_B \oplus K_{AB}\}_{\pi_B}, \{n'_A + 1\}_{K_{AB}}$

Protocol 7.21: Yen–Liu protocol

On receipt of message 2, S decrypts both ciphertexts to obtain two pairs of values, say X_A, Y_A and X_B, Y_B. Then S sets $n_A = X_A \oplus \pi_A$ and $n_B = X_B \oplus \pi_B$ and checks that $n_A \oplus Y_A \oplus B = n_B \oplus Y_B \oplus A$. If not, then S will abort the protocol. Despite a detailed analysis by its authors, the Yen–Liu protocol does not provide authentication of both parties. An insider adversary C is able to masquerade as B and successfully complete a protocol run with A, including obtaining the new session key.

---

1. $A \rightarrow C_B : A, n'_A, E_S(n_A \oplus \pi_A, n_A \oplus B \oplus n'_A)$
2. $C \rightarrow S : \quad A, C, E_S(n_A \oplus \pi_A, n_A \oplus B \oplus n'_A), E_S(n_C \oplus \pi_C, n_C \oplus A \oplus n''_A)$
3. $S \rightarrow C_A : n_A \oplus \{n_A \oplus K_{AC}\}_{\pi_A}, \{n''_A\}_{K_{AC}}, n_C \oplus \{n_C \oplus K_{AC}\}_{\pi_C}$

3′. $C_S \rightarrow A : n_A \oplus \{n_A \oplus K_{AC}\}_{\pi_A}, \{n'_A\}_{K_{AC}}, n_C \oplus \{n_C \oplus K_{AC}\}_{\pi_C}$

4. $A \rightarrow C_B : n_C \oplus \{n_C \oplus K_{AC}\}_{\pi_C}, \{n'_A + 1\}_{K_{AC}}$

Attack 7.3: Attack on Protocol 7.21

In Attack 7.3, A wishes to complete the protocol with B, but in fact completes it with the adversary C. After receiving message 1 from A, C generates a new value n''_A such that $n''_A \oplus C = n'_A \oplus B$. This enables C to send a correct message 2 to S as if A is intending to run the protocol with C. C needs to intercept message 3 from S in order to replace the field $\{n''_A\}_{K_{AC}}$ with the field $\{n'_A\}_{K_{AC}}$ expected by A. This can be done since C is able to extract K_{AC} from message 3. When A receives the altered message 3′ the session key K_{AC} will be extracted by A and used to confirm that the value n'_A was correctly received. Thus A will accept the key as shared with B whereas it is actually shared with C.

Petitioner Samsung Ex-1036, 0302

7.7 Other Protocols

This section contains some protocols that do not fit into the pattern of the main trends of protocols.

7.7.1 Lee–Sohn–Yang–Won Protocol

The protocol of Lee *et al.* [202] is based on the Nyberg–Rueppel protocol which was examined in Chap. 5 (Protocol 5.3). Protocol 7.22 consists of basic Diffie–Hellman key agreement with a form of signature using the password as key; an image of the password (corresponding to a public key in the Nyberg–Rueppel protocol) is used for verification by the server. In addition, the exchanged hashes are intended to provide key confirmation.

Shared information: Hash functions H_1 and H_2. Element g generating group G of order q.
Information held by A: Password π.
Information held by B: Password image $P = g^{-H_2(\pi)}$.

A		B
$r_A, s_A \in_R \mathbb{Z}_q$ $e = g^{r_A - s_A}$		
$f = s_A + eH_2(\pi) \bmod q$	$\xrightarrow{A, e, f}$	$t_A = P^e g^f e$ $r_B \in_R \mathbb{Z}_q$ $Z_{AB} = t_A^{r_B}$ $t_B = g^{r_B}$
$Z_{AB} = t_B^{r_A}$ Verify hash	$\xleftarrow{t_B, H_1(Z_{AB}, e, f)}$	
	$\xrightarrow{H_1(Z_{AB}, t_B)}$	Verify hash

Protocol 7.22: Lee–Sohn–Yang–Won protocol

Unfortunately this protocol can be broken by an active adversary. The adversary C chooses r_C randomly, calculates $e = g^{r_C}$ and chooses f at random. The adversary C masquerades as A and sends A, e, f as the first message. Now C knows that B will calculate the shared secret as:

$$Z_{AB} = (P^e g^{f+r_C})^{r_B} = t_B^{-H_2(\pi)e+f+r_C}$$

and the only element unknown to C here is π. Therefore, on receipt of $t_B, H_1(Z_{AB}, e, f)$ from B, the adversary can guess the password as $\tilde{\pi}$ and

Petitioner Samsung Ex-1036, 0303

check if this is the correct π by calculating the corresponding Z_{AB} value and checking against the hash. This process allows C to engage in repeated password guessing until the correct password of A is found.

7.7.2 Anderson–Lomas Protocol

We will now describe an ingenious proposal of Anderson and Lomas [13]. Although it does not turn out as effective as many of the alternative protocols it is worth examining because of its originality. The novelty is the use of a *collisionful* hash function. In contrast to the usual requirement for hash functions these are functions for which it is *easy* to find many collisions for any input. Such functions are described by Gong [128] and we adopt his definition here.

Definition 7.1. *A* collisionful hash function *is a function q that maps a key k and a bit-string x and satisfies:*

- *Given k and x it is easy to compute $q_k(x)$.*
- *Given k and $q_k(x)$ it is hard to find a value y such that $q_k(x) = q_k(y)$ but $x \neq y$.*
- *Given pairs of values x and $q_k(x)$ it is hard to compute k, though it is less hard to find k' such that $q_k(x) = q_{k'}(x)$ but $k \neq k'$.*

The protocol of Anderson and Lomas uses a specific collisionful function:

$$q_k(x) = H(MAC_k(x) \bmod 2^m, x)$$

where m is a parameter and H is a one-way hash function. Since only m bits of the function depend on k it should be easy to find many k values that give the same output when m is small. Anderson and Lomas suggest to take $m = n/2$ where 2^n is the size of the password space. The idea of the protocol is then to use π as the key for q and so a guess for π can only be verified with probability $2^{-n/2}$.

A and B share password π.
A chooses random value r_A and sets $t_A = g^{r_A}$.
B chooses random value r_B and sets $t_B = g^{r_B}$.

1. $A \rightarrow B : t_A$
2. $B \rightarrow A : t_B$
3. $A \rightarrow B : q_\pi(Z_{AB})$
4. $B \rightarrow A : q_{H(\pi)}(Z_{AB})$

Protocol 7.23: Anderson–Lomas protocol

Petitioner Samsung Ex-1036, 0304

Protocol 7.23 shows the Anderson–Lomas protocol which is simply the basic Diffie–Hellman exchange followed by an exchange of authenticating values using function q. The shared secret Z_{AB} is the Diffie–Hellman ephemeral secret, $g^{r_A r_B}$. An adversary can try to mount an active attack by posing as B and attempting to guess the password from message 3; this will succeed with probability only $2^{-n/2}$.

An active adversary can reduce the number of possible passwords to the square root of the initial number. Thus the protocol leaks much more information than is desirable. Anderson and Lomas state that if A detects a failed login then she must change her password, which is somewhat inconvenient. Bakhtiari *et al.* [23] have pointed out that if Z_{AB} becomes known to the adversary then an off-line guessing attack becomes possible.

7.7.3 Strengthening Passwords

We mention here some interesting ideas that are aimed at 'strengthening' passwords. These techniques may be regarded as alternatives to using password-based authentication since they allow a password known by a user to be converted into a better quality shared secret. Each of these proposals, however, seems to have some significant limitations if a high level of security is required.

- Abadi *et al.* [3] suggested an ingenious scheme for randomising a password π shared between a user and server. At authentication time the user chooses a random value ρ of, say, 20 bits and sends the value $H(\pi, \rho)$, using a one-way hash function H. The server then tries all 2^{20} possible ρ values together with the known value of π and when the correct one is found the pair (π, ρ) may be used as a longer shared secret to form a session key. The adversary who eavesdrops and attempts a dictionary attack on the combination of (π, ρ) finds that the effort is 2^{20} times as hard as just to guess π. Although the extra computational effort required by the server may be acceptable in some scenarios, it places a limit on the level of security that can be obtained by this method.
- Perlman and Kaufman [268] explored 'bootstrapping' of a public key infrastructure on a workstation with no stored user information. The user's public and private keys, as well as any other necessary information, are stored on a server together with the user's password. By setting up a secure session with the server, this information can be safely downloaded from the server.
- Ford and Kaliski [113] proposed a method of *password hardening* which allows a user A to obtain a strong secret from her password π in cooperation with a server S. The server does not know the user's password but must have a user-specific exponent d_A which is used in a multiplicative group. (We assume here that π is an element in the group and is of high order, but in practice a suitable mapping from the string representing π

Petitioner Samsung Ex-1036, 0305

into the group may be necessary.) Each time A wishes to recover her hardened password she generates a random k and sends π^k to S. The server S returns π^{kd_A} and then A removes k by calculating $(\pi^{kd_A})^{k^{-1}}$ to obtain the strengthened password π^{d_A}. Notice that A recovers the same long secret each time. Ford and Kaliski discuss how to use this technique with multiple password servers to provide a secure password-based authentication service. One useful feature is that compromise of a subset of these servers does not compromise the password. Jablon [166] pointed out that an authenticated channel is required between the user and servers to prevent the adversary mounting a password guessing attack. He proposes an enhanced protocol to deal with this problem.

7.8 Conclusion

Password-based protocols allow users to establish a strong shared session key with other principals using no secret other than a short string that can be committed to human memory. Considering their more stringent requirements, it may be expected that such protocols are harder to design than authentication and key establishment protocols with full-length keys. However, understanding of password-based protocols has advanced rapidly in little more than 10 years since it was first realised that they are possible at all. Today there are several examples that have security assurance and practical performance similar to what can be achieved for protocols using full-length keys.

Some of the properties of the most prominent protocols examined in this chapter are summarised in Table 7.2. Notice that it is possible to run the SPEKE protocol without authenticators so that it uses only two messages like PPK. It is worth repeating that all the security proofs for protocols indicated in Table 7.2 rely on the random oracle model, except for the Katz–Ostrovsky–Yung (KOY) protocol.

The number of messages is one measure of the communications efficiency that is easy to compare in the different protocols. The computational efficiency is much harder to compare because most authors are not explicit about the size of parameters to be used. Furthermore, when security is based on different computational problems the relationship between security and parameter size can be quite different too.

The public key operations dominate the other computations in all the protocols in this chapter and so we only count these. Table 7.3 attempts to compare the performance of the class of EKE and augmented EKE protocols from the client side. It is unfortunate that in many protocols the client side, which may be much more limited in computational ability, has higher computational requirements than the server side. This is because the client usually needs to calculate an image of the password, whereas this can be stored directly by the server. An exception to this rule is PAK-R which can also be combined with other PAK variants.

Petitioner Samsung Ex-1036, 0306

Table 7.2. Properties of password-based protocols

	No. of messages	Security proof	Server holds image of π	Server PK required
DH-EKE (7.1)	4	No	No	No
PAK (7.2)	3	Yes	No	No
PPK (7.3)	2	Yes	No	No
PAK-R (7.4)	3	Yes	No	No
SPEKE (7.5)	3	Yes	No	No
KOY (7.6)	3	Yes	No	No
Augmented EKE (7.7)	5	Broken	Yes	No
PAK-Y (7.8)	3	Yes	Yes	No
B-SPEKE (7.9)	3	No	Yes	No
SRP (7.10)	4	No	Yes	No
AMP (7.11)	4	No	Yes	No
OKE	4	Broken	No	No
SNAPI (7.16)	4	Yes	No	No
SNAPI-X	4	Yes	Yes	No
GLNS (7.17)	5	No	No	Yes
Halevi–Krawczyk (7.20)	3	Partial	No	Yes

In Table 7.3 we assume that modulus sizes are chosen at 1024 bits for either Diffie–Hellman or RSA protocols. For the Diffie–Hellman-based protocols the basic requirements for each principal are two exponentiations: one to calculate t_A and the other to calculate Z_{AB}. Although this minimum is apparently achieved in the original Diffie–Hellman EKE its vulnerabilities make it a dubious choice today. For B-SPEKE we have assumed that the password size is 32 bits. Also for SRP we have assumed that the small hash $H(s, \pi)$ yields a 32-bit value. The requirements for each protocol are estimated by counting the number of public key operations; it must be appreciated that this is only an indication of the relative computational effort which depends on several detailed factors. For the Diffie–Hellman-based protocols the public key operations are taken as the exponentiations required for each side. It is perhaps remarkable that today it seems that the best password-based protocols enjoy most of the good properties that we expect of protocols using long secrets.

When selecting a protocol for use, undoubtedly the most important factor is its security. Since there are now a number of protocols with proven security, and especially bearing in mind the many subtle attacks found on earlier protocols, it seems prudent to use one of these. Examination of Tables 7.2 and 7.3

Petitioner Samsung Ex-1036, 0307

Table 7.3. Client side computation in password-based protocols

	No. of exponentiations of this size			
	32 bits	160 bits	864 bits	1024 bits
EKE				
DH-EKE		2		
PAK		2	1	
PPK		2	2	
PAK-R		3		
SPEKE		2		
SNAPI				1†
Augmented EKE				
Augmented EKE		2	2	
PAK-Y		3	1	
B-SPEKE	1	2		
SRP	2	2		
AMP		2		

† With additional checks an exponent as small as 512 bits is possible [219].

reveals that for EKE using Diffie–Hellman or RSA we can select a protocol with a formal proof of security with little sacrifice in performance. However, for the class of augmented EKE protocols there is currently a significant gap between the most efficient protocols with a formal security proof and those without one.

Of those protocols based on discrete logarithms that have a security proof, PAK and SPEKE are the most efficient. Options to exclude explicit authentication and to store only an image of the password at the server (PAK-Y and B-SPEKE) make these flexible choices too. The client side computation can be reduced in PAK-Y by the same trick as PAK-R; this replaces the 864-bit exponentiation with an additional 160-bit exponentiation. The only reasonable choice for RSA-based protocols is SNAPI. Although SNAPI can be more efficient than PAK if a fixed RSA modulus is used, the loss of forward secrecy may be a significant price to pay in many applications, while the increased number of messages may be an important drawback too. The emerging IEEE P1363.2 standard [155] includes detailed specifications of all the protocols in Table 7.3 in various different algebraic settings.

The GLNS protocols provide a means for two users to establish a session key with each other when they both share only weak keys with a server. These protocols, together with Protocol 7.15 and its variants, seem to be the only

Petitioner Samsung Ex-1036, 0308

ones that have been published for this purpose. For client–server protocols where the user possesses (or can obtain) an authentic copy of the server's public key, the protocols of Halevi and Krawczyk are efficient and enjoy a proof of security as long as Boyarsky's advice, to use a public key encryption algorithm secure against adaptive chosen ciphertext attacks, is followed.

As in Chap. 5, we have described all the protocols in this chapter in the context of subgroups of $\mathbb{Z}_p^*$, but many of them can be generalised to a variety of different groups. Since these protocols may well be useful in applications employing mobile computing devices, the computational efficiency and storage gains in using elliptic curve groups can be very attractive. Although it is straightforward to generalise the protocol definitions to different groups, there may be undesirable consequences with respect to security. For example, consider Protocol 7.1 when the Diffie–Hellman exchange takes place in an elliptic curve group. Encryption of an elliptic curve point using a password is more tricky than encrypting an element of $\mathbb{Z}_p^*$ with its natural mapping to a bit-string. However, protocols where the symmetric encryption algorithm is matched to the group used, such as Protocol 7.2, do not seem to have this problem as long as a suitable means to map the password to an elliptic curve point is available. The emerging IEEE P1363.2 standard [155] has addressed this issue.

Petitioner Samsung Ex-1036, 0309

A

Standards for Authentication and Key Establishment

Practitioners often look to standards bodies to recommend techniques that can be used with the assurance of independent verification of correctness and suitability. Several standards exist covering protocols of the type we have examined in this book. This appendix lists the main relevant standards and briefly summarises their contents. In many cases specific protocols have been examined in the body of the book and we refer to these where appropriate.

Standards are issued by many different bodies, both national and international. We have included mainly international standards; many national standards bodies issue their own versions of international standards with little or no alterations. Because of their international influence we also mention some US national standards.

A.1 ISO Standards

The International Organisation for Standardisation (`http://www.iso.ch`), known as ISO, has published numerous standards on cryptographic mechanisms and protocols, mainly through its subcommittee number 27 which works on IT security techniques.

A.1.1 ISO/IEC 9798

ISO issued the five-part standard ISO/IEC 9798 on the topic of entity authentication.

Part 1: General (2nd edition 1997)
Part 2: Mechanisms using symmetric encipherment algorithms (2nd edition 1999)
Part 3: Mechanisms using digital signature techniques (2nd edition 1998)
Part 4: Mechanisms using a cryptographic check function (2nd edition 1999)
Part 5: Mechanisms using zero knowledge techniques (1999)

Petitioner Samsung Ex-1036, 0310

The protocols in Part 2 of the standard were examined in Sect. 3.2.3. Some of the protocols in Part 3 of the standard were examined in Sect. 4.2.1.

A.1.2 ISO/IEC 11770

ISO issued the three-part standard ISO/IEC 11770 on the topic of key management. Some of the protocols in Parts 2 and 3 of the standard are strongly related to protocols in Parts 2 and 3 of ISO/IEC 9798.

Part 1: Framework (1996)
Part 2: Mechanisms using symmetric techniques (1996)
Part 3: Mechanisms using asymmetric techniques (1999)

The authentication and key establishment protocols in Part 2 of the standard were examined in Sects. 3.3.4 (the server-less protocols) and 3.4.4 (the server-based protocols). The key transport protocols in Part 3 of the standard were examined in Sect. 4.3.1. The key agreement protocols in Part 3 of the standard were summarised in Sect. 5.8.

A related standard, published in 2002, is ISO/IEC 15946 Part 3, which covers key establishment based on elliptic curves. This includes elliptic curve key agreement and key transport mechanisms that can be used with the protocols in Part 3 of ISO/IEC 11770. In addition, key agreement using the Unified Model and MQV on elliptic curves are included.

A.1.3 ISO 9594-8/ITU X.509

A series of standards for directory systems was first issued in 1988 jointly by ISO and CCITT (which was later re-formed as ITU). The section of the standard numbered 9594-8 (ISO version) or X.509 (ITU version) was known as the *Authentication Framework*. This section of the standard provides information on how to use a directory to store public key certificates, including the format of certificates. It also includes examples of how to use the certificates to provide authentication and key establishment. In the most recent version of the standard the *Authentication Framework* has been renamed as *Public-Key and Attribute Certificate Frameworks*.

Under the heading of *Strong Authentication* three key establishment protocols were presented. Unfortunately there were some problems with the protocols in the first version of the standard and they were subsequently updated. The protocols have been examined in Sect. 4.3.4.

A.2 Other Standards

A.2.1 IETF Standards

The Internet Engineering Task Force (IETF) (`http://www.ietf.org`) produces standards concerned with development of Internet technology. In contrast to many other standards bodies the IETF works in an open way and all

Petitioner Samsung Ex-1036, 0311

its documents are freely available on the Internet. Documents are published in a series known as RFCs (Requests for Comment) which include proposed and draft standards as well as full standards. Some RFCs are only intended for information.

Table A.1 summarises some prominent RFCs that cover protocols we have examined earlier with pointers to where they are described in this book. In each of these cases there are additional RFCs available which cover related information or extensions of these protocols for different applications.

Table A.1. Some RFCs for key establishment protocols

RFC	Year	Description	Status	Section
1510	1993	Kerberos version 5	Proposed standard	3.4.3
2246	1999	TLS protocol	Proposed standard	4.3.5
2409	1998	IKE protocol	Proposed standard	5.5.4
2412	1998	Oakley protocol	Informational	5.5.2

A.2.2 IEEE P1363-2000

The Institute of Electrical and Electronics Engineers is a US-based institution but has a worldwide membership. The IEEE Standards Association (`http://standards.ieee.org`) has issued standards in a wide range of electronics and communications areas. The IEEE Standard Specifications for Public-Key Cryptography were published in 2000 and are widely known as P1363-2000. The standard includes specifications for public key algorithms and key agreement protocols, and incorporates implementation details for elliptic curves as well as conventional groups for protocols based on discrete logarithms.

The only key establishment protocols included in P1363-2000 are basic Diffie–Hellman key agreement and authenticated versions using the Unified Model and MQV. These were examined in Sects. 5.4.3 and 5.4.4. Additional techniques are currently undergoing the standardisation process. In particular there is a draft document dedicated to password-based protocols which covers many of the protocols described in Chap. 7.

A.2.3 NIST and ANSI Standards

The US National Institute of Standards and Technology (NIST) (`http://www.nist.gov`) has issued a number of Federal Information Processing Standards (FIPS) covering cryptographic techniques. They have recently held two key management workshops and issued a white paper indicating the intention to

Petitioner Samsung Ex-1036, 0312

develop a standard in the coming years. This is intended to include the key agreement protocols in the P1363 standard as well as key transport and will align with related work in the American National Standards Institute (ANSI).

The earlier FIPS PUB 196 published in 1997 is entitled *Entity Authentication using Public Key Cryptography.* It includes two of the protocols contained in ISO/IEC 9798-3.

ANSI standards committee X9 (`http://www.x9.org`) provides standards for financial services industries. It has published numerous standards covering cryptographic algorithms and authentication mechanisms. Two recent standards, X9.42 and X9.63, are devoted to key agreement protocols.

- **X9.42 (2001) Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys using Discrete Logarithm Cryptography.** This covers key agreement for protocols based on conventional discrete logarithms. It includes Diffie–Hellman in static, ephemeral and hybrid (one-pass) versions, as well as the Unified Model and MQV in full and one-pass versions.
- **X9.63 (2002) Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport using Elliptic Curve Cryptography.** This includes elliptic curve versions of all but one of the protocols in X9.42. In addition it includes versions with key confirmation, an elliptic curve STS protocol (see Sect. 5.5.1) and two elliptic curve key transport protocols.

Petitioner Samsung Ex-1036, 0313

B

Summary of Notation

Notation is described in each chapter as it is introduced. In this appendix the main notational conventions are summarised.

A and B	Two users who wish to share a new session key
S	A trusted server
N_P	Random nonce value chosen by principal P
T_P	Timestamp chosen by principal P
K_{AB}	Key shared by A and B
C_P	Adversary C masquerading as principal P
$E_P(M)$	Public key encryption of message M with public key of principal P
$\mathrm{MAC}_K(M)$	Message authentication code of M using shared key K
$Sig_P(M)$	Digital signature with appendix of message M by principal P
$\{M\}_K$	Symmetric encryption of message M with shared key K to provide confidentiality and integrity
$[\![M]\!]_K$	Encryption of message M with key K to provide confidentiality
$[M]_K$	One-way transformation of message M with key K to provide integrity
p	A large prime (usually at least 1024 bits)
q	A prime (typically of 160 bits) with $q \mid p-1$
$\mathbb{Z}_p$	The field of integers (under addition and multiplication) modulo p
$\mathbb{Z}_p^*$	The multiplicative group of non-zero integers modulo p
G	A subgroup of $\mathbb{Z}_p^*$. Often a subgroup of order q, but sometimes equal to $\mathbb{Z}_p^*$
g	A generator of G
r_P	Random integer chosen by principal P
t_P	Ephemeral public keys: $t_P = g^{r_P}$
x_P	The private long-term key of principal P
y_P	The public key of principal P: $y_P = g^{x_P}$
Z_{AB}	The shared secret calculated by the principals A and B
S_{AB}	The static Diffie–Hellman key of P and Q: $S_{AB} = g^{x_A x_B}$

Petitioner Samsung Ex-1036, 0314

$H(.)$	A one-way hash function
$x \in_R X$	The element x is chosen uniformly at random from the set X
$F \stackrel{?}{=} G$	Verify that F and G evaluate to the same value
$\mathcal{U}$	The set of principals intended to share a conference session key
U_i	The i'th principal in $\mathcal{U}$, where $1 \leq i \leq m$
$\mathcal{Z}$	Group shared secret calculated by the principals
$\mathcal{K}$	Group shared session key
π	A key of short length, such as a password

Petitioner Samsung Ex-1036, 0315

References

1. Martín Abadi. Explicit communication revisited: Two new attacks on authentication protocols. *IEEE Transactions on Software Engineering*, 23(3):185–186, March 1997.
2. Martín Abadi. Two facets of authentication. In *11th IEEE Computer Security Foundations Workshop*, pages 27–32. IEEE Computer Society Press, 1998.
3. Martín Abadi, T. Mark A. Lomas, and Roger Needham. Strengthening passwords. Technical Report 1997-033, Digital Systems Research Center, Palo Alto, California, December 1997. `http://www.research.compaq.com/SRC/publications/src-tn.html`.
4. Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. In *IEEE Symposium on Research in Security and Privacy*, pages 122–136. IEEE Computer Society Press, 1994.
5. Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
6. Martín Abadi and Mark R. Tuttle. A semantics for a logic of authentication. In *10th ACM Symposium on Principles of Distributed Computing*, pages 201–216. ACM Press, 1991.
7. Michel Abdalla, Yuval Shavitt, and Avishai Wool. Key management for restricted multicast using broadcast encryption. *IEEE/ACM Transactions on Networking*, 8(4):443–454, August 2000.
8. G. Agnew, R. Mullin, and S. Vanstone. An interactive data exchange protocol based on discrete exponentiation. In C. G. Günther, editor, *Advances in Cryptology - Eurocrypt '88*, pages 159–166. Springer-Verlag, 1988. Lecture Notes in Computer Science Volume 330.
9. Sattam S. Al-Riyami and Kenneth G. Paterson. Authenticated three party key agreement protocols from pairings. Cryptology ePrint Archive, Report 2002/035, 2002. `http://eprint.iacr.org/2002/035/`.
10. N. Alexandris, M. Burmester, V. Chrissikopoulos, and Y. Desmedt. A proven secure public key distribution system. In *Proceedings of 3rd Symposium on State and Progress of Research in Cryptography*, Rome, February 1993.
11. Jim Alves-Foss. Provably insecure mutual authentication protocols: The two-party symmetric-encryption case. In *22nd National Information Systems Se-*

Petitioner Samsung Ex-1036, 0316

curity Conference, October 1999. Available at `http://csrc.nist.gov/nissc/1999/proceeding/papers/p25.pdf`.

12. Yair Amir, Yongdae Kim, Cristina Nita-Rotaru, and Gene Tsudik. On the performance of group key agreement protocols. Technical Report CNDS-2001-5, The Center for Networking and Distributed Systems (CNDS), Johns Hopkins University, November 2001. `http://www.cnds.jhu.edu/pub/papers/cnds-2001-5.ps`.
13. Ross Anderson and T. M. A. Lomas. Fortifying key negotiation schemes with poorly chosen passwords. *Electronics Letters*, 30(13):1040–1041, June 1994.
14. Ross Anderson and Roger Needham. Programming Satan's computer. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, pages 426–440. Springer-Verlag, 1995. Lecture Notes in Computer Science Volume 1000.
15. Ross Anderson and Roger Needham. Robustness principles for public key protocols. In D. Coppersmith, editor, *Advances in Cryptology – Crypto '95*, pages 236–247. Springer-Verlag, 1995. Lecture Notes in Computer Science Volume 963.
16. Jun Anzai, Natsume Matsuzaki, and Tsutomu Matsumoto. A quick group key distribution scheme with 'entity revocation'. In K. Y. Lam et al., editors, *Advances in Cryptology – Asiacrypt '99*, pages 333–347. Springer-Verlag, 1999. Lecture Notes in Computer Science Volume 1716.
17. B. Arazi. Integrating a key distribution procedure into the digital signature standard. *Electronics Letters*, 29(11):966–967, 1993.
18. Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. Authenticated group key agreement and friends. In *5th ACM Conference on Computer and Communications Security*, pages 17–26. ACM Press, 1998.
19. Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4):628–639, April 2000.
20. Tuomas Aura. Strategies against replay attacks. In *10th IEEE Computer Security Foundations Workshop*, pages 59–68. IEEE Computer Society Press, 1997.
21. Tuomas Aura and Pekka Nikander. Stateless connections. In Y. Han et al., editors, *Information and Computer Security, First International Conference*, pages 87–97, Beijing, 1997. Springer-Verlag. Lecture Notes in Computer Science Volume 1334.
22. Joonsang Baek and Kwangjo Kim. Remarks on the unknown key share attacks. *IEICE Transactions Fundamentals*, E83-A(12):2766–2769, December 2000.
23. S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. On password-based authenticated key exchange using collisionful hash functions. In J. Pieprzyk et al., editors, *Information Security and Privacy, First Australasian Conference*, pages 299–310. Springer-Verlag, 1996. Lecture Notes in Computer Science Volume 1172.
24. R. K. Bauer, T. A. Berson, and R. J. Feiertag. A key distribution protocol using event markers. *ACM Transactions on Computer Systems*, 1(3):249–255, August 1983.
25. Klaus Becker and Uta Wille. Communication complexity of group key distribution. In *5th ACM Conference on Computer and Communications Security*, pages 1–6. ACM Press, 1998.

Petitioner Samsung Ex-1036, 0317

26. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In N. Koblitz, editor, *Advances in Cryptology – Crypto '96*, pages 1–15. Springer-Verlag, 1996. Lecture Notes in Computer Science Volume 1109.
27. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *30th ACM Symposium on Theory of Computing*, pages 419–428. ACM Press, 1998. Full version at `http://www-cse.ucsd.edu/users/mihir/papers/key-distribution.html`.
28. Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In H. Krawczyk, editor, *Advances in Cryptology – Crypto '98*, pages 26–45. Springer-Verlag, 1998. Lecture Notes in Computer Science Volume 1462.
29. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *Advances in Cryptology – Eurocrypt 2000*, pages 139–155. Springer-Verlag, 2000. Lecture Notes in Computer Science Volume 1807.
30. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *Advances in Cryptology – Crypto '93*, pages 232–249. Springer-Verlag, 1993. Lecture Notes in Computer Science Volume 773. Full version at `http://www-cse.ucsd.edu/users/mihir`.
31. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security*, pages 62–73. ACM Press, 1993.
32. Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption - how to encrypt with RSA. In A. De Santis, editor, *Advances in Cryptology – Eurocrypt '94*, pages 92–111. Springer-Verlag, 1995. Lecture Notes in Computer Science Volume 950.
33. Mihir Bellare and Phillip Rogaway. Provably secure session key distribution – the three party case. In *27th ACM Symposium on Theory of Computing*, pages 57–66. ACM Press, 1995.
34. Michael J. Beller, Li-Fung Chang, and Yacov Yacobi. Privacy and authentication on a portable communications system. In *GLOBECOM'91*, pages 1922–1927. IEEE Press, 1991.
35. Michael J. Beller, Li-Fung Chang, and Yacov Yacobi. Security for personal communication services: Public-key vs. private key approaches. In *3rd IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'92)*, pages 26–31. IEEE Press, October 1992.
36. Michael J. Beller, Li-Fung Chang, and Yacov Yacobi. Privacy and authentication on a portable communications system. *IEEE Journal on Selected Areas in Communications*, 11(6):821–829, August 1993.
37. Michael J. Beller and Yacov Yacobi. Fully-fledged two-way public key authentication and key agreement for low-cost terminals. *Electronics Letters*, 29(11):999–1001, May 1993.
38. Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE Symposium on Research in Security and Privacy*, pages 72–84. IEEE Computer Society Press, 1992.
39. Steven M. Bellovin and Michael Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password

Petitioner Samsung Ex-1036, 0318

file compromise. In *1st ACM Conference on Computer and Communications Security*, pages 244–250. ACM Press, 1993.

40. Shimshon Berkovits. How to broadcast a secret. In D. W. Davies, editor, *Advances in Cryptology – Eurocrypt '91*, pages 535–541. Springer-Verlag, 1991. Lecture Notes in Computer Science Volume 547.
41. Ray Bird, I. Gopal, Amir Herzberg, Philippe A. Janson, Shay Kutten, Refik Molva, and Moti Yung. Systematic design of a family of attack-resistant authentication protocols. *IEEE Journal on Selected Areas in Communications*, 11(5):679–693, June 1993.
42. I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, 1999.
43. Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key agreement protocols and their security analysis. In M. Darnell, editor, *Crypography and Coding - 6th IMA Conference*, pages 30–45. Springer-Verlag, 1997. Lecture Notes in Computer Science Volume 1355.
44. Simon Blake-Wilson and Alfred Menezes. Entity authentication and authenticated key transport protocols employing asymmetric techniques. In B. Christianson et al., editors, *Security Protocols – 5th International Workshop*, pages 137–158. Springer-Verlag, 1998. Lecture Notes in Computer Science Volume 1361.
45. Simon Blake-Wilson and Alfred Menezes. Authenticated Diffie-Hellman key agreement protocols. In S. Tavares et al., editors, *Selected Areas in Cryptography, 5th International Workshop*, pages 339–361. Springer-Verlag, 1999. Lecture Notes in Computer Science Volume 1556.
46. Simon Blake-Wilson and Alfred Menezes. Unknown key-share attacks on the Station-to-Station (STS) protocol. In H. Imai et al., editors, *Public Key Cryptography*, pages 154–170. Springer-Verlag, 1999. Lecture Notes in Computer Science Volume 1560. Also at `http://www.cacr.math.uwaterloo.ca/~ajmeneze/publications/sts.ps`.
47. Daniel Bleichenbacher. Personal Communication, April 2001.
48. Rolf Blom. Non-public key distribution. In D. Chaum et al., editors, *Advances in Cryptology: Proceedings of Crypto '82*, pages 231–236, New York, 1982. Plenum Publishing.
49. Rolf Blom. An optimal class of symmetric key generation systems. In T. Beth et al., editors, *Advances in Cryptology: Proceedings of Eurocrypt '84*, pages 335–338. Springer-Verlag, 1984. Lecture Notes in Computer Science Volume 209.
50. Carlo Blundo and Antonella Cresti. Space requirements for broadcast encryption. In A. De Santis, editor, *Advances in Cryptology – Eurocrypt '94*, pages 287–298. Springer-Verlag, 1995. Lecture Notes in Computer Science Volume 950.
51. Carlo Blundo, Luiz A. Frota Mattos, and Douglas R. Stinson. Trade-offs between communication and storage in unconditionally secure schemes for broadcast encryption and interactive key distribution. In N. Koblitz, editor, *Advances in Cryptology – Crypto '96*, pages 387–400. Springer-Verlag, 1996. Lecture Notes in Computer Science Volume 1109.
52. Michael S. Borella. Methods and protocols for secure key negotiation using IKE. *IEEE Network*, 14(4):18–29, July/August 2000.

Petitioner Samsung Ex-1036, 0319

53. Maurizio Kliban Boyarsky. Public-key cryptography and password protocols: The multi-user case. In *6th ACM Conference on Computer and Communications Security*, pages 63–72. ACM Press, 1999.
54. Colin Boyd. Hidden assumptions in cryptographic protocols. *IEE Proceedings - Computers and Digital Techniques*, 137(6):433–436, November 1990.
55. Colin Boyd. Security architectures using formal methods. *IEEE Journal on Selected Areas in Communications*, 11(5):694–701, 1993.
56. Colin Boyd. Towards a classification of key agreement protocols. In *8th IEEE Computer Security Foundations Workshop*, pages 38–43. IEEE Computer Society Press, 1995.
57. Colin Boyd. A class of flexible and efficient key management protocols. In *9th IEEE Computer Security Foundations Workshop*, pages 2–8. IEEE Computer Society Press, 1996.
58. Colin Boyd. On key agreement and conference key agreement. In V. Varadharajan et al., editors, *Security and Privacy – Proceedings of ACISP'97*, pages 294–302. Springer-Verlag, 1997. Lecture Notes in Computer Science Volume 1270.
59. Colin Boyd and Wenbo Mao. Limitations of logical analysis of cryptographic protocols. In *Pre-proceedings – Eurocrypt '93*, 1993.
60. Colin Boyd and Wenbo Mao. On a limitation of BAN logic. In T. Helleseth, editor, *Advances in Cryptology – Eurocrypt '93*, pages 240–247. Springer-Verlag, 1994. Lecture Notes in Computer Science Volume 765.
61. Colin Boyd and Anish Mathuria. Systematic design of key establishment protocols based on one-way functions. *IEE Proceedings - Computers and Digital Techniques*, 144(2):93–99, 1997.
62. Colin Boyd and Anish Mathuria. Key establishment protocols for secure mobile communications: A critical survey. *Computer Communications*, 23:575–587, 2000.
63. Colin Boyd and Juan Manuel González Nieto. Round-optimal contributory conference key agreement. In Y. Desmedt, editor, *Public Key Cryptography – PKC 2003*, pages 161–174. Springer-Verlag, 2003. Lecture Notes in Computer Science Volume 2567.
64. Colin Boyd and DongGook Park. Public key protocols for wireless communications. In *1st International Conference on Information Security and Cryptology*, pages 47–57. Korea Institute of Information Security and Cryptology, 1998. `http://sky.fit.qut.edu.au/~boydc/papers/icisc98.ps.gz`.
65. Victor Boyko, Phillip MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In B. Preneel, editor, *Advances in Cryptology – Eurocrypt 2000*, pages 156–171. Springer-Verlag, 2000. Lecture Notes in Computer Science Volume 1807.
66. Stephen H. Brackin. Automatically detecting most vulnerabilities in cryptographic protocols. In *DARPA Information Survivability Conference and Exposition (DISCEX '00)*, pages 222–236. IEEE Computer Society Press, 1999. `http://www.arca.com/projects/docs/brackin/discex00.pdf`.
67. Stephen H. Brackin. Complete, automatic analysis of cryptographic protocols: Final report. `http://www.arca.com/projects/docs/brackin/FinalReport.pdf`, July 1999.
68. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Provably authenticated group Diffie-Hellman key exchange – the dynamic case. In C. Boyd,

Petitioner Samsung Ex-1036, 0320

editor, *Advances in Cryptology – Asiacrypt 2001*, pages 290–309. Springer-Verlag, 2001. Lecture Notes in Computer Science Volume 2248.

69. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In L. Knudsen, editor, *Advances in Cryptology – Eurocrypt 2002*, pages 321–336. Springer-Verlag, 2002. Lecture Notes in Computer Science Volume 2332.
70. Emmanuel Bresson, Olivier Chevassut, David Pointcheval, and Jean-Jacques Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *8th ACM Conference on Computer and Communications Security*, pages 255–264. ACM Press, 2001.
71. A. E. Brouwer, R. Pellikaan, and E. R. Verheul. Doing more with fewer bits. In K. Y. Lam et al., editors, *Advances in Cryptology – Asiacrypt '99*, pages 321–332. Springer-Verlag, 1999. Lecture Notes in Computer Science Volume 1716.
72. Dan Brown and Alfred Menezes. A small subgroup attack on Arazi's key agreement protocol. *Bulletin of the ICA*, 37:45–50, January 2003.
73. Mike Burmester. Cryptanalysis of the Chang-Wu-Chen key distribution system. In T. Helleseth, editor, *Advances in Cryptology – Eurocrypt '93*, pages 440–442. Springer-Verlag, 1994. Lecture Notes in Computer Science Volume 765.
74. Mike Burmester. On the risk of opening distributed keys. In Y. Desmedt, editor, *Advances in Cryptology – Crypto '94*, pages 308–317. Springer-Verlag, 1994. Lecture Notes in Computer Science Volume 839.
75. Mike Burmester and Yvo Desmedt. A secure and efficient conference key distribution system. In A. De Santis, editor, *Advances in Cryptology – Eurocrypt '94*, pages 275–286. Springer-Verlag, 1995. Lecture Notes in Computer Science Volume 950.
76. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Proceedings of the Royal Society of London*, A426:233–271, 1989.
77. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.
78. Michael Burrows, Martín Abadi, and Roger Needham. Rejoinder to Nessett. *ACM Operating Systems Review*, 24:39–40, April 1990.
79. Michael Burrows, Martín Abadi, and Roger Needham. The scope of a logic of authentication. In J. Feigenbaum et al., editors, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 2, pages 119–126. AMS/ACM, 1991.
80. Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *Advances in Cryptology – Eurocrypt '99*, pages 107–122. Springer-Verlag, 1999. Lecture Notes in Computer Science Volume 1592.
81. Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast security: A taxonomy and some efficient constructions. In *INFOCOM'99*, volume 2, pages 708–716. IEEE Press, March 1999.
82. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *Advances in Cryptology – Eurocrypt 2001*, pages 453–474. Springer-Verlag, 2001. Lecture Notes in Computer Science Volume 2045. `http://eprint.iacr.org/2001/040/`.

Petitioner Samsung Ex-1036, 0321

83. Ulf Carlsen. Cryptographic protocol flaws - know your enemy. In *7th IEEE Computer Security Foundations Workshop*, pages 192–200. IEEE Computer Society Press, June 1994.
84. Ulf Carlsen. Optimal privacy and authentication on a portable communications system. *ACM Operating Systems Review*, 28(3):16–23, 1994.
85. Chin-Chen Chang, Tzong-Chen Wu, and C.P. Chen. The design of a conference key distribution system. In J. Seberry et al., editors, *Advances in Cryptology – Auscrypt '92*, pages 459–466. Springer-Verlag, 1993. Lecture Notes in Computer Science Volume 718.
86. Jau Liang Chen and Tzonelih Hwang. Identity-based conference key broadcast schemes with user authentication. *Computers and Security*, 13:53–57, 1994.
87. L. Chen, D. Gollmann, and Chris J. Mitchell. Key distribution without individual trusted authentication servers. In *8th IEEE Computer Security Foundations Workshop*, pages 30–36. IEEE Computer Society Press, June 1995.
88. P.-C. Cheng. An architecture for the Internet Key Exchange protocol. *IBM Systems Journal*, 40(3):721–746, 2001.
89. Takeshi Chikazawa and Tohru Inoue. A new key sharing system for global telecommunications. In *GLOBECOM '90*, pages 1069–1072. IEEE Press, 1990.
90. Takeshi Chikazawa and Atsuhiro Yamagishi. An improved identity-based one-way conference key sharing system. In *Proceedings of ICCS/ISITA '92*, pages 270–273, Singapore, November 1992. IEEE Press.
91. Guang-Huei Chiou and Wen-Tsuen Chen. Secure broadcasting using the secure lock. *IEEE Transactions on Software Engineering*, 15(8):929–934, August 1989.
92. B. Chor, A. Fiat, and M. Naor. Tracing traitors. In Y. Desmedt, editor, *Advances in Cryptology – Crypto '94*, pages 257–270. Springer-Verlag, 1994. Lecture Notes in Computer Science Volume 839.
93. John Clark and Jeremy Jacob. On the security of recent protocols. *Information Processing Letters*, 56(3):151–155, November 1995.
94. John Clark and Jeremy Jacob. Attacking authentication protocols. *High Integrity Systems*, 1(5):465–473, August 1996.
95. John Clark and Jeremy Jacob. A survey of authentication protocol literature: Version 1.0. `http://www.cs.york.ac.uk/~jac/`, 1997.
96. E. M. Clarke, S. Jha, and W. Marrero. Verifying security protocols with Brutus. *ACM Transactions on Software Engineering and Methodology*, 9(4):443–487, October 2000.
97. Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In H. Krawczyk, editor, *Advances in Cryptology – Crypto '98*, pages 13–25. Springer-Verlag, 1998. Lecture Notes in Computer Science Volume 1462.
98. Joan Daemen and Vincent Rijmen. *The Design of Rijndael.* Springer-Verlag, 2002.
99. Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.
100. Yvo Desmedt and Mike Burmester. Towards practical 'proven secure' authenticated key distribution. In *1st ACM Conference on Computer and Communications Security*, pages 228–231. ACM Press, 1993.
101. Tim Dierks and Christopher Allen. *The TLS Protocol Version 1.0.* The Internet Society, January 1999. RFC 2246.
102. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

Petitioner Samsung Ex-1036, 0322

103. Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchange. *Designs, Codes and Cryptography*, 2:107–125, 1992.
104. Yun Ding and Patrick Horster. Undetectable on-line password guessing attacks. *ACM Operating Systems Review*, 29(4):77–86, October 1995.
105. Régis Dupont and Andreas Enge. Practical non-interactive key distribution based on pairings. Cryptology ePrint Archive, Report 2002/136, 2002. `http://eprint.iacr.org/2002/136/`.
106. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, July 1985.
107. J. H. Ellis. The history of non-secret encryption. Technical Report, CESG, UK, 1987. `http://www.cesg.gov.uk/publications/media/nsecret/ellis.pdf`.
108. F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *IEEE Symposium on Security and Privacy*, pages 160–171. IEEE Computer Society Press, 1998.
109. D. C. Feldmeier and P. R. Karn. UNIX password security—ten years later (invited). In G. Brassard, editor, *Advances in Cryptology – Crypto '89*, pages 44–63. Springer-Verlag, 1990. Lecture Notes in Computer Science Volume 435.
110. Niels Ferguson and Bruce Schneier. A cryptographic evaluation of IPsec. `http://www.counterpane.com/ipsec.html`, 2000.
111. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology – Crypto '86*, pages 186–194. Springer-Verlag, 1987. Lecture Notes in Computer Science Volume 263.
112. Amos Fiat and Moni Naor. Broadcast encryption. In D. R. Stinson, editor, *Advances in Cryptology – Crypto '93*, pages 480–491. Springer-Verlag, 1993. Lecture Notes in Computer Science Volume 773.
113. Warwick Ford and Burton S. Kaliski Jr. Server-assisted generation of a strong secret from a password. In *9th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2000*, pages 176–180. IEEE Press, 2000.
114. Juan A. Garay, Jessica Staddon, and Avishai Wool. Long-lived broadcast encryption. In M. Bellare, editor, *Advances in Cryptology – Crypto 2000*, pages 333–352. Springer-Verlag, 2000. Lecture Notes in Computer Science Volume 1880.
115. Marc Girault. Self-certified public keys. In D. W. Davies, editor, *Advances in Cryptology – Eurocrypt '91*, pages 490–497. Springer-Verlag, 1991. Lecture Notes in Computer Science Volume 547.
116. Marc Girault and Jean-Claude Paillès. An identity-based scheme providing zero-knowledge authentication and authenticated key exchange. In *European Symposium on Research in Computer Security*, pages 173–184, Toulouse, October 1990. AFCET.
117. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
118. Dieter Gollmann. What do we mean by entity authentication? In *IEEE Symposium on Security and Privacy*, pages 46–54. IEEE Computer Society Press, 1996.

Petitioner Samsung Ex-1036, 0323

119. Dieter Gollmann. Insider fraud (position paper). In B. Christianson et al., editors, *Security Protocols – 6th International Workshop*, pages 213–226. Springer-Verlag, 1999. Lecture Notes in Computer Science Volume 1550.
120. Dieter Gollmann. Authentication – myths and misconceptions. *Progress in Computer Science and Applied Logic*, 20:203–225, 2001.
121. Dieter Gollmann. Authentication by correspondence. *IEEE Journal on Selected Areas in Communications*, 21(1):88–95, January 2003.
122. Li Gong. Using one-way functions for authentication. *ACM Computer Communication Review*, 19(5):8–11, October 1989.
123. Li Gong. A security risk of depending on synchronized clocks. *ACM Operating Systems Review*, 26(1):49–53, January 1992.
124. Li Gong. Increasing availability and security of an authentication service. *IEEE Journal on Selected Areas in Communications*, 11(5):657–662, June 1993.
125. Li Gong. Lower bounds on messages and rounds for network authentication protocols. In *1st ACM Conference on Computer and Communications Security*, pages 26–37. ACM Press, 1993.
126. Li Gong. Variations on the themes of message freshness and replay. In *6th IEEE Computer Security Foundations Workshop*, pages 131–136. IEEE Computer Society Press, June 1993.
127. Li Gong. New protocols for third-party-based authentication and secure broadcast. In *2nd ACM Conference on Computer and Communications Security*, pages 176–183. ACM Press, 1994.
128. Li Gong. Collisionful keyed hash functions with selectable collisions. *Information Processing Letters*, 55(3):167–170, August 1995.
129. Li Gong. Optimal authentication protocols resistant to password guessing attacks. In *8th IEEE Computer Security Foundations Workshop*, pages 24–29. IEEE Computer Society Press, June 1995.
130. Li Gong, Mark A. Lomas, Roger Needham, and Jerome H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, June 1993.
131. Li Gong, Roger Needham, and Raphael Yahalom. Reasoning about belief in cryptographic protocols. In *IEEE Symposium on Security and Privacy*, pages 234–248. IEEE Computer Society Press, 1990.
132. Li Gong and Paul Syverson. Fail-stop protocols: An approach to designing secure protocols. In *Dependable Computing for Critical Applications 5*. IEEE Computer Society, 1998. `http://java.sun.com/people/gong/papers/sri-csl-tr94-14.ps.gz`.
133. Li Gong and David J. Wheeler. A matrix key-distribution scheme. *Journal of Cryptology*, 2(1):51–59, 1990.
134. Andrzej Goscinski and Michael Wang. Conference authentication and key distribution service in the RHODOS distributed system. In *Communications on the Move, ICCS/ISITA '92*, pages 284–289, Singapore, November 1992. IEEE Press.
135. Kenneth C. Goss. *Cryptographic Method and Apparatus for Public Key Exchange with Authentication*. US Patent 4,956,863, September 1990.
136. James W. Gray III. On the Clark–Jacob version of SPLICE/AS. *Information Processing Letters*, 62:251–254, 1997.
137. L. Guillou and J.-J. Quisquater. A practical zero knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In

Petitioner Samsung Ex-1036, 0324

C. G. Günther, editor, *Advances in Cryptology – Eurocrypt '88*, pages 123–128. Springer-Verlag, 1988. Lecture Notes in Computer Science Volume 330.
138. Christof G. Günther. An identity-based key exchange protocol. In J.-J. Quisquater et al., editors, *Advances in Cryptology – Eurocrypt '89*, pages 29–37. Springer-Verlag, 1989. Lecture Notes in Computer Science Volume 434.
139. Shai Halevi and Hugo Krawczyk. Public-key cryptography and password protocols. In *5th ACM Conference on Computer and Communications Security*, pages 122–131. ACM Press, 1998.
140. Shai Halevi and Hugo Krawczyk. Public-key cryptography and password protocols. *ACM Transactions on Information and Systems Security*, 2(3):230–268, August 1999.
141. Thomas Hardjono and Gene Tsudik. IP multicast security: Issues and directions. *Annales de Telecom*, pages 324–340, July–August 2000.
142. D. Harkins and D. Carrel. *The Internet Key Exchange (IKE)*. The Internet Society, November 1998. RFC 2409.
143. Shouichi Hirose and Katsuo Ikeda. A conference key distribution system for the star configuration based on the discrete logarithm problem. *Information Processing Letters*, 62:189–192, 1997.
144. Shouichi Hirose and Susumu Yoshida. An authenticated Diffie-Hellman key agreement protocol secure against active attacks. In H. Imai et al., editors, *Public Key Cryptography*, pages 135–148. Springer-Verlag, 1998. Lecture Notes in Computer Science Volume 1431.
145. Günther Horn, Keith M. Martin, and Chris J. Mitchell. Authentication protocols for mobile network environment value-added services. *IEEE Transactions on Vehicular Technology*, 51(2):383–392, March 2002.
146. Günther Horn and Bart Preneel. Authentication and payment in future mobile systems. In J.-J. Quisquater et al., editors, *Computer Security - ESORICS '98*, pages 277–293. Springer-Verlag, 1998. Lecture Notes in Computer Science Volume 1485.
147. Gwoboa Horng. An efficient and secure protocol for multi-party key establishment. *The Computer Journal*, 44(5):463–470, 2001.
148. Gwoboa Horng and Chi-Kuo Hsu. Weaknesses in the Helsinki protocol. *Electronics Letters*, 34(4):354–355, February 1998.
149. Patrick Horster, Markus Michels, and Holger Petersen. Authenticated encryption schemes with low communication costs. Technical Report TR-94-2-R, University of Technology Chemnitz-Zwickau, May 1994. Revised version at `http://www.geocities.com/CapeCanaveral/Lab/8967/TR-94-2.ps.gz`.
150. Patrick Horster, Markus Michels, and Holger Petersen. Authenticated encryption schemes with low communications costs. *Electronics Letters*, 30(15):1212–1213, July 1994.
151. T. Hwang and J. L. Chen. Identity-based conference key broadcast systems. *IEE Proceedings - Computers and Digital Techniques*, 141(1):57–60, January 1994.
152. Tzonelih Hwang and Yung-Hsiang Chen. On the security of SPLICE/AS – the authentication system in WIDE internet. *Information Processing Letters*, 53:97–101, 1995.
153. Colin I'Anson and Chris J. Mitchell. Security defects in CCITT recommendation X.509 – the directory authentication framework. *ACM Computer Communication Review*, 20(2):30–34, 1990.

Petitioner Samsung Ex-1036, 0325

154. IEEE. *P1363 Standard Specifications for Public-Key Cryptography*, January 2000. IEEE Std 1363-2000.
155. IEEE. *P1363.2 Standard Specifications for Password-based Public-Key Cryptographic Techniques*, December 2002. Draft Version `http://grouper.ieee.org/groups/1363/passwdPK/draft.html`.
156. Ingemar Ingemarsson, Donald T. Tang, and C. K. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, IT-28(5):714–720, September 1982.
157. ISO. *Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture ISO 7498-2*, 1989. International Standard.
158. ISO. *Information Technology – Security Techniques – Key Management – Part 2: Mechanisms Using Symmetric Techniques ISO/IEC 11770-2*, 1996. International Standard.
159. ISO. *Information Technology – Security Techniques – Entity Authentication Mechanisms – Part 3: Entity Authentication Using a Public Key Algorithm ISO/IEC 9798-3*, 2nd edition, 1998. International Standard.
160. ISO. *Information Technology – Security Techniques – Entity Authentication – Part 2: Mechanisms Using Symmetric Encipherment Algorithms ISO/IEC 9798-2*, 2nd edition, 1999. International Standard.
161. ISO. *Information Technology – Security Techniques – Entity Authentication Mechanisms – Part 5: Mechanisms using Zero Knowledge Techniques ISO/IEC 9798-5*, 1999. International Standard.
162. ISO. *Information Technology – Security Techniques – Key Management – Part 3: Mechanisms Using Asymmetric Techniques ISO/IEC 11770-3*, 1999. International Standard.
163. ITU/ISO. *Information Technology – Open Systems Interconnection – The Directory – Part 8: Authentication Framework, ITU-T Rec. X.509 – ISO/IEC 9594-8*, 1995. International Standard.
164. David P. Jablon. Strong password-only authenticated key exchange. *ACM Computer Communication Review*, 26(5):5–26, October 1996. `http://www.IntegritySciences.com`.
165. David P. Jablon. Extended password key exchange protocols immune to dictionary attack. In *6th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 248–255. IEEE Press, 1997.
166. David P. Jablon. Password authentication using multiple servers. In D. Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, pages 344–360. Springer-Verlag, 2001. Lecture Notes in Computer Science Volume 2020.
167. Markus Jakobsson and David Pointcheval. Mutual authentication for low-power mobile devices. In P. Syverson, editor, *Financial Cryptography*, pages 178–195. Springer-Verlag, 2001. Lecture Notes in Computer Science Volume 2339.
168. C. J. A. Jansen. On the key storage requirements for secure terminals. *Computers and Security*, 5:145–149, 1986.
169. Philippe Janson and Gene Tsudik. Secure and minimal protocols for authenticated key distribution. *Computer Communications*, 18(9):645–653, September 1995.
170. Barry Jaspan. Dual-workfactor encrypted key exchange: Efficiently preventing password chaining and dictionary attacks. In *6th USENIX Security Symposium*, San Jose, California, July 1996. `http://www.usenix.org/publications/library/proceedings/sec96/`.

Petitioner Samsung Ex-1036, 0326

171. Antoine Joux. A one round protocol for tripartite Diffie-Hellman. In W. Bosma, editor, *Algorithmic Number Theory, 4th International Symposium, ANTS-IV*, pages 385–393. Springer-Verlag, 2000. Lecture Notes in Computer Science Volume 1838.
172. Ari Juels and John Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Network and Distributed System Security Symposium*. Internet Society, February 1999. `http://www.isoc.org/isoc/conferences/ndss/99/proceedings/`.
173. Mike Just, Evangelos Kranakis, Danny Krizanc, and Paul van Oorschot. On key distribution via true broadcasting. In *2nd ACM Conference on Computer and Communications Security*, pages 81–88. ACM Press, 1994.
174. Mike Just and Paul C. van Oorschot. Addressing the problem of undetected signature key compromise. In *Network and Distributed System Security Symposium*. Internet Society, February 1999. `http://www.isoc.org/isoc/conferences/ndss/99/proceedings/`.
175. Mike Just and Serge Vaudenay. Authenticated multi-party key agreement. In K. Kim et al., editors, *Advances in Cryptology – Asiacrypt '96*, pages 36–49. Springer-Verlag, 1996. Lecture Notes in Computer Science Volume 1163.
176. Burton S. Kaliski Jr. An unknown key-share attack on the MQV key agreement protocol. *ACM Transactions on Information and System Security*, 4(3):275–288, August 2001.
177. P. Karn and W. Simpson. *Photuris: Session-Key Management Protocol*. The Internet Society, March 1999. RFC 2522.
178. Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In B. Pfitzmann, editor, *Advances in Cryptology – Eurocrypt 2001*, pages 475–494. Springer-Verlag, 2001. Lecture Notes in Computer Science Volume 2045.
179. Charlie Kaufman and Radia Perlman. PDM: A new strong password-based protocol. In *10th USENIX Security Symposium*, August 2001.
180. John Kelsey, Bruce Schneier, and David Wagner. Protocol interactions and the chosen protocol attack. In B. Christianson et al., editors, *Security Protocols – 5th International Workshop*, pages 91–104. Springer-Verlag, 1998. Lecture Notes in Computer Science Volume 1361.
181. R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.
182. Seungjoo Kim, Masahiro Mambo, Takeshi Okamoto, Hiroki Shizuya, Mitsuru Tada, and Dongho Won. On the security of the Okamoto-Tanaka ID-based key exchange scheme against active attacks. *IEICE Transactions Fundamentals*, E84-A(1):231–238, January 2001. `http://search.ieice.or.jp/2001/files/e000a01.htm#e84-a,1,231`.
183. Seungjoo Kim, Soohyun Oh, Sangjoon Park, and Dongho Won. On Saeednia's key-exchange protocols. Technical Report TR-98-1, Sungkyunkwan University, Department of Information Engineering, May 1998. `http://dosan.skku.ac.kr/~sjkim/tr/TR981.PS`.
184. Yongdae Kim, Adrian Perrig, and Gene Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *7th ACM Conference on Computer and Communications Security*, pages 235–244. ACM Press, 2000.
185. B. Klein, M. Otten, and T. Beth. Conference key distribution protocols in distributed systems. In P. G. Farrell, editor, *Codes and Cyphers – Cryptography and Coding IV*, pages 225–241. IMA, 1995.

Petitioner Samsung Ex-1036, 0327

186. Neal Koblitz. *Algebraic Aspects of Cryptography.* Springer-Verlag, 1998.
187. John T. Kohl. The use of encryption in Kerberos for network authentication. In G. Brassard, editor, *Advances in Cryptology – Crypto '89*, pages 35–43. Springer-Verlag, 1989. Lecture Notes in Computer Science Volume 435.
188. John T. Kohl and B. Clifford Neuman. *The Kerberos network authentication service (V5).* The Internet Society, September 1993. RFC 1510.
189. John T. Kohl, B. Clifford Neuman, and Theodore Y. Ts'o. The evolution of the Kerberos authentication system. In F. Brazier et al., editors, *Distributed Open Systems*, pages 78–94. IEEE Computer Society Press, 1994.
190. Kenji Koyama. Secure conference key distribution schemes for conspiracy attack. In R. A. Rueppel, editor, *Advances in Cryptology – Eurocrypt '92*, pages 449–453. Springer-Verlag, 1992. Lecture Notes in Computer Science Volume 658.
191. Kenji Koyama and Kazuo Ohta. Identity-based conference key distribution systems. In C. Pomerance, editor, *Advances in Cryptology – Crypto '87*, pages 175–184. Springer-Verlag, 1987. Lecture Notes in Computer Science Volume 293.
192. Kenji Koyama and Kazuo Ohta. Security of improved identity-based conference key distribution systems. In C. G. Günther, editor, *Advances in Cryptology – Eurocrypt '88*, pages 11–19. Springer-Verlag, 1988. Lecture Notes in Computer Science Volume 330.
193. Hugo Krawczyk. SKEME: A versatile secure key exchange mechanism for Internet. In *Symposium on Network and Distributed System Security*, pages 114–127. IEEE Computer Society Press, 1996.
194. Wei-Chi Ku and Sheng-De Wang. Cryptanalysis of modified authenticated key agreement protocol. *Electronics Letters*, 36(21):1770–1771, October 2000.
195. Taekyoung Kwon. Ultimate solution to authentication via memorable password. IEEE P1363 Standards Group contibution, 2000. `http://grouper.ieee.org/groups/1363/passwdPK/contributions.html#amp`.
196. Taekyoung Kwon and Jooseok Song. Efficient and secure password-based authentication protocols against guessing attacks. *Computer Communications*, 21:853–861, 1998.
197. Taekyoung Kwon and Jooseok Song. Efficient key exchange and authentication protocols protecting weak secrets. *IEICE Transactions Fundamentals*, E81-A(1):156–163, January 1998.
198. Taekyoung Kwon and Jooseok Song. Secure agreement scheme for g^{xy} via password authentication. *Electronics Letters*, 35(11):892–893, May 1999.
199. Chi Sung Laih, Jau Yien Lee, and Lein Harn. A new threshold scheme and its application in designing the conference key distribution cryptosystem. *Information Processing Letters*, 32:95–99, 1989.
200. Chi-Sung Laih and Sung-Ming Yen. On the design of conference key distribution systems for the broadcasting networks. In *INFOCOM '93 Conference on Computer Communications*, pages 1406–1413. IEEE Computer Society Press, 1993.
201. Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, and Scott Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28(2):119–134, March 2003.
202. Hyoungkyu Lee, Kiwook Sohn, Hyoungkyu Yang, and Dongho Won. The efficient 3-pass password-based key exchange protocol with low computational cost

Petitioner Samsung Ex-1036, 0328

for client. In J. Song, editor, *Information Security and Cryptology – ICISC'99*, pages 147–155. Springer-Verlag, 2000. Lecture Notes in Computer Science Volume 1787.
203. T. Leighton and S. Micali. Secret-key agreement without public-key cryptography. In D. R. Stinson, editor, *Advances in Cryptology – Crypto '93*, pages 456–479. Springer-Verlag, 1993. Lecture Notes in Computer Science Volume 773.
204. Arjen Lenstra and Eric Verheul. The XTR public key system. In M. Bellare, editor, *Advances in Cryptology – Crypto 2000*, pages 1–19. Springer-Verlag, 2000. Lecture Notes in Computer Science Volume 1880.
205. Chae Hoon Lim and Pil Joong Lee. Several practical protocols for authentication and key exchange. *Information Processing Letters*, 53:91–96, 1995.
206. Chae Hoon Lim and Pil Joong Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In B. Kaliski, editor, *Advances in Cryptology – Crypto '97*, pages 249–263. Springer-Verlag, 1997. Lecture Notes in Computer Science Volume 1294.
207. Chun-Li Lin, Hung-Min Sun, and Tzonelih Hwang. Three-party encrypted key exchange: Attacks and a solution. *ACM Operating Systems Review*, 34(4):12–20, October 2000.
208. Chun-Li Lin, Hung-Min Sun, Michael Steiner, and Tzonelih Hwang. Three-party encrypted key exchange without server public-keys. *IEEE Communications Letters*, 5(12):497–499, December 2001.
209. Iuon-Chang Lin, Chin-Chen Chang, and Min-Shiang Hwang. Security enhancement for the 'simple authentication key agreement algorithm'. In *24th Computer Software and Applications Conference (COMPSAC 2000)*, pages 113–115. IEEE Computer Society Press, 2000.
210. T. Mark A. Lomas, Li Gong, Jerome H. Saltzer, and Roger Needham. Reducing risks from poorly chosen keys. *ACM Operating Systems Review*, 23(5):14–18, December 1989.
211. Gavin Lowe. Breaking and fixing the Needham-Schroeder public key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 147–166. Springer-Verlag, 1996.
212. Gavin Lowe. Some new attacks upon security protocols. In *9th IEEE Computer Security Foundations Workshop*, pages 162–169. IEEE Computer Society Press, June 1996.
213. Gavin Lowe. Casper: A compiler for the analysis of security protocols. In *10th IEEE Computer Security Foundations Workshop*, pages 18–30. IEEE Computer Society Press, June 1997.
214. Gavin Lowe. A hierarchy of authentication specification. In *10th IEEE Computer Security Foundations Workshop*, pages 31–43. IEEE Computer Society Press, June 1997.
215. Stefan Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In B. Christianson et al., editors, *Security Protocols – 5th International Workshop*, pages 79–90. Springer-Verlag, 1998. Lecture Notes in Computer Science Volume 1361.
216. Philip MacKenzie. More efficient password-authenticated key exchange. In D. Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, pages 361–377. Springer-Verlag, 2001. Lecture Notes in Computer Science Volume 2020.
217. Philip MacKenzie. On the security of the SPEKE password-authenticated key exchange protocol. `http://eprint.iacr.org/2001/057`, July 2001.

Petitioner Samsung Ex-1036, 0329

218. Philip MacKenzie. The PAK suite: Protocols for password-authenticated key exchange. Technical Report 2002-46, DIMACS, October 2002. `http://dimacs.rutgers.edu/TechnicalReports/abstracts/2002/2002-46.html`.
219. Philip MacKenzie, Sarvar Patel, and Ram Swaminathan. Password-authenticated key exchange based on RSA. In T. Okamoto, editor, *Advances in Cryptology – Asiacrypt 2000*, pages 599–613. Springer-Verlag, 2000. Lecture Notes in Computer Science Volume 1976.
220. Masahiro Mambo and Hiroki Shizuya. A note on the complexity of breaking Okamoto-Tanaka ID-based key exchange scheme. *IEICE Transactions Fundamentals*, E82-A(1):77–80, January 1999.
221. Wenbo Mao. *Modern Cryptography: Theory and Practice.* Prentice Hall, 2003.
222. Wenbo Mao and Colin Boyd. Towards formal analysis of security protocols. In *6th IEEE Computer Security Foundations Workshop*, pages 147–158. IEEE Computer Society Press, 1993.
223. Wenbo Mao and Colin Boyd. On the use of encryption in cryptographic protocols. In P. G. Farrell, editor, *Codes and Cyphers – Cryptography and Coding IV*, pages 251–262, 1995.
224. Tsutomu Matsumoto, Youichi Takashima, and Hideki Imai. On seeking smart public-key-distribution systems. *Transactions of the IECE of Japan*, E69(2):99–106, February 1986.
225. Ueli Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In Y. Desmedt, editor, *Advances in Cryptology – Crypto '94*, pages 271–281. Springer-Verlag, 1994. Lecture Notes in Computer Science Volume 839.
226. Alain Mayer and Moti Yung. Secure protocol transformation via 'expansion': From two-party to groups. In *6th ACM Conference on Computer and Communications Security*, pages 83–92. ACM Press, 1999.
227. Keven S. McCurley. A key distribution system equivalent to factoring. *Journal of Cryptology*, 1(2):95–105, 1988.
228. David A. McGrew and Alan T. Sherman. Key establishment in large dynamic groups using one-way function trees. `http://www.cs.umbc.edu/~sherman/Papers/itse.ps`, 1998.
229. Catherine Meadows. The NRL Protocol Analyzer: An overview. *The Journal of Logic Programming*, 26(2):113–131, 1996.
230. Catherine Meadows. Analysis of the Internet Key Exchange protocol using the NRL Protocol Analyzer. In *IEEE Symposium on Security and Privacy*, pages 216–231. IEEE Computer Society Press, 1999.
231. Catherine Meadows. A formal framework and evaluation method for network denial of service. In *12th IEEE Computer Security Foundations Workshop*, pages 4–13. IEEE Computer Society Press, June 1999.
232. Catherine Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE Journal on Selected Areas in Communications*, 21(1):44–54, January 2003.
233. Alfred J. Menezes, Minqhua Qu, and Scott A. Vanstone. Some new key agreement protocols providing implicit authentication. In *Workshop on Selected Areas in Cryptography (SAC'95)*, pages 22–32, 1995.
234. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.
235. Chris J. Mitchell. Making serial number based authentication robust against loss of state. *ACM Operating Systems Review*, 34(3):56–59, July 2000.

Petitioner Samsung Ex-1036, 0330

236. Chris J. Mitchell. Breaking the simple authenticated key agreement (SAKA) protocol. Technical Report RHUL-MA-2001-2, Royal Holloway, University of London, Department of Mathematics, 2001. `http://www.rhul.ac.uk/mathematics/techreports`.
237. Chris J. Mitchell and Fred Piper. The cost of reducing key-storage requirements in secure networks. *Computers and Security*, 6:339–341, 1987.
238. Chris J. Mitchell and Andy Thomas. Standardising authentication protocols based on public key techniques. *Journal of Computer Security*, 2:23–36, 1993.
239. Chris J. Mitchell, Mike Ward, and Piers Wilson. On key control in key agreement protocols. *Electronics Letters*, 34:980–981, 1998.
240. Chris J. Mitchell and Chan Yeob Yeun. Fixing a problem in the Helsinki protocol. *ACM Operating Systems Review*, 32(4):21–24, October 1998.
241. John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murϕ. In *IEEE Symposium on Security and Privacy*, pages 141–151. IEEE Computer Society Press, 1997.
242. John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state analysis of SSL 3.0. In *7th USENIX Security Symposium*, January 1998. `http://www.usenix.org/publications/library/proceedings/sec98/technical.%html`.
243. Yi Mu and Vijay Varadharajan. On the design of security protocols for mobile communications. In J. Pieprzyk et al., editors, *Information Security and Privacy – ACISP'96*, pages 134–145. Springer-Verlag, 1996. Lecture Notes in Computer Science Volume 1172.
244. National Institute of Standards and Technology. *Escrowed Encryption Standard (EES)*, February 1994. `http://csrc.nist.gov/publications/fips/fips185/fips185.txt`.
245. National Institute of Standards and Technology. *Secure Hash Standard*, April 1995. `http://www.itl.nist.gov/fipspubs/fip180-1.htm`.
246. National Institute of Standards and Technology. *Entity Authentication Using Public Key Cryptography*, February 1997. `http://csrc.nist.gov/CryptoToolkit/tkentauth.html`.
247. National Institute of Standards and Technology. *Digital Signature Standard (DSS)*, February 2000. `http://csrc.nist.gov/CryptoToolkit/tkdigsigs.html`.
248. National Security Agency. *SKIPJACK and KEA Algorithm Specification*, June 1998. `http://csrc.nist.gov/CryptoToolkit/skipjack/skipjack.pdf`.
249. Roger Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.
250. Dan M. Nessett. A critique of the Burrows, Abadi and Needham logic. *ACM Operating Systems Review*, 24(2):35–38, April 1990.
251. B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, September 1994.
252. Kaisa Nyberg. On one-pass authenticated key establshment schemes. In *Workshop on Selected Areas in Cryptography (SAC'95)*, pages 2–8, 1995.
253. Kaisa Nyberg and Rainer A. Rueppel. A new signature scheme based on the DSA giving message recovery. In *1st Conference on Computer and Communications Security*, pages 58–61. ACM Press, 1993.
254. Kaisa Nyberg and Rainer A. Rueppel. Weaknesses in some recent key agreement protocols. *Electronics Letters*, 30(1):26–27, January 1994.

Petitioner Samsung Ex-1036, 0331

255. Kaisa Nyberg and Rainer A. Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. In A. De Santis, editor, *Advances in Cryptology – Eurocrypt '94*, pages 182–193. Springer-Verlag, 1995. Lecture Notes in Computer Science Volume 950.
256. Eiji Okamoto. Key distribution systems based on identification information. In C. Pomerance, editor, *Advances in Cryptology – Crypto '87*, pages 194–202. Springer-Verlag, 1987. Lecture Notes in Computer Science Volume 293.
257. Eiji Okamoto and Kazue Tanaka. Key distribution system based on identification information. *IEEE Journal on Selected Areas in Communications*, 7(4):481–485, May 1989.
258. H. Orman. *The OAKLEY Key Determination Protocol.* The Internet Society, November 1998. RFC 2412.
259. Dave Otway and Owen Rees. Efficient and timely mutual authentication. *ACM Operating Systems Review*, 21(1):8–10, January 1987.
260. C. Park, K. Kurosawa, T. Okamoto, and S. Tsujii. On key distribution and authentication in mobile radio networks. In T. Helleseth, editor, *Advances in Cryptology – Eurocrypt '93*, pages 461–465. Springer-Verlag, 1994. Lecture Notes in Computer Science Volume 765.
261. Chang-Seop Park. On certificate-based security protocols for wireless mobile communication systems. *IEEE Network*, 11(5):50–55, September/October 1997.
262. Sarvar Patel. Number theoretic attacks on secure password schemes. In *IEEE Symposium on Security and Privacy*, pages 236–247. IEEE Computer Society Press, 1997.
263. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.
264. Lawrence C. Paulson. Inductive analysis of the Internet protocol TLS. *ACM Transactions on Information and System Security*, 2(3):332–351, August 1999.
265. Lawrence C. Paulson. Inductive analysis of the internet protocol TLS. In B. Christianson et al., editors, *Security Protocols – 6th International Workshop*, pages 1–23. Springer-Verlag, 1999. Lecture Notes in Computer Science Volume 1550.
266. Lawrence C. Paulson. Relation between secrets: Two formal analyses of the Yahalom protocol. *Journal of Computer Security*, 9:197–216, 2001.
267. Olivier Pereira and Jean-Jacques Quisquater. A security analysis of the Cliques protocols suites. In *14th IEEE Computer Security Foundations Workshop*, pages 73–81. IEEE Computer Society Press, June 2001.
268. Radia Perlman and Charlie Kaufman. Secure password-based protocol for downloading a private key. In *Network and Distributed System Security Symposium.* Internet Society, February 1999. `http://www.isoc.org/isoc/conferences/ndss/99/proceedings/`.
269. Radia Perlman and Charlie Kaufman. Key exchange in IPSec: Analysis of IKE. *IEEE Internet Computing*, 4(6):50–56, November–December 2000.
270. Adrian Perrig. Efficient collaborative key management protocols for secure autonomous group communication. In *1999 International Workshop on Cryptographic Techniques and Electronic Commerce*, pages 192–202. City University of Hong Kong Press, 1999.
271. J. Pieprzyk and C.-H. Li. Multiparty key agreement protocols. *IEE Proceedings - Computers and Digital Techniques*, 147(4):229–236, July 2000.

Petitioner Samsung Ex-1036, 0332

272. M. O. Rabin. Digitalized signatures and public key functions as intractable as factorization. Technical Report MIT-LCS-TR-212, MIT Laboratory for Computer Science, 1979. http://www.lcs.mit.edu/publications/pubs/pdf/MIT-LCS-TR-212.pdf.
273. Michael K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in Rampart. In *2nd ACM Conference on Computer and Communications Security*, pages 68–80. ACM Press, 1994.
274. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.
275. Ohad Rodeh, Kenneth P. Birman, and Danny Dolev. Optimized group rekey for group communication systems. In *Network and Distributed System Security Symposium*. Internet Society, February 2000. http://www.isoc.org/isoc/conferences/ndss/2000/proceedings/.
276. Michael Roe, Bruce Christianson, and David Wheeler. Secure sessions from weak secrets. Technical Report, Microsoft Research, 1998. http://www.research.microsoft.com/users/mroe/ccsr-tr4.pdf.
277. A. W. Roscoe. Intensional specifications of security protocols. In *9th IEEE Computer Security Foundations Workshop*, pages 28–38. IEEE Computer Society Press, June 1996.
278. A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998.
279. Rainer A. Rueppel and Paul C. van Oorschot. Modern key agreement techniques. *Computer Communications*, 17(7):458–465, July 1994.
280. Peter Ryan and Steve Schneider. An attack on a recursive authentication protocol: A cautionary tale. *Information Processing Letters*, 65:7–10, 1998.
281. Peter Ryan and Steve Schneider. *Modelling and Analysis of Security Protocols*. Addison-Wesley, 2001.
282. S. Saeednia. Improvement of Günther's identity-based key exchange protocol. *Electronics Letters*, 36(18):1535–1536, August 2000.
283. Shahrokh Saeednia. A note on Girault's self-certified model. Technical Report 2001/100, Cryptology ePrint Archive, 2001. http://eprint.iacr.org/2001/100/.
284. Shahrokh Saeednia and Rei Safavi-Naini. Identity-based and self-certified key-exchange protocols. In V. Varadharajan et al., editors, *Security and Privacy – Second Australasian Conference*, pages 303–313. Springer-Verlag, 1997. Lecture Notes in Computer Science Volume 1270.
285. Shahrokh Saeednia and Rei Safavi-Naini. Efficient identity-based conference key distribution protocols. In C. Boyd et al., editors, *Information Security and Privacy – Third Australasian Conference*, pages 320–331. Springer-Verlag, 1998. Lecture Notes in Computer Science Volume 1438.
286. David Safford, David K. Hess, and Douglas Lee Schales. Texas A&M University anarchistic key authorization (AKA). In *Sixth USENIX Security Symposium*, July 1996. http://www.usenix.org/publications/library/proceedings/sec96/.
287. Ryuichi Sakai, Kiyoshi Ohgishi, and Masao Kasahara. Cryptosystems based on pairing. In *Symposium on Cryptography and Information Security*, Okinawa, January 2000.
288. Hisao Sakazaki, Eiji Okamoto, and Masahiro Mambo. Constructing identity-based key distribution systems over elliptic curves. *IEICE Transactions Fundamentals*, E81-A(10):2138–2143, October 1998.

Petitioner Samsung Ex-1036, 0333

289. M. Satyanarayanan. Integrating security in a large distributed system. *ACM Transactions on Computer Systems*, 7(3):247–280, August 1989.
290. Renate Scheidler, Johannes A. Buchmann, and Hugh C. Williams. A key-exchange protocol using real quadratic fields. *Journal of Cryptology*, 7(3):171–199, 1994.
291. C. P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *Advances in Cryptology – Crypto '89*, pages 239–252. Springer-Verlag, 1990. Lecture Notes in Computer Science Volume 435.
292. M. Scott. Security of ID-based key exchange scheme. *Electronics Letters*, 34(7):653–654, April 1998.
293. Dong Hwi Seo and P. Sweeney. Simple authenticated key agreement algorithm. *Electronics Letters*, 35(13):1073–1074, June 1999.
294. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.
295. Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley et al., editors, *Proceedings of Crypto '84*, pages 47–53. Springer-Verlag, 1985. Lecture Notes in Computer Science Volume 196.
296. Shiuh-Pyng Shieh, Wen-Her Yang, and Hun-Min Sun. An authentication protocol without trusted third party. *IEEE Communications Letters*, 1(3):87–89, May 1997.
297. Kyungah Shim. Some attacks on Chikazawa-Yamagishi ID-based key sharing scheme. *IEEE Communications Letters*, 7(3):145–147, March 2003.
298. Atsushi Shimbo and Shin ichi Kawamura. Cryptanalysis of several conference key distribution schemes. In H. Imai et al., editors, *Advances in Cryptology – Asiacrypt '91*, pages 265–276. Springer-Verlag, 1993. Lecture Notes in Computer Science Volume 739.
299. Victor Shoup. On formal models for secure key exchange. `http://www.shoup.net`, November 1999.
300. William Allen Simpson. Photuris: Design criteria. In H. Heys et al., editors, *Selected Areas in Cryptography, 6th International Workshop*, pages 226–241. Springer-Verlag, 2000. Lecture Notes in Computer Science Volume 1758.
301. N. P. Smart. Identity-based authenticated key agreement protocol based on Weil pairing. *Electronics Letters*, 38(13):630–632, June 2002.
302. N. P. Smart and S. Siksek. A fast Diffie-Hellman protocol in genus 2. *Journal of Cryptology*, 12(1):67–73, 1999.
303. Boyeon Song and Kwangjo Kim. Two-pass authenticated key agreement protocol with key confirmation. In B. Roy et al., editors, *Progress in Cryptology – Indocrypt 2000*, pages 237–249. Springer-Verlag, 2000. Lecture Notes in Computer Science Volume 1977.
304. Dawn Xiaodong Song. Athena: A new efficient automatic checker for security protocol analysis. In *12th IEEE Computer Security Foundations Workshop*, pages 192–202. IEEE Computer Society Press, June 1999.
305. D. G. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A secure audio teleconference system. In S. Goldwasser, editor, *Advances in Cryptology – Crypto '88*, pages 520–528. Springer-Verlag, 1989. Lecture Notes in Computer Science Volume 403.
306. Michael Steiner, Peter Buhler, Thomas Eirich, and Michael Waidner. Secure password-based cipher suite for TLS. *ACM Transactions on Information and System Security*, 4(2):134–157, May 2001.

Petitioner Samsung Ex-1036, 0334

307. Michael Steiner, Gene Tsudik, and Michael Waidner. Refinement and extension of encrypted key exchange. *ACM Operating Systems Review*, 29(3):22–30, July 1995.
308. Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman key distribution extended to group communication. In *3rd ACM Conference on Computer and Communications Security*, pages 31–37. ACM Press, 1996.
309. Michael Steiner, Gene Tsudik, and Michael Waidner. Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–780, August 2000.
310. Doug R. Stinson. On some methods for unconditionally secure key distribution and broadcast encryption. *Designs, Codes and Cryptography*, 12:215–243, 1997.
311. Stuart G. Stubblebine and Virgil D. Gligor. On message integrity in cryptographic protocols. In *IEEE Symposium on Research in Security and Privacy*, pages 85–104. IEEE Computer Society Press, 1992.
312. Stuart G. Stubblebine and Catherine A. Meadows. Formal characterization and automated analysis of known-pair and chosen-text attacks. *IEEE Journal on Selected Areas in Communications*, 18(4):571–581, April 2000.
313. Paul Syverson. A taxonomy of replay attacks. In *7th IEEE Computer Security Foundations Workshop*, pages 187–191. IEEE Computer Society Press, June 1994.
314. Paul Syverson. Limitations on design principles for public key protocols. In *IEEE Symposium on Security and Privacy*, pages 62–72. IEEE Computer Society Press, 1996.
315. Paul Syverson and Paul C. van Oorschot. On unifying some cryptographic protocol logics. In *IEEE Symposium on Research in Security and Privacy*, pages 14–28. IEEE Computer Society Press, 1994.
316. Kazue Tanaka and Eiji Okamoto. Key distribution system for mail systems using ID-related information directory. *Computers and Security*, 10:25–33, 1991.
317. M. Tatebayashi, N. Matsuzaki, and D. B. Newman Jr. Key distribution protocol for digital mobile communication systems. In G. Brassard, editor, *Advances in Cryptology – Crypto '89*, pages 324–334. Springer-Verlag, 1989. Lecture Notes in Computer Science Volume 435.
318. Y.-W. Tsai and T. Hwang. ID based public key cryptosystems based on Okamoto and Tanaka's ID based one way communication scheme. *Electronics Letters*, 26(10):666–668, May 1990.
319. Yuh-Min Tseng. Weakness in simple authenticated key agreement protocol. *Electronics Letters*, 36(1):48–49, January 2000.
320. Yuh-Min Tseng and Jinn-Ke Jan. ID-based cryptographic schemes using a non-interactive public-key distribution system. In *14th Annual Computer Security Applications Conference*, pages 237–243. IEEE Computer Society Press, December 1998.
321. Gene Tsudik and Els Van Herreweghen. Some remarks on protecting weak keys and poorly-chosen secrets from guessing attacks. In *12th Symposium on Reliable Distributed Sytems*, pages 136–142, October 1993.
322. Wen-Guey Tzeng and Zhi-Jha Tzeng. Round-efficient conference key agreement protocols with provable security. In T. Okamoto, editor, *Advances in Cryptology – Asiacrypt 2000*, pages 614–627. Springer-Verlag, 2000. Lecture Notes in Computer Science Volume 1976.

Petitioner Samsung Ex-1036, 0335

323. Paul C. van Oorschot. An alternate explanation of two BAN-logic 'failures'. In T. Helleseth, editor, *Advances in Cryptology – Eurocrypt '93*, pages 443–447. Springer-Verlag, 1994. Lecture Notes in Computer Science Volume 765.
324. Paul C. van Oorschot and Michael J. Wiener. On Diffie-Hellman key agreement with short exponents. In U. Maurer, editor, *Advances in Cryptology – Eurocrypt '96*, pages 332–343. Springer-Verlag, 1996. Lecture Notes in Computer Science Volume 1070.
325. David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *Second USENIX Workshop on Electronic Commerce*, November 1996. `http://www.usenix.org/publications/library/proceedings/ec96/index.html`.
326. D. Wallner, E. Harder, and R. Agee. *Key Management for Multicast: Issues and Architectures.* The Internet Society, June 1999. RFC 2627.
327. Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. Secure group communications using key graphs. *ACM Computer Communication Review*, 28(4):68–79, 1998.
328. Duncan S. Wong and Agnes H. Chan. Efficient and mutually authenticated key exchange for low power computing devices. In C. Boyd, editor, *Advances in Cryptology – Asiacrypt 2001*, pages 272–289. Springer-Verlag, 2001. Lecture Notes in Computer Science Volume 2248.
329. Thomas Y. C. Woo and Simon S. Lam. Authentication for distributed systems. *IEEE Computer*, 25(1):39–52, January 1992.
330. Thomas Y. C. Woo and Simon S. Lam. A lesson on authentication protocol design. *ACM Operating Systems Review*, 28(3):24–37, July 1994.
331. T.-C. Wu. Conference key distribution system with user anonymity based on algebraic approach. *IEE Proceedings - Computers and Digital Techniques*, 144(2):145–148, March 1997.
332. Thomas Wu. The secure remote password protocol. In *Network and Distributed System Security Symposium.* Internet Society, February 1998. `http://www.isoc.org/isoc/conferences/ndss/98/ndss98.htm`.
333. Thomas Wu. A real-world analysis of Kerberos password security. In *Network and Distributed System Security Symposium.* Internet Society, February 1999. `http://www.isoc.org/isoc/conferences/ndss/99/proceedings/`.
334. Yacov Yacobi. Attack on the Koyama-Ohta identity based key distribution scheme. In C. Pomerance, editor, *Advances in Cryptology – Crypto '87*, pages 429–433. Springer-Verlag, 1987. Lecture Notes in Computer Science Volume 293.
335. Yacov Yacobi. A key distribution 'paradox'. In A. J. Menezes et al., editors, *Advances in Cryptology – Crypto '90*, pages 268–273. Springer-Verlag, 1991. Lecture Notes in Computer Science Volume 537.
336. Yacov Yacobi and Z. Shmuely. On key distribution systems. In G. Brassard, editor, *Advances in Cryptology – Crypto '89*, pages pp.344–355. Springer-Verlag, 1989. Lecture Notes in Computer Science Volume 435.
337. Suguru Yamaguchi, Kiyohiko Okayama, and Hideo Miyahara. Design and implementation of an authentication system in WIDE internet environment. In *IEEE Region 10 Conference on Computer and Communications Systems*, pages 653–657, Hong Kong, September 1990.
338. Sung-Ming Yen. Cryptanalysis of an authentication and key distribution protocol. *IEEE Communications Letters*, 3(1):7–8, January 1999.

Petitioner Samsung Ex-1036, 0336

339. Sung-Ming Yen and Meng-Tzung Liu. High performance nonce-based authentication and key distribution protocols against password guessing attacks. *IEICE Transactions Fundamentals*, E80-A(11):2209–2217, November 1997.
340. Yuliang Zheng. On key agreement protocols based on tamper-proof hardware. *Information Processing Letters*, 53:49–54, 1995.
341. Yuliang Zheng. Digital signcryption or how to achieve cost (signature & encryption) << cost (signature) + cost (encryption). In B. Kaliski, editor, *Advances in Cryptology – Crypto '97*, pages 165–179. Springer-Verlag, 1997. Lecture Notes in Computer Science Volume 1294.
342. J. Zhou. Fixing of security flaw in IKE protocols. *Electronics Letters*, 35(13):1072–1073, June 1999.
343. J. Zhou. Further analysis of the Internet key exchange protocol. *Computer Communications*, 23:1606–1612, 2000.
344. Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.
345. R. Zuccherato. *Methods for Avoiding the 'Small-Subgroup' Attacks on the Diffie-Hellman Key Agreement Method for S/MIME*. The Internet Society, March 2000. RFC 2785.
346. Edward Zuk. Remarks on 'The design of a conference key distribution system'. In J. Seberry et al., editors, *Advances in Cryptology – Auscrypt '92*, pages 467–468. Springer-Verlag, 1993. Lecture Notes in Computer Science Volume 718.

Petitioner Samsung Ex-1036, 0337

Index of Protocols

Agnew–Mullin–Vanstone, 144–145
AKA, 132–133
AMP, 265–266
Anderson–Lomas, 283–284
Andrew, 81–83
Anzai, Matsuzaki and Matsumoto, 232
Arazi's key agreement, 178–179
ASPeCT, 192–193
Ateniese–Steiner–Tsudik key agreement, 161–162, 221–225
augmented EKE, 260–266
Bauer–Berson–Feiertag, 88
Becker–Wille's Octopus, 212–214
Bellare–Rogaway 3PKD, 98–99
Bellare–Rogaway MAP1, 76–77
Beller–Chang–Yacobi, 126–131
Bellovin–Merritt EKE, *See* EKE
Berkovits, 244
Bird *et al.* canonical, 75
Blake-Wilson–Menezes key transport, 120–121, 237
Blom key pre-distribution, 198
Boyd conference key agreement, 233–235
Boyd key agreement, 101
Boyd–Park, 194
Boyd two-pass, 83–84
B-SPEKE, 263
Burmester–Desmedt key agreement, 214–215
Burmester–Desmedt star, 235–237
Burmester–Desmedt tree, 235–237
Chang–Wu–Chen, 244
Chen–Gollmann–Mitchell, 105–106
Chiou–Chen's key broadcasting, 243
Denning–Sacco, 88
Denning–Sacco public key, 118
Diffie–Hellman, 51, 141–147, 204–219
EKE, 248, 250–253, 260–272
encrypted key exchange, *See* EKE
Girault, 188–189
Girault and Paillès, 185
Gong hybrid, 102
Gong key agreement, 100–101
Gong, Lomas, Needham and Saltzer (GLNS), 267–271, 277–278
Gong multiple server, 104–105
Gong–Wheeler key pre-distribution, 198
Günther identity-based, 186–188
Halevi–Krawczyk, 279–280
Helsinki, 119–120
Hirose–Yoshida, 236
Hirose–Yoshida key agreement, 180–181
IKE, 59, 174–178
Ingemarsson–Tang–Wong key agreement, 204–205
ISO-IEC
  9798-2, 77–78, 289
  9798-3, 110–113, 289
  11770-2, 84–86, 93–94, 290
  11770-3, 116–121, 195–196, 290
Jakobsson–Pointcheval, 193–194
Janson–Tsudik 2PKDP, 83
Janson–Tsudik 3PKDP, 97–98
Joux's tripartite, 215–216
Just–Vaudenay, 220

Petitioner Samsung Ex-1036, 0338

Just–Vaudenay–Song–Kim, 162–163
Katz–Ostrovsky–Yung, 258–260
KEA, 157
Kerberos, 57, 91–93
Klein–Otten–Beth, 220–221
Koyama–Ohta identity-based, 226–229
Kwon–Song, 278–279
Laih–Yen key broadcasting, 245
Lee–Sohn–Yang–Won, 282–283
Leighton–Micali key pre-distribution, 198
Lim–Lee key agreement, 146–147, 179–180
Mayer–Yung, 237–239
MQV, 159–160, 165, 290–292
MTI, 30, 147–157, 160, 162, 165, 185, 187, 189
Needham–Schroeder public key, 53–58, 61–64, 120–122
Needham–Schroeder shared key, 10, 87
Nyberg–Rueppel, 145–146, 282
Oakley, 168–172, 174, 196, 198
Octopus, *See* Becker–Wille's Octopus
Okamoto identity-based, 184–186, 188
Okamoto–Tanaka, 185
OKE, 274
Otway–Rees, 28–29, 64, 88–91, 268
  recursive, 64
PAK, 252–256, 261–262
PAK-R, 256
PAK-X, 261
PAK-Y, 262
Park key agreement, 192
PDM, 258
Perrig key agreement, 210–211
Photuris, 174
Pieprzyk–Li key agreement, 230–232
PPK, 255
Roe–Christianson–Wheeler, 276
Saeednia–Safavi-Naini identity-based, 229–230
Saeendia's variant of Günther's scheme, 189
SAKA, 258
SKEME, 172–174, 197–198, 279
SNAPI, 274–276
SPEKE, 256–258, 263–264
SPLICE, 113–115
SRP, 264–265
SSL, *See also* TLS, 57, 126
Steer–Strawczynski–Diffie–Wiener key agreement, 209–210
Steiner–Tsudik–Waidner key agreement, 206–209, 221–225
STS, 44–46, 166–168
three-party EKE, 266–272
TLS, 64, 124–126, 250
TMN, 57, 131–132
Tseng–Jan, 245
Tzeng–Tzeng, 232–233
Unified Model, 158–159, 162, 165
Wide-mouthed-frog, 57, 94–95
Wong–Chan, 195
Woo–Lam authentication, 78–80
Woo–Lam key transport, 99
Wu, 245
X.509, 122–124, 290
Yacobi, 160–161
Yacobi–Shmuely, 191–192
Yahalom, 64, 95–97
Yen–Liu, 280–281

Petitioner Samsung Ex-1036, 0339

General Index

AAPA, 65
Abadi, M., 1, 5, 31–32, 48–49, 59, 62, 79, 90, 108, 118, 133, 284
active attack, 25
Advanced Encryption Standard, 19
aggressive protocol, 169, 174
anonymity, 127–128, 171–172, 175, 241–242
asymmetric encryption, 16
authentication framework, 122, 290
authenticator, 71

backward secrecy, 204
BAN logic, 35, 59–62, 65, 95, 121
Bellare–Rogaway model, 66–71, 76, 98, 111, 119, 121, 134, 158, 163, 173, 185, 194, 225, 235, 237, 252–253, 258, 274
bootstrapping, 284
broadcast encryption, 241
Brutus, 57–58

Canadian attack, 112
canonical intensional specification, 46
cascade protocol, 106
CASPER, 56
certificate, 12, 24, 70, 108, 113, 122, 129, 139–140, 151, 167, 182–183, 187, 290
certificate manipulation, 30–31
challenge–response, 9, 82, 96, 128–129, 279
chosen ciphertext attack, 17
chosen plaintext attack, 17
chosen protocol attack, 31, 76–77
client puzzles, 28
complete graph, 227
confidentiality, 14
confounder, 268
connection depletion attacks, 27
conservative protocol, 170
consistency, 238–239
cookies, 27, 168, 170, 174
counters, 22
credit, 48–49, 139
cryptanalysis, 29
cyclic function, 214

data integrity, 15
data origin authentication, 15
Decision Diffie–Hellman (DDH) assumption, 142
denial of service, 24, 27–28, 168, 172, 174, 181, 199, 239
design principles, 31–32, 108–109, 117, 121
dictionary attack, 248
Diffie–Hellman assumption, 142
digital signature, 15, 18–19, 107, 289
Digital Signature Standard (DSS), 19, 125, 178
duplicate signature, 167, 181
dynamic group, 203–204, 208, 239

easy computation, 16
ElGamal encryption, 144–145
ElGamal signature, 129, 186, 261

Petitioner Samsung Ex-1036, 0340

elliptic curve, 141, 159, 162, 184, 190, 196, 199, 216, 255, 258, 288
encryption scheme, 16
entity authentication, 2, 15, 36–44, 46–47, 75–80, 110–115, 289, 292
ephemeral key, 50–51, 142–143, 216, 232, 237
exchanges, 218
explicit key authentication, 42
extensional properties, 46–47

fail–stop protocols, 28
FDR, 53–56
forward secrecy, 50–51, 70, 132–133, 137, 140, 143, 154–155, 204, 216, 220, 223, 228–230, 232, 234, 237, 245, 248
freshness, 21–23, 37, 40–41, 60, 84, 87, 92, 95–96, 101, 138, 202

Gong, L., 22–23, 28, 61, 65, 100–102, 198, 242, 267–270, 277–278, 283
good key, 40–41
group controller, 208
group manager, 235

handshake, 42
hard computation, 16
HMAC, 19, 175
honest insiders, 36
hybrid protocol, 13–14, 73, 87, 102
hyperelliptic curve, 196

idealisation, 60
identity-based protocols, 182–190, 225–230
IEEE P1363 standard, 19, 138, 158–159, 258, 287–288
implicit certificate, 183
implicit key authentication, 40
indistinguishability, 17, 66, 69, 71
inductive model, 63–65
insider attacks, 7, 36
intensional properties, 46–47, 75

key agreement, 13, 21, 41, 73, 100–101, 116, 125, 137–199, 204–272, 280, 290, 292
key broadcasting, 240–245
key compromise impersonation, 52, 154–155, 162, 164, 166
key confirmation, 12, 37, 42–45, 140, 163–164, 175, 202–203
key control, 138–139
key derivation function, 84, 138, 140, 151
key establishment, 36–38
key hierarchy, 239–240
key independence, 204
key integrity, 41
key-oriented goals, 40–41
key pre-distribution, 198–199, 235
key separation, 172
key translation, 79, 93, 104
key transport, 13, 87–99, 116–134, 235–240, 290, 292
knowledge of peer entity, 39

MAC, *see* message authentication code
malicious insiders, 36
man-in-the-middle attack, 142
manipulation detection code (MDC), 17
matching conversations, 45, 47, 75, 111
Menezes, A., 14, 30, 38, 40, 111, 120, 137, 140, 146, 150–151, 157–160, 164, 167, 179, 181, 237
message authentication code (MAC), 18, 74, 168, 175
mobile communications, 126–132, 190–194
model checking, 53–59
modes of operation, 19
modular proofs, 71
multiple servers, 104–106
multisignature, 237
Murϕ , 56–57, 126
mutual authentication, 39
mutual belief, 43

Needham, R., 1, 10, 31–32, 59, 65, 87, 90, 95, 108–109, 117–118, 121, 267
nonce, 9, 22, 32, 40, 60, 82, 96, 100, 171, 175, 278
non-malleability, 17, 57–58, 108, 114, 116–117, 119, 121–122, 259, 268, 277
non-repudiation, 15
NRL Analyzer, 58–59, 177

Petitioner Samsung Ex-1036, 0341

one-pass key establishment, 144–146
one-time pad, 15
one-way function, 17
oracle attack, 26, 76

P1363, *see* IEEE P1363 standard
pairings, 184, 216
partial forward secrecy, 50
partition attack, 251
passive attack, 24
password hardening, 284
penultimate authentication, 177
perfect forward secrecy, *see* forward secrecy
performance bounds for conference key agreement, 218
plaintext cipher-block chaining, 91
preplay, 23–25
principal, 2, 13
proof of knowledge, 232
protocol efficiency, 47–48
protocol interaction, 31
public key certificate, *see* certificate
public key encryption, 16
public key validation, 31
public password, 279

random oracle model, 67, 225, 235, 253, 258, 285
reflection, 24–26, 78, 177
repeated authentication, 92
replay, 8–12, 21–22, 24–25, 28–29, 32
resource depletion attacks, 27
responsibility, 48–49
round, 48, 218
RSA algorithm, 19, 107, 125, 131–132, 184, 188, 273–276

safe prime, 273
salt, 260
Schnorr signature, 261
secret certificate, 127
secret public key, 267
secret sharing, 20–21, 230–232, 244–245
security association, 174
self-certified key, 183, 188
semantic security, 17, 108, 268
session key, 2
Shoup simulation model, 69–71, 118, 253, 257, 275
signature with appendix, 19, 116
signature with message recovery, 19, 116, 145, 190
signcryption, 146
simple round, 218
SKIPJACK algorithm, 157
small subgroup attack, 144, 149–150, 252, 257
split certificate, 129
split-key, 129
stateless connection, 27
static Diffie–Hellman key, 142
static group, 203–204
strand space model, 62–63
strengthening passwords, 284
strong entity authentication, 39
symmetric encryption, 16
symmetric function, 204

theorem proving, 53, 58–65
threshold scheme, 20
ticket, 92
timestamp, 22–23, 32, 92, 100
traitor tracing, 241
triangle attack, 25, 153–154, 156, 161, 187
typing attack, 28–29, 81

undetectable on-line attack, 270
unilateral authentication, 39
unknown key-share, 139–140, 150–151, 156, 160, 166–168, 220
user, 13
user-oriented goals, 38–40

weak key, 29

XTR algorithm, 196, 255

Petitioner Samsung Ex-1036, 0342

Be the first to know
with the new online notification service

Springer Alert

You decide how we keep you up to date on new publications:

- Select a specialist field within a subject area
- Take your pick from various information formats
- Choose how often you'd like to be informed

And receive customised information to suit your needs

Register now

http://www.springer.de/alert

and then you are one click away from a world of computer science information!

Come and visit Springer's Computer Science Online Library

http://www.springer.de/comp

Springer

008939_001x_1c