US011178116B2

(12) **United States Patent**
O'Hare et al.

(10) **Patent No.: US 11,178,116 B2**
(45) **Date of Patent: Nov. 16, 2021**

(54) **SECURE DATA PARSER METHOD AND SYSTEM**

(71) Applicant: **Security First Corp.**, Rancho Santa Margarita, CA (US)

(72) Inventors: **Mark S. O'Hare**, Coto de Caza, CA (US); **Rick L. Orsini**, Flower Mound, TX (US); **Roger S. Davenport**, Campbell, TX (US); **Steven Winick**, Roslyn Heights, NY (US)

(73) Assignee: **Security First Corp.**, Rancho Santa Margarita, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **15/977,748**

(22) Filed: **May 11, 2018**

(65) **Prior Publication Data**

US 2018/0367509 A1 Dec. 20, 2018

Related U.S. Application Data

(60) Continuation of application No. 13/915,081, filed on Jun. 11, 2013, now Pat. No. 9,992,170, which is a (Continued)

(51) **Int. Cl.**
G06F 7/04 (2006.01)
H04L 29/06 (2006.01)
(Continued)

(52) **U.S. Cl.**
CPC ***H04L 63/0428*** (2013.01); ***G06F 16/22*** (2019.01); ***G06F 21/602*** (2013.01);
(Continued)

(58) **Field of Classification Search**
CPC H04L 63/04; H04L 63/0428; H04L 63/08; H04L 63/0823; H04L 63/0876;
(Continued)

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,453,074 A 6/1984 Weinstein
4,802,220 A 1/1989 Marker, Jr.
(Continued)

FOREIGN PATENT DOCUMENTS

AU 2004248616 6/2009
CN 1329417 A 1/2002
(Continued)

OTHER PUBLICATIONS

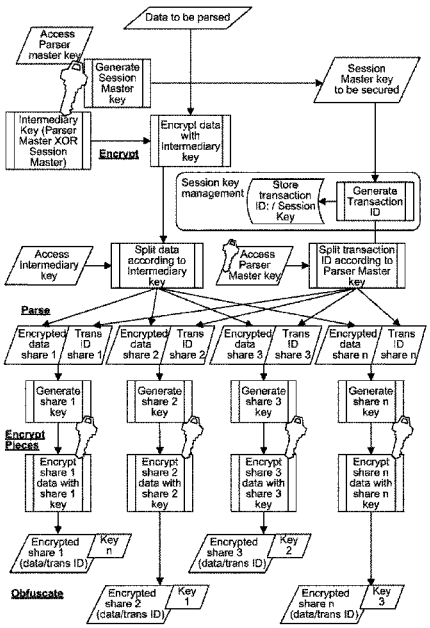
Barlas, "RSA's Security Showcase", Line56.com—The E-Business Executive Daily, Apr. 15, 2003.
(Continued)

Primary Examiner — Samson B Lemma

(57) **ABSTRACT**

A secure data parser is provided that may be integrated into any suitable system for securely storing and communicating data. The secure data parser parses data and then splits the data into multiple portions that are stored or communicated distinctly. Encryption of the original data, the portions of data, or both may be employed for additional security. The secure data parser may be used to protect data in motion by splitting original data into portions of data that may be communicated using multiple communications paths.

24 Claims, 68 Drawing Sheets



Related U.S. Application Data

division of application No. 13/468,383, filed on May 10, 2012, now Pat. No. 9,338,140, which is a continuation of application No. 11/258,839, filed on Oct. 25, 2005, now Pat. No. 8,266,438.

(60) Provisional application No. 60/718,185, filed on Sep. 16, 2005, provisional application No. 60/622,146, filed on Oct. 25, 2004.

(51) **Int. Cl.**

H04L 29/08	(2006.01)
G06F 16/22	(2019.01)
G06F 21/60	(2013.01)
G06F 21/62	(2013.01)
H04L 9/08	(2006.01)
H04L 9/32	(2006.01)
G06F 11/10	(2006.01)

(52) **U.S. Cl.**

CPC ***G06F 21/606*** (2013.01); ***G06F 21/62*** (2013.01); ***G06F 21/6218*** (2013.01); ***H04L 9/085*** (2013.01); ***H04L 9/3226*** (2013.01); ***H04L 9/3263*** (2013.01); ***H04L 63/04*** (2013.01); ***H04L 63/08*** (2013.01); ***H04L 63/0823*** (2013.01); ***H04L 63/0876*** (2013.01); ***H04L 67/108*** (2013.01); ***H04L 69/14*** (2013.01); *G06F 11/1092* (2013.01); *H04L 2209/80* (2013.01)

(58) **Field of Classification Search**

CPC H04L 9/085; H04L 9/3226; H04L 9/3263; H04L 2209/80; G06F 21/602; G06F 21/606; G06F 21/62; G06F 21/6218

USPC 726/5

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,924,513 A 5/1990 Herbison et al.
4,932,057 A 6/1990 Kolbert
5,010,572 A 4/1991 Bathrick et al.
5,016,274 A 5/1991 Micali et al.
5,051,745 A 9/1991 Katz
5,150,407 A 9/1992 Chan
5,163,096 A 11/1992 Clark et al.
5,239,659 A 8/1993 Rudeseal et al.
5,268,963 A 12/1993 Monroe et al.
5,375,244 A 12/1994 McNair
5,386,104 A 1/1995 Sime
5,450,099 A 9/1995 Stephenson et al.
5,485,474 A 1/1996 Rabin
5,524,073 A 6/1996 Stambler
5,603,003 A 2/1997 Akizawa et al.
5,615,269 A 3/1997 Micali
5,623,546 A 4/1997 Hardy et al.
5,623,549 A 4/1997 Ritter
5,642,508 A 6/1997 Miyazawa
5,666,414 A 9/1997 Micali
5,666,416 A 9/1997 Micali
5,666,514 A 9/1997 Cheriton
5,682,425 A 10/1997 Enari
5,703,907 A 12/1997 James
5,717,758 A 2/1998 Micall
5,748,735 A 5/1998 Ganesan
5,761,306 A 6/1998 Lewis
5,761,311 A 6/1998 Spelman et al.
5,768,382 A 6/1998 Schneier et al.
5,768,388 A * 6/1998 Goldwasser H04L 9/0894 380/286
5,768,519 A 6/1998 Swift et al.
5,790,677 A 8/1998 Fox et al.
5,823,948 A 10/1998 Ross, Jr. et al.
5,903,652 A 5/1999 Mital
5,903,882 A 5/1999 Asay et al.
5,910,987 A 6/1999 Ginter et al.
5,915,019 A 6/1999 Ginter et al.
5,915,024 A 6/1999 Kitaori et al.
5,917,913 A 6/1999 Wang
5,937,066 A 8/1999 Gennaro et al.
5,940,507 A 8/1999 Cane et al.
5,960,083 A 9/1999 Micali
5,966,444 A 10/1999 Yuan et al.
5,966,448 A 10/1999 Namba et al.
5,974,144 A 10/1999 Brandman
5,982,322 A 11/1999 Bickley et al.
5,987,232 A 11/1999 Tabuki
5,991,414 A 11/1999 Garay et al.
5,995,630 A 11/1999 Borza
6,009,173 A 12/1999 Sumner
6,009,177 A 12/1999 Sudia
6,023,508 A 2/2000 Bombard et al.
6,026,163 A 2/2000 Micali
6,058,188 A 5/2000 Chandersekaran et al.
6,061,790 A 5/2000 Bodnar
6,073,237 A 6/2000 Ellison
6,078,888 A 6/2000 Johnson, Jr.
6,084,968 A 7/2000 Kennedy et al.
6,088,454 A * 7/2000 Nagashima H04L 9/0836 380/277
6,092,201 A 7/2000 Turnbull et al.
6,094,485 A 7/2000 Weinstein et al.
6,118,874 A 9/2000 Okamoto et al.
6,134,550 A 10/2000 Van Oorschot et al.
6,182,214 B1 1/2001 Hardjono
6,192,472 B1 2/2001 Garay et al.
6,226,385 B1 5/2001 Taguchi et al.
6,229,894 B1 5/2001 Van Oorschot et al.
6,240,183 B1 5/2001 Marchant
6,240,187 B1 5/2001 Lewis
6,240,188 B1 5/2001 Dondeti et al.
6,256,737 B1 7/2001 Bianco et al.
6,260,125 B1 7/2001 McDowell
6,266,413 B1 7/2001 Shefi
6,269,432 B1 7/2001 Smith
6,289,455 B1 9/2001 Kocher et al.
6,289,509 B1 9/2001 Kryloff
6,292,568 B1 9/2001 Akins, III et al.
6,292,782 B1 9/2001 Weideman
6,301,659 B1 10/2001 Micali
6,307,940 B1 10/2001 Yamamoto et al.
6,314,409 B2 11/2001 Schneck et al.
6,324,650 B1 11/2001 Ogilvie
6,336,186 B1 1/2002 Dyksterhouse et al.
6,345,101 B1 2/2002 Shukla
6,345,314 B1 2/2002 Cole et al.
6,347,143 B1 2/2002 Goff et al.
6,356,941 B1 3/2002 Cohen
6,363,425 B1 3/2002 Hook et al.
6,363,481 B1 3/2002 Hardjono
6,381,331 B1 4/2002 Kato
6,385,318 B1 5/2002 Oishi
6,386,451 B1 5/2002 Sehr
6,411,716 B1 6/2002 Brickell
6,415,373 B1 7/2002 Peters et al.
6,424,718 B1 7/2002 Holloway
6,438,690 B1 8/2002 Patel et al.
6,446,204 B1 9/2002 Pang et al.
6,449,730 B2 9/2002 Mann et al.
6,453,416 B1 9/2002 Epstein
6,466,552 B1 10/2002 Haumont
6,473,858 B1 10/2002 Shimomura et al.
6,473,860 B1 10/2002 Chan
6,483,921 B1 11/2002 Harkins
6,505,216 B1 1/2003 Schutzman et al.
6,519,262 B1 2/2003 Stephens et al.
6,553,493 B1 4/2003 Okumura et al.
6,557,123 B1 4/2003 Wiencko, Jr. et al.
6,615,347 B1 9/2003 De Silva et al.
6,625,734 B1 9/2003 Marvit et al.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,631,201 B1 10/2003 Dickinson et al.
6,684,330 B1 1/2004 Wack et al.
6,687,375 B1 2/2004 Matyas, Jr. et al.
6,701,303 B1 3/2004 Dunn et al.
6,731,755 B1 5/2004 Cocks
6,785,768 B2 8/2004 Peters et al.
6,789,198 B1 9/2004 Chan
6,807,649 B1 10/2004 Murthy
6,816,970 B2 11/2004 Morgan et al.
6,819,766 B1 11/2004 Weidong
6,852,988 B2 2/2005 Li
6,853,988 B1 2/2005 Dickinson et al.
6,856,383 B1 2/2005 Vachris et al.
6,898,285 B1 5/2005 Hutchings et al.
6,915,436 B1 7/2005 Booth, III et al.
6,931,549 B1 8/2005 Ananda
6,947,557 B1 9/2005 Megiddo et al.
6,952,684 B2 10/2005 Toshikage et al.
6,959,383 B1 10/2005 Terada et al.
6,965,993 B2 11/2005 Baker
6,978,367 B1 12/2005 Hind et al.
6,981,141 B1 12/2005 Mahne et al.
7,003,531 B2 2/2006 Holenstein et al.
7,003,668 B2 2/2006 Berson et al.
7,035,405 B1 4/2006 Schwenk
7,039,189 B1 5/2006 Kienzle et al.
7,043,637 B2 * 5/2006 Bolosky H04L 9/0656 713/171
7,050,580 B1 5/2006 Ferre Herrero
7,050,583 B2 5/2006 Montgomery
7,058,605 B2 6/2006 Gupta
7,069,451 B1 6/2006 Ginter et al.
7,072,917 B2 7/2006 Wong et al.
7,076,651 B2 7/2006 Droge
7,085,854 B2 8/2006 Corente
7,096,494 B1 8/2006 Chen
7,107,385 B2 9/2006 Rajan et al.
7,111,173 B1 9/2006 Scheidt
7,111,303 B2 9/2006 Macchiano et al.
7,117,365 B1 10/2006 Rump et al.
7,133,845 B1 11/2006 Ginter et al.
7,140,044 B2 11/2006 Redlich et al.
7,143,289 B2 11/2006 Denning et al.
7,171,685 B2 1/2007 Batra et al.
7,174,385 B2 2/2007 Li
7,187,771 B1 3/2007 Dickinson et al.
7,188,203 B2 3/2007 Mowry et al.
7,191,252 B2 3/2007 Redlich et al.
7,191,410 B1 3/2007 Kruempelmann et al.
7,203,844 B1 4/2007 Oxford
7,203,871 B2 4/2007 Turner et al.
7,206,250 B2 4/2007 Groux
7,213,158 B2 5/2007 Bantz et al.
7,218,736 B1 5/2007 Nishimura et al.
7,222,062 B2 5/2007 Goud et al.
7,225,158 B2 5/2007 Toshikage et al.
7,228,337 B1 6/2007 Bornstein et al.
7,234,063 B1 6/2007 Baugher et al.
7,260,724 B1 8/2007 Dickinson et al.
7,260,727 B2 8/2007 Fougeroux et al.
7,277,958 B2 10/2007 Chung et al.
7,302,583 B2 11/2007 Forrest
7,304,990 B2 12/2007 Rajwan
7,313,825 B2 12/2007 Redlich et al.
7,322,047 B2 1/2008 Redlich et al.
7,337,320 B2 2/2008 Tada et al.
7,346,773 B2 3/2008 Cam-Winget et al.
7,346,925 B2 3/2008 Marcjan
7,349,539 B2 3/2008 Wee et al.
7,349,987 B2 3/2008 Redlich et al.
7,356,818 B2 4/2008 Carollo et al.
7,363,361 B2 4/2008 Tewari et al.
7,391,865 B2 6/2008 Orsini et al.
7,392,403 B1 6/2008 Munetoh
7,412,462 B2 8/2008 Margolus et al.
7,428,751 B2 9/2008 Oom Temudo de Castro et al.
7,428,754 B2 9/2008 Neumann et al.
7,440,953 B2 10/2008 Sidman
7,444,421 B2 10/2008 Katayama
7,464,152 B2 12/2008 Ishizaki et al.
7,469,340 B2 12/2008 Karamchedu et al.
7,472,105 B2 12/2008 Staddon et al.
7,478,427 B2 1/2009 Mukherjee et al.
7,499,905 B2 3/2009 Jaschek et al.
7,535,905 B2 5/2009 Narayanan et al.
7,546,427 B2 6/2009 Gladwin et al.
7,548,556 B1 6/2009 Wittenschlaeger
7,549,061 B2 6/2009 Nonaka et al.
7,552,482 B2 6/2009 Redlich et al.
7,577,689 B1 8/2009 Masinter et al.
7,584,285 B2 9/2009 Hudson et al.
7,587,368 B2 9/2009 Felsher
7,596,570 B1 9/2009 Emigh et al.
7,596,741 B2 9/2009 LaVigne et al.
7,609,687 B2 10/2009 Kobayashi et al.
7,623,809 B2 11/2009 Nakano et al.
7,627,125 B2 12/2009 Lumsden et al.
7,649,992 B2 1/2010 Raju et al.
7,657,760 B2 2/2010 Teramoto et al.
7,669,051 B2 2/2010 Redlich et al.
7,693,992 B2 4/2010 Watson
7,721,150 B2 5/2010 Belyakov et al.
7,721,344 B2 5/2010 Redlich et al.
7,730,543 B1 6/2010 Nath
7,743,261 B2 6/2010 Nonaka et al.
7,752,482 B2 7/2010 Gredone et al.
7,756,792 B2 7/2010 Hughes
7,765,411 B2 7/2010 Hennessey et al.
7,788,532 B2 8/2010 Kawamura
7,801,781 B2 9/2010 Olin et al.
7,831,995 B2 11/2010 Futoransky et al.
7,865,741 B1 1/2011 Wood et al.
7,869,597 B2 1/2011 Nakai et al.
7,916,871 B2 * 3/2011 Brookner H04L 9/085 380/279
7,921,288 B1 4/2011 Hildebrand
7,926,090 B2 4/2011 Blevins et al.
7,929,697 B2 4/2011 McNeely et al.
7,934,100 B2 4/2011 Fujimaki
7,940,932 B2 5/2011 Paksoy et al.
7,983,994 B2 7/2011 Hurry
8,009,830 B2 8/2011 Orsini et al.
8,041,875 B1 10/2011 Shah et al.
8,060,088 B2 11/2011 Ihattula
8,060,832 B2 11/2011 Kruempelmann et al.
8,095,800 B2 1/2012 Creary et al.
8,112,622 B2 2/2012 Qi
8,135,134 B2 3/2012 Orsini et al.
8,135,980 B2 3/2012 Dodgson et al.
8,151,333 B2 4/2012 Zhu et al.
8,155,322 B2 4/2012 Bellare et al.
8,189,769 B2 5/2012 Ramasamy et al.
8,190,905 B1 5/2012 Chang et al.
8,195,976 B2 6/2012 Rao et al.
8,208,897 B2 6/2012 Kaneko et al.
8,213,387 B2 7/2012 Xia et al.
8,250,643 B2 8/2012 Enomoto et al.
8,261,084 B2 9/2012 Nonaka et al.
8,261,265 B2 9/2012 Chen et al.
8,266,438 B2 9/2012 Orsini et al.
8,271,802 B2 9/2012 Orsini et al.
8,300,822 B2 10/2012 Nakai et al.
8,307,443 B2 11/2012 Wang et al.
8,332,638 B2 12/2012 Orsini et al.
8,341,115 B1 12/2012 Natanzon et al.
8,347,084 B2 1/2013 Tavernier et al.
8,352,941 B1 1/2013 Protopopov et al.
8,392,682 B2 3/2013 Dodgson et al.
8,396,895 B2 3/2013 Miloushev et al.
8,397,083 B1 3/2013 Sussland et al.
8,397,084 B2 3/2013 Ranade
8,397,288 B2 3/2013 Melvin, Jr. et al.
8,473,757 B2 6/2013 Gladstone et al.
8,578,076 B2 11/2013 Van der Linden et al.

(56) **References Cited**

U.S. PATENT DOCUMENTS

8,635,456 B2 1/2014 Fascenda
8,644,502 B2 2/2014 Orsini
8,755,322 B2 6/2014 Miettinen et al.
8,761,401 B2 6/2014 Sprunk et al.
8,788,803 B2 * 7/2014 Irvine G06F 21/6218 713/150
8,959,199 B2 2/2015 Peleg et al.
9,275,071 B2 3/2016 O'Hare
9,317,705 B2 4/2016 O'Hare
9,338,140 B2 5/2016 O'Hare
9,954,680 B1 4/2018 Machani
10,108,807 B2 10/2018 O'Hare
2001/0024502 A1 9/2001 Ohkuma et al.
2001/0050990 A1 12/2001 Sudia
2001/0051902 A1 12/2001 Messner
2002/0016773 A1 2/2002 Ohkuma et al.
2002/0016926 A1 2/2002 Nguyen et al.
2002/0023210 A1 2/2002 Tuomenoksa et al.
2002/0031230 A1 3/2002 Sweet et al.
2002/0032663 A1 3/2002 Messner
2002/0046359 A1 4/2002 Boden
2002/0071566 A1 6/2002 Kurn
2002/0075844 A1 6/2002 Hagen
2002/0080888 A1 6/2002 Shu et al.
2002/0087866 A1 7/2002 Berson et al.
2002/0129235 A1 9/2002 Okamoto et al.
2002/0129245 A1 9/2002 Cassagnol et al.
2002/0157007 A1 10/2002 Sashihara
2002/0162047 A1 10/2002 Peters et al.
2002/0172358 A1 11/2002 Hurich
2002/0178355 A1 11/2002 D'Sa et al.
2002/0178361 A1 11/2002 Genty et al.
2002/0184444 A1 12/2002 Shandony
2002/0191796 A1 12/2002 Muschenborn
2002/0194209 A1 12/2002 Bolosky
2003/0005094 A1 1/2003 Yuan et al.
2003/0023958 A1 1/2003 Patel et al.
2003/0051159 A1 3/2003 McCown et al.
2003/0055905 A1 3/2003 Nishiyama et al.
2003/0058274 A1 3/2003 Hill et al.
2003/0061481 A1 3/2003 Levine et al.
2003/0084020 A1 5/2003 Shu
2003/0084290 A1 5/2003 Murty et al.
2003/0084397 A1 5/2003 Peleg
2003/0110188 A1 6/2003 Howard et al.
2003/0126434 A1 7/2003 Lim et al.
2003/0167408 A1 9/2003 Fitzpatrick et al.
2003/0182435 A1 9/2003 Redlich et al.
2003/0188153 A1 10/2003 Demoff et al.
2003/0204605 A1 10/2003 Hudson et al.
2003/0236943 A1 12/2003 Delaney
2004/0022390 A1 2/2004 McDonald et al.
2004/0049687 A1 3/2004 Orsini et al.
2004/0078542 A1 4/2004 Fuller et al.
2004/0122960 A1 6/2004 Hall et al.
2004/0193878 A1 9/2004 Dillinger et al.
2004/0208170 A1 10/2004 Miyazaki et al.
2004/0215955 A1 10/2004 Tamai et al.
2004/0268148 A1 12/2004 Karjala et al.
2005/0050004 A1 3/2005 Sheu et al.
2005/0071631 A1 3/2005 Langer
2005/0111657 A1 * 5/2005 Lee H04L 9/085 380/28
2005/0132060 A1 6/2005 Mo et al.
2005/0160290 A1 7/2005 Moon et al.
2005/0198306 A1 9/2005 Palojarvi et al.
2005/0210243 A1 9/2005 Archard et al.
2005/0289218 A1 12/2005 Rothman et al.
2006/0020646 A1 1/2006 Tee et al.
2006/0046728 A1 3/2006 Jung et al.
2006/0075222 A1 4/2006 Moloney et al.
2006/0075225 A1 4/2006 Flynn et al.
2006/0078127 A1 4/2006 Cacayorin
2006/0090200 A1 4/2006 Oie
2006/0165060 A1 7/2006 Dua
2006/0282681 A1 12/2006 Scheidt et al.
2007/0006015 A1 1/2007 Rao et al.
2007/0033644 A1 2/2007 Thomas et al.
2007/0064704 A1 3/2007 Balay et al.
2007/0071234 A1 3/2007 Lagrange et al.
2007/0094272 A1 4/2007 Yeh
2007/0097885 A1 5/2007 Traversat et al.
2007/0122111 A1 5/2007 Yamamoto et al.
2007/0124584 A1 5/2007 Gupta
2007/0157025 A1 7/2007 Sastry et al.
2007/0160198 A1 7/2007 Orsini et al.
2007/0162744 A1 7/2007 Hoshino et al.
2007/0165865 A1 7/2007 Talvitie
2008/0016334 A1 1/2008 Kurapati et al.
2008/0037557 A1 2/2008 Fujita et al.
2008/0046752 A1 2/2008 Berger et al.
2008/0072035 A1 3/2008 Johnson et al.
2008/0126614 A1 5/2008 Ooi et al.
2008/0147821 A1 6/2008 Dietrich et al.
2008/0155252 A1 6/2008 Nambiar
2008/0170693 A1 7/2008 Spies et al.
2008/0183975 A1 7/2008 Foster et al.
2008/0183992 A1 7/2008 Martin et al.
2008/0232592 A1 9/2008 Lee et al.
2008/0240441 A1 10/2008 Kawakami
2008/0244277 A1 10/2008 Orsini et al.
2008/0270603 A1 10/2008 Berger et al.
2008/0294646 A1 11/2008 Chang et al.
2008/0297326 A1 12/2008 Chakraborty et al.
2009/0077379 A1 3/2009 Geyzel et al.
2009/0092252 A1 4/2009 Noll et al.
2009/0097657 A1 4/2009 Scheidt et al.
2009/0113203 A1 4/2009 Tsuge et al.
2009/0144202 A1 6/2009 Hurry
2009/0172417 A1 7/2009 Mikami et al.
2009/0177894 A1 7/2009 Orsini et al.
2009/0209232 A1 8/2009 Cha et al.
2009/0254572 A1 10/2009 Redlich et al.
2009/0254750 A1 10/2009 Bono et al.
2009/0288167 A1 11/2009 Freericks et al.
2009/0292919 A1 11/2009 England
2009/0292930 A1 11/2009 Marano et al.
2009/0300284 A1 12/2009 Murotani et al.
2009/0304004 A1 12/2009 Huynh Van et al.
2009/0305900 A1 12/2009 Belouchi et al.
2010/0005028 A1 1/2010 Hartley et al.
2010/0046750 A1 2/2010 Asfour
2010/0054474 A1 3/2010 Schneider
2010/0077252 A1 3/2010 Siewert et al.
2010/0095354 A1 4/2010 Uchida et al.
2010/0125730 A1 5/2010 Dodgson et al.
2010/0150341 A1 6/2010 Dodgson et al.
2010/0153703 A1 6/2010 Dodgson et al.
2010/0154053 A1 6/2010 Dodgson et al.
2010/0158244 A1 6/2010 Sun
2010/0162001 A1 6/2010 Dodgson
2010/0162004 A1 6/2010 Dodgson et al.
2010/0191958 A1 7/2010 Chen
2010/0223613 A1 9/2010 Schneider
2010/0268966 A1 10/2010 Leggette et al.
2010/0283734 A1 11/2010 Hallbauer et al.
2010/0293380 A1 11/2010 Wiseman et al.
2010/0299313 A1 11/2010 Orsini et al.
2010/0306578 A1 12/2010 Thornton et al.
2010/0318812 A1 12/2010 Auradkar et al.
2010/0332617 A1 12/2010 Goodwin et al.
2011/0085667 A1 4/2011 Berrios et al.
2011/0093113 A1 4/2011 Sager et al.
2011/0125849 A1 5/2011 Boyd et al.
2011/0135094 A1 6/2011 Muto et al.
2011/0145413 A1 6/2011 Dawson et al.
2011/0179214 A1 7/2011 Goggin et al.
2011/0179271 A1 7/2011 Orsini et al.
2011/0179287 A1 7/2011 Orsini et al.
2011/0246766 A1 10/2011 Orsini et al.
2011/0246817 A1 10/2011 Orsini et al.
2011/0296440 A1 12/2011 Laurich
2012/0042034 A1 2/2012 Goggin et al.
2012/0072723 A1 3/2012 Orsini et al.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2012/0166576	A1	6/2012	Orsini et al.
2012/0166818	A1	6/2012	Orsini et al.
2012/0179916	A1	7/2012	Staker et al.
2012/0198338	A1	8/2012	Flint et al.
2012/0221854	A1	8/2012	Orsini et al.
2012/0221855	A1	8/2012	Orsini et al.
2012/0221856	A1	8/2012	Orsini et al.
2012/0222134	A1	8/2012	Orsini et al.
2012/0226904	A1	9/2012	Orsini et al.
2012/0255034	A1	10/2012	Orsini et al.
2012/0255035	A1	10/2012	Orsini et al.
2012/0266231	A1	10/2012	Spiers et al.
2013/0042105	A1	2/2013	Orsini et al.
2014/0201539	A1	7/2014	Astigarraga
2015/0378842	A1	12/2015	Tomlinson

FOREIGN PATENT DOCUMENTS

CN	1347225	A	5/2002
CN	1607762	A	4/2005
CN	1613111	A	5/2005
CN	101375284	A	2/2009
EP	346180		12/1989
EP	354774		2/1990
EP	0485090	A2	5/1992
EP	636259		2/1995
EP	793367		9/1997
EP	0798892	A2	10/1997
EP	0821504		1/1998
EP	0851629	A2	7/1998
EP	0862301		9/1998
EP	1011222		6/2000
EP	1239384	A2	9/2002
EP	2309700	A1	4/2011
FR	2632469		12/1989
GB	2237670	A	5/1991
JP	04297157	A	10/1992
JP	2003/132234		5/2003
JP	2004-185573		7/2004
JP	2005/136739		5/2005
JP	2005-244379	A	9/2005
JP	2005-267520		9/2005
JP	2006-174152	A	6/2006
JP	2007-334836		12/2007
JP	2008-103936		5/2008
JP	2008/250369		10/2008
JP	2008-250931		10/2008
RU	2124814		1/1999
WO	WO-98/47091	A1	10/1998
WO	WO-99/19845	A1	4/1999
WO	WO-99/46720	A1	9/1999
WO	WO-99/65207	A1	12/1999
WO	WO-00/36786	A1	6/2000
WO	WO-00/49766	A1	8/2000
WO	WO-00/76118	A1	12/2000
WO	WO-00/79367	A1	12/2000
WO	WO-01/22201	A1	3/2001
WO	WO-01/22319	A1	3/2001
WO	WO-01/22322	A2	3/2001
WO	WO-01/22650	A2	3/2001
WO	WO-01/22651	A2	3/2001
WO	WO-2001/054370		7/2001
WO	WO-02/21283	A1	3/2002
WO	WO-02/21761	A2	3/2002
WO	WO-02/062032		8/2002
WO	WO-03/036538		5/2003
WO	WO-2004/001561	A2	12/2003
WO	WO-04/111791	A2	12/2004
WO	WO-06/047694	A1	5/2006
WO	WO-2007/078846		7/2007
WO	WO-08/054406	A2	5/2008
WO	WO-08/070167	A1	6/2008
WO	WO-08/127309	A2	10/2008
WO	WO-08/142440	A1	11/2008
WO	WO-09/035674	A1	3/2009
WO	WO-09/089015	A1	7/2009
WO	WO-09/105280	A2	8/2009
WO	WO-2010/057181	A2	5/2010
WO	WO-2010/135412	A2	11/2010
WO	WO-2011/068738	A2	6/2011
WO	WO-2011/123692	A2	10/2011
WO	WO-2011/123699	A2	10/2011

OTHER PUBLICATIONS

Brainard, "A New Two-Server Approach for Authentication with Short Secrets" RSA Laboratories, Bedford, MA (13 pages).

Cachin, "On-Line secret Sharing," Cryptography and Coding. IMA Conference, Proceedings, Dec. 18, 1995, pp. 190-198, XP002137681.

Chan et. al., "Distributed Server Networks for Secure Multicast", GLOBCOM'01: IEEE Global Telecommunications Conference (IEEE, Piscataway, NJ) 3:1974-1978 (2001).

Chan et. al., "Distributed Servers Approach for Large-Scale Multicast", IEEE Journal on Selected Areas in Communications (IEEE, Piscataway, NJ) Oct. 2002, 20(8):1500-1510.

Crescenzo et al., "Non-Interactive and Non-Malleable Commitment," Proceedings of the 30th Annual ACM Symposium on Theory of Computing. Dallas, TX, May 23-26, 1998, [Proceedings of the 30th Annual ACM Symposium on Theory of Computing], New York, NY: ACM, US, pp. 141-150; XP000970902; ISBN: 978-0-89791-962-3.

Damgard et. al., "Non-interactive and Reusable Non-malleable Commitment Schemes," ACM STOC '03; pp. 427-428; Jun. 9-11, 2003.

Decru Unveils Security Appliances for Storage Networks; Decru DataFort (TM) Security Alliances Protect SAN and NAS Environments with Wire-Speed Encryption and Transparent Deployment, PR Newswire (PR Newswire Association. Inc.), Oct. 14, 2002.

Dierks, "The Transport Layer Security Protocol," RFC 4346m Version 1.1, Apr. 2006, 86 pages.

Doyle, "RSA Splits Data to Stop Hackers", vnunet.com, Apr. 16, 2003.

Easter et al. "S/390 parallel enterprise server CMOS cryptographic coprocessor," IBM Journal of Research and Development, International Business Machines Corporation, New York, NY, US, vol. 43, No. 5, Jan. 1, 1999, pp. 761-776, XP002335589, ISSN: 0018-8646.

Fisher, "RSA Looks to Lock Down Personal Data", eWeek—Enterprise News & Reviews, Apr. 14, 2003.

Ganger et al., "PASIS: A Distributed Framework for Perpetually Available and Secure Information Systems, Final Technical rept. Jun. 1999-Dec. 2003," (Jul. 1, 2005),pp. 1-302, XP55011444, Retrieved from the Internet: URL:http://www.dtic/mil/cgi-bin/GetTRDoc?AD=ADA436245&Location-U2&doc=GetTRDoc.pdf (retrieved 0n Nov. 7, 2011].

Ganger et al., "Survivable storage systems," DARPA Information Survivability Conference & Exposition II, 2001. DISC EX '01. Proc. Jun. 12-14, 2001, Piscataway, NJ, USA, IEEE, vo. 2, pp. 184-195, XP010548746.

Garay et. al., "Secure distributed storage and retrieval," Theoretical Comput. Sci., 243(1-2):363-389, Jul. 2000.

Gibson, "Opinion", eWeek—Enterprise News & Reviews, Apr. 14, 2003.

Grant et. al., "Secret Sharing and Splitting", (White Paper) Notre Dame, Indiana, Dec. 16, 2002.

Hand et al., Spread Spectrum Storage with Mnemosyne, 2003, Retrieved from the Internet <URL: springerlink.com/content/9vdp5b40ep2pjvba/>, pp. 1-5 as printed.

Haniotakis et al., "Security Enhancement Through Multiple Path Transmission in Ad Hoc Networks," IEEE Intl. Conference on Communications, Jun. 20-24, 2004, 5 pgs.

Horne et al., Escrow services and incentives in Peer-to-Peer Networks, 2001, Retrieved from the Internet URL:dl.acm.org.citation.cfm?id=501168, pp. 1-10 as printed.

Hunter, "Simplifying PKI Usage Through a Client-Server Architecture and Dynamic Propagation of Certificate Paths and Repository Addresses", Proceedings 13th International Workshop on Database and Expert Systems Applications (IEEE Computer Soc., Los Alamitos, CA), Sep. 2-6, 2002, p. 505-510.

(56) **References Cited**

OTHER PUBLICATIONS

Johnson, "MLS-Net and Secure Parser: A New Method for Securing and Segregating Network Data" [Online] Jul. 8, 2007"XP002582437 5th International Symposium on Risk Management and Informatic: WMS12007, Orlando Florida, USA.
Kent, "Security Architecture for the Internet Protocol," RFC 2401, Nov. 1998, 67 pages.
Klensin, J., "Simple Mail Transfer Protocol; rfs5321.txt," Simple Mail Transfer Protocol; RFC5321.TXT, Internet Engineering Task Force, IETF; Standard, Internet Society (ISOC) 4, Rue Des Falaises CH—1205 Geneva, Switzerland, XP015060297 (Oct. 2008).
Krawczyk, "Distributed Fingerprints and Secure Information Dispersal," 12th ACM, Symposium on Principles on Distributed Computing, Ithaca, NY, ACM 0-89191-613-1/93/0008/0207, 1993, pp. 207-218.
Krawczyk, "Secret sharing made Short," IBM T.J. Watson Research Center, [Online] 1998, retrieved from the Internet: URL:http://www.cs.cornell.edu/courses/cs754/2001fa/secretshort.pdf> [retrieved on Nov. 24, 2008].
Kroç, Erol, "Access Control in Peer-to-Peer Storage Systems," Oct. 1, 2006, XP55036196, Retrieved from the Internet: URL:ftp://ftp.tik.ee.ethz.ch/pub/studfents/2006-So/MA-2006-10.pdf [retrieved on Aug. 23, 2012] p. 61-62.
Kubiatowicz et al., OceanStore: an architecture for global-scale persistent storage, Retrieved from the Internet <URL: dl.acm.org/citation.cfm?id=356989.357007>, pp. 1-12 as printed.
Lancope Announces Stealthwatch 3.0 for Enhanced Enterprise-Wide Security and Improved Manageability, Business Wire (Newswire Association, Inc.), Apr. 14, 2003.
Loutrel et. al. "An EAP-BT Smartcard for Authentication in the Next Generation of Wireless Communications", Conference on Network Control and Engineering for QoS Security and Mobility (Kluwer Academic Publishers, Norwell, MA) Oct. 23-25, 2002, pp. 1-4-114.
Mayer et. al., "Generalized Secret Sharing and Group-Key Distribution Using Short Keys," Compression and Complexity of Sequences 1997, Proceedings Salerno, Italy, Jun. 11-13, 1997, Los Alamitos, CA, USA, IEEE Comput,. SOC, US, Jun. 11, 1997, pp. 30-44, XP010274905, ISBN: 978-0-8186-8132-5.
McNamara, "Strong Crypto Freeware" (Secret Sharer Version 1.0) Jul. 11, 1995.
Menezes, "Handbook of Applied Cryptography, Ch. 7 Block Ciphers" 1997, pp. 223-271 CRC Press LLC, XP001525007.
Mitchell, "Making Serial Number Based Authentication Robust Against Loss of State", Operating Systems Review, 34:3, pp. 1-5, Nov. 24, 1999.
Myers et. al., "A secure, publisher-centric Web caching infrastructure" In: INFOCOM 2001 Proceedings. IEEE Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies [online], vol. 3 p. 1235-1243. Published Apr. 22, 2001. [retrieved on Jul. 8, 2008]. Retrieved from the internet <URL: http://people.ischool.berkeley.edu/-chuang/pubs/gemini.pdf>.
Nightingale, The New Secret-Splitting Technology from RSA . . . NGBK DS 0403 http://developer.rsasecurity.com/labs/nightingale/developer.rsasecurity.com/labs/nightingale/files/nightingale-brochure.pdf.
Pattam, "Enhancing Security in 802.11 and 802.1X Networks with Intrusion Detection," Jan. 20, 2006, Univ. of New Orleans Theses and Dissertations, Paper 1034, 111 pages.
Rabin, "Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance," Journal of the Association for Computing Machinery, vol. 36, No. 2, pp. 335-348, Apr. 1989.
Rivest, "All-Or-Nothing Encryption and the Package Transform," Proc. of the 4th Intl. Workshop on Fast Software Encryption (1997), 9 pgs.
RSA SureFile: Software Powered by PKZIP . . . BSSF DS 0103 Authorized Reseller: Technical Specifications Platforms Microsoft® Windows® 98 Second Edition ME NT 4.0 Workstation SP6A 2000 Professional SP2 . . . www.rsasecurity.com/products/bsafe/datasheets/BSSF_DS_103.pdf.
Savage, "RSA Unveils Nightingale Technology", CRN.com, Apr. 14, 2003.
Sengar et al., "MTPSec: Customizable Secure MTP3 Tunnels in the SS7 Network," Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, 8 pages (2005).
Shamir, "How to Share a Secret," Communications of the Association for Computing Machinery, ACM, New York, NY, US, vol. 22, No. 11, Nov. 1, 1979, pp. 1-4, XP002241399; ISSN: 0001-0782.
Shin et. al., "Design a Working Model of Secure Data Transfer Using a Data Mart", Proceedings of the ISCA 14th International Conference Computer Applications in Industry and Engineering (ISCA, Cary, NC) Nov. 27-29, 2001, p. 66-69.
Tactilesense TM White Paper—A Breakthrough in Fingerprint Authentication, Ethentica, Inc. by Security First Corporation, Jan. 2003.
Trustengine TM White Paper—Enthentication Services, Secure Storage and Authentication Solutions, Ethentica, Inc. by Security First Corporation, Jun. 2002.
Vijayan, "RSA unveils Management, Encryption Products", Computerworld, Apr. 15, 2003.
Waldman et al., "Publius: A robust, tamper-evident, censorship-resistant web publishing sytem," Proceedings of the 9th USENIX Security Symposum, Aug. 2000.
Waters, "RSA Integrates ID Management; discloses 'Nightingale'", ADTmag.com, Apr. 21, 2003.
Office Actions and Replies issued for U.S. Appl. No. 13/468,383.
Office Actions and Replies issued for U.S. Appl. No. 13/468,428.
Office Actions and Replies issued for U.S. Appl. No. 13/468,450.
Office Actions and Replies issued for U.S. Appl. No. 13/468,523.
Office Actions and Replies issued for U.S. Appl. No. 13/468,605.
Office Actions and Replies issued for U.S. Appl. No. 13/468,562.
Office Actions and Replies issued for U.S. Appl. No. 13/468,584.
Xu, "Locality Driven Key Management Architecture for Mobile Ad-hoc Networks," 2004 IEEE International Conference on Mobile Ad-hoc and Sensor Systems, pp. 436-446.
Nacht, "The Spectrum of Modern Firewalls," Computers & Security, 16 (1997), pp. 54-56.
Shaw et al (eds.), Handbook on Electronic Commerce, Conway, Chapter 31, Springer-Verlag, Berlin Heidelbert, pp. 649-690 (2000).
Mondal, "Mobile IP," Chapter 7, Fluwer Academic/Plenum Publishers, New York, pp. 173-202 (2003).
Rosen, "BGP/MPLS VPNs," Network Working Group, RFC 2547, Mar. 1999, 25 pages.
Tewari et al.; High Availability in Clustered Multimedia Servers; 1996; Retrieved from the Internet <URL:ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=492215>; pp. 1-10 as printed.
Wells; The OceanStore Archive: Goals, Structures, and Self-Repair; 2001; Retrieved from the Internet <URL: oceanstore.org/publications/papers/pdf/>; pp. 1-20 as printed.
Wylie et al.; Selecting the right data distribution scheme for a survivable storage system; 2001; Retrieved from the Internet <URL: repository.cmu./edu/compsci/2164/?utm_source=repository.cmu.edu%2Fcompsci%2F2164&utm_medium=PDF&utm_campaign=PDFCoverPages.; pp. 1-25 as printed.
Buckhard et al., MDP Disk Array Storage System Reliability; 1992; Retrieved from the Internet <URL:ieexplore.ieee.org/xpls/abs_all.jsp?arnumber=627346&tag=1>; pp. 1-26 as printed.
Kim et al., "Secure Group Key Management for Storage Area Networks," IEEE Communications Magazine, Aug. 2003, pp. 92-99.
Bendix, "RAID and Informix Databases," White Paper, Advanced Technology Group, Jun. 1995; Retrieved from the Internet <URL: informix.com.ua/articles/raid/raid.htm>; retrieved from the internet on Dec. 3, 2014, pp. 1-15 as printed.
D'Arco et al.: "Fault Tolerant and distributed Broadcast Encryption"; 2003: Retrieved from the Internet<URL:link.springer.com/chapter/10.1007/3-540-36563-x18#page-1>; pp. 1-18 as printed.
Schwartz, Richard; "Using field encryption in applications"; 2001; Retrieved from the Internet<URL:ibm.com/developerworks/lotus/library/IS-filed_encryption/>; pp. 1-16 as printed.
Goodson et al., "Efficient consistency for erasure-coded data via versioning servers," Retrieved from the Internet<URL:oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA461126> pp. 1-25, Mar. 2003.

(56) **References Cited**

OTHER PUBLICATIONS

Long et al., "Swift/RAID: A Distributed RAID System," Retrieved from the Internet<URL:http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.7183>pp. 1-20 (1994).
Guo et al. "Design and implementation of the KioskNet System", Information and Communication Technologies and Development, 2007. ICTD 2007. International Conference on, Dec. 31, 2007 (Dec. 31, 2007) pp. 264-281 [retrieved on Nov. 26, 2015].
Chen, et al., "RAID: High-Performance, Reliable Secondary Storage" Retrieved from the internet, <URL:https://web.archive.org/web/20040721062927/http://meseec.ce.rit.edu/eecc722-fall2002/papers/io/3/chen94raid.pdf> 26:145-184 (1994).
Plank, James A., "A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems," Retrieved from the Internet <URL: citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.160.4742&rep=rep1&type=pdf>; pp. 1-18 as printed (1997).
Office Actions and Replies for U.S. Appl. No. 11/258,839.
Office Actions and Replies for U.S. Appl. No. 12/148,365.
Office Actions and Replies for U.S. Appl. No. 12/272,012.
Office Actions and Replies for U.S. Appl. No. 12/336,558.
Restriction Requirement dated Sep. 29, 2011 for U.S. Appl. No. 12/336,559.
Office Action dated Nov. 28, 2011 for U.S. Appl. No. 12/336,562.
Office Action dated Sep. 27, 2011 for U.S. Appl. No. 12/336,568.
Office Action dated Nov. 8, 2011 for U.S. Appl. No. 12/342,379.
Office Action dated Sep. 14, 2011 for U.S. Appl. No. 12/342,500.
Office Action dated Oct. 7, 2011 for U.S. Appl. No. 12/342,523.
Office Actions dated Oct. 6, 2011 for U.S. Appl. No. 12/342,547.
Office Action dated Nov. 10, 2011 for U.S. Appl. No. 12/342,575.
Office Actions and Replies for U.S. Appl. No. 12/342,610.
Office Action dated Nov. 23, 2011 for U.S. Appl. No. 12/342,636.
Office Actions and Replies for U.S. Appl. No. 12/346,578.
Office Actions and Replies for U.S. Appl. No. 13/024,791.
Office Action dated Jun. 11, 2012 for U.S. Appl. No. 13/371,361.
Office Actions and Replies for U.S. Appl. No. 13/371,363.
Office Actions and Replies for U.S. Appl. No. 13/371,364.

* cited by examiner

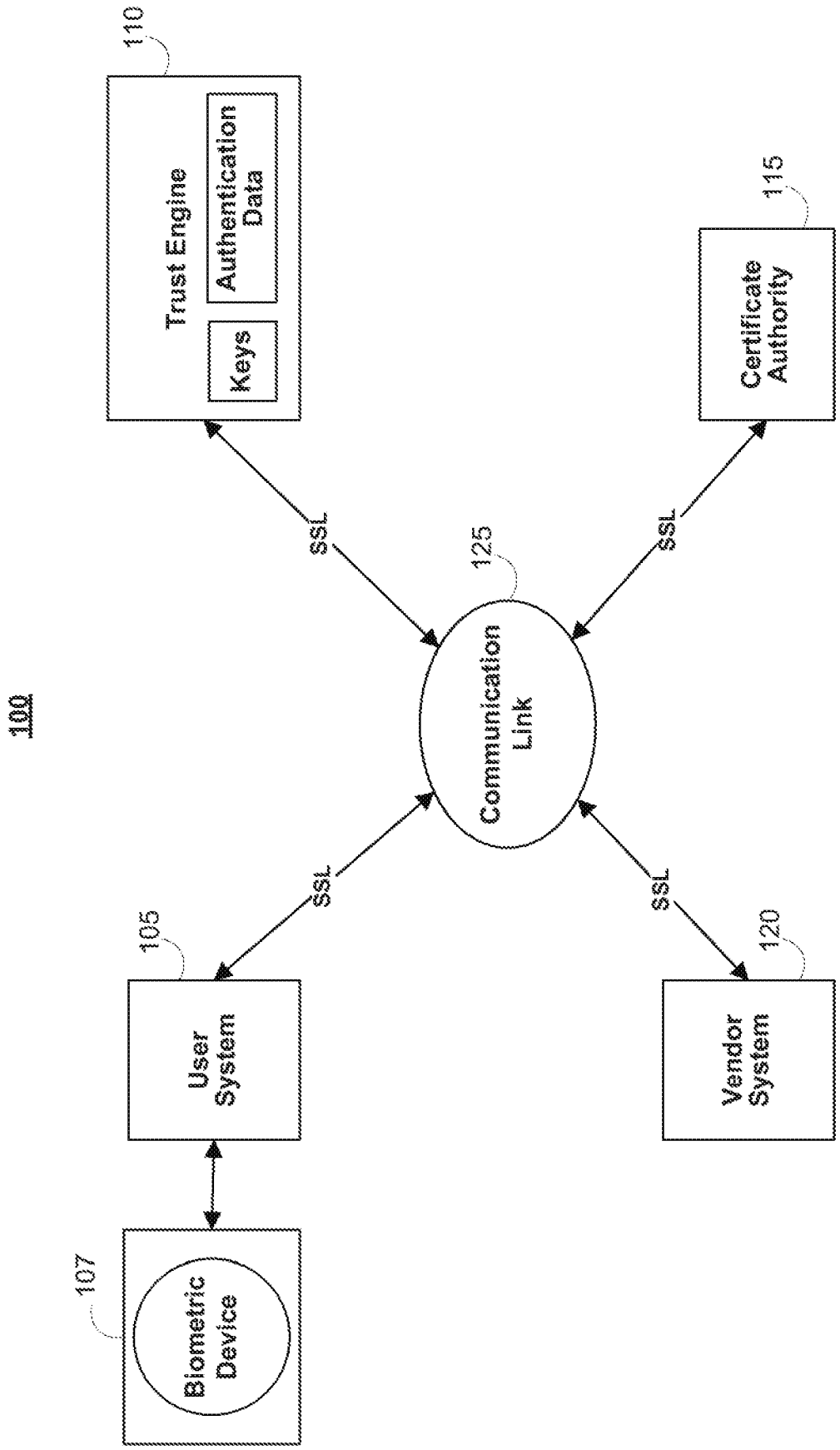

100
110
Trust Engine
Keys
Authentication Data
115
Certificate Authority
SSL
SSL
125
Communication Link
SSL
SSL
105
User System
120
Vendor System
107
Biometric Device

FIG. 1

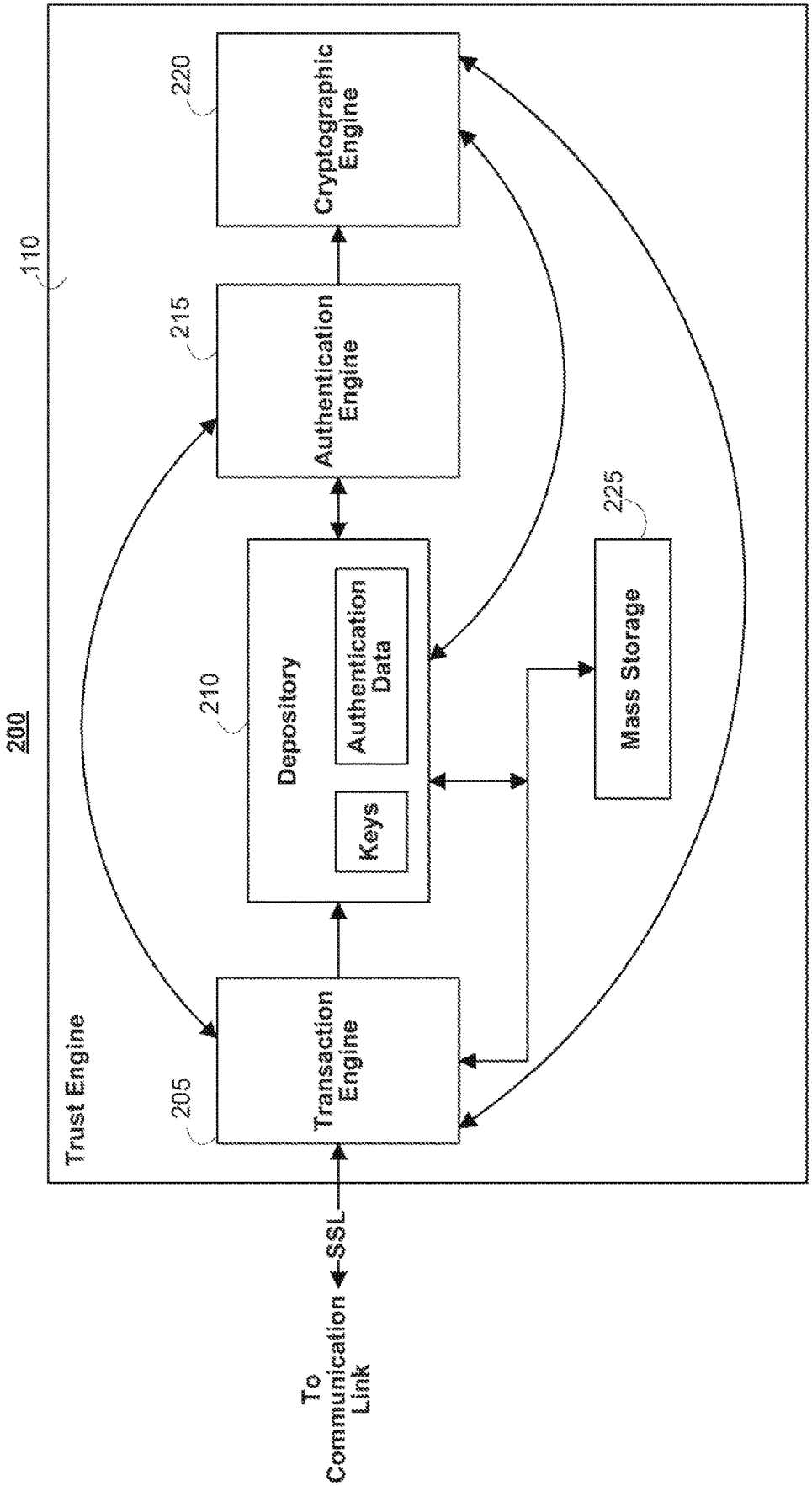

200
110
Trust Engine
205
Transaction Engine
210
Depository
Keys
Authentication Data
215
Authentication Engine
220
Cryptographic Engine
225
Mass Storage
SSL
To Communication Link

FIG. 2

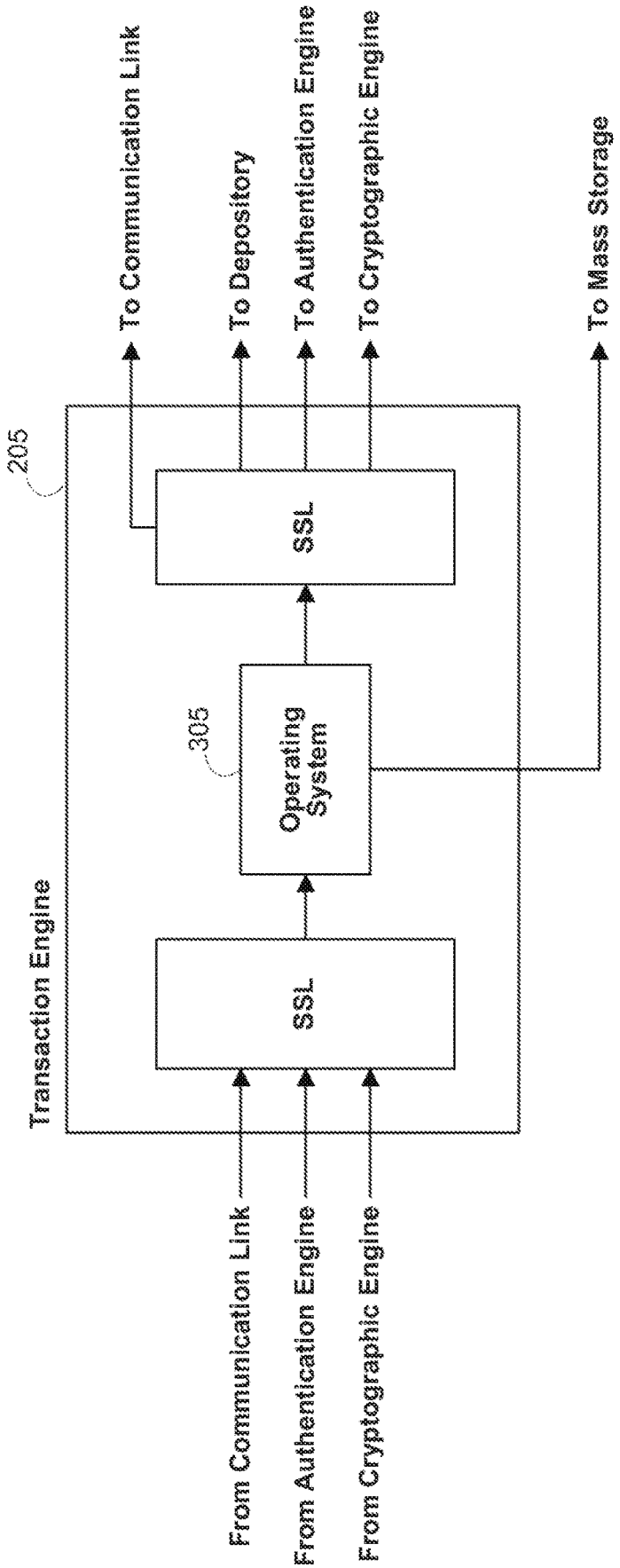

Transaction Engine
205
From Communication Link
From Authentication Engine
From Cryptographic Engine
SSL
305
Operating System
SSL
To Communication Link
To Depository
To Authentication Engine
To Cryptographic Engine
To Mass Storage

FIG. 3

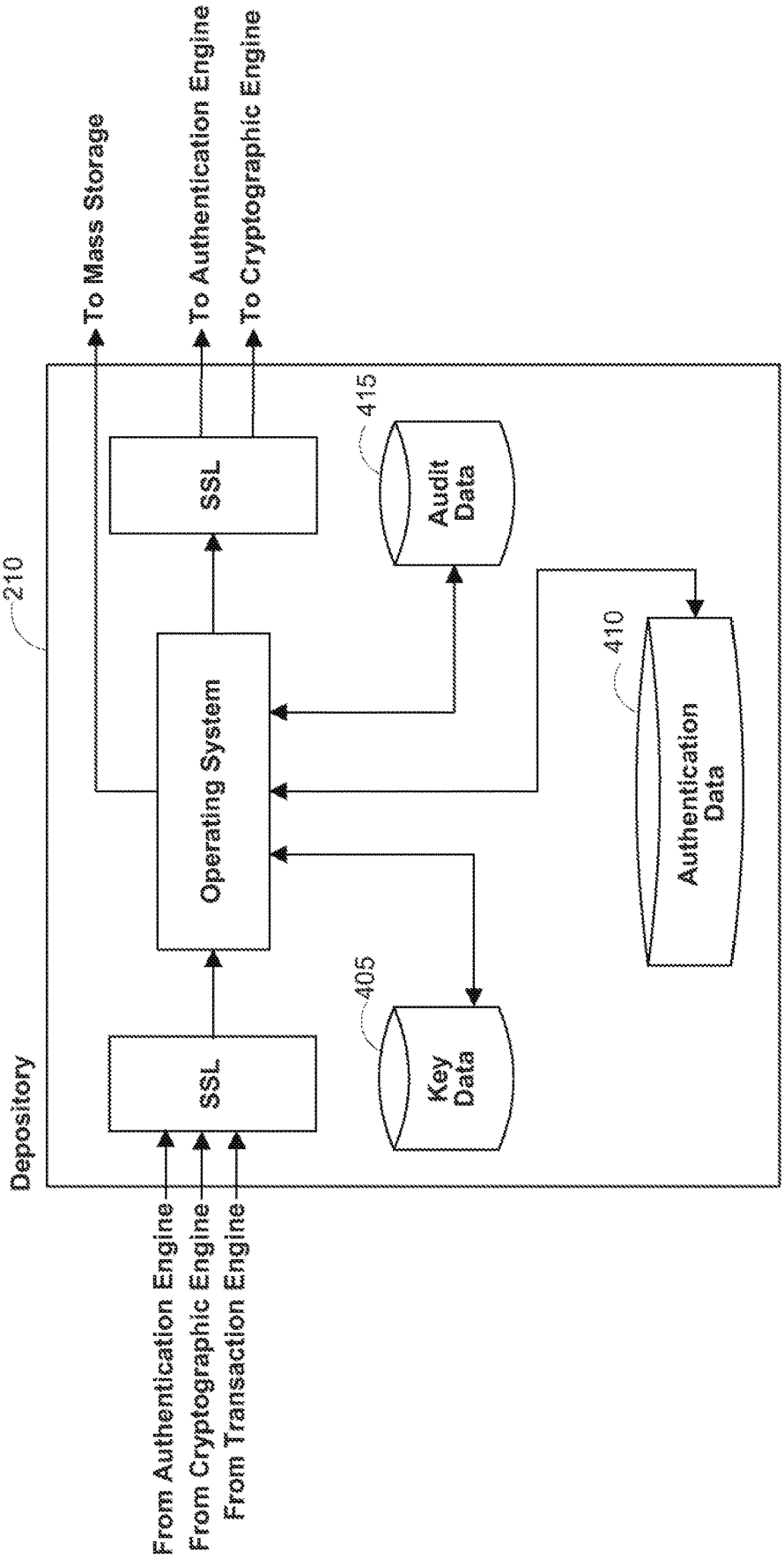

To Mass Storage
To Authentication Engine
To Cryptographic Engine
210
Depository
SSL
SSL
Operating System
415
Audit Data
405
Key Data
410
Authentication Data
From Authentication Engine
From Cryptographic Engine
From Transaction Engine

FIG. 4

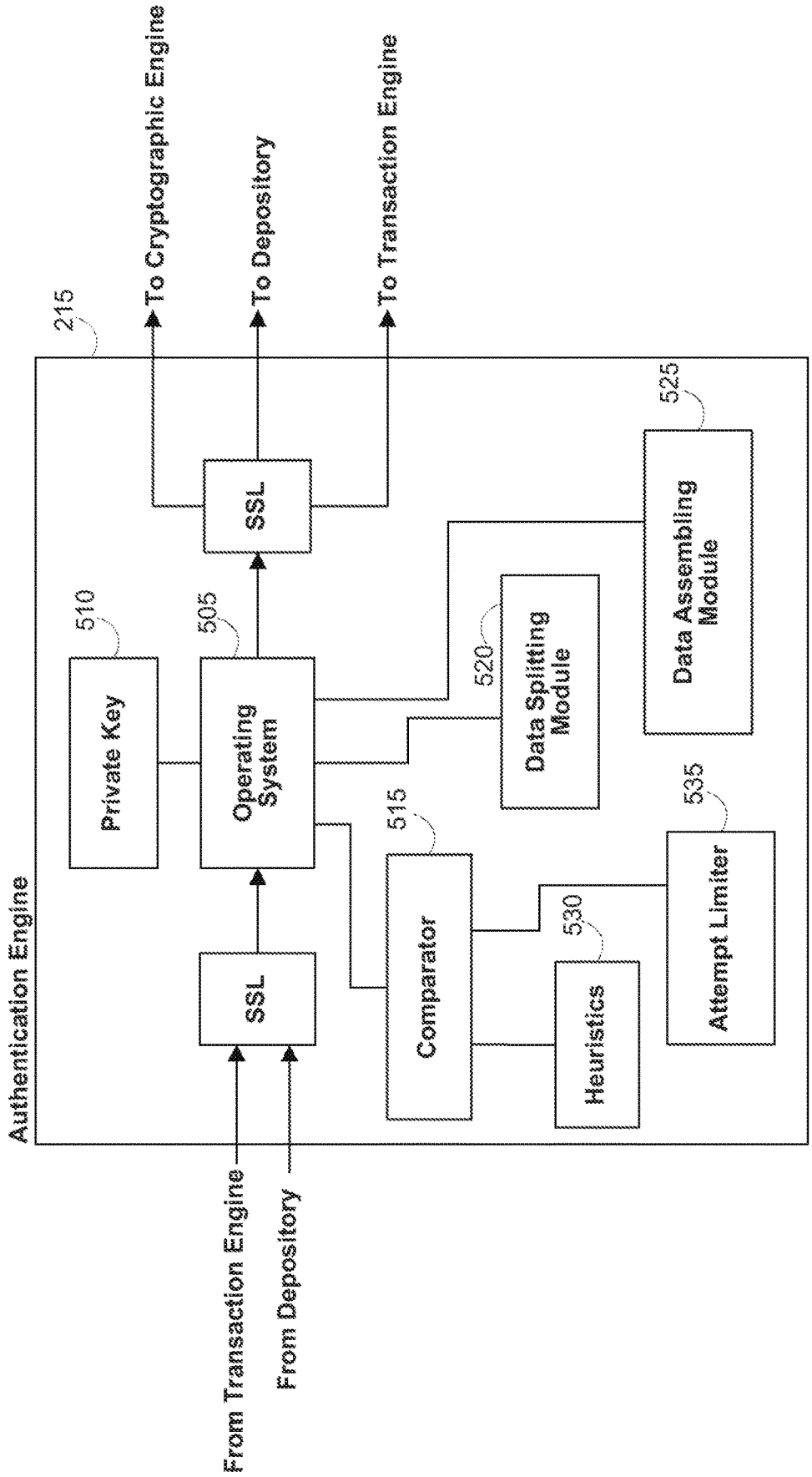

Authentication Engine
215
From Transaction Engine
From Depository
SSL
Private Key
510
Operating System
505
Comparator
515
Heuristics
530
Attempt Limiter
535
Data Splitting Module
520
Data Assembling Module
525
SSL
To Cryptographic Engine
To Depository
To Transaction Engine

FIG. 5

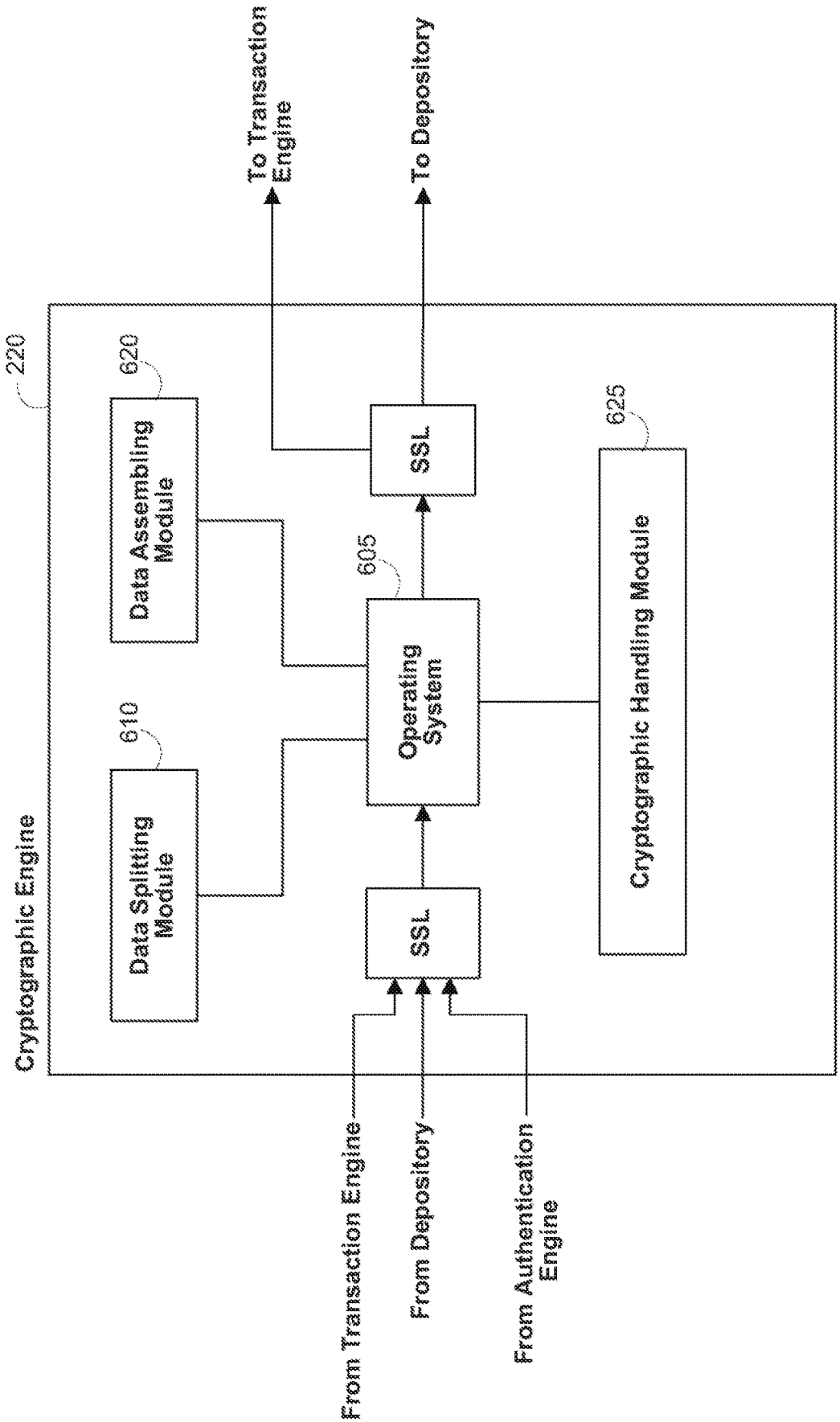

Cryptographic Engine
220
Data Splitting Module
610
Data Assembling Module
620
SSL
SSL
Operating System
605
Cryptographic Handling Module
625
From Transaction Engine
From Depository
From Authentication Engine
To Transaction Engine
To Depository

FIG. 6

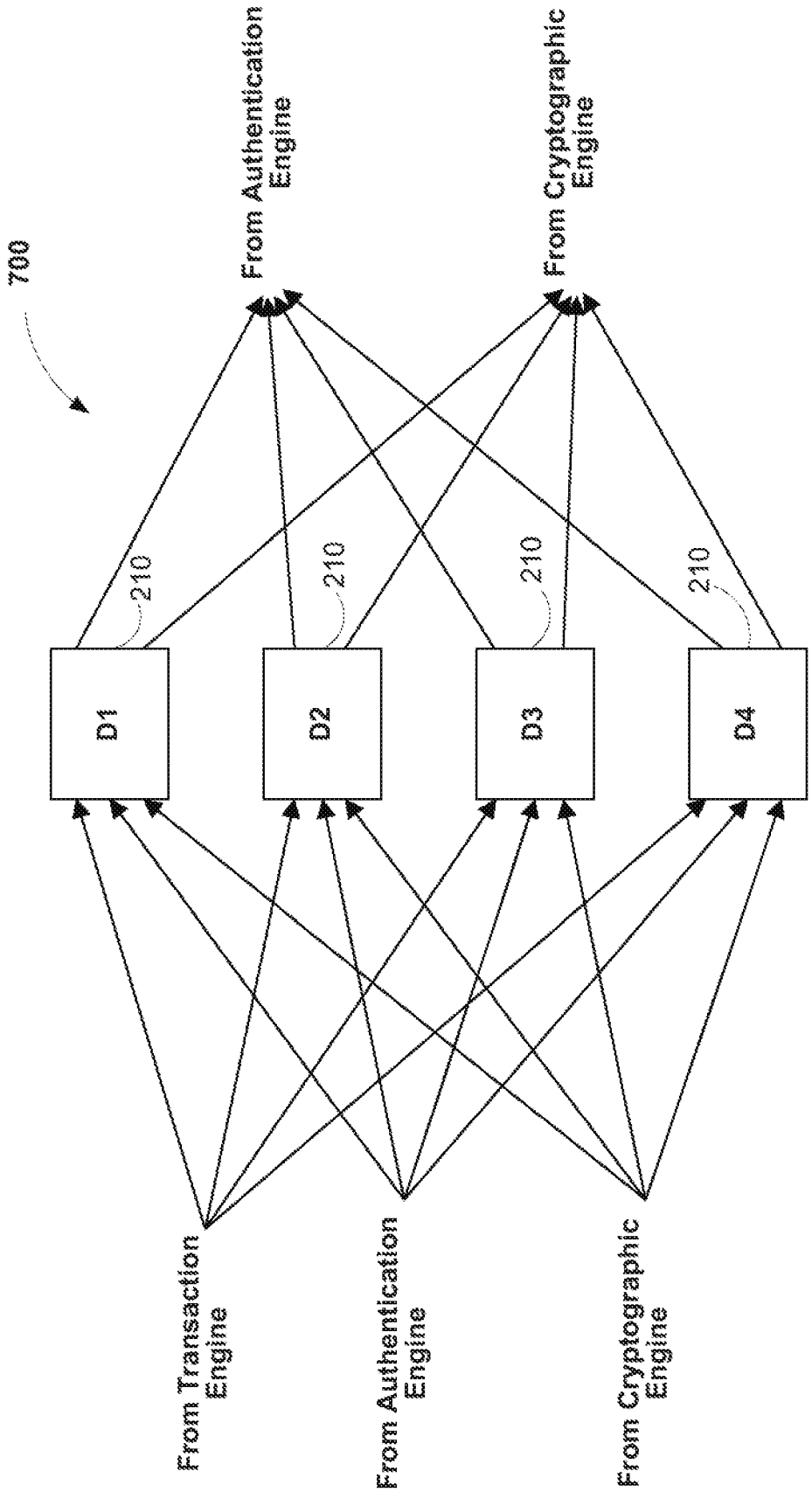

700
From Authentication Engine
From Cryptographic Engine
210
210
210
210
D1
D2
D3
D4
From Transaction Engine
From Authentication Engine
From Cryptographic Engine

FIG. 7

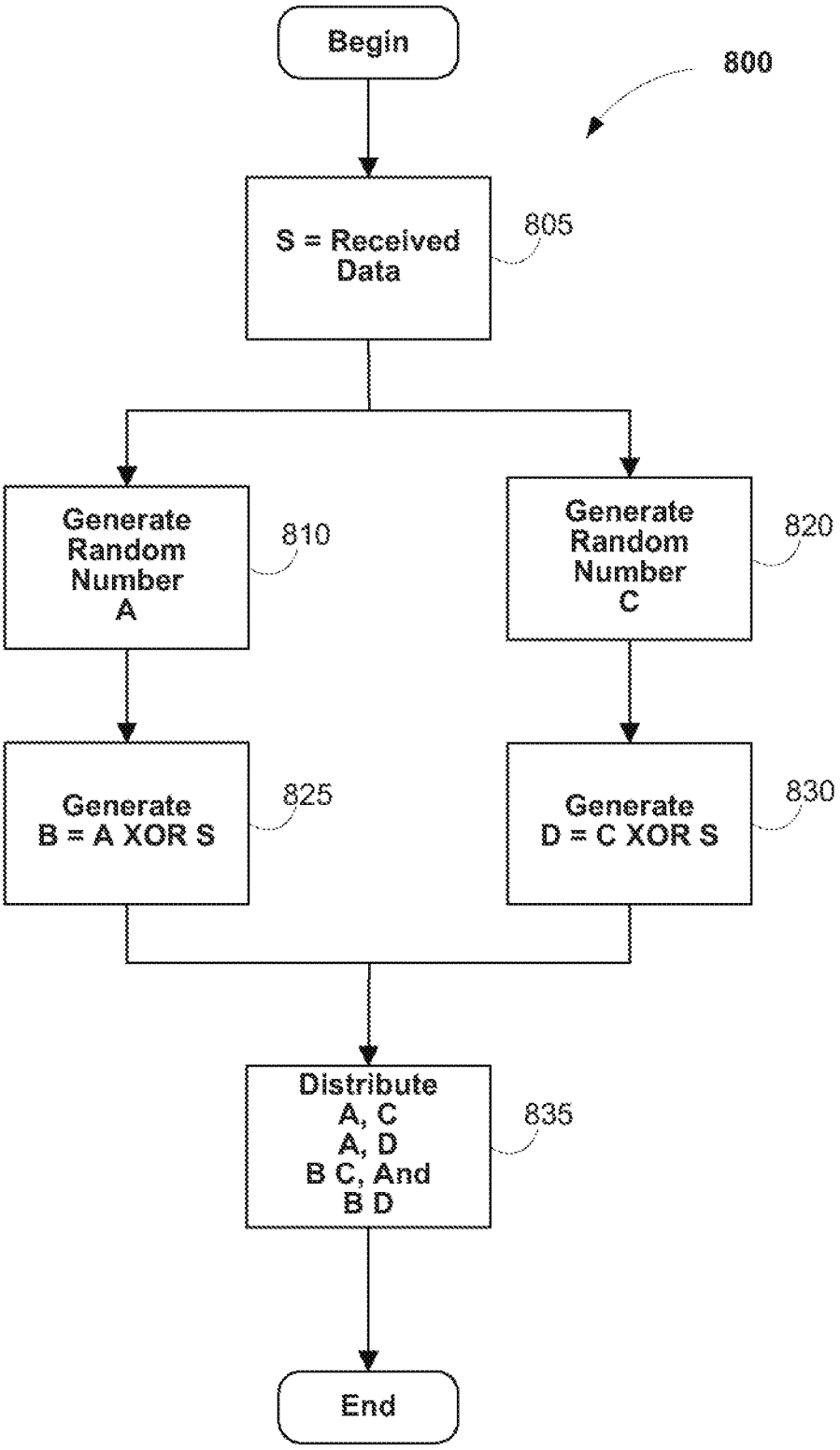

Begin
800
S = Received Data
805
Generate Random Number A
810
Generate Random Number C
820
Generate B = A XOR S
825
Generate D = C XOR S
830
Distribute A, C A, D B C, And B D
835
End

FIG. 8

900

Enrollment Data Flow				
Send	Receive	SSL	Action	
User	Transaction Engine (TE)	1/2	Transmit Enrollment Authentication Data (B) and the User ID (UID) encrypted with the Public Key of the Authentication Engine (AE) as (PUB_AE(UID,B))	905
TE	AE	Full	Forward Transmission	915
			AE Decrypts and Splits Forwarded Data	920
AE	The Xth Depository (DX)	Full	Store Respective Portion of Data	925
When Digital Certificate Requested				
AE	Cryptographic Engine (CE)	Full	Request Key Generation	930
			CE Generates and Splits Key	935
CE	TE	Full	Transmit Request for Digital Certificate	945
TE	Certification Authority (CA)	1/2	Transmit Request	950
CA	TE	1/2	Transmit Digital Certificate	955
TE	User	1/2	Transmit Digital Certificate	960
TE	MS	Full	Store Digital Certificate	
CE	DX	Full	Store Respective Portion of Key	965

FIG. 9A

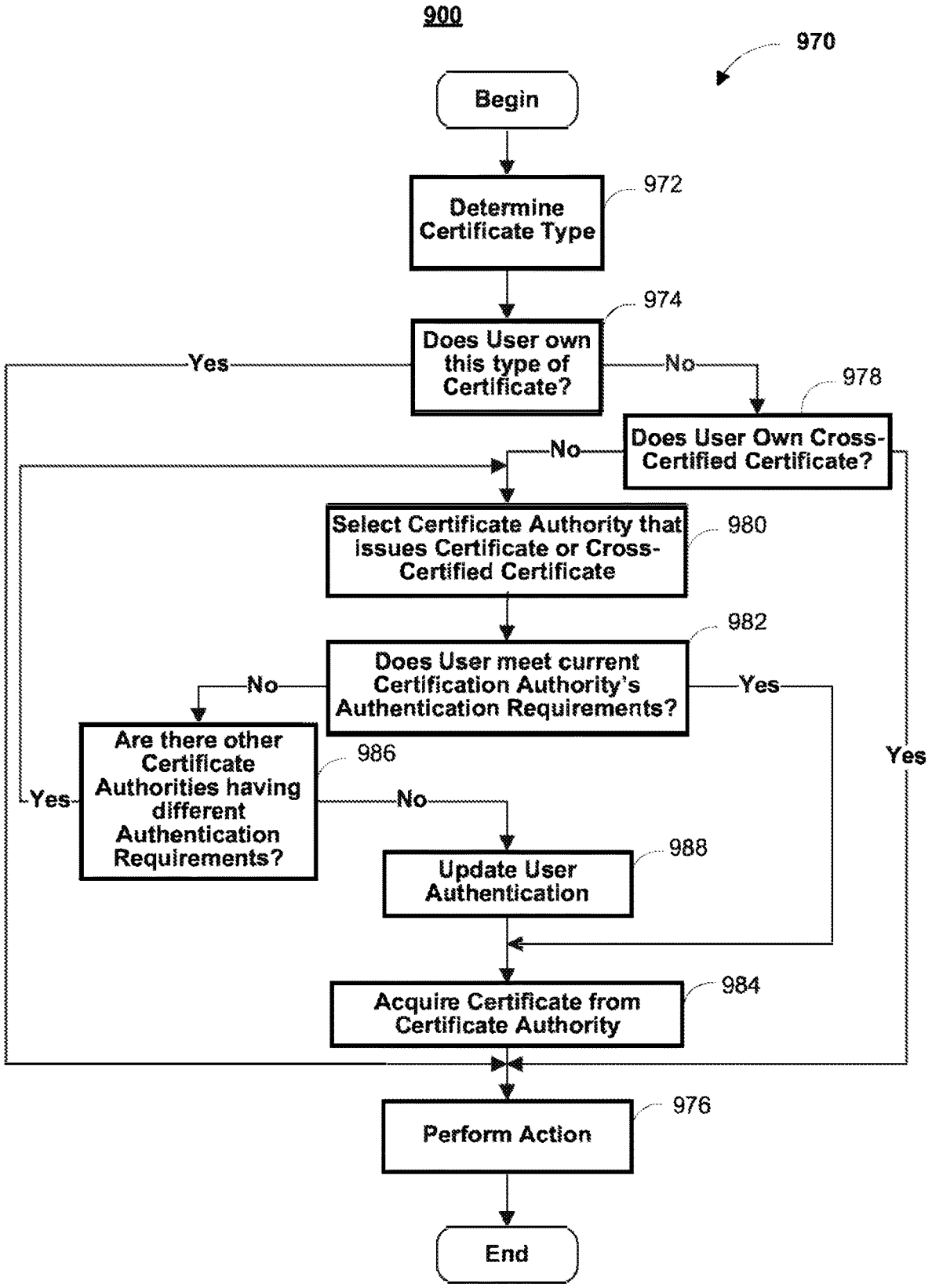

900
970
Begin
Determine Certificate Type
972
Does User own this type of Certificate?
974
Yes
No
Does User Own Cross-Certified Certificate?
978
No
Yes
Select Certificate Authority that issues Certificate or Cross-Certified Certificate
980
Does User meet current Certification Authority's Authentication Requirements?
982
No
Yes
Are there other Certificate Authorities having different Authentication Requirements?
986
Yes
No
Update User Authentication
988
Acquire Certificate from Certificate Authority
984
Perform Action
976
End

FIG. 9B

1000

	Authentication Data Flow			
	SEND	RECEIVE	SSL	ACTION
1005	User	Vendor	1/2	Transaction occurs, such as selecting purchase
1010	Vendor	User	1/2	Transmit transaction ID (TID) and authentication request (AR)
				Authentication data (B') is gathered from User
1015	User	TE	1/2	Transmit TID and B' wrapped in the Public Key of the Authentication Engine (AE), as (PUB_AE(TID, B'))
1020	TE	AE	Full	Forward transmission
				Enrollment authentication data (B) is requested and gathered
1025	Vendor	Transaction Engine (TE)	Full	Transmits TID, AR
1030	TE	Mass Storage (MS)	Full	Create Record in database
1035	TE	The Xth Depository (DX)	Full	UID, TID
1040	DX	AE	Full	Transmit the TID and the portion of the authentication data stored at enrollment (BX) as (PUB_AE(TID, BX))
1045				AE assembles B and compares to B'
1050	AE	TE	Full	TID, the filled in AR
1055	TE	Vendor	Full	TID, Yes/No
	TE	User	1/2	TID, confirmation message

FIG. 10

1100

	Signing Data Flow			
	SEND	RECEIVE	SSL	ACTION
	User	Vendor	1/2	Transaction occurs, such as agreeing on a deal
	Vendor	User	1/2	Transmit transaction identification number (TID), authentication request (AR), and agreement or message (M)
				Current authentication data (B') and a hash of the message received by the User (h(M')) is is gathered from User
	User	TE	1/2	Transmit TID, B', AR, and h(M') wrapped in the Public Key of the Authentication Engine (AE), as (PUB_AE(TID, B', h(M'))
	TE	AE	Full	Forward transmission
				Gather enrollment authentication data
	Vendor	Transaction Engine (TE)	Full	Transmits UID, TID, AR, and a hash of the message (h(M')).
	TE	Mass Storage (MS)	Full	Create Record in database
	TE	The Xth Depository (DX)	Full	UID, TID
	DX	AE	Full	Transmit the TID and the portion of the authentication data stored at Enrollment (BX), as (PUB_AE(TID, BX))
				The original vendor message is transmitted to the AE
	TE	AE	Full	Transmit h(M)
1103				AE assembles B, compares to B' and compares h(M) to h(M')
1105	AE	Cryptographic Engine (CE)	Full	Request for digital signature and a message to be signed, for example, the hashed message
1110	AE	DX	Full	TID, signing UID
1115	DX	CE	Full	Transmit the portion of the Crypto-graphic Key corresponding to the signing party
1120				CE assembles key and signs
1125	CE	AE	Full	Transmit the digital signature (S) of signing party
1130	AE	TE	Full	TID, the filled in AR, h(M), and S
1135	TE	Vendor	Full	TID, a receipt=(TID, Yes/No, and S), and the digital signature of the trust engine, for example, a hash of the receipt encrypted with the trust engine's Private Key (Priv_TE(h(receipt)))
1140	TE	User	1/2	TID, confirmation message

FIG. 11

1200

Encryption/Decryption Data Flow				
Send	Receive	SSL	Action	
Decryption				
			Perform Authentication Data Process 1000, include the Session Key (sync) in the AR, where the sync has been encrypted with the Public Key of the User as PUB_USER(SYNC)	
			Authenticate the User	
AE	CE	Full	Forward PUB_USER(SYNC) to CE	1205
AE	DX	Full	UID, TID	1210
DX	CE	Full	Transmit the TID and the portion of the Private Key as (PUB_AE(TID, KEY_USER))	1215
			CE assembles the Cryptographic Key and decrypts the sync	1220
CE	AE	Full	TID, the filled in AR including decrypted sync	1225
AE	TE	Full	Forward to TE	1230
TE	Requesting APP/Vendor	1/2	TID, Yes/No, Sync	
Encryption				
Requesting APP/ Vendor	TE	1/2	Request for Public Key of User	1235
TE	MS	Full	Request Digital Certificate	1240
MS	TE	Full	Transmit Digital Certificate	1245
TE	Requesting APP/Vendor	1/2	Transmit Digital Certificate	1250

FIG. 12

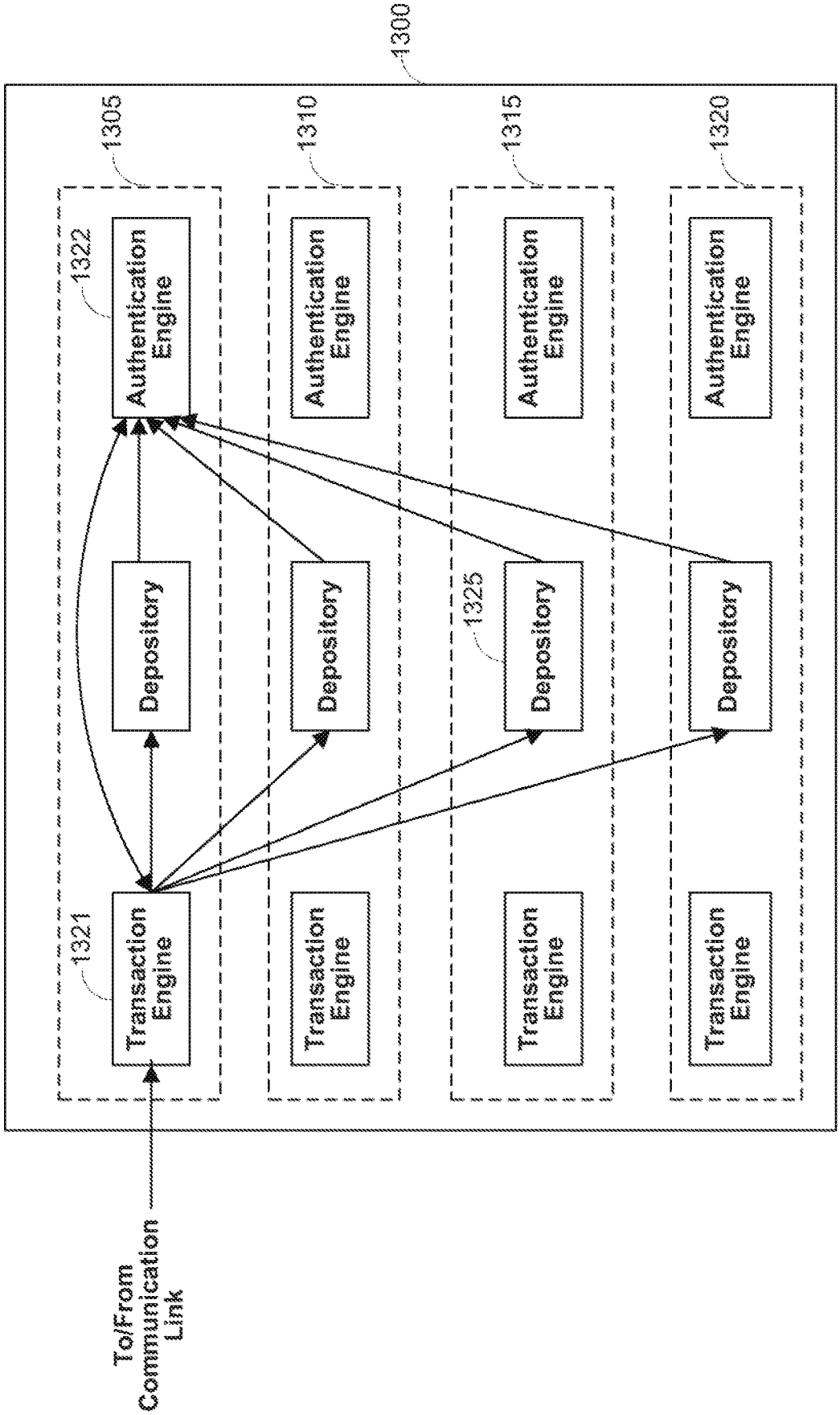

1300
1305
1310
1315
1320
1322
Authentication Engine
Authentication Engine
Authentication Engine
Authentication Engine
Depository
Depository
1325
Depository
Depository
1321
Transaction Engine
Transaction Engine
Transaction Engine
Transaction Engine
To/From Communication Link

FIG. 13

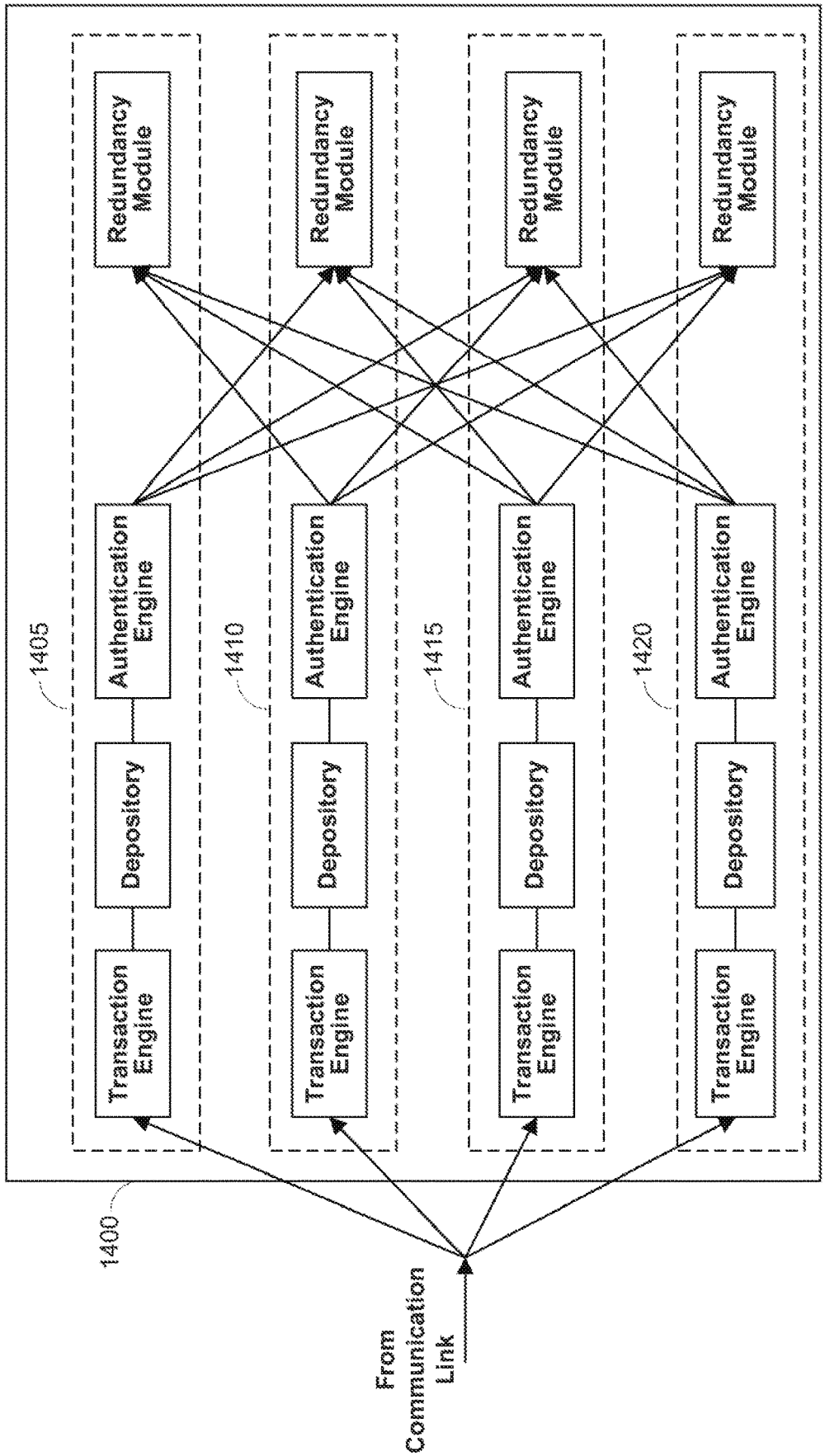

1400
From Communication Link
1405
1410
1415
1420
Transaction Engine
Depository
Authentication Engine
Redundancy Module

FIG. 14

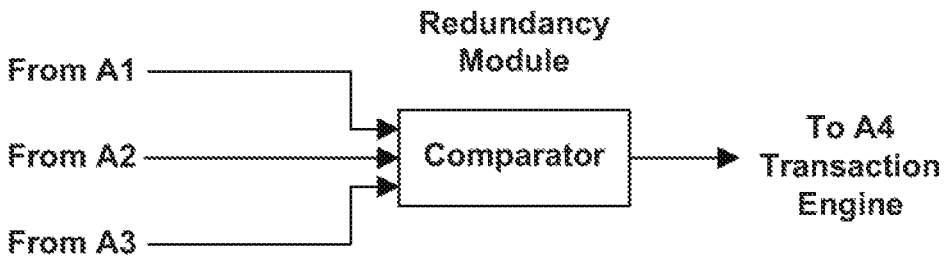

Redundancy
Module
From A1
From A2
From A3
Comparator
To A4
Transaction
Engine

FIG. 15

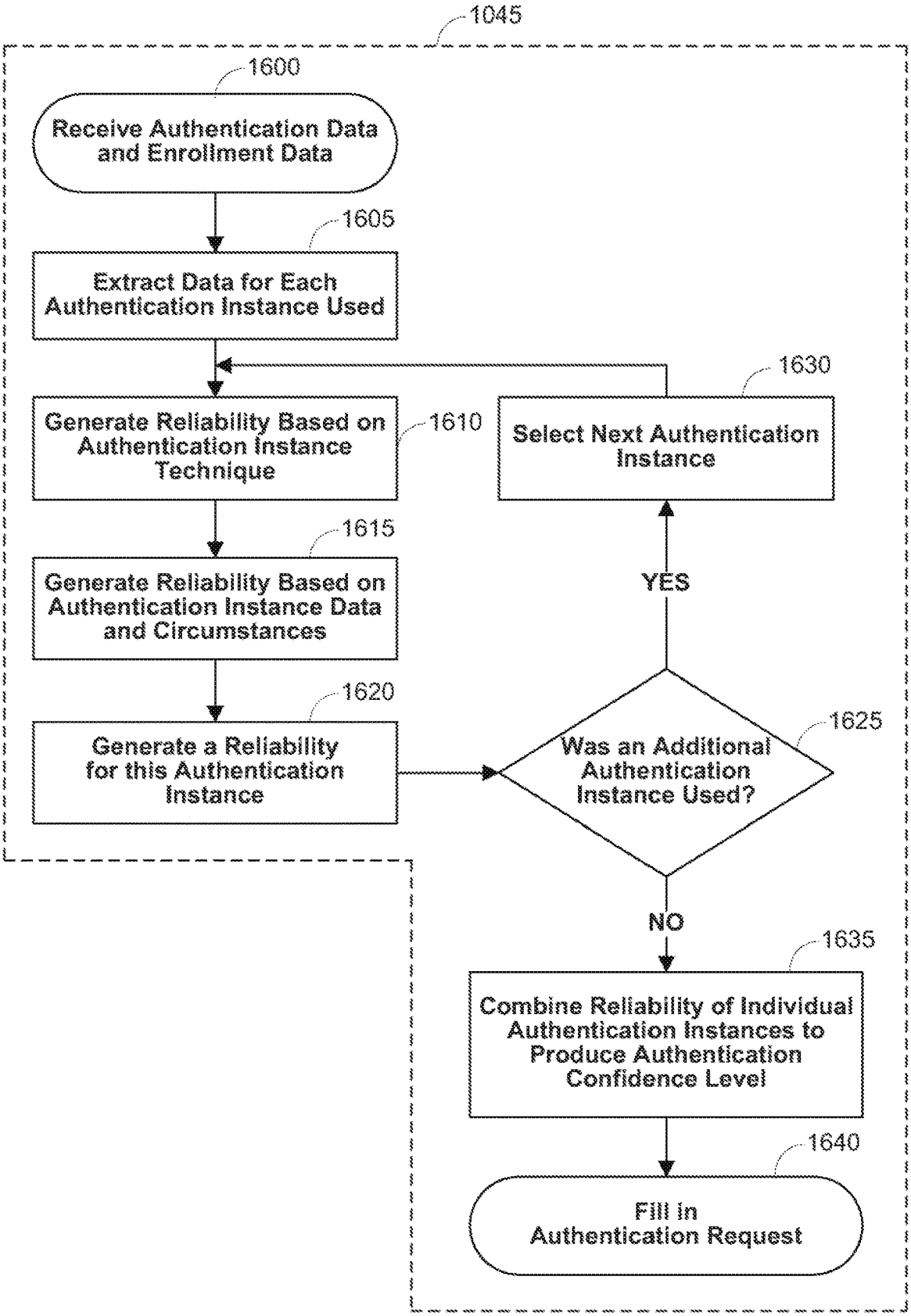

1045
1600
Receive Authentication Data and Enrollment Data
1605
Extract Data for Each Authentication Instance Used
1630
Generate Reliability Based on Authentication Instance Technique
1610
Select Next Authentication Instance
1615
Generate Reliability Based on Authentication Instance Data and Circumstances
YES
1620
Generate a Reliability for this Authentication Instance
1625
Was an Additional Authentication Instance Used?
NO
1635
Combine Reliability of Individual Authentication Instances to Produce Authentication Confidence Level
1640
Fill in Authentication Request

FIG. 16

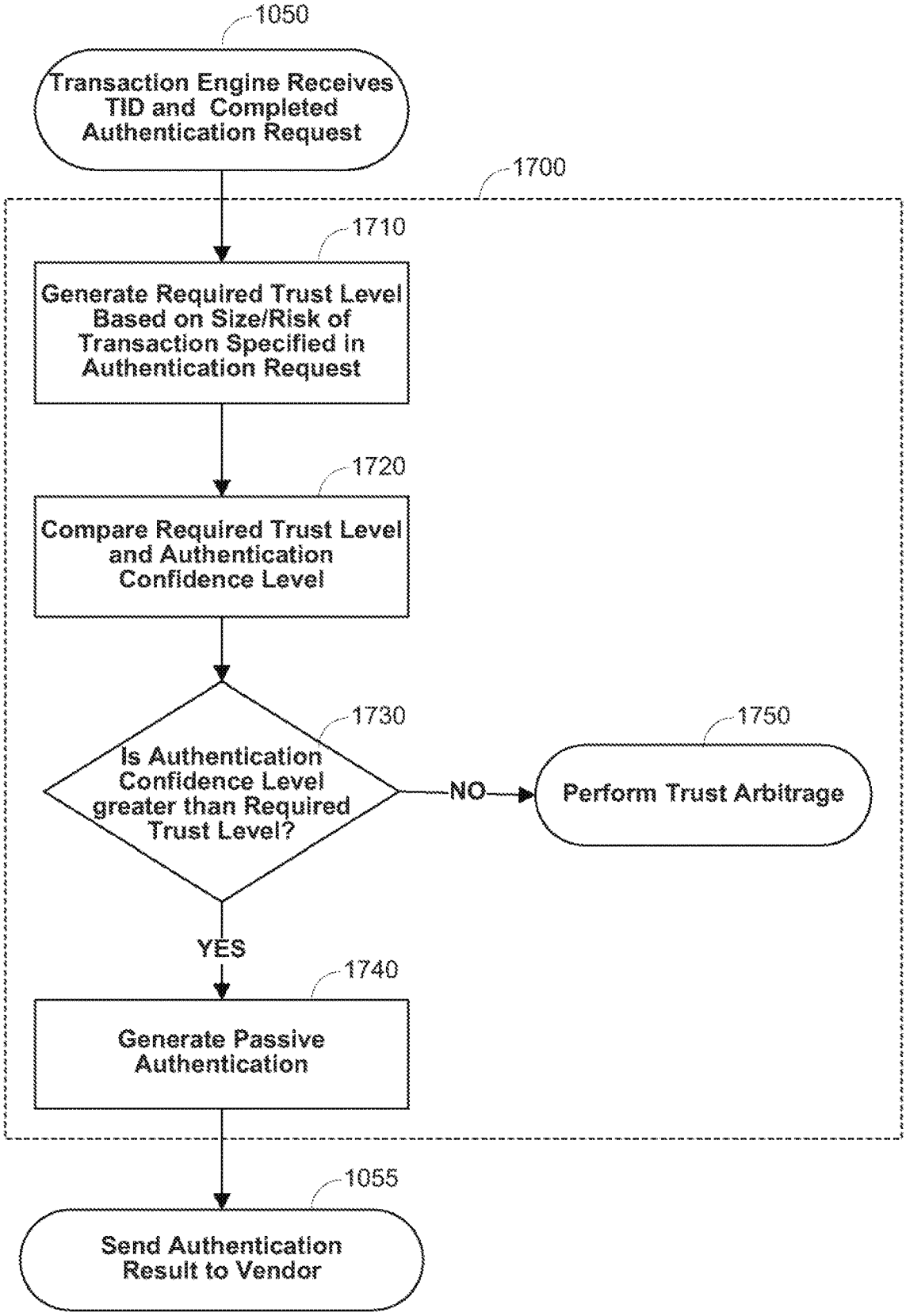

1050
Transaction Engine Receives TID and Completed Authentication Request
1700
1710
Generate Required Trust Level Based on Size/Risk of Transaction Specified in Authentication Request
1720
Compare Required Trust Level and Authentication Confidence Level
1730
Is Authentication Confidence Level greater than Required Trust Level?
NO
1750
Perform Trust Arbitrage
YES
1740
Generate Passive Authentication
1055
Send Authentication Result to Vendor

FIG. 17

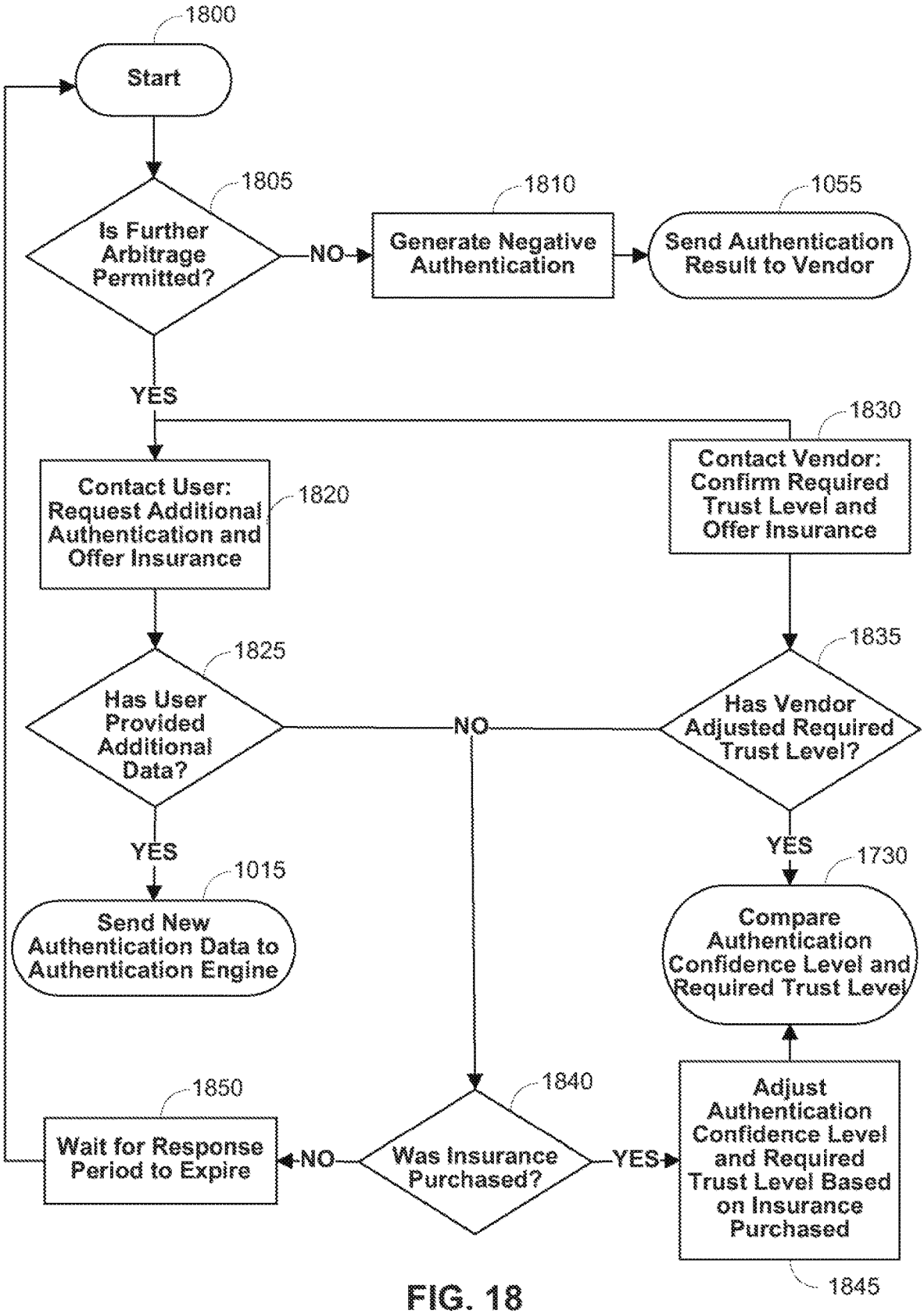

1800
Start
1805
Is Further Arbitrage Permitted?
NO
1810
Generate Negative Authentication
1055
Send Authentication Result to Vendor
YES
1830
Contact Vendor: Confirm Required Trust Level and Offer Insurance
Contact User: Request Additional Authentication and Offer Insurance
1820
1825
Has User Provided Additional Data?
NO
1835
Has Vendor Adjusted Required Trust Level?
YES
1015
Send New Authentication Data to Authentication Engine
YES
1730
Compare Authentication Confidence Level and Required Trust Level
1850
Wait for Response Period to Expire
NO
1840
Was Insurance Purchased?
YES
Adjust Authentication Confidence Level and Required Trust Level Based on Insurance Purchased
1845

FIG. 18

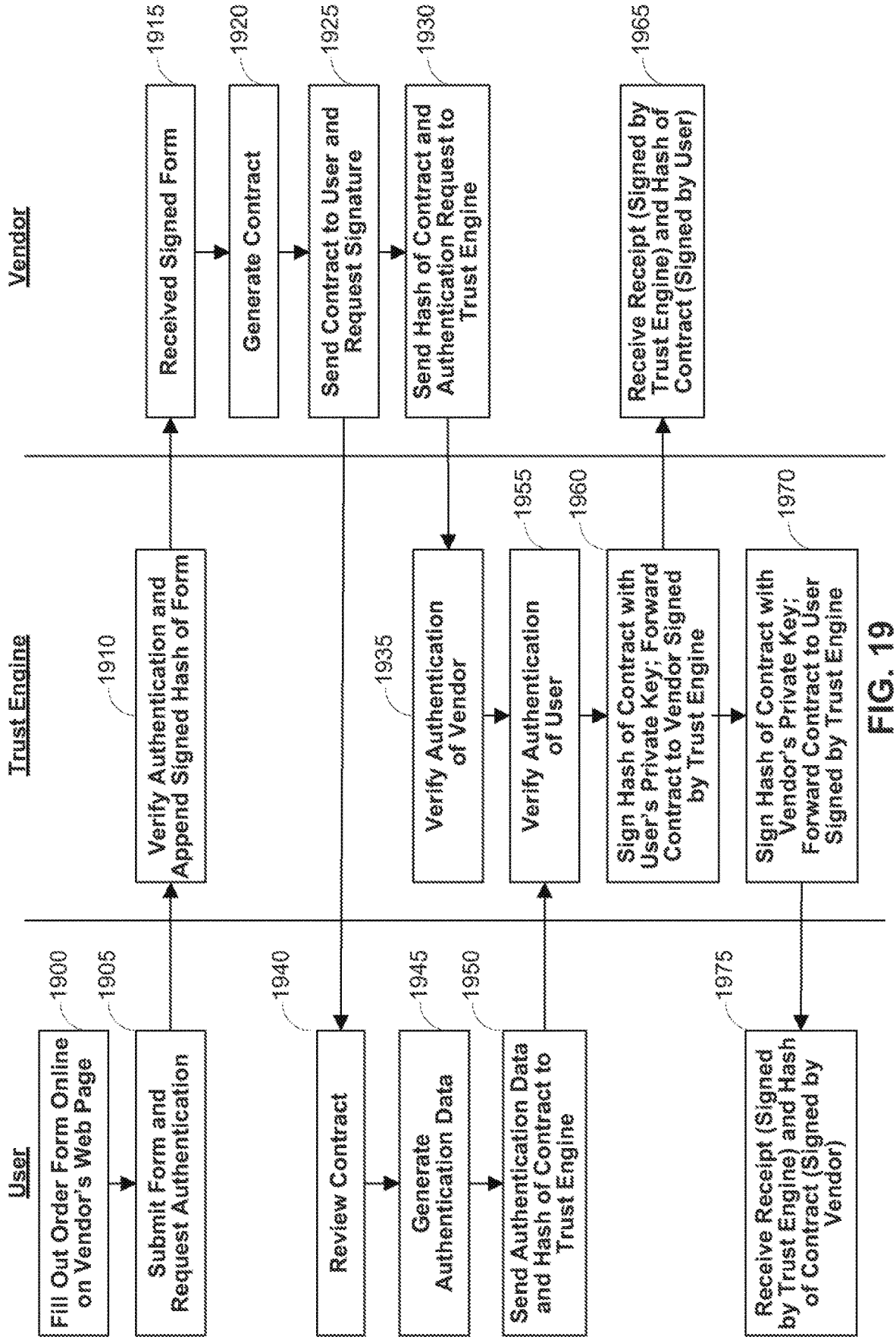

User
Trust Engine
Vendor
1900
Fill Out Order Form Online on Vendor's Web Page
1905
Submit Form and Request Authentication
1910
Verify Authentication and Append Signed Hash of Form
1915
Received Signed Form
1920
Generate Contract
1925
Send Contract to User and Request Signature
1930
Send Hash of Contract and Authentication Request to Trust Engine
1935
Verify Authentication of Vendor
1940
Review Contract
1945
Generate Authentication Data
1950
Send Authentication Data and Hash of Contract to Trust Engine
1955
Verify Authentication of User
1960
Sign Hash of Contract with User's Private Key; Forward Contract to Vendor Signed by Trust Engine
1965
Receive Receipt (Signed by Trust Engine) and Hash of Contract (Signed by User)
1970
Sign Hash of Contract with Vendor's Private Key; Forward Contract to User Signed by Trust Engine
1975
Receive Receipt (Signed by Trust Engine) and Hash of Contract (Signed by Vendor)

FIG. 19

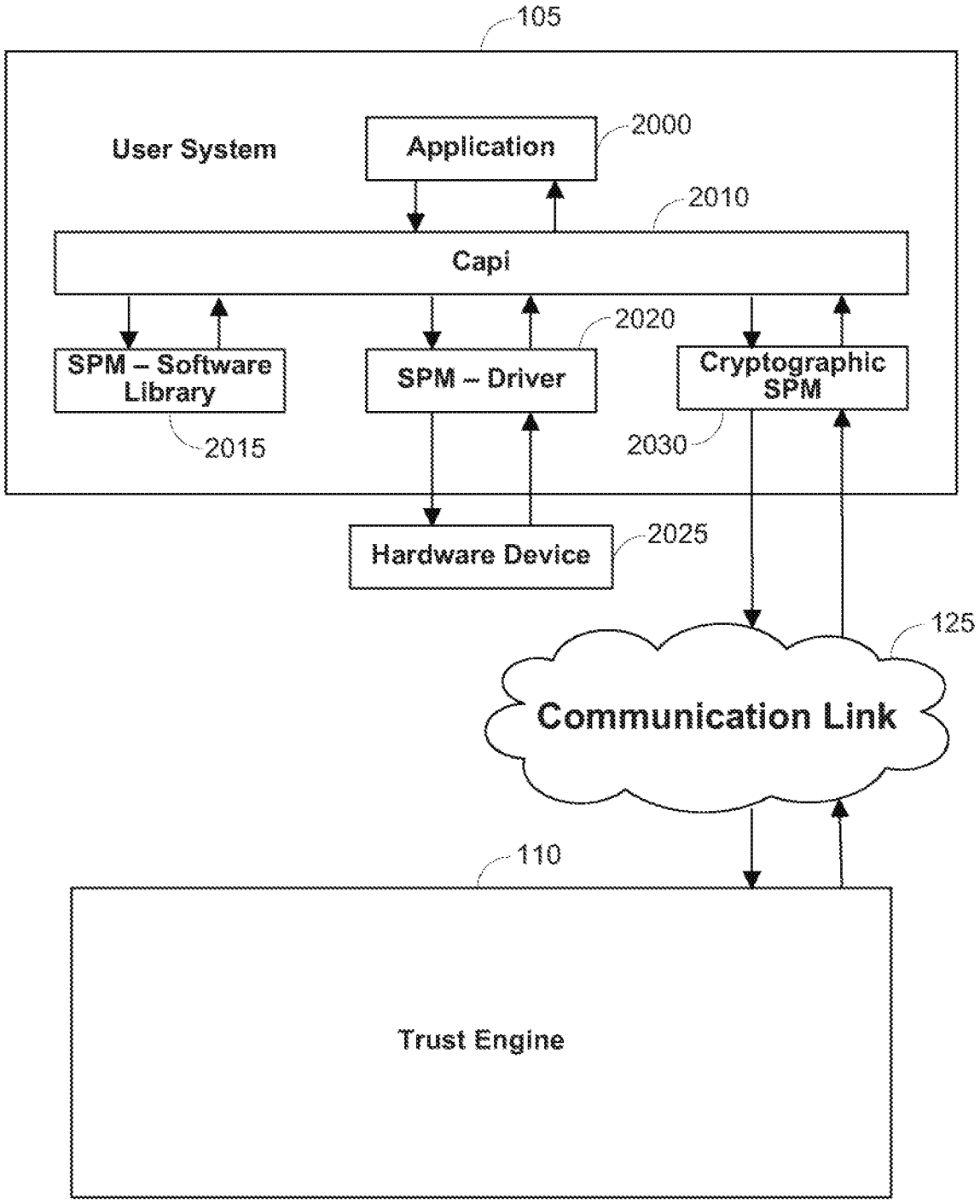

105
User System
Application
2000
2010
Capi
2020
SPM – Software Library
SPM – Driver
Cryptographic SPM
2015
2030
Hardware Device
2025
125
Communication Link
110
Trust Engine

FIG. 20

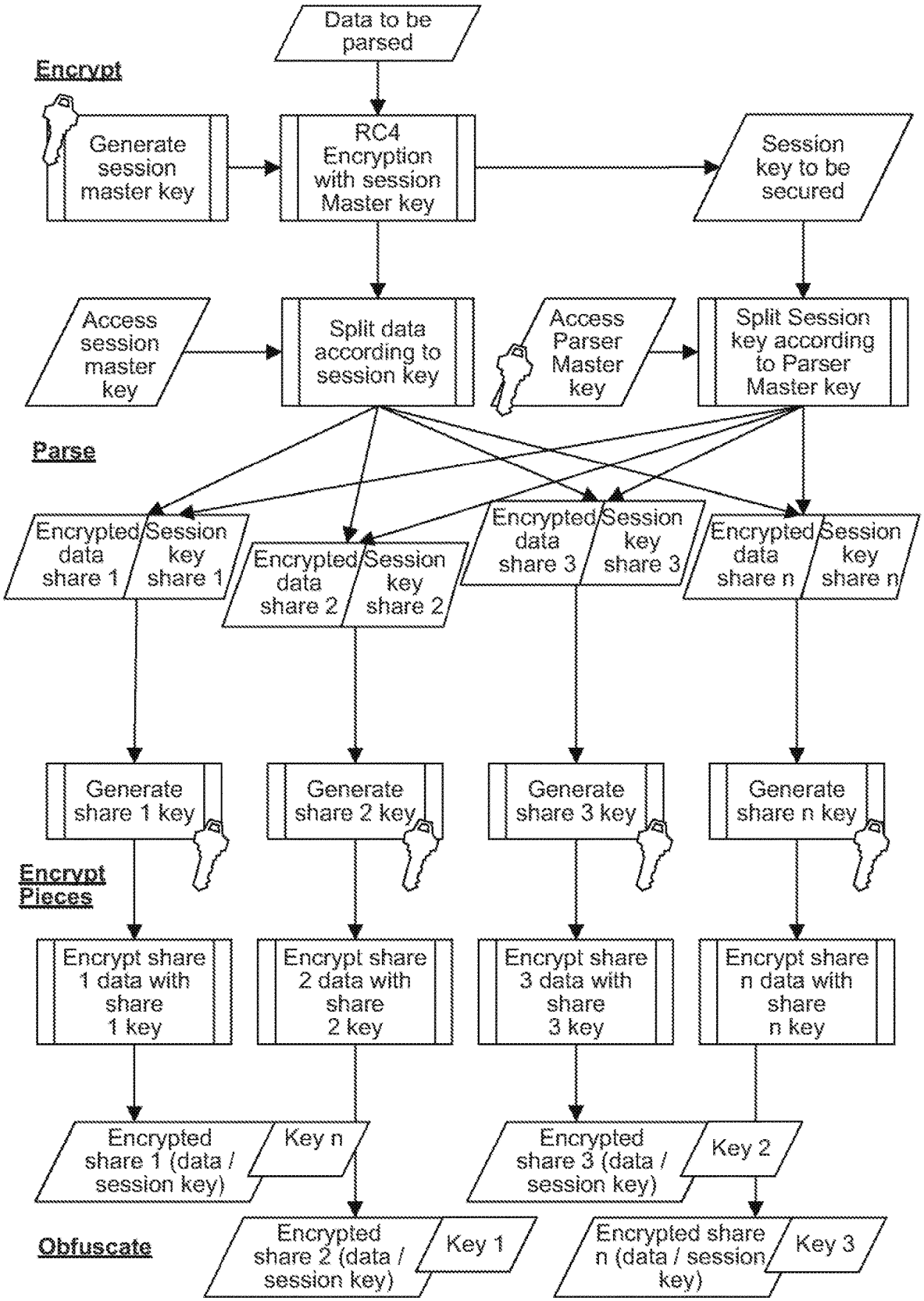

Data to be parsed
Encrypt
Generate session master key
RC4 Encryption with session Master key
Session key to be secured
Access session master key
Split data according to session key
Access Parser Master key
Split Session key according to Parser Master key
Parse
Encrypted data share 1
Session key share 1
Encrypted data share 2
Session key share 2
Encrypted data share 3
Session key share 3
Encrypted data share n
Session key share n
Generate share 1 key
Generate share 2 key
Generate share 3 key
Generate share n key
Encrypt Pieces
Encrypt share 1 data with share 1 key
Encrypt share 2 data with share 2 key
Encrypt share 3 data with share 3 key
Encrypt share n data with share n key
Encrypted share 1 (data / session key)
Key n
Encrypted share 3 (data / session key)
Key 2
Obfuscate
Encrypted share 2 (data / session key)
Key 1
Encrypted share n (data / session key)
Key 3

FIG. 21

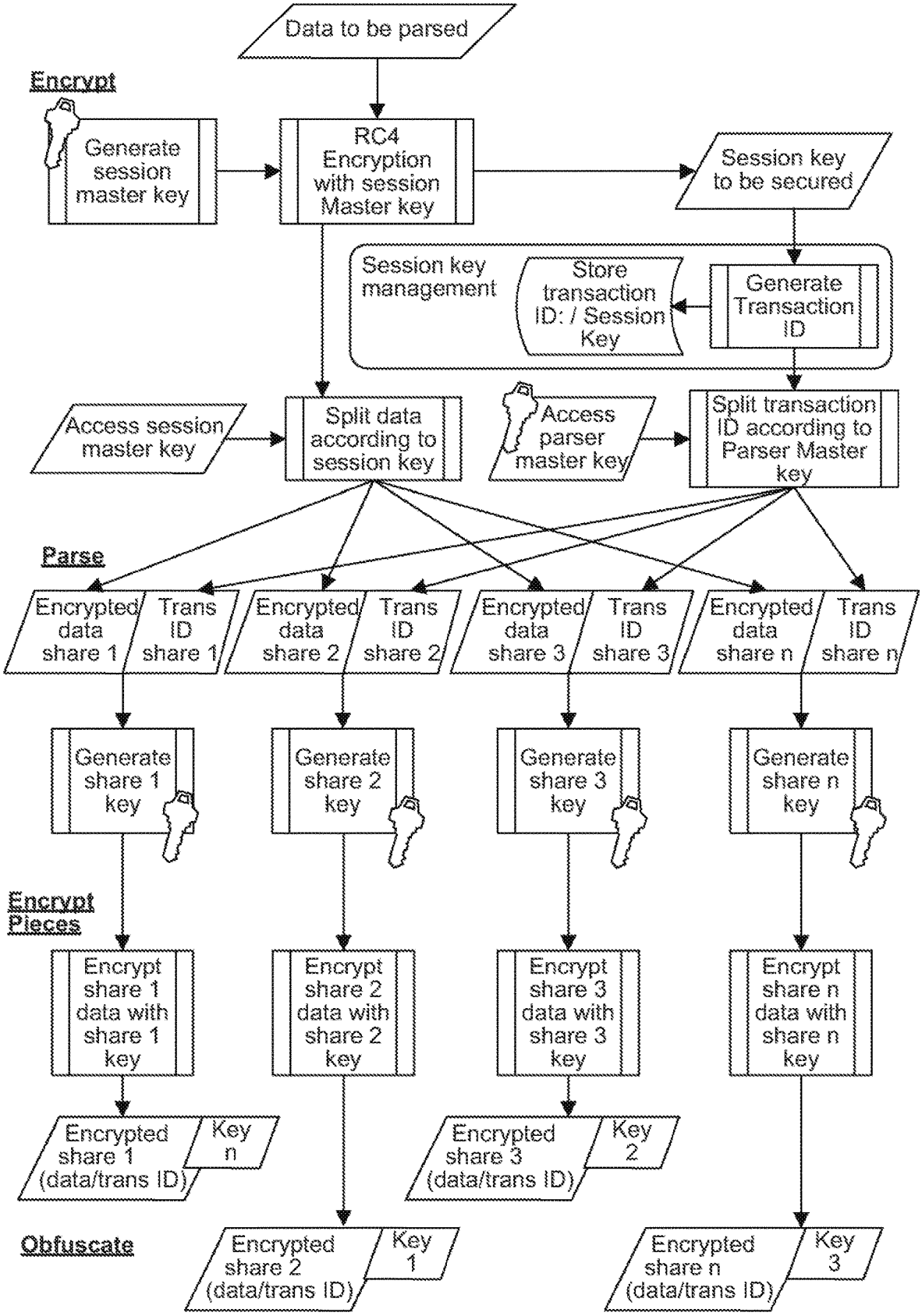

Data to be parsed
Encrypt
Generate session master key
RC4 Encryption with session Master key
Session key to be secured
Session key management
Store transaction ID: / Session Key
Generate Transaction ID
Access session master key
Split data according to session key
Access parser master key
Split transaction ID according to Parser Master key
Parse
Encrypted data share 1
Trans ID share 1
Encrypted data share 2
Trans ID share 2
Encrypted data share 3
Trans ID share 3
Encrypted data share n
Trans ID share n
Generate share 1 key
Generate share 2 key
Generate share 3 key
Generate share n key
Encrypt Pieces
Encrypt share 1 data with share 1 key
Encrypt share 2 data with share 2 key
Encrypt share 3 data with share 3 key
Encrypt share n data with share n key
Encrypted share 1 (data/trans ID)
Key n
Encrypted share 3 (data/trans ID)
Key 2
Obfuscate
Encrypted share 2 (data/trans ID)
Key 1
Encrypted share n (data/trans ID)
Key 3

FIG. 22

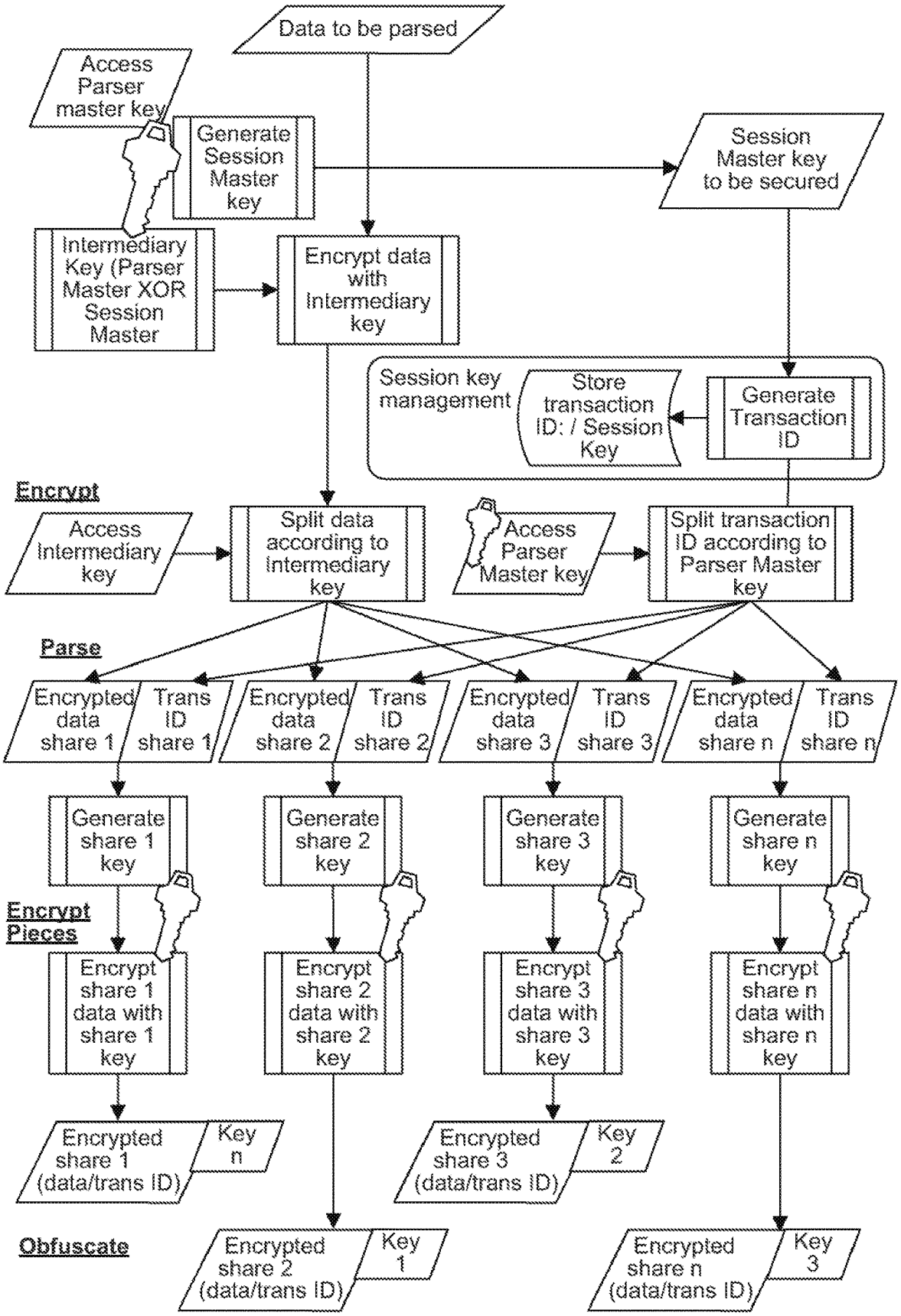

Data to be parsed
Access Parser master key
Generate Session Master key
Session Master key to be secured
Intermediary Key (Parser Master XOR Session Master
Encrypt data with Intermediary key
Session key management
Store transaction ID: / Session Key
Generate Transaction ID
Encrypt
Access Intermediary key
Split data according to Intermediary key
Access Parser Master key
Split transaction ID according to Parser Master key
Parse
Encrypted data share 1
Trans ID share 1
Encrypted data share 2
Trans ID share 2
Encrypted data share 3
Trans ID share 3
Encrypted data share n
Trans ID share n
Generate share 1 key
Generate share 2 key
Generate share 3 key
Generate share n key
Encrypt Pieces
Encrypt share 1 data with share 1 key
Encrypt share 2 data with share 2 key
Encrypt share 3 data with share 3 key
Encrypt share n data with share n key
Encrypted share 1 (data/trans ID)
Key n
Encrypted share 3 (data/trans ID)
Key 2
Obfuscate
Encrypted share 2 (data/trans ID)
Key 1
Encrypted share n (data/trans ID)
Key 3

FIG. 23

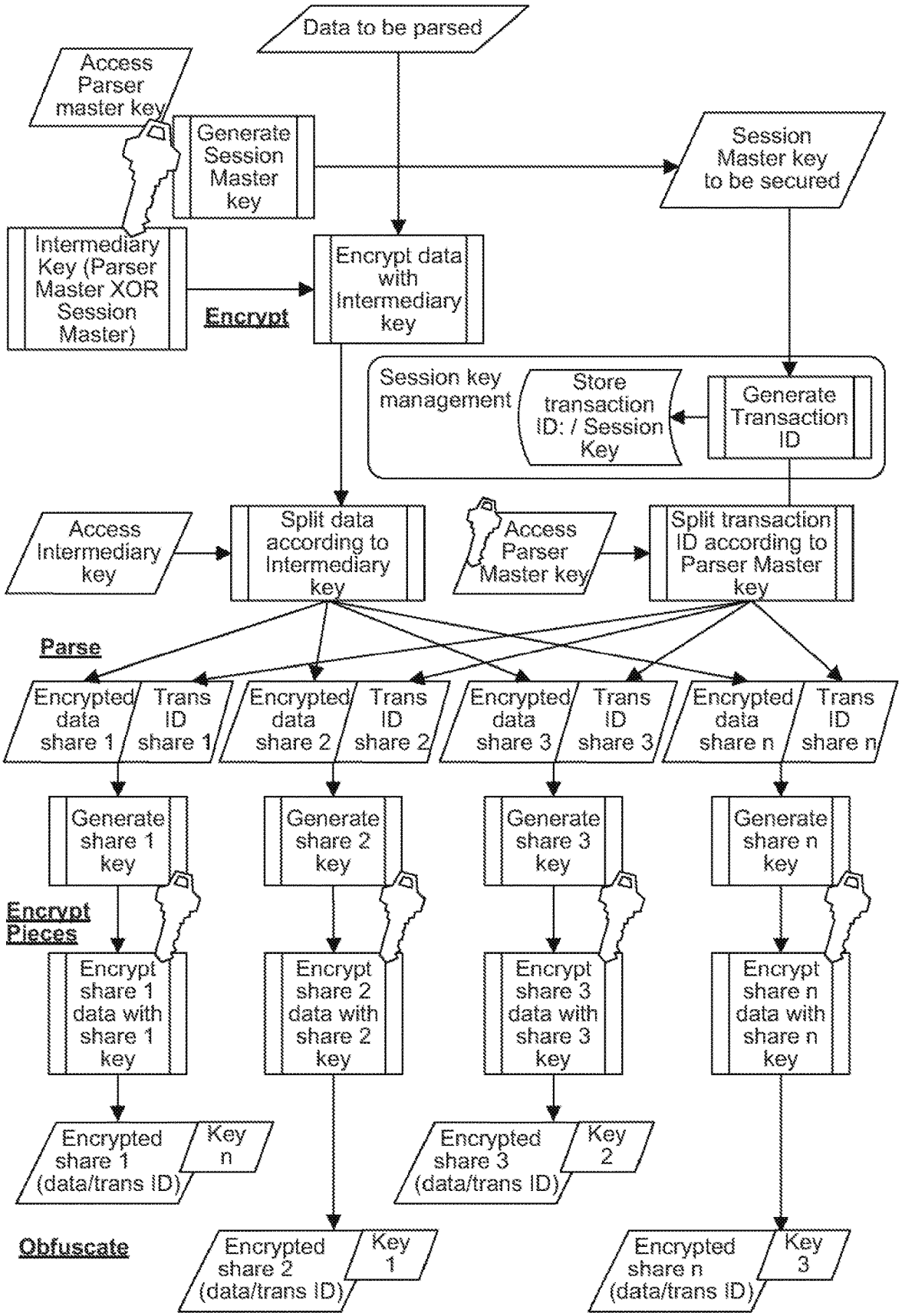

Data to be parsed
Access Parser master key
Generate Session Master key
Session Master key to be secured
Intermediary Key (Parser Master XOR Session Master)
Encrypt
Encrypt data with Intermediary key
Session key management
Store transaction ID: / Session Key
Generate Transaction ID
Access Intermediary key
Split data according to Intermediary key
Access Parser Master key
Split transaction ID according to Parser Master key
Parse
Encrypted data share 1
Trans ID share 1
Encrypted data share 2
Trans ID share 2
Encrypted data share 3
Trans ID share 3
Encrypted data share n
Trans ID share n
Generate share 1 key
Generate share 2 key
Generate share 3 key
Generate share n key
Encrypt Pieces
Encrypt share 1 data with share 1 key
Encrypt share 2 data with share 2 key
Encrypt share 3 data with share 3 key
Encrypt share n data with share n key
Encrypted share 1 (data/trans ID)
Key n
Encrypted share 3 (data/trans ID)
Key 2
Obfuscate
Encrypted share 2 (data/trans ID)
Key 1
Encrypted share n (data/trans ID)
Key 3

FIG. 24

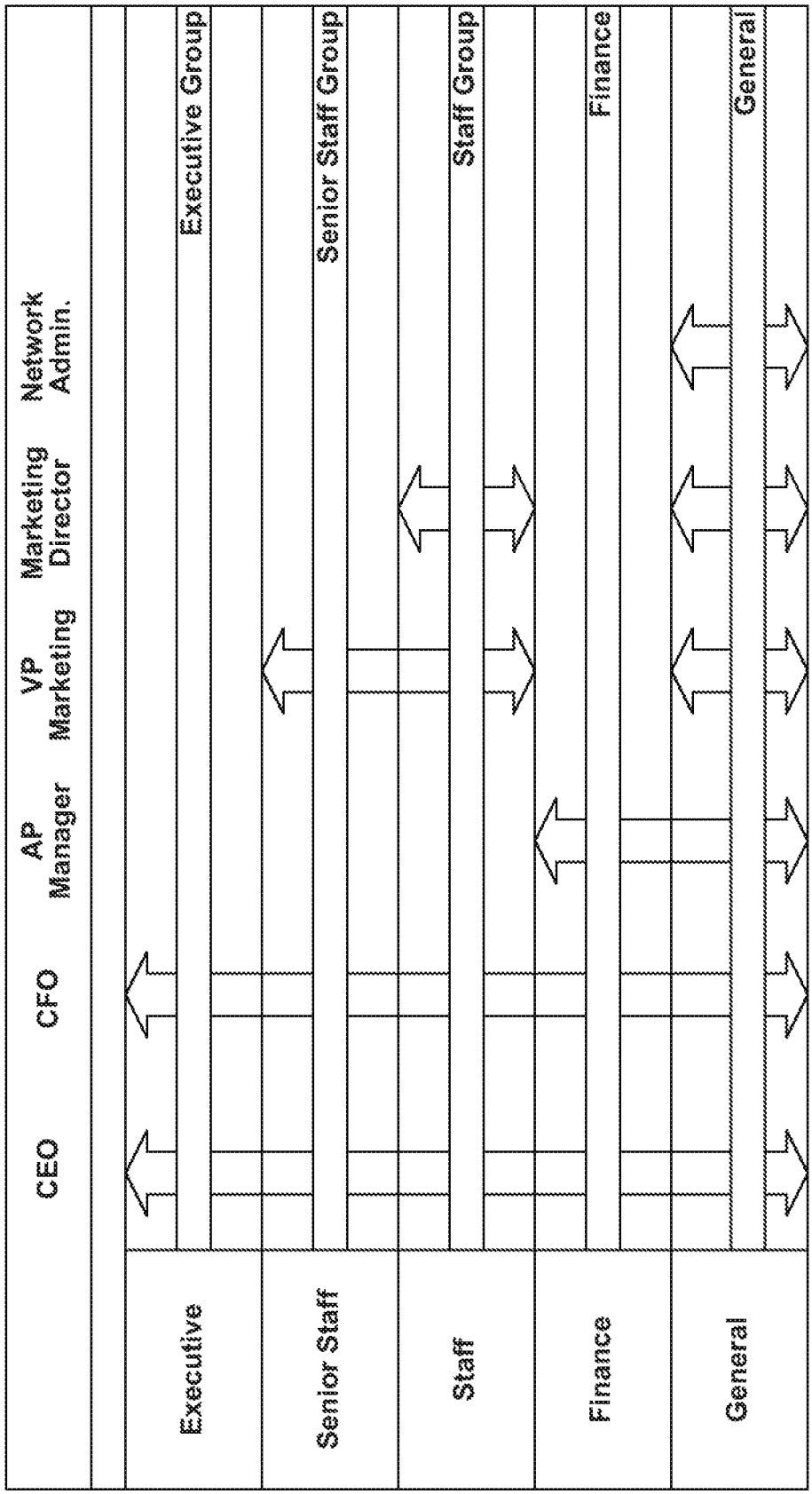

CEO
CFO
AP Manager
VP Marketing
Marketing Director
Network Admin.
Executive
Executive Group
Senior Staff
Senior Staff Group
Staff
Staff Group
Finance
Finance
General
General

FIG. 25

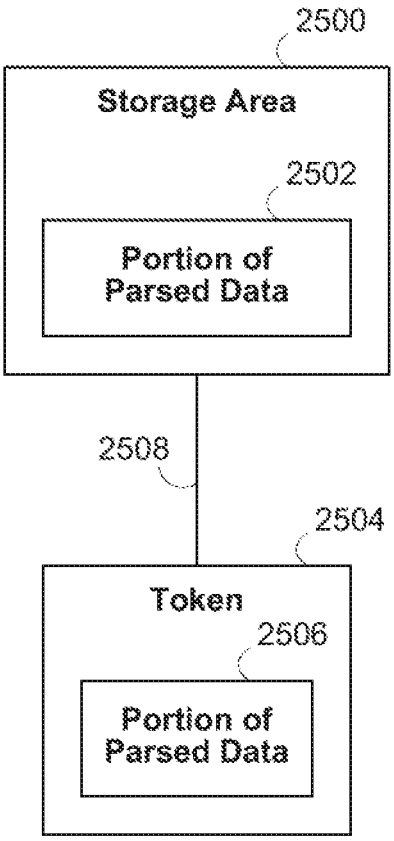

2500
Storage Area
2502
Portion of Parsed Data
2508
2504
Token
2506
Portion of Parsed Data

FIG. 26

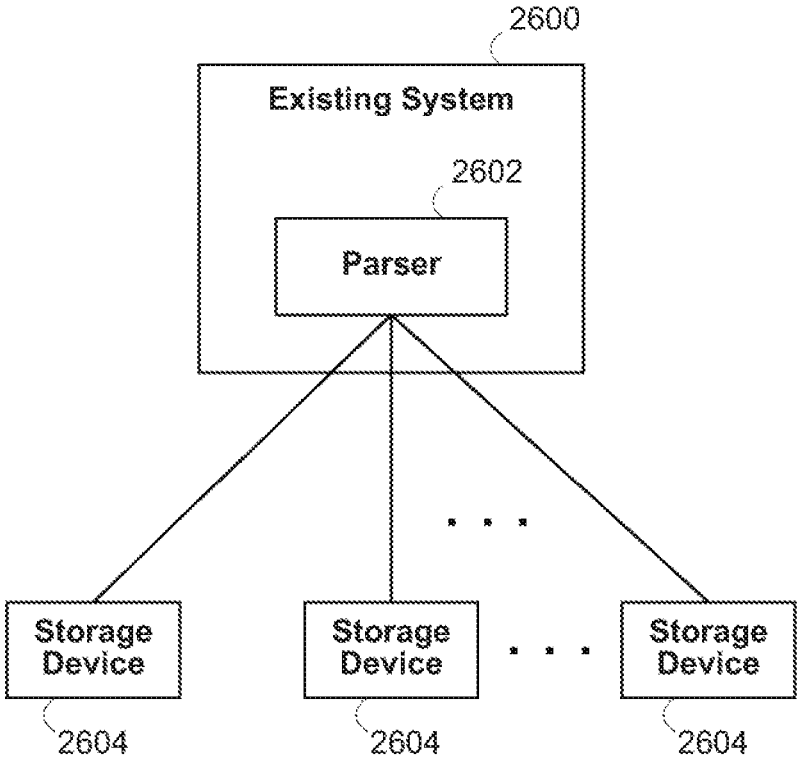

2600
Existing System
2602
Parser
Storage Device
Storage Device
Storage Device
2604
2604
2604

FIG. 27

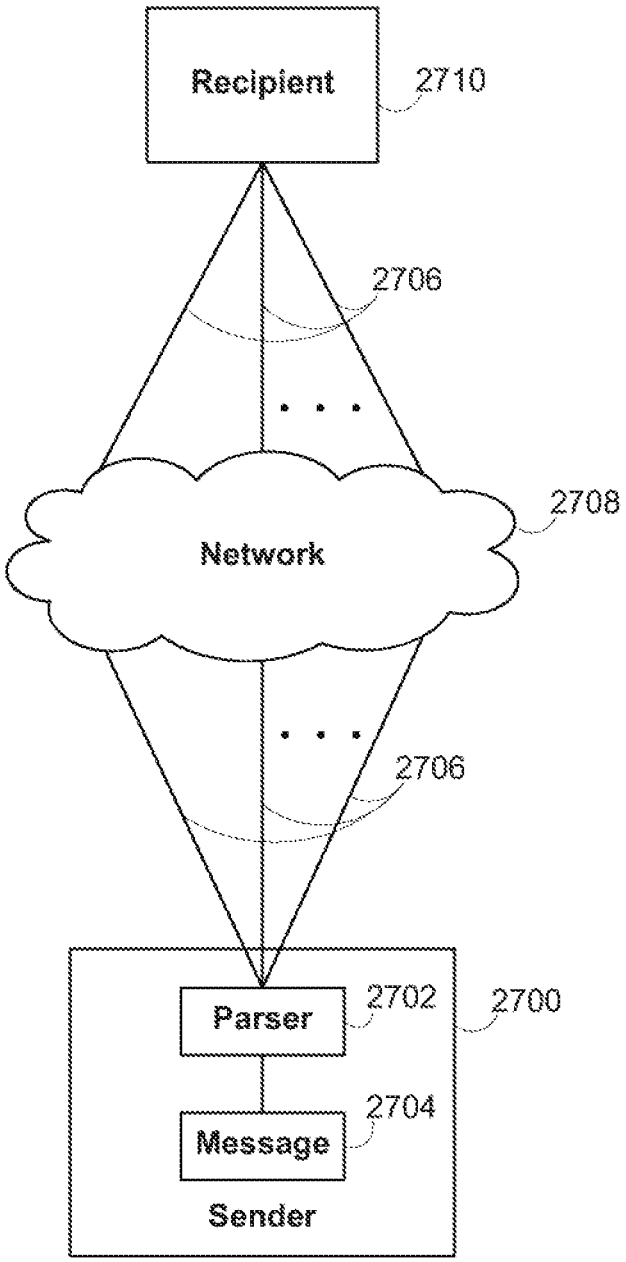

Recipient
2710
2706
Network
2708
2706
Parser
2702
2700
Message
2704
Sender

FIG. 28

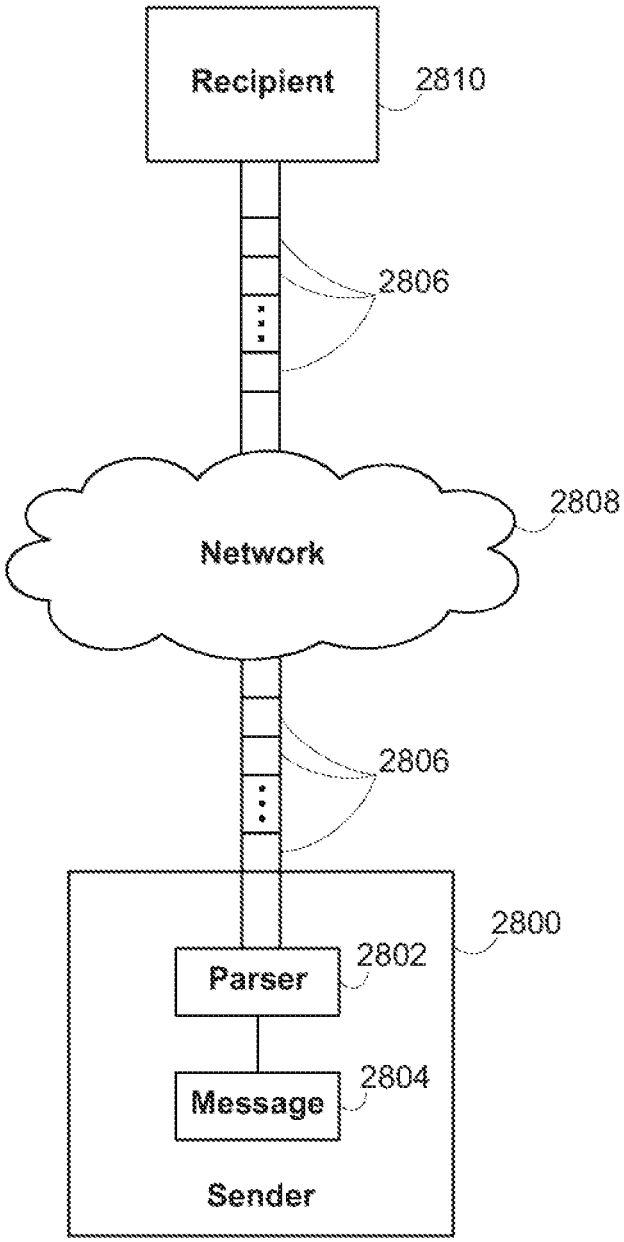

Recipient
2810
2806
Network
2808
2806
2800
Parser
2802
Message
2804
Sender

FIG. 29

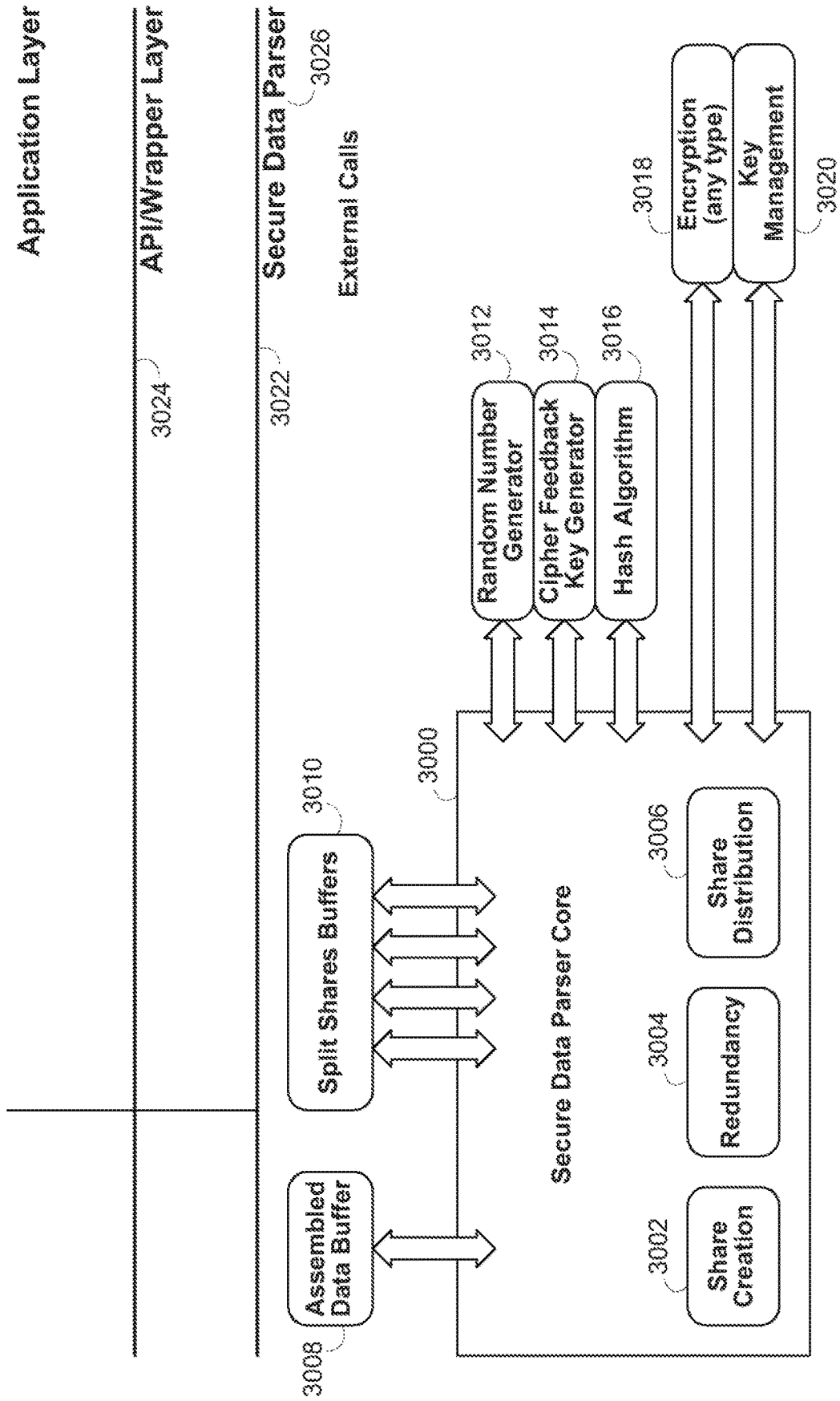

Application Layer
API/Wrapper Layer
Secure Data Parser
3026
External Calls
3024
3022
3012
3014
3016
3018
3020
Random Number Generator
Cipher Feedback Key Generator
Hash Algorithm
Encryption (any type)
Key Management
3000
3010
Split Shares Buffers
Secure Data Parser Core
3006
Share Distribution
3004
Redundancy
3002
Share Creation
Assembled Data Buffer
3008

FIG. 30

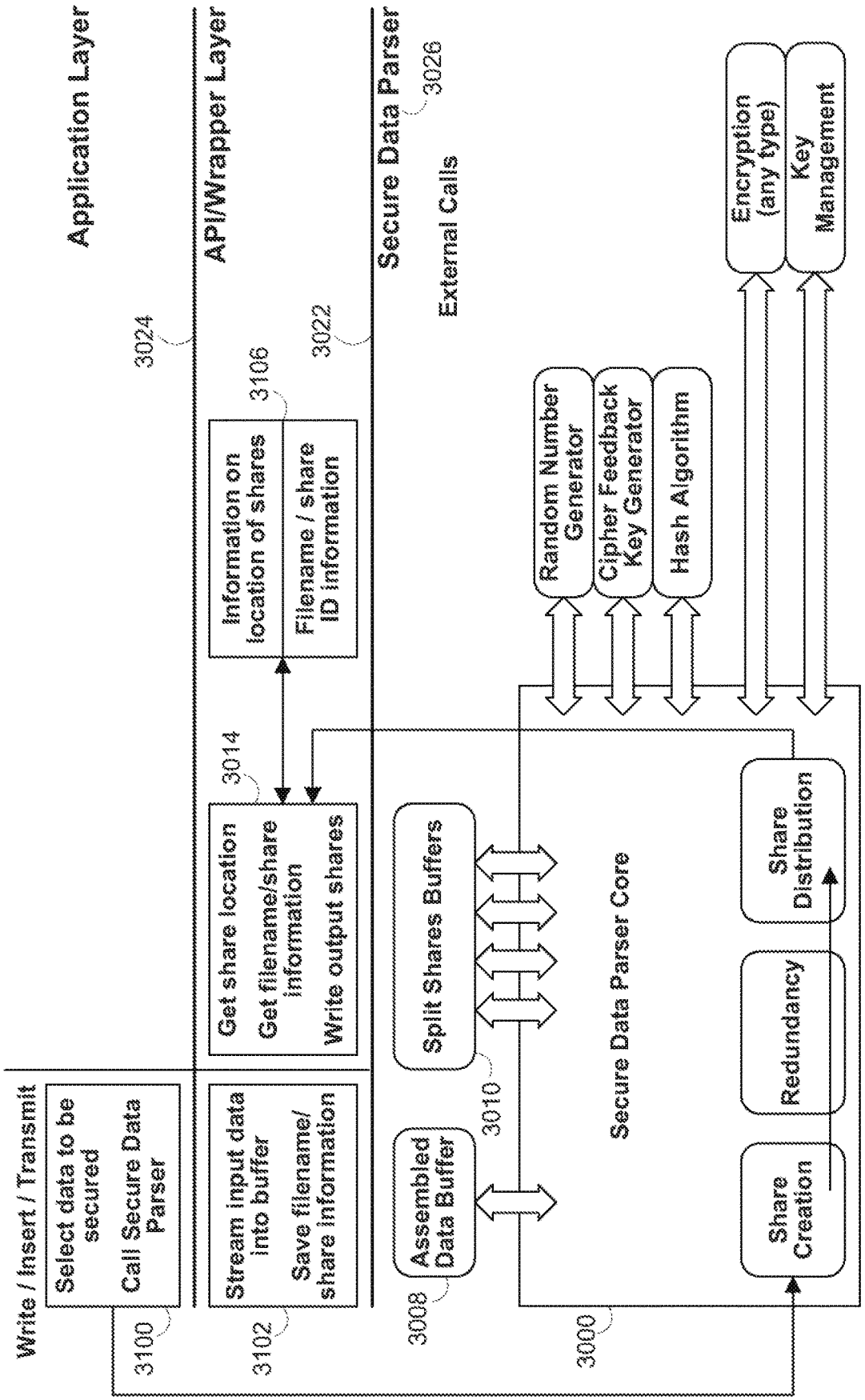

Application Layer
API/Wrapper Layer
Secure Data Parser
3026
External Calls
3024
3022
3106
Information on location of shares
Filename / share ID information
3014
Get share location
Get filename/share information
Write output shares
Write / Insert / Transmit
Select data to be secured
Call Secure Data Parser
Stream input data into buffer
Save filename/ share information
3100
3102
Random Number Generator
Cipher Feedback Key Generator
Hash Algorithm
Encryption (any type)
Key Management
Split Shares Buffers
3010
Assembled Data Buffer
3008
Secure Data Parser Core
3000
Share Distribution
Redundancy
Share Creation

FIG. 31

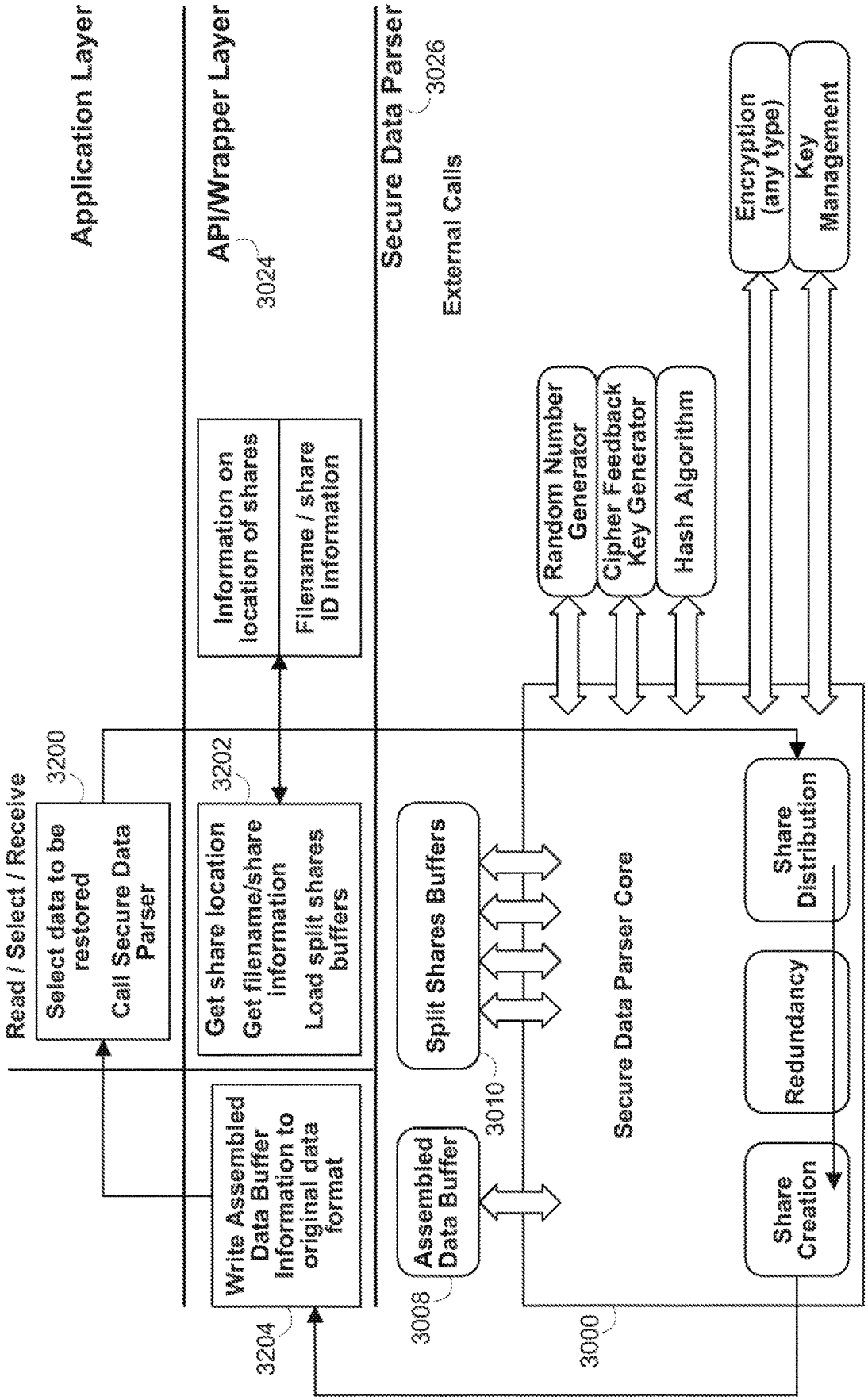

Application Layer
API/Wrapper Layer
3024
Secure Data Parser
3026
External Calls
Read / Select / Receive
3200
Select data to be restored
Call Secure Data Parser
3202
Get share location
Get filename/share information
Load split shares buffers
Information on location of shares
Filename / share ID information
Write Assembled Data Buffer information to original data format
3204
Split Shares Buffers
3010
Assembled Data Buffer
3008
Secure Data Parser Core
3000
Random Number Generator
Cipher Feedback Key Generator
Hash Algorithm
Encryption (any type)
Key Management
Share Distribution
Redundancy
Share Creation

FIG. 32

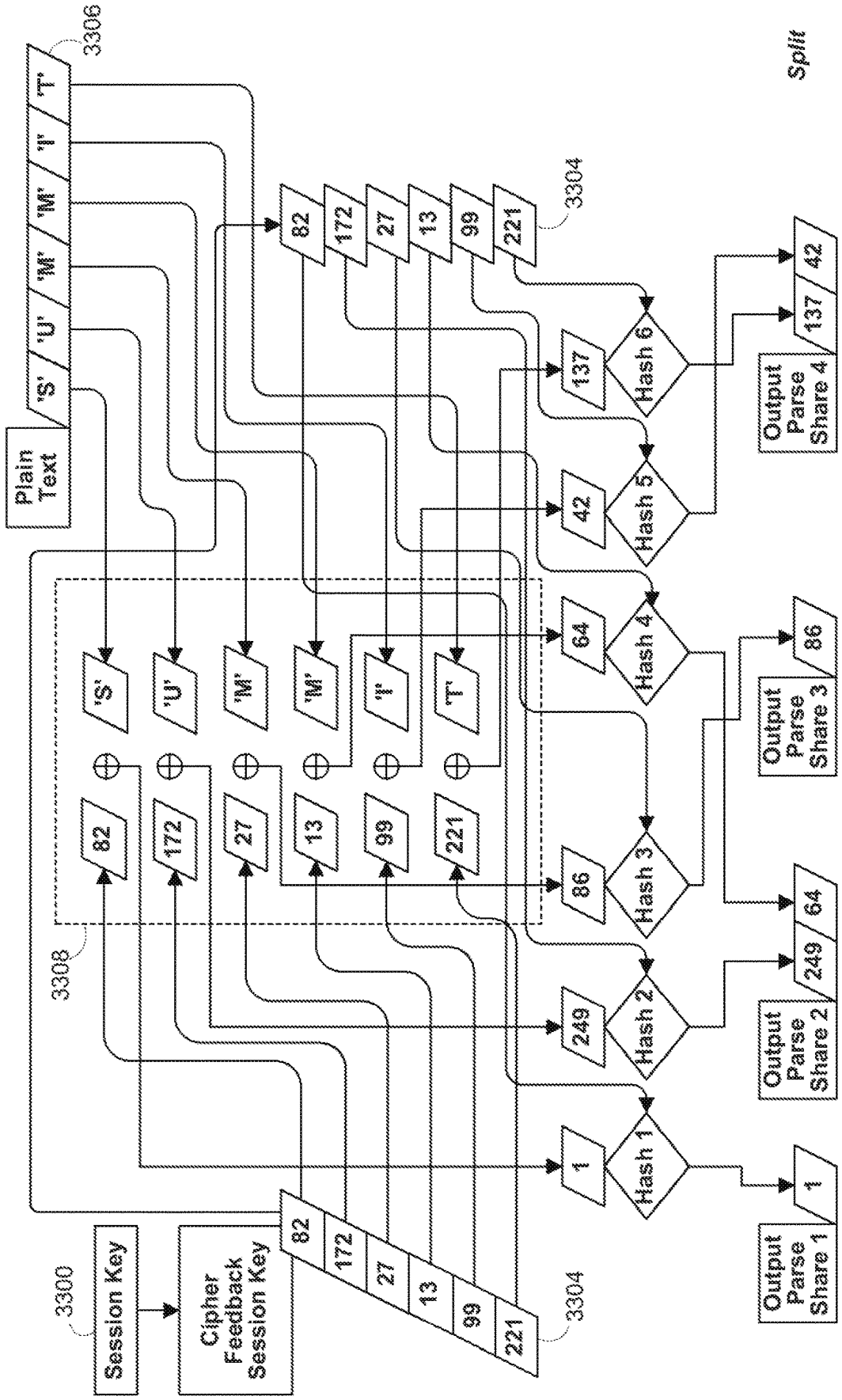

Plain Text
'S'
'U'
'M'
'M'
'I'
'T'
3306
Split
82
172
27
13
99
221
3304
137
42
64
86
249
1
Hash 1
Hash 2
Hash 3
Hash 4
Hash 5
Hash 6
Output Parse Share 1
Output Parse Share 2
Output Parse Share 3
Output Parse Share 4
3308
3300
Session Key
Cipher Feedback Session Key

FIG. 33

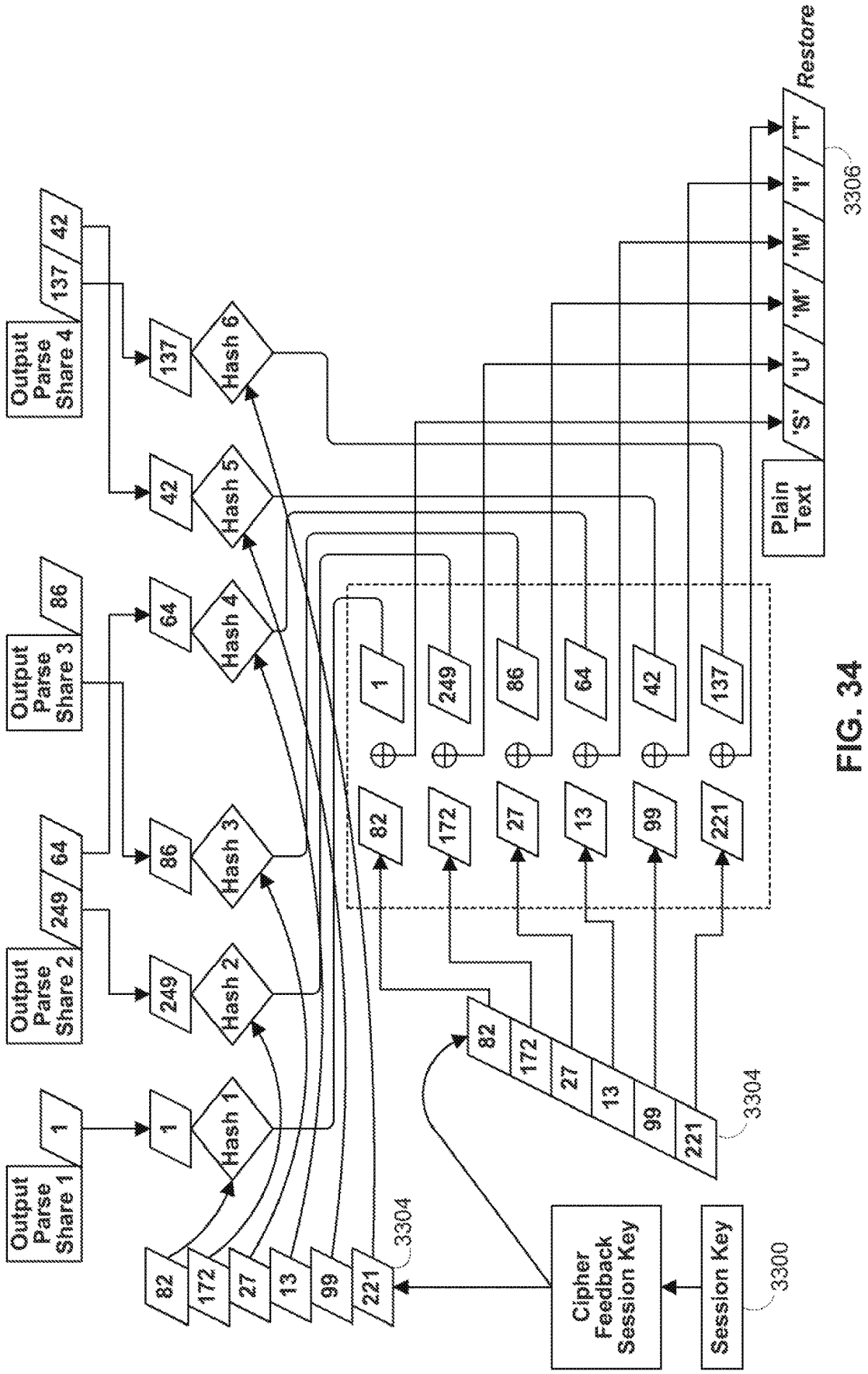

Output Parse Share 1
Output Parse Share 2
Output Parse Share 3
Output Parse Share 4
1
249
64
86
137
42
Hash 1
Hash 2
Hash 3
Hash 4
Hash 5
Hash 6
82
172
27
13
99
221
3304
Cipher Feedback Session Key
Session Key
3300
Plain Text
'S'
'U'
'M'
'M'
'I'
'T'
Restore
3306

FIG. 34

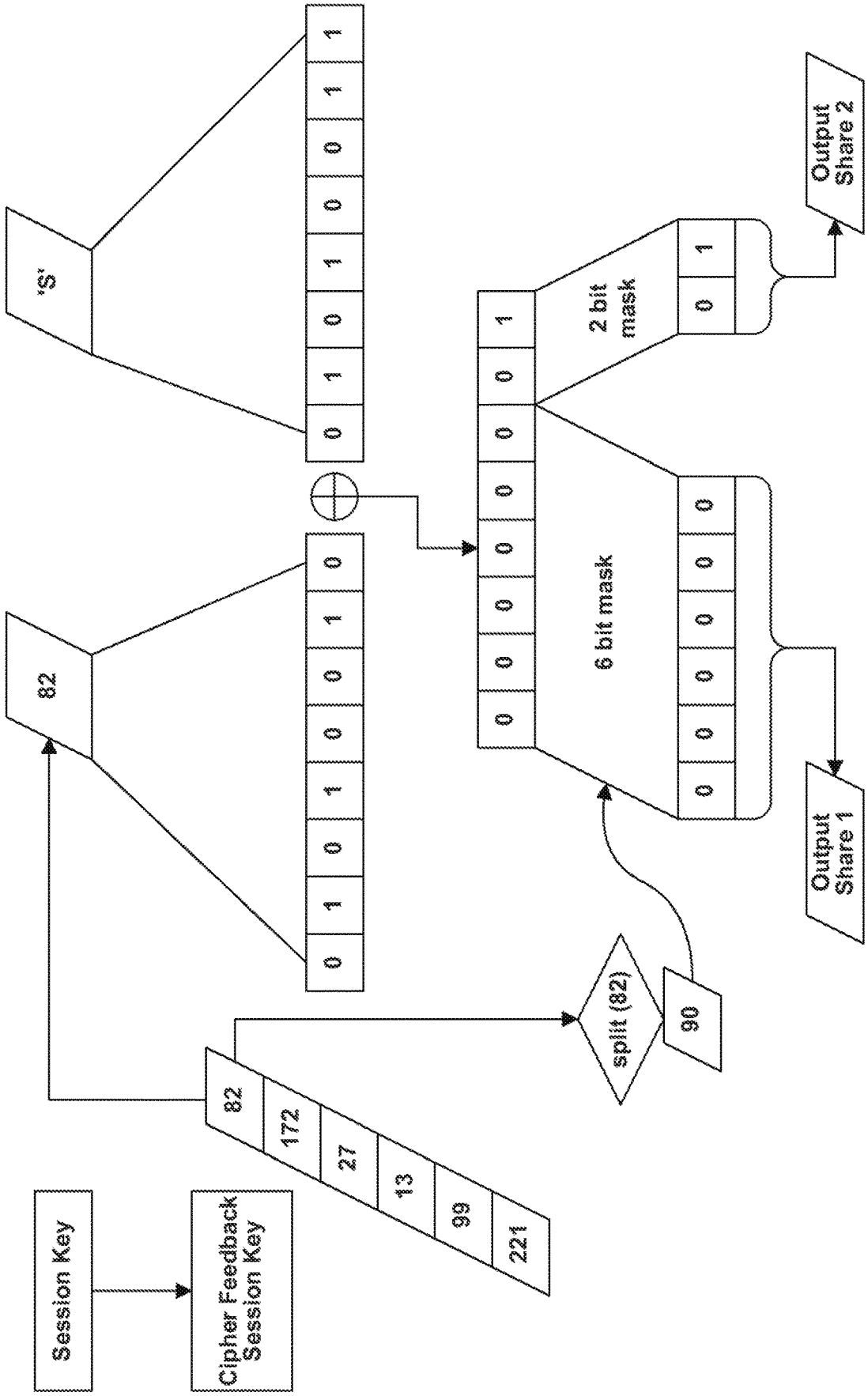

'S'
0 1 0 1 0 0 1 1
82
0 1 0 1 0 0 1 0
0 0 0 0 0 0 0 1
6 bit mask
2 bit mask
0 1
0 0 0 0 0 0
Output Share 2
Output Share 1
split (82)
90
82
172
27
13
99
221
Session Key
Cipher Feedback Session Key

FIG. 35

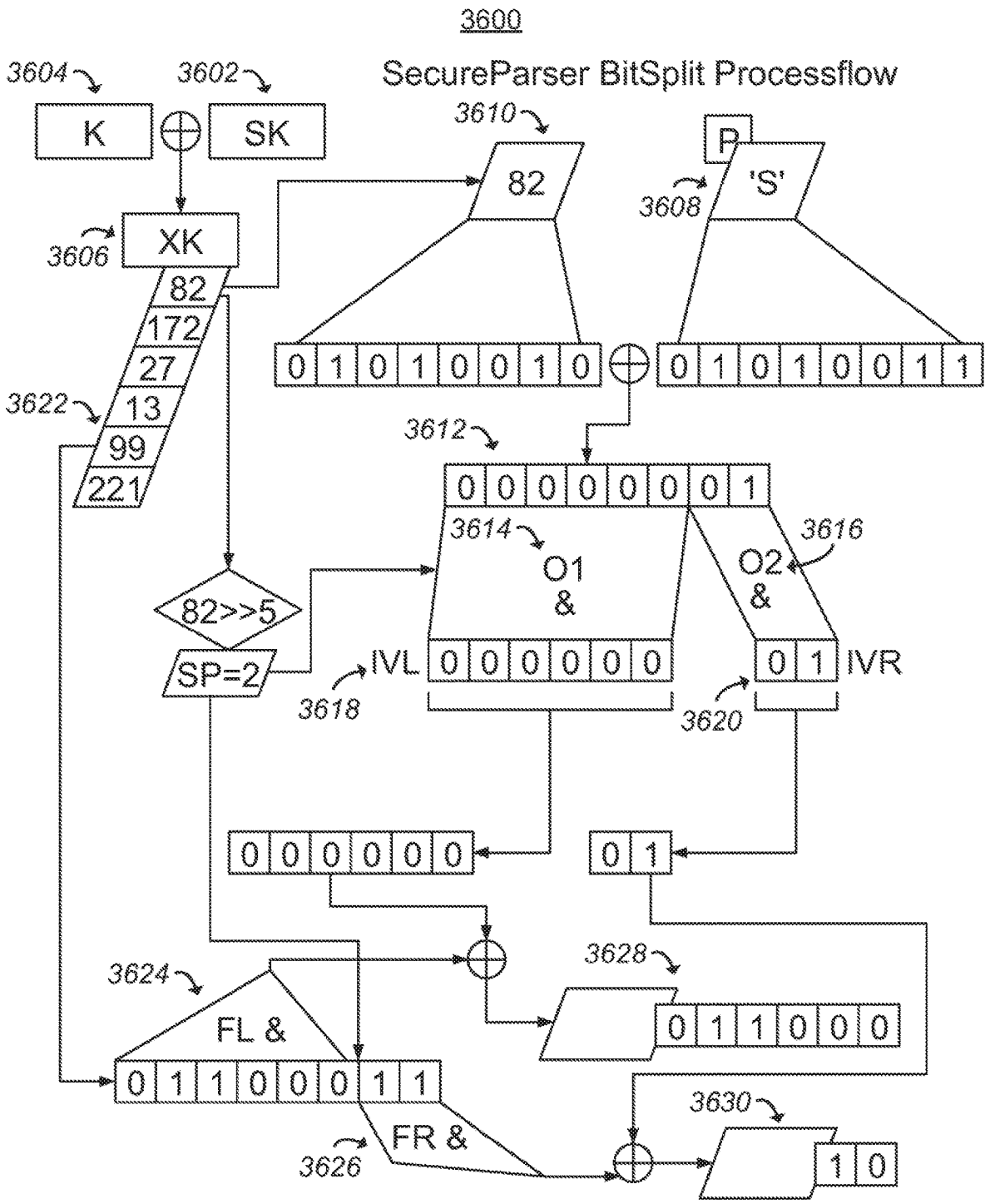

3600
SecureParser BitSplit Processflow
3604
3602
K
SK
3610
82
P
'S'
3608
3606
XK
82
172
27
13
99
221
3622
0 1 0 1 0 0 1 0
0 1 0 1 0 0 1 1
3612
0 0 0 0 0 0 0 1
3614
O1
&
3616
O2
&
82>>5
SP=2
IVL
0 0 0 0 0 0
0 1
IVR
3618
3620
0 0 0 0 0 0
0 1
3624
FL &
3628
0 1 1 0 0 0
0 1 1 0 0 0 1 1
3630
FR &
3626
1 0

FIG. 36

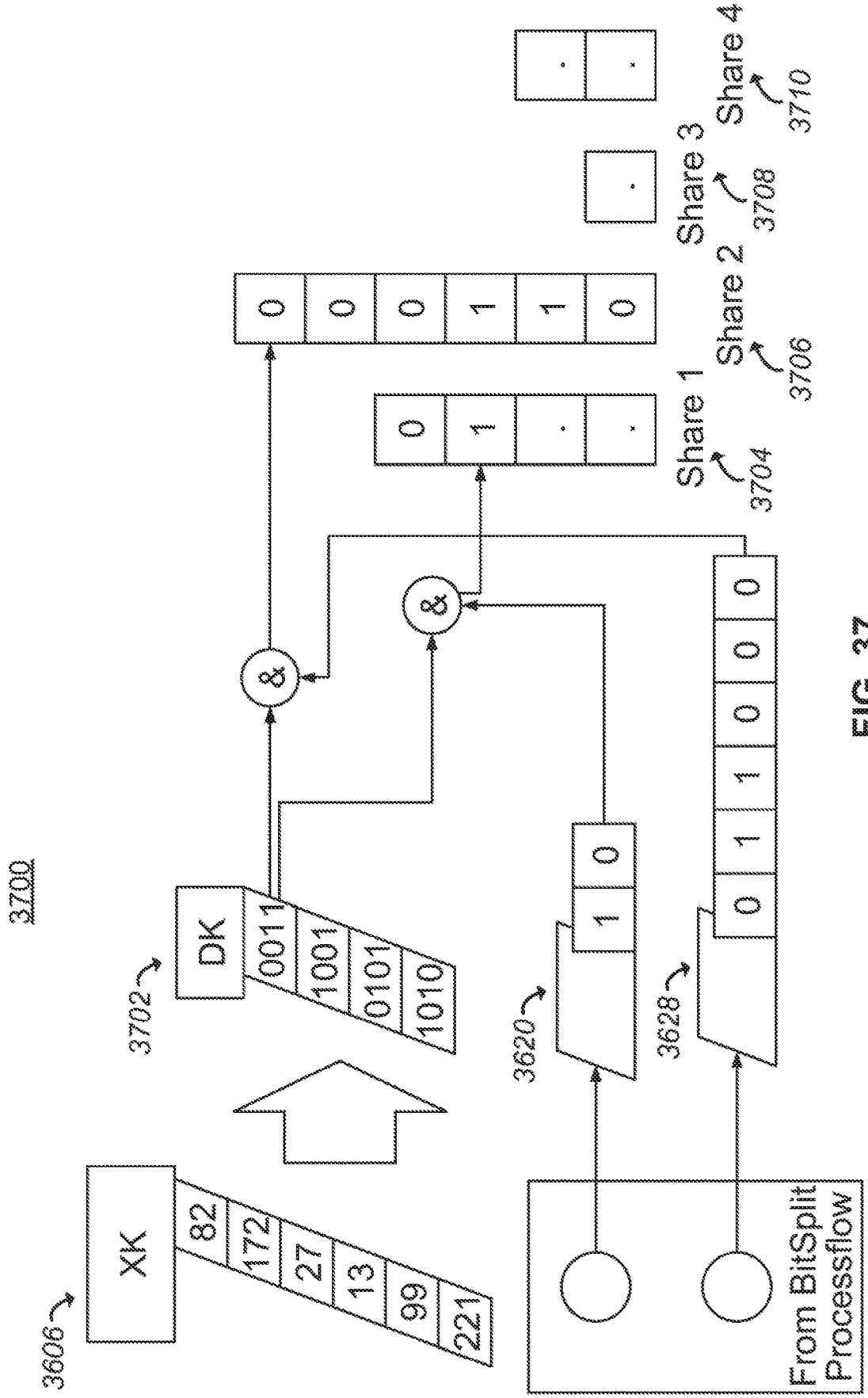

3700
3606
XK
82
172
27
13
99
221
3702
DK
0011
1001
0101
1010
&
&
0
0
0
1
1
0
0
1
.
.
Share 1
3704
Share 2
3706
Share 3
3708
Share 4
3710
3620
1
0
3628
0
1
1
0
0
0
From BitSplit Processflow

FIG. 37

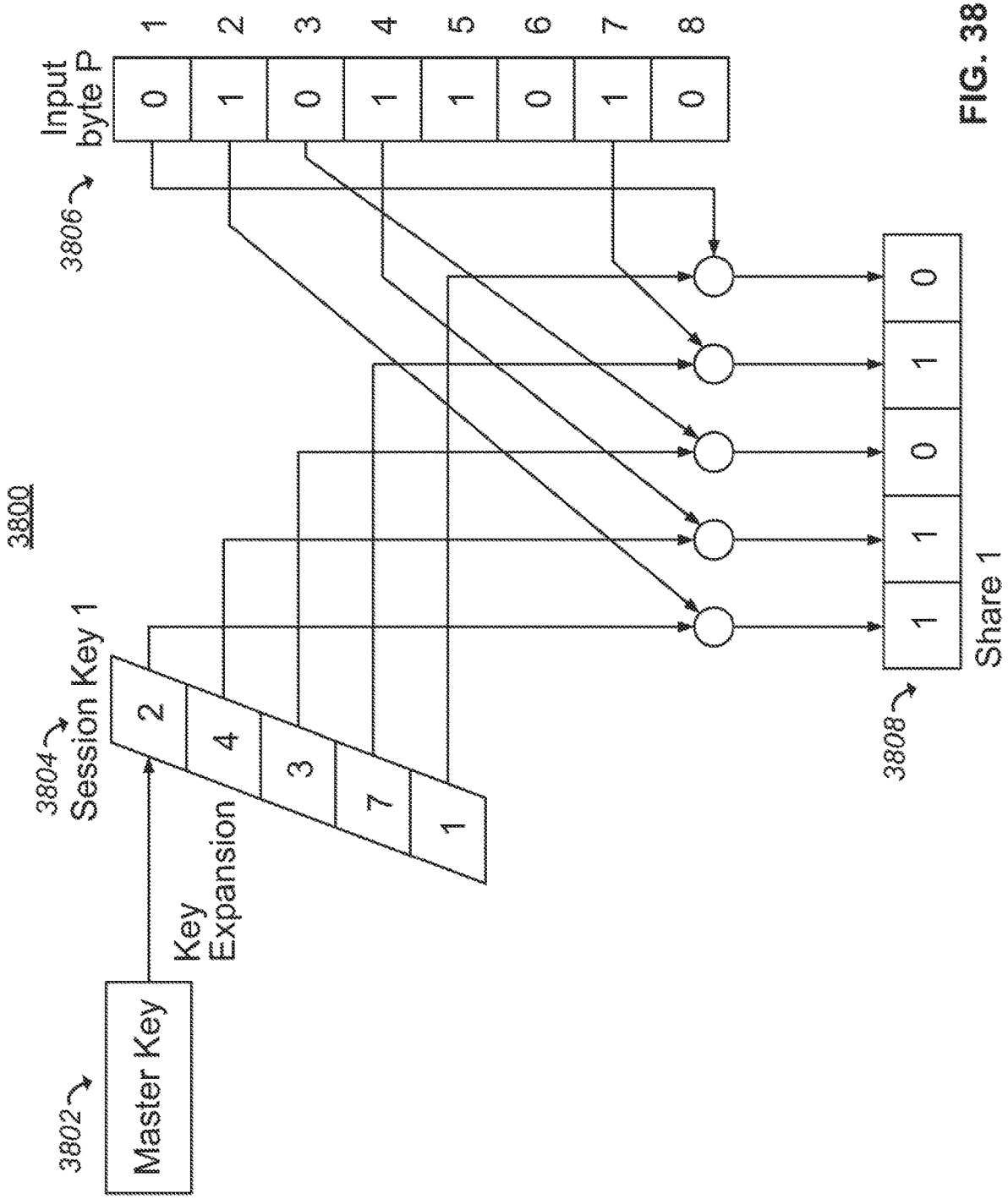

3800
3802
Master Key
Key Expansion
3804
Session Key 1
2
4
3
7
1
3806
Input byte P
0
1
0
1
1
0
1
0
1
2
3
4
5
6
7
8
3808
Share 1
1
1
0
1
0

FIG. 38

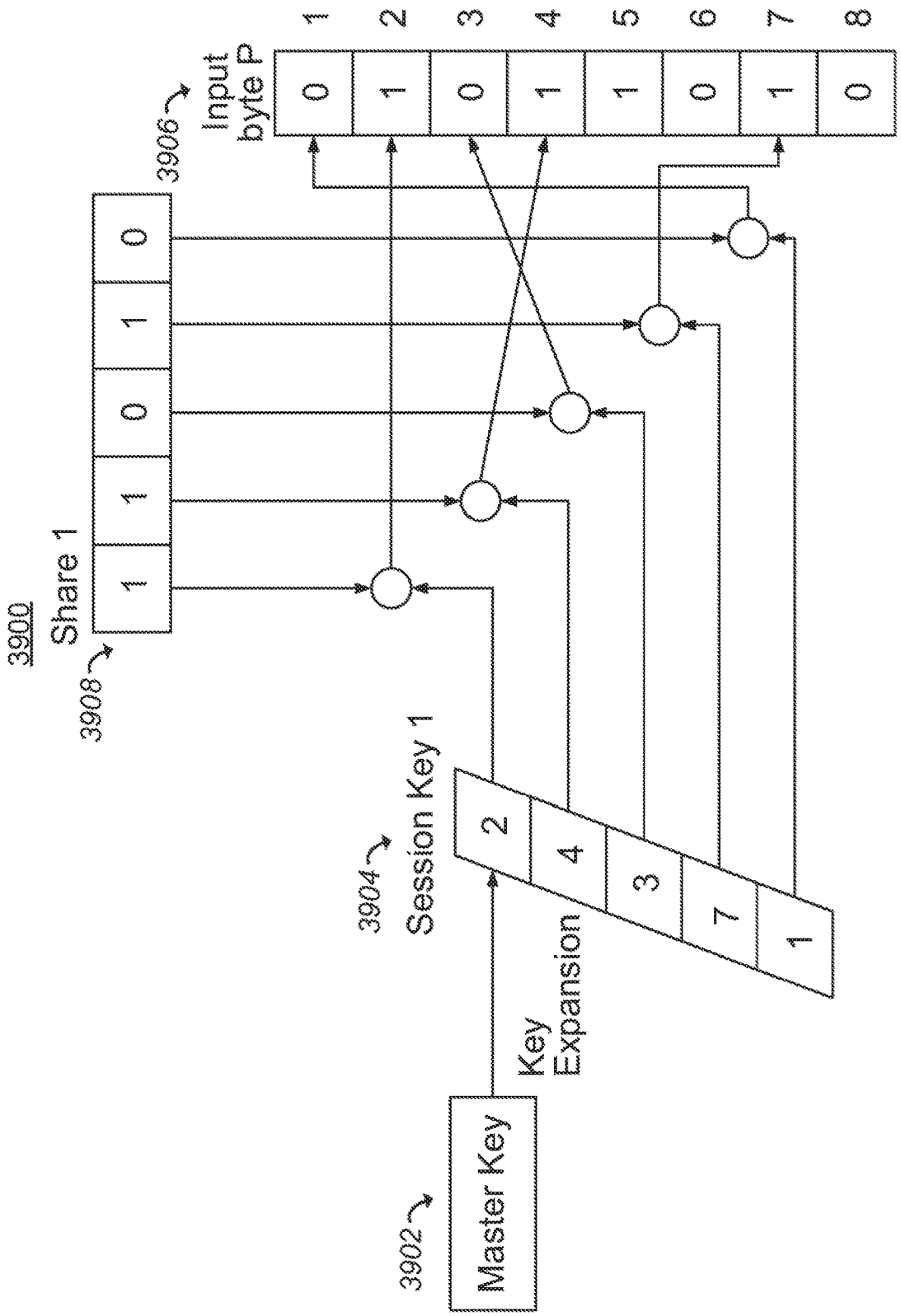

3900
Share 1
3908
1 1 0 1 0
3906
Input byte P
1 2 3 4 5 6 7 8
0 1 0 1 1 0 1 0
3904
Session Key 1
2 4 3 7 1
Key Expansion
3902
Master Key

FIG. 39

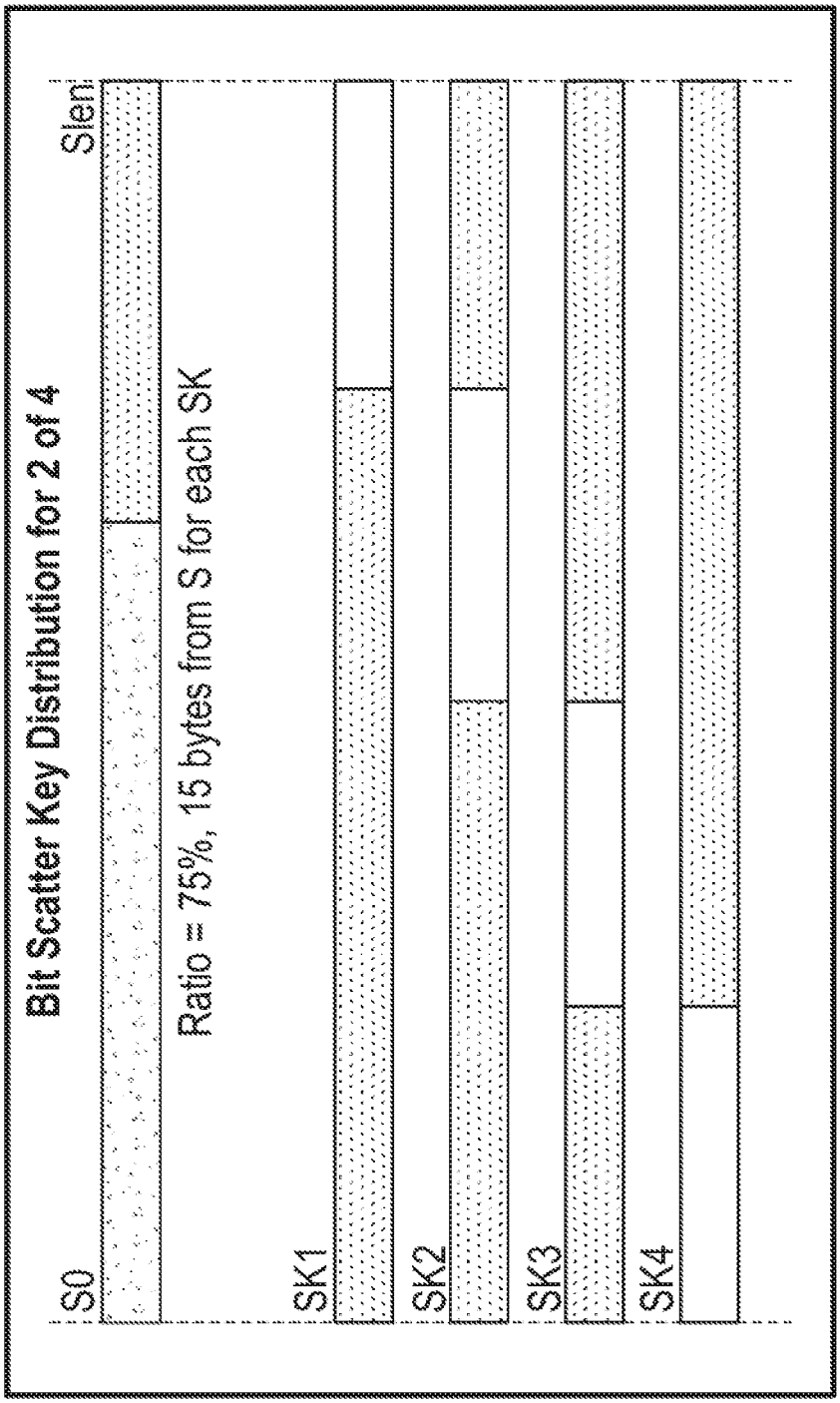

Bit Scatter Key Distribution for 2 of 4
S0
Slen
Ratio = 75%, 15 bytes from S for each SK
SK1
SK2
SK3
SK4

FIG. 40

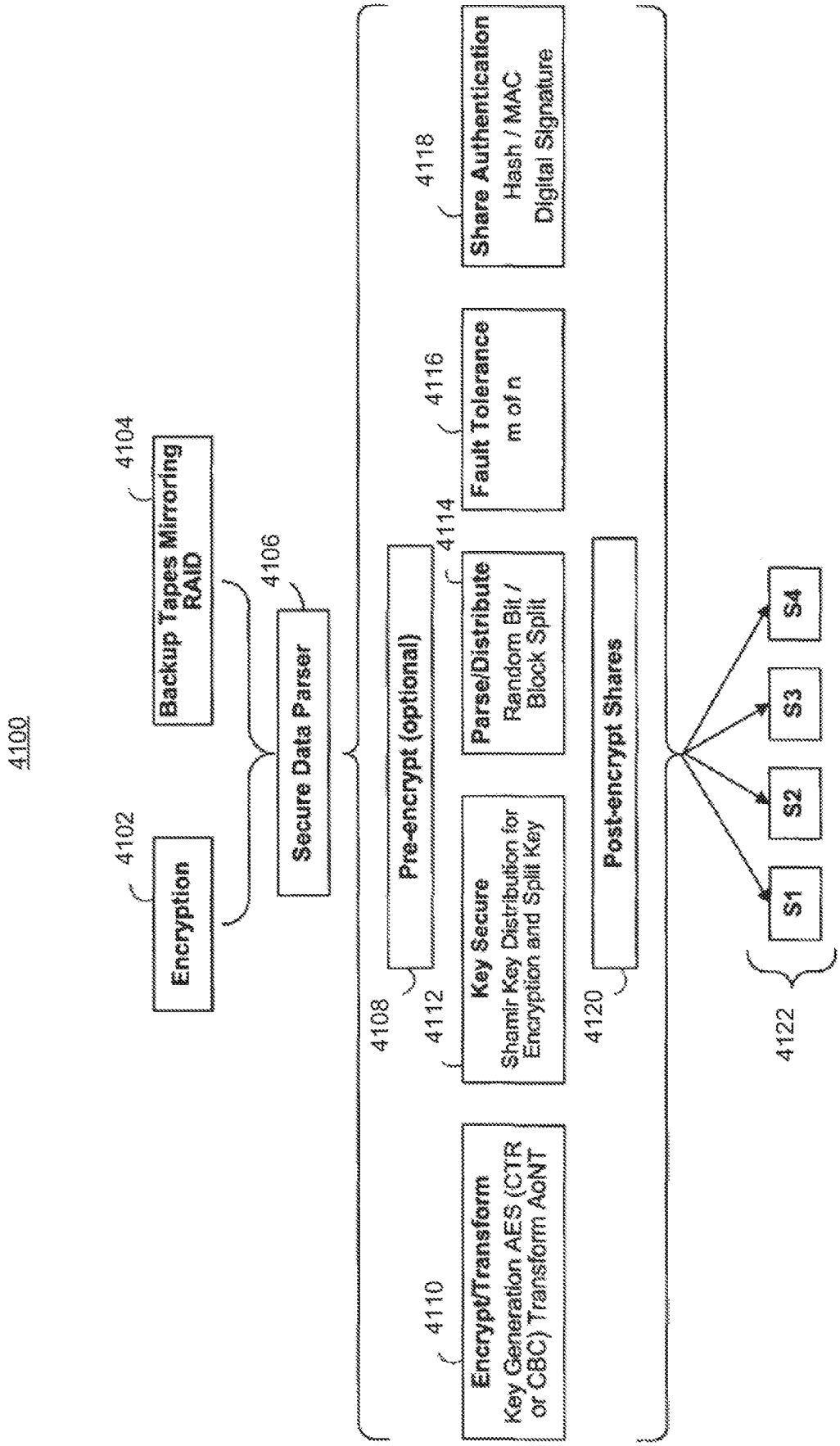

4100
4102
Encryption
4104
Backup Tapes Mirroring RAID
4106
Secure Data Parser
4108
Pre-encrypt (optional)
4110
Encrypt/Transform
Key Generation AES (CTR or CBC) Transform AONT
4112
Key Secure
Shamir Key Distribution for Encryption and Split Key
4114
Parse/Distribute
Random Bit / Block Split
4116
Fault Tolerance
m of n
4118
Share Authentication
Hash / MAC
Digital Signature
4120
Post-encrypt Shares
4122
S1
S2
S3
S4

FIG. 41

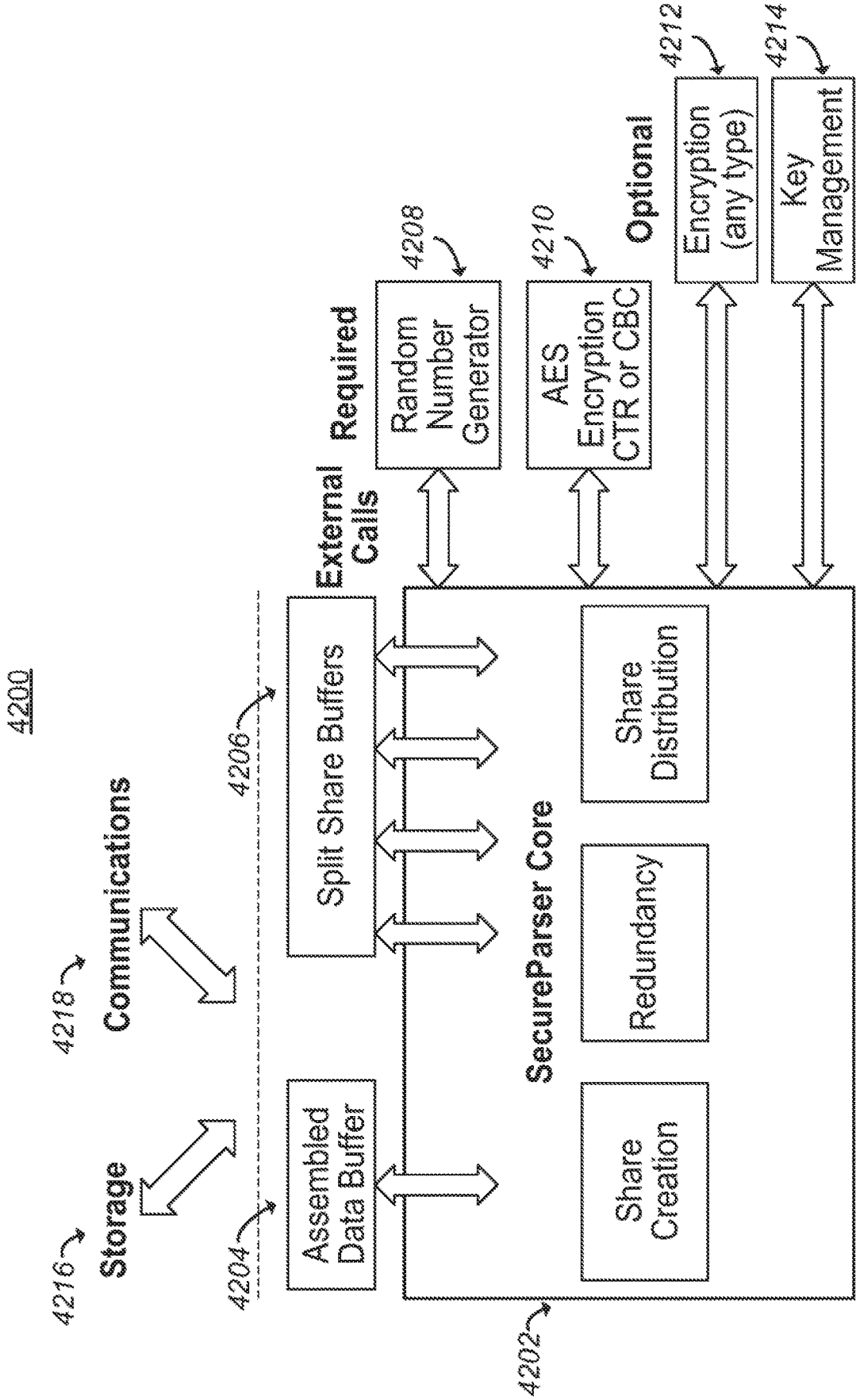

4200
4216
Storage
4218
Communications
4204
Assembled Data Buffer
4206
Split Share Buffers
External Calls
Required
4208
Random Number Generator
4210
AES Encryption CTR or CBC
Optional
4212
Encryption (any type)
4214
Key Management
SecureParser Core
Share Distribution
Redundancy
Share Creation
4202

FIG. 42

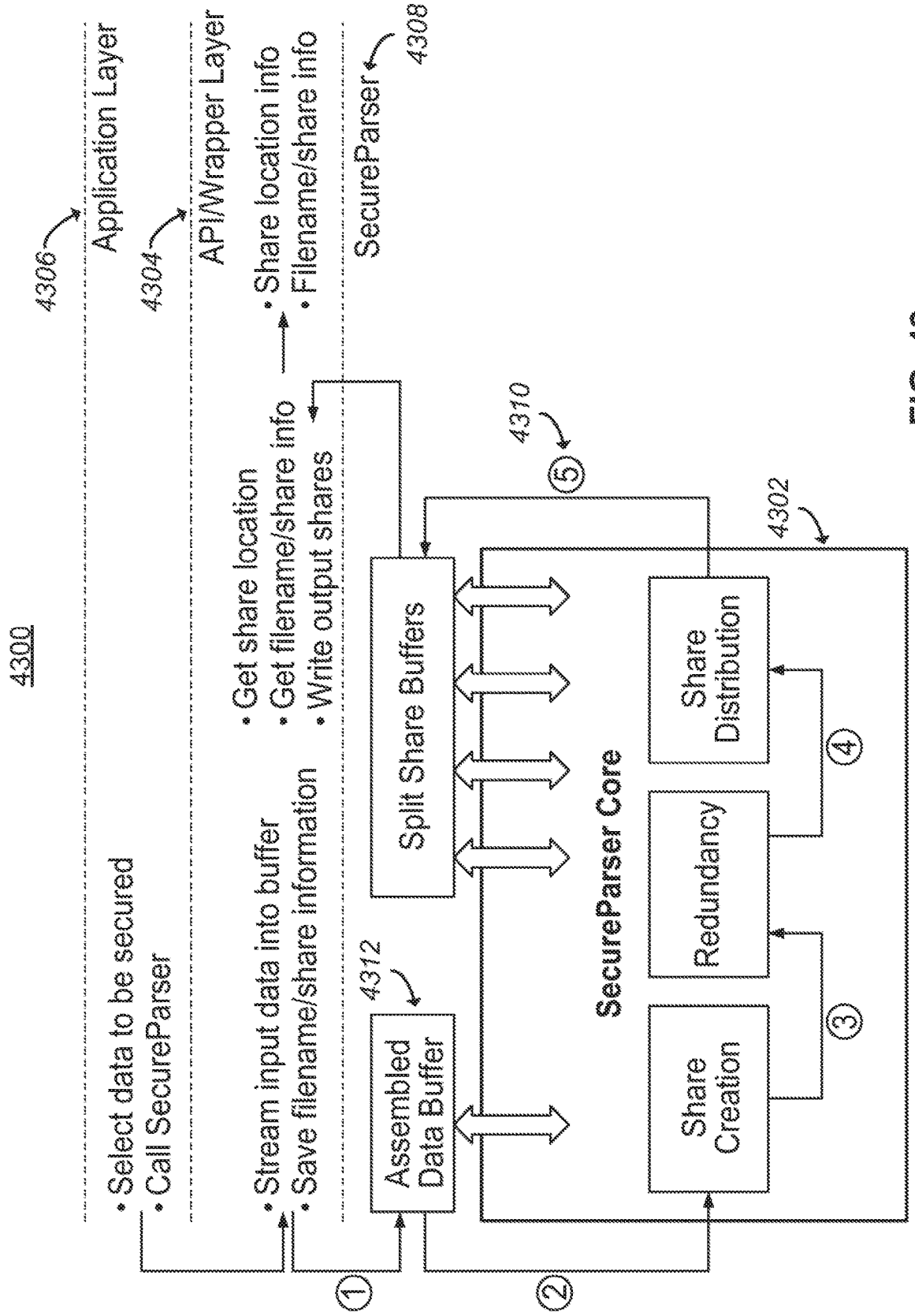

4300
4306
Application Layer
4304
API/Wrapper Layer
4308
SecureParser
• Select data to be secured
• Call SecureParser
• Stream input data into buffer
• Save filename/share information
• Share location info
• Filename/share info
• Get share location
• Get filename/share info
• Write output shares
4310
4302
Split Share Buffers
4312
Assembled Data Buffer
SecureParser Core
Share Distribution
Redundancy
Share Creation
1
2
3
4
5

FIG. 43

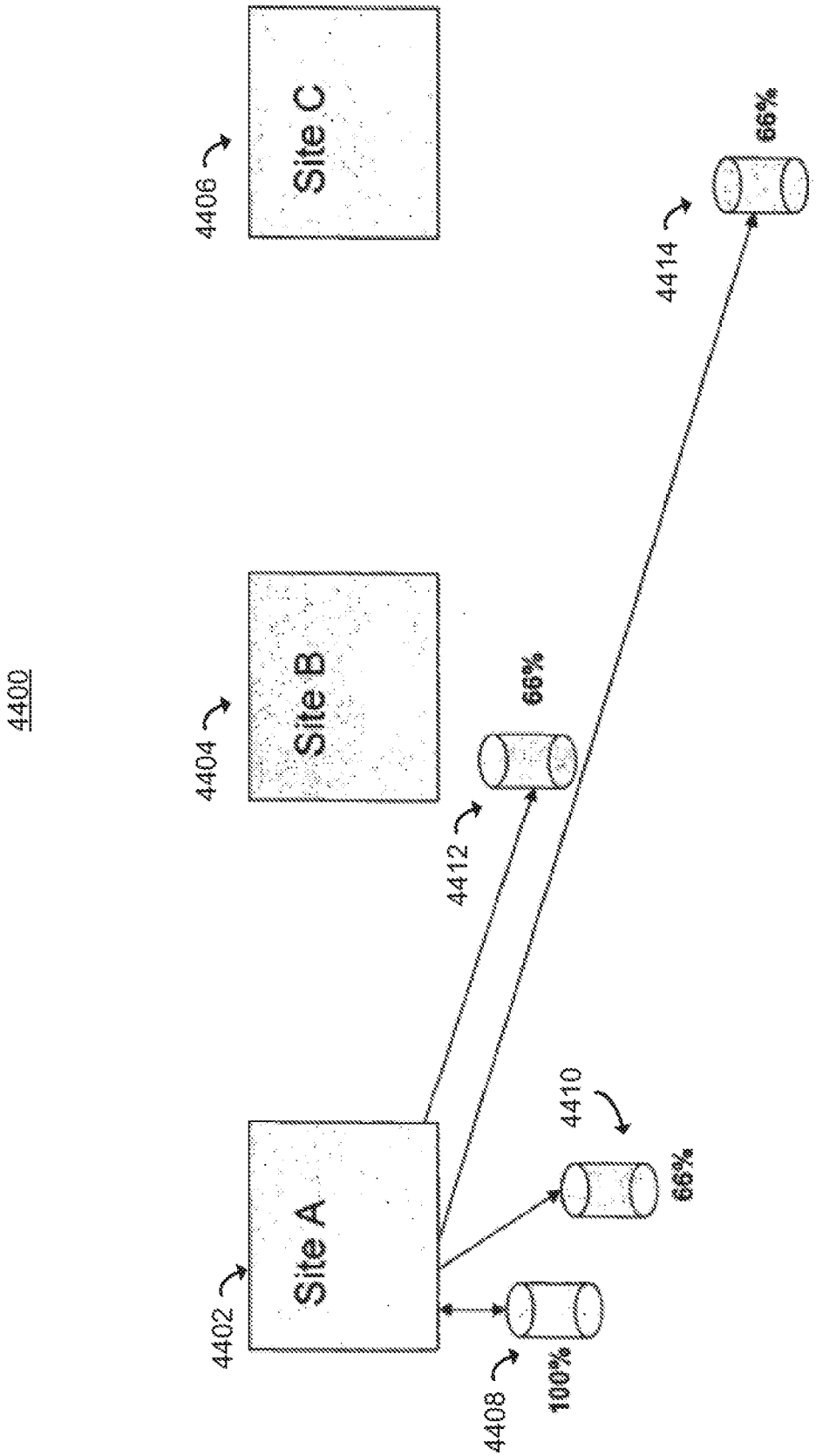

4400
4402
Site A
4404
Site B
4406
Site C
4408
100%
4410
66%
4412
66%
4414
66%

FIG. 44

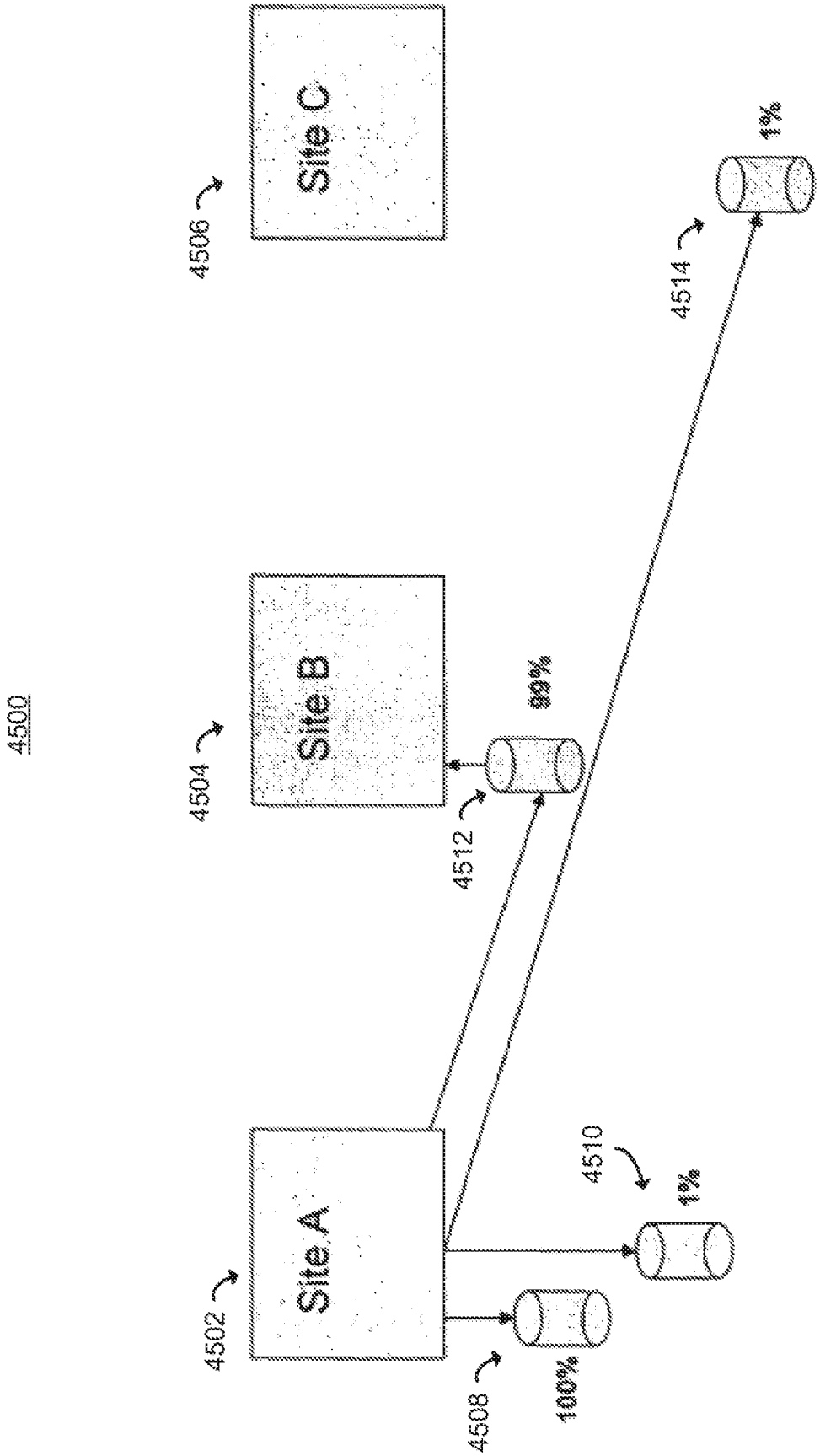

4500
Site A
4502
Site B
4504
Site C
4506
4508
100%
4510
1%
4512
99%
4514
1%

FIG. 45

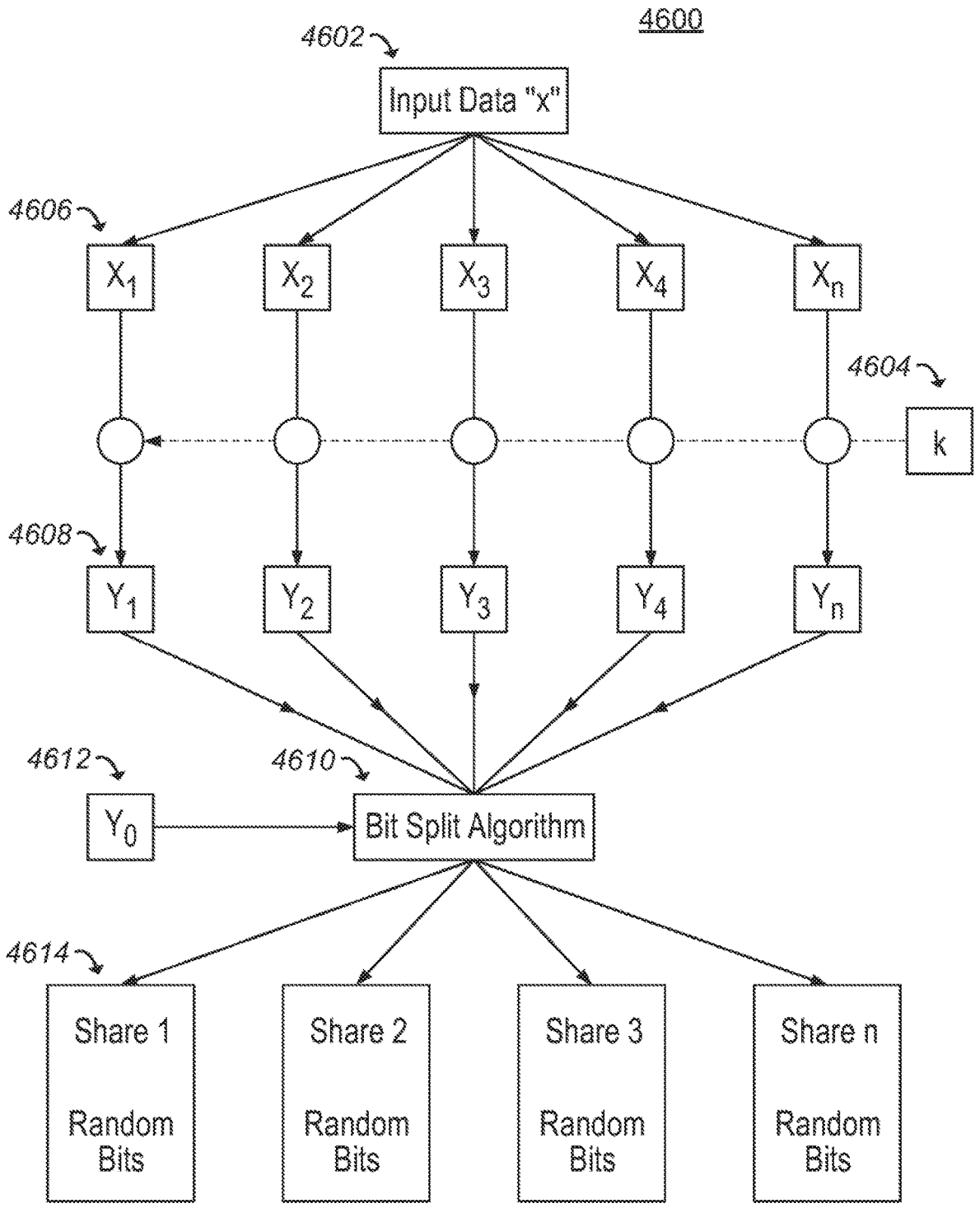

4600
4602
Input Data "x"
4606
X1
X2
X3
X4
Xn
4604
k
4608
Y1
Y2
Y3
Y4
Yn
4612
4610
Y0
Bit Split Algorithm
4614
Share 1
Random Bits
Share 2
Random Bits
Share 3
Random Bits
Share n
Random Bits

FIG. 46

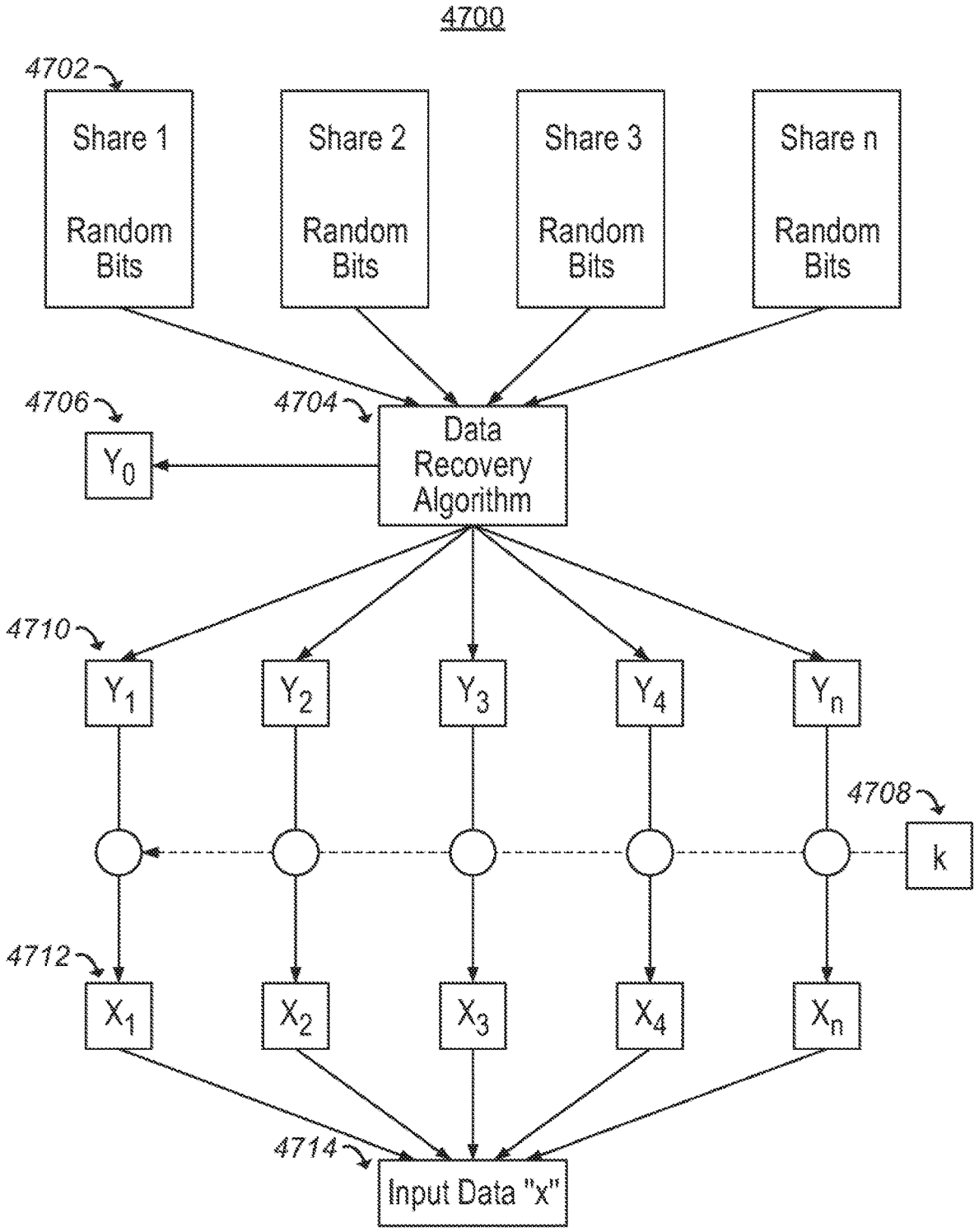

4700
4702
Share 1
Random Bits
Share 2
Random Bits
Share 3
Random Bits
Share n
Random Bits
4706
Y_0
4704
Data Recovery Algorithm
4710
Y_1
Y_2
Y_3
Y_4
Y_n
4708
k
4712
X_1
X_2
X_3
X_4
X_n
4714
Input Data "x"

FIG. 47

4800

2.2.1. ENC_TYPE

1. **ENC_NONE** - no encryption
2. **ENC_AES128CTR** - AES 128 bit in CTR mode
3. **ENC_AES192CTR** - AES 192 bit in CTR mode
4. **ENC_AES256CTR** - AES 256 bit in CTR mode
5. **ENC_AES128CBC** - AES 128 bit in CBC mode
6. **ENC_AES192CBC** - AES 192 bit in CBC mode
7. **ENC_AES256CBC** - AES 256 bit in CBC mode
8. **ENC_3DESCBC** - Triple DES in CBC mode

4802

2.2.2. AUTH_TYPE

1. **SIGN_RSAPSS** - RSA-PSS algorithm
2. **SIGN_DSA** - Digital Signature Algorithm
3. **SIGN_ECSA** - Elliptic Curve Digital Signature Algorithm
4. **SIGN_HMACSHA1** - HMAC-SHA1 symmetric Message Authentication Code

4804

2.2.3. HASH_TYPE

1. **HASH_NONE** - no hashing algorithm.
2. **HASH_SHA256** - SHA-256 hashing algorithm.

4806

2.2.4. SPLIT_TYPE

1. **SPLIT_NONE** - No split is used.
2. **SPLIT_BLOCK** - Target Data will be split at an block level determined by the splitting algorithm
3. **SPLIT_BYTE** - Target Data will be split at the byte level
4. **SPLIT_BIT** - Target Data will be split at the bit level
5. **SPLIT_BIT_SCATTER** - Target Data will be bit scattered.

4808

FIG. 48A

2.2.5. KEY_TYPE

1. **KEY_SYM** (keys for all symmetric algorithms)
2. **KEY_RSAPUB**
3. **KEY_RSAPRI**
4. **KEY_DSAPUB**
5. **KEY_DSAPRI**
6. **KEY_ECDSAPUB**
7. **KEY_ECDSAPRI**

4810

2.2.6. ERROR_TYPE

1. **ERROR_NONE** - No error
2. **ERROR_NO_MASTER_KEY** - No master key, can not continue.
3. **ERROR_INVALID_CALLBACK** - An invalid callback was registered
4. **ERROR_INVALID_METHOD** - An invalid parse split method occurred.
5. **ERROR_NO_TARGET** - No target was set to parse.
6. **ERROR_NO_TARGET_LENGTH** - Target buffer has no length
7. **ERROR_NOT_ENOUGH_SHARES** - Unable to restore, not enough shares are present.
8. **ERROR_INVALID_SHARE** - An invalid share was set
9. **ERROR_INVALID_KEY** - An invalid key was set
10. **ERROR_INTEGRITY_CHECK** - An integrity check failure occurred
11. **ERROR_NO_MEMORY** - Out of memory

4812

FIG. 48B

4900

4902

Name	Type	Description
originalShare	Share	stores the original unsplit data.
shares	Share []	an array of Share data structures of size MAX_SHARE_COUNT.
shareCount	int	actual number of shares used.
splitEncContext	EncContext *	buffer holding pre-encryption context between calls to doSplit and doRestore.
finalFlag	int	indicates that the final call to doSplit or doRestore will be made next (default = 0).

4904

Name	Type	Description
L	int	total number of mandatory shares (default = 0).
N	int	total number of non-mandatory shares (default = 0).
M	int	total number of non-mandatory shares needed for recovery (default = 0).
splitType	SPLIT_TYPE	the split algorithm to use (default = NONE).
splitEncMode	ENC_TYPE	the encryption mode to use during the split (default = NONE).
preEncMode	ENC_TYPE	the pre-encryption mode to use (default = NONE).
postEncMode	ENC_TYPE	the post-encryption mode to use (default = NONE).
splitHashMode	HASH_TYPE	the hash mode to use (default = HASH_SHA256).
preAuthMode	AUTH_TYPE	the pre-signing mode to use (default = NONE).
postAuthMode	AUTH_TYPE	the post-signing mode to use (default = NONE).
preEncKey	char *	handle for the pre-encryption key (default = NONE).
postEncKey	char *	handle for the post-encryption key (default = NONE).
preAuthKey	char *	handle for the pre-signature key (default = NONE).
postAuthKey	char *	handle for the post-signature key (default = NONE).
maxOrigBufSize	int	max size for the original buffer (default = MAX_ORIG_BUF_SIZE).
maxShareBufSize	int	max size for each share buffer (default = MAX_SHARE_BUF_SIZE).

FIG. 49A

4906

Name	Type	Description
id	char *	unique identifier for this key.
idLen	int	length of buffer holding id.

4908

Name	Type	Description
data	uint8 *	key share data for this share.
dataLen	int	length of data (in bits) of data buffer.

4910

Name	Type	Description
keyId	KeyId	unique identifier for this key.
type	KEY_TYPE	type of key that this.
data	uint8 *	pointer to the buffer holding the key data.
dataLen	int	length of the buffer holding the key data.

4912

Name	Type	Description
data	uint8 *	data buffer for this share.
dataLen	int	length indicating the amount of data found in this share.
encContext	EncContext	data buffer storing the encryption context between calls to doSplit and doRestore.
authContext	SigContext	data buffer storing the authentication context between calls to doSplit and doRestore.
hashContext	HashContext	data buffer storing the hash context between calls to doSplit and doRestore.

FIG. 49B

5002 5000

Name	Type	Description
encMode	ENC_TYPE	the type of encryption being used.
...	Union	various types of encryption contexts can be placed here.

5004

Name	Type	Description
encKeyId	KeyId	unique identifier of the encryption key to use for this context.
currentIV	uint8 *	the current IV being stored for this context.
currentIdLen	int	the length of the current IV in bytes.

5006

Name	Type	Description
encKeyId	KeyId	unique identifier of the encryption key to use for this context.
currentIV	uint8 *	the current IV being stored for this context.
currentIVLen	int	the length of the current IV in bytes.

5008

Name	Type	Description
authMode	AUTH_TYPE	the type of authentication being used.
...	Union	various types of authentication contexts can be placed here.

5020

Name	Type	Description
hashContext	openSSL?	Pointer to the OpenSSL hash context data type.

FIG. 50A

5010

Name	Type	Description
pubKeyId	KeyId	unique identifier of the public key to use for this context.
privKeyId	KeyId	unique identifier of the private key to use for this context.
hashContext	openSSL?	Pointer to the OpenSSL hash context data type.

5012

Name	Type	Description
pubKeyId	KeyId	unique identifier of the public key to use for this context.
privKeyId	KeyId	unique identifier of the private key to use for this context.
hashContext	openSSL?	Pointer to the OpenSSL hash context data type.

5014

Name	Type	Description
pubKeyId	KeyId	unique identifier of the public key to use for this context.
privKeyId	KeyId	unique identifier of the private key to use for this context.
hashContext	openSSL?	Pointer to the OpenSSL hash context data type.

5016

Name	Type	Description
macContext	openSSL?	Pointer to the OpenSSL mac context data type.

5018

Name	Type	Description
hashMode	HASH_TYPE	the type of hashing being used.
...	Union	various types of hashing contexts can be placed here.

FIG. 50B

5100

Share Data
\n----- BEGIN POST-ENC KEY ID -----\n
post-encryption key id
\n----- END POST-ENC KEY ID -----\n
\n----- BEGIN POST-SIG KEY ID -----\n
post-signature key id
\n----- END POST-SIG KEY ID -----\n
\n----- BEGIN POST-ENC HEADER -----\n
post-encrypted header
\n----- END POST-ENC HEADER -----\n
\n----- BEGIN POST-ENC DATA -----\n
post-encrypted data
\n----- END POST-ENC DATA-----\n
\n----- BEGIN POST-ENC FOOTER -----\n
post-encrypted footer
\n----- END POST-ENC FOOTER -----\n
\n----- BEGIN POST-SIG DATA -----\n
post-signature data
\n----- END POST-SIG DATA-----\n

5102

Encrypted footer
\n----- BEGIN SPLIT-HASH DATA -----\n
split-hash data
\n----- END SPLIT-HASH DATA -----\n
\n----- BEGIN PRE-SIG DATA -----\n
pre-signature data
\n----- END PRE-SIG DATA -----\n

5108

FIG. 51A

Post-encrypted footer
\n------ BEGIN ENCRYPTED FOOTER ------\n
encrypted footer
\n------ END ENCRYPTED FOOTER ------\n

5104

Post-encrypted header
\n------ BEGIN HEADER ------\n
Key share (len):key share information\n
Share id (2):share ID\n
L (2):L\n
M (2):M\n
N (2):N\n
Split mode (1):split mode\n
Split-encryption key (len):split-encryption key id\n
Split-encryption IV (len):split-encryption IV data\n
Split-hash mode (1):split-hash mode\n
Original buffer (4):original buffer size\n
Share buffers (4):share buffer size\n
Pre-encryption key (len):pre-encryption key id\n
Pre-signature key (len):pre-signature key id\n
\n------ END HEADER ------\n

5106

FIG. 51B

5200

parser_createParser 5202
Parameters none
Returns Parser *p
Description Allocates the SecureParser context, and sets default values.

parser_destroyParser 5204
Parameters Parser *p
Returns void
Description Destroys the SecureParser context, including parser parameter objects and share buffers.

parser_createKey 5206
Parameters char *id
uint8 *data
size_t length
KEYTYPE type
Returns Key *k
Description Constructs the structure **k** for use with the parser from the unique id, key data buffer, length indicater and key type given.

parser_destroyKey 5208
Parameters Key *k
Returns *none*
Description Destroys the key pointed to by k. Note: Do not destroy a key currently in use with the Parser.

parser_createShare 5210
Parameters int length
Key *encKey
Key *authKey
HASH_MODE h
Returns Share *s
Description Constructs a Share data structure based on the given length, the encryption key type, authentication key type and hash mode.

FIG. 52A

parser_destroyShare
Parameters Share *s
Returns none
Description Destroys the share pointed to by s, including all encryption and authentication keys.

5212

parser_initParameters
Parameters ParserParams *params
Returns int error
Description Initializes a ParserParams data structure to the default parameter set. The return error should be 0 for success, and non-zero error message for failure.

5214

parser_setParameters
Parameters Parser *p, ParserParams *params
Returns int error
Description Deletes any old ParserParams that may have already been added to the Parser struct. Creates a copy of the new ParserParams pointed to by params for internal use and stores in the Parser pointed to by p. If params is altered or destroyed by the user, Parser p will still have retained a copy for itself that is not affected. Any data structures that must be created by the parameters struct, such as Shares or Keys will be created. The Parser will destroy the params object itself upon being closed.

5216

parser_setFinalize
Parameters Parser *p
Returns none
Description Sets the internal state of the Parser to finalize the output stream during the next SecureParser split session. After setFinalize is called, only one more call to doSessionSplit may be made.

5218

parser_getOriginalShare
Parameters Parser *p
Returns Share *s
Description Returns a Share structure representing the original buffer used by the Parser.

5220

FIG. 52B

5200

parser_getShare
Parameters Parser *p,
int ShareID
Returns Share *s
Description Returns a Share containing the Share structure for the Share matching the given ShareID.

5222

parser_doSplit
Parameters Parser *p
Returns int error
Description Performs the SecureParser split. This action treats data that is in the target buffer as input and places split data that is output into buffers for each Share. A return value of 0 indicates success, otherwise a failure has occurred and the error is returned. If setFinalize has been called, this must be the last

5224

parser_doRestore
Parameters Parser *p
Returns int error
Description Performs the SecureParser restore. This action treats all data in share buffers as input and places output data (the reconstructed original data) into the target buffer. A return value of 0 indicates success, otherwise a failure has occured and the error is returned.

5226

parser_generateHeaders
Parameters Parser *p
Returns int error
Description Generates the necessary header information based on the parser's current parameter set and state, and puts this header information appropriately within each Share buffer.

5228

parser_generateFooters
Parameters Parser *p
Returns int error
Description Generates the necessary footer information based on the parser's current parameter set and state, and puts this footer information appropriately within each Share buffer.

5230

FIG. 52C

parser_restoreHeaders

Parameters Parser *p
Returns int error
Description This function assumes that the data within each Share buffer is header information and will be restored into the Parser, its parameters reset and its state changed appropriately.

5232

parser_restoreFooters

Parameters Parser *p
Returns int error
Description This function assumes that the data within each Share buffer is footer information and will be restored into the Parser, its parameters reset and its state changed appropriately.

5234

parser_getRestoreStatus

Parameters Parser *p
Returns int status
Description This function will return the status of the prior restore process. Either a success value will be returned if the restore process has completed with no errors, or the appropriate error status.

5236

FIG. 52D

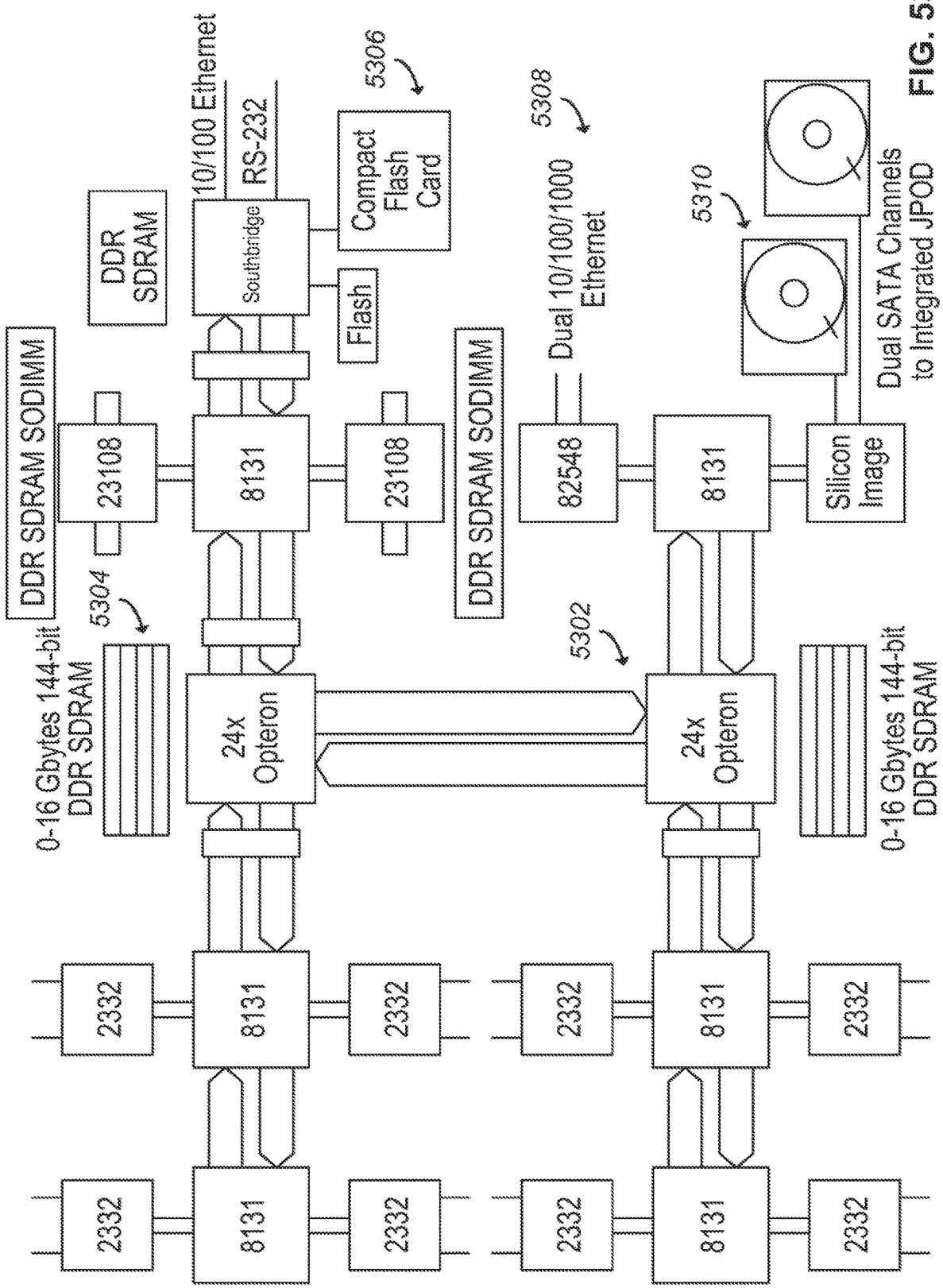

DDR SDRAM SODIMM
23108
DDR SDRAM
10/100 Ethernet
RS-232
Southbridge
8131
Flash
Compact Flash Card
5306
23108
DDR SDRAM SODIMM
5308
Dual 10/100/1000 Ethernet
82548
8131
5310
Silicon Image
Dual SATA Channels to Integrated JPOD
5304
0-16 Gbytes 144-bit DDR SDRAM
24x Opteron
5302
24x Opteron
0-16 Gbytes 144-bit DDR SDRAM
2332
8131
2332
2332
8131
2332
2332
8131
2332
2332
8131
2332

FIG. 53

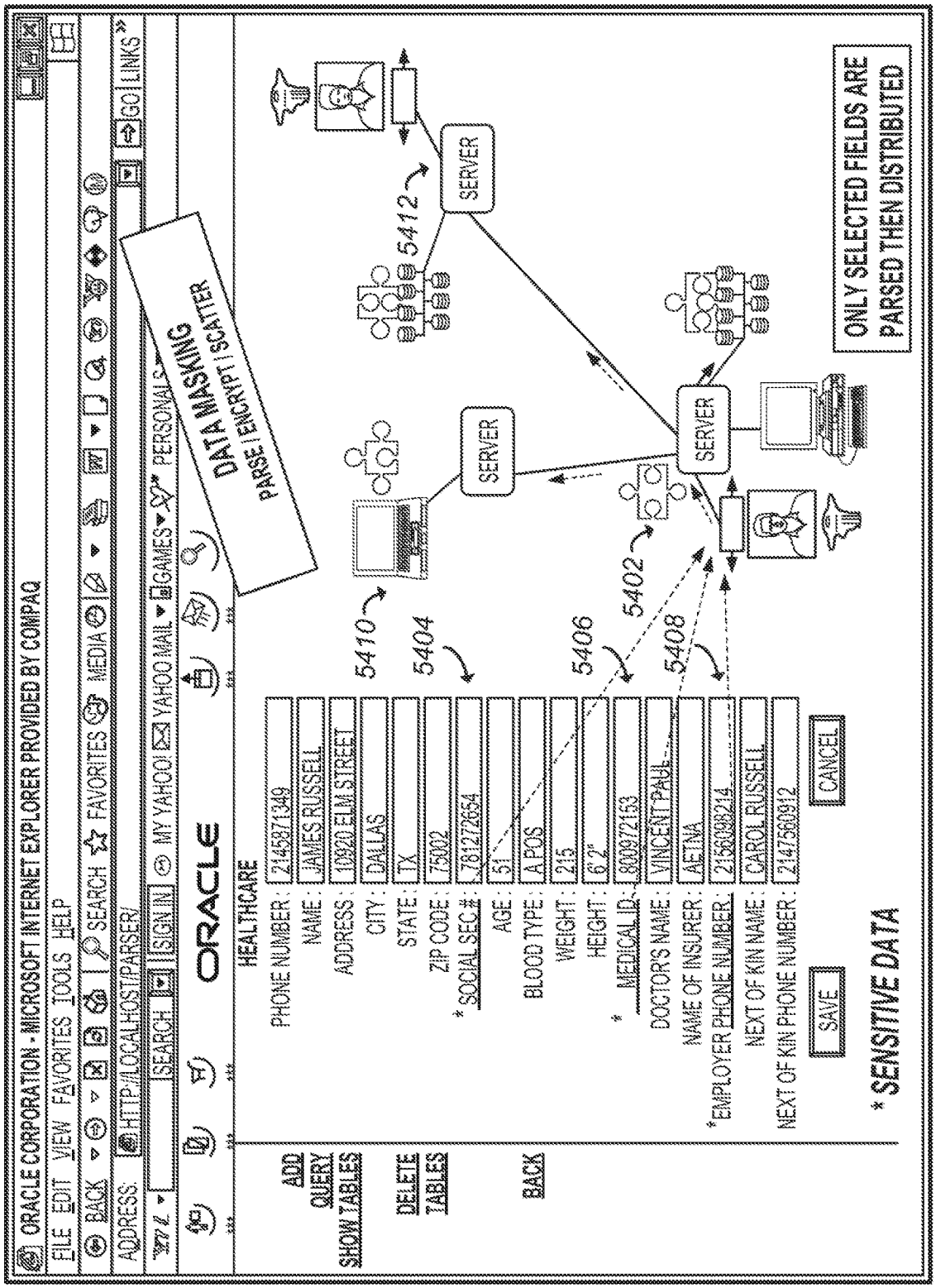

ORACLE CORPORATION - MICROSOFT INTERNET EXPLORER PROVIDED BY COMPAQ
FILE EDIT VIEW FAVORITES TOOLS HELP
BACK SEARCH FAVORITES MEDIA
ADDRESS: HTTP://LOCALHOST/PARSER/
GO LINKS
SEARCH SIGN IN MY YAHOO! YAHOO MAIL GAMES PERSONAL
ORACLE
DATA MASKING
PARSE / ENCRYPT / SCATTER
ADD QUERY SHOW TABLES
DELETE TABLES
BACK
HEALTHCARE
PHONE NUMBER: 2145871349
NAME: JAMES RUSSELL
ADDRESS: 10920 ELM STREET
CITY: DALLAS
STATE: TX
ZIP CODE: 75002
* SOCIAL SEC #: 781272654
AGE: 51
BLOOD TYPE: A POS
WEIGHT: 215
HEIGHT: 6' 2"
* MEDICAL ID: 800972163
DOCTOR'S NAME: VINCENT PAUL
NAME OF INSURER: AETNA
*EMPLOYER PHONE NUMBER: 2156098214
NEXT OF KIN NAME: CAROL RUSSELL
NEXT OF KIN PHONE NUMBER: 2147560912
SAVE
CANCEL
* SENSITIVE DATA
5410
5404
5406
5408
5402
5412
SERVER
SERVER
SERVER
ONLY SELECTED FIELDS ARE PARSED THEN DISTRIBUTED

FIG. 54

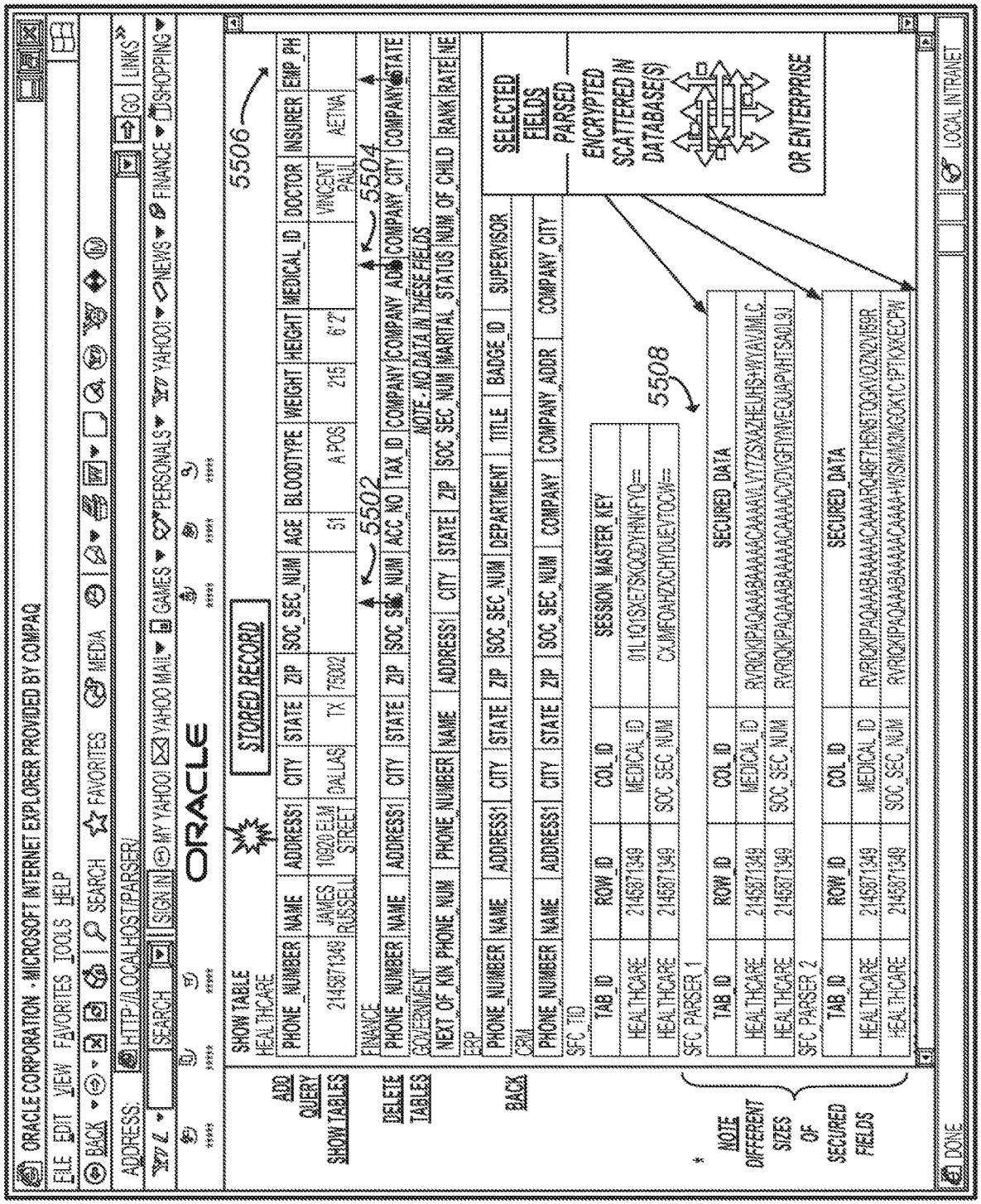

ORACLE CORPORATION - MICROSOFT INTERNET EXPLORER PROVIDED BY COMPAQ
FILE EDIT VIEW FAVORITES TOOLS HELP
BACK SEARCH FAVORITES MEDIA
ADDRESS HTTP://LOCALHOST/PARSER/
GO LINKS
SIGN IN MY YAHOO! YAHOO MAIL GAMES PERSONALS YAHOO! NEWS FINANCE SHOPPING
ORACLE
ADD QUERY
SHOW TABLES
DELETE TABLES
BACK
SHOW TABLE
HEALTHCARE
STORED RECORD
5506
5504
5502
5508
SELECTED FIELDS PARSED
ENCRYPTED SCATTERED IN DATABASE(S)
OR ENTERPRISE
SECURED DATA
SESSION_MASTER_KEY
NOTE - NO DATA IN THESE FIELDS
NOTE DIFFERENT SIZES OF SECURED FIELDS
DONE
LOCAL INTRANET

FIG. 55

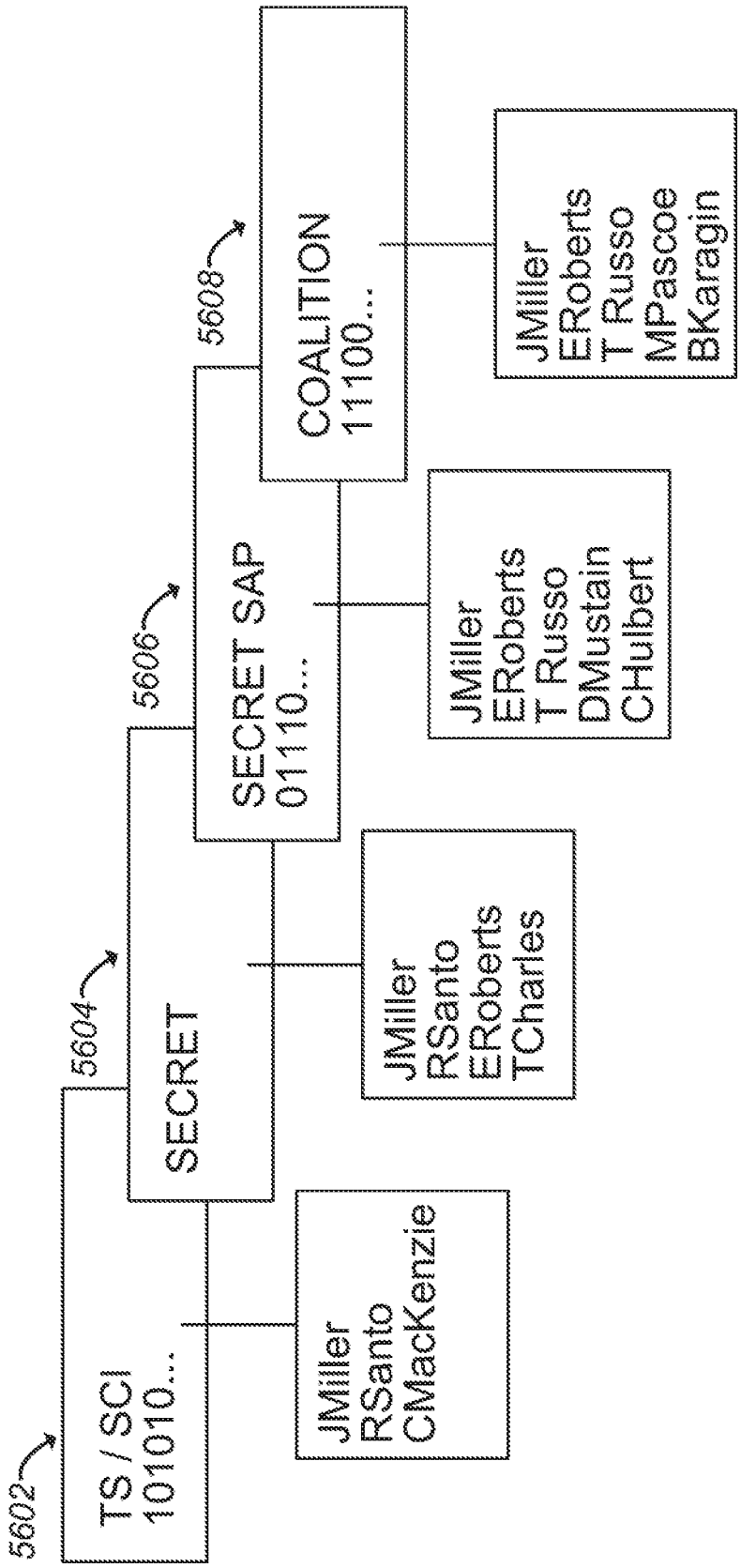

5602
TS / SCI
101010...
5604
SECRET
5606
SECRET SAP
01110...
5608
COALITION
11100...
JMiller
RSanto
CMacKenzie
JMiller
RSanto
ERoberts
TCharles
JMiller
ERoberts
T Russo
DMustain
CHulbert
JMiller
ERoberts
T Russo
MPascoe
BKaragin

FIG. 56

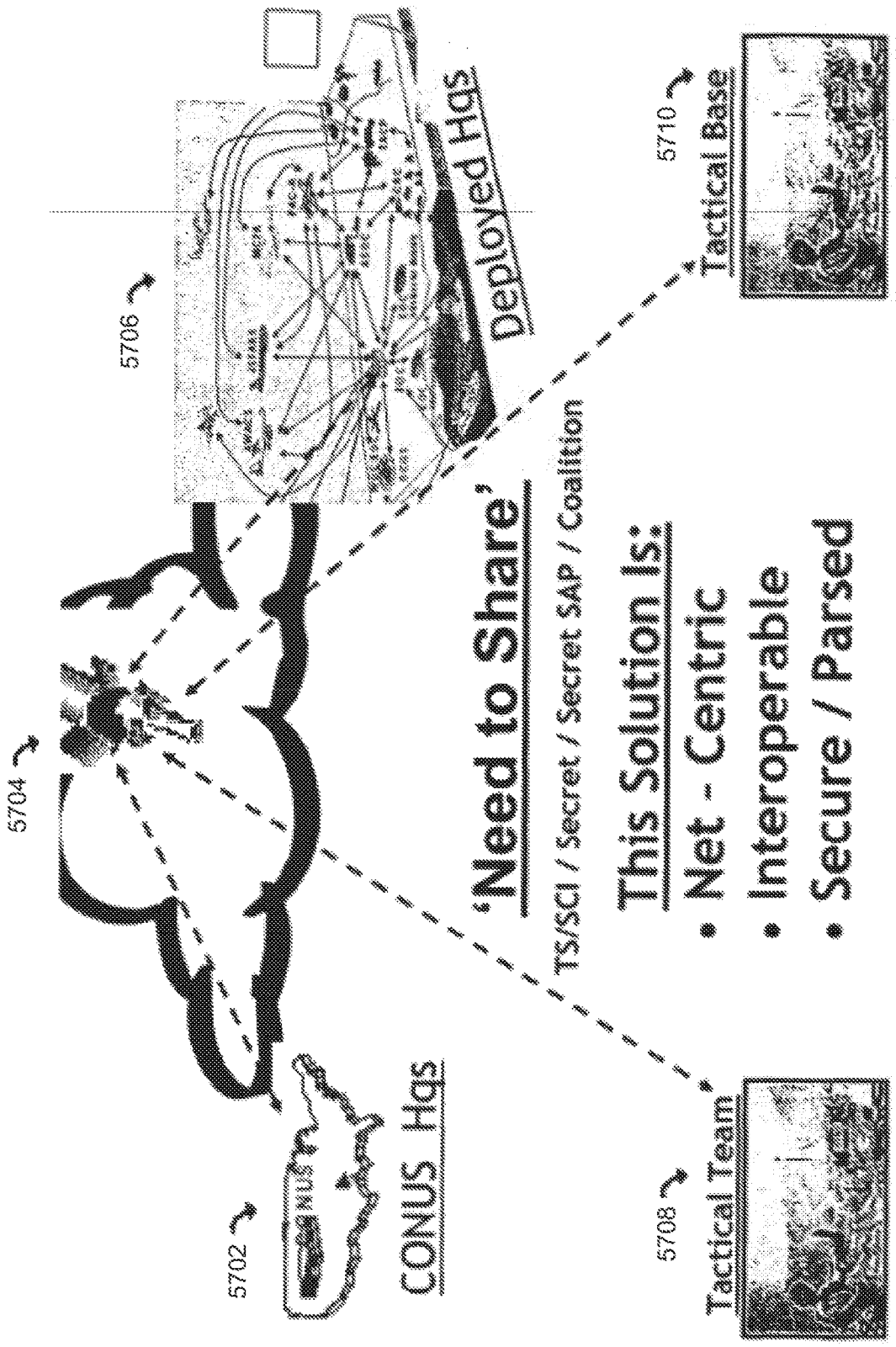

5704
5706
5710
Tactical Base
Deployed Hqs
'Need to Share'
TS/SCI / Secret / Secret SAP / Coalition
This Solution Is:
• Net - Centric
• Interoperable
• Secure / Parsed
5702
CONUS Hqs
5708
Tactical Team

FIG. 57

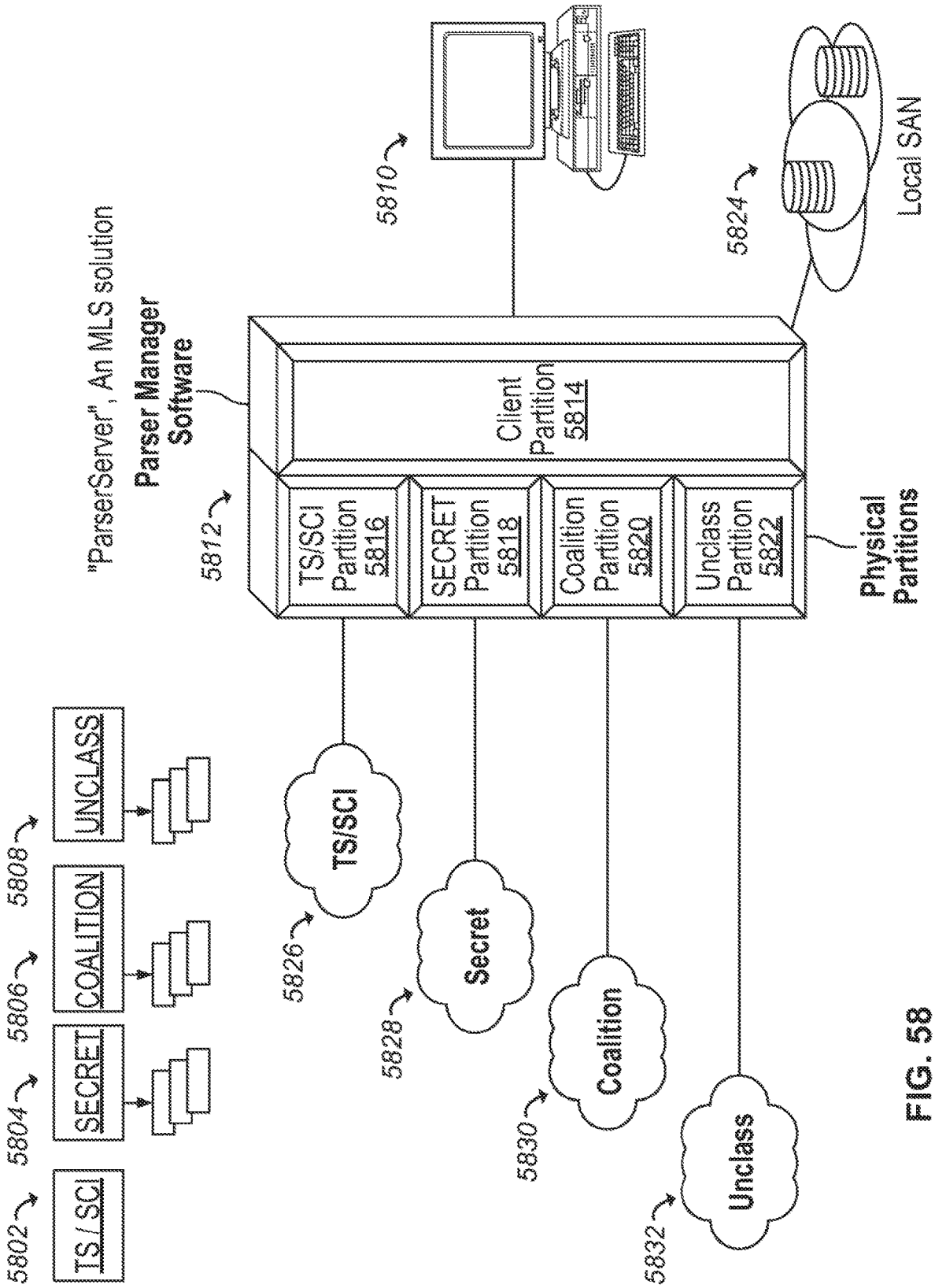

"ParserServer", An MLS solution
Parser Manager Software
5812
Client Partition 5814
TS/SCI Partition 5816
SECRET Partition 5818
Coalition Partition 5820
Unclass Partition 5822
Physical Partitions
5810
5824
Local SAN
TS/SCI
Secret
Coalition
Unclass
5826
5828
5830
5832
TS / SCI
SECRET
COALITION
UNCLASS
5802
5804
5806
5808

FIG. 58

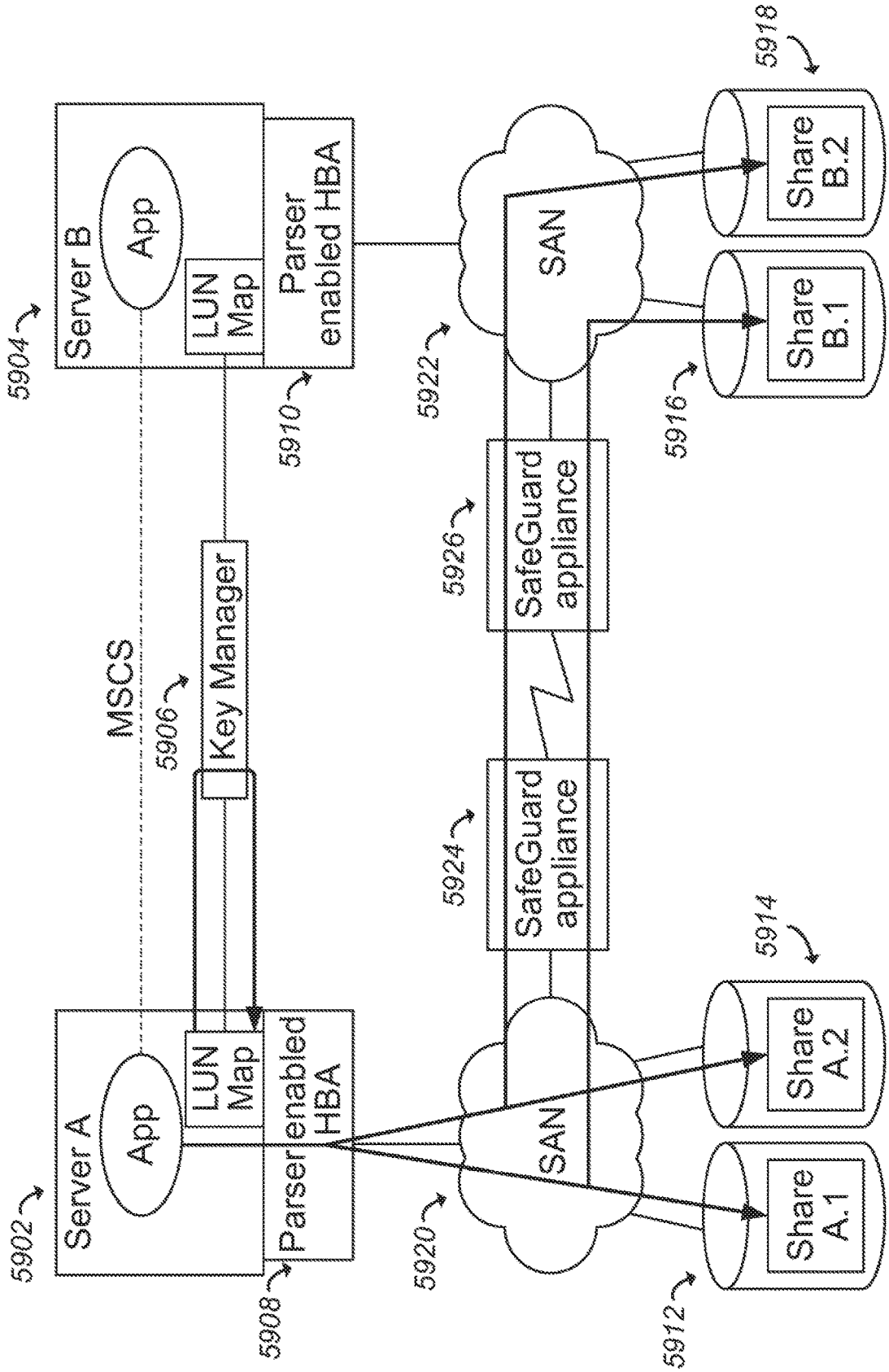

Server B
App
LUN Map
Parser enabled HBA
5904
5910
MSCS
Key Manager
5906
Server A
App
LUN Map
Parser enabled HBA
5902
5908
SAN
5922
SAN
5920
SafeGuard appliance
5926
SafeGuard appliance
5924
Share B.2
Share B.1
5918
5916
Share A.2
Share A.1
5914
5912

FIG. 59

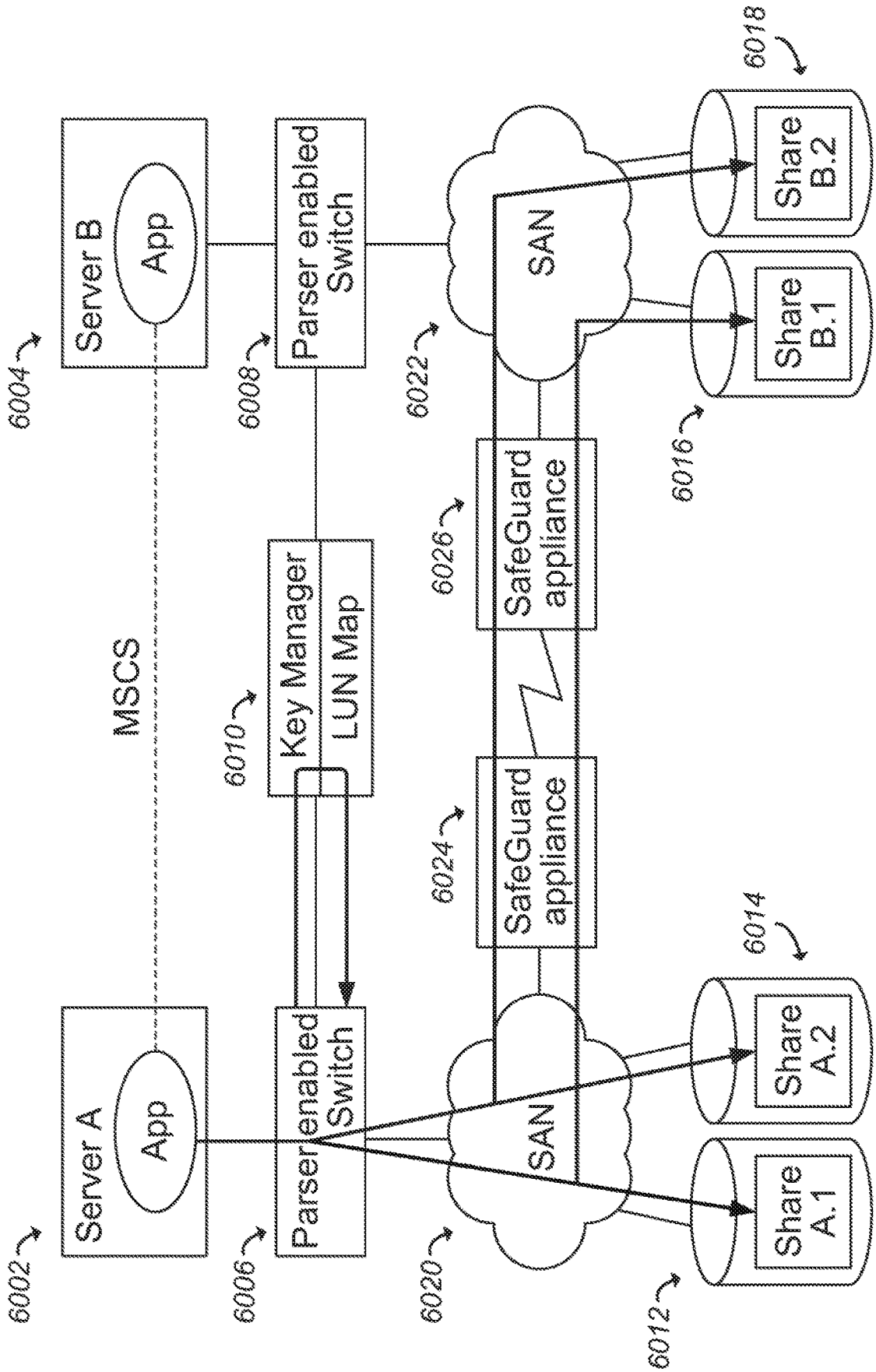

Server B
App
Parser enabled Switch
SAN
Share B.2
Share B.1
6004
6008
6022
6018
6016
MSCS
Key Manager
LUN Map
SafeGuard appliance
6010
6026
Server A
App
Parser enabled Switch
SAN
Share A.2
Share A.1
SafeGuard appliance
6024
6014
6002
6006
6020
6012

FIG. 60

SECURE DATA PARSER METHOD AND SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation of U.S. patent application Ser. No. 13/915,081, filed Jun. 11, 2013, which is a divisional of U.S. patent application Ser. No. 13/468,383, filed on May 10, 2012, now U.S. Pat. No. 9,338,140, which is a continuation of U.S. patent application Ser. No. 11/258,839, filed on Oct. 25, 2005, now U.S. Pat. No. 8,266,438, which claims priority benefit from U.S. provisional application No. 60/622,146, filed on Oct. 25, 2004, and U.S. provisional application No. 60/718,185, filed Sep. 16, 2005. The aforementioned, earlier-filed applications are hereby incorporated by reference herein in their entireties.

FIELD OF THE INVENTION

The present invention relates in general to a system for securing data from unauthorized access or use.

BACKGROUND OF THE INVENTION

In today's society, individuals and businesses conduct an ever-increasing amount of activities on and over computer systems. These computer systems, including proprietary and non-proprietary computer networks, are often storing, archiving, and transmitting all types of sensitive information. Thus, an ever-increasing need exists for ensuring data stored and transmitted over these systems cannot be read or otherwise compromised.

One common solution for securing computer systems is to provide login and password functionality. However, password management has proven to be quite costly with a large percentage of help desk calls relating to password issues. Moreover, passwords provide little security in that they are generally stored in a file susceptible to inappropriate access, through, for example, brute-force attacks.

Another solution for securing computer systems is to provide cryptographic infrastructures. Cryptography, in general, refers to protecting data by transforming, or encrypting, it into an unreadable format. Only those who possess the key(s) to the encryption can decrypt the data into a usable format. Cryptography is used to identify users, e.g., authentication, to allow access privileges, e.g., authorization, to create digital certificates and signatures, and the like. One popular cryptography system is a public key system that uses two keys, a public key known to everyone and a private key known only to the individual or business owner thereof. Generally, the data encrypted with one key is decrypted with the other and neither key is recreatable from the other.

Unfortunately, even the foregoing typical public-key cryptographic systems are still highly reliant on the user for security. For example, cryptographic systems issue the private key to the user, for example, through the user's browser. Unsophisticated users then generally store the private key on a hard drive accessible to others through an open computer system, such as, for example, the Internet. On the other hand, users may choose poor names for files containing their private key, such as, for example, "key." The result of the foregoing and other acts is to allow the key or keys to be susceptible to compromise.

In addition to the foregoing compromises, a user may save his or her private key on a computer system configured with an archiving or backup system, potentially resulting in

copies of the private key traveling through multiple computer storage devices or other systems. This security breach is often referred to as "key migration." Similar to key migration, many applications provide access to a user's private key through, at most, simple login and password access. As mentioned in the foregoing, login and password access often does not provide adequate security.

One solution for increasing the security of the foregoing cryptographic systems is to include biometrics as part of the authentication or authorization. Biometrics generally include measurable physical characteristics, such as, for example, finger prints or speech that can be checked by an automated system, such as, for example, pattern matching or recognition of finger print patterns or speech patterns. In such systems, a user's biometric and/or keys may be stored on mobile computing devices, such as, for example, a smartcard, laptop, personal digital assistant, or mobile phone, thereby allowing the biometric or keys to be usable in a mobile environment.

The foregoing mobile biometric cryptographic system still suffers from a variety of drawbacks. For example, the mobile user may lose or break the smartcard or portable computing device, thereby having his or her access to potentially important data entirely cut-off. Alternatively, a malicious person may steal the mobile user's smartcard or portable computing device and use it to effectively steal the mobile user's digital credentials. On the other hand, the portable-computing device may be connected to an open system, such as the Internet, and, like passwords, the file where the biometric is stored may be susceptible to compromise through user inattentiveness to security or malicious intruders.

SUMMARY OF THE INVENTION

Based on the foregoing, a need exists to provide a cryptographic system whose security is user-independent while still supporting mobile users.

Accordingly, one aspect of the present invention is to provide a method for securing virtually any type of data from unauthorized access or use. The method comprises one or more steps of parsing, splitting and/or separating the data to be secured into two or more parts or portions. The method also comprises encrypting the data to be secured. Encryption of the data may be performed prior to or after the first parsing, splitting and/or separating of the data. In addition, the encrypting step may be repeated for one or more portions of the data. Similarly, the parsing, splitting and/or separating steps may be repeated for one or more portions of the data. The method also optionally comprises storing the parsed, split and/or separated data that has been encrypted in one location or in multiple locations. This method also optionally comprises reconstituting or re-assembling the secured data into its original form for authorized access or use. This method may be incorporated into the operations of any computer, server, engine or the like, that is capable of executing the desired steps of the method.

Another aspect of the present invention provides a system for securing virtually any type of data from unauthorized access or use. This system comprises a data splitting module, a cryptographic handling module, and, optionally, a data assembly module. The system may, in one embodiment, further comprise one or more data storage facilities where secure data may be stored.

Accordingly, one aspect of the invention is to provide a secure server, or trust engine, having server-centric keys, or in other words, storing cryptographic keys and user authen-

tication data on a server. According to this embodiment, a user accesses the trust engine in order to perform authentication and cryptographic functions, such as, but not limited to, for example, authentication, authorization, digital signing and generation, storage, and retrieval of certificates, encryption, notary-like and power-of-attorney-like actions, and the like.

Another aspect of the invention is to provide a reliable, or trusted, authentication process. Moreover, subsequent to a trustworthy positive authentication, a wide number of differing actions may be taken, from providing cryptographic technology, to system or device authorization and access, to permitting use or control of one or a wide number of electronic devices.

Another aspect of the invention is to provide cryptographic keys and authentication data in an environment where they are not lost, stolen, or compromised, thereby advantageously avoiding a need to continually reissue and manage new keys and authentication data. According to another aspect of the invention, the trust engine allows a user to use one key pair for multiple activities, vendors, and/or authentication requests. According to yet another aspect of the invention, the trust engine performs at least one step of cryptographic processing, such as, but not limited to, encrypting, authenticating, or signing, on the server side, thereby allowing clients or users to possess only minimal computing resources.

According to yet another aspect of the invention, the trust engine includes one or multiple depositories for storing portions of each cryptographic key and authentication data. The portions are created through a data splitting process that prohibits reconstruction without a predetermined portion from more than one location in one depository or from multiple depositories. According to another embodiment, the multiple depositories may be geographically remote such that a rogue employee or otherwise compromised system at one depository will not provide access to a user's key or authentication data.

According to yet another embodiment, the authentication process advantageously allows the trust engine to process multiple authentication activities in parallel. According to yet another embodiment, the trust engine may advantageously track failed access attempts and thereby limit the number of times malicious intruders may attempt to subvert the system.

According to yet another embodiment, the trust engine may include multiple instantiations where each trust engine may predict and share processing loads with the others. According to yet another embodiment, the trust engine may include a redundancy module for polling a plurality of authentication results to ensure that more than one system authenticates the user.

Therefore, one aspect of the invention includes a secure cryptographic system, which may be remotely accessible, for storing data of any type, including, but not limited to, a plurality of private cryptographic keys to be associated with a plurality of users. The cryptographic system associates each of the plurality of users with one or more different keys from the plurality of private cryptographic keys and performs cryptographic functions for each user using the associated one or more different keys without releasing the plurality of private cryptographic keys to the users. The cryptographic system comprises a depository system having at least one server which stores the data to be secured, such as a plurality of private cryptographic keys and a plurality of enrollment authentication data. Each enrollment authentication data identifies one of multiple users and each of the multiple users is associated with one or more different keys from the plurality of private cryptographic keys. The cryptographic system also may comprise an authentication engine which compares authentication data received by one of the multiple users to enrollment authentication data corresponding to the one of multiple users and received from the depository system, thereby producing an authentication result. The cryptographic system also may comprise a cryptographic engine which, when the authentication result indicates proper identification of the one of the multiple users, performs cryptographic functions on behalf of the one of the multiple users using the associated one or more different keys received from the depository system. The cryptographic system also may comprise a transaction engine connected to route data from the multiple users to the depository server system, the authentication engine, and the cryptographic engine.

Another aspect of the invention includes a secure cryptographic system that is optionally remotely accessible. The cryptographic system comprises a depository system having at least one server which stores at least one private key and any other data, such as, but not limited to, a plurality of enrollment authentication data, wherein each enrollment authentication data identifies one of possibly multiple users. The cryptographic system may also optionally comprise an authentication engine which compares authentication data received by users to enrollment authentication data corresponding to the user and received from the depository system, thereby producing an authentication result. The cryptographic system also comprises a cryptographic engine which, when the authentication result indicates proper identification of the user, performs cryptographic functions on behalf of the user using at least said private key, which may be received from the depository system. The cryptographic system may also optionally comprise a transaction engine connected to route data from the users to other engines or systems such as, but not limited to, the depository server system, the authentication engine, and the cryptographic engine.

Another aspect of the invention includes a method of facilitating cryptographic functions. The method comprises associating a user from multiple users with one or more keys from a plurality of private cryptographic keys stored on a secure location, such as a secure server. The method also comprises receiving authentication data from the user, and comparing the authentication data to authentication data corresponding to the user, thereby verifying the identity of the user. The method also comprises utilizing the one or more keys to perform cryptographic functions without releasing the one or more keys to the user.

Another aspect of the invention includes an authentication system for uniquely identifying a user through secure storage of the user's enrollment authentication data. The authentication system comprises one or more data storage facilities, wherein each data storage facility includes a computer accessible storage medium which stores at least one of portions of enrollment authentication data.

The authentication system also comprises an authentication engine which communicates with the data storage facility or facilities. The authentication engine comprises a data splitting module which operates on the enrollment authentication data to create portions, a data assembling module which processes the portions from at least one of the data storage facilities to assemble the enrollment authentication data, and a data comparator module which receives current authentication data from a user and compares the

current authentication data with the assembled enrollment authentication data to determine whether the user has been uniquely identified.

Another aspect of the invention includes a cryptographic system. The cryptographic system comprises one or more data storage facilities, wherein each data storage facility includes a computer accessible storage medium which stores at least one portion of one ore more cryptographic keys. The cryptographic system also comprises a cryptographic engine which communicates with the data storage facilities. The cryptographic engine also comprises a data splitting module which operate on the cryptographic keys to create portions, a data assembling module which processes the portions from at least one of the data storage facilities to assemble the cryptographic keys, and a cryptographic handling module which receives the assembled cryptographic keys and performs cryptographic functions therewith.

Another aspect of the invention includes a method of storing any type of data, including, but not limited to, authentication data in geographically remote secure data storage facilities thereby protecting the data against composition of any individual data storage facility. The method comprises receiving data at a trust engine, combining at the trust engine the data with a first substantially random value to form a first combined value, and combining the data with a second substantially random value to form a second combined value. The method comprises creating a first pairing of the first substantially random value with the second combined value, creating a second pairing of the first substantially random value with the second substantially random value, and storing the first pairing in a first secure data storage facility. The method comprises storing the second pairing in a second secure data storage facility remote from the first secure data storage facility.

Another aspect of the invention includes a method of storing any type of data, including, but not limited to, authentication data comprising receiving data, combining the data with a first set of bits to form a second set of bits, and combining the data with a third set of bits to form a fourth set of bits. The method also comprises creating a first pairing of the first set of bits with the third set of bits. The method also comprises creating a second pairing of the first set of bits with the fourth set of bits, and storing one of the first and second pairings in a first computer accessible storage medium. The method also comprises storing the other of the first and second pairings in a second computer accessible storage medium.

Another aspect of the invention includes a method of storing cryptographic data in geographically remote secure data storage facilities thereby protecting the cryptographic data against comprise of any individual data storage facility. The method comprises receiving cryptographic data at a trust engine, combining at the trust engine the cryptographic data with a first substantially random value to form a first combined value, and combining the cryptographic data with a second substantially random value to form a second combined value. The method also comprises creating a first pairing of the first substantially random value with the second combined value, creating a second pairing of the first substantially random value with the second substantially random value, and storing the first pairing in a first secure data storage facility. The method also comprises storing the second pairing in a secure second data storage facility remote from the first secure data storage facility.

Another aspect of the invention includes a method of storing cryptographic data comprising receiving authentication data and combining the cryptographic data with a first set of bits to form a second set of bits. The method also comprises combining the cryptographic data with a third set of bits to form a fourth set of bits, creating a first pairing of the first set of bits with the third set of bits, and creating a second pairing of the first set of bits with the fourth set of bits. The method also comprises storing one of the first and second pairings in a first computer accessible storage medium, and storing the other of the first and second pairings in a second computer accessible storage medium.

Another aspect of the invention includes a method of handling sensitive data of any type or form in a cryptographic system, wherein the sensitive data exists in a usable form only during actions by authorized users, employing the sensitive data. The method also comprises receiving in a software module, substantially randomized or encrypted sensitive data from a first computer accessible storage medium, and receiving in the software module, substantially randomized or encrypted data which may or may not be sensitive data, from one or more other computer accessible storage medium. The method also comprises processing the substantially randomized pre-encrypted sensitive data and the substantially randomized or encrypted data which may or may not be sensitive data, in the software module to assemble the sensitive data and employing the sensitive data in a software engine to perform an action. The action includes, but is not limited to, one of authenticating a user and performing a cryptographic function.

Another aspect of the invention includes a secure authentication system. The secure authentication system comprises a plurality of authentication engines. Each authentication engine receives enrollment authentication data designed to uniquely identify a user to a degree of certainty. Each authentication engine receives current authentication data to compare to the enrollment authentication data, and each authentication engine determines an authentication result. The secure authentication system also comprises a redundancy system which receives the authentication result of at least two of the authentication engines and determines whether the user has been uniquely identified.

Another aspect of the invention includes a secure data in motion system whereby data may be transmitted in different portions that are secured in accordance with the present invention such that any one portion becoming compromised shall not provide sufficient data to restore the original data. This may be applied to any transmission of data, whether it be wired, wireless, or physical.

Another aspect of the invention includes integration of the secure data parser of the present invention into any suitable system where data is stored or communicated. For example, email system, RAID systems, video broadcasting systems, database systems, or any other suitable system may have the secure data parser integrated at any suitable level.

Another aspect of the invention includes using any suitable parsing and splitting algorithm to generate shares of data. Either random, pseudo-random, deterministic, or any combination thereof may be employed for parsing and splitting data.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is described in more detail below in connection with the attached drawings, which are meant to illustrate and not to limit the invention, and in which:

FIG. **1** illustrates a block diagram of a cryptographic system, according to aspects of an embodiment of the invention;

FIG. **2** illustrates a block diagram of the trust engine of FIG. **1**, according to aspects of an embodiment of the invention;

FIG. **3** illustrates a block diagram of the transaction engine of FIG. **2**, according to aspects of an embodiment of the invention;

FIG. **4** illustrates a block diagram of the depository of FIG. **2**, according to aspects of an embodiment of the invention;

FIG. **5** illustrates a block diagram of the authentication engine of FIG. **2**, according to aspects of an embodiment of the invention;

FIG. **6** illustrates a block diagram of the cryptographic engine of FIG. **2**, according to aspects of an embodiment of the invention;

FIG. **7** illustrates a block diagram of a depository system, according to aspects of another embodiment of the invention;

FIG. **8** illustrates a flow chart of a data splitting process according to aspects of an embodiment of the invention;

FIG. **9**A illustrates a data flow of an enrollment process according to aspects of an embodiment of the invention;

FIG. **9**B illustrates a flow chart of an interoperability process according to aspects of an embodiment of the invention;

FIG. **10** illustrates a data flow of an authentication process according to aspects of an embodiment of the invention;

FIG. **11** illustrates a data flow of a signing process according to aspects of an embodiment of the invention;

FIG. **12** illustrates a data flow and an encryption/decryption process according to aspects and yet another embodiment of the invention;

FIG. **13** illustrates a simplified block diagram of a trust engine system according to aspects of another embodiment of the invention;

FIG. **14** illustrates a simplified block diagram of a trust engine system according to aspects of another embodiment of the invention;

FIG. **15** illustrates a block diagram of the redundancy module of FIG. **14**, according to aspects of an embodiment of the invention;

FIG. **16** illustrates a process for evaluating authentications according to one aspect of the invention;

FIG. **17** illustrates a process for assigning a value to an authentication according to one aspect as shown in FIG. **16** of the invention;

FIG. **18** illustrates a process for performing trust arbitrage in an aspect of the invention as shown in FIG. **17**; and

FIG. **19** illustrates a sample transaction between a user and a vendor according to aspects of an embodiment of the invention where an initial web based contact leads to a sales contract signed by both parties.

FIG. **20** illustrates a sample user system with a cryptographic service provider module which provides security functions to a user system.

FIG. **21** illustrates a process for parsing, splitting and/or separating data with encryption and storage of the encryption master key with the data.

FIG. **22** illustrates a process for parsing, splitting and/or separating data with encryption and storing the encryption master key separately from the data.

FIG. **23** illustrates the intermediary key process for parsing, splitting and/or separating data with encryption and storage of the encryption master key with the data.

FIG. **24** illustrates the intermediary key process for parsing, splitting and/or separating data with encryption and storing the encryption master key separately from the data.

FIG. **25** illustrates utilization of the cryptographic methods and systems of the present invention with a small working group.

FIG. **26** is a block diagram of an illustrative physical token security system employing the secure data parser in accordance with one embodiment of the present invention.

FIG. **27** is a block diagram of an illustrative arrangement in which the secure data parser is integrated into a system in accordance with one embodiment of the present invention.

FIG. **28** is a block diagram of an illustrative data in motion system in accordance with one embodiment of the present invention.

FIG. **29** is a block diagram of another illustrative data in motion system in accordance with one embodiment of the present invention.

FIG. **30-32** are block diagrams of an illustrative system having the secure data parser integrated in accordance with one embodiment of the present invention.

FIG. **33** is a process flow diagram of an illustrative process for parsing and splitting data in accordance with one embodiment of the present invention.

FIG. **34** is a process flow diagram of an illustrative process for restoring portions of data into original data in accordance with one embodiment of the present invention.

FIG. **35** is a process flow diagram of an illustrative process for splitting data at the bit level in accordance with one embodiment of the present invention.

FIG. **36** depicts an exemplary bit-splitting technique in accordance with one embodiment of the present invention.

FIG. **37** depicts a data flow for distributing the output of a bit-splitting process into one or more shares in accordance with one embodiment of the present invention.

FIG. **38** depicts an exemplary bit scatter technique for the distribution of data into shares in accordance with one embodiment of the present invention.

FIG. **39** depicts an exemplary technique for restoring data that was distributed to a share using the bit scatter technique.

FIG. **40** illustrates example of such an allocation for a 2 of 4 key split with a ratio of 75% in accordance with one embodiment of the present invention.

FIG. **41** is an illustrative overview process for using the secure data parser of the present invention.

FIG. **42** depicts an exemplary secure parser system in accordance with one embodiment of the present invention.

FIG. **43** depicts an exemplary integration of a secure parser into a system utilizing an Application Programming Interface (API).

FIG. **44** depicts an example configuration of a share fault tolerance system in accordance with one embodiment of the present invention.

FIG. **45** depicts an example configuration of a share fault tolerance system with a mandatory share in accordance with one embodiment of the present invention.

FIG. **46** depicts an illustrative process for parsing and distributing data into shares in accordance with one embodiment of the present invention.

FIG. **47** depicts an illustrative process for recovering data from shares in accordance with one embodiment of the present invention.

FIGS. **48**A and **48**B depict various parameters used in parsing operations in accordance with one embodiment of the present invention.

FIGS. **49**A and **49**B depict data structures for the parser, keys, and shares in accordance with one embodiment of the present invention.

FIGS. **50**A and **50**B depict data structures relating to encryption, authentication, and hashing functionality in accordance with one embodiment of the present invention.

FIGS. **51**A and **51**B depict data formats in accordance with one embodiment of the present invention.

FIGS. **52**A, **52**B, **52**C, and **52**D depict a function library in accordance with one embodiment of the present invention.

FIG. **53** depicts an exemplary implementation of a secure parser within a hardware system in accordance with one embodiment of the present invention.

FIG. **54** depicts an exemplary implementation of a secure parser for data masking in accordance with one embodiment of the present invention.

FIG. **55** depicts an exemplary data record in a data masking implementation in accordance with one embodiment of the present invention.

FIG. **56** depicts an exemplary set of workgroup keys in accordance with one embodiment of the present invention.

FIG. **57** depicts the use of workgroup keys in an example application of military use in accordance with one embodiment of the present invention.

FIG. **58** depicts the use of workgroup keys in a multi-level security (MLS) solution in accordance with one embodiment of the present invention.

FIG. **59** depicts a backup application in accordance with one embodiment of the present invention.

FIG. **60** depicts another backup application in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

One aspect of the present invention is to provide a cryptographic system where one or more secure servers, or a trust engine, stores cryptographic keys and user authentication data. Users access the functionality of conventional cryptographic systems through network access to the trust engine, however, the trust engine does not release actual keys and other authentication data and therefore, the keys and data remain secure. This server-centric storage of keys and authentication data provides for user-independent security, portability, availability, and straightforwardness.

Because users can be confident in, or trust, the cryptographic system to perform user and document authentication and other cryptographic functions, a wide variety of functionality may be incorporated into the system. For example, the trust engine provider can ensure against agreement repudiation by, for example, authenticating the agreement participants, digitally signing the agreement on behalf of or for the participants, and storing a record of the agreement digitally signed by each participant. In addition, the cryptographic system may monitor agreements and determine to apply varying degrees of authentication, based on, for example, price, user, vendor, geographic location, place of use, or the like.

To facilitate a complete understanding of the invention, the remainder of the detailed description describes the invention with reference to the figures, wherein like elements are referenced with like numerals throughout.

FIG. **1** illustrates a block diagram of a cryptographic system **100**, according to aspects of an embodiment of the invention. As shown in FIG. **1**, the cryptographic system **100** includes a user system **105**, a trust engine **110**, a certificate authority **115**, and a vendor system **120**, communicating through a communication link **125**.

According to one embodiment of the invention, the user system **105** comprises a conventional general-purpose computer having one or more microprocessors, such as, for example, an Intel-based processor. Moreover, the user system **105** includes an appropriate operating system, such as, for example, an operating system capable of including graphics or windows, such as Windows, Unix, Linux, or the like. As shown in FIG. **1**, the user system **105** may include a biometric device **107**. The biometric device **107** may advantageously capture a user's biometric and transfer the captured biometric to the trust engine **110**.

According to one embodiment of the invention, the biometric device may advantageously comprise a device having attributes and features similar to those disclosed in U.S. patent application Ser. No. 08/926,277, filed on Sep. 5, 1997, entitled "RELIEF OBJECT IMAGE GENERATOR," U.S. patent application Ser. No. 09/558,634, filed on Apr. 26, 2000, entitled "IMAGING DEVICE FOR A RELIEF OBJECT AND SYSTEM AND METHOD OF USING THE IMAGE DEVICE," U.S. patent application Ser. No. 09/435,011, filed on Nov. 5, 1999, entitled "RELIEF OBJECT SENSOR ADAPTOR," and U.S. patent application Ser. No. 09/477,943, filed on Jan. 5, 2000, entitled "PLANAR OPTICAL IMAGE SENSOR AND SYSTEM FOR GENERATING AN ELECTRONIC IMAGE OF A RELIEF OBJECT FOR FINGERPRINT READING," all of which are owned by the instant assignee, and all of which are hereby incorporated by reference herein.

In addition, the user system **105** may connect to the communication link **125** through a conventional service provider, such as, for example, a dial up, digital subscriber line (DSL), cable modem, fiber connection, or the like. According to another embodiment, the user system **105** connects the communication link **125** through network connectivity such as, for example, a local or wide area network. According to one embodiment, the operating system includes a TCP/IP stack that handles all incoming and outgoing message traffic passed over the communication link **125**.

Although the user system **105** is disclosed with reference to the foregoing embodiments, the invention is not intended to be limited thereby. Rather, a skilled artisan will recognize from the disclosure herein, a wide number of alternatives embodiments of the user system **105**, including almost any computing device capable of sending or receiving information from another computer system. For example, the user system **105** may include, but is not limited to, a computer workstation, an interactive television, an interactive kiosk, a personal mobile computing device, such as a digital assistant, mobile phone, laptop, or the like, a wireless communications device, a smartcard, an embedded computing device, or the like, which can interact with the communication link **125**. In such alternative systems, the operating systems will likely differ and be adapted for the particular device. However, according to one embodiment, the operating systems advantageously continue to provide the appropriate communications protocols needed to establish communication with the communication link **125**.

FIG. **1** illustrates the trust engine **110**. According to one embodiment, the trust engine **110** comprises one or more secure servers for accessing and storing sensitive information, which may be any type or form of data, such as, but not limited to text, audio, video, user authentication data and public and private cryptographic keys. According to one embodiment, the authentication data includes data designed to uniquely identify a user of the cryptographic system **100**. For example, the authentication data may include a user

identification number, one or more biometrics, and a series of questions and answers generated by the trust engine **110** or the user, but answered initially by the user at enrollment. The foregoing questions may include demographic data, such as place of birth, address, anniversary, or the like, personal data, such as mother's maiden name, favorite ice cream, or the like, or other data designed to uniquely identify the user. The trust engine **110** compares a user's authentication data associated with a current transaction, to the authentication data provided at an earlier time, such as, for example, during enrollment. The trust engine **110** may advantageously require the user to produce the authentication data at the time of each transaction, or, the trust engine **110** may advantageously allow the user to periodically produce authentication data, such as at the beginning of a string of transactions or the logging onto a particular vendor website.

According to the embodiment where the user produces biometric data, the user provides a physical characteristic, such as, but not limited to, facial scan, hand scan, ear scan, iris scan, retinal scan, vascular pattern, DNA, a fingerprint, writing or speech, to the biometric device **107**. The biometric device advantageously produces an electronic pattern, or biometric, of the physical characteristic. The electronic pattern is transferred through the user system **105** to the trust engine **110** for either enrollment or authentication purposes.

Once the user produces the appropriate authentication data and the trust engine **110** determines a positive match between that authentication data (current authentication data) and the authentication data provided at the time of enrollment (enrollment authentication data), the trust engine **110** provides the user with complete cryptographic functionality. For example, the properly authenticated user may advantageously employ the trust engine **110** to perform hashing, digitally signing, encrypting and decrypting (often together referred to only as encrypting), creating or distributing digital certificates, and the like. However, the private cryptographic keys used in the cryptographic functions will not be available outside the trust engine **110**, thereby ensuring the integrity of the cryptographic keys.

According to one embodiment, the trust engine **110** generates and stores cryptographic keys. According to another embodiment, at least one cryptographic key is associated with each user. Moreover, when the cryptographic keys include public-key technology, each private key associated with a user is generated within, and not released from, the trust engine **110**. Thus, so long as the user has access to the trust engine **110**, the user may perform cryptographic functions using his or her private or public key. Such remote access advantageously allows users to remain completely mobile and access cryptographic functionality through practically any Internet connection, such as cellular and satellite phones, kiosks, laptops, hotel rooms and the like.

According to another embodiment, the trust engine **110** performs the cryptographic functionality using a key pair generated for the trust engine **110**. According to this embodiment, the trust engine **110** first authenticates the user, and after the user has properly produced authentication data matching the enrollment authentication data, the trust engine **110** uses its own cryptographic key pair to perform cryptographic functions on behalf of the authenticated user.

A skilled artisan will recognize from the disclosure herein that the cryptographic keys may advantageously include some or all of symmetric keys, public keys, and private keys. In addition, a skilled artisan will recognize from the disclosure herein that the foregoing keys may be implemented with a wide number of algorithms available from commercial technologies, such as, for example, RSA, ELGAMAL, or the like.

FIG. **1** also illustrates the certificate authority **115**. According to one embodiment, the certificate authority **115** may advantageously comprise a trusted third-party organization or company that issues digital certificates, such as, for example, VeriSign, Baltimore, Entrust, or the like. The trust engine **110** may advantageously transmit requests for digital certificates, through one or more conventional digital certificate protocols, such as, for example, PKCS10, to the certificate authority **115**. In response, the certificate authority **115** will issue a digital certificate in one or more of a number of differing protocols, such as, for example, PKCS7. According to one embodiment of the invention, the trust engine **110** requests digital certificates from several or all of the prominent certificate authorities **115** such that the trust engine **110** has access to a digital certificate corresponding to the certificate standard of any requesting party.

According to another embodiment, the trust engine **110** internally performs certificate issuances. In this embodiment, the trust engine **110** may access a certificate system for generating certificates and/or may internally generate certificates when they are requested, such as, for example, at the time of key generation or in the certificate standard requested at the time of the request. The trust engine **110** will be disclosed in greater detail below.

FIG. **1** also illustrates the vendor system **120**. According to one embodiment, the vendor system **120** advantageously comprises a Web server. Typical Web servers generally serve content over the Internet using one of several internet markup languages or document format standards, such as the Hyper-Text Markup Language (HTML) or the Extensible Markup Language (XML). The Web server accepts requests from browsers like Netscape and Internet Explorer and then returns the appropriate electronic documents. A number of server or client-side technologies can be used to increase the power of the Web server beyond its ability to deliver standard electronic documents. For example, these technologies include Common Gateway Interface (CGI) scripts, Secure Sockets Layer (SSL) security, and Active Server Pages (ASPs). The vendor system **120** may advantageously provide electronic content relating to commercial, personal, educational, or other transactions.

Although the vendor system **120** is disclosed with reference to the foregoing embodiments, the invention is not intended to be limited thereby. Rather, a skilled artisan will recognize from the disclosure herein that the vendor system **120** may advantageously comprise any of the devices described with reference to the user system **105** or combination thereof.

FIG. **1** also illustrates the communication link **125** connecting the user system **105**, the trust engine **110**, the certificate authority **115**, and the vendor system **120**. According to one embodiment, the communication link **125** preferably comprises the Internet. The Internet, as used throughout this disclosure is a global network of computers. The structure of the Internet, which is well known to those of ordinary skill in the art, includes a network backbone with networks branching from the backbone. These branches, in turn, have networks branching from them, and so on. Routers move information packets between network levels, and then from network to network, until the packet reaches the neighborhood of its destination. From the destination, the destination network's host directs the information packet to the appropriate terminal, or node. In one advantageous embodiment, the Internet routing hubs comprise domain

name system (DNS) servers using Transmission Control Protocol/Internet Protocol (TCP/IP) as is well known in the art. The routing hubs connect to one or more other routing hubs via high-speed communication links.

One popular part of the Internet is the World Wide Web. The World Wide Web contains different computers, which store documents capable of displaying graphical and textual information. The computers that provide information on the World Wide Web are typically called "websites." A website is defined by an Internet address that has an associated electronic page. The electronic page can be identified by a Uniform Resource Locator (URL). Generally, an electronic page is a document that organizes the presentation of text, graphical images, audio, video, and so forth.

Although the communication link **125** is disclosed in terms of its preferred embodiment, one of ordinary skill in the art will recognize from the disclosure herein that the communication link **125** may include a wide range of interactive communications links. For example, the communication link **125** may include interactive television networks, telephone networks, wireless data transmission systems, two-way cable systems, customized private or public computer networks, interactive kiosk networks, automatic teller machine networks, direct links, satellite or cellular networks, and the like.

FIG. **2** illustrates a block diagram of the trust engine **110** of FIG. **1** according to aspects of an embodiment of the invention. As shown in FIG. **2**, the trust engine **110** includes a transaction engine **205**, a depository **210**, an authentication engine **215**, and a cryptographic engine **220**. According to one embodiment of the invention, the trust engine **110** also includes mass storage **225**. As further shown in FIG. **2**, the transaction engine **205** communicates with the depository **210**, the authentication engine **215**, and the cryptographic engine **220**, along with the mass storage **225**. In addition, the depository **210** communicates with the authentication engine **215**, the cryptographic engine **220**, and the mass storage **225**. Moreover, the authentication engine **215** communicates with the cryptographic engine **220**. According to one embodiment of the invention, some or all of the foregoing communications may advantageously comprise the transmission of XML documents to IP addresses that correspond to the receiving device. As mentioned in the foregoing, XML documents advantageously allow designers to create their own customized document tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations. Moreover, some or all of the foregoing communications may include conventional SSL technologies.

According to one embodiment, the transaction engine **205** comprises a data routing device, such as a conventional Web server available from Netscape, Microsoft, Apache, or the like. For example, the Web server may advantageously receive incoming data from the communication link **125**. According to one embodiment of the invention, the incoming data is addressed to a front-end security system for the trust engine **110**. For example, the front-end security system may advantageously include a firewall, an intrusion detection system searching for known attack profiles, and/or a virus scanner. After clearing the front-end security system, the data is received by the transaction engine **205** and routed to one of the depository **210**, the authentication engine **215**, the cryptographic engine **220**, and the mass storage **225**. In addition, the transaction engine **205** monitors incoming data from the authentication engine **215** and cryptographic engine **220**, and routes the data to particular systems through the communication link **125**. For example, the transaction engine **205** may advantageously route data to the user system **105**, the certificate authority **115**, or the vendor system **120**.

According to one embodiment, the data is routed using conventional HTTP routing techniques, such as, for example, employing URLs or Uniform Resource Indicators (URIs). URIs are similar to URLs, however, URIs typically indicate the source of files or actions, such as, for example, executables, scripts, and the like. Therefore, according to the one embodiment, the user system **105**, the certificate authority **115**, the vendor system **120**, and the components of the trust engine **210**, advantageously include sufficient data within communication URLs or URIs for the transaction engine **205** to properly route data throughout the cryptographic system.

Although the data routing is disclosed with reference to its preferred embodiment, a skilled artisan will recognize a wide number of possible data routing solutions or strategies. For example, XML or other data packets may advantageously be unpacked and recognized by their format, content, or the like, such that the transaction engine **205** may properly route data throughout the trust engine **110**. Moreover, a skilled artisan will recognize that the data routing may advantageously be adapted to the data transfer protocols conforming to particular network systems, such as, for example, when the communication link **125** comprises a local network.

According to yet another embodiment of the invention, the transaction engine **205** includes conventional SSL encryption technologies, such that the foregoing systems may authenticate themselves, and vise-versa, with transaction engine **205**, during particular communications. As will be used throughout this disclosure, the term "½ SSL" refers to communications where a server but not necessarily the client, is SSL authenticated, and the term "FULL SSL" refers to communications where the client and the server are SSL authenticated. When the instant disclosure uses the term "SSL", the communication may comprise ½ or FULL SSL.

As the transaction engine **205** routes data to the various components of the cryptographic system **100**, the transaction engine **205** may advantageously create an audit trail. According to one embodiment, the audit trail includes a record of at least the type and format of data routed by the transaction engine **205** throughout the cryptographic system **100**. Such audit data may advantageously be stored in the mass storage **225**.

FIG. **2** also illustrates the depository **210**. According to one embodiment, the depository **210** comprises one or more data storage facilities, such as, for example, a directory server, a database server, or the like. As shown in FIG. **2**, the depository **210** stores cryptographic keys and enrollment authentication data. The cryptographic keys may advantageously correspond to the trust engine **110** or to users of the cryptographic system **100**, such as the user or vendor. The enrollment authentication data may advantageously include data designed to uniquely identify a user, such as, user ID, passwords, answers to questions, biometric data, or the like. This enrollment authentication data may advantageously be acquired at enrollment of a user or another alternative later time. For example, the trust engine **110** may include periodic or other renewal or reissue of enrollment authentication data.

According to one embodiment, the communication from the transaction engine **205** to and from the authentication engine **215** and the cryptographic engine **220** comprises secure communication, such as, for example conventional SSL technology. In addition, as mentioned in the foregoing, the data of the communications to and from the depository

210 may be transferred using URLs, URIs, HTTP or XML documents, with any of the foregoing advantageously having data requests and formats embedded therein.

As mentioned above, the depository 210 may advantageously comprises a plurality of secure data storage facilities. In such an embodiment, the secure data storage facilities may be configured such that a compromise of the security in one individual data storage facility will not compromise the cryptographic keys or the authentication data stored therein. For example, according to this embodiment, the cryptographic keys and the authentication data are mathematically operated on so as to statistically and substantially randomize the data stored in each data storage facility. According to one embodiment, the randomization of the data of an individual data storage facility renders that data undecipherable. Thus, compromise of an individual data storage facility produces only a randomized undecipherable number and does not compromise the security of any cryptographic keys or the authentication data as a whole.

FIG. 2 also illustrates the trust engine 110 including the authentication engine 215. According to one embodiment, the authentication engine 215 comprises a data comparator configured to compare data from the transaction engine 205 with data from the depository 210. For example, during authentication, a user supplies current authentication data to the trust engine 110 such that the transaction engine 205 receives the current authentication data. As mentioned in the foregoing, the transaction engine 205 recognizes the data requests, preferably in the URL or URI, and routes the authentication data to the authentication engine 215. Moreover, upon request, the depository 210 forwards enrollment authentication data corresponding to the user to the authentication engine 215. Thus, the authentication engine 215 has both the current authentication data and the enrollment authentication data for comparison.

According to one embodiment, the communications to the authentication engine comprise secure communications, such as, for example, SSL technology. Additionally, security can be provided within the trust engine 110 components, such as, for example, super-encryption using public key technologies. For example, according to one embodiment, the user encrypts the current authentication data with the public key of the authentication engine 215. In addition, the depository 210 also encrypts the enrollment authentication data with the public key of the authentication engine 215. In this way, only the authentication engine's private key can be used to decrypt the transmissions.

As shown in FIG. 2, the trust engine 110 also includes the cryptographic engine 220. According to one embodiment, the cryptographic engine comprises a cryptographic handling module, configured to advantageously provide conventional cryptographic functions, such as, for example, public-key infrastructure (PKI) functionality. For example, the cryptographic engine 220 may advantageously issue public and private keys for users of the cryptographic system 100. In this manner, the cryptographic keys are generated at the cryptographic engine 220 and forwarded to the depository 210 such that at least the private cryptographic keys are not available outside of the trust engine 110. According to another embodiment, the cryptographic engine 220 randomizes and splits at least the private cryptographic key data, thereby storing only the randomized split data. Similar to the splitting of the enrollment authentication data, the splitting process ensures the stored keys are not available outside the cryptographic engine 220. According to another embodiment, the functions of the cryptographic engine can be combined with and performed by the authentication engine 215.

According to one embodiment, communications to and from the cryptographic engine include secure communications, such as SSL technology. In addition, XML documents may advantageously be employed to transfer data and/or make cryptographic function requests.

FIG. 2 also illustrates the trust engine 110 having the mass storage 225. As mentioned in the foregoing, the transaction engine 205 keeps data corresponding to an audit trail and stores such data in the mass storage 225. Similarly, according to one embodiment of the invention, the depository 210 keeps data corresponding to an audit trail and stores such data in the mass storage device 225. The depository audit trail data is similar to that of the transaction engine 205 in that the audit trail data comprises a record of the requests received by the depository 210 and the response thereof. In addition, the mass storage 225 may be used to store digital certificates having the public key of a user contained therein.

Although the trust engine 110 is disclosed with reference to its preferred and alternative embodiments, the invention is not intended to be limited thereby. Rather, a skilled artisan will recognize in the disclosure herein, a wide number of alternatives for the trust engine 110. For example, the trust engine 110, may advantageously perform only authentication, or alternatively, only some or all of the cryptographic functions, such as data encryption and decryption. According to such embodiments, one of the authentication engine 215 and the cryptographic engine 220 may advantageously be removed, thereby creating a more straightforward design for the trust engine 110. In addition, the cryptographic engine 220 may also communicate with a certificate authority such that the certificate authority is embodied within the trust engine 110. According to yet another embodiment, the trust engine 110 may advantageously perform authentication and one or more cryptographic functions, such as, for example, digital signing.

FIG. 3 illustrates a block diagram of the transaction engine 205 of FIG. 2, according to aspects of an embodiment of the invention. According to this embodiment, the transaction engine 205 comprises an operating system 305 having a handling thread and a listening thread. The operating system 305 may advantageously be similar to those found in conventional high volume servers, such as, for example, Web servers available from Apache. The listening thread monitors the incoming communication from one of the communication link 125, the authentication engine 215, and the cryptographic engine 220 for incoming data flow. The handling thread recognizes particular data structures of the incoming data flow, such as, for example, the foregoing data structures, thereby routing the incoming data to one of the communication link 125, the depository 210, the authentication engine 215, the cryptographic engine 220, or the mass storage 225. As shown in FIG. 3, the incoming and outgoing data may advantageously be secured through, for example, SSL technology.

FIG. 4 illustrates a block diagram of the depository 210 of FIG. 2 according to aspects of an embodiment of the invention. According to this embodiment, the depository 210 comprises one or more lightweight directory access protocol (LDAP) servers. LDAP directory servers are available from a wide variety of manufacturers such as Netscape, ISO, and others. FIG. 4 also shows that the directory server preferably stores data 405 corresponding to the cryptographic keys and data 410 corresponding to the enrollment authentication data. According to one embodiment, the

depository **210** comprises a single logical memory structure indexing authentication data and cryptographic key data to a unique user ID. The single logical memory structure preferably includes mechanisms to ensure a high degree of trust, or security, in the data stored therein. For example, the physical location of the depository **210** may advantageously include a wide number of conventional security measures, such as limited employee access, modern surveillance systems, and the like. In addition to, or in lieu of, the physical securities, the computer system or server may advantageously include software solutions to protect the stored data. For example, the depository **210** may advantageously create and store data **415** corresponding to an audit trail of actions taken. In addition, the incoming and outgoing communications may advantageously be encrypted with public key encryption coupled with conventional SSL technologies.

According to another embodiment, the depository **210** may comprise distinct and physically separated data storage facilities, as disclosed further with reference to FIG. **7**.

FIG. **5** illustrates a block diagram of the authentication engine **215** of FIG. **2** according to aspects of an embodiment of the invention. Similar to the transaction engine **205** of FIG. **3**, the authentication engine **215** comprises an operating system **505** having at least a listening and a handling thread of a modified version of a conventional Web server, such as, for example, Web servers available from Apache. As shown in FIG. **5**, the authentication engine **215** includes access to at least one private key **510**. The private key **510** may advantageously be used for example, to decrypt data from the transaction engine **205** or the depository **210**, which was encrypted with a corresponding public key of the authentication engine **215**.

FIG. **5** also illustrates the authentication engine **215** comprising a comparator **515**, a data splitting module **520**, and a data assembling module **525**. According to the preferred embodiment of the invention, the comparator **515** includes technology capable of comparing potentially complex patterns related to the foregoing biometric authentication data. The technology may include hardware, software, or combined solutions for pattern comparisons, such as, for example, those representing finger print patterns or voice patterns. In addition, according to one embodiment, the comparator **515** of the authentication engine **215** may advantageously compare conventional hashes of documents in order to render a comparison result. According to one embodiment of the invention, the comparator **515** includes the application of heuristics **530** to the comparison. The heuristics **530** may advantageously address circumstances surrounding an authentication attempt, such as, for example, the time of day, IP address or subnet mask, purchasing profile, email address, processor serial number or ID, or the like.

Moreover, the nature of biometric data comparisons may result in varying degrees of confidence being produced from the matching of current biometric authentication data to enrollment data. For example, unlike a traditional password which may only return a positive or negative match, a fingerprint may be determined to be a partial match, e.g. a 90% match, a 75% match, or a 10% match, rather than simply being correct or incorrect. Other biometric identifiers such as voice print analysis or face recognition may share this property of probabilistic authentication, rather than absolute authentication.

When working with such probabilistic authentication or in other cases where an authentication is considered less than absolutely reliable, it is desirable to apply the heuristics **530** to determine whether the level of confidence in the authentication provided is sufficiently high to authenticate the transaction which is being made.

It will sometimes be the case that the transaction at issue is a relatively low value transaction where it is acceptable to be authenticated to a lower level of confidence. This could include a transaction which has a low dollar value associated with it (e.g., a $10 purchase) or a transaction with low risk (e.g., admission to a members-only web site).

Conversely, for authenticating other transactions, it may be desirable to require a high degree of confidence in the authentication before allowing the transaction to proceed. Such transactions may include transactions of large dollar value (e.g., signing a multi-million dollar supply contract) or transaction with a high risk if an improper authentication occurs (e.g., remotely logging onto a government computer).

The use of the heuristics **530** in combination with confidence levels and transactions values may be used as will be described below to allow the comparator to provide a dynamic context-sensitive authentication system.

According to another embodiment of the invention, the comparator **515** may advantageously track authentication attempts for a particular transaction. For example, when a transaction fails, the trust engine **110** may request the user to re-enter his or her current authentication data. The comparator **515** of the authentication engine **215** may advantageously employ an attempt limiter **535** to limit the number of authentication attempts, thereby prohibiting brute-force attempts to impersonate a user's authentication data. According to one embodiment, the attempt limiter **535** comprises a software module monitoring transactions for repeating authentication attempts and, for example, limiting the authentication attempts for a given transaction to three. Thus, the attempt limiter **535** will limit an automated attempt to impersonate an individual's authentication data to, for example, simply three "guesses." Upon three failures, the attempt limiter **535** may advantageously deny additional authentication attempts. Such denial may advantageously be implemented through, for example, the comparator **515** returning a negative result regardless of the current authentication data being transmitted. On the other hand, the transaction engine **205** may advantageously block any additional authentication attempts pertaining to a transaction in which three attempts have previously failed.

The authentication engine **215** also includes the data splitting module **520** and the data assembling module **525**. The data splitting module **520** advantageously comprises a software, hardware, or combination module having the ability to mathematically operate on various data so as to substantially randomize and split the data into portions. According to one embodiment, original data is not recreatable from an individual portion. The data assembling module **525** advantageously comprises a software, hardware, or combination module configured to mathematically operate on the foregoing substantially randomized portions, such that the combination thereof provides the original deciphered data. According to one embodiment, the authentication engine **215** employs the data splitting module **520** to randomize and split enrollment authentication data into portions, and employs the data assembling module **525** to reassemble the portions into usable enrollment authentication data.

FIG. **6** illustrates a block diagram of the cryptographic engine **220** of the trust engine **200** of FIG. **2** according to aspects of one embodiment of the invention. Similar to the transaction engine **205** of FIG. **3**, the cryptographic engine **220** comprises an operating system **605** having at least a listening and a handling thread of a modified version of a

conventional Web server, such as, for example, Web servers available from Apache. As shown in FIG. **6**, the cryptographic engine **220** comprises a data splitting module **610** and a data assembling module **620** that function similar to those of FIG. **5**. However, according to one embodiment, the data splitting module **610** and the data assembling module **620** process cryptographic key data, as opposed to the foregoing enrollment authentication data. Although, a skilled artisan will recognize from the disclosure herein that the data splitting module **910** and the data splitting module **620** may be combined with those of the authentication engine **215**.

The cryptographic engine **220** also comprises a cryptographic handling module **625** configured to perform one, some or all of a wide number of cryptographic functions. According to one embodiment, the cryptographic handling module **625** may comprise software modules or programs, hardware, or both. According to another embodiment, the cryptographic handling module **625** may perform data comparisons, data parsing, data splitting, data separating, data hashing, data encryption or decryption, digital signature verification or creation, digital certificate generation, storage, or requests, cryptographic key generation, or the like. Moreover, a skilled artisan will recognize from the disclosure herein that the cryptographic handling module **825** may advantageously comprises a public-key infrastructure, such as Pretty Good Privacy (PGP), an RSA-based public-key system, or a wide number of alternative key management systems. In addition, the cryptographic handling module **625** may perform public-key encryption, symmetric-key encryption, or both. In addition to the foregoing, the cryptographic handling module **625** may include one or more computer programs or modules, hardware, or both, for implementing seamless, transparent, interoperability functions.

A skilled artisan will also recognize from the disclosure herein that the cryptographic functionality may include a wide number or variety of functions generally relating to cryptographic key management systems.

FIG. **7** illustrates a simplified block diagram of a depository system **700** according to aspects of an embodiment of the invention. As shown in FIG. **7**, the depository system **700** advantageously comprises multiple data storage facilities, for example, data storage facilities D**1**, D**2**, D**3**, and D**4**. However, it is readily understood by those of ordinary skill in the art that the depository system may have only one data storage facility. According to one embodiment of the invention, each of the data storage facilities D**1** through D**4** may advantageously comprise some or all of the elements disclosed with reference to the depository **210** of FIG. **4**. Similar to the depository **210**, the data storage facilities D**1** through D**4** communicate with the transaction engine **205**, the authentication engine **215**, and the cryptographic engine **220**, preferably through conventional SSL. Communication links transferring, for example, XML documents. Communications from the transaction engine **205** may advantageously include requests for data, wherein the request is advantageously broadcast to the IP address of each data storage facility D**1** through D**4**. On the other hand, the transaction engine **205** may broadcast requests to particular data storage facilities based on a wide number of criteria, such as, for example, response time, server loads, maintenance schedules, or the like.

In response to requests for data from the transaction engine **205**, the depository system **700** advantageously forwards stored data to the authentication engine **215** and the cryptographic engine **220**. The respective data assembling modules receive the forwarded data and assemble the data into usable formats. On the other hand, communications from the authentication engine **215** and the cryptographic engine **220** to the data storage facilities D**1** through D**4** may include the transmission of sensitive data to be stored. For example, according to one embodiment, the authentication engine **215** and the cryptographic engine **220** may advantageously employ their respective data splitting modules to divide sensitive data into undecipherable portions, and then transmit one or more undecipherable portions of the sensitive data to a particular data storage facility.

According to one embodiment, each data storage facility, D**1** through D**4**, comprises a separate and independent storage system, such as, for example, a directory server. According to another embodiment of the invention, the depository system **700** comprises multiple geographically separated independent data storage systems. By distributing the sensitive data into distinct and independent storage facilities D**1** through D**4**, some or all of which may be advantageously geographically separated, the depository system **700** provides redundancy along with additional security measures. For example, according to one embodiment, only data from two of the multiple data storage facilities, D**1** through D**4**, are needed to decipher and reassemble the sensitive data. Thus, as many as two of the four data storage facilities D**1** through D**4** may be inoperative due to maintenance, system failure, power failure, or the like, without affecting the functionality of the trust engine **110**. In addition, because, according to one embodiment, the data stored in each data storage facility is randomized and undecipherable, compromise of any individual data storage facility does not necessarily compromise the sensitive data. Moreover, in the embodiment having geographical separation of the data storage facilities, a compromise of multiple geographically remote facilities becomes increasingly difficult. In fact, even a rogue employee will be greatly challenged to subvert the needed multiple independent geographically remote data storage facilities.

Although the depository system **700** is disclosed with reference to its preferred and alternative embodiments, the invention is not intended to be limited thereby. Rather, a skilled artisan will recognize from the disclosure herein, a wide number of alternatives for the depository system **700**. For example, the depository system **700** may comprise one, two or more data storage facilities. In addition, sensitive data may be mathematically operated such that portions from two or more data storage facilities are needed to reassemble and decipher the sensitive data.

As mentioned in the foregoing, the authentication engine **215** and the cryptographic engine **220** each include a data splitting module **520** and **610**, respectively, for splitting any type or form of sensitive data, such as, for example, text, audio, video, the authentication data and the cryptographic key data. FIG. **8** illustrates a flowchart of a data splitting process **800** performed by the data splitting module according to aspects of an embodiment of the invention. As shown in FIG. **8**, the data splitting process **800** begins at step **805** when sensitive data "S" is received by the data splitting module of the authentication engine **215** or the cryptographic engine **220**. Preferably, in step **810**, the data splitting module then generates a substantially random number, value, or string or set of bits, "A." For example, the random number A may be generated in a wide number of varying conventional techniques available to one of ordinary skill in the art, for producing high quality random numbers suitable for use in cryptographic applications. In addition, according to one embodiment, the random number A comprises a bit

length which may be any suitable length, such as shorter, longer or equal to the bit length of the sensitive data, S.

In addition, in step **820** the data splitting process **800** generates another statistically random number "C." According to the preferred embodiment, the generation of the statistically random numbers A and C may advantageously be done in parallel. The data splitting module then combines the numbers A and C with the sensitive data S such that new numbers "B" and "D" are generated. For example, number B may comprise the binary combination of A XOR S and number D may comprise the binary combination of C XOR S. The XOR function, or the "exclusive-or" function, is well known to those of ordinary skill in the art. The foregoing combinations preferably occur in steps **825** and **830**, respectively, and, according to one embodiment, the foregoing combinations also occur in parallel. The data splitting process **800** then proceeds to step **835** where the random numbers A and C and the numbers B and D are paired such that none of the pairings contain sufficient data, by themselves, to reorganize and decipher the original sensitive data S. For example, the numbers may be paired as follows: AC, AD, BC, and BD. According to one embodiment, each of the foregoing pairings is distributed to one of the depositories D**1** through D**4** of FIG. **7**. According to another embodiment, each of the foregoing pairings is randomly distributed to one of the depositories D**1** through D**4**. For example, during a first data splitting process **800**, the pairing AC may be sent to depository D**2**, through, for example, a random selection of D**2**'s IP address. Then, during a second data splitting process **800**, the pairing AC may be sent to depository D**4**, through, for example, a random selection of D**4**'s IP address. In addition, the pairings may all be stored on one depository, and may be stored in separate locations on said depository.

Based on the foregoing, the data splitting process **800** advantageously places portions of the sensitive data in each of the four data storage facilities D**1** through D**4**, such that no single data storage facility D**1** through D**4** includes sufficient encrypted data to recreate the original sensitive data S. As mentioned in the foregoing, such randomization of the data into individually unusable encrypted portions increases security and provides for maintained trust in the data even if one of the data storage facilities, D**1** through D**4**, is compromised.

Although the data splitting process **800** is disclosed with reference to its preferred embodiment, the invention is not intended to be limited thereby. Rather a skilled artisan will recognize from the disclosure herein, a wide number of alternatives for the data splitting process **800**. For example, the data splitting process may advantageously split the data into two numbers, for example, random number A and number B and, randomly distribute A and B through two data storage facilities. Moreover, the data splitting process **800** may advantageously split the data among a wide number of data storage facilities through generation of additional random numbers. The data may be split into any desired, selected, predetermined, or randomly assigned size unit, including but not limited to, a bit, bits, bytes, kilobytes, megabytes or larger, or any combination or sequence of sizes. In addition, varying the sizes of the data units resulting from the splitting process may render the data more difficult to restore to a usable form, thereby increasing security of sensitive data. It is readily apparent to those of ordinary skill in the art that the split data unit sizes may be a wide variety of data unit sizes or patterns of sizes or combinations of sizes. For example, the data unit sizes may be selected or predetermined to be all of the same size, a fixed set of different sizes, a combination of sizes, or randomly generates sizes. Similarly, the data units may be distributed into one or more shares according to a fixed or predetermined data unit size, a pattern or combination of data unit sizes, or a randomly generated data unit size or sizes per share.

As mentioned in the foregoing, in order to recreate the sensitive data S, the data portions need to be derandomized and reorganized. This process may advantageously occur in the data assembling modules, **525** and **620**, of the authentication engine **215** and the cryptographic engine **220**, respectively. The data assembling module, for example, data assembly module **525**, receives data portions from the data storage facilities D**1** through D**4**, and reassembles the data into usable form. For example, according to one embodiment where the data splitting module **520** employed the data splitting process **800** of FIG. **8**, the data assembling module **525** uses data portions from at least two of the data storage facilities D**1** through D**4** to recreate the sensitive data S. For example, the pairings of AC, AD, BC, and BD, were distributed such that any two provide one of A and B, or, C and D. Noting that S=A XOR B or S=C XOR D indicates that when the data assembling module receives one of A and B, or, C and D, the data assembling module **525** can advantageously reassemble the sensitive data S. Thus, the data assembling module **525** may assemble the sensitive data S, when, for example, it receives data portions from at least the first two of the data storage facilities D**1** through D**4** to respond to an assemble request by the trust engine **110**.

Based on the above data splitting and assembling processes, the sensitive data S exists in usable format only in a limited area of the trust engine **110**. For example, when the sensitive data S includes enrollment authentication data, usable, nonrandomized enrollment authentication data is available only in the authentication engine **215**. Likewise, when the sensitive data S includes private cryptographic key data, usable, nonrandomized private cryptographic key data is available only in the cryptographic engine **220**.

Although the data splitting and assembling processes are disclosed with reference to their preferred embodiments, the invention is not intended to be limited thereby. Rather, a skilled artisan will recognize from the disclosure herein, a wide number of alternatives for splitting and reassembling the sensitive data S. For example, public-key encryption may be used to further secure the data at the data storage facilities D**1** through D**4**. In addition, it is readily apparent to those of ordinary skill in the art that the data splitting module described herein is also a separate and distinct embodiment of the present invention that may be incorporated into, combined with or otherwise made part of any pre-existing computer systems, software suites, database, or combinations thereof, or other embodiments of the present invention, such as the trust engine, authentication engine, and transaction engine disclosed and described herein.

FIG. **9**A illustrates a data flow of an enrollment process **900** according to aspects of an embodiment of the invention. As shown in FIG. **9**A, the enrollment process **900** begins at step **905** when a user desires to enroll with the trust engine **110** of the cryptographic system **100**. According to this embodiment, the user system **105** advantageously includes a client-side applet, such as a Java-based, that queries the user to enter enrollment data, such as demographic data and enrollment authentication data. According to one embodiment, the enrollment authentication data includes user ID, password(s), biometric(s), or the like. According to one embodiment, during the querying process, the client-side applet preferably communicates with the trust engine **110** to ensure that a chosen user ID is unique. When the user ID is nonunique, the trust engine **110** may advantageously suggest

a unique user ID. The client-side applet gathers the enrollment data and transmits the enrollment data, for example, through and XML document, to the trust engine **110**, and in particular, to the transaction engine **205**. According to one embodiment, the transmission is encoded with the public key of the authentication engine **215**.

According to one embodiment, the user performs a single enrollment during step **905** of the enrollment process **900**. For example, the user enrolls himself or herself as a particular person, such as Joe User. When Joe User desires to enroll as Joe User, CEO of Mega Corp., then according to this embodiment, Joe User enrolls a second time, receives a second unique user ID and the trust engine **110** does not associate the two identities. According to another embodiment of the invention, the enrollment process **900** provides for multiple user identities for a single user ID. Thus, in the above example, the trust engine **110** will advantageously associate the two identities of Joe User. As will be understood by a skilled artisan from the disclosure herein, a user may have many identities, for example, Joe User the head of household, Joe User the member of the Charitable Foundations, and the like. Even though the user may have multiple identities, according to this embodiment, the trust engine **110** preferably stores only one set of enrollment data. Moreover, users may advantageously add, edit/update, or delete identities as they are needed.

Although the enrollment process **900** is disclosed with reference to its preferred embodiment, the invention is not intended to be limited thereby. Rather, a skilled artisan will recognize from the disclosure herein, a wide number of alternatives for gathering of enrollment data, and in particular, enrollment authentication data. For example, the applet may be common object model (COM) based applet or the like.

On the other hand, the enrollment process may include graded enrollment. For example, at a lowest level of enrollment, the user may enroll over the communication link **125** without producing documentation as to his or her identity. According to an increased level of enrollment, the user enrolls using a trusted third party, such as a digital notary. For example, and the user may appear in person to the trusted third party, produce credentials such as a birth certificate, driver's license, military ID, or the like, and the trusted third party may advantageously include, for example, their digital signature in enrollment submission. The trusted third party may include an actual notary, a government agency, such as the Post Office or Department of Motor Vehicles, a human resources person in a large company enrolling an employee, or the like. A skilled artisan will understand from the disclosure herein that a wide number of varying levels of enrollment may occur during the enrollment process **900**.

After receiving the enrollment authentication data, at step **915**, the transaction engine **205**, using conventional FULL SSL technology forwards the enrollment authentication data to the authentication engine **215**. In step **920**, the authentication engine **215** decrypts the enrollment authentication data using the private key of the authentication engine **215**. In addition, the authentication engine **215** employs the data splitting module to mathematically operate on the enrollment authentication data so as to split the data into at least two independently undecipherable, randomized, numbers. As mentioned in the foregoing, at least two numbers may comprise a statistically random number and a binary X0Red number. In step **925**, the authentication engine **215** forwards each portion of the randomized numbers to one of the data storage facilities D**1** through D**4**. As mentioned in the foregoing, the authentication engine **215** may also advantageously randomize which portions are transferred to which depositories.

Often during the enrollment process **900**, the user will also desire to have a digital certificate issued such that he or she may receive encrypted documents from others outside the cryptographic system **100**. As mentioned in the foregoing, the certificate authority **115** generally issues digital certificates according to one or more of several conventional standards. Generally, the digital certificate includes a public key of the user or system, which is known to everyone.

Whether the user requests a digital certificate at enrollment, or at another time, the request is transferred through the trust engine **110** to the authentication engine **215**. According to one embodiment, the request includes an XML document having, for example, the proper name of the user. According to step **935**, the authentication engine **215** transfers the request to the cryptographic engine **220** instructing the cryptographic engine **220** to generate a cryptographic key or key pair.

Upon request, at step **935**, the cryptographic engine **220** generates at least one cryptographic key. According to one embodiment, the cryptographic handling module **625** generates a key pair, where one key is used as a private key, and one is used as a public key. The cryptographic engine **220** stores the private key and, according to one embodiment, a copy of the public key. In step **945**, the cryptographic engine **220** transmits a request for a digital certificate to the transaction engine **205**. According to one embodiment, the request advantageously includes a standardized request, such as PKCS10, embedded in, for example, an XML document. The request for a digital certificate may advantageously correspond to one or more certificate authorities and the one or more standard formats the certificate authorities require.

In step **950** the transaction engine **205** forwards this request to the certificate authority **115**, who, in step **955**, returns a digital certificate. The return digital certificate may advantageously be in a standardized format, such as PKCS7, or in a proprietary format of one or more of the certificate authorities **115**. In step **960**, the digital certificate is received by the transaction engine **205**, and a copy is forwarded to the user and a copy is stored with the trust engine **110**. The trust engine **110** stores a copy of the certificate such that the trust engine **110** will not need to rely on the availability of the certificate authority **115**. For example, when the user desires to send a digital certificate, or a third party requests the user's digital certificate, the request for the digital certificate is typically sent to the certificate authority **115**. However, if the certificate authority **115** is conducting maintenance or has been victim of a failure or security compromise, the digital certificate may not be available.

At any time after issuing the cryptographic keys, the cryptographic engine **220** may advantageously employ the data splitting process **800** described above such that the cryptographic keys are split into independently undecipherable randomized numbers. Similar to the authentication data, at step **965** the cryptographic engine **220** transfers the randomized numbers to the data storage facilities D**1** through D**4**.

A skilled artisan will recognize from the disclosure herein that the user may request a digital certificate anytime after enrollment. Moreover, the communications between systems may advantageously include FULL SSL or public-key encryption technologies. Moreover, the enrollment process may issue multiple digital certificates from multiple certifi-

cate authorities, including one or more proprietary certificate authorities internal or external to the trust engine **110**.

As disclosed in steps **935** through **960**, one embodiment of the invention includes the request for a certificate that is eventually stored on the trust engine **110**. Because, according to one embodiment, the cryptographic handling module **625** issues the keys used by the trust engine **110**, each certificate corresponds to a private key. Therefore, the trust engine **110** may advantageously provide for interoperability through monitoring the certificates owned by, or associated with, a user. For example, when the cryptographic engine **220** receives a request for a cryptographic function, the cryptographic handling module **625** may investigate the certificates owned by the requesting user to determine whether the user owns a private key matching the attributes of the request. When such a certificate exists, the cryptographic handling module **625** may use the certificate or the public or private keys associated therewith, to perform the requested function. When such a certificate does not exist, the cryptographic handling module **625** may advantageously and transparently perform a number of actions to attempt to remedy the lack of an appropriate key. For example, FIG. **9**B illustrates a flowchart of an interoperability process **970**, which according to aspects of an embodiment of the invention, discloses the foregoing steps to ensure the cryptographic handling module **625** performs cryptographic functions using appropriate keys.

As shown in FIG. **9**B, the interoperability process **970** begins with step **972** where the cryptographic handling module **925** determines the type of certificate desired. According to one embodiment of the invention, the type of certificate may advantageously be specified in the request for cryptographic functions, or other data provided by the requestor. According to another embodiment, the certificate type may be ascertained by the data format of the request. For example, the cryptographic handling module **925** may advantageously recognize the request corresponds to a particular type.

According to one embodiment, the certificate type may include one or more algorithm standards, for example, RSA, ELGAMAL, or the like. In addition, the certificate type may include one or more key types, such as symmetric keys, public keys, strong encryption keys such as 256 bit keys, less secure keys, or the like. Moreover, the certificate type may include upgrades or replacements of one or more of the foregoing algorithm standards or keys, one or more message or data formats, one or more data encapsulation or encoding schemes, such as Base 32 or Base 64. The certificate type may also include compatibility with one or more third-party cryptographic applications or interfaces, one or more communication protocols, or one or more certificate standards or protocols. A skilled artisan will recognize from the disclosure herein that other differences may exist in certificate types, and translations to and from those differences may be implemented as disclosed herein.

Once the cryptographic handling module **625** determines the certificate type, the interoperability process **970** proceeds to step **974**, and determines whether the user owns a certificate matching the type determined in step **974**. When the user owns a matching certificate, for example, the trust engine **110** has access to the matching certificate through, for example, prior storage thereof, the cryptographic handling module **825** knows that a matching private key is also stored within the trust engine **110**. For example, the matching private key may be stored within the depository **210** or depository system **700**. The cryptographic handling module **625** may advantageously request the matching private key be assembled from, for example, the depository **210**, and then in step **976**, use the matching private key to perform cryptographic actions or functions. For example, as mentioned in the foregoing, the cryptographic handling module **625** may advantageously perform hashing, hash comparisons, data encryption or decryption, digital signature verification or creation, or the like.

When the user does not own a matching certificate, the interoperability process **970** proceeds to step **978** where the cryptographic handling module **625** determines whether the users owns a cross-certified certificate. According to one embodiment, cross-certification between certificate authorities occurs when a first certificate authority determines to trust certificates from a second certificate authority. In other words, the first certificate authority determines that certificates from the second certificate authority meets certain quality standards, and therefore, may be "certified" as equivalent to the first certificate authority's own certificates. Cross-certification becomes more complex when the certificate authorities issue, for example, certificates having levels of trust. For example, the first certificate authority may provide three levels of trust for a particular certificate, usually based on the degree of reliability in the enrollment process, while the second certificate authority may provide seven levels of trust. Cross-certification may advantageously track which levels and which certificates from the second certificate authority may be substituted for which levels and which certificates from the first. When the foregoing cross-certification is done officially and publicly between two certification authorities, the mapping of certificates and levels to one another is often called "chaining."

According to another embodiment of the invention, the cryptographic handling module **625** may advantageously develop cross-certifications outside those agreed upon by the certificate authorities. For example, the cryptographic handling module **625** may access a first certificate authority's certificate practice statement (CPS), or other published policy statement, and using, for example, the authentication tokens required by particular trust levels, match the first certificate authority's certificates to those of another certificate authority.

When, in step **978**, the cryptographic handling module **625** determines that the users owns a cross-certified certificate, the interoperability process **970** proceeds to step **976**, and performs the cryptographic action or function using the cross-certified public key, private key, or both. Alternatively, when the cryptographic handling module **625** determines that the users does not own a cross-certified certificate, the interoperability process **970** proceeds to step **980**, where the cryptographic handling module **625** selects a certificate authority that issues the requested certificate type, or a certificate cross-certified thereto. In step **982**, the cryptographic handling module **625** determines whether the user enrollment authentication data, discussed in the foregoing, meets the authentication requirements of the chosen certificate authority. For example, if the user enrolled over a network by, for example, answering demographic and other questions, the authentication data provided may establish a lower level of trust than a user providing biometric data and appearing before a third-party, such as, for example, a notary. According to one embodiment, the foregoing authentication requirements may advantageously be provided in the chosen authentication authority's CPS.

When the user has provided the trust engine **110** with enrollment authentication data meeting the requirements of chosen certificate authority, the interoperability process **970** proceeds to step **984**, where the cryptographic handling

module **825** acquires the certificate from the chosen certificate authority. According to one embodiment, the cryptographic handling module **625** acquires the certificate by following steps **945** through **960** of the enrollment process **900**. For example, the cryptographic handling module **625** may advantageously employ one or more public keys from one or more of the key pairs already available to the cryptographic engine **220**, to request the certificate from the certificate authority. According to another embodiment, the cryptographic handling module **625** may advantageously generate one or more new key pairs, and use the public keys corresponding thereto, to request the certificate from the certificate authority.

According to another embodiment, the trust engine **110** may advantageously include one or more certificate issuing modules capable of issuing one or more certificate types. According to this embodiment, the certificate issuing module may provide the foregoing certificate. When the cryptographic handling module **625** acquires the certificate, the interoperability process **970** proceeds to step **976**, and performs the cryptographic action or function using the public key, private key, or both corresponding to the acquired certificate.

When the user, in step **982**, has not provided the trust engine **110** with enrollment authentication data meeting the requirements of chosen certificate authority, the cryptographic handling module **625** determines, in step **986** whether there are other certificate authorities that have different authentication requirements. For example, the cryptographic handling module **625** may look for certificate authorities having lower authentication requirements, but still issue the chosen certificates, or cross-certifications thereof.

When the foregoing certificate authority having lower requirements exists, the interoperability process **970** proceeds to step **980** and chooses that certificate authority. Alternatively, when no such certificate authority exists, in step **988**, the trust engine **110** may request additional authentication tokens from the user. For example, the trust engine **110** may request new enrollment authentication data comprising, for example, biometric data. Also, the trust engine **110** may request the user appear before a trusted third party and provide appropriate authenticating credentials, such as, for example, appearing before a notary with a drivers license, social security card, bank card, birth certificate, military ID, or the like. When the trust engine **110** receives updated authentication data, the interoperability process **970** proceeds to step **984** and acquires the foregoing chosen certificate.

Through the foregoing interoperability process **970**, the cryptographic handling module **625** advantageously provides seamless, transparent, translations and conversions between differing cryptographic systems. A skilled artisan will recognize from the disclosure herein, a wide number of advantages and implementations of the foregoing interoperable system. For example, the foregoing step **986** of the interoperability process **970** may advantageously include aspects of trust arbitrage, discussed in further detail below, where the certificate authority may under special circumstances accept lower levels of cross-certification. In addition, the interoperability process **970** may include ensuring interoperability between and employment of standard certificate revocations, such as employing certificate revocation lists (CRL), online certificate status protocols (OCSP), or the like.

FIG. **10** illustrates a data flow of an authentication process **1000** according to aspects of an embodiment of the invention. According to one embodiment, the authentication process **1000** includes gathering current authentication data from a user and comparing that to the enrollment authentication data of the user. For example, the authentication process **1000** begins at step **1005** where a user desires to perform a transaction with, for example, a vendor. Such transactions may include, for example, selecting a purchase option, requesting access to a restricted area or device of the vendor system **120**, or the like. At step **1010**, a vendor provides the user with a transaction ID and an authentication request. The transaction ID may advantageously include a 192 bit quantity having a 32 bit timestamp concatenated with a 128 bit random quantity, or a "nonce," concatenated with a 32 bit vendor specific constant. Such a transaction ID uniquely identifies the transaction such that copycat transactions can be refused by the trust engine **110**.

The authentication request may advantageously include what level of authentication is needed for a particular transaction. For example, the vendor may specify a particular level of confidence that is required for the transaction at issue. If authentication cannot be made to this level of confidence, as will be discussed below, the transaction will not occur without either further authentication by the user to raise the level of confidence, or a change in the terms of the authentication between the vendor and the server. These issues are discussed more completely below.

According to one embodiment, the transaction ID and the authentication request may be advantageously generated by a vendor-side applet or other software program. In addition, the transmission of the transaction ID and authentication data may include one or more XML documents encrypted using conventional SSL technology, such as, for example, ½ SSL, or, in other words vendor-side authenticated SSL.

After the user system **105** receives the transaction ID and authentication request, the user system **105** gathers the current authentication data, potentially including current biometric information, from the user. The user system **105**, at step **1015**, encrypts at least the current authentication data "B" and the transaction ID, with the public key of the authentication engine **215**, and transfers that data to the trust engine **110**. The transmission preferably comprises XML documents encrypted with at least conventional ½ SSL technology. In step **1020**, the transaction engine **205** receives the transmission, preferably recognizes the data format or request in the URL or URI, and forwards the transmission to the authentication engine **215**.

During steps **1015** and **1020**, the vendor system **120**, at step **1025**, forwards the transaction ID and the authentication request to the trust engine **110**, using the preferred FULL SSL technology. This communication may also include a vendor ID, although vendor identification may also be communicated through a non-random portion of the transaction ID. At steps **1030** and **1035**, the transaction engine **205** receives the communication, creates a record in the audit trail, and generates a request for the user's enrollment authentication data to be assembled from the data storage facilities D**1** through D**4**. At step **1040**, the depository system **700** transfers the portions of the enrollment authentication data corresponding to the user to the authentication engine **215**. At step **1045**, the authentication engine **215** decrypts the transmission using its private key and compares the enrollment authentication data to the current authentication data provided by the user.

The comparison of step **1045** may advantageously apply heuristical context sensitive authentication, as referred to in the forgoing, and discussed in further detail below. For example, if the biometric information received does not

match perfectly, a lower confidence match results. In particular embodiments, the level of confidence of the authentication is balanced against the nature of the transaction and the desires of both the user and the vendor. Again, this is discussed in greater detail below.

At step **1050**, the authentication engine **215** fills in the authentication request with the result of the comparison of step **1045**. According to one embodiment of the invention, the authentication request is filled with a YES/NO or TRUE/FALSE result of the authentication process **1000**. In step **1055** the filled-in authentication request is returned to the vendor for the vendor to act upon, for example, allowing the user to complete the transaction that initiated the authentication request. According to one embodiment, a confirmation message is passed to the user.

Based on the foregoing, the authentication process **1000** advantageously keeps sensitive data secure and produces results configured to maintain the integrity of the sensitive data. For example, the sensitive data is assembled only inside the authentication engine **215**. For example, the enrollment authentication data is undecipherable until it is assembled in the authentication engine **215** by the data assembling module, and the current authentication data is undecipherable until it is unwrapped by the conventional SSL technology and the private key of the authentication engine **215**. Moreover, the authentication result transmitted to the vendor does not include the sensitive data, and the user may not even know whether he or she produced valid authentication data.

Although the authentication process **1000** is disclosed with reference to its preferred and alternative embodiments, the invention is not intended to be limited thereby. Rather, a skilled artisan will recognize from the disclosure herein, a wide number of alternatives for the authentication process **1000**. For example, the vendor may advantageously be replaced by almost any requesting application, even those residing with the user system **105**. For example, a client application, such as Microsoft Word, may use an application program interface (API) or a cryptographic API (CAPI) to request authentication before unlocking a document. Alternatively, a mail server, a network, a cellular phone, a personal or mobile computing device, a workstation, or the like, may all make authentication requests that can be filled by the authentication process **1000**. In fact, after providing the foregoing trusted authentication process **1000**, the requesting application or device may provide access to or use of a wide number of electronic or computer devices or systems.

Moreover, the authentication process **1000** may employ a wide number of alternative procedures in the event of authentication failure. For example, authentication failure may maintain the same transaction ID and request that the user reenter his or her current authentication data. As mentioned in the foregoing, use of the same transaction ID allows the comparator of the authentication engine **215** to monitor and limit the number of authentication attempts for a particular transaction, thereby creating a more secure cryptographic system **100**.

In addition, the authentication process **1000** may be advantageously be employed to develop elegant single sign-on solutions, such as, unlocking a sensitive data vault. For example, successful or positive authentication may provide the authenticated user the ability to automatically access any number of passwords for an almost limitless number of systems and applications. For example, authentication of a user may provide the user access to password, login, financial credentials, or the like, associated with multiple online vendors, a local area network, various personal computing devices, Internet service providers, auction providers, investment brokerages, or the like. By employing a sensitive data vault, users may choose truly large and random passwords because they no longer need to remember them through association. Rather, the authentication process **1000** provides access thereto. For example, a user may choose a random alphanumeric string that is twenty plus digits in length rather than something associated with a memorable data, name, etc.

According to one embodiment, a sensitive data vault associated with a given user may advantageously be stored in the data storage facilities of the depository **210**, or split and stored in the depository system **700**. According to this embodiment, after positive user authentication, the trust engine **110** serves the requested sensitive data, such as, for example, to the appropriate password to the requesting application. According to another embodiment, the trust engine **110** may include a separate system for storing the sensitive data vault. For example, the trust engine **110** may include a stand-alone software engine implementing the data vault functionality and figuratively residing "behind" the foregoing front-end security system of the trust engine **110**. According to this embodiment, the software engine serves the requested sensitive data after the software engine receives a signal indicating positive user authentication from the trust engine **110**.

In yet another embodiment, the data vault may be implemented by a third-party system. Similar to the software engine embodiment, the third-party system may advantageously serve the requested sensitive data after the third-party system receives a signal indicating positive user authentication from the trust engine **110**. According to yet another embodiment, the data vault may be implemented on the user system **105**. A user-side software engine may advantageously serve the foregoing data after receiving a signal indicating positive user authentication from the trust engine **110**.

Although the foregoing data vaults are disclosed with reference to alternative embodiments, a skilled artisan will recognize from the disclosure herein, a wide number of additional implementations thereof. For example, a particular data vault may include aspects from some or all of the foregoing embodiments. In addition, any of the foregoing data vaults may employ one or more authentication requests at varying times. For example, any of the data vaults may require authentication every one or more transactions, periodically, every one or more sessions, every access to one or more Webpages or Websites, at one or more other specified intervals, or the like.

FIG. **11** illustrates a data flow of a signing process **1100** according to aspects of an embodiment of the invention. As shown in FIG. **11**, the signing process **1100** includes steps similar to those of the authentication process **1000** described in the foregoing with reference to FIG. **10**. According to one embodiment of the invention, the signing process **1100** first authenticates the user and then performs one or more of several digital signing functions as will be discussed in further detail below. According to another embodiment, the signing process **1100** may advantageously store data related thereto, such as hashes of messages or documents, or the like. This data may advantageously be used in an audit or any other event, such as for example, when a participating party attempts to repudiate a transaction.

As shown in FIG. **11**, during the authentication steps, the user and vendor may advantageously agree on a message, such as, for example, a contract. During signing, the signing

process **1100** advantageously ensures that the contract signed by the user is identical to the contract supplied by the vendor. Therefore, according to one embodiment, during authentication, the vendor and the user include a hash of their respective copies of the message or contract, in the data transmitted to the authentication engine **215**. By employing only a hash of a message or contract, the trust engine **110** may advantageously store a significantly reduced amount of data, providing for a more efficient and cost effective cryptographic system. In addition, the stored hash may be advantageously compared to a hash of a document in question to determine whether the document in question matches one signed by any of the parties. The ability to determine whether the document is identical to one relating to a transaction provides for additional evidence that can be used against a claim for repudiation by a party to a transaction.

In step **1103**, the authentication engine **215** assembles the enrollment authentication data and compares it to the current authentication data provided by the user. When the comparator of the authentication engine **215** indicates that the enrollment authentication data matches the current authentication data, the comparator of the authentication engine **215** also compares the hash of the message supplied by the vendor to the hash of the message supplied by the user. Thus, the authentication engine **215** advantageously ensures that the message agreed to by the user is identical to that agreed to by the vendor.

In step **1105**, the authentication engine **215** transmits a digital signature request to the cryptographic engine **220**. According to one embodiment of the invention, the request includes a hash of the message or contract. However, a skilled artisan will recognize from the disclosure herein that the cryptographic engine **220** may encrypt virtually any type of data, including, but not limited to, video, audio, biometrics, images or text to form the desired digital signature. Returning to step **1105**, the digital signature request preferably comprises an XML document communicated through conventional SSL technologies.

In step **1110**, the authentication engine **215** transmits a request to each of the data storage facilities D**1** through D**4**, such that each of the data storage facilities D**1** through D**4** transmit their respective portion of the cryptographic key or keys corresponding to a signing party. According to another embodiment, the cryptographic engine **220** employs some or all of the steps of the interoperability process **970** discussed in the foregoing, such that the cryptographic engine **220** first determines the appropriate key or keys to request from the depository **210** or the depository system **700** for the signing party, and takes actions to provide appropriate matching keys. According to still another embodiment, the authentication engine **215** or the cryptographic engine **220** may advantageously request one or more of the keys associated with the signing party and stored in the depository **210** or depository system **700**.

According to one embodiment, the signing party includes one or both the user and the vendor. In such case, the authentication engine **215** advantageously requests the cryptographic keys corresponding to the user and/or the vendor. According to another embodiment, the signing party includes the trust engine **110**. In this embodiment, the trust engine **110** is certifying that the authentication process **1000** properly authenticated the user, vendor, or both. Therefore, the authentication engine **215** requests the cryptographic key of the trust engine **110**, such as, for example, the key belonging to the cryptographic engine **220**, to perform the digital signature. According to another embodiment, the trust engine **110** performs a digital notary-like function. In this embodiment, the signing party includes the user, vendor, or both, along with the trust engine **110**. Thus, the trust engine **110** provides the digital signature of the user and/or vendor, and then indicates with its own digital signature that the user and/or vendor were properly authenticated. In this embodiment, the authentication engine **215** may advantageously request assembly of the cryptographic keys corresponding to the user, the vendor, or both. According to another embodiment, the authentication engine **215** may advantageously request assembly of the cryptographic keys corresponding to the trust engine **110**.

According to another embodiment, the trust engine **110** performs power of attorney-like functions. For example, the trust engine **110** may digitally sign the message on behalf of a third party. In such case, the authentication engine **215** requests the cryptographic keys associated with the third party. According to this embodiment, the signing process **1100** may advantageously include authentication of the third party, before allowing power of attorney-like functions. In addition, the authentication process **1000** may include a check for third party constraints, such as, for example, business logic or the like dictating when and in what circumstances a particular third-party's signature may be used.

Based on the foregoing, in step **1110**, the authentication engine requested the cryptographic keys from the data storage facilities D**1** through D**4** corresponding to the signing party. In step **1115**, the data storage facilities D**1** through D**4** transmit their respective portions of the cryptographic key corresponding to the signing party to the cryptographic engine **220**. According to one embodiment, the foregoing transmissions include SSL technologies. According to another embodiment, the foregoing transmissions may advantageously be super-encrypted with the public key of the cryptographic engine **220**.

In step **1120**, the cryptographic engine **220** assembles the foregoing cryptographic keys of the signing party and encrypts the message therewith, thereby forming the digital signature(s). In step **1125** of the signing process **1100**, the cryptographic engine **220** transmits the digital signature(s) to the authentication engine **215**. In step **1130**, the authentication engine **215** transmits the filled-in authentication request along with a copy of the hashed message and the digital signature(s) to the transaction engine **205**. In step **1135**, the transaction engine **205** transmits a receipt comprising the transaction ID, an indication of whether the authentication was successful, and the digital signature(s), to the vendor. According to one embodiment, the foregoing transmission may advantageously include the digital signature of the trust engine **110**. For example, the trust engine **110** may encrypt the hash of the receipt with its private key, thereby forming a digital signature to be attached to the transmission to the vendor.

According to one embodiment, the transaction engine **205** also transmits a confirmation message to the user. Although the signing process **1100** is disclosed with reference to its preferred and alternative embodiments, the invention is not intended to be limited thereby. Rather, a skilled artisan will recognize from the disclosure herein, a wide number of alternatives for the signing process **1100**. For example, the vendor may be replaced with a user application, such as an email application. For example, the user may wish to digitally sign a particular email with his or her digital signature. In such an embodiment, the transmission throughout the signing process **1100** may advantageously include only one copy of a hash of the message. Moreover, a skilled

artisan will recognize from the disclosure herein that a wide number of client applications may request digital signatures. For example, the client applications may comprise word processors, spreadsheets, emails, voicemail, access to restricted system areas, or the like.

In addition, a skilled artisan will recognize from the disclosure herein that steps **1105** through **1120** of the signing process **1100** may advantageously employ some or all of the steps of the interoperability process **970** of FIG. **9**B, thereby providing interoperability between differing cryptographic systems that may, for example, need to process the digital signature under differing signature types.

FIG. **12** illustrates a data flow of an encryption/decryption process **1200** according to aspects of an embodiment of the invention. As shown in FIG. **12**, the decryption process **1200** begins by authenticating the user using the authentication process **1000**. According to one embodiment, the authentication process **1000** includes in the authentication request, a synchronous session key. For example, in conventional PKI technologies, it is understood by skilled artisans that encrypting or decrypting data using public and private keys is mathematically intensive and may require significant system resources. However, in symmetric key cryptographic systems, or systems where the sender and receiver of a message share a single common key that is used to encrypt and decrypt a message, the mathematical operations are significantly simpler and faster. Thus, in the conventional PKI technologies, the sender of a message will generate synchronous session key, and encrypt the message using the simpler, faster symmetric key system. Then, the sender will encrypt the session key with the public key of the receiver. The encrypted session key will be attached to the synchronously encrypted message and both data are sent to the receiver. The receiver uses his or her private key to decrypt the session key, and then uses the session key to decrypt the message. Based on the foregoing, the simpler and faster symmetric key system is used for the majority of the encryption/decryption processing. Thus, in the decryption process **1200**, the decryption advantageously assumes that a synchronous key has been encrypted with the public key of the user. Thus, as mentioned in the foregoing, the encrypted session key is included in the authentication request.

Returning to the decryption process **1200**, after the user has been authenticated in step **1205**, the authentication engine **215** forwards the encrypted session key to the cryptographic engine **220**. In step **1210**, the authentication engine **215** forwards a request to each of the data storage facilities, D**1** through D**4**, requesting the cryptographic key data of the user. In step **1215**, each data storage facility, D**1** through D**4**, transmits their respective portion of the cryptographic key to the cryptographic engine **220**. According to one embodiment, the foregoing transmission is encrypted with the public key of the cryptographic engine **220**.

In step **1220** of the decryption process **1200**, the cryptographic engine **220** assembles the cryptographic key and decrypts the session key therewith. In step **1225**, the cryptographic engine forwards the session key to the authentication engine **215**. In step **1227**, the authentication engine **215** fills in the authentication request including the decrypted session key, and transmits the filled-in authentication request to the transaction engine **205**. In step **1230**, the transaction engine **205** forwards the authentication request along with the session key to the requesting application or vendor. Then, according to one embodiment, the requesting application or vendor uses the session key to decrypt the encrypted message.

Although the decryption process **1200** is disclosed with reference to its preferred and alternative embodiments, a skilled artisan will recognize from the disclosure herein, a wide number of alternatives for the decryption process **1200**. For example, the decryption process **1200** may forego synchronous key encryption and rely on full public-key technology. In such an embodiment, the requesting application may transmit the entire message to the cryptographic engine **220**, or, may employ some type of compression or reversible hash in order to transmit the message to the cryptographic engine **220**. A skilled artisan will also recognize from the disclosure herein that the foregoing communications may advantageously include XML documents wrapped in SSL technology.

The encryption/decryption process **1200** also provides for encryption of documents or other data. Thus, in step **1235**, a requesting application or vendor may advantageously transmit to the transaction engine **205** of the trust engine **110**, a request for the public key of the user. The requesting application or vendor makes this request because the requesting application or vendor uses the public key of the user, for example, to encrypt the session key that will be used to encrypt the document or message. As mentioned in the enrollment process **900**, the transaction engine **205** stores a copy of the digital certificate of the user, for example, in the mass storage **225**. Thus, in step **1240** of the encryption process **1200**, the transaction engine **205** requests the digital certificate of the user from the mass storage **225**. In step **1245**, the mass storage **225** transmits the digital certificate corresponding to the user, to the transaction engine **205**. In step **1250**, the transaction engine **205** transmits the digital certificate to the requesting application or vendor. According to one embodiment, the encryption portion of the encryption process **1200** does not include the authentication of a user. This is because the requesting vendor needs only the public key of the user, and is not requesting any sensitive data.

A skilled artisan will recognize from the disclosure herein that if a particular user does not have a digital certificate, the trust engine **110** may employ some or all of the enrollment process **900** in order to generate a digital certificate for that particular user. Then, the trust engine **110** may initiate the encryption/decryption process **1200** and thereby provide the appropriate digital certificate. In addition, a skilled artisan will recognize from the disclosure herein that steps **1220** and **1235** through **1250** of the encryption/decryption process **1200** may advantageously employ some or all of the steps of the interoperability process of FIG. **9**B, thereby providing interoperability between differing cryptographic systems that may, for example, need to process the encryption.

FIG. **13** illustrates a simplified block diagram of a trust engine system **1300** according to aspects of yet another embodiment of the invention. As shown in FIG. **13**, the trust engine system **1300** comprises a plurality of distinct trust engines **1305**, **1310**, **1315**, and **1320**, respectively. To facilitate a more complete understanding of the invention, FIG. **13** illustrates each trust engine, **1305**, **1310**, **1315**, and **1320** as having a transaction engine, a depository, and an authentication engine. However, a skilled artisan will recognize that each transaction engine may advantageously comprise some, a combination, or all of the elements and communication channels disclosed with reference to FIGS. **1**-**8**. For example, one embodiment may advantageously include trust engines having one or more transaction engines, depositories, and cryptographic servers or any combinations thereof.

According to one embodiment of the invention, each of the trust engines **1305**, **1310**, **1315** and **1320** are geographi-

cally separated, such that, for example, the trust engine **1305** may reside in a first location, the trust engine **1310** may reside in a second location, the trust engine **1315** may reside in a third location, and the trust engine **1320** may reside in a fourth location. The foregoing geographic separation advantageously decreases system response time while increasing the security of the overall trust engine system **1300**.

For example, when a user logs onto the cryptographic system **100**, the user may be nearest the first location and may desire to be authenticated. As described with reference to FIG. **10**, to be authenticated, the user provides current authentication data, such as a biometric or the like, and the current authentication data is compared to that user's enrollment authentication data. Therefore, according to one example, the user advantageously provides current authentication data to the geographically nearest trust engine **1305**. The transaction engine **1321** of the trust engine **1305** then forwards the current authentication data to the authentication engine **1322** also residing at the first location. According to another embodiment, the transaction engine **1321** forwards the current authentication data to one or more of the authentication engines of the trust engines **1310**, **1315**, or **1320**.

The transaction engine **1321** also requests the assembly of the enrollment authentication data from the depositories of, for example, each of the trust engines, **1305** through **1320**. According to this embodiment, each depository provides its portion of the enrollment authentication data to the authentication engine **1322** of the trust engine **1305**. The authentication engine **1322** then employs the encrypted data portions from, for example, the first two depositories to respond, and assembles the enrollment authentication data into deciphered form. The authentication engine **1322** compares the enrollment authentication data with the current authentication data and returns an authentication result to the transaction engine **1321** of the trust engine **1305**.

Based on the above, the trust engine system **1300** employs the nearest one of a plurality of geographically separated trust engines, **1305** through **1320**, to perform the authentication process. According to one embodiment of the invention, the routing of information to the nearest transaction engine may advantageously be performed at client-side applets executing on one or more of the user system **105**, vendor system **120**, or certificate authority **115**. According to an alternative embodiment, a more sophisticated decision process may be employed to select from the trust engines **1305** through **1320**. For example, the decision may be based on the availability, operability, speed of connections, load, performance, geographic proximity, or a combination thereof, of a given trust engine.

In this way, the trust engine system **1300** lowers its response time while maintaining the security advantages associated with geographically remote data storage facilities, such as those discussed with reference to FIG. **7** where each data storage facility stores randomized portions of sensitive data. For example, a security compromise at, for example, the depository **1325** of the trust engine **1315** does not necessarily compromise the sensitive data of the trust engine system **1300**. This is because the depository **1325** contains only non-decipherable randomized data that, without more, is entirely useless.

According to another embodiment, the trust engine system **1300** may advantageously include multiple cryptographic engines arranged similar to the authentication engines. The cryptographic engines may advantageously perform cryptographic functions such as those disclosed with reference to FIGS. **1-8**. According to yet another embodiment, the trust engine system **1300** may advantageously replace the multiple authentication engines with multiple cryptographic engines, thereby performing cryptographic functions such as those disclosed with reference to FIGS. **1-8**. According to yet another embodiment of the invention, the trust engine system **1300** may replace each multiple authentication engine with an engine having some or all of the functionality of the authentication engines, cryptographic engines, or both, as disclosed in the foregoing,

Although the trust engine system **1300** is disclosed with reference to its preferred and alternative embodiments, a skilled artisan will recognize that the trust engine system **1300** may comprise portions of trust engines **1305** through **1320**. For example, the trust engine system **1300** may include one or more transaction engines, one or more depositories, one or more authentication engines, or one or more cryptographic engines or combinations thereof.

FIG. **14** illustrates a simplified block diagram of a trust engine System **1400** according to aspects of yet another embodiment of the invention. As shown in FIG. **14**, the trust engine system **1400** includes multiple trust engines **1405**, **1410**, **1415** and **1420**. According to one embodiment, each of the trust engines **1405**, **1410**, **1415** and **1420**, comprise some or all of the elements of trust engine **110** disclosed with reference to FIGS. **1-8**. According to this embodiment, when the client side applets of the user system **105**, the vendor system **120**, or the certificate authority **115**, communicate with the trust engine system **1400**, those communications are sent to the IP address of each of the trust engines **1405** through **1420**. Further, each transaction engine of each of the trust engines, **1405**, **1410**, **1415**, and **1420**, behaves similar to the transaction engine **1321** of the trust engine **1305** disclosed with reference to FIG. **13**. For example, during an authentication process, each transaction engine of each of the trust engines **1405**, **1410**, **1415**, and **1420** transmits the current authentication data to their respective authentication engines and transmits a request to assemble the randomized data stored in each of the depositories of each of the trust engines **1405** through **1420**. FIG. **14** does not illustrate all of these communications; as such illustration would become overly complex. Continuing with the authentication process, each of the depositories then communicates its portion of the randomized data to each of the authentication engines of the each of the trust engines **1405** through **1420**. Each of the authentication engines of the each of the trust engines employs its comparator to determine whether the current authentication data matches the enrollment authentication data provided by the depositories of each of the trust engines **1405** through **1420**. According to this embodiment, the result of the comparison by each of the authentication engines is then transmitted to a redundancy module of the other three trust engines. For example, the result of the authentication engine from the trust engine **1405** is transmitted to the redundancy modules of the trust engines **1410**, **1415**, and **1420**. Thus, the redundancy module of the trust engine **1405** likewise receives the result of the authentication engines from the trust engines **1410**, **1415**, and **1420**.

FIG. **15** illustrates a block diagram of the redundancy module of FIG. **14**. The redundancy module comprises a comparator configured to receive the authentication result from three authentication engines and transmit that result to the transaction engine of the fourth trust engine. The comparator compares the authentication result form the three authentication engines, and if two of the results agree, the comparator concludes that the authentication result should match that of the two agreeing authentication engines. This

result is then transmitted back to the transaction engine corresponding to the trust engine not associated with the three authentication engines.

Based on the foregoing, the redundancy module determines an authentication result from data received from authentication engines that are preferably geographically remote from the trust engine of that the redundancy module. By providing such redundancy functionality, the trust engine system **1400** ensures that a compromise of the authentication engine of one of the trust engines **1405** through **1420**, is insufficient to compromise the authentication result of the redundancy module of that particular trust engine. A skilled artisan will recognize that redundancy module functionality of the trust engine system **1400** may also be applied to the cryptographic engine of each of the trust engines **1405** through **1420**. However, such cryptographic engine communication was not shown in FIG. **14** to avoid complexity. Moreover, a skilled artisan will recognize a wide number of alternative authentication result conflict resolution algorithms for the comparator of FIG. **15** are suitable for use in the present invention.

According to yet another embodiment of the invention, the trust engine system **1400** may advantageously employ the redundancy module during cryptographic comparison steps. For example, some or all of the foregoing redundancy module disclosure with reference to FIGS. **14** and **15** may advantageously be implemented during a hash comparison of documents provided by one or more parties during a particular transaction.

Although the foregoing invention has been described in terms of certain preferred and alternative embodiments, other embodiments will be apparent to those of ordinary skill in the art from the disclosure herein. For example, the trust engine **110** may issue short-term certificates, where the private cryptographic key is released to the user for a predetermined period of time. For example, current certificate standards include a validity field that can be set to expire after a predetermined amount of time. Thus, the trust engine **110** may release a private key to a user where the private key would be valid for, for example, 24 hours. According to such an embodiment, the trust engine **110** may advantageously issue a new cryptographic key pair to be associated with a particular user and then release the private key of the new cryptographic key pair. Then, once the private cryptographic key is released, the trust engine **110** immediately expires any internal valid use of such private key, as it is no longer securable by the trust engine **110**.

In addition, a skilled artisan will recognize that the cryptographic system **100** or the trust engine **110** may include the ability to recognize any type of devices, such as, but not limited to, a laptop, a cell phone, a network, a biometric device or the like. According to one embodiment, such recognition may come from data supplied in the request for a particular service, such as, a request for authentication leading to access or use, a request for cryptographic functionality, or the like. According to one embodiment, the foregoing request may include a unique device identifier, such as, for example, a processor ID. Alternatively, the request may include data in a particular recognizable data format. For example, mobile and satellite phones often do not include the processing power for full X509.v3 heavy encryption certificates, and therefore do not request them. According to this embodiment, the trust engine **110** may recognize the type of data format presented, and respond only in kind.

In an additional aspect of the system described above, context sensitive authentication can be provided using various techniques as will be described below. Context sensitive authentication, for example as shown in FIG. **16**, provides the possibility of evaluating not only the actual data which is sent by the user when attempting to authenticate himself, but also the circumstances surrounding the generation and delivery of that data. Such techniques may also support transaction specific trust arbitrage between the user and trust engine **110** or between the vendor and trust engine **110**, as will be described below.

As discussed above, authentication is the process of proving that a user is who he says he is. Generally, authentication requires demonstrating some fact to an authentication authority. The trust engine **110** of the present invention represents the authority to which a user must authenticate himself. The user must demonstrate to the trust engine **110** that he is who he says he is by either: knowing something that only the user should know (knowledge-based authentication), having something that only the user should have (token-based authentication), or by being something that only the user should be (biometric-based authentication).

Examples of knowledge-based authentication include without limitation a password, PIN number, or lock combination. Examples of token-based authentication include without limitation a house key, a physical credit card, a driver's license, or a particular phone number. Examples of biometric-based authentication include without limitation a fingerprint, handwriting analysis, facial scan, hand scan, ear scan, iris scan, vascular pattern, DNA, a voice analysis, or a retinal scan.

Each type of authentication has particular advantages and disadvantages, and each provides a different level of security. For example, it is generally harder to create a false fingerprint that matches someone else's than it is to overhear someone's password and repeat it. Each type of authentication also requires a different type of data to be known to the authenticating authority in order to verify someone using that form of authentication.

As used herein, "authentication" will refer broadly to the overall process of verifying someone's identity to be who he says he is. An "authentication technique" will refer to a particular type of authentication based upon a particular piece of knowledge, physical token, or biometric reading. "Authentication data" refers to information which is sent to or otherwise demonstrated to an authentication authority in order to establish identity. "Enrollment data" will refer to the data which is initially submitted to an authentication authority in order to establish a baseline for comparison with authentication data. An "authentication instance" will refer to the data associated with an attempt to authenticate by an authentication technique.

The internal protocols and communications involved in the process of authenticating a user is described with reference to FIG. **10** above. The part of this process within which the context sensitive authentication takes place occurs within the comparison step shown as step **1045** of FIG. **10**. This step takes place within the authentication engine **215** and involves assembling the enrollment data **410** retrieved from the depository **210** and comparing the authentication data provided by the user to it. One particular embodiment of this process is shown in FIG. **16** and described below.

The current authentication data provided by the user and the enrollment data retrieved from the depository **210** are received by the authentication engine **215** in step **1600** of FIG. **16**. Both of these sets of data may contain data which is related to separate techniques of authentication. The authentication engine **215** separates the authentication data associated with each individual authentication instance in

step **1605**. This is necessary so that the authentication data is compared with the appropriate subset of the enrollment data for the user (e.g. fingerprint authentication data should be compared with fingerprint enrollment data, rather than password enrollment data).

Generally, authenticating a user involves one or more individual authentication instances, depending on which authentication techniques are available to the user. These methods are limited by the enrollment data which were provided by the user during his enrollment process (if the user did not provide a retinal scan when enrolling, he will not be able to authenticate himself using a retinal scan), as well as the means which may be currently available to the user (e.g. if the user does not have a fingerprint reader at his current location, fingerprint authentication will not be practical). In some cases, a single authentication instance may be sufficient to authenticate a user; however, in certain circumstances a combination of multiple authentication instances may be used in order to more confidently authenticate a user for a particular transaction.

Each authentication instance consists of data related to a particular authentication technique (e.g. fingerprint, password, smart card, etc.) and the circumstances which surround the capture and delivery of the data for that particular technique. For example, a particular instance of attempting to authenticate via password will generate not only the data related to the password itself, but also circumstantial data, known as "metadata", related to that password attempt. This circumstantial data includes information such as: the time at which the particular authentication instance took place, the network address from which the authentication information was delivered, as well as any other information as is known to those of skill in the art which may be determined about the origin of the authentication data (the type of connection, the processor serial number, etc.).

In many cases, only a small amount of circumstantial metadata will be available. For example, if the user is located on a network which uses proxies or network address translation or another technique which masks the address of the originating computer, only the address of the proxy or router may be determined. Similarly, in many cases information such as the processor serial number will not be available because of either limitations of the hardware or operating system being used, disabling of such features by the operator of the system, or other limitations of the connection between the user's system and the trust engine **110**.

As shown in FIG. **16**, once the individual authentication instances represented within the authentication data are extracted and separated in step **1605**, the authentication engine **215** evaluates each instance for its reliability in indicating that the user is who he claims to be. The reliability for a single authentication instance will generally be determined based on several factors. These may be grouped as factors relating to the reliability associated with the authentication technique, which are evaluated in step **1610**, and factors relating to the reliability of the particular authentication data provided, which are evaluated in step **1815**. The first group includes without limitation the inherent reliability of the authentication technique being used, and the reliability of the enrollment data being used with that method. The second group includes without limitation the degree of match between the enrollment data and the data provided with the authentication instance, and the metadata associated with that authentication instance. Each of these factors may vary independently of the others.

The inherent reliability of an authentication technique is based on how hard it is for an imposter to provide someone else's correct data, as well as the overall error rates for the authentication technique. For passwords and knowledge based authentication methods, this reliability is often fairly low because there is nothing that prevents someone from revealing their password to another person and for that second person to use that password. Even a more complex knowledge based system may have only moderate reliability since knowledge may be transferred from person to person fairly easily. Token based authentication, such as having a proper smart card or using a particular terminal to perform the authentication, is similarly of low reliability used by itself, since there is no guarantee that the right person is in possession of the proper token.

However, biometric techniques are more inherently reliable because it is generally difficult to provide someone else with the ability to use your fingerprints in a convenient manner, even intentionally. Because subverting biometric authentication techniques is more difficult, the inherent reliability of biometric methods is generally higher than that of purely knowledge or token based authentication techniques. However, even biometric techniques may have some occasions in which a false acceptance or false rejection is generated. These occurrences may be reflected by differing reliabilities for different implementations of the same biometric technique. For example, a fingerprint matching system provided by one company may provide a higher reliability than one provided by a different company because one uses higher quality optics or a better scanning resolution or some other improvement which reduces the occurrence of false acceptances or false rejections.

Note that this reliability may be expressed in different manners. The reliability is desirably expressed in some metric which can be used by the heuristics **530** and algorithms of the authentication engine **215** to calculate the confidence level of each authentication. One preferred mode of expressing these reliabilities is as a percentage or fraction. For instance, fingerprints might be assigned an inherent reliability of 97%, while passwords might only be assigned an inherent reliability of 50%. Those of skill in the art will recognize that these particular values are merely exemplary and may vary between specific implementations.

The second factor for which reliability must be assessed is the reliability of the enrollment. This is part of the "graded enrollment" process referred to above. This reliability factor reflects the reliability of the identification provided during the initial enrollment process. For instance, if the individual initially enrolls in a manner where they physically produce evidence of their identity to a notary or other public official, and enrollment data is recorded at that time and notarized, the data will be more reliable than data which is provided over a network during enrollment and only vouched for by a digital signature or other information which is not truly tied to the individual.

Other enrollment techniques with varying levels of reliability include without limitation: enrollment at a physical office of the trust engine **110** operator; enrollment at a user's place of employment; enrollment at a post office or passport office; enrollment through an affiliated or trusted party to the trust engine **110** operator; anonymous or pseudonymous enrollment in which the enrolled identity is not yet identified with a particular real individual, as well as such other means as are known in the art.

These factors reflect the trust between the trust engine **110** and the source of identification provided during the enrollment process. For instance, if enrollment is performed in

association with an employer during the initial process of providing evidence of identity, this information may be considered extremely reliable for purposes within the company, but may be trusted to a lesser degree by a government agency, or by a competitor. Therefore, trust engines operated by each of these other organizations may assign different levels of reliability to this enrollment.

Similarly, additional data which is submitted across a network, but which is authenticated by other trusted data provided during a previous enrollment with the same trust engine **110** may be considered as reliable as the original enrollment data was, even though the latter data were submitted across an open network. In such circumstances, a subsequent notarization will effectively increase the level of reliability associated with the original enrollment data. In this way for example, an anonymous or pseudonymous enrollment may then be raised to a full enrollment by demonstrating to some enrollment official the identity of the individual matching the enrolled data.

The reliability factors discussed above are generally values which may be determined in advance of any particular authentication instance. This is because they are based upon the enrollment and the technique, rather than the actual authentication. In one embodiment, the step of generating reliability based upon these factors involves looking up previously determined values for this particular authentication technique and the enrollment data of the user. In a further aspect of an advantageous embodiment of the present invention, such reliabilities may be included with the enrollment data itself. In this way, these factors are automatically delivered to the authentication engine **215** along with the enrollment data sent from the depository **210**.

While these factors may generally be determined in advance of any individual authentication instance, they still have an effect on each authentication instance which uses that particular technique of authentication for that user. Furthermore, although the values may change over time (e.g. if the user re-enrolls in a more reliable fashion), they are not dependent on the authentication data itself. By contrast, the reliability factors associated with a single specific instance's data may vary on each occasion. These factors, as discussed below, must be evaluated for each new authentication in order to generate reliability scores in step **1815**.

The reliability of the authentication data reflects the match between the data provided by the user in a particular authentication instance and the data provided during the authentication enrollment. This is the fundamental question of whether the authentication data matches the enrollment data for the individual the user is claiming to be. Normally, when the data do not match, the user is considered to not be successfully authenticated, and the authentication fails. The manner in which this is evaluated may change depending on the authentication technique used. The comparison of such data is performed by the comparator **515** function of the authentication engine **215** as shown in FIG. **5**.

For instance, matches of passwords are generally evaluated in a binary fashion. In other words, a password is either a perfect match, or a failed match. It is usually not desirable to accept as even a partial match a password which is close to the correct password if it is not exactly correct. Therefore, when evaluating a password authentication, the reliability of the authentication returned by the comparator **515** is typically either 100% (correct) or 0% (wrong), with no possibility of intermediate values.

Similar rules to those for passwords are generally applied to token based authentication methods, such as smart cards. This is because having a smart card which has a similar identifier or which is similar to the correct one, is still just as wrong as having any other incorrect token. Therefore tokens tend also to be binary authenticators: a user either has the right token, or he doesn't.

However, certain types of authentication data, such as questionnaires and biometrics, are generally not binary authenticators. For example, a fingerprint may match a reference fingerprint to varying degrees. To some extent, this may be due to variations in the quality of the data captured either during the initial enrollment or in subsequent authentications. (A fingerprint may be smudged or a person may have a still healing scar or burn on a particular finger.) In other instances the data may match less than perfectly because the information itself is somewhat variable and based upon pattern matching. (A voice analysis may seem close but not quite right because of background noise, or the acoustics of the environment in which the voice is recorded, or because the person has a cold.) Finally, in situations where large amounts of data are being compared, it may simply be the case that much of the data matches well, but some doesn't. (A ten-question questionnaire may have resulted in eight correct answers to personal questions, but two incorrect answers.) For any of these reasons, the match between the enrollment data and the data for a particular authentication instance may be desirably assigned a partial match value by the comparator **515**. In this way, the fingerprint might be said to be a 85% match, the voice print a 65% match, and the questionnaire an 80% match, for example.

This measure (degree of match) produced by the comparator **515** is the factor representing the basic issue of whether an authentication is correct or not. However, as discussed above, this is only one of the factors which may be used in determining the reliability of a given authentication instance. Note also that even though a match to some partial degree may be determined, that ultimately, it may be desirable to provide a binary result based upon a partial match. In an alternate mode of operation, it is also possible to treat partial matches as binary, i.e. either perfect (100%) or failed (0%) matches, based upon whether or not the degree of match passes a particular threshold level of match. Such a process may be used to provide a simple pass/fail level of matching for systems which would otherwise produce partial matches.

Another factor to be considered in evaluating the reliability of a given authentication instance concerns the circumstances under which the authentication data for this particular instance are provided. As discussed above, the circumstances refer to the metadata associated with a particular authentication instance. This may include without limitation such information as: the network address of the authenticator, to the extent that it can be determined; the time of the authentication; the mode of transmission of the authentication data (phone line, cellular, network, etc.); and the serial number of the system of the authenticator.

These factors can be used to produce a profile of the type of authentication that is normally requested by the user. Then, this information can be used to assess reliability in at least two manners. One manner is to consider whether the user is requesting authentication in a manner which is consistent with the normal profile of authentication by this user. If the user normally makes authentication requests from one network address during business days (when she is at work) and from a different network address during evenings or weekends (when she is at home), an authentication which occurs from the home address during the business day is less reliable because it is outside the normal authentication

profile. Similarly, if the user normally authenticates using a fingerprint biometric and in the evenings, an authentication which originates during the day using only a password is less reliable.

An additional way in which the circumstantial metadata can be used to evaluate the reliability of an instance of authentication is to determine how much corroboration the circumstance provides that the authenticator is the individual he claims to be. For instance, if the authentication comes from a system with a serial number known to be associated with the user, this is a good circumstantial indicator that the user is who they claim to be. Conversely, if the authentication is coming from a network address which is known to be in Los Angeles when the user is known to reside in London, this is an indication that this authentication is less reliable based on its circumstances.

It is also possible that a cookie or other electronic data may be placed upon the system being used by a user when they interact with a vendor system or with the trust engine **110**. This data is written to the storage of the system of the user and may contain an identification which may be read by a Web browser or other software on the user system. If this data is allowed to reside on the user system between sessions (a "persistent cookie"), it may be sent with the authentication data as further evidence of the past use of this system during authentication of a particular user. In effect, the metadata of a given instance, particularly a persistent cookie, may form a sort of token based authenticator itself.

Once the appropriate reliability factors based on the technique and data of the authentication instance are generated as described above in steps **1610** and **1615** respectively, they are used to produce an overall reliability for the authentication instance provided in step **1620**. One means of doing this is simply to express each reliability as a percentage and then to multiply them together.

For example, suppose the authentication data is being sent in from a network address known to be the user's home computer completely in accordance with the user's past authentication profile (100%), and the technique being used is fingerprint identification (97%), and the initial finger print data was roistered through the user's employer with the trust engine **110** (90%), and the match between the authentication data and the original fingerprint template in the enrollment data is very good (99%). The overall reliability of this authentication instance could then be calculated as the product of these reliabilities: 100%*97%*90%*99%–86.4% reliability.

This calculated reliability represents the reliability of one single instance of authentication. The overall reliability of a single authentication instance may also be calculated using techniques which treat the different reliability factors differently, for example by using formulas where different weights are assigned to each reliability factor. Furthermore, those of skill in the art will recognize that the actual values used may represent values other than percentages and may use non-arithmetic systems. One embodiment may include a module used by an authentication requestor to set the weights for each factor and the algorithms used in establishing the overall reliability of the authentication instance.

The authentication engine **215** may use the above techniques and variations thereof to determine the reliability of a single authentication instance, indicated as step **1620**. However, it may be useful in many authentication situations for multiple authentication instances to be provided at the same time. For example, while attempting to authenticate himself using the system of the present invention, a user may provide a user identification, fingerprint authentication data, a smart card, and a password. In such a case, three independent authentication instances are being provided to the trust engine **110** for evaluation. Proceeding to step **1625**, if the authentication engine **215** determines that the data provided by the user includes more than one authentication instance, then each instance in turn will be selected as shown in step **1630** and evaluated as described above in steps **1610**, **1615** and **1620**.

Note that many of the reliability factors discussed may vary from one of these instances to another. For instance, the inherent reliability of these techniques is likely to be different, as well as the degree of match provided between the authentication data and the enrollment data. Furthermore, the user may have provided enrollment data at different times and under different circumstances for each of these techniques, providing different enrollment reliabilities for each of these instances as well. Finally, even though the circumstances under which the data for each of these instances is being submitted is the same, the use of such techniques may each fit the profile of the user differently, and so may be assigned different circumstantial reliabilities. (For example, the user may normally use their password and fingerprint, but not their smart card.)

As a result, the final reliability for each of these authentication instances may be different from One another. However, by using multiple instances together, the overall confidence level for the authentication will tend to increase.

Once the authentication engine has performed steps **1610** through **1620** for all of the authentication instances provided in the authentication data, the reliability of each instance is used in step **1635** to evaluate the overall authentication confidence level. This process of combining the individual authentication instance reliabilities into the authentication confidence level may be modeled by various methods relating the individual reliabilities produced, and may also address the particular interaction between some of these authentication techniques. (For example, multiple knowledge-based systems such as passwords may produce less confidence than a single password and even a fairly weak biometric, such as a basic voice analysis.)

One means in which the authentication engine **215** may combine the reliabilities of multiple concurrent authentication instances to generate a final confidence level is to multiply the unreliability of each instance to arrive at a total unreliability. The unreliability is generally the complementary percentage of the reliability. For example, a technique which is 84% reliable is 16% unreliable. The three authentication instances described above (fingerprint, smart card, password) which produce reliabilities of 86%, 75%, and 72% would have corresponding unreliabilities of (100–86)%, (100–75)% and (100–72)%, or 14%, 25%, and 28%, respectively. By multiplying these unreliabilities, we get a cumulative unreliability of 14%*25%*28%–0.98% unreliability, which corresponds to a reliability of 99.02%.

In an additional mode of operation, additional factors and heuristics **530** may be applied within the authentication engine **215** to account for the interdependence of various authentication techniques. For example, if someone has unauthorized access to a particular home computer, they probably have access to the phone line at that address as well. Therefore, authenticating based on an originating phone number as well as upon the serial number of the authenticating system does not add much to the overall confidence in the authentication. However, knowledge based authentication is largely independent of token based authen-

tication (i.e. if someone steals your cellular phone or keys, they are no more likely to know your PIN or password than if they hadn't).

Furthermore, different vendors or other authentication requestors may wish to weigh different aspects of the authentication differently. This may include the use of separate weighing factors or algorithms used in calculating the reliability of individual instances as well as the use of different means to evaluate authentication events with multiple instances.

For instance, vendors for certain types of transactions, for instance corporate email systems, may desire to authenticate primarily based upon heuristics and other circumstantial data by default. Therefore, they may apply high weights to factors related to the metadata and other profile related information associated with the circumstances surrounding authentication events. This arrangement could be used to ease the burden on users during normal operating hours, by not requiring more from the user than that he be logged on to the correct machine during business hours. However, another vendor may weigh authentications coming from a particular technique most heavily, for instance fingerprint matching, because of a policy decision that such a technique is most suited to authentication for the particular vendor's purposes.

Such varying weights may be defined by the authentication requestor in generating the authentication request and sent to the trust engine **110** with the authentication request in one mode of operation. Such options could also be set as preferences during an initial enrollment process for the authentication requestor and stored within the authentication engine in another mode of operation.

Once the authentication engine **215** produces an authentication confidence level for the authentication data provided, this confidence level is used to complete the authentication request in step **1640**, and this information is forwarded from the authentication engine **215** to the transaction engine **205** for inclusion in a message to the authentication requestor.

The process described above is merely exemplary, and those of skill in the art will recognize that the steps need not be performed in the order shown or that only certain of the steps are desired to be performed, or that a variety of combinations of steps may be desired. Furthermore, certain steps, such as the evaluation of the reliability of each authentication instance provided, may be carried out in parallel with one another if circumstances permit.

In a further aspect of this invention, a method is provided to accommodate conditions when the authentication confidence level produced by the process described above fails to meet the required trust level of the vendor or other party requiring the authentication. In circumstances such as these where a gap exists between the level of confidence provided and the level of trust desired, the operator of the trust engine **110** is in a position to provide opportunities for one or both parties to provide alternate data or requirements in order to close this trust gap. This process will be referred to as "trust arbitrage" herein.

Trust arbitrage may take place within a framework of cryptographic authentication as described above with reference to FIGS. **10** and **11**. As shown therein, a vendor or other party will request authentication of a particular user in association with a particular transaction. In one circumstance, the vendor simply requests an authentication, either positive or negative, and after receiving appropriate data from the user, the trust engine **110** will provide such a binary authentication. In circumstances such as these, the degree of confidence required in order to secure a positive authentication is determined based upon preferences set within the trust engine **110**.

However, it is also possible that the vendor may request a particular level of trust in order to complete a particular transaction. This required level may be included with the authentication request (e.g. authenticate this user to 98% confidence) or may be determined by the trust engine **110** based on other factors associated with the transaction (i.e. authenticate this user as appropriate for this transaction). One such factor might be the economic value of the transaction. For transactions which have greater economic value, a higher degree of trust may be required. Similarly, for transactions with high degrees of risk a high degree of trust may be required. Conversely, for transactions which are either of low risk or of low value, lower trust levels may be required by the vendor or other authentication requestor.

The process of trust arbitrage occurs between the steps of the trust engine **110** receiving the authentication data in step **1050** of FIG. **10** and the return of an authentication result to the vendor in step **1055** of FIG. **10**. Between these steps, the process which leads to the evaluation of trust levels and the potential trust arbitrage occurs as shown in FIG. **17**. In circumstances where simple binary authentication is performed, the process shown in FIG. **17** reduces to having the transaction engine **205** directly compare the authentication data provided with the enrollment data for the identified user as discussed above with reference to FIG. **10**, flagging any difference as a negative authentication.

As shown in FIG. **17**, the first step after receiving the data in step **1050** is for the transaction engine **205** to determine the trust level which is required for a positive authentication for this particular transaction in step **1710**. This step may be performed by one of several different methods. The required trust level may be specified to the trust engine **110** by the authentication requestor at the time when the authentication request is made. The authentication requestor may also set a preference in advance which is stored within the depository **210** or other storage which is accessible by the transaction engine **205**. This preference may then be read and used each time an authentication request is made by this authentication requestor. The preference may also be associated with a particular user as a security measure such that a particular level of trust is always required in order to authenticate that user, the user preference being stored in the depository **210** or other storage media accessible by the transaction engine **205**. The required level may also be derived by the transaction engine **205** or authentication engine **215** based upon information provided in the authentication request, such as the value and risk level of the transaction to be authenticated.

In one mode of operation, a policy management module or other software which is used when generating the authentication request is used to specify the required degree of trust for the authentication of the transaction. This may be used to provide a series of rules to follow when assigning the required level of trust based upon the policies which are specified within the policy management module. One advantageous mode of operation is for such a module to be incorporated with the web server of a vendor in order to appropriately determine required level of trust for transactions initiated with the vendor's web server. In this way, transaction requests from users may be assigned a required trust level in accordance with the policies of the vendor and such information may be forwarded to the trust engine **110** along with the authentication request.

This required trust level correlates with the degree of certainty that the vendor wants to have that the individual authenticating is in fact who he identifies himself as. For example, if the transaction is one where the vendor wants a fair degree of certainty because goods are changing hands, the vendor may require a trust level of 85%. For situation where the vendor is merely authenticating the user to allow him to view members only content or exercise privileges on a chat room, the downside risk may be small enough that the vendor requires only a 60% trust level. However, to enter into a production contract with a value of tens of thousands of dollars, the vendor may require a trust level of 99% or more.

This required trust level represents a metric to which the user must authenticate himself in order to complete the transaction. If the required trust level is 85% for example, the user must provide authentication to the trust engine **110** sufficient for the trust engine **110** to say with 85% confidence that the user is who they say they are. It is the balance between this required trust level and the authentication confidence level which produces either a positive authentication (to the satisfaction of the vendor) or a possibility of trust arbitrage.

As shown in FIG. **17**, after the transaction engine **205** receives the required trust level, it compares in step **1720** the required trust level to the authentication confidence level which the authentication engine **215** calculated for the current authentication (as discussed with reference to FIG. **16**). If the authentication confidence level is higher than the required trust level for the transaction in step **1730**, then the process moves to step **1740** where a positive authentication for this transaction is produced by the transaction engine **205**. A message to this effect will then be inserted into the authentication results and returned to the vendor by the transaction engine **205** as shown in step **1055** (see FIG. **10**).

However, if the authentication confidence level does not fulfill the required trust level in step **1730**, then a confidence gap exists for the current authentication, and trust arbitrage is conducted in step **1750**. Trust arbitrage is described more completely with reference to FIG. **18** below. This process as described below takes place within the transaction engine **205** of the trust engine **110**. Because no authentication or other cryptographic operations are needed to execute trust arbitrage (other than those required for the SSL communication between the transaction engine **205** and other components), the process may be performed outside the authentication engine **215**. However, as will be discussed below, any reevaluation of authentication data or other cryptographic or authentication events will require the transaction engine **205** to resubmit the appropriate data to the authentication engine **215**. Those of skill in the art will recognize that the trust arbitrage process could alternately be structured to take place partially or entirely within the authentication engine **215** itself.

As mentioned above, trust arbitrage is a process where the trust engine **110** mediates a negotiation between the vendor and user in an attempt to secure a positive authentication where appropriate. As shown in step **1805**, the transaction engine **205** first determines whether or not the current situation is appropriate for trust arbitrage. This may be determined based upon the circumstances of the authentication, e.g. whether this authentication has already been through multiple cycles of arbitrage, as well as upon the preferences of either the vendor or user, as will be discussed further below.

In such circumstances where arbitrage is not possible, the process proceeds to step **1810** where the transaction engine **205** generates a negative authentication and then inserts it into the authentication results which are sent to the vendor in step **1055** (see FIG. **10**). One limit which may be advantageously used to prevent authentications from pending indefinitely is to set a time-out period from the initial authentication request. In this way, any transaction which is not positively authenticated within the time limit is denied further arbitrage and negatively authenticated. Those of skill in the art will recognize that such a time limit may vary depending upon the circumstances of the transaction and the desires of the user and vendor. Limitations may also be placed upon the number of attempts that may be made at providing a successful authentication. Such limitations may be handled by an attempt limiter **535** as shown in FIG. **5**.

If arbitrage is not prohibited in step **1805**, the transaction engine **205** will then engage in negotiation with one or both of the transacting parties. The transaction engine **205** may send a message to the user requesting some form of additional authentication in order to boost the authentication confidence level produced as shown in step **1820**. In the simplest form, this may simply indicates that authentication was insufficient. A request to produce one or more additional authentication instances to improve the overall confidence level of the authentication may also be sent.

If the user provides some additional authentication instances in step **1825**, then the transaction engine **205** adds these authentication instances to the authentication data for the transaction and forwards it to the authentication engine **215** as shown in step **1015** (see FIG. **10**), and the authentication is reevaluated based upon both the pre-existing authentication instances for this transaction and the newly provided authentication instances.

An additional type of authentication may be a request from the trust engine **110** to make some form of person-to-person contact between the trust engine **110** operator (or a trusted associate) and the user, for example, by phone call. This phone call or other non-computer authentication can be used to provide personal contact with the individual and also to conduct some form of questionnaire based authentication. This also may give the opportunity to verify an originating telephone number and potentially a voice analysis of the user when he calls in. Even if no additional authentication data can be provided, the additional context associated with the user's phone number may improve the reliability of the authentication context. Any revised data or circumstances based upon this phone call are fed into the trust engine **110** for use in consideration of the authentication request.

Additionally, in step **1820** the trust engine **110** may provide an opportunity for the user to purchase insurance, effectively buying a more confident authentication. The operator of the trust engine **110** may, at times, only want to make such an option available if the confidence level of the authentication is above a certain threshold to begin with. In effect, this user side insurance is a way for the trust engine **110** to vouch for the user when the authentication meets the normal required trust level of the trust engine **110** for authentication, but does not meet the required trust level of the vendor for this transaction. In this way, the user may still successfully authenticate to a very high level as may be required by the vendor, even though he only has authentication instances which produce confidence sufficient for the trust engine **110**.

This function of the trust engine **110** allows the trust engine **110** to vouch for someone who is authenticated to the satisfaction of the trust engine **110**, but not of the vendor. This is analogous to the function performed by a notary in adding his signature to a document in order to indicate to

someone reading the document at a later time that the person whose signature appears on the document is in fact the person who signed it. The signature of the notary testifies to the act of signing by the user. In the same way, the trust engine is providing an indication that the person transacting is who they say they are.

However, because the trust engine **110** is artificially boosting the level of confidence provided by the user, there is a greater risk to the trust engine **110** operator, since the user is not actually meeting the required trust level of the vendor. The cost of the insurance is designed to offset the risk of a false positive authentication to the trust engine **110** (who may be effectively notarizing the authentications of the user). The user pays the trust engine **110** operator to take the risk of authenticating to a higher level of confidence than has actually been provided.

Because such an insurance system allows someone to effectively buy a higher confidence rating from the trust engine **110**, both vendors and users may wish to prevent the use of user side insurance in certain transactions. Vendors may wish to limit positive authentications to circumstances where they know that actual authentication data supports the degree of confidence which they require and so may indicate to the trust engine **110** that user side insurance is not to be allowed. Similarly, to protect his online identity, a user may wish to prevent the use of user side insurance on his account, or may wish to limit its use to situations where the authentication confidence level without the insurance is higher than a certain limit. This may be used as a security measure to prevent someone from overhearing a password or stealing a smart card and using them to falsely authenticate to a low level of confidence, and then purchasing insurance to produce a very high level of (false) confidence. These factors may be evaluated in determining whether user side insurance is allowed.

If user purchases insurance in step **1840**, then the authentication confidence level is adjusted based upon the insurance purchased in step **1845**, and the authentication confidence level and required trust level are again compared in step **1730** (see FIG. **17**). The process continues from there, and may lead to either a positive authentication in step **1740** (see FIG. **17**), or back into the trust arbitrage process in step **1750** for either further arbitrage (if allowed) or a negative authentication in step **1810** if further arbitrage is prohibited.

In addition to sending a message to the user in step **1820**, the transaction engine **205** may also send a message to the vendor in step **1830** which indicates that a pending authentication is currently below the required trust level. The message may also offer various options on how to proceed to the vendor. One of these Options is to simply inform the vendor of what the current authentication confidence level is and ask if the vendor wishes to maintain their current unfulfilled required trust level. This may be beneficial because in some cases, the vendor may have independent means for authenticating the transaction or may have been using a default set of requirements which generally result in a higher required level being initially specified than is actually needed for the particular transaction at hand.

For instance, it may be standard practice that all incoming purchase order transactions with the vendor are expected to meet a 98% trust level. However, if an order was recently discussed by phone between the vendor and a long-standing customer, and immediately thereafter the transaction is authenticated, but only to a 93% confidence level, the vendor may wish to simply lower the acceptance threshold for this transaction, because the phone call effectively provides additional authentication to the vendor. In certain circumstances, the vendor may be willing to lower their required trust level, but not all the way to the level of the current authentication confidence. For instance, the vendor in the above example might consider that the phone call prior to the order might merit a 4% reduction in the degree of trust needed; however, this is still greater than the 93% confidence produced by the user.

If the vendor does adjust their required trust level in step **1835**, then the authentication confidence level produced by the authentication and the required trust level are compared in step **1730** (see FIG. **17**). If the confidence level now exceeds the required trust level, a positive authentication may be generated in the transaction engine **205** in step **1740** (see FIG. **17**). If not, further arbitrage may be attempted as discussed above if it is permitted.

In addition to requesting an adjustment to the required trust level, the transaction engine **205** may also offer vendor side insurance to the vendor requesting the authentication. This insurance serves a similar purpose to that described above for the user side insurance. Here, however, rather than the cost corresponding to the risk being taken by the trust engine **110** in authenticating above the actual authentication confidence level produced, the cost of the insurance corresponds to the risk being taken by the vendor in accepting a lower trust level in the authentication.

Instead of just lowering their actual required trust level, the vendor has the option of purchasing insurance to protect itself from the additional risk associated with a lower level of trust in the authentication of the user. As described above, it may be advantageous for the vendor to only consider purchasing such insurance to cover the trust gap in conditions where the existing authentication is already above a certain threshold.

The availability of such vendor side insurance allows the vendor the option to either: lower his trust requirement directly at no additional cost to himself, bearing the risk of a false authentication himself (based on the lower trust level required); or, buying insurance for the trust gap between the authentication confidence level and his requirement, with the trust engine **110** operator bearing the risk of the lower confidence level which has been provided. By purchasing the insurance, the vendor effectively keeps his high trust level requirement; because the risk of a false authentication is shifted to the trust engine **110** operator.

If the vendor purchases insurance in step **1840**, the authentication confidence level and required trust levels are compared in step **1730** (see FIG. **17**), and the process continues as described above.

Note that it is also possible that both the user and the vendor respond to messages from the trust engine **110**. Those of skill in the art will recognize that there are multiple ways in which such situations can be handled. One advantageous mode of handling the possibility of multiple responses is simply to treat the responses in a first-come, first-served manner. For example, if the vendor responds with a lowered required trust level and immediately thereafter the user also purchases insurance to raise his authentication level, the authentication is first reevaluated based upon the lowered trust requirement from the vendor. If the authentication is now positive, the user's insurance purchase is ignored. In another advantageous mode of operation, the user might only be charged for the level of insurance required to meet the new, lowered trust requirement of the vendor (if a trust gap remained even with the lowered vendor trust requirement).

If no response from either party is received during the trust arbitrage process at step **1850** within the time limit set

for the authentication, the arbitrage is reevaluated in step **1805**. This effectively begins the arbitrage process again. If the time limit was final or other circumstances prevent further arbitrage in step **1805**, a negative authentication is generated by the transaction engine **205** in step **1810** and returned to the vendor in step **1055** (see FIG. **10**). If not, new messages may be sent to the user and vendor, and the process may be repeated as desired.

Note that for certain types of transactions, for instance, digitally signing documents which are not part of a transaction, there may not necessarily be a vendor or other third party; therefore the transaction is primarily between the user and the trust engine **110**. In circumstances such as these, the trust engine **110** will have its own required trust level which must be satisfied in order to generate a positive authentication. However, in such circumstances, it will often not be desirable for the trust engine **110** to offer insurance to the user in order for him to raise the confidence of his own signature.

The process described above and shown in FIGS. **16-18** may be carried out using various communications modes as described above with reference to the trust engine **110**. For instance, the messages may be web-based and sent using SSL connections between the trust engine **110** and applets downloaded in real time to browsers running on the user or vendor systems. In an alternate mode of operation, certain dedicated applications may be in use by the user and vendor which facilitate such arbitrage and insurance transactions. In another alternate mode of operation, secure email operations may be used to mediate the arbitrage described above, thereby allowing deferred evaluations and batch processing of authentications. Those of skill in the art will recognize that different communications modes may be used as are appropriate for the circumstances and authentication requirements of the vendor.

The following description with reference to FIG. **19** describes a sample transaction which integrates the various aspects of the present invention as described above. This example illustrates the overall process between a user and a vendor as mediates by the trust engine **110**. Although the various steps and components as described in detail above may be used to carry out the following transaction, the process illustrated focuses on the interaction between the trust engine **110**, user and vendor.

The transaction begins when the user, while viewing web pages online, fills out an order form on the web site of the vendor in step **1900**. The user wishes to submit this order form to the vendor, signed with his digital signature. In order to do this, the user submits the order form with his request for a signature to the trust engine **110** in step **1905**. The user will also provide authentication data which will be used as described above to authenticate his identity.

In step **1910** the authentication data is compared to the enrollment data by the trust engine **110** as discussed above, and if a positive authentication is produced, the hash of the order form, signed with the private key of the user, is forwarded to the vendor along with the order form itself.

The vendor receives the signed form in step **1915**, and then the vendor will generate an invoice or other contract related to the purchase to be made in step **1920**. This contract is sent back to the user with a request for a signature in step **1925**. The vendor also sends an authentication request for this contract transaction to the trust engine **110** in step **1930** including a hash of the contract which will be signed by both parties. To allow the contract to be digitally signed by both parties, the vendor also includes authentication data for itself so that the vendor's signature upon the contract can later be verified if necessary.

As discussed above, the trust engine **110** then verifies the authentication data provided by the vendor to confirm the vendor's identity, and if the data produces a positive authentication in step **1935**, continues with step **1955** when the data is received from the user. If the vendor's authentication data does not match the enrollment data of the vendor to the desired degree, a message is returned to the vendor requesting further authentication. Trust arbitrage may be performed here if necessary, as described above, in order for the vendor to successfully authenticate itself to the trust engine **110**.

When the user receives the contract in step **1940**, he reviews it, generates authentication data to sign it if it is acceptable in step **1945**, and then sends a hash of the contract and his authentication data to the trust engine **110** in step **1950**. The trust engine **110** verifies the authentication data in step **1955** and if the authentication is good, proceeds to process the contract as described below. As discussed above with reference to FIGS. **17** and **18**, trust arbitrage may be performed as appropriate to close any trust gap which exists between the authentication confidence level and the required authentication level for the transaction.

The trust engine **110** signs the hash of the contract with the user's private key, and sends this signed hash to the vendor in step **1960**, signing the complete message on its own behalf, i.e., including a hash of the complete message (including the user's signature) encrypted with the private key **510** of the trust engine **110**. This message is received by the vendor in step **1965**. The message represents a signed contract (hash of contract encrypted using user's private key) and a receipt from the trust engine **110** (the hash of the message including the signed contract, encrypted using the trust engine **110**'s private key).

The trust engine **110** similarly prepares a hash of the contract with the vendor's private key in step **1970**, and forwards this to the user, signed by the trust engine **110**. In this way, the user also receives a copy of the contract, signed by the vendor, as well as a receipt, signed by the trust engine **110**, for delivery of the signed contract in step **1975**.

In addition to the foregoing, an additional aspect of the invention provides a cryptographic Service Provider Module (SPM) which may be available to a client side application as a means to access functions provided by the trust engine **110** described above. One advantageous way to provide such a service is for the cryptographic SPM is to mediate communications between a third party Application Programming Interface (API) and a trust engine **110** which is accessible via a network or other remote connection. A sample cryptographic SPM is described below with reference to FIG. **20**.

For example, on a typical system, a number of API's are available to programmers. Each API provides a set of function calls which may be made by an application **2000** running upon the system. Examples of API's which provide programming interfaces suitable for cryptographic functions, authentication functions, and other security function include the Cryptographic API (CAPI) **2010** provided by Microsoft with its Windows operating systems, and the Common Data Security Architecture (CDSA), sponsored by IBM, Intel and other members of the Open Group. CAPI will be used as an exemplary security API in the discussion that follows. However, the cryptographic SPM described could be used with CDSA or other security API's as are known in the art.

This API is used by a user system **105** or vendor system **120** when a call is made for a cryptographic function.

Included among these functions may be requests associated with performing various cryptographic operations, such as encrypting a document with a particular key, signing a document, requesting a digital certificate, verifying a signature upon a signed document, and such other cryptographic functions as are described herein or known to those of skill in the art.

Such cryptographic functions are normally performed locally to the system upon which CAPI **2010** is located. This is because generally the functions called require the use of either resources of the local user system **105**, such as a fingerprint reader, or software functions which are programmed using libraries which are executed on the local machine. Access to these local resources is normally provided by one or more Service Provider Modules (SPM's) **2015**, **2020** as referred to above which provide resources with which the cryptographic functions are carried out. Such SPM's may include software libraries **2015** to perform encrypting or decrypting operations, or drivers and applications **2020** which are capable of accessing specialized hardware **2025**, such as biometric scanning devices. In much the way that CAPI **2010** provides functions which may be used by an application **2000** of the system **105**, the SPM's **2015**, **2020** provide CAPI with access to the lower level functions and resources associated with the available services upon the system.

In accordance with the invention, it is possible to provide a cryptographic SPM **2030** which is capable of accessing the cryptographic functions provided by the trust engine **110** and making these functions available to an application **2000** through CAPI **2010**. Unlike embodiments where CAPI **2010** is only able to access resources which are locally available through SPM's **2015**, **2020**, a cryptographic SPM **2030** as described herein would be able to submit requests for cryptographic operations to a remotely-located, network-accessible trust engine **110** in order to perform the operations desired.

For instance, if an application **2000** has a need for a cryptographic operation, such as signing a document, the application **2000** makes a function call to the appropriate CAPI **2010** function. CAPI **2010** in turn will execute this function, making use of the resources which are made available to it by the SPM's **2015**, **2020** and the cryptographic SPM **2030**. In the case of a digital signature function, the cryptographic SPM **2030** will generate an appropriate request which will be sent to the trust engine **110** across the communication link **125**.

The operations which occur between the cryptographic SPM **2030** and the trust engine **110** are the same operations that would be possible between any other system and the trust engine **110**. However, these functions are effectively made available to a user system **105** through CAPI **2010** such that they appear to be locally available upon the user system **105** itself. However, unlike ordinary SPM's **2015**, **2020**, the functions are being carried out on the remote trust engine **110** and the results relayed to the cryptographic SPM **2030** in response to appropriate requests across the communication link **125**.

This cryptographic SPM **2030** makes a number of operations available to the user system **105** or a vendor system **120** which might not otherwise be available. These functions include without limitation: encryption and decryption of documents; issuance of digital certificates; digital signing of documents; verification of digital signatures; and such other operations as will be apparent to those of skill in the art.

In a separate embodiment, the present invention comprises a complete system for performing the data securing methods of the present invention on any data set. The computer system of this embodiment comprises a data splitting module that comprises the functionality shown in FIG. **8** and described herein. In one embodiment of the present invention, the data splitting module, sometimes referred to herein as a secure data parser, comprises a parser program or software suite which comprises data splitting, encryption and decryption, reconstitution or reassembly functionality. This embodiment may further comprise a data storage facility or multiple data storage facilities, as well. The data splitting module, or secure data parser, comprises a cross-platform software module suite which integrates within an electronic infrastructure, or as an add-on to any application which requires the ultimate security of its data elements. This parsing process operates on any type of data set, and on any and all file types, or in a database on any row, column or cell of data in that database.

The parsing process of the present invention may, in one embodiment, be designed in a modular tiered fashion, and any encryption process is suitable for use in the process of the present invention. The modular tiers of the parsing and splitting process of the present invention may include, but are not limited to, 1) cryptographic split, dispersed and securely stored in multiple locations; 2) encrypt, cryptographically split, dispersed and securely stored in multiple locations; 3) encrypt, cryptographically split, encrypt each share, then dispersed and securely stored in multiple locations; and 4) encrypt, cryptographically split, encrypt each share with a different type of encryption than was used in the first step, then dispersed and securely stored in multiple locations.

The process comprises, in one embodiment, splitting of the data according to the contents of a generated random number, or key and performing the same cryptographic splitting of the key used in the encryption of splitting of the data to be secured into two or more portions, or shares, of parsed and split data, and in one embodiment, preferably into four or more portions of parsed and split data, encrypting all of the portions, then scattering and storing these portions back into the database, or relocating them to any named device, fixed or removable, depending on the requestor's need for privacy and security. Alternatively, in another embodiment, encryption may occur prior to the splitting of the data set by the splitting module or secure data parser. The original data processed as described in this embodiment is encrypted and obfuscated and is secured. The dispersion of the encrypted elements, if desired, can be virtually anywhere, including, but not limited to, a single server or data storage device, or among separate data storage facilities or devices. Encryption key management in one embodiment may be included within the software suite, or in another embodiment may be integrated into an existing infrastructure or any other desired location.

A cryptographic split (cryptosplit) partitions the data into N number of shares. The partitioning can be on any size unit of data, including an individual bit, bits, bytes, kilobytes, megabytes, or larger units, as well as any pattern or combination of data unit sizes whether predetermined or randomly generated. The units can also be of different sized, based on either a random or predetermined set of values. This means the data can be viewed as a sequence of these units. In this manner the size of the data units themselves may render the data more secure, for example by using one or more predetermined or randomly generated pattern, sequence or combination of data unit sizes. The units are then distributed (either randomly or by a predetermined set of values) into the N shares. This distribution could also

involve a shuffling of the order of the units in the shares. It is readily apparent to those of ordinary skill in the art that the distribution of the data units into the shares may be performed according to a wide variety of possible selections, including but not limited to size-fixed, predetermined sizes, or one or more combination, pattern or sequence of data unit sizes that are predetermined or randomly generated.

One example of this cryptographic split process, or cryptosplit, would be to consider the data to be 23 bytes in size, with the data unit size chosen to be one byte, and with the number of shares selected to be 4. Each byte would be distributed into one of the 4 shares. Assuming a random distribution, a key would be obtained to create a sequence of 23 random numbers (r**1**, r**2**, r**3** through r**23**), each with a value between 1 and 4 corresponding to the four shares. Each of the units of data (in this example 23 individual bytes of data) is associated with one of the 23 random numbers corresponding to one of the four shares. The distribution of the bytes of data into the four shares would occur by placing the first byte of the data into share number r**1**, byte two into share r**2**, byte three into share r**3**, through the 23rd byte of data into share r**23**. It is readily apparent to those of ordinary skill in the art that a wide variety of other possible steps or combination or sequence of steps, including the size of the data units, may be used in the cryptosplit process of the present invention, and the above example is a non-limiting description of one process for cryptosplitting data. To recreate the original data, the reverse operation would be performed.

In another embodiment of the cryptosplit process of the present invention, an option for the cryptosplitting process is to provide sufficient redundancy in the shares such that only a subset of the shares are needed to reassemble or restore the data to its original or usable form. As a non-limiting example, the cryptosplit may be done as a "3 of 4" cryptosplit such that only three of the four shares are necessary to reassemble or restore the data to its original or usable form. This is also referred to as a "M of N cryptosplit" wherein N is the total number of shares, and M is at least one less than N. It is readily apparent to those of ordinary skill in the art that there are many possibilities for creating this redundancy in the cryptosplitting process of the present invention.

In one embodiment of the cryptosplitting process of the present invention, each unit of data is stored in two shares, the primary share and the backup share. Using the "3 of 4" cryptosplitting process described above, any one share can be missing, and this is sufficient to reassemble or restore the original data with no missing data units since only three of the total four shares are required. As described herein, a random number is generated that corresponds to one of the shares. The random number is associated with a data unit, and stored in the corresponding share, based on a key. One key is used, in this embodiment, to generate the primary and backup share random number. As described herein for the cryptosplitting process of the present invention, a set of random numbers (also referred to as primary share numbers) from 0 to 3 are generated equal to the number of data units. Then another set of random numbers is generated (also referred to as backup share numbers) from 1 to 3 equal to the number of data units. Each unit of data is then associated with a primary share number and a backup share number. Alternatively, a set of random numbers may be generated that is fewer than the number of data units, and repeating the random number set, but this may reduce the security of the sensitive data. The primary share number is used to determine into which share the data unit is stored. The backup share number is combined with the primary share number to create a third share number between 0 and 3, and this number is used to determine into which share the data unit is stored. In this example, the equation to determine the third share number is: (primary share number+backup share number) MOD 4=third share number.

In the embodiment described above where the primary share number is between 0 and 3, and the backup share number is between 1 and 3 ensures that the third share number is different from the primary share number. This results in the data unit being stored in two different shares. It is readily apparent to those of ordinary skill in the art that there are many ways of performing redundant cryptosplitting and non-redundant cryptosplitting in addition to the embodiments disclosed herein. For example, the data units in each share could be shuffled utilizing a different algorithm. This data unit shuffling may be performed as the original data is split into the data units, or after the data units are placed into the shares, or after the share is full, for example.

The various cryptosplitting processes and data shuffling processes described herein, and all other embodiments of the cryptosplitting and data shuffling methods of the present invention may be performed on data units of any size, including but not limited to, as small as an individual bit, bits, bytes, kilobytes, megabytes or larger.

An example of one embodiment of source code that would perform the cryptosplitting process described herein is:

```
DATA [1:24] - array of bytes with the data to be split
SHARES[0:3; 1:24] - 2-dimensionalarray with each row representing one
of the shares
RANDOM[1:24] - array random numbers in the range of 0..3
S1 = 1;
S2 = 1;
S3 = 1;
S4 = 1;
For J = 1 to 24 do
    Begin
    IF RANDOM[J[ ==0 then
        Begin
        SHARES[1,S1] = DATA [J];
        S1 = S1 + 1;
        End
    ELSE IF RANDOM[J[ ==1 then
        Begin
        SHARES[2,S2] = DATA [J];
        S2 = S2 + 1;
        END
    ELSE IF RANDOM[J[ ==2 then
        Begin
        Shares[3,S3] = data [J];
        S3 = S3 + 1;
        End
    Else begin
        Shares[4,S4] = data [J];
        S4 = S4 + 1;
        End;
    END;
```

An example of one embodiment of source code that would perform the cryptosplitting RAID process described herein is:

Generate two sets of numbers, PrimaryShare is 0 to 3, BackupShare is 1 to 3. Then put each data unit into share [primaryshare[1]] and shareRprimaryshare[1]+backupshare [1]) mod 4, with the same process as in cryptosplitting described above. This method will be scalable to any size N, where only N−1 shares are necessary to restore the data.

The retrieval, recombining, reassembly or reconstituting of the encrypted data elements may utilize any number of authentication techniques, including, but not limited to, biometrics, such as fingerprint recognition, facial scan, hand scan, iris scan, retinal scan, ear scan, vascular pattern recognition or DNA analysis. The data splitting and/or parser modules of the present invention may be integrated into a wide variety of infrastructure products or applications as desired.

Traditional encryption technologies known in the art rely on one or more key used to encrypt the data and render it unusable without the key. The data, however, remains whole and intact and subject to attack. The secure data parser of the present invention, in one embodiment, addresses this problem by performing a cryptographic parsing and splitting of the encrypted file into two or more portions or shares, and in another embodiment, preferably four or more shares, adding another layer of encryption to each share of the data, then storing the shares in different physical and/or logical locations. When one or more data shares are physically removed from the system, either by using a removable device, such as a data storage device, or by placing the share under another party's control, any possibility of compromise of secured data is effectively removed.

An example of one embodiment of the secure data parser of the present invention and an example of how it may be utilized is shown in FIG. **21** and described below. However, it is readily apparent to those of ordinary skill in the art that the secure data parser of the present invention may be utilized in a wide variety of ways in addition to the non-limiting example below. As a deployment option, and in one embodiment, the secure data parser may be implemented with external session key management or secure internal storage of session keys. Upon implementation, a Parser Master Key will be generated which will be used for securing the application and for encryption purposes. It should be also noted that the incorporation of the Parser Master key in the resulting secured data allows for a flexibility of sharing of secured data by individuals within a workgroup, enterprise or extended audience.

As shown in FIG. **21**, this embodiment of the present invention shows the steps of the process performed by the secure data parser on data to store the session master key with the parsed data:

1. Generating a session master key and encrypt the data using RS1 stream cipher.

2. Separating the resulting encrypted data into four shares or portions of parsed data according to the pattern of the session master key.

3. In this embodiment of the method, the session master key will be stored along with the secured data shares in a data depository. Separating the session master key according to the pattern of the Parser Master Key and append the key data to the encrypted parsed data.

4. The resulting four shares of data will contain encrypted portions of the original data and portions of the session master key. Generate a stream cipher key for each of the four data shares.

5. Encrypting each share, then store the encryption keys in different locations from the encrypted data portions or shares: Share 1 gets Key 4, Share 2 gets Key 1, Share 3 gets Key 2, Share 4 gets Key 3.

To restore the original data format, the steps are reversed.

It is readily apparent to those of ordinary skill in the art that certain steps of the methods described herein may be performed in different order, or repeated multiple times, as desired. It is also readily apparent to those skilled in the art that the portions of the data may be handled differently from one another. For example, multiple parsing steps may be performed on only one portion of the parsed data. Each portion of parsed data may be uniquely secured in any desirable way provided only that the data may be reassembled, reconstituted, reformed, decrypted or restored to its original or other usable form.

As shown in FIG. **22** and described herein, another embodiment of the present invention comprises the steps of the process performed by the secure data parser on data to store the session master key data in one or more separate key management table:

1. Generating a session master key and encrypt the data using RS1 stream cipher.

2. Separating the resulting encrypted data into four shares or portions of parsed data according to the pattern of the session master key.

3. In this embodiment of the method of the present invention, the session master key will be stored in a separate key management table in a data depository. Generating a unique transaction ID for this transaction. Storing the transaction ID and session master key in a separate key management table. Separating the transaction ID according to the pattern of the Parser Master Key and append the data to the encrypted parsed or separated data.

4. The resulting four shares of data will contain encrypted portions of the original data and portions of the transaction ID.

5. Generating a stream cipher key for each of the four data shares.

6. Encrypting each share, then store the encryption keys in different locations from the encrypted data portions or shares: Share 1 gets Key 4, Share 2 gets Key 1, Share 3 gets Key 2, Share 4 gets Key 3.

To restore the original data format, the steps are reversed.

It is readily apparent to those of ordinary skill in the art that certain steps of the method described herein may be performed in different order, or repeated multiple times, as desired. It is also readily apparent to those skilled in the art that the portions of the data may be handled differently from one another. For example, multiple separating or parsing steps may be performed on only one portion of the parsed data. Each portion of parsed data may be uniquely secured in any desirable way provided only that the data may be reassembled, reconstituted, reformed, decrypted or restored to its original or other usable form.

As shown in FIG. **23**, this embodiment of the present invention shows the steps of the process performed by the secure data parser on data to store the session master key with the parsed data:

1. Accessing the parser master key associated with the authenticated user

2. Generating a unique Session Master key

3. Derive an Intermediary Key from an exclusive OR function of the Parser Master Key and Session Master key

4. Optional encryption of the data using an existing or new encryption algorithm keyed with the Intermediary Key.

5. Separating the resulting optionally encrypted data into four shares or portions of parsed data according to the pattern of the Intermediary key.

6. In this embodiment of the method, the session master key will be stored along with the secured data shares in a data depository. Separating the session master key according to the pattern of the Parser Master Key and append the key data to the optionally encrypted parsed data shares.

7. The resulting multiple shares of data will contain optionally encrypted portions of the original data and portions of the session master key.

8. Optionally generate an encryption key for each of the four data shares.

9. Optionally encrypting each share with an existing or new encryption algorithm, then store the encryption keys in different locations from the encrypted data portions or shares: for example, Share 1 gets Key 4, Share 2 gets Key 1, Share 3 gets Key 2, Share 4 gets Key 3.

To restore the original data format, the steps are reversed.

It is readily apparent to those of ordinary skill in the art that certain steps of the methods described herein may be performed in different order, or repeated multiple times, as desired. It is also readily apparent to those skilled in the art that the portions of the data may be handled differently from one another. For example, multiple parsing steps may be performed on only one portion of the parsed data. Each portion of parsed data may be uniquely secured in any desirable way provided only that the data may be reassembled, reconstituted, reformed, decrypted or restored to its original or other usable form.

As shown in FIG. **24** and described herein, another embodiment of the present invention comprises the steps of the process performed by the secure data parser on data to store the session master key data in one or more separate key management table:

1. Accessing the Parser Master Key associated with the authenticated user

2. Generating a unique Session Master Key

3. Derive an Intermediary Key from an exclusive OR function of the Parser Master Key and Session Master key

4. Optionally encrypt the data using an existing or new encryption algorithm keyed with the Intermediary Key.

5. Separating the resulting optionally encrypted data into four shares or portions of parsed data according to the pattern of the Intermediary Key.

6. In this embodiment of the method of the present invention, the session master key will be stored in a separate key management table in a data depository. Generating a unique transaction ID for this transaction. Storing the transaction ID and session master key in a separate key management table or passing the Session Master Key and transaction ID back to the calling program for external management. Separating the transaction ID according to the pattern of the Parser Master Key and append the data to the optionally encrypted parsed or separated data.

7. The resulting four shares of data will contain optionally encrypted portions of the original data and portions of the transaction ID.

8. Optionally generate an encryption key for each of the four data shares.

9. Optionally encrypting each share, then store the encryption keys in different locations from the encrypted data portions or shares. For example: Share 1 gets Key 4, Share 2 gets Key 1, Share 3 gets Key 2, Share 4 gets Key 3.

To restore the original data format, the steps are reversed.

It is readily apparent to those of ordinary skill in the art that certain steps of the method described herein may be performed in different order, or repeated multiple times, as desired. It is also readily apparent to those skilled in the art that the portions of the data may be handled differently from one another. For example, multiple separating or parsing steps may be performed on only one portion of the parsed data. Each portion of parsed data may be uniquely secured in any desirable way provided only that the data may be reassembled, reconstituted, reformed, decrypted or restored to its original or other usable form.

A wide variety of encryption methodologies are suitable for use in the methods of the present invention, as is readily apparent to those skilled in the art. The One Time Pad algorithm, is often considered one of the most secure encryption methods, and is suitable for use in the method of the present invention. Using the One Time Pad algorithm requires that a key be generated which is as long as the data to be secured. The use of this method may be less desirable in certain circumstances such as those resulting in the generation and management of very long keys because of the size of the data set to be secured. In the One-Time Pad (OTP) algorithm, the simple exclusive-or function, XOR, is used. For two binary streams x and y of the same length, x XOR y means the bitwise exclusive-or of x and y.

At the bit level is generated:

0 XOR 0=0
0 XOR 1=1
1 XOR 0=1
1 XOR 1=0

An example of this process is described herein for an n-byte secret, s, (or data set) to be split. The process will generate an n-byte random value, a, and then set:

b=a XOR s.

Note that one can derive "s" via the equation:

s=a XOR b.

The values a and b are referred to as shares or portions and are placed in separate depositories. Once the secret s is split into two or more shares, it is discarded in a secure manner.

The secure data parser of the present invention may utilize this function, performing multiple XOR functions incorporating multiple distinct secret key values: K1, K2, K3, Kn, K5. At the beginning of the operation, the data to be secured is passed through the first encryption operation, secure data=data XOR secret key 5:

S=D XOR K5

In order to securely store the resulting encrypted data in, for example, four shares, S1, S2, S3, Sn, the data is parsed and split into "n" segments, or shares, according to the value of K5. This operation results in "n" pseudorandom shares of the original encrypted data. Subsequent XOR functions may then be performed on each share with the remaining secret key values, for example: Secure data segment 1=encrypted data share 1 XOR secret key 1:

SD1=S1 XOR K1
SD2=S2 XOR K2
SD3=S3 XOR K3
SDn=Sn XOR Kn.

In one embodiment, it may not be desired to have any one depository contain enough information to decrypt the information held there, so the key required to decrypt the share is stored in a different data depository:

Depository 1: SD1, Kn
Depository 2: SD2, K1
Depository 3: SD3, K2
Depository n: SDn, K3.

Additionally, appended to each share may be the information required to retrieve the original session encryption key, K5. Therefore, in the key management example described herein, the original session master key is referenced by a transaction ID split into "n" shares according to the contents of the installation dependant Parser Master Key (TID1, TID2, TID3, TIDn):

Depository 1: SD1, Kn, TID1
Depository 2: SD2, K1, TID2
Depository 3: SD3, K2, TID3
Depository n: SDn, K3, TIDn.

In the incorporated session key example described herein, the session master key is split into "n" shares according to the contents of the installation dependant Parser Master Key (SK1, SK2, SK3, SKn):

Depository 1: SD1, Kn, SK1
Depository 2: SD2, K1, SK2
Depository 3: SD3, K2, SK3
Depository n: SDn, K3, SKn.

Unless all four shares are retrieved, the data cannot be reassembled according to this example. Even if all four shares are captured, there is no possibility of reassembling or restoring the original information without access to the session master key and the Parser Master Key.

This example has described an embodiment of the method of the present invention, and also describes, in another embodiment, the algorithm used to place shares into depositories so that shares from all depositories can be combined to form the secret authentication material. The computations needed are very simple and fast. However, with the One Time Pad (OTP) algorithm there may be circumstances that cause it to be less desirable, such as a large data set to be secured, because the key size is the same size as the data to be stored. Therefore, there would be a need to store and transmit about twice the amount of the original data which may be less desirable under certain circumstances.

Stream Cipher RS1

The stream cipher RS1 splitting technique is very similar to the OTP splitting technique described herein. Instead of an n-byte random value, an n'=min(n, 16)-byte random value is generated and used to key the RS1 Stream Cipher algorithm. The advantage of the RS1 Stream Cipher algorithm is that a pseudorandom key is generated from a much smaller seed number. The speed of execution of the RS1 Stream Cipher encryption is also rated at approximately 10 times the speed of the well known in the art Triple DES encryption without compromising security. The RS1 Stream Cipher algorithm is well known in the art, and may be used to generate the keys used in the XOR function. The RS1 Stream Cipher algorithm is interoperable with other commercially available stream cipher algorithms, such as the RC4™ stream cipher algorithm of RSA Security, Inc and is suitable for use in the methods of the present invention.

Using the key notation above, K1 thru K5 are now an n' byte random values and we set:

SD1=S1 XOR E(K1)
SD2=S2 XOR E(K2)
SD3=S3 XOR E(K3)
SDn=Sn XOR E(Kn)

where E(K1) thru E(Kn) are the first n' bytes of output from the RS1 Stream Cipher algorithm keyed by K1 thru Kn. The shares are now placed into data depositories as described herein.

In this stream cipher RS1 algorithm, the required computations needed are nearly as simple and fast as the OTP algorithm. The benefit in this example using the RS1 Stream Cipher is that the system needs to store and transmit on average only about 16 bytes more than the size of the original data to be secured per share. When the size of the original data is more than 16 bytes, this RS1 algorithm is more efficient than the OTP algorithm because it is simply shorter. It is readily apparent to those of ordinary skill in the art that a wide variety of encryption methods or algorithms are suitable for use in the present invention, including, but not limited to RS1, OTP, RC4™, Triple DES and AES.

There are major advantages provided by the data security methods and computer systems of the present invention over traditional encryption methods. One advantage is the security gained from moving shares of the data to different locations on one or more data depositories or storage devices, that may be in different logical, physical or geographical locations. When the shares of data are split physically and under the control of different personnel, for example, the possibility of compromising the data is greatly reduced.

Another advantage provided by the methods and system of the present invention is the combination of the steps of the method of the present invention for securing data to provide a comprehensive process of maintaining security of sensitive data. The data is encrypted with a secure key and split into one or more shares, and in one embodiment, four shares, according to the secure key. The secure key is stored safely with a reference pointer which is secured into four shares according to a secure key. The data shares are then encrypted individually and the keys are stored safely with different encrypted shares. When combined, the entire process for securing data according to the methods disclosed herein becomes a comprehensive package for data security.

The data secured according to the methods of the present invention is readily retrievable and restored, reconstituted, reassembled, decrypted, or otherwise returned into its original or other suitable form for use. In order to restore the original data, the following items may be utilized:

1. All shares or portions of the data set.
2. Knowledge of and ability to reproduce the process flow of the method used to secure the data.
3. Access to the session master key.
4. Access to the Parser Master Key.

Therefore, it may be desirable to plan a secure installation wherein at least one of the above elements may be physically separated from the remaining components of the system (under the control of a different system administrator for example).

Protection against a rogue application invoking the data securing methods application may be enforced by use of the Parser Master Key. A mutual authentication handshake between the secure data parser and the application may be required in this embodiment of the present invention prior to any action taken.

The security of the system dictates that there be no "backdoor" method for recreation of the original data. For installations where data recovery issues may arise, the secure data parser can be enhanced to provide a mirror of the four shares and session master key depository. Hardware options such as RAID (redundant array of inexpensive disks, used to spread information over several disks) and software options such as replication can assist as well in the data recovery planning.

Key Management

In one embodiment of the present invention, the data securing method uses three sets of keys for an encryption operation. Each set of keys may have individual key storage, retrieval, security and recovery options, based on the installation. The keys that may be used, include, but are not limited to:

The Parser Master Key

This key is an individual key associated with the installation of the secure data parser. It is installed on the server on which the secure data parser has been deployed. There are a variety of options suitable for securing this key including, but not limited to, a smart card, separate hardware key store, standard key stores, custom key stores or within a secured database table, for example.

The Session Master Key

A Session Master Key may be generated each time data is secured. The Session Master Key is used to encrypt the data prior to the parsing and splitting operations. It may also be incorporated (if the Session Master Key is not integrated into the parsed data) as a means of parsing the encrypted data. The Session Master Key may be secured in a variety of manners, including, but not limited to, a standard key store, custom key store, separate database table, or secured within the encrypted shares, for example.

The Share Encryption Keys

For each share or portions of a data set that is created, an individual Share Encryption Key may be generated to further encrypt the shares. The Share Encryption Keys may be stored in different shares than the share that was encrypted.

It is readily apparent to those of ordinary skill in the art that the data securing methods and computer system of the present invention are widely applicable to any type of data in any setting or environment. In addition to commercial applications conducted over the Internet or between customers and vendors, the data securing methods and computer systems of the present invention are highly applicable to non-commercial or private settings or environments. Any data set that is desired to be kept secure from any unauthorized user may be secured using the methods and systems described herein. For example, access to a particular database within a company or organization may be advantageously restricted to only selected users by employing the methods and systems of the present invention for securing data. Another example is the generation, modification or access to documents wherein it is desired to restrict access or prevent unauthorized or accidental access or disclosure outside a group of selected individuals, computers or workstations. These and other examples of the ways in which the methods and systems of data securing of the present invention are applicable to any non-commercial or commercial environment or setting for any setting, including, but not limited to any organization, government agency or corporation.

In another embodiment of the present invention, the data securing method uses three sets of keys for an encryption operation. Each set of keys may have individual key storage, retrieval, security and recovery options, based on the installation. The keys that may be used, include, but are not limited to:

1. The Parser Master Key

This key is an individual key associated with the installation of the secure data parser. It is installed on the server on which the secure data parser has been deployed. There are a variety of options suitable for securing this key including, but not limited to, a smart card, separate hardware key store, standard key stores, custom key stores or within a secured database table, for example.

2. The Session Master Key

A Session Master Key may be generated each time data is secured. The Session Master Key is used in conjunction with the Parser Master key to derive the Intermediary Key. The Session Master Key may be secured in a variety of manners, including, but not limited to, a standard key store, custom key store, separate database table, or secured within the encrypted shares, for example.

3. The Intermediary Key

An Intermediary Key may be generated each time data is secured. The Intermediary Key is used to encrypt the data prior to the parsing and splitting operation. It may also be incorporated as a means of parsing the encrypted data.

4. The Share Encryption Keys

For each share or portions of a data set that is created, an individual Share Encryption Key may be generated to further encrypt the shares. The Share Encryption Keys may be stored in different shares than the share that was encrypted.

It is readily apparent to those of ordinary skill in the art that the data securing methods and computer system of the present invention are widely applicable to any type of data in any setting or environment. In addition to commercial applications conducted over the Internet or between customers and vendors, the data securing methods and computer systems of the present invention are highly applicable to non-commercial or private settings or environments. Any data set that is desired to be kept secure from any unauthorized user may be secured using the methods and systems described herein. For example, access to a particular database within a company or organization may be advantageously restricted to only selected users by employing the methods and systems of the present invention for securing data. Another example is the generation, modification or access to documents wherein it is desired to restrict access or prevent unauthorized or accidental access or disclosure outside a group of selected individuals, computers or workstations. These and other examples of the ways in which the methods and systems of data securing of the present invention are applicable to any non-commercial or commercial environment or setting for any setting, including, but not limited to any organization, government agency or corporation.

Workgroup, Project, Individual PC/Laptop or Cross Platform Data Security

The data securing methods and computer systems of the present invention are also useful in securing data by workgroup, project, individual PC/Laptop and any other platform that is in use in, for example, businesses, offices, government agencies, or any setting in which sensitive data is created, handled or stored. The present invention provides methods and computer systems to secure data that is known to be sought after by organizations, such as the U.S. Government, for implementation across the entire government organization or between governments at a state or federal level.

The data securing methods and computer systems of the present invention provide the ability to not only parse and split flat files but also data fields, sets and or table of any type. Additionally, all forms of data are capable of being secured under this process, including, but not limited to, text, video, images, biometrics and voice data. Scalability, speed and data throughput of the methods of securing data of the present invention are only limited to the hardware the user has at their disposal.

In one embodiment of the present invention, the data securing methods are utilized as described below in a workgroup environment. In one embodiment, as shown in FIG. **23** and described below, the Workgroup Scale data securing method of the present invention uses the private key management functionality of the TrustEngine to store the user/group relationships and the associated private keys (Parser Group Master Keys) necessary for a group of users to share secure data. The method of the present invention has the capability to secure data for an enterprise, workgroup, or individual user, depending on how the Parser Master Key was deployed.

In one embodiment, additional key management and user/group management programs may be provided, enabling wide scale workgroup implementation with a single point of administration and key management. Key generation, management and revocation are handled by the single maintenance program, which all become especially

important as the number of users increase. In another embodiment, key management may also be set up across one or several different system administrators, which may not allow any one person or group to control data as needed. This allows for the management of secured data to be obtained by roles, responsibilities, membership, rights, etc., as defined by an organization, and the access to secured data can be limited to just those who are permitted or required to have access only to the portion they are working on, while others, such as managers or executives, may have access to all of the secured data. This embodiment allows for the sharing of secured data among different groups within a company or organization while at the same time only allowing certain selected individuals, such as those with the authorized and predetermined roles and responsibilities, to observe the data as a whole. In addition, this embodiment of the methods and systems of the present invention also allows for the sharing of data among, for example, separate companies, or separate departments or divisions of companies, or any separate organization departments, groups, agencies, or offices, or the like, of any government or organization or any kind, where some sharing is required, but not any one party may be permitted to have access to all the data. Particularly apparent examples of the need and utility for such a method and system of the present invention are to allow sharing, but maintain security, in between government areas, agencies and offices, and between different divisions, departments or offices of a large company, or any other organization, for example.

An example of the applicability of the methods of the present invention on a smaller scale is as follows. A Parser Master key is used as a serialization or branding of the secure data parser to an organization. As the scale of use of the Parser Master key is reduced from the whole enterprise to a smaller workgroup, the data securing methods described herein are used to share files within groups of users.

In the example shown in FIG. **25** and described below, there are six users defined along with their title or role within the organization. The side bar represents five possible groups that the users can belong to according to their role. The arrow represents membership by the user in one or more of the groups.

When configuring the secure data parser for use in this example, the system administrator accesses the user and group information from the operating system by a maintenance program. This maintenance program generates and assigns Parser Group Master Keys to users based on their membership in groups.

In this example, there are three members in the Senior Staff group. For this group, the actions would be:

1. Access Parser Group Master Key for the Senior Staff group (generate a key if not available);
2. Generate a digital certificate associating CEO with the Senior Staff group;
3. Generate a digital certificate associating CEO with the Senior Staff group;
4. Generate a digital certificate associating Vice President, Marketing with the Senior Staff group.

The same set of actions would be done for each group, and each member within each group. When the maintenance program is complete, the Parser Group Master Key becomes a shared credential for each member of the group. Revocation of the assigned digital certificate may be done automatically when a user is removed from a group through the maintenance program without affecting the remaining members of the group.

Once the shared credentials have been defined, the parsing and splitting process remains the same. When a file, document or data element is to be secured, the user is prompted for the target group to be used when securing the data. The resulting secured data is only accessible by other members of the target group. This functionality of the methods and systems of the present invention may be used with any other computer system or software platform, any may be, for example, integrated into existing application programs or used standalone for file security.

It is readily apparent to those of ordinary skill in the art that any one or combination of encryption algorithms are suitable for use in the methods and systems of the present invention. For example, the encryption steps may, in one embodiment, be repeated to produce a multi-layered encryption scheme. In addition, a different encryption algorithm, or combination of encryption algorithms, may be used in repeat encryption steps such that different encryption algorithms are applied to the different layers of the multi-layered encryption scheme. As such, the encryption scheme itself may become a component of the methods of the present invention for securing sensitive data from unauthorized use or access.

The secure data parser may include as an internal component, as an external component, or as both an error-checking component. For example, in one suitable approach, as portions of data are created using the secure data parser in accordance with the present invention, to assure the integrity of the data within a portion, a hash value is taken at preset intervals within the portion and is appended to the end of the interval. The hash value is a predictable and reproducible numeric representation of the data. If any bit within the data changes, the hash value would be different. A scanning module (either as a stand-alone component external to the secure data parser or as an internal component) may then scan the portions of data generated by the secure data parser. Each portion of data (or alternatively, less than all portions of data according to some interval or by a random or pseudo-random sampling) is compared to the appended hash value or values and an action may be taken. This action may include a report of values that match and do not match, an alert for values that do not match, or invoking of some external or internal program to trigger a recovery of the data. For example, recovery of the data could be performed by invoking a recovery module based on the concept that fewer than all portions may be needed to generate original data in accordance with the present invention.

Any other suitable integrity checking may be implemented using any suitable integrity information appended anywhere in all or a subset of data portions. Integrity information may include any suitable information that can be used to determine the integrity of data portions. Examples of integrity information may include hash values computed based on any suitable parameter (e.g., based on respective data portions), digital signature information, message authentication code (MAC) information, any other suitable information, or any combination thereof.

The secure data parser of the present invention may be used in any suitable application. Namely, the secure data parser described herein has a variety of applications in different areas of computing and technology. Several such areas are discussed below. It will be understood that these are merely illustrative in nature and that any other suitable applications may make use of the secure data parser. It will further be understood that the examples described are merely illustrative embodiments that may be modified in any suitable way in order to satisfy any suitable desires. For

example, parsing and splitting may be based on any suitable units, such as by bits, by bytes, by kilobytes, by megabytes, by any combination thereof, or by any other suitable unit.

The secure data parser of the present invention may be used to implement secure physical tokens, whereby data stored in a physical token may be required in order to access additional data stored in another storage area. In one suitable approach, a physical token, such as a compact USB flash drive, a floppy disk, an optical disk, a smart card, or any other suitable physical token, may be used to store one of at least two portions of parsed data in accordance with the present invention. In order to access the original data, the USB flash drive would need to be accessed. Thus, a personal computer holding one portion of parsed data would need to have the USB flash drive, having the other portion of parsed data, attached before the original data can be accessed. FIG. **26** illustrates this application. Storage area **2500** includes a portion of parsed data **2502**. Physical token **2504**, having a portion of parsed data **2506** would need to be coupled to storage area **2500** using any suitable communications interface **2508** (e.g., USB, serial, parallel, Bluetooth, IR, IEEE 1394, ethernet, or any other suitable communications interface) in order to access the original data. This is useful in a situation where, for example, sensitive data on a computer is left alone and subject to unauthorized access attempts. By removing the physical token (e.g., the USB flash drive), the sensitive data is inaccessible. It will be understood that any other suitable approach for using physical tokens may be used.

The secure data parser of the present invention may be used to implement a secure authentication system whereby user enrollment data (e.g., passwords, private encryption keys, fingerprint templates, biometric data or any other suitable user enrollment data) is parsed and split using the secure data parser. The user enrollment data may be parsed and split whereby one or more portions are stored on a smart card, a government Common Access Card, any suitable physical storage device (e.g., magnetic or optical disk, USB key drive, etc.), or any other suitable device. One or more other portions of the parsed user enrollment data may be stored in the system performing the authentication. This provides an added level of security to the authentication process (e.g., in addition to the biometric authentication information obtained from the biometric source, the user enrollment data must also be obtained via the appropriate parsed and split data portion).

The secure data parser of the present invention may be integrated into any suitable existing system in order to provide the use of its functionality in each system's respective environment. FIG. **27** shows a block diagram of an illustrative system **2600**, which may include software, hardware, or both for implementing any suitable application. System **2600** may be an existing system in which secure data parser **2602** may be retrofitted as an integrated component. Alternatively, secure data parser **2602** may be integrated into any suitable system **2600** from, for example, its earliest design stage. Secure data parser **2600** may be integrated at any suitable level of system **2600**. For example, secure data parser **2602** may be integrated into system **2600** at a sufficiently back-end level such that the presence of secure data parser **2602** may be substantially transparent to an end user of system **2600**. Secure data parser **2602** may be used for parsing and splitting data among one or more storage devices **2604** in accordance with the present invention. Some illustrative examples of systems having the secure data parser integrated therein are discussed below.

The secure data parser of the present invention may be integrated into an operating system kernel (e.g., Linux, Unix, or any other suitable commercial or proprietary operating system). This integration may be used to protect data at the device level whereby, for example, data that would ordinarily be stored in one or more devices is separated into a certain number of portions by the secure data parser integrated into the operating system and stored among the one or more devices. When original data is attempted to be accessed, the appropriate software, also integrated into the operating system, may recombine the parsed data portions into the original data in a way that may be transparent to the end user.

The secure data parser of the present invention may be integrated into a volume manager or any other suitable component of a storage system to protect local and networked data storage across any or all supported platforms. For example, with the secure data parser integrated, a storage system may make use of the redundancy offered by the secure data parser (i.e., which is used to implement the feature of needing fewer than all separated portions of data in order to reconstruct the original data) to protect against data loss. The secure data parser also allows all data written to storage devices, whether using redundancy or not, to be in the form of multiple portions that are generated according to the parsing of the present invention. When original data is attempted to be accessed, the appropriate software, also integrated into the volume manager or other suitable component of the storage system, may recombine the parsed data portions into the original data in a way that may be transparent to the end user.

In one suitable approach, the secure data parser of the present invention may be integrated into a RAID controller (as either hardware or software). This allows for the secure storage of data to multiple drives while maintaining fault tolerance in case of drive failure.

The secure data parser of the present invention may be integrated into a database in order to, for example, protect sensitive table information. For example, in one suitable approach, data associated with particular cells of a database table (e.g., individual cells, one or more particular columns, one or more particular rows, any combination thereof, or an entire database table) may be parsed and separated according to the present invention (e.g., where the different portions are stored on one or more storage devices at one or more locations or on a single storage device). Access to recombine the portions in order to view the original data may be granted by traditional authentication methods (e.g., username and password query).

The secure parser of the present invention may be integrated in any suitable system that involves data in motion (i.e., transfer of data from one location to another). Such systems include, for example, email, streaming data broadcasts, and wireless (e.g., WiFi) communications. With respect to email, in one suitable approach, the secure parser may be used to parse outgoing messages (i.e., containing text, binary data, or both (e.g., files attached to an email message)) and sending the different portions of the parsed data along different paths thus creating multiple streams of data. If any one of these streams of data is compromised, the original message remains secure because the system may require that more than one of the portions be combined, in accordance with the present invention, in order to generate the original data. In another suitable approach, the different portions of data may be communicated along one path sequentially so that if one portion is obtained, it may not be sufficient to generate the original data. The different portions

arrive at the intended recipient's location and may be combined to generate the original data in accordance with the present invention.

FIGS. **27** and **28** are illustrative block diagrams of such email systems. FIG. **28** shows a sender system **2700**, which may include any suitable hardware, such as a computer terminal, personal computer, handheld device (e.g., PDA, Blackberry), cellular telephone, computer network, any other suitable hardware, or any combination thereof. Sender system **2700** is used to generate and/or store a message **2704**, which may be, for example, an email message, a binary data file (e.g., graphics, voice, video, etc.), or both. Message **2704** is parsed and split by secure data parser **2702** in accordance with the present invention. The resultant data portions may be communicated across one or more separate communications paths **2706** over network **2708** (e.g., the Internet, an intranet, a LAN, WiFi, Bluetooth, any other suitable hard-wired or wireless communications means, or any combination thereof) to recipient system **2710**. The data portions may be communicated parallel in time or alternatively, according to any suitable time delay between the communication of the different data portions. Recipient system **2710** may be any suitable hardware as described above with respect to sender system **2700**. The separate data portions carried along communications paths **2706** are recombined at recipient system **2710** to generate the original message or data in accordance with the present invention.

FIG. **28** shows a sender system **2800**, which may include any suitable hardware, such as a computer terminal, personal computer, handheld device (e.g., PDA), cellular telephone, computer network, any other suitable hardware, or any combination thereof. Sender system **2800** is used to generate and/or store a message **2804**, which may be, for example, an email message, a binary data file (e.g., graphics, voice, video, etc.), or both. Message **2804** is parsed and split by secure data parser **2802** in accordance with the present invention. The resultant data portions may be communicated across a single communications paths **2806** over network **2808** (e.g., the Internet, an intranet, a LAN, WiFi, Bluetooth, any other suitable communications means, or any combination thereof) to recipient system **2810**. The data portions may be communicated serially across communications path **2806** with respect to one another. Recipient system **2810** may be any suitable hardware as described above with respect to sender system **2800**. The separate data portions carried along communications path **2806** are recombined at recipient system **2810** to generate the original message or data in accordance with the present invention.

It will be understood that the arrangement of FIGS. **27** and **28** are merely illustrative. Any other suitable arrangement may be used. For example, in another suitable approach, the systems of FIGS. **27** and **28** may be incorporated whereby the multi-path approach of FIG. **28** is used and in which one or more of communications paths **2706** are used to carry more than one portion of data as communications path **2806** does in the context of FIG. **29**. In general

The secure data parser may be integrated at any suitable level of a data-in motion system. For example, in the context of an email system, the secure parser may be integrated at the user-interface level (e.g., into Microsoft® Outlook) in which case, the user may have control over the use of the secure data parser features when using email. Alternatively, the secure parser may be implemented in a back-end component such as at the exchange server, in which case, messages may be automatically parsed, split, and communicated along different paths in accordance with the present invention without any user intervention.

Similarly, in the case of streaming broadcasts of data (e.g., audio, video), the outgoing data may be parsed and separated into multiple streams each containing a portion of the parsed data. The multiple streams may be transmitted along one or more paths and recombined at the recipient's location in accordance with the present invention. One of the benefits of this approach is that it avoids the relatively large overhead associated with traditional encryption of data followed by transmission of the encrypted data over a single communications channel. The secure parser of the present invention allows data in motion to be sent in multiple parallel streams, increasing speed and efficiency.

It will be understand that the secure data parser may be integrated for protection of and fault tolerance of any type of data in motion through any transport medium, including, for example, wired, wireless, or physical. For example, voice over Internet protocol (VoIP) applications may make use of the secure data parser of the present invention. Wireless or wired data transport from or to any suitable personal digital assistant (PDA) devices such as Blackberries and SmartPhones may be secured using the secure data parser of the present invention. Communications using wireless 802.11 protocols for peer to peer and hub based wireless networks, satellite communications, point to point wireless communications, Internet client/server communications, or any other suitable communications may involve the data in motion capabilities of the secure data parser in accordance with the present invention. Data communication between computer peripheral device (e.g., printer, scanner, monitor, keyboard, network router, biometric authentication device (e.g., fingerprint scanner), or any other suitable peripheral device) between a computer and a computer peripheral device, between a computer peripheral device and any other suitable device, or any combination thereof may make use of the data in motion features of the present invention.

The data in motion features of the present invention may also apply to physical transportation of secure shares using for example, separate routes, vehicles, methods, any other suitable physical transportation, or any combination thereof. For example, physical transportation of data may take place on digital/magnetic tapes, floppy disks, optical disks, physical tokens, USB drives, removable hard drives, consumer electronic devices with flash memory (e.g., Apple IPODs or other MP3 players), flash memory, any other suitable medium used for transporting data, or any combination thereof.

The secure data parser of the present invention may provide security with the ability for disaster recovery. According to the present invention, fewer than all portions of the separated data generated by the secure data parser may be necessary in order to retrieve the original data. That is, out of m portions stored, n may be the minimum number of these m portions necessary to retrieve the original data, where $n<=m$. For example, if each of four portions is stored in a different physical location relative to the other three portions, then, if $n=2$ in this example, two of the locations may be compromised whereby data is destroyed or inaccessible, and the original data may still be retrieved from the portions in the other two locations. Any suitable value for n or m may be used.

In addition, the n of m feature of the present invention may be used to create a "two man rule" whereby in order to avoid entrusting a single individual or any other entity with full access to what may be sensitive data, two or more distinct entities, each with a portion of the separated data

parsed by the secure parser of the present invention may need to agree to put their portions together in order to retrieve the original data.

The secure data parser of the present invention may be used to provide a group of entities with a group-wide key that allows the group members to access particular information authorized to be accessed by that particular group. The group key may be one of the data portions generated by the secure parser in accordance with the present invention that may be required to be combined with another portion centrally stored, for example in order to retrieve the information sought. This feature allows for, for example, secure collaboration among a group. It may be applied in for example, dedicated networks, virtual private networks, intranets, or any other suitable network.

Specific applications of this use of the secure parser include, for example, coalition information sharing in which, for example, multi-national friendly government forces are given the capability to communicate operational and otherwise sensitive data on a security level authorized to each respective country over a single network or a dual network (i.e., as compared to the many networks involving relatively substantial manual processes currently used). This capability is also applicable for companies or other organizations in which information needed to be known by one or more specific individuals (within the organization or without) may be communicated over a single network without the need to worry about unauthorized individuals viewing the information.

Another specific application includes a multi-level security hierarchy for government systems. That is, the secure parser of the present invention may provide for the ability to operate a government system at different levels of classified information (e.g., unclassified, classified, secret, top secret) using a single network. If desired, more networks may be used (e.g., a separate network for top secret), but the present invention allows for substantially fewer than current arrangement in which a separate network is used for each level of classification.

It will be understood that any combination of the above described applications of the secure parser of the present invention may be used. For example, the group key application can be used together with the data in motion security application (i.e., whereby data that is communicated over a network can only be accessed by a member of the respective group and where, while the data is in motion, it is split among multiple paths (or sent in sequential portions) in accordance with the present invention).

The secure data parser of the present invention may be integrated into any middleware application to enable applications to securely store data to different database products or to different devices without modification to either the applications or the database. Middleware is a general term for any product that allows two separate and already existing programs to communicate. For example, in one suitable approach, middleware having the secure data parser integrated, may be used to allow programs written for a particular database to communicate with other databases without custom coding.

The secure data parser of the present invention may be implemented having any combination of any suitable capabilities, such as those discussed herein. In some embodiments of the present invention, for example, the secure data parser may be implemented having only certain capabilities whereas other capabilities may be obtained through the use of external software, hardware, or both interfaced either directly or indirectly with the secure data parser.

FIG. **30**, for example, shows an illustrative implementation of the secure data parser as secure data parser **3000**. Secure data parser **3000** is implemented with very few built-in capabilities. As illustrated, secure data parser **3000** may include built-in capabilities for parsing and splitting data into portions (also referred to herein as shares) of data using module **3002** in accordance with the present invention. Secure data parser **3000** may also include built in capabilities for performing redundancy in order to be able to implement, for example, the m of n feature described above (i.e., recreating the original data using fewer than all shares of parsed and split data) using module **3004**. Secure data parser **3000** may also include share distribution capabilities using module **3006** for placing the shares of data into buffers from which they are sent for communication to a remote location, for storage, etc. in accordance with the present invention. It will be understood that any other suitable capabilities may be built into secure data parser **3000**.

Assembled data buffer **3008** may be any suitable memory used to store the original data (although not necessarily in its original form) that will be parsed and split by secure data parser **3000**. In a splitting operation, assembled data buffer **3008** provides input to secure data parser **3008**. In a restore operation, assembled data buffer **3008** may be used to store the output of secure data parser **3000**.

Split shares buffers **3010** may be one or more memory modules that may be used to store the multiple shares of data that resulted from the parsing and splitting of original data. In a splitting operation, split shares buffers **3010** hold the output of the secure data parser. In a restore operation, split shares buffers hold the input to secure data parser **3000**.

It will be understood that any other suitable arrangement of capabilities may be built-in for secure data parser **3000**. Any additional features may be built-in and any of the features illustrated may be removed, made more robust, made less robust, or may otherwise be modified in any suitable way. Buffers **3008** and **3010** are likewise merely illustrative and may be modified, removed, or added to in any suitable way.

Any suitable modules implemented in software, hardware or both may be called by or may call to secure data parser **3000**. If desired, even capabilities that are built into secure data parser **3000** may be replaced by one or more external modules. As illustrated, some external modules include random number generator **3012**, cipher feedback key generator **3014**, hash algorithm **3016**, any one or more types of encryption **3018**, and key management **3020**. It will be understood that these are merely illustrative external modules. Any other suitable modules may be used in addition to or in place of those illustrated.

Cipher feedback key generator **3014** may, externally to secure data parser **3000**, generate for each secure data parser operation, a unique key, or random number (using for example random number generator **3012**), to be used as a seed value for an operation that extends an original session key size (e.g., a value of 128, 256, 512, or 1024 bits) into a value equal to the length of the data to be parsed and split. Any suitable algorithm may be used for the cipher feedback key generation, including, for example, the AES cipher feedback key generation algorithm.

In order to facilitate integration of secure data parser **3000** and its external modules (i.e., secure data parser layer **3026**) into an application layer **3024** (e.g., email application, database application, etc.), a wrapping layer that may make use of, for example, API function calls may be used. Any

other suitable arrangement for facilitating integration of secure data parser layer **3026** into application layer **3024** may be used.

FIG. **31** illustratively shows how the arrangement of FIG. **30** may be used when a write (e.g., to a storage device), insert (e.g., in a database field), or transmit (e.g., across a network) command is issued in application layer **3024**. At step **3100** data to be secured is identified and a call is made to the secure data parser. The call is passed through wrapper layer **3022** where at step **3102**, wrapper layer **3022** streams the input data identified at step **3100** into assembled data buffer **3008**. Also at step **3102**, any suitable share information, filenames, any other suitable information, or any combination thereof may be stored (e.g., as information **3106** at wrapper layer **3022**). Secure data processor **3000** then parses and splits the data it takes as input from assembled data buffer **3008** in accordance with the present invention. It outputs the data shares into split shares buffers **3010**. At step **3104**, wrapper layer **3022** obtains from stored information **3106** any suitable share information (i.e., stored by wrapper **3022** at step **3102**) and share location(s) (e.g., from one or more configuration files). Wrapper layer **3022** then writes the output shares (obtained from split shares buffers **3010**) appropriately (e.g., written to one or more storage devices, communicated onto a network, etc.).

FIG. **32** illustratively shows how the arrangement of FIG. **30** may be used when a read (e.g., from a storage device), select (e.g., from a database field), or receive (e.g., from a network) occurs. At step **3200**, data to be restored is identified and a call to secure data parser **3000** is made from application layer **3024**. At step **3202**, from wrapper layer **3022**, any suitable share information is obtained and share location is determined. Wrapper layer **3022** loads the portions of data identified at step **3200** into split shares buffers **3010**. Secure data parser **3000** then processes these shares in accordance with the present invention (e.g., if only three of four shares are available, then the redundancy capabilities of secure data parser **3000** may be used to restore the original data using only the three shares). The restored data is then stored in assembled data buffer **3008**. At step **3204**, application layer **3022** converts the data stored in assembled data buffer **3008** into its original data format (if necessary) and provides the original data in its original format to application layer **3024**.

It will be understood that the parsing and splitting of original data illustrated in FIG. **31** and the restoring of portions of data into original data illustrated in FIG. **32** is merely illustrative. Any other suitable processes, components, or both may be used in addition to or in place of those illustrated.

FIG. **33** is a block diagram of an illustrative process flow for parsing and splitting original data into two or more portions of data in accordance with one embodiment of the present invention. As illustrated, the original data desired to be parsed and split is plain text **3306** (i.e., the word "SUMMIT" is used as an example). It will be understood that any other type of data may be parsed and split in accordance with the present invention. A session key **3300** is generated. If the length of session key **3300** is not compatible with the length of original data **3306**, then cipher feedback session key **3304** may be generated.

In one suitable approach, original data **3306** may be encrypted prior to parsing, splitting, or both. For example, as FIG. **33** illustrates, original data **3306** may be XORed with any suitable value (e.g., with cipher feedback session key **3304**, or with any other suitable value). It will be understood that any other suitable encryption technique may be used in place of or in addition to the XOR technique illustrate. It will further be understood that although FIG. **33** is illustrated in terms of byte by byte operations, the operation may take place at the bit level or at any other suitable level. It will further be understood that, if desired, there need not be any encryption whatsoever of original data **3306**.

The resultant encrypted data (or original data if no encryption took place) is then hashed to determine how to split the encrypted (or original) data among the output buckets (e.g., of which there are four in the illustrated example). In the illustrated example, the hashing takes place by bytes and is a function of cipher feedback session key **3304**. It will be understood that this is merely illustrative. The hashing may be performed at the bit level, if desired. The hashing may be a function of any other suitable value besides cipher feedback session key **3304**. In another suitable approach, hashing need not be used. Rather, any other suitable technique for splitting data may be employed.

FIG. **34** is a block diagram of an illustrative process flow for restoring original data **3306** from two or more parsed and split portions of original data **3306** in accordance with one embodiment of the present invention. The process involves hashing the portions in reverse (i.e., to the process of FIG. **33**) as a function of cipher feedback session key **3304** to restore the encrypted original data (or original data if there was no encryption prior to the parsing and splitting). The encryption key may then be used to restore the original data (i.e., in the illustrated example, cipher feedback session key **3304** is used to decrypt the XOR encryption by XORing it with the encrypted data). This the restores original data **3306**.

FIG. **35** shows how bit-splitting may be implemented in the example of FIGS. **33** and **34**. A hash may be used (e.g., as a function of the cipher feedback session key, as a function of any other suitable value) to determine a bit value at which to split each byte of data. It will be understood that this is merely one illustrative way in which to implement splitting at the bit level. Any other suitable technique may be used.

It will be understood that any reference to hash functionality made herein may be made with respect to any suitable hash algorithm. These include for example, MD5 and SHA-1. Different hash algorithms may be used at different times and by different components of the present invention.

After a split point has been determined in accordance with the above illustrative procedure or through any other procedure or algorithm, a determination may be made with regard to which data portions to append each of the left and right segments. Any suitable algorithm may be used for making this determination. For example, in one suitable approach, a table of all possible distributions (e.g., in the form of pairings of destinations for the left segment and for the right segment) may be created, whereby a destination share value for each of the left and right segment may be determined by using any suitable hash function on corresponding data in the session key, cipher feedback session key, or any other suitable random or pseudo-random value, which may be generated and extended to the size of the original data. For example, a hash function of a corresponding byte in the random or pseudo-random value may be made. The output of the hash function is used to determine which pairing of destinations (i.e., one for the left segment and one for the right segment) to select from the table of all the destination combinations. Based on this result, each segment of the split data unit is appended to the respective two shares indicated by the table value selected as a result of the hash function.

Redundancy information may be appended to the data portions in accordance with the present invention to allow for the restoration of the original data using fewer than all the data portions. For example, if two out of four portions are desired to be sufficient for restoration of data, then additional data from the shares may be accordingly appended to each share in, for example, a round-robin manner (e.g., where the size of the original data is 4 MB, then share 1 gets its own shares as well as those of shares 2 and 3; share 2 gets its own share as well as those of shares 3 and 4; share 3 gets its own share as well as those of shares 4 and 1; and share 4 gets its own shares as well as those of shares 1 and 2). Any such suitable redundancy may be used in accordance with the present invention.

It will be understood that any other suitable parsing and splitting approach may be used to generate portions of data from an original data set in accordance with the present invention. For example, parsing and splitting may be randomly or pseudo-randomly processed on a bit by bit basis. A random or pseudo-random value may be used (e.g., session key, cipher feedback session key, etc.) whereby for each bit in the original data, the result of a hash function on corresponding data in the random or pseudo-random value may indicate to which share to append the respective bit. In one suitable approach the random or pseudo-random value may be generated as, or extended to, 8 times the size of the original data so that the hash function may be performed on a corresponding byte of the random or pseudo-random value with respect to each bit of the original data. Any other suitable algorithm for parsing and splitting data on a bit by bit level may be used in accordance with the present invention. It will further be appreciated that redundancy data may be appended to the data shares such as, for example, in the manner described immediately above in accordance with the present invention.

In one suitable approach, parsing and splitting need not be random or pseudo-random. Rather, any suitable deterministic algorithm for parsing and splitting data may be used. For example, breaking up the original data into sequential shares may be employed as a parsing and splitting algorithm. Another example is to parse and split the original data bit by bit, appending each respective bit to the data shares sequentially in a round-robin manner. It will further be appreciated that redundancy data may be appended to the data shares such as, for example, in the manner described above in accordance with the present invention.

In one embodiment of the present invention, after the secure data parser generates a number of portions of original data, in order to restore the original data, certain one or more of the generated portions may be mandatory. For example, if one of the portions is used as an authentication share (e.g., saved on a physical token device), and if the fault tolerance feature of the secure data parser is being used (i.e., where fewer than all portions are necessary to restore the original data), then even though the secure data parser may have access to a sufficient number of portions of the original data in order to restore the original data, it may require the authentication share stored on the physical token device before it restores the original data. It will be understood that any number and types of particular shares may be required based on, for example, application, type of data, user, any other suitable factors, or any combination thereof.

In one suitable approach, the secure data parser or some external component to the secure data parser may encrypt one or more portions of the original data. The encrypted portions may be required to be provided and decrypted in order to restore the original data. The different encrypted portions may be encrypted with different encryption keys. For example, this feature may be used to implement a more secure "two man rule" whereby a first user would need to have a particular share encrypted using a first encryption and a second user would need to have a particular share encrypted using a second encryption key. In order to access the original data, both users would need to have their respective encryption keys and provide their respective portions of the original data. In one suitable approach, a public key may be used to encrypt one or more data portions that may be a mandatory share required to restore the original data. A private key may then be used to decrypt the share in order to be used to restore to the original data.

Any such suitable paradigm may be used that makes use of mandatory shares where fewer than all shares are needed to restore original data.

In one suitable embodiment of the present invention, distribution of data into a finite number of shares of data may be processed randomly or pseudo-randomly such that from a statistical perspective, the probability that any particular share of data receives a particular unit of data is equal to the probability that any one of the remaining shares will receive the unit of data. As a result, each share of data will have an approximately equal amount of data bits.

According to another embodiment of the present invention, each of the finite number of shares of data need not have an equal probability of receiving units of data from the parsing and splitting of the original data. Rather certain one or more shares may have a higher or lower probability than the remaining shares. As a result, certain shares may be larger or smaller in terms of bit size relative to other shares. For example, in a two-share scenario, one share may have a 1% probability of receiving a unit of data whereas the second share has a 99% probability. It should follow, therefore that once the data units have been distributed by the secure data parser among the two share, the first share should have approximately 1% of the data and the second share 99%. Any suitable probabilities may be used in accordance with the present invention.

It will be understood that the secure data parser may be programmed to distribute data to shares according to an exact (or near exact) percentage as well. For example, the secure data parser may be programmed to distribute 80% of data to a first share and the remaining 20% of data to a second share.

According to another embodiment of the present invention, the secure data parser may generate data shares, one or more of which have predefined sizes. For example, the secure data parser may split original data into data portions where one of the portions is exactly 256 bits. In one suitable approach, if it is not possible to generate a data portion having the requisite size, then the secure data parser may pad the portion to make it the correct size. Any suitable size may be used.

In one suitable approach, the size of a data portion may be the size of an encryption key, a splitting key, any other suitable key, or any other suitable data element.

As previously discussed, the secure data parser may use keys in the parsing and splitting of data. For purposes of clarity and brevity, these keys shall be referred to herein as "splitting keys." For example, the Session Master Key, previously introduced, is one type of splitting key. Also, as previously discussed, splitting keys may be secured within shares of data generated by the secure data parser. Any suitable algorithms for securing splitting keys may be used to secure them among the shares of data. For example, the Shamir algorithm may be used to secure the splitting keys

whereby information that may be used to reconstruct a splitting key is generated and appended to the shares of data. Any other such suitable algorithm may be used in accordance with the present invention.

Similarly, any suitable encryption keys may be secured within one or more shares of data according to any suitable algorithm such as the Shamir algorithm. For example, encryption keys used to encrypt a data set prior to parsing and splitting, encryption keys used to encrypt a data portions after parsing and splitting, or both may be secured using, for example, the Shamir algorithm or any other suitable algorithm.

According to one embodiment of the present invention, an All or Nothing Transform (AoNT), such as a Full Package Transform, may be used to further secure data by transforming splitting keys, encryption keys, any other suitable data elements, or any combination thereof. For example, an encryption key used to encrypt a data set prior to parsing and splitting in accordance with the present invention may be transformed by an AoNT algorithm. The transformed encryption key may then be distributed among the data shares according to, for example, the Shamir algorithm or any other suitable algorithm. In order to reconstruct the encryption key, the encrypted data set must be restored (e.g., not necessarily using all the data shares if redundancy was used in accordance with the present invention) in order to access the necessary information regarding the transformation in accordance with AoNTs as is well known by one skilled in the art. When the original encryption key is retrieved, it may be used to decrypt the encrypted data set to retrieve the original data set. It will be understood that the fault tolerance features of the present invention may be used in conjunction with the AoNT feature. Namely, redundancy data may be included in the data portions such that fewer than all data portions are necessary to restore the encrypted data set.

It will be understood that the AoNT may be applied to encryption keys used to encrypt the data portions following parsing and splitting either in place of or in addition to the encryption and AoNT of the respective encryption key corresponding to the data set prior to parsing and splitting. Likewise, AoNT may be applied to splitting keys.

In one embodiment of the present invention, encryption keys, splitting keys, or both as used in accordance with the present invention may be further encrypted using, for example, a workgroup key in order to provide an extra level of security to a secured data set.

In one embodiment of the present invention, an audit module may be provided that tracks whenever the secure data parser is invoked to parse and/or split data.

FIG. **36** depicts an exemplary bit-splitting technique **3600** in accordance with the present disclosure. A session key **3606** may be used for bit-splitting, and may be created as described herein, or in accordance with any technique as understood in the art. For example, a session master key **3604** may be XORed with an initial session key **3602** to create the session key **3606**. The session key **3606** may be key expanded, such as by using ANSI X9.82 as a cryptographically secure pseudo-random number generator and an RC4 type technique for randomizing the order. ANSI x9.82 and RC4 are merely exemplary, and any key expansion technique known in the art may be used. The key expansion may be performed in a manner such that if any M of the N expanded keys are recovered, the entire expanded session key **3606** may be reconstructed.

An exemplary byte of data **3608** to be split by the bit-splitting may be the ASCII character, "S" (i.e., 01010011). A byte of the expanded session key **3606**, such as the first session key byte **3610** (82 or 01010010) may optionally be XORed with the byte of data **3608** to produce data **3612**. If session key byte **3610** is not XORed with data **3608**, data **3608** may be used in subsequent steps as data **3612**. Session key byte **3610** may be right shifted to produce a split point. In one exemplary embodiment, session key byte **3610** may be right shifted by five bits to produce a split point of 2 or 00000010. The split point may be used to create two masks that separate byte **3612** into two portions. In this example the split point is two, and the two lowest significant bits may be set to 1 to create mask **3616** (00000011) while the remaining six most significant bits may be set to 1 to create mask **3614** (11111100). The result of ANDing byte **3612** with masks **3614** and **3616** and discarding the 0-mask bytes is to create left split **3618** (six most significant bits of data **3612**) and right split **3620** (two least significant bits of data **3612**).

A feedback value **3622** may be selected, such as by a 4-byte look ahead within session key **3606**. An exemplary feedback value **3622** may be 99 or 01100011. The same split point (and masks **3614** and **3616**) may be used to separate the feedback value **3622** into a left feedback value **3624** (six most significant bits of feedback value **3622**) and a right feedback value **3626** (two least significant bits of feedback value **3622**). The left feedback value **3624** may be XORed with the left split **3618** to create split value left **3628**. The right feedback value **3626** may be XORed with the right split **3620** to create split value right **3630**. Split value left **3628** and split value right **3630** may be distributed to shares as described herein.

FIG. **37** depicts a data flow **3700** for distributing the output of the bit-splitting process **3600** (split value right **3630** and split value left **3628**) to one or more of shares **3704**, **3706**, **3708**, and **3710**. The bytes of DK **3702** may indicate the distribution of data to respective shares **3704**, **3706**, **3708**, and **3710**. For example, the first bit of DK **3702** may correspond to share **3704**, the second bit of DK **3702** may correspond to share **3706**, the third bit of DK **3702** may correspond to share **3708**, and the fourth bit of DK **3702** may correspond to share **3710**. An exemplary first DK byte **3702** (e.g., 0011) may result in the distribution of split value right **3630** to share **3704** and the distribution of split value left **3628** to share **3706**.

Bit splitting as described herein may be used with various M-of-N share recovery configurations. For example, the distribution may be a 4 of 4 split such that all shares must be recovered for the underlying keys and data to be recovered. The session key **3606** may be divided into 4 parts, with each part distributed to one of the four shares **3704**, **3706**, **3708**, and **3710**. Alternatively, the session keys described herein may be expanded such that the resulting bit split distribution will allow the data to be recovered from M shares where M<N, e.g., a 2 of 4 key expansion or 3 of 4 key expansion.

Another exemplary technique for the distribution of data into shares may be a bit scatter technique as depicted in FIG. **38**. The bit scatter technique may separate a stream of input bytes into separate output streams by bit. A master session key **3802** may be M of N expanded to create a session key **3804** with N key elements and equal in length to the number of data **3806** bytes times 8. Each session key **3804** element may be associated with a share **3808** and may include an array of integer values. Each integer value may correspond to a bit position within each byte of input data **3806** to be scattered.

Bits may be scattered to shares by iterating through the integer values of an element of session key **3804**. The selection of which bits of data **3806** will be assigned to a share **3808** as well as the order in which they are assigned may be based on the integer value from the element of session key **3804** associated with a particular share as described below. An exemplary session key **3804** element may have integer values 2, 4, 3, 7, and 1 respectively. The integer value of 2 from the session key **3804** element may indicate that the value from the second bit of each byte of data **3806** may be distributed to the first bit location of share **3808**. The integer value of 4 from the session key **3804** element may indicate that the value from the fourth bit of data **3806** may be distributed to the next bit of share **3808**, and so on for values 3, 7, and 1 of the session key **3804** element. This process may be repeated until each bit of data **3806** has been distributed to the share **3808** based on the session key **3804** element. The process may then be repeated on data **3806** for each respective share and associated session key element.

FIG. **39** depicts restoring the portion of a byte of data **3906** that was distributed to a share **3908** using the bit scatter technique. The master session key **3902** may be established or recovered as is described herein, e.g., by recovering a key distributed among shares. Master session key **3902** may be M of N key expanded to create session key **3904** (which may match the session key used to scatter the bits originally) having elements associated with shares as described herein. The integer values of the session key **3904** elements may be used to extract bits from the shares to recreate data **3906**. The first integer value of 2 from the session key **3904** element may indicate that the first bit from share **3908** may be distributed to the second bit of data **3906**. The second integer value of 4 from the session key **3904** element may indicate that the second bit from share **3908** may be distributed to the fourth bit of data **3906**, and so on for the values 3, 7, and 1 of the session key **3904** element associated with share **3908**. This process may be continued for share **3908** until all of the bits from the share **3908** are distributed to data **3906**, and then repeated in its entirety for each additional share based on the element of session key **3904** associated with that share.

Any key expansion technique known in the art may be used to create the expanded session keys associated with the distribution of data to shares and reconstruction of data from shares. An exemplary M of N key expansion utilizing ANSI X9.82 as a cryptographically secure pseudo-random number generator and an RC4 type technique for randomizing the order will be discussed below with respect to performing a 2 of 4 key expansion for use with a bit scatter technique. A master session key may feed the ANSI X9.82 cryptographically secure pseudo-random number generator to create key value k. In an exemplary embodiment the X9.82 technique may be operating in AES-OFB (output feedback) mode. Each resulting k may have a length (K_{len}) equivalent to the length of the plain text data input, padded for AES as necessary, plus 4. The key blocks k may be created based on the master key (MK) and the key length (K_{len}) as represented by k=aes_ofb(MK, aes_pad_length (K_{len})).

An array of integers S may be instantiated in a manner such that the order of the integers S corresponds to the order of the plain text data. The array may be initialized from S_0, S_1, . . . S_{len}, where the key length is assumed to be equal to the length (len) of the array S. The integer values S may then be randomized based on the key k. Counters i,j, and x may be initially set to 0, and the elements of array S swapped according to the following:

```
for (i ... len)
    j = (j + k_x + S_i) mod len
    swap ( S[i], S[j])
        x++;
```

The resulting randomized S values may be utilized to create the expanded session key elements SK_0 . . . SK_N, each associated with a corresponding share of the N shares. In order to create an M of N share distribution such that there is sufficient overlap of the scattered data for the data can be recovered from any M of N shares, a ratio may be computed as follows: ratio=1−(1/N)*(M−1). In the case of a 2 of 4 key split, the ratio may equal 1−(¼)*(2−1) or 75%. The randomized S values may then be allocated to each of session keys SK_0 . . . SK_N based on the ratio, with each successive copying from the randomized S values beginning where the previous session key SK left off, and looping back to the beginning of the randomized S array as necessary. An example of such an allocation for a 2 of 4 key split with a ratio of 75% is depicted in FIG. **40**. SK1 may be the initial 75% of the S array, SK2 the final 25% and initial 50% of the S array, SK3 the final 50% and initial 25% of the S array, and SK4 the final 75% of the S array. Once the session keys SK_0 . . . SK_N are created, each may then be re-randomized with a random nonce n in accordance with the following:

```
for (SK_0 ... SK_N)
    i = j = x =0;
    for (i ... len)
        j = (j + n_x + S_i) mod len
        swap ( S[i], S[j])
        x= (x+1) mod keylen;
```

The resulting SK values may then be utilized as the expanded session key elements for use in data splitting as described herein.

Another exemplary technique for the distribution of data into shares may be a block split technique. Data such as a file that is to be split and distributed to shares may include blocks, i.e., consecutive groupings of a fixed number of bits. Examples of block sizes may include 128, 192, or 256. A segment may be a group of blocks that is utilized in the block split technique. If L represents the number of mandatory shares, M represents the number of non-mandatory shares necessary to recover a file, and N represents the total number of non-mandatory shares, a file may be split into L+C(N, M−1) segments, where C(N,M−1) denotes the binomial coefficient N choose M−1. If a file is split into L+C(N,M−1) segments, the file may not be recovered unless all L mandatory shares and at least M non-mandatory shares are recovered.

Blocks may be assigned to segments based on any appropriate technique. For example, F may be the number of blocks in a file being split. Segmenting values a and b may be calculated according to the following:

$$a = \left\lceil \frac{F}{\binom{N-1}{M-1}\cdot\left(\frac{N}{N-M+1}+L\right)} \right\rceil = \left\lceil \frac{F}{\binom{N}{M-1}+\binom{N-1}{M-1}\cdot L} \right\rceil,$$

$$b = a\cdot\binom{N}{M-1}.$$

Blocks are assigned to segments such that each ith segment contains blocks (a*i) through (a(i+1)−1) for all i segments where i<C(N,M−1). The remaining L segments may be partitioned evenly from the end of the file and contain blocks b+[(F−b)*(i−C(N,M−1))/L] through b+[(F−b)*((i+1)−C(N,M−1))/L]−1, inclusive.

The last L segments of the file may be assigned to the L mandatory shares with one segment per share. The first C(N,M−1) segments may be assigned to the N non-mandatory shares by first enumerating all C(N,M−1) possibilities of the (M−1)-element subsets of {0, 1, . . . , N−1} in lexicographical order. If and only if the ith (M−1)-element subset does not include the element j, then the jth non-mandatory share will contain segment i. Based on this distribution, every possible combination of M−1 non-mandatory shares is missing at least one of the C(N,M−1) segments, while every collection of M non-mandatory shares includes all C(N,M−1) segments.

Each share may include a preamble. An exemplary preamble may have a first line including six space-separated integers: (1) i; (2) L; (3) M; (4) N; (5) C(N−1,M−1); and (6) the size of a block in bits. Lines 1+i (for i<C(N−1,M−1)) may correspond to each segment stored in the share, and may include three space-separated integers relating to the segment: (1) the segment number; (2) the number of the first block in the segment; and (3) the number of the last block in the segment. If the share is mandatory, the preamble may be the same except it only contains two lines.

To recover the data, the shares may be joined if all L mandatory shares and at least M non-mandatory shares have been received. This may be determined by examining the share preambles to insure that all L+C(N,M−1) segments are present in the received shares. Shares may be written out based on the offsets indicating the location of segments within the shares.

Another exemplary technique for the distribution of data into shares may be a bit segment technique. For purposes of the bit segment technique, a segment is a group of bits that is used for purposes of the splitting and joining process. Splittable data such as a file may split into C(N+L,M−1) segments, such that the file cannot be recovered from any less than L+M shares. Segmenting values a and b may be calculated according to the following:

$$a = \left\lfloor \frac{F}{\binom{N-1}{M-1}\cdot\left(\frac{N}{N-M+1}+L\right)} \right\rfloor = \left\lfloor \frac{F}{\binom{N}{M-1}+\binom{N-1}{M-1}\cdot L} \right\rfloor,$$

$$b = a\cdot\binom{N}{M-1}.$$

An offset S may be equal to C(N,M−1). For i=0 to C(N,M−1)−1, the ith segment may contain bits i, S+i, 2*S+i, . . . , a*S+i. Segments for the L mandatory shares may be computed by dividing the remaining (b*L) bits such that a mandatory share j may contain bits (a+1)*S+j, (a+1)*S+L+j, (a+1)*S+2L+j, . . . , (a+1)*S+bL+j.

The first C(N,M−1) segments may be assigned to the N non-mandatory shares by enumerating all C(N,M−1) possibilities of the (M−1)-element subsets of {0, 1, . . . , N−1} in lexicographical order. The remaining L segments are then assigned to the L mandatory shares. Bits may be distributed from chunks of C(N,M−1) bits, wherein each chunk contains one bit from each segment. The chunks may be parsed sequentially one bit at a time, with each bit written to the share assigned to its segment.

Each share i may include a preamble. An exemplary preamble may include a first line including six space-separated integers: (1) i; (2) L; (3) M; (4) N; (5) C(N−1, M−1); and (6) the size of a block in bits. Lines 1+i (for i<C(N−1,M−1)) may each include three space-separated integers for each segment assigned to the share: (1) the segment number; (2) the number of the first bit in the segment; and (3) the number of total bits in the segment. If the share is mandatory, the preamble may be the same except it only contains two lines.

The shares may be joined if all L mandatory shares and at least M non-mandatory shares have been received. This may be determined by examining the share preambles to insure that all L+C(N,M−1) segments are present in the received shares. Shares may be written out based on the offsets indicating the location of segments within the shares.

FIG. **41** shows illustrative overview process **4100** for using the secure data parser of the present invention in some embodiments. The secure data parser may be compliant with Federal Information Processing Standards (FIPS) such as FIPS 140-2, including but not limited to standards relating to data encryption, key storage techniques, and cryptographically secure pseudo-random number generation. As described above, two well-suited functions for secure data parser **4106** may include encryption **4102** of data for storage or transmission and backup **4104** (e.g., backup tapes, mirroring, RAID). As such, secure data parser **4106** may be integrated with a RAID or backup system, a file storage system, a data transmission system, or a hardware or software encryption engine in some embodiments. The secure data parser may be modular in nature, allowing for any known technique to be used within each of the function blocks shown in FIG. **41**. For example, AES encryption could be replaced by other known encryption techniques such as Triple DES (3DES).

The processes associated with secure data parser **4106** may include one or more of pre-encryption process **4108**, encrypt/transform process **4110**, key secure process **4112**, parse/distribute process **4114**, fault tolerance process **4116**, share authentication process **4118**, and post-encryption process **4120**. These processes may be executed in several suitable orders or combinations. The combination and order of processes used may depend on the particular application or use, the level of security desired, whether optional pre-encryption, post-encryption, or both, are desired, the redundancy desired, the capabilities or performance of an underlying or integrated system, or any other suitable factor or combination of factors.

Pre-encryption process **4108** may operate independently and prior to any encryption of the data splitting process, and may utilize encryption techniques described herein or known in the art, including AES (FIPS 197), DES, and 3DES. Meta-data such as the file name, file length, creation/modification dates, and any other host-specific file information may be pre-pended to the data prior to pre-encryption. The key for pre-encryption may be generated by the system, or may be provided by an external system. Keys may be symmetric or may include public/private key pairs such as RSA public/private key, DSA public private key, and ECDSA public/private key. Pre-encryption process may also include the generation of a MAC or digital signal for the pre-encrypted data as described herein.

Encryption/transform **4110** process may receive pre-encrypted or non-encrypted data. Encryption/transform **4110** process may perform key generation, data encryption, and

key transform steps as described herein. For example, AES encryption may be used with a cryptographically secure pseudo-random number generator to create a 128, 192, or 256-bit key. That key may be used to encrypt data using AES cipher modes such as AES counter (AES-CTR) or AES cipher block chaining (AES-CBC). Utilizing non-sequential encryption modes such as AES-CTR may allow for parallel processing of data, which may allow faster multi-processing of data. The key may then be transformed for splitting and storage such as with a package transform or AoNT as described herein.

Key secure process **4112** may control the splitting and distribution of the key to shares, such as through a Shamir key distribution or other key distribution techniques as described herein. Parse/distribute process **4114** may split data according to a technique such as bit split, block segment, bit scatter, or block split. The split data may in turn be assigned to shares as described herein and based on the type of splitting technique used. The operation of the splitting technique and assignment of split data and split keys to shares may also be dependent on the M of N fault tolerance employed by fault tolerance process **4116** in a particular embodiment. A N of N embodiment may provide for a higher level of data security, since all shares must be received to recover the underlying keys and data. A M of N embodiment may provide redundancy. In addition, fault tolerance process **4116** may also implement mandatory shares L that must be recovered even in a M of N embodiment.

Share authentication process **4118** may provide integrity, authentication and digital signature functionality for information to be distributed to shares. Integrity information may include computing share integrity information as described herein, such as by utilizing a SHA-256 (FIPS 180-2) hash and computing a hash tree. Authentication may include the generating of a MAC for each share, e.g., based on the underlying data, the initialization vector, and any attached meta-data for the share. A shared secret MAC key may be any appropriate key as is known in the art, which may in turn be used as a key for a MAC technique such as HMAC-SHA1 (FIPS 198) to produce, e.g., a 160-bit MAC tag. Shares may also be signed by a digital signature technique for authentication of the sender. The sending party may utilize FIPS PUB 186-2 signature techniques such as ECDSA using 384/512-bit curves or DSA using a modulus size of 512-1024 bits to sign the share using a private signing key. Other example signature techniques include RSASSA-PSSA.

Post-encryption process **4120** may optionally operate to further encrypt the shares, including data, meta-data, keys, initialization vector, and any integrity, authentication or signature information. Encryption techniques used by post-encryption process **4120** may include those described herein and known in the art, including AES, DES, and 3DES. The key for post-encryption may be generated by the system, may be provided by a user, or may be provided by an external system, and may include symmetric or public/private key pairs such as RSA public/private key, DSA public private key, and ECDSA public/private key. Each post-encrypted share may also include additional authenticity or integrity information.

As noted above, pre-encryption process and post-encryption process may be optional. In addition, because data parsing provided by parse/distribute process **4114** provides security unless all required shares are discovered, any encryption performed by the data parser may also be optional. Also, any of the encryption processes may be performed on only some portion of the data, e.g., such that only a subset of the data or shares is encrypted, and thus reduce any burdens of the encryption processes.

FIG. **42** depicts an exemplary secure parser **4200**. The secure parser core **4202** isolates the functions of data splitting (e.g., share creation, redundancy, and distribution to share buffers) into a callable module. Data may be placed in assembled data buffer **4204** to be split and the secure parser core **4202** may be invoked. The resulting shares may be placed into the split share buffers **4206** to be securely stored (storage **4216**) or transmitted (communications **4218**) by the host application. Cryptographic functions may be external calls for the secure parser core **4202**. This may allow for the custom integration of various cryptographic libraries, giving flexibility for "bolting on" other encryption modules, key management systems, random number generators, etc. as needed.

Random number generator **4208** may, for each split operation, generate a unique random value to be used as the basis for the split. Random numbers may be generated in a number of ways, via software program or hardware interface. AES encryption module **4210** may utilize encryption such as AES-CTR or AES-CBC for encryption operations for the secure parser core **4202**. Encryption module **4212** may be an external encryption process that may provide functionality for pre-encrypting data prior to splitting or post-encrypting data after splitting. Key management system **4214** may provide for external key management for keys utilized in the encryption and/or splitting operations of the secure parser.

FIG. **43** depicts an exemplary integration of the secure parser **4308** into a system **4300** utilizing an Application Programming Interface (API). A secure parser **4308** may include a secure parser core **4302**, split share buffers **4310**, and assembled data buffer **4312**. API/Wrapper Layer **4304** may be an integration layer that interfaces with an application **4306**. API/Wrapper Layer **4304** may define the location of shares, such as paths to send the information or storage devices to store the shares. API/Wrapper Layer **4304** may also define the naming convention of the resulting data, or parameters to store with the information such as row, column, and table ID in the case of a database application or the original filename in the case of unstructured data such as documents, images, maps, multimedia, or streaming data. API/Wrapper Layer **4304** may also handle the streaming of data to the secure parser **4308** and secure storage or communications of the resulting shares.

Application **4306** may select data to be secured and call the secure parser. API/Wrapper layer **4304** may stream input data into assembled data buffer **4312** and save filename and share information relating to the data. Assembled data buffer **4312** may provide data to secure parser core **4302** which may create shares, implement redundancy, and assign shares to locations for distribution as described herein. The shares to be distributed may be output to split share buffer **4310** which may provide the shares to API/Wrapper Layer **4304**.

The various operations described herein may be chosen to match the threat and risk scenario for particular data, creating over a thousand different configuration options for a secure parser. Data may be pre-encrypted before it is parsed into shares, and/or post-encrypted after shares are created. Keys utilized in parsing may be transformed such as by the AoNT transform or package transform. Data parsing can be by various techniques such as bit split, block split, bit scatter, bit segment, block shuffle, or ordered block. The encryption utilized in the parsing technique may include various AES modes as described herein (e.g., AES-CTR) or DES/3DES encryption, and keys used in encryption and or parsing of

data may be various lengths such as 128, 296, or 256 bits. Fault tolerance may be implemented in various manners such as N of N, M of N, or parity. Keys may be secured such as with a Shamir key splitting technique or other techniques, including external splitting techniques. Integrity information may be provided and shares may be authenticated utilizing a hash, a MAC and/or a digital signature.

FIG. **44** depicts an example configuration of a 2 of 3 share fault tolerance system. Data **4408** may be created locally at Site A **4402**. A secure parser may secure and split each block of data **4408** from Site A **4402** and distribute 66% of the split data to each of Site A **4402**, Site B **4404**, and Site C **4406**. The original data may be recovered from any 2 or 3 of shares **4410**, **4412**, and **4414**.

FIG. **45** depicts an example configuration of a 2 of 3 share fault tolerance system with a mandatory share **4512**. Data **4508** may be created locally at Site A **4502**. The secure parser may secure and split each block of data **4508** from Site A **4502**, store 1% of the data locally at share **4510**, distribute 99% to share **4512** at Site B **4504**, and distribute 1% to share **4514** at Site C **4506**. In order to recover the original data, share **4512** must be recovered from Site B **4504**, and either one of share **4510** from Site A **4502** or share **4514** from Site C **4506** must be recovered. Data may be recovered to cleartext, e.g., at Site B **4504**.

FIG. **46** depicts a flowchart **4600** of aspects of encrypt/transform process **4110**, key secure process **4112**, parse/distribute process **4114**, fault tolerance process **4116**, and share authentication process **4118**. Input data **4602** to be split may be received. The received input data **4602** may be un-encrypted data or may have optionally been pre-encrypted using AES or other encryption techniques as described herein or known in the art. The received input data **4602** may then be encrypted using an encryption technique such as AES. A session key **4604** may be generated using a random number generator such as a cryptographically secure pseudo-random number generator. In the example of AES encryption, the session key **4604** length may be 128, 192, or 256 bits. AES encryption may use any known AES technique such as AES-ENC or AES DEC, and may operate in any known cipher mode such as AES electronic codebook (AES-ECB), AES cipher block chaining (AES-CBC), AES output feedback (AES-OFB), AES Counter (AES-CTR), AES cipher feedback (AES-CFB), or any other cipher mode that is known in the art.

Depending on the type of AES cipher mode utilized for encryption, input data **4602** may be padded until it is equal in size to an integer number of AES blocks. The padding may consist of a 1-bit followed by 0-bits. The padded input data **4602** may be split into n blocks **4606**. An initialization vector (IV) may be generated using a cryptographically secure pseudo-random number generator, and the n blocks **4606** may then be encrypted with the IV and Key **4604** utilizing known AES cipher modes, e.g., AES-CBC or AES-CTR.

In an exemplary embodiment, AES-CTR mode may use the AES-ENC primitive to implement a stream cipher. For every 128-bit plaintext block to be encrypted, CTR mode may first generate a 128-bit keystream block by calling AES-ENC(K, T), where T is a counter value incremented at each iteration (when T exceeds the maximum counter value, it wraps back to 0). The plaintext block may then be encrypted by XORing it with the keystream block.

CTR-mode encryption may be defined by the following piece of pseudo-code. K may be an encryption key, and To may be an initial counter value (Initialization Vector). The plaintext may be represented as a series of 128-bit blocks ($M_1, \ldots, M_n$) where M_j represents the ith block of plaintext. AES-CTR may not require plaintext padding: when the plaintext does not divide perfectly into an integer number of blocks, then the final block Mn may be a partial block of size p bits, where p<128. When a p-bit plaintext block is XORed with a normal keystream block, only the first p bits of the two blocks may be considered. The resulting ciphertext may be represented as a series of blocks ($C_1, \ldots, C_n$) along with the initial counter value T_0. To encrypt a message using AES-CTR:

$T=T_0$

For I=1 to n

$C_i=M_i$ XOR AES-ENC(K,T)

T=T+1

CTR-mode decryption may be identical to encryption, except that the technique operates on ciphertext blocks. To correctly decrypt a CTR-mode ciphertext (T0, $C_1, \ldots, C_n$), compute:

$T=T_0$

For I=1 to n

$C_i=M_i$ XOR AES-ENC(K,T)

T=T+1

CTR mode encryption may require that the 128-bit initial counter value T_0 be initialized to any value between 0 and $2^{128}-1$ inclusive. The default value for T_0 may be 0. This counter may be subsequently incremented for each 128-bit block encrypted or decrypted (if the counter exceeds $2^{128}-1$, it wraps back to 0). To ensure the security of CTR-mode encryptions, it may be necessary that no value of T ever be re-used without re-keying. The technique of choosing the initial counter may be dependent on the implementation.

CTR mode may be parallelized across multiple processes or processors: after selecting an initial counter T_0, the plaintext may be divided into separate segments and encrypted separately, provided that each encryption process is given the correct counter offset for the first block it encrypts.

In another exemplary embodiment, AES-CBC mode may use the AES-ENC and AES-DEC primitives to encrypt/decrypt multi-block plaintexts. In addition to the encryption key K, CBC mode may require a 128-bit Initialization Vector (IV), denoted by T_0. Note that for security, this Initialization Vector may be 1) unique for each message encrypted, and 2) generated using a cryptographically-secure random number generator. AES-CBC mode may require that the plaintext consist of an even number of AES blocks, i.e., is a multiple of 128-bits. When this is not the case, the plaintext may be padded by appending a 1 followed by as many 0s as necessary to reach a block boundary. If there is insufficient room to insert the shortest unambiguous padding sequence "10", an additional plaintext block may be added.

AES-CBC may compute each ciphertext block by XORing a plaintext block with the previous ciphertext block, then encrypting the result using AES-ENC. In the case of the first ciphertext block, the Initialization Vector T_0 replaces the previous ciphertext value. CBC-mode encryption may be defined by the following piece of pseudo-code. The plaintext may be represented as a series of 128-bit blocks ($M_1, \ldots, M_n$) where M_i represents the ith block of plaintext. The resulting ciphertext may be represented as a series of blocks

(C_1, . . . , C_n) along with the Initialization Vector T_0. To encrypt a message using AES-CBC:

C_1=AES-ENC(K, M_1 XOR T_0)

For i=2 to n

C_i=AES-ENC(K, M_1 XOR C_{i-1})

To correctly decrypt a CBC-mode ciphertext (T_0, C_1, . . . , C_n), compute:

M_1=AES-DEC(K, XOR T_0

For i=2 to n

M_i=AES-DEC (K, C_1)XOR C_{i-1}

Once the AES encryption is completed, the n encrypted blocks **4608** may then be recombined, with the IV prepended, to be passed to the splitting technique **4610**. Before splitting occurs, a meta-data block may also be pre-pended to indicate information such as file length, encryption mode, key size, share ID, L/M/N share values, encryption key ID, encryption IV, hash mode, original buffer size, share buffer size, pre-encryption key ID, pre-encryption IV, pre-signature ID, pre-signature IV, and a flag indicating whether the a key transform such as package or AoNT transform was used to process the key **4604**. The data (including IV and metadata) may then be split for distribution to the shares **4614** in accordance with a splitting technique such as bit split, block split, bit segment, or bit scatter as described herein.

The session key **4604** may optionally be transformed into transform key **4612** such as through a AoNT or a package transform. In the example of a package transform, the session key **4604** may be XORed with the n encrypted blocks **4608**, resulting in a transform key **4612** that cannot be recovered without also recovering all of then encrypted blocks. The session key **4604** (if no transform takes place) or transform key **4612** may then be split using the Shamir technique or other key-splitting techniques that are well known in the art. For example, the Shamir technique may create L+1 shares, where L is the number of mandatory shares of shares **4614**. Each of the L shares may then be distributed to a corresponding mandatory share of shares **4614**. The remaining share may then be split into N shares of shares **4614** and distributed in a manner such that the remaining share can be recovered from any M of N shares of shares **4614** as may be required for the particular application. The session key **4604** (if no transform takes place) or transform key **4612** may then be optionally held as a session key, encrypted such as with a public key, or discarded.

Once the data and/or key has been split for distribution to the shares **4614**, each resulting share may further implement integrity protection. As one example, a collision resistant cryptographic hash function (implemented by SHA-256) using hash-tree construction may be used to compute an integrity protection value. A hash value may be calculated for each share using a hashing technique such as SHA-256. If the total number of shares is less than 8, the hash values for all shares may be concatenated and the concatenated result may be stored with each of the shares. If there are 8 or more shares, the integrity protection value may be computed by the following steps:

1. Pre-pend 0x00 to each share, and compute a hash value H(0x00|SHARE) for each share;
2. Compute a hash value H(0xFF|A|B) for each consecutive pair of shares, where A and B are the respective outputs of step (1) for each share (e.g., A=H (0x00|$SHARE_x$); B=H(0x00|$SHARE_{x+1}$). If the total number of shares is odd, compute a hash value for the final share for (0xFF|A), where A=H (0x00|$SHARE_{FINAL}$).
3. Repeat step (2) on the resulting hash values, until only a single hash result (the hash tree root) remains;
4. Distribute to each share $SHARE_i$ the hash of $SHARE_i$ (H($SHARE_i$)), the tree root, and up to one value from each level of the tree (i.e., iteration of steps (2) & (3)), as necessary to calculate the tree root from H($SHARE_i$).

The resulting integrity protection values may be appended to the corresponding share data. Because all hash values are provided to all shares (n<8) or a hash tree is provided (n≥8), The integrity protection as described herein embeds with each share integrity check information about other shares. Thus, if the integrity of any one share in a set is known to be authentic, the integrity of the remaining shares may be evaluated based on the integrity check information of the authentic share.

The resulting share (including the integrity protection value) may optionally be authenticated (e.g., creating a MAC as described herein) and signed (e.g., a digital signature as described herein). The resulting information may optionally be post-encrypted, for example, using AES or other encryption techniques as described herein and known in the art. The resulting encrypted share may then be distributed, such as to shares for storage or to protect data in motion.

Data may be recovered from shares (e.g., accessed from storage in shares or received as data in motion) according to the steps of FIG. **47**. Some or all of shares **4702** may be recovered. If the data was post-encrypted prior to distribution to the shares **4702**, the data may be decrypted prior to data recovery as described herein. If a share was signed with a digital signature, or includes a MAC, the signature and/or MAC may be confirmed as described herein.

Each share may have stored a hash value for integrity protection as described above. To confirm the integrity of a received share, a hash value may be computed based on the received data (SHARE) pre-pended with 0x00, in the same manner as was performed to create the integrity protection value. If the computed hash value from the received SHARE data does not match the hash value stored with the share SHARE, then the integrity of SHARE has been compromised and data recovery should be aborted.

If the total number of shares is less than 8, the computed hash value for SHARE may also be compared against the stored hash value for SHARE stored in other shares. If the computed hash of SHARE fails to match any stored hash of SHARE, the data recovery should be aborted. This step of checking a computed hash of $SHARE_i$ against all other stored hashes of $SHARE_i$ may be repeated for all shares. If the total number of shares is 8 or more, then the root of the hash tree may be determined from the computed hash of $SHARE_i$ along with the information stored with the share. This determination may be repeated for each share, and if the hash tree root determined for any individual share does not match, the data recovery may be aborted.

A key **4706** may be recovered from shares **4702** if all mandatory shares are available as well at least M of N non-mandatory shares. The split data from the shares **4702** may be recovered by recovery technique **4704** utilizing the appropriate recovery technique associated with the data-splitting technique used to create the shares, e.g., bit split, block split, bit segment, or bit scatter as described herein. Data recovered by recovery technique **4704** may include the AES-encrypted data and the initialization vector.

If a transform such as a package transform or AoNT was used to protect the original session key, the session key **4708** may be recovered from the key **4706** by reversing the

transform process. In the example of a package transform, the encrypted data portion of the recovered data from recovery technique **4704** may be split into blocks **4710** of equal length to the key **4706** and XORed with the key **4706** to create the session key **4708**. The session key **4708** and initialization vector may then be used as inputs to decrypt the data as data blocks **4712** which may be combined as data **4714**. Any padding that was originally necessary to match the data to the AES block size may be discarded. If data **4714** was pre-encrypted, decryption may be performed as described herein.

A Parsed File System (PFS) is a distributed file system which incorporates a directory, shared among file servers, which is itself parsed. The PFS directory is the clients' view of all file assets. The set of files visible to an individual client is controlled by that client's group keys. The PFS may maintain a directory that contains information about the locations of the shares (for example, the file server location) and the names of the files at the share locations that include the share data. The PFS directory may store information about files including M of N properties, URLs or other globally unique identifiers (GUID) of the shares, a key-id of the key that was used to parse the file, client access times of the key referenced by the key-id, validity check data, and any other data relating to the clients, users, files, or shares. This PFS directory of files may itself be parsed to prevent unauthorized access. The directory data may be parsed with separate, replaceable group keys to allow keys to be rescinded and replenished as necessary (for example, when there are personnel changes or suspected security breaches).

The underlying distribution of shares may be hidden from users/clients by the PFS such that files appear as a file storage system based on the PFS directory. The PFS provides an interface that allows users to access parsed files without knowledge of the location of shares. Clients working on a device may enter login credentials. Each client may have group keys associated with that user's login credentials, the group keys determining which files within the directory a client may access. An example set of group keys that might be suitable for a government application may include unclassified, secret, top secret, special 1, special 2, and special 3. Similar group key categories could be set up for various entities based on the shared interests, credentials, position, or other characteristics of the groups. A data set such as a file or a group of files may be associated with a group key. Based on the group keys available to a user, a user may be able to access files associated with that group key. For example, the PFS may display only unclassified files in the directory when the PFS is accessed by a user with only unclassified clearance. Each client may access any files that are authorized by workgroup keys held by that client. Workgroups may be tiered such that a higher classification or clearance level allows access to all lower classification or clearance levels. For example, a user with a top secret workgroup key may receive access to all top secret, secret, and unclassified files.

The PFS may also monitor events relating to file access and usage. For example, the PFS may audit any file open and close operations as well as any attempts to modify data or move data between files. The auditing process may monitor any attempts to move data from a higher security level to a lower security level. One example of a data monitoring technique is to create a signature of any data that is read from a file (e.g., a cut or copy operation). Signature generators may be available for different file types such as .txt, .wrd, .doc, .xls, .jpg, etc. through a plug-in interface. The signature may be stored by the PFS with a file's group key until all of the following are met: 1) the file is closed; 2) all processes running when the file was opened are terminated; and 3) all processes which were started after the file was opened but while there was data in the clipboard are terminated. If an attempt to move information from a higher security level to a lower security level is detected, the client may be told that the data will be sequestered for declassification review and may be asked whether they wish to continue the operation. If the user does complete the operation, the data may be sequestered for review by an appropriate party such as an administrator. An exemplary use case for the PFS is outlined below:

1) USER1 logs in to the network and is granted {USER1, SAP1, SECRET, UNCLASS} keys.

2) USER1 opens Notepad and types "The quick fox jumped over the lazy brown dog."

3) USER1 saves the file as TEST1.TXT.

4) A dialog box appears asking what secure parser group key should be used: USER1, SAP1, SECRET, or UNCLASS.

5) USER1 chooses SECRET and TEST1.TXT is saved.

6) USER2 logs in and is granted {USER2, UNCLASS} keys.

7) TEST1.TXT is not visible when USER2 opens Explorer.

8) USER3 logs in and is granted {USER3, TS_SCI SAP1, SECRET, UNCLASS} keys.

9) USER3 sees TEST1.TXT and opens it.

10) USER3 copies "quick fox" to clipboard.

11) USER3 pastes clipboard ("quick fox") to a new file.

12) USER3 saves the new file as TEST2.TXT and picks the UNCLASS group key.

13) A warning dialog box appears with the message "TEST2.TXT will be sequestered for declassification review."

14) USER3 cancels the warning—the new file is not saved.

15) USER3 saves TEST2.TXT using SAP1 key. Since SAP1 includes SECRET, which was TEST1.TXT's key, no warning is issued.

16) USER1 sees and reads TEST2.TXT (since he has the SAP1 key), but USER2 cannot.

17) USER3 copies/pastes "brown dog" to a new file and saves the new file as TEST1.TXT using key UNCLASS.

18) The declassification warning appears and USER3 OKs it.

19) ADMIN logs in and is granted all keys.

20) ADMIN receives a message to review TEST3.TXT for possible declassification.

21) ADMIN shows Details>>>File TEST3.TXT, written by USER3 using key UNCLASS. It contains "brown dog", which may have been copied from TEST1.TXT (key SECRET).

22) ADMIN OKs the save.

23) USER1, USER2, and USER3 all see and can access TEST3.TXT.

A secure parser may also be implemented at the device driver level. An exemplary device driver level implementation may be a parsed fabric, wherein specific logical unit numbers (LUNs) or drives are parsed. In the parsed fabric environment, any applications and servers may access and store files to the LUN or drive and issue commands as if the data were stored at the LUN or drive. The secure parser may access a key manager and LUN map which includes a mapping from application visible LUNs to the split physical location of the corresponding shares. The parsed fabric option may allow application and operating system software

to operate normally because the parsing is transparent to the programs which can read and write to the LUNs.

A secure parser may also be implemented at the host bus adapter (HBA) level or the switch level. HBA and switch level implementations may be transparent to the existing software and file systems. HBA and switch level implementations may offload parsing, distribution, and LUN processing from the servers or file system.

A secure parser may also be implemented within applications, such as through custom integration into the application. A secure parser may be implemented through APIs, function calls, libraries, etc., e.g., as described with respect to FIGS. **30-32** and **42-43**, to allow applications to directly or indirectly utilize the functionality of a secure parser to store, access, or communicate data.

Although a number of specific implementation points have been described, the secure parser may be implemented at any level or device, such as application servers, storage fabric switches and directors, storage arrays, tape devices, virtualization abstraction layers, remote mirroring/replication storage applications, routing and communications, backup and recovery software, storage management software and interfaces, remote storage management access points, notebook computers, storage network management software or storage MAN/WAN connectivity. For example, the secure parser may be implemented at the application level such as in database software to create hardware-independent implementations, at the SAN controller lever to create disk and application independent implementations, at the switch or router level such as in the router IOS to create hardware and software independent implementation, or at the embedded hardware or controller level for application independent implementations.

An exemplary API may provide an interface for developing applications for use with a secure parser. The API may include setup files, installer programs, dynamic link libraries, header files, standard libraries such as C++ libraries, visual C++ project files, executables, source code, a ReadMe file, a test harness program, and/or additional or similar files as necessary for a particular platform. The secure parser may utilize several data types and structures, such as those depicted in the table below:

Type	Description
Parser	Structure for holding the state and associated I/O data.
Key	Structure for holding a generic key.
Share	Structure for holding a generic buffer.
ParserParams	Structure for holding parser parameters.
HeaderInfo	Structure for holding header information.
FooterInfo	Structure for holding footer information.
EncContext	Structure for storing various encryption contexts.
SigContext	Structure for storing various signature contexts.
HashingContext	Structure for storing various hash contexts.
SplitContext	Structure for storing the split context.
ERROR_TYPE	Enumerator indicating a specific error type.
SPLIT_TYPE	Enumerator identifying the type of split algorithm to utilize.
ENC_TYPE	Enumerator identifying the encryption mode to use.
AUTH_TYPE	Enumerator identifying the authentication mode to use.
HASH_TYPE	Enumerator identifying the hashing mode to use.
TRANFORM_TYPE	Enumerator identifying the transform type to use.

-continued

Type	Description
KEY_TYPE	Enumerator indicating a specific key type.
uint8	An unsigned 8 bit integer.
uint16	An unsigned 16 bit integer.
uint32	An unsigned 32 bit integer.

As is depicted in FIGS. **48**A and **48**B, various enumerators may indicate parameters used in parsing operations such as operating modes, encryption techniques, etc. For example, encryption type **4802** options may include no encryption, AES 128 bit in CTR mode, AES 192 bit in CTR mode, AES 256 bit in CTR mode, AES 128 bit in CBC mode, AES 192 bit in CBC mode, AES 256 bit in CBC mode, or 3DES in CBC mode. Authentication type **4804** options may include a RSA-PSS technique, a DSA digital signature technique, a _generateFooters function **5230** may generate share footer information for each Message Authentication Code. Hash type **4806** options may include no hashing technique or SHA-256. Split type **4808** options may include no split, block level split as determined by the splitting technique, byte level split, bit level split, or bit scattering. Key type **4810** options may include symmetric keys, RSA public key, RSA private key, DSA public key, DSA private key, ECDSA public key, or ECDSA private key. Error type **4812** options may include no error, no master key, invalid callback, invalid parse split technique, no target set to parse, target buffer has no length, unable to restore because not enough shares are present, invalid share was set, invalid key was set, an integrity check failure occurred, or out of memory.

FIGS. **49**A and **49**B depict data structures for the parser, keys and shares. Parser data structure **4902** may include information related to the data to be split and the splitting process, such as originalShare, shares, shareCount, splitEncContext, and finalFlag. ParserParams data structure **4904** may include information relating to the parameters used in the parsing process, such as L (# of mandatory shares), N (total # of non-mandatory shares), M (# of non-mandatory shares necessary to recover data), splitType, splitEncMode, preEncMode, postEncMode, splitHashMode, preAuthMode, preEncKey, postEncKey, preAuthKey, postAuthKey, maxOrigBufSize, and maxShareBufSize.

KeyId data structure **4906** may include information for identifying a key, such as id and idLen. KeyShare data structure **4908** may include information relating to the key share data for a share, such as a data field and dataLen. Key data structure **4910** may include additional key information, including keyID (of type keyID 4906), type, data, and dataLen. Share data structure **4912** may include information relating to the status of a share, including data, dataLen, encContext, authContext, and hashContext.

FIGS. **50**A and **50**B depict data structures relating to encryption, authentication, and hashing functionality. EncContext data structure **5002** may be utilized in various other data structures, and may include information relating to the type of encryption being used and encryption contexts as described below. EncContext_AESCBC data structure **5004** may include information relating to AES-CBC encryption, including encKeyId, currentIV (initialization vector) and currentIVLen. EncContext_AESCTR data structure **5006** may include information relating to AES-CTR encryption, including encKeyId, currentIV (initialization vector) and currentIVLen.

AuthContext data structure **5008** may include information relating to the type of authentication being used and information relating to the various authentication contexts as described below. AuthContext_RSAPSS data structure **5010** may include information relating to a RSA_PS S digital signature, including a pubKeyId, privKeyId, and hashContext. AuthContext_DSA data structure **5012** may include information relating to a DSA digital signature, including a pubKeyId, privKeyId, and hashContext. AuthContext_ECSA data structure **5014** may include information relating to a ECDSA digital signature, including a pubKeyId, privKeyId, and hashContext. AuthContext_HMACSHA1 data structure **5016** may include information relating to a HMAC-SHA1 type MAC, including a macContext pointer to the OpenSSL mac context data type.

hashContext data structure **5018** may include information relating to the type of hashing being utilized, such as a hashMode and any appropriate hash contexts. An example of hash context may be HashContext_SHA data structure **5020** which may include a pointer to the OpenSSL hash context data type.

FIGS. **51**A and **51**B depict data formats including share data **5102**, post-encrypted footer **5104**, post-encrypted header **5106**, and encrypted footer **5108**. Share data **5102** may include a post-encryption key id, post-signature key id, post-encrypted header **5106**, post-encrypted footer **5104**, and post-signature data. Post-encrypted footer **5104** may include encrypted footer **5108**. Encrypted footer **5108** may include split-hash data, and pre-signature data. Post-encrypted header **5106** may include a key share, share id, L, M, N, split mode, split-encryption key, split-encryption IV, split-hash mode, original buffer, share buffer, pre-encryption key, and pre-signature key.

The secure parser API may include a function library with various functions as depicted in FIGS. **52**A, **52**B, **52**C, and **52**D. A number of functions may relate to initializing and ending parser operations. A parser_createParser function **5202** may be called to allocate a secure parser context and set default values for the parser, while a parser_destroyParser function **5204** may be passed a Parser data structure and may destroy the secure parser context, including parser parameter objects and share buffers. A parser_createKey function **5206** may construct the key structure, while a parser_destroyKey function **5208** may destroy the key. A parser_createShare function **5210** may create a share data structure for use with the secure parser, while a parser_destroyShare function **5212** may destroy a share, including all encryption and authentication keys. A parser_initParameters function **5214** may initialize the ParserParams data structure to the default parameter set, while a parser_setParameters **5216** may set any ParserParames settings for the current Parser data structure. A parser_setFinalize function **5218** may set the internal state of the Parser to finalize the output stream during the next split session.

Other functions may relate to the creation and splitting of shares, and the creation of the headers and footers associated with shares. parser_getOriginalShare function **5220** may return a share structure representing the original buffer used by the Parser. parser_getShare function **5222** may return a share containing the share structure for a share matching a share ID. parser_doSplit function **5224** may perform the secure parser split. parser_doRestore function **5226** may restore shares into reconstructed original data, while parser_getRestoreStatus function **5236** may return the status of the prior restore process. parser_generateHeaders function **5228** and parser_generateFooters function **5230** may generate header and footer information, while parser_restoreHeaders function **5232** and parser_restoreFooters function **5234** may restore header and footer information from shares.

parser_generateHeaders function **5228** may generate share header information for each share based on any meta-data that must necessarily be communicated to the secure parser when later performing a restore technique. The header information may only contain data that is available to the secure parser prior to the start of the data splitting process. The parser generateHeaders routine creates the header blocks for each share and places them within each Share buffer. A call to parser_generateHeaders may be made prior to any calls to parser_doSplit. The parser_generateHeaders function involves the following steps:

1. Split key using Shamir splitting algorithm.
2. Create a temporary buffer.
3. For each share . . .
   3.1 Include key share information in temporary buffer.
   3.2 Include share ID in temporary buffer.
   3.3 Include L, M and N in temporary buffer.
   3.4 Include split mode in temporary buffer.
   3.5 Include split-encryption key ID in temporary buffer.
   3.6 Include split-encryption IV in temporary buffer.
   3.7 Include split-hash mode in temporary buffer.
   3.8 Include original buffer size in temporary buffer.
   3.9 Include share buffer size in temporary buffer.
   3.10 Include pre-encryption key ID in temporary buffer.
   3.11 Include pre-encryption IV in temporary buffer.
   3.12 Include pre-signature key ID in temporary buffer.
   3.13 Include pre-signature IV in temporary buffer.
   3.14 Replace key share information in temporary buffer with Share i's key share.
   3.15 Replace share ID in temporary buffer with Share i's ID.
   3.16 Begin split-hash on the data in temporary buffer and store context for later.
   3.17 Encrypt temporary buffer with post-encryption key and post-encryption IV and include resulting ciphertext in Share i's output buffer. Maintain i's post-encryption context for later use.
   3.18 Begin post-signature on Share i's output buffer and store context for later.

The parser_doSplit function **5224** may split all data from the target buffer into share buffers to be later handled by the calling function. A series of steps may be taken to provide confidentiality, integrity, authenticity and data partitioning for the data that is split, before and after the data is split. These steps may include:

1. Test if pre-encrypt is on.
   If Yes,
   1.1 Encrypt target buffer under pre-encrypt key and pre-encrypt IV.
2. Test if pre-sign is on.
   If Yes,
   2.1 Sign target buffer under pre-sign key.
   2.2 Test if finalize has been set.
      If Yes,
      2.2.1 Finalize pre-signature context.
3. Do split-encryption using split-encryption context. (Should be initialized during the first call to this function.)
4. Call appropriate split algorithm . . .
5. Test if hashing is on.
   If Yes,
   5.1 For each output share . . .
      5.1.1 Continue hash for share i's buffer.

6. Test if post-encrypt is on.
   If Yes,
   6.1 For each output share . . .
6.1.1 Encrypt share i's buffer using share i's post-encryption key and post-encryption context.
7. Test if post-sign is on.
   If Yes,
   7.1 For each output share . . .
      7.1.1 Continue signature for share i's output buffer under share i's post-signature context.

parser_generateFooters function **5230** may generate share footer information for each share based on any meta-data that must necessarily be communicated to the secure parser when later performing a restore technique. The footer information may typically only available to the secure parser after the body of the data has been parsed. The parser_generateFooters routine creates the footer blocks for each share and places them within each Share buffer. parser_generateFooters may be called following all calls to parser_doSplit and parser_setFinalize. parser_generateFooters function **5230** may perform the following steps:

1. Create temporary buffer 1 for unencrypted footer information.
2. Create temporary buffer 2 for partially encrypted footer information.
3. For each share . . .
   3.1 Include pre-signature data in unencrypted buffer.
   3.2 Encrypt temporary buffer 1 using split-key and split-encryption context and include resulting ciphertext in temporary buffer 2.
   3.3 Continue split-hash data in temporary buffer 2.
   3.4 Include split-hash data in temporary buffer 2.
   3.5 Encrypt temporary buffer 2 using share i's post-encryption key and post-encryption context and place resulting ciphertext in share i's output buffer.
   3.6 Continue post-authentication signature over share i's output buffer using share i's post-signature context.
   3.7 Finalize share i's post-signature context.
   3.8 Include share i's post-signature in share i's output buffer.

parser_restoreHeaders function **5232** may restore header information for each share based on any meta-data that was stored by the secure parser for later restoration. The header information may contain data that is required for future calls to the parser_doRestore process to function properly. The parser_restoreHeaders routine may require that the header blocks for each share be placed within each Share buffer. A call to parser_restoreHeaders may be made prior to any calls to parser_doRestore. Steps of parser_restore headers function **5232** may include the following:

1. For each share . . .
   1.1 Create a temporary buffer for share i.
   1.2 Begin signature over share i's buffer using share i's post-signature key and post signature IV. Maintain post-signature context for share i.
   1.3 Decrypt share i's buffer using share i's post-encryption key and post-encryption IV and place contents into share i's temporary buffer.
   1.4 Recover key share information for share i from share i's temporary buffer.
   1.5 Test if this is share 0.
   If no,
      1.5.1 Compare share i's header information in temporary buffer with share 0's header information in temporary buffer. If there is a mismatch, return an error.
2. Recover split-encryption key, based on all key share information in each share's temporary buffer.
3. Recover parser state based on header information found in share 0's temporary buffer.

parser_doRestore function **5226** may restore all shares to their original single source. A series of steps verify that the data has not been tampered with before and after data restoration has completed. Steps for parser_doRestore function **5226** may include the following:

1. Test if post-sign is on.
   If Yes,
   1.1 For each output share . . .
      1.1.1 Sign share i's output buffer under share i's post-signature key.
2. Test if post-encrypt is on.
   If Yes,
   2.1 For each output share . . .
      2.1.1 Decrypt share i's buffer using key post-decrypt i and IV i.
3. Test if hashing is on.
   If Yes,
   3.1 For each output share . . .
      3.1.1 Continue hash for share i's buffer.
4. Call appropriate restore algorithm.
5. Do split-decryption using split-decryption key and split-decryption context. (Should be initialized during first call to this function.)
6. Test if pre-sign is on.
   If Yes,
   6.1 Sign original buffer under pre-sign key using pre-signature context or pre-signature IV.
   6.2 Test if finalize has been set.
      If Yes,
      6.2.1 Finalize pre-signature context.
7. Test if pre-encrypt is on.
   If Yes,
   7.1 Decrypt original buffer under pre-decryption key.

parser_restoreFooters function **5234** may restore footer information for each share based on any meta-data that was stored by the secure parser for later restoration. The footer information may contain data that is needed for valid completion of the restore process. The parser_restoreFooters routine may require that the footer blocks for each share be placed within each Share buffer. A call to parser_restoreFooters may be after all calls to parser_doRestore and parser_setFinalize. Steps for parser_restoreFooters function **5234** may include the following:

1. Create a temporary pre-signature buffer.
2. For each share . . .
   2.1 Create a temporary buffer for share i.
   2.2 Continue post-signature over share i's buffer using share i's post-signature key and post-signature context.
   2.3 Finalize share i's post-signature context.
   2.4 Compare share i's post-signature result with expected post-signature result and return an error on mismatch.
   2.5 Decrypt share i's buffer using share i's post-decryption key and post-decryption context and place contents into share i's temporary buffer.
   2.6 Continue split-hash over share i's encrypted footer portion of share i's temporary buffer using share i's split-hash context.
   2.7 Finalize share i's split hash context.
   2.8 Compare share i's split-hash result with expected split-hash result found in remaining portion of share i's temporary buffer and return an error on mismatch.

2.9 Decrypt the footer portion of share i's temporary buffer using the split-decryption key and split-decryption context and place resulting plaintext in to the temporary pre-signature buffer.

2.10 Compare temporary pre-signature buffer with finalized pre-signature result and return an error on mismatch.

FIG. **53** depicts an exemplary implementation of a secure parser within a hardware system such as a data storage control node. The secure parser may operate on a processor **5302**. The secure parser may be in communication with various forms of internal memory such as SDRAM (e.g., **5304**), hard drives (e.g., **5310**), and flash memory (e.g., **5306**). The secure parser may also be in communication with other internal circuitry and functionality and may also communicate over networks, e.g., utilizing Ethernet connectivity **5308**.

FIG. **54** depicts an exemplary implementation of a secure parser for data masking. It may be desirable to store certain information of a set of information securely. For example, a healthcare record for an individual may include less sensitive information such as name, address, height, and weight. The healthcare record may also include sensitive information such as a social security number **5404**, medical ID number **5406**, and employer phone number **5408**. The less sensitive information may be stored locally, e.g., at local server **5402**. The sensitive information **5404**, **5406**, and **5408** may be parsed and distributed to multiple locations, e.g., to local server **5402** and remote servers **5410** and **5412**.

FIG. **55** depicts an exemplary data record in a data masking implementation. As described above, the local server may not have local access to records that have been parsed and distributed to multiple locations, and thus the sensitive records **5502**, **5504**, and **5506** may not be visible. To the extent that local information **5508** relating to the sensitive records exists, it may not be in a human readable form.

An exemplary set of workgroup keys is depicted in FIG. **56**. For example, four groups may include top secret/sensitive compartmented information (TS/SCI) **5602**, secret **5604**, secret SAP **5606**, and coalition **5608**. Each group may include a plurality of users who are associated with that group. When a user logs into the system from a device, the login may include credentials, username, password, or other authenticating information that provides access to one or more workgroup keys associated with one or more of workgroups **5602**, **5604**, **5606**, and **5608**.

Accordingly, if the login credentials match a member of an appropriate group, the user may access parsed files, drives, and data associated with that workgroup, or in some embodiments any information associated with groups having a lower classification.

FIG. **57** depicts the use of workgroup keys in an example application of military use. Command operations headquarters **5702**, deployed headquarters **5706**, tactical team **5708**, and tactical base **5710** may all have access to information stored as described herein via a satellite link **5704**. Data may be stored at one or more of locations **5702**, **5706**, **5708**, and/or **5710**, as well as additional offsite storage locations. Data may be stored in distributed shares as described herein and/or may be transmitted on satellite link **5704** as shares as described herein. Individuals located at locations **57025706**, **5708**, and/or **5710** may submit login credentials to gain access to information associated with one or more workgroup keys TS/SCI, secret, secret SAP, or coalition.

Workgroup keys may be implemented as in a MLS solution as depicted in FIG. **58**. Example workgroup keys may include TS/SCI **5802**, secret **5804**, coalition **5806**, and unclassified **5808**. The types, names, and number of workgroups are exemplary only, and may be modified based on the particular application. Workgroup keys may be managed and stored as described herein (e.g., locally, remotely, in shares). Clients may access information associated with a workgroup through a parser server **5812**.

Parser server **5812** may include one or more servers, and may include client partition **5814**, TS/SCI partition **5816**, secret partition **5818**, coalition partition **5820**, and unclassified partition **5822**. Partitions may be based on workgroup keys and may be logical, or as in the example of FIG. **58**, may be physical. Client-facing aspects of the parser server may be implemented at client partition **5814**. Client partition **5814** may also interface with local storage area network **5824**, which may provide storage of information for parser server **5812**.

A secure parser may support backup applications. For example, a conventional file system may be maintained while a backup of the files is parsed to multiple backup locations in accordance with the present invention. For backup applications parsing may provide additional security as well as redundancy through M on N parsing. A backup application may also utilize one or more mandatory shares while maintaining M of N parsing for the non-mandatory shares. It may also be possible to use public key cryptography to further secure backup data. Any storage operations for backup data may utilize a public key to encrypt the data. Only the holder of the private key may recover the backup data.

A secure parser may also be used for secure deletion of data. When one or more mandatory shares are utilized in the splitting and distribution process, the underlying data may not be recoverable without all of the mandatory shares. The permanent deletion of any mandatory shares may therefore result in the effective deletion of the underlying data.

A secure parser may also be used to create flexible security policies. For example, utilizing an M-of-N data split, a backup operator may produce N shares (e.g., as backup tapes), with each accessible to distinct individuals or departments. A quorum of M individuals may therefore be required to combine their shares to reconstruct data.

An embodiment of the present invention for a backup application is depicted at FIG. **59**. Server **5902** and server **5904** may be MSCS clustered servers running an application. Servers **5902** and **5904** may each include a LUN map associating a LUN with shares used for storage of parsed drives, files, or other data. Each of server **5902** and **5904** may be in communication with key manager **5906** to access keys for encryption and parsing of data. Parser enabled HBA **5908** may implement parsing, distribution, recovery, and access of parsed data to shares **5912** and **5914** as provided in the LUN map of server **5902**. Communication between HBA **5908** and shares **5912** and **5914** may be implemented through storage area network (SAN) **5920**. Parser enabled HBA **5910** may implement parsing, distribution, recovery, and access of parsed data to shares **5916** and **5918** as provided in the LUN map of server **5904**. Communication between HBA **5910** and shares **5916** and **5918** may be implemented through SAN **5922**.

Backup protection may be provided by SafeGuard appliance **5924** and SafeGuard appliance **5926**. As depicted in FIG. **59**, SafeGuard appliance **5924** may capture parsed data that is written to shares **5912** and **5914** and route complete copies of the parsed data for storage at shares **5916** and **5918** through SafeGuard appliance **5926** and SAN **5922**. Similarly, SafeGuard appliance **5926** may capture parsed data

that is written to shares **5916** and **5918** and route complete copies of the parsed data for storage at shares **5912** and **5914** through SafeGuard appliance **5924** and SAN **5920**. In the event of a disaster or other event, the data split between shares **5912** and **5914** may be fully recovered from shares **5916** and **5918**. Similarly, the data split between shares **5916** and **5918** may be fully recovered from shares **5912** and **5914**.

An embodiment of the present invention for backup applications is depicted at FIG. **60**. Server **6002** and server **6004** may be MSCS clustered servers running an application. Server **6002** may be in communication with parser enabled switch **6006** and server **6004** may be in communication with parser enabled switch **6008**. Each of parser enabled switch **6006** and **6008** may be in communication with key manager/LUN map **6010**. Key manager/LUN map **6010** may provide access to keys for encryption and parsing of data, and a LUN map associating a LUN with shares used for storage of parsed drives, files, or other data. Parser enabled switch **6006** may implement parsing, distribution, recovery, and access of parsed data to shares **6012** and **6014** as provided in the key manager/LUN map **6010**. Communication between parser enabled switch **6006** and shares **6012** and **6014** may be implemented through storage area network (SAN) **6020**. Parser enabled switch **6008** may implement parsing, distribution, recovery, and access of parsed data to shares **6016** and **6018** as provided in the key manager/LUN map **6010**. Communication between parser enabled switch **6008** and shares **6016** and **6018** may be implemented through SAN **6022**.

Disaster recovery protection may be provided by SafeGuard appliance **6024** and SafeGuard appliance **6026**. As depicted in FIG. **60**, SafeGuard appliance **6024** may capture parsed data that is written to shares **6012** and **6014** and route complete copies of the parsed data for storage at shares **6016** and **6018** through SafeGuard appliance **6026** and SAN **6022**. Similarly, SafeGuard appliance **6026** may capture parsed data that is written to shares **6016** and **6018** and route complete copies of the parsed data for storage at shares **6012** and **6014** through SafeGuard appliance **6024** and SAN **6020**. In the event of a disaster or other event, the data split between shares **6012** and **6014** may be fully recovered from shares **6016** and **6018**. Similarly, the data split between shares **6016** and **6018** may be fully recovered from shares **6012** and **6014**.

A secure parser may include a variety of techniques such as FIPS standard techniques for managing keys. These include systems for encrypting key data, use of protected memory for key material, and "split-knowledge" techniques which allow keys to be safely divided among multiple participants. Authenticated key agreement techniques may also be provided to rapidly establish connections between secure parser modules in a manner that is compliant with industry standards such as IPSEC, TLS and SSL.

A secure data parser may communicate with the electronic storage locally or remotely. The secure data parser may be implemented through any network or communication protocol such as IPV4 and IPV6. Shares used to reconstruct files, file systems, or any other data distributed over shares may be accessed over one or more transmission medium, including physical delivery of a drive, disk, or other storage medium, delivery via physical networks such as twisted pair, coaxial cable, optical fiber, or wireless delivery such as via AM radio, shortwave, FM/TV broadcast, cellular technology, satellites, microwave, terrestrial, or any other known wireless medium.

A secure parser may be implemented in any number of environments, from servers to desktops, cell phones to PDAs, and dedicated appliances to smartcards.

Pricing for usage of the secure parser may be accomplished in a number of ways. For example, a user could be charged based on storage or data processing (e.g., throughput), e.g., on a per-Gigabyte basis. Pricing could be on an enterprise bases, server, basis, client basis, or site license basis. Charges may also be on a percentage of effort on value basis, e.g., based on a measure of the security improvement in the system.

Additionally, other combinations, additions, substitutions and modifications will be apparent to the skilled artisan in view of the disclosure herein. Accordingly, the present invention is not intended to be limited by the reaction of the preferred embodiments but is to be defined by a reference to the appended claims.

What is claimed is:

1. A method for securing a data set, the method comprising:

distributing the data set into a plurality of data chunks, wherein none of the data chunks are, by themselves, sufficient to reconstruct the data set;

encrypting each of the data chunks with a respective one of a plurality of encryption keys;

obfuscating each of the plurality of different encryption keys; and

separately storing each data chunk of the plurality of data chunks together with one of the plurality of obfuscated different encryption keys on a plurality of different storage devices.

2. The method of claim **1**, wherein obfuscating each of the plurality of different encryption keys comprises performing a transform on data indicative of at least one of the plurality of different encryption keys.

3. The method of claim **2**, wherein performing a transform on data indicative of at least one of the plurality of different encryption keys comprises performing an all-or-nothing transform on the data indicative of at least one of the plurality of different encryption keys.

4. The method of claim **1**, wherein obfuscating each of the plurality of different encryption keys comprises applying Shamir's Secret Sharing algorithm to data indicative of at least one of the plurality of different encryption keys.

5. The method of claim **1**, wherein storing the plurality of data chunks together with the plurality of obfuscated different encryption keys comprises storing each of the plurality of data chunks together with a respective one of the plurality of obfuscated different encryption keys.

6. The method of claim **1**, wherein storing the plurality of data chunks together with the plurality of obfuscated different encryption keys comprises including, with a data chunk of the plurality of data chunks, data indicative of an encryption key that was used to encrypt a different data chunk of the plurality of data chunks.

7. The method of claim **1**, wherein distributing the data set into a plurality of data chunks comprises using a random technique or pseudorandom technique.

8. The method of claim **1**, further comprising generating the data set by encrypting primary data.

9. A method for securing a data set, the method comprising:

distributing the data set into a plurality of data chunks, wherein none of the data chunks are, by themselves, sufficient to reconstruct the data set;

encrypting each of the data chunks with a respective one of a plurality of encryption keys;

transforming each of the plurality of different encryption keys; and

separately storing each data chunk of the plurality of data chunks together with one of the plurality of transformed different encryption keys on a plurality of different storage devices.

10. The method of claim **9**, wherein transforming each of the plurality of different encryption keys comprises performing an all-or-nothing transform on data indicative of at least one of the plurality of different encryption keys.

11. The method of claim **9**, wherein storing the plurality of data chunks with the plurality of transformed different encryption keys comprises storing each of the plurality of data chunks with a respective one of the plurality of transformed different encryption keys.

12. The method of claim **9**, wherein storing the plurality of data chunks with the plurality of transformed different encryption keys comprises including, with a data chunk of the plurality of data chunks, data indicative of an encryption key of the plurality of encryption keys that was used to encrypt a different data chunk of the plurality of data chunks.

13. The method of claim **9**, wherein distributing the data set into a plurality of data chunks comprises using a random technique or pseudorandom technique.

14. The method of claim **9**, further comprising generating the data by encrypting primary data.

15. A computer system for securing a data set, the system comprising:

at least one hardware processor, configured to:

access the data set;

distribute the data set into a plurality of data chunks, wherein none of the data chunks are, by themselves, sufficient to reconstruct the data set;

encrypt each of the data chunks with a respective one of a plurality of encryption keys;

obfuscate each of the plurality of different encryption keys; and

storing the plurality of data chunks together with the plurality of obfuscated different encryption keys;

separately store each data chunk of the plurality of data chunks together with one of the plurality of obfuscated different encryption keys, wherein the plurality of data chunks is stored on a plurality of different storage devices.

16. The computer system of claim **15**, wherein, when obfuscating each of the plurality of different encryption keys, the at least one hardware processor is configured to perform a transform on data indicative of at least one of the plurality of different encryption keys.

17. The computer system of claim **16**, wherein, when performing a transform on data indicative of at least one of the plurality of different encryption keys, the at least one hardware processor is configured to perform an all-or-nothing transform on the data indicative of at least one of the plurality of different encryption keys.

18. The computer system of claim **15**, wherein, when obfuscating each of the plurality of different encryption keys, the at least one hardware processor is configured to apply Shamir's Secret Sharing algorithm to data indicative of at least one of the plurality of different encryption keys.

19. The computer system of claim **15**, wherein, when storing the plurality of data chunks together with the plurality of obfuscated different encryption keys, the at least one hardware processor is configured to store each of the plurality of data chunks together with a respective one of the plurality of obfuscated different encryption keys.

20. The computer system of claim **15**, wherein, when storing the plurality of data chunks together with the plurality of obfuscated different encryption keys, the at least one hardware processor is configured to include, with a data chunk of the plurality of data chunks, data indicative of an encryption key that was used to encrypt a different data chunk of the plurality of data chunks.

21. A computer system for securing a data set, the system comprising:

at least one hardware processor, configured to:

access the data set;

distribute the data set into a plurality of data chunks, wherein none of the data chunks are, by themselves, sufficient to reconstruct the data set;

encrypt each of the data chunks with a respective one of a plurality of encryption keys;

transform each of the plurality of different encryption keys; and

storing the plurality of data chunks together with the plurality of transformed different encryption keys;

separately store each data chunk of the plurality of data chunks together with one of the plurality of transformed different encryption keys, wherein the plurality of data chunks is stored on a plurality of different storage devices.

22. The computer system of claim **21**, wherein, when transforming each of the plurality of different encryption keys, the at least one hardware processor is configured to perform an all-or-nothing transform on data indicative of at least one of the plurality of different encryption keys.

23. The computer system of claim **21**, wherein, when storing the plurality of data chunks with the plurality of transformed different encryption keys, the at least one hardware processor is configured to store each of the plurality of data chunks with a respective one of the plurality of transformed different encryption keys.

24. The computer system of claim **21**, wherein, when storing the plurality of data chunks with the plurality of transformed different encryption keys, the at least one hardware processor is configured to include, with a data chunk of the plurality of data chunks, data indicative of an encryption key of the plurality of encryption keys that was used to encrypt a different data chunk of the plurality of data chunks.

* * * * *