

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

SAMSUNG ELECTRONICS CO., LTD.; AND
SAMSUNG ELECTRONICS AMERICA, INC.

Petitioner

v.

DAEDALUS PRIME LLC
Patent Owner.

**DECLARATION OF DR. MARK E. CROVELLA IN SUPPORT OF
PETITION FOR *INTER PARTES* REVIEW OF
U.S. PATENT NO. 9,887,838**

TABLE OF CONTENTS

	Page
I. INTRODUCTION	1
II. BACKGROUND AND QUALIFICATIONS	1
III. INFORMATION CONSIDERED	5
IV. RELEVANT LEGAL STANDARDS	6
A. Claim Interpretation	6
B. Perspective of One of Ordinary Skill in the Art.....	7
C. Obviousness.....	7
V. LEVEL OF ORDINARY SKILL IN THE ART	10
VI. SUMMARY OF MY OPINIONS	11
VII. TECHNOLOGICAL BACKGROUND	12
A. Secure Key Exchange Algorithms	12
1. Symmetric Encryption v. Asymmetric Encryption	12
2. Diffie-Hellman Key Exchange Algorithm.....	14
3. RSA Encryption	18
B. SSL/TLS Protocol	20
C. Digital Signature.....	26
VIII. THE '838 PATENT AND ITS FILE HISTORY	28
A. Overview of the '838 Patent.....	28
B. Challenged Claims	32
C. Prosecution History of the '838 Patent	32
D. Priority Date	34
E. Claim Construction	34
IX. OVERVIEW OF THE APPLIED PRIOR ART REFERENCES	34
A. United States Patent Application Publication No. 2011/0191599 ("Chou") (Ex-1008).....	34
B. U.S. Patent No. 7,773,754 ("Buer") (Ex-1009)	37
C. TLS-1.0.....	40

TABLE OF CONTENTS
(continued)

	Page
X. DETAILED EXPLANATION OF THE UNPATENTABILITY GROUNDS	41
A. Ground 1: Chou in view of knowledge of a POSITA (Claims 1, 9, 17, 24).....	42
1. Independent Claim 1	42
a. 1[pre]: A system-on-a-chip apparatus comprising:	42
b. 1[a] a system-on-a-chip comprising a security engine that is separate from a processor core of the system-on-a-chip and has a secure memory accessible only by the security engine, wherein the secure memory includes a security key that was encoded in the secure memory during a manufacturing process of the system-on-a-chip, the security engine to:.....	43
c. 1[b] generate a random nonce for initiating a request for a secure communication session with a remote server over a network using the nonce;	52
d. 1[c] perform a cryptographic key exchange with the remote server;	56
e. 1[d] generate a symmetric session key, based on the cryptographic key exchange, to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session;	59
f. 1[e] encrypt the symmetric session key based on the security key; and.....	63
g. 1[f] store the encrypted session key in the secure memory, ...	66
h. 1[g] the system-on-a-chip to establish the secure communication session with the remote server over the network using the session key.	69
2. Claim 24	70

TABLE OF CONTENTS
(continued)

	Page
a. 24. The system-on-a-chip apparatus of claim 1, wherein the security engine is communicatively coupled to the processor core through an interconnect of the system-on-a-chip.	70
3. Independent Claims 9, 17	72
a. 9[pre] A method comprising;. 17[pre] One or more non-transitory machine-readable storage media comprising a plurality of instructions stored thereon that, in response to execution by a computing device, cause the computing device to:	72
b. 9[a] generating a random nonce in a security engine that is separate from a processor core of a system-on-a-chip of a client device; 17[a] generate a random nonce in a security engine that is separate from a processor core of a system-on-a-chip of the computing device;	74
c. 9[b] initiating, using the client device, a request for a secure communication session with a remote server over a network, the request including the random nonce; 17[b] initiate a request for a secure communication session with a remote server over a network, the request including the random nonce;.....	74
d. 9[c] performing a cryptographic key exchange , using the security engine of the system-on-a-chip, with the remote server; 17[c] perform a cryptographic key exchange, using the security engine of the system-on-a-chip, with the remote server;	75

TABLE OF CONTENTS

(continued)

Page

e.	9[d] generate a symmetric session key to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session; 17[d] generate, based on the cryptographic key exchange, a symmetric session key to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session;.....	75
f.	9[e] encrypting the session key based on a security key that was encoded in a secure memory of the security engine during a manufacturing process of the system-on-a-chip; 17[e] encrypt the session key based on a security key that was encoded in a secure memory of the security engine during a manufacturing process of the system-on-a-chip;.....	76
g.	9[f] storing the encrypted session key in the secure memory of the security engine of the system-on-a-chip; and 17[f] store the session key in the secure memory of the security engine of the system-on-a-chip; and.....	76
h.	9[g] establishing, using the client device, the secure communication session with the remote server using the session key. 17[g] establish the secure communication session with the remote server using the session key.....	77
B.	Ground 2: Chou, in combination with TLS-1.0	77
1.	Motivation to Combine	77
2.	Dependent Claim 3	80
a.	3. The system-on-a-chip apparatus of claim 1, wherein the security engine is further to generate a pre-master key, encrypt the pre-master key using the security key of the security engine, and send the encrypted pre-master key to the remote server.....	80

TABLE OF CONTENTS

(continued)

	Page
3. Dependent Claim 4	82
a. 4. The system-on-a-chip apparatus of claim 3 wherein the security engine is further to wrap the pre-master key using a server public key and store the wrapped pre-master key in the secure memory.	82
4. Dependent Claim 5	86
a. 5[pre]/5[a]: The system-on-a-chip apparatus of claim 3, wherein the security engine is further to: generate the session key as a function of the pre- master key	86
b. 5[b] calculate a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server	88
5. Dependent Claim 6	91
a. 6[pre]/6[a]. The system-on-a-chip apparatus of claim 1, wherein the security engine is further to generate a public Diffie-Hellman client key and a private Diffie-Hellman client key, ...	91
b. 6[b] sign the public Diffie-Hellman client key using the security key of the security engine, ...	95
c. 6[c] send the signed public Diffie-Hellman key to the remote server	99
6. Dependent Claim 7	100
a. 7[a] The system-on-a-chip apparatus of claim 6, wherein the security engine is further to receive a public Diffie-Hellman server key from the remote server and	100
b. 7[b] generate the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key.	101
7. Dependent Claims 10, 18	104

TABLE OF CONTENTS
(continued)

	Page
a. 10[pre]. The method of claim 9, wherein initiating a request for a secure communication session comprises sending a ClientHello message, including the nonce, to the remote server; and further comprising: 18[pre]. The one or more non-transitory machine-readable storage media of claim 17, wherein to initiate a request for a secure communication session comprises to send a ClientHello message, including the nonce, to the remote server; and.....	104
b. 10[a] receiving a ServerHello message from the remote server, the ServerHello message including a server nonce; and 18[a] wherein the plurality of instructions further cause the computing device to: receive a ServerHello message from the remote server, the ServerHello message including a server nonce; and.....	106
c. 10[b] storing the server nonce in the secure memory of the security engine of the system-on-a-chip. 18[b] store the server nonce in the secure memory of the security engine of the system-on-a-chip.....	108
8. Dependent Claims 11, 19.....	113

TABLE OF CONTENTS

(continued)

	Page
a. 11 The method of claim 9, wherein performing the cryptographic key exchange with the remote server comprises: generating a pre-master key in the security engine; encrypting the pre-master key using the security key of the security engine; and sending the encrypted pre-master key to the remote server. 19 The one or more non-transitory machine-readable storage media of claim 17, wherein to perform the cryptographic key exchange with the remote server comprises to: generate a pre-master key in the security engine encrypt the pre-master key using the security key of the security engine; and send the encrypted pre-master key to the remote server.....	113
9. Dependent Claims 12, 20.....	114
a. 12[pre]/12[a]: The method of claim 11, further comprising: wrapping the pre-master key in the security engine of the system-on-a-chip using a server public key, and 12[b] storing the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip. 20[pre]/20[a]: The one or more non-transitory machine-readable storage media of claim 19, wherein the plurality of instructions further cause the computing device to: wrap the pre-master key in the security engine of the system-on-a-chip with a server public key, and 20[b] store the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip.....	114
10. Dependent Claims 13, 21	115

TABLE OF CONTENTS
(continued)

	Page
a. 13. The method of claim 11, wherein performing the cryptographic key exchange with the remote server comprises generating the session key in the security engine of the system-on-a-chip as a function of the pre-master key. 21. The one or more non-transitory machine-readable storage media of claim 19, wherein to perform the cryptographic key exchange with the remote server comprises to generate the session key in the security engine of the system-on-a-chip as a function of the pre-master key.....	115
11. Dependent Claims 14, 22	116
a. 14. The method of claim 13, wherein generating the session key in the security engine comprises calculating, using the security engine of the system-on-a-chip, a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server. 22. The one or more non-transitory machine-readable storage media of claim 21, wherein to generate the session key in the security engine comprises to calculate, using the security engine of the system-on-a-chip, a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server.....	116
12. Claims 15, 23	117

TABLE OF CONTENTS

(continued)

Page

a.	15[pre]/15[a] The method of claim 9 wherein performing the cryptographic key exchange with the remote server comprises: generating, in the security engine of the system-on-a-chip, a public Diffie-Hellman client key and a private Diffie-Hellman client key; 23[pre]/23[a] The one or more non-transitory machine-readable storage media of claim 17, wherein to perform the cryptographic key exchange with the remote server comprises to: sign the public Diffie-Hellman client key using the security key of the security engine, and.....	117
b.	15[b] signing the public Diffie-Hellman client key using the security key of the security engine of the system-on-a-chip; 23[b] sign the public Diffie-Hellman client key using the security key of the security engine of the system-on-a-chip;.....	117
c.	15[c] sending the signed public Diffie-Hellman key to the remote server; 23[c] send the signed public Diffie-Hellman key to the remote server;.....	118
d.	15[d] receiving a public Diffie-Hellman server key from the remote server; and 23[d] receive a public Diffie-Hellman server key from the remote server; and.....	118
e.	15[e] generating, in the security engine of the system-on-a-chip, the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key. 23[e] generate, in the security engine of the system-on-a-chip, the session key as a function of the private DiffieHellman client key and the public Diffie-Hellman server key.....	118
C.	Grounds 3 and 4: Buer, in view of knowledge of a POSITA or in view of Chou (Claims 1, 3-7, 9-15, 17-24).....	119

TABLE OF CONTENTS

(continued)

	Page
1. Motivation to Combine Buer with Chou	121
2. Independent Claim 1	124
a. 1[pre]: A system-on-a-chip apparatus comprising:	124
b. 1[a] a system-on-a-chip comprising a security engine that is separate from a processor core of the system-on-a-chip and has a secure memory accessible only by the security engine, wherein the secure memory includes a security key that was encoded in the secure memory during a manufacturing process of the system-on-a-chip, the security engine to:.....	126
c. 1[b] generate a random nonce for initiating a request for a secure communication session with a remote server over a network using the nonce;	129
d. 1[c] perform a cryptographic key exchange with the remote server;	134
e. 1[d] generate a symmetric session key, based on the cryptographic key exchange, to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session;	135
f. 1[e] encrypt the symmetric session key based on the security key; and	140
g. 1[f] store the encrypted session key in the secure memory, ...	142
h. 1[g] the system-on-a-chip to establish the secure communication session with the remote server over the network using the session key.	148
3. Dependent Claim 3	149

TABLE OF CONTENTS

(continued)

	Page
a. 3. The system-on-a-chip apparatus of claim 1, wherein the security engine is further to generate a pre-master key, encrypt the pre-master key using the security key of the security engine, and send the encrypted pre-master key to the remote server.....	149
4. Dependent Claim 4	152
a. 4. The system-on-a-chip apparatus of claim 3 wherein the security engine is further to wrap the pre-master key using a server public key and store the wrapped pre-master key in the secure memory.	152
5. Dependent Claim 5	158
a. 5[pre]/5[a]: The system-on-a-chip apparatus of claim 3, wherein the security engine is further to: generate the session key as a function of the pre-master key	158
b. 5[b] calculate a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server.....	159
6. Dependent Claim 6	163
a. 6[pre]/6[a]. The system-on-a-chip apparatus of claim 1, wherein the security engine is further to generate a public Diffie-Hellman client key and a private Diffie-Hellman client key,.....	163
b. 6[b] sign the public Diffie-Hellman client key using the security key of the security engine,.....	164
c. 6[c] send the signed public Diffie-Hellman key to the remote server.....	165
7. Dependent Claim 7	165
a. 7[a] The system-on-a-chip apparatus of claim 6, wherein the security engine is further to receive a public Diffie-Hellman server key from the remote server and.....	165

TABLE OF CONTENTS

(continued)

	Page
b. 7[b] generate the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key.	167
8. Claim 24	168
a. 24. The system-on-a-chip apparatus of claim 1, wherein the security engine is communicatively coupled to the processor core through an interconnect of the system-on-a-chip.	168
9. Independent Claims 9, 17	170
a. 9[pre] A method comprising; 17[pre] One or more non-transitory machine-readable storage media comprising a plurality of instructions stored thereon that, in response to execution by a computing device, cause the computing device to:	170
b. 9[a] generating a random nonce in a security engine that is separate from a processor core of a system-on-a-chip of a client device; 17[a] generate a random nonce in a security engine that is separate from a processor core of a system-on-a-chip of the computing device;	171
c. 9[b] initiating, using the client device, a request for a secure communication session with a remote server over a network, the request including the random nonce; 17[b] initiate a request for a secure communication session with a remote server over a network, the request including the random nonce;.....	172
d. 9[c] performing a cryptographic key exchange, using the security engine of the system-on-a-chip, with the remote server; 17[c] perform a cryptographic key exchange, using the security engine of the system-on-a-chip, with the remote server;	172

TABLE OF CONTENTS
(continued)

	Page
e. 9[d] generate a symmetric session key to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session; 17[d] generate, based on the cryptographic key exchange, a symmetric session key to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session;.....	173
f. 9[e] encrypting the session key based on a security key that was encoded in a secure memory of the security engine during a manufacturing process of the system-on-a-chip; 17[e] encrypt the session key based on a security key that was encoded in a secure memory of the security engine during a manufacturing process of the system-on-a-chip;.....	173
g. 9[f] storing the encrypted session key in the secure memory of the security engine of the system-on-a-chip; and 17[f] store the session key in the secure memory of the security engine of the system-on-a-chip; and.....	174
h. 9[g] establishing, using the client device, the secure communication session with the remote server using the session key. 17[g] establish the secure communication session with the remote server using the session key.....	174
10. Dependent Claims 10, 18.....	175

TABLE OF CONTENTS
(continued)

	Page
a. 10[pre]. The method of claim 9, wherein initiating a request for a secure communication session comprises sending a ClientHello message, including the nonce, to the remote server; and further comprising: 18[pre]. The one or more non-transitory machine-readable storage media of claim 17, wherein to initiate a request for a secure communication session comprises to send a ClientHello message, including the nonce, to the remote server; and.....	175
b. 10[a] receiving a ServerHello message from the remote server, the ServerHello message including a server nonce; and 18[a] wherein the plurality of instructions further cause the computing device to: receive a ServerHello message from the remote server, the ServerHello message including a server nonce; and.....	177
c. 10[b] storing the server nonce in the secure memory of the security engine of the system-on-a-chip. 18[b] store the server nonce in the secure memory of the security engine of the system-on-a-chip.....	179
11. Dependent Claims 11, 19.....	182
a. 11[pre]/11[a]. “The method of claim 9, wherein performing the cryptographic key exchange with the remote server comprises: generating a pre-master key in the security engine” 19[pre]/19[a]. “The one or more non-transitory machine-readable storage media of claim 17, wherein to perform the cryptographic key exchange with the remote server comprises to: generate a pre-master key in the security engine”	182

TABLE OF CONTENTS

(continued)

	Page
b. 11[b] “encrypting the pre-master key using the security key of the security engine” 19[b] “encrypt the pre-master key using the security key of the security engine	183
c. 11[c] “sending the encrypted pre-master key to the remote server” 19[c] “send the encrypted pre-master key to the remote server”	183
12. Dependent Claims 12, 20	183
a. 12[pre]/12[a]: The method of claim 11, further comprising: wrapping the pre-master key in the security engine of the system-on-a-chip using a server public key, and 20[pre]/20[a]: The one or more non-transitory machine-readable storage media of claim 19, wherein the plurality of instructions further cause the computing device to: wrap the pre-master key in the security engine of the system-on-a-chip with a server public key, and	183
b. 12[b] storing the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip. 20[b] store the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip.....	184
13. Dependent Claims 13, 21	184
a. 13. The method of claim 11, wherein performing the cryptographic key exchange with the remote server comprises generating the session key in the security engine of the system-on-a-chip as a function of the pre-master key. 21. The one or more non-transitory machine-readable storage media of claim 19, wherein to perform the cryptographic key exchange with the remote server comprises to generate the session key in the security engine of the system-on-a-chip as a function of the pre-master key.....	184

TABLE OF CONTENTS
(continued)

	Page
14. Dependent Claims 14, 22	185
a. 14. The method of claim 13, wherein generating the session key in the security engine comprises calculating, using the security engine of the system-on-a-chip, a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server. 22. The one or more non-transitory machine-readable storage media of claim 21, wherein to generate the session key in the security engine comprises to calculate, using the security engine of the system-on-a-chip, a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server.	185
15. Claims 15, 23	186
a. 15[pre]/15[a] The method of claim 9 wherein performing the cryptographic key exchange with the remote server comprises: generating, in the security engine of the system-on-a-chip, a public Diffie-Hellman client key and a private Diffie-Hellman client key; 23[pre]/23[a] The one or more non-transitory machine-readable storage media of claim 17, wherein to perform the cryptographic key exchange with the remote server comprises to: sign the public Diffie-Hellman client key using the security key of the security engine, and.....	186
b. 15[b] signing the public Diffie-Hellman client key using the security key of the security engine of the system-on-a-chip; 23[b] sign the public Diffie-Hellman client key using the security key of the security engine of the system-on-a-chip;.....	186

TABLE OF CONTENTS

(continued)

	Page
c. 15[c] sending the signed public Diffie-Hellman key to the remote server; 23[c] send the signed public Diffie-Hellman key to the remote server;..... d. 15[d] receiving a public Diffie-Hellman server key from the remote server; and 23[d] receive a public Diffie-Hellman server key from the remote server; and..... e. 15[e] generating, in the security engine of the system-on-a-chip, the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key. 23[e] generate, in the security engine of the system-on-a-chip, the session key as a function of the private DiffieHellman client key and the public Diffie-Hellman server key.....	187
D. Grounds 5 and 6: Grounds 3 and 4, in further combination with TLS-1.0.....	188
1. Motivation to Combine	188
2. Dependent Claim 3	191
a. 3. The system-on-a-chip apparatus of claim 1, wherein the security engine is further to generate a pre-master key, encrypt the pre-master key using the security key of the security engine, and send the encrypted pre-master key to the remote server.....	191
3. Dependent Claim 4	192
a. 4. The system-on-a-chip apparatus of claim 3 wherein the security engine is further to wrap the pre-master key using a server public key and store the wrapped pre-master key in the secure memory.	192
4. Dependent Claim 5	193

TABLE OF CONTENTS
(continued)

	Page
a. 5[pre]/5[a]: The system-on-a-chip apparatus of claim 3, wherein the security engine is further to: generate the session key as a function of the pre-master key 5[b] calculate a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server	193
5. Dependent Claim 7	195
7[a] The system-on-a-chip apparatus of claim 6, wherein the security engine is further to receive a public Diffie-Hellman server key from the remote server and 7[b] generate the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key.	195
6. Dependent Claims 10, 18.....	197
a. 10[pre]. The method of claim 9, wherein initiating a request for a secure communication session comprises sending a ClientHello message, including the nonce, to the remote server; and further comprising: 18[pre]. The one or more non-transitory machine-readable storage media of claim 17, wherein to initiate a request for a secure communication session comprises to send a ClientHello message, including the nonce, to the remote server; and.....	197
b. 10[a] receiving a ServerHello message from the remote server, the ServerHello message including a server nonce; and 18[a] wherein the plurality of instructions further cause the computing device to: receive a ServerHello message from the remote server, the ServerHello message including a server nonce; and.....	199

TABLE OF CONTENTS
(continued)

	Page
c. 10[b] storing the server nonce in the secure memory of the security engine of the system-on-a-chip. 18[b] store the server nonce in the secure memory of the security engine of the system-on-a-chip.....	202
7. Dependent Claims 11, 19.....	202
a. 11 The method of claim 9, wherein performing the cryptographic key exchange with the remote server comprises: generating a pre-master key in the security engine; encrypting the pre-master key using the security key of the security engine; and sending the encrypted pre-master key to the remote server. 19 The one or more non-transitory machine-readable storage media of claim 17, wherein to perform the cryptographic key exchange with the remote server comprises to: generate a pre-master key in the security engine encrypt the pre-master key using the security key of the security engine; and send the encrypted pre-master key to the remote server.....	202
8. Dependent Claims 12, 20.....	203
a. 12[pre]/12[a]: The method of claim 11, further comprising: wrapping the pre-master key in the security engine of the system-on-a-chip using a server public key, and 12[b] storing the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip. 20[pre]/20[a]: The one or more non-transitory machine-readable storage media of claim 19, wherein the plurality of instructions further cause the computing device to: wrap the pre-master key in the security engine of the system-on-a-chip with a server public key, and 20[b] store the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip.....	203

TABLE OF CONTENTS
(continued)

	Page
9. Dependent Claims 13, 21	204
a. 13. The method of claim 11, wherein performing the cryptographic key exchange with the remote server comprises generating the session key in the security engine of the system-on-a-chip as a function of the pre-master key. 21. The one or more non-transitory machine-readable storage media of claim 19, wherein to perform the cryptographic key exchange with the remote server comprises to generate the session key in the security engine of the system-on-a-chip as a function of the pre-master key.....	204
10. Dependent Claims 14, 22	205
a. 14. The method of claim 13, wherein generating the session key in the security engine comprises calculating, using the security engine of the system-on-a-chip, a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server. 22. The one or more non-transitory machine-readable storage media of claim 21, wherein to generate the session key in the security engine comprises to calculate, using the security engine of the system-on-a-chip, a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server.....	205
11. Claims 15, 23	206

TABLE OF CONTENTS
(continued)

	Page
a. 15[e] generating, in the security engine of the system-on-a-chip, the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key. 23[e] generate, in the security engine of the system-on-a-chip, the session key as a function of the private DiffieHellman client key and the public Diffie-Hellman server key.....	206
XI. CONCLUSION.....	206

1. I, Dr. Mark E. Crovella, declare as follows:

I. INTRODUCTION

2. I have been retained by Samsung Electronics Co., Ltd., Samsung Electronics America, Inc. (collectively, “Samsung” or “Petitioner”) as an independent expert consultant in this *inter partes* review (“IPR”) proceeding before the United States Patent and Trademark Office (“PTO”).

3. I have been asked by Counsel for Samsung (“Counsel”) to consider whether certain references disclose, teach and/or suggest the features recited in Claims 1, 3-7, 9-15, 17-24 of U.S. Patent No. 9,887,838 (“the ’838 Patent”) (Ex-1001)¹. My opinions and the bases for my opinions are set forth below.

4. I am being compensated at my ordinary and customary consulting rate for my work, which is \$650 per hour. My compensation is in no way contingent on the nature of my findings, the presentation of my findings in testimony, or the outcome of this or any other proceeding. I have no other financial interest in this proceeding.

II. BACKGROUND AND QUALIFICATIONS

5. All of my opinions stated in this Declaration are based on my own personal knowledge and professional judgment. In forming my opinions, I have

¹ Where appropriate, I refer to exhibits that I understand are attached to the petition for IPR of the ’838 Patent.

relied on my knowledge and experience in designing, developing, researching, and teaching the technology referenced in this Declaration.

6. I am over 18 years of age and, if I am called upon to do so, I would be competent to testify as to the matters set forth herein. I understand that a copy of my current curriculum vitae, which details my education and professional and academic experience, is being submitted as Ex-1003. The following is a brief summary of my qualifications and professional experiences that are relevant to this matter. My expertise qualifies me to do the type of analysis required in this case in the field of network security systems, including firewalls and other packet filtering devices for protecting computer networks from cyber-attacks.

7. I am a Professor in the Department of Computer Science at Boston University. A brief summary of my background is as follows: I earned an undergraduate degree in Biology from Cornell University, which I received in 1982. I received a master's degree in Computer Science from the University of Buffalo in 1989. I received a Ph.D. in Computer Science from the University of Rochester in 1994. The subject of my Ph.D. thesis was "Performance Prediction and Tuning of Parallel Programs."

8. From 1982 to 1984, I worked as a computer programmer for the State of Colorado. From 1984 to 1994, I was employed at Calspan Corporation, a research and development firm in Buffalo, NY, where I rose to the level of Senior

Computer Scientist. My work at Calspan focused on development of experimental software and large-scale simulation software in support of contracts between Calspan and the US Department of Defense.

9. In 1994, I joined the faculty of Boston University, initially as an Assistant Professor of Computer Science. I was promoted to the rank of Associate Professor in 2000 and became a full Professor in 2006. From 2013 to 2018 I served as Chair of the Department of Computer Science.

10. I have been pursuing research in the area of computer networking since 1994. I have conducted research in a variety of areas related to the Internet and the World Wide Web. Among other areas, I have studied the efficient design of Web servers and content distribution systems; I have studied the statistical properties of Internet traffic; and I have made extensive measurements of Internet infrastructure and the behavior of Internet protocols. From 2007 to 2009, I was the Chair of ACM SIGCOMM (the Special Interest Group in Computer Communication), the main professional organization for scientists in the field of computer networking.

11. In addition, I have been pursuing research in the area of computer security since around 2000. I have published extensively on techniques for intrusion detection, security analysis, and the identification of malicious behavior in networks and computer systems. My work in the area of computer and network

security is highly cited.

12. I regularly teach courses that cover aspects of computer systems security, such as *Introduction to Computer Systems*, and *Computer Networking*. In these courses we cover topics such as authentication, secure storage of data, and methods for establishing secure connections between devices. In *Computer Networking*, I teach methods for secure communication over networks, including the principles of symmetric cryptography and public-key cryptography. In addition, I teach methods of using cryptography to establish data confidentiality, integrity, and authentication using well-known protocols such as Diffie-Hellman key exchange and the RSA cryptosystem. I also cover how these technologies are used in the Internet in the context of protocols such as TLS/SSL and virtual private networks (VPNs). I regularly advise students on research projects in which these technologies play a central role.

13. I am co-author of *Internet Measurement: Infrastructure, Traffic, and Applications* (Wiley Press, 2006) and I am the author of over two hundred research papers. According to Google Scholar, my work has been cited by over 30,000 other publications. Also according to Google Scholar, as of March 2014, only nine other researchers worldwide are more cited in the area of computer systems. I hold ten U.S. Patents derived from my Internet related research. I am a past editor of principal journals in the field of networking: *Computer Communication Review*,

IEEE/ACM Transactions on Networking, Computer Networks, and IEEE

Transactions on Computers.

III. INFORMATION CONSIDERED

14. In preparation for this Declaration, I have considered the materials discussed in this Declaration, including, for example, the '838 Patent, the references cited by the '838 Patent, the prosecution history of the '838 Patent, various background articles and materials referenced in this Declaration, and the prior art references identified in this Declaration. In addition, my opinions are further based on my education, training, experience, and knowledge in the relevant field.

15. In forming my opinions, I have reviewed the following documents:

Exhibit	Description
Ex-1001	U.S. Patent No. 9,887,838 (“the '838 Patent”)
Ex-1002	Declaration of Dr. Mark E. Crovella
Ex-1003	Curriculum Vitae of Dr. Mark E. Crovella
Ex-1004	Prosecution History of U.S. Patent No. 9,887,838
Ex-1005	W. Diffie and M. E. Hellman, <i>New Directions in Cryptography</i> , IEEE Transactions on Information Theory, IEEE Transactions on Information Theory, Vol. 22, No. 6, 644-654 (1976)
Ex-1006	T. Dierks and C. Allen, The TLS Protocol Version 1.0, Internet Engineering Task Force (IETF), Network Working Group Request for Comments No. 2246 (January 1999) (“TLS-1.0”)

Exhibit	Description
Ex-1007	R.L. Rivest, A. Shamir, and L. Adleman, <i>A method for obtaining digital signatures and public-key cryptosystems</i> , Communications of the ACM, Vol. 21, 120–126 (1978) (the “RSA Paper”)
Ex-1008	United States Patent Application Publication No. 2011/0191599 (“Chou”)
Ex-1009	U.S. Patent No. 7,773,754 (“Buer”)
Ex-1010	Declaration of Sandy Ginoza for IETF, RFC 2246: The TLS Protocol Version 1.0, RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2 (the “IETF Declaration”)
Ex-1011	U.S. Patent No. 4,405,829 (“Rivest”)

IV. RELEVANT LEGAL STANDARDS

16. I am not an attorney and offer no legal opinions. For the purposes of this Declaration, I have been informed about certain aspects of the law that are relevant to my analysis, as summarized below.

A. Claim Interpretation

17. I have been informed and understand that in an IPR proceeding, claims are to be interpreted according to the *Phillips* claim construction standard. I have been informed and understand that claim construction is a matter of law and that the final claim constructions for this proceeding will be determined by the Patent Trial and Appeal Board (“PTAB”).

18. To resolve the particular grounds presented in this Petition I do not believe any term requires explicit construction.

B. Perspective of One of Ordinary Skill in the Art

19. I have been informed and understand that a patent is to be understood from the perspective of a hypothetical “person of ordinary skill in the art” (“POSITA”). Such an individual is considered to possess normal skills and knowledge in a particular technical field (as opposed to being a genius). I understand that in considering what the claims of a patent require, what was known prior to that patent, what a prior art reference discloses, and whether an invention is obvious or not, one must use the perspective of such a POSITA.

C. Obviousness

20. I have been informed and understand that a patent claim is obvious under 35 U.S.C. §103, and therefore invalid, if the claimed subject matter, as a whole, would have been obvious to a POSITA as of the priority date of the patent based on one or more prior art references and/or the knowledge of a POSITA.

21. I understand that an obviousness analysis must consider (1) the scope and content of the prior art, (2) the differences between the claims and the prior art, (3) the level of ordinary skill in the pertinent art, and (4) secondary considerations, if any, of non-obviousness (such as unexpected results, commercial success, long-felt but unmet need, failure of others, copying by others, and skepticism of experts).

22. I understand that a prior art reference may be combined with other

references to disclose each element of the invention under 35 U.S.C. § 103. I understand that a reference may also be combined with the knowledge of a POSITA, and that this knowledge may be used to combine multiple references. I further understand that a POSITA is presumed to know the relevant prior art. I understand that the obviousness analysis may take into account the inferences and creative steps that a POSITA would employ.

23. In determining whether a prior art reference would have been combined with other prior art or other information known to a POSITA, I understand that the following principles may be considered:

- a. whether the references to be combined involve non-analogous art;
- b. whether the references to be combined are in different fields of endeavor than the alleged invention in the Patent;
- c. whether the references to be combined are reasonably pertinent to the problems to which the inventions of the Patent are directed;
- d. whether the combination is of familiar elements according to known methods that yields predictable results;
- e. whether a combination involves the substitution of one known element for another that yields predictable results;
- f. whether the combination involves the use of a known technique to improve similar items or methods in the same way that yields

predictable results;

- g. whether the combination involves the application of a known technique to a prior art reference that is ready for improvement, to yield predictable results;
- h. whether the combination is “obvious to try”;
- i. whether the combination involves the known work in one field of endeavor prompting variations of it for use in either the same field or a different one based on design incentives or other market forces, where the variations are predictable to a POSITA;
- j. whether there is some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill in the art to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention;
- k. whether the combination requires modifications that render the prior art unsatisfactory for its intended use;
- l. whether the combination requires modifications that change the principle of operation of the reference;
- m. whether the combination is reasonably expected to be a success; and
- n. whether the combination possesses the requisite degree of predictability at the time the invention was made.

24. I understand that in determining whether a combination of prior art references renders a claim obvious, it is helpful to consider whether there is some teaching, suggestion, or motivation to combine the references and a reasonable expectation of success in doing so. I understand, however, that a teaching, suggestion, or motivation to combine is not required.

V. LEVEL OF ORDINARY SKILL IN THE ART

25. I am familiar with the level of ordinary skill in the art with respect to the '838 Patent around its filing date. Based on my experience working, teaching, and conducting research in the relevant field, and based on my review of the '838 Patent specification, claims, file history, and prior art, I believe one of ordinary skill in the art around the time of the alleged invention of the '838 Patent would have been someone with at least a bachelor's degree in electrical engineering, computer engineering, computer science, or a related field, and at least two years of experience in the research, design, development, and/or testing of secure communication and/or security methods, and related firmware and software, or the equivalent, with additional education substituting for experience and vice versa.

26. In determining the level of ordinary skill in the art, I considered, for example, the type of problems encountered in the art, prior art solutions to those problems, the rapidity with which innovations are made, the sophistication of the technology, and the educational level of active workers in the field.

27. I met the definition of a POSITA in 2011. I also had greater knowledge and experience than a POSITA. In my work as a professor and computer scientist, I worked with graduate students and others who had the knowledge and experience of a POSITA in 2011, and I am able to render opinions from the perspective of a POSITA based on my knowledge and experience. My opinions concerning the '838 Patent claims and the prior art are from the perspective of a POSITA, as set forth above.

VI. SUMMARY OF MY OPINIONS

28. I have been asked to consider whether the claims of the '838 Patent are obvious over certain prior art references. As explained below in detail in this Declaration, it is my opinion that:

- Ground 1: Chou, in view of knowledge of a POSITA, renders Claims 1, 9, 17, and 24 of the '838 Patent obvious;
- Ground 2: Chou in combination with TLS-1.0 renders Claims 3-7, 10-15, 18-23 of the '838 Patent obvious;
- Ground 3: Buer, in view of knowledge of a POSITA, renders Claims 1, 3-7, 9-15, 17-24 of the '838 Patent obvious;
- Ground 4: Buer combination with Chou renders Claims 1, 3-7, 9-15, 17-24 of the '838 Patent obvious;
- Ground 5: Buer in combination with TLS-1.0 renders Claims 3-5, 7, 10-

15, 18-23 of the '838 Patent obvious; and

- Ground 6: Buer in combination with Chou and TLS-1.0 renders Claims 3-5, 7, 10-15, 18-23 of the '838 Patent obvious.

VII. TECHNOLOGICAL BACKGROUND

A. Secure Key Exchange Algorithms

29. Encryption is a process of transforming information or data into a coded format to protect it from unauthorized access. The goal of encryption is for the contents of the transformed message, also called ciphertext, to only be deciphered and read by someone who has the appropriate key. Encryption is widely used in communication over networks; for example, it is commonly used to protect sensitive information such as personal data, financial information, and trade secrets. This is particularly important in networks like the Internet because messages passing over such networks are subject to potential inspection, interception, and modification by third parties.

30. Encryption works by using mathematical algorithms to scramble the original message, making it unreadable to anyone who does not have the appropriate key. When the recipient receives the encrypted message, they use the key to unscramble the message and read its original contents.

1. Symmetric Encryption v. Asymmetric Encryption

31. Symmetric encryption, also known as shared secret encryption, is a

method of encryption where the same key is used for both encryption and decryption. This means that the sender and receiver of the encrypted message both have access to the same key. Symmetric encryption was one of the earliest forms of encryption and has been used for centuries to protect sensitive information.

32. The main problem with symmetric encryption is the need for a shared secret between the sender and the receiver of the encrypted message. Because the same key is used for both encryption and decryption, the key must be securely transmitted from the sender to the receiver before communication can take place. This creates a vulnerability, as the key could be intercepted by an attacker during transmission, allowing the attacker to access the encrypted messages. In addition, if the key is lost or otherwise compromised, the encrypted messages will no longer be secure.

33. These limitations led to the search for a more secure encryption method that would allow communication between two parties without the need for a shared secret. This eventually led to the development of asymmetric encryption and public key cryptography, and specifically the RSA encryption algorithm.

34. Asymmetric encryption, in contrast to symmetric cryptography, uses two different (yet complimentary) keys for encryption and decryption. One key is used to encrypt the message and the other key is used to decrypt the message. This method of encryption is also known as public key encryption because one of the

keys is made public while the other key is kept private. With public key cryptography, two different keys are used: one for encryption and one for decryption. This eliminates the need for a shared secret, making it much more secure and practical for use in secure communication over the internet.

2. Diffie-Hellman Key Exchange Algorithm

35. The Diffie-Hellman key exchange algorithm was proposed by Whitfield Diffie and Martin Hellman in 1976. *See* Ex-1005, Diffie and Hellman, “New Directions in Cryptography” IEEE Transactions on Information Theory, Vol. 22, No. 6 (November 1976). It was one of the first algorithms to solve the key distribution problem in cryptography, which is the challenge of securely sharing secret keys between two parties who have not met.

36. Prior to the advent of the Diffie-Hellman algorithm cryptography required communicating parties to have exchanged a common key by some physical means before beginning to exchange encrypted messages. The Diffie-Hellman algorithm eliminated this need by allowing two parties to securely exchange keys over an insecure channel, such as the Internet. The Diffie-Hellman algorithm uses a public key to establish a shared secret between the two parties, which can then be used as a symmetric key to encrypt and decrypt messages.

37. The basic idea of Diffie-Hellman key exchange is that two parties can each generate public-private key pairs, and then exchange their public keys. They

can then use their private keys and the other party's public key to derive a shared secret key. Because only the public keys are exchanged over the insecure channel and the private keys are never exchanged, an eavesdropper cannot determine the shared secret key, even if they could intercept the public keys.

38. The Diffie-Hellman key exchange generally follows the following steps:

39. **Key Generation**: Both parties generate their own private key and corresponding public key. The private key is kept secret, while the public key is made available to the other party.

40. To generate their private key, each party selects a random number (denoted here as “ x ,” “ a ,” or “ b ”).

41. The public key is then generated using a prime number (often referred to as “ p ”) and a primitive root modulo p (often referred to as “ g ”). The prime number and the modulo can be exchanged between the two parties on an open channel. Using these public numbers and his/her own private key, each party calculates its public key as below:

- $\text{Public Key} = g^x \bmod p$

42. For example, if Alice selects a as her private key, and Bob selects b as his private key, Alice's public key is calculated as:

- $\text{Public Key (Alice)} = g^a \bmod p$

And Bob's public key is calculated as:

- $\text{Public Key (Bob)} = g^b \bmod p$

43. **Key Exchange**: Both parties exchange their public keys. This exchange can occur on a public channel. After this exchange, each side has its own private key and the other side's public key.

44. **Secret Key Computation**: Once both parties have each other's public key, they can compute the shared secret key as follows:

- $\text{Shared Secret Key} = (\text{Public Key of Other Party})^x \bmod p$

45. For example, using Alice and Bob's public keys from the example above:

- $\text{Shared Secret Key (Alice)} = g^{ba} \bmod p$
- $\text{Shared Secret Key (Bob)} = g^{ab} \bmod p$

46. The resulting shared secret key will be the same for both parties, even though their private keys are different and were never communicated. This is because the calculation of the shared secret key uses modular exponentiation, which is commutative, meaning that the order of the operands does not affect the result.

47. **Secret Key Use**: The shared secret key can then be used for a variety of purposes, such as encryption, decryption, and digital signature verification. The shared secret key can also be used to derive other shared keys according to

predetermined algorithms.

48. The security of the Diffie-Hellman key exchange relies on the computational difficulty of computing discrete logarithms. In particular, computing $g^{ab} \bmod p$ given only the knowledge of g , p , $g^a \bmod p$, and $g^b \bmod p$ is computationally very expensive. For example, if p is a prime number of 600 digits, no modern computer can find $g^{ab} \bmod p$ (without knowledge of a and b) in any reasonable amount of time (e.g. decades). This makes it computationally infeasible for an eavesdropper to determine the shared secret key from the public keys alone, even if they are able to intercept the public key exchange.

49. Diffie-Hellman has become a widely-used key exchange algorithm due to its simplicity and security, and it has been incorporated into many cryptographic protocols, including SSL/TLS, IPsec, and SSH. It provides a secure means of establishing a shared secret between two parties over an insecure channel, and is an essential component of many modern secure communication systems. The algorithm has also been adopted in standards by various organizations, including the Internet Engineering Task Force (IETF) and the National Institute of Standards and Technology (NIST).

50. I teach the Diffie-Hellman key exchange algorithm in my *Computer Networking* class, directed to third-year and fourth-year computer science and computer engineering students. I expect a POSITA at around 2011 would have

been familiar with the Diffie-Hellman key exchange algorithm and its components.

3. RSA Encryption

51. The RSA encryption algorithm was invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman, who were all researchers at the Massachusetts Institute of Technology (MIT). *See* Ex-1011. Rivest, Shamir, and Adelman's paper, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, Vol. 21 Issue 2, 120-125 (Feb. 1978), describes the mathematical basis for the RSA encryption algorithm and outlines the first practical method for implementing public-key cryptography. Ex-1007. It is considered a landmark in the field of cryptography and is widely recognized as one of the most important contributions to the development of secure communication methods over the internet.

52. Before the invention of RSA encryption, symmetric encryption was the primary method for providing confidentiality when communicating over the Internet. However, symmetric encryption has a major drawback when used in the context of the Internet: a key has to be shared between the sender and the receiver of the encrypted message. If the key is simply sent over the network in the clear, it is vulnerable to interception by a third party. Rivest, Shamir, and Adleman, responding to this limitation of symmetric encryption, proposed their method to allow secure communication between two parties *without the need for a shared*

secret. They proposed the use of two different keys: one for encryption and one for decryption. This marked the birth of public key cryptography.

53. RSA encryption uses mathematical algorithms to generate two keys: a public key and a private key. The public key is used to encrypt the message, and the private key is used to decrypt the message. This allows secure communication without the need for a shared secret, as the private key is kept secret by the recipient of the message. The algorithm works as follows:

54. **Key Generation**: A party wishing to receive private messages generates an RSA key pair. This involves selecting two large prime numbers, p and q , and calculating their product, $n = pq$. The public key, e , is then selected such that it is relatively prime to $(p-1)(q-1)$. The private key, d , is calculated such that it satisfies the equation $de = 1 \bmod (p-1)(q-1)$.

55. **Key Exchange**: One of the parties, say Alice, wants to send a message to the other party, Bob. Bob sends Alice his public key, e and the modulus n . These values can be sent in the clear, i.e. over an insecure network.

56. **Encryption**: Alice wants to encrypt the message, m , so she calculates the ciphertext, c , using the following equation: $c = m^e \bmod n$. This ciphertext can only be decrypted using Bob's private key.

57. **Decryption**: Bob uses his private key, d , to calculate the original message, m , using the following equation: $m = c^d \bmod n$.

58. The security of RSA lies in the fact that factoring a large number into its prime components is computationally expensive. As a result, an attacker cannot in a reasonable amount of time determine the values of p and q from knowledge of the public key e and the modulus n . This prevents the attacker from finding the value of the private key d .

59. RSA is widely used in protocols such as SSL/TLS, which provides secure communication for applications such as email, instant messaging, and e-commerce.

60. I teach the RSA algorithm in my *Computer Networking* class, directed to third- and fourth-year computer science and computer engineering students. I expect a POSITA at around 2011 would have been familiar with the details of the RSA algorithm and its components.

B. SSL/TLS Protocol

61. Transport Layer Security (TLS) is a protocol that provides secure communication over the Internet. The TLS protocol is not, however, an encryption method by itself. Instead, the TLS protocol provides the framework and mechanisms for secure communication, including the negotiation of encryption algorithms, the exchange of encryption keys, and the verification of the authenticity of the communicating parties. The actual encryption of data is performed using one or more encryption algorithms, such as RSA, Diffie-Hellman,

or a symmetric encryption algorithm explained above, that are agreed upon during the TLS handshake. The encryption keys used for this encryption are generated and exchanged during the TLS handshake and are used to encrypt and decrypt data transmitted between the communicating parties.

62. The predecessor to TLS is called the Secure Sockets Layer (SSL) protocol. SSL was first developed in the 1990s by Netscape Communications Corporation as an extension to the standard web protocol, HTTP, as a response to the increasing need for secure communication over the Internet. At the time, the Internet was rapidly growing, but it was also becoming clear that traditional methods of communication, such as email and file transfer, were not sufficiently secure. The SSL protocol was designed to provide a secure, encrypted connection between a client and a server, allowing sensitive information to be transmitted securely over the Internet. The primary use case for SSL was to secure online transactions, such as e-commerce, online banking, and other activities that involved the transmission of sensitive information, such as credit card numbers, social security numbers, and other personal data.

63. The SSL protocol operates at the transport layer, between the application layer and the network layer, and provides end-to-end encryption for all data transmitted over the connection. It uses a combination of public key cryptography and symmetric key cryptography to provide authentication,

confidentiality, and data integrity.

64. The initial SSL version 1.0 had several security flaws, and SSL 2.0 was quickly released as an upgrade. Despite being an improvement over SSL 1.0, SSL 2.0 was still considered to have security weaknesses, so SSL 3.0 was introduced in 1996. SSL 3.0 addressed many of the security concerns of SSL 2.0 and was widely adopted as the standard for securing internet communication.

65. In 1999, the Internet Engineering Task Force (IETF) took over the development of SSL and renamed it TLS (Transport Layer Security). Since then, several versions of TLS have been released, each with increased security and improved performance.

66. The TLS Protocol Version 1.0 is defined in IETF publication RFC 2246 in 1999. *See* Ex-1006. RFC stands for “Request for Comments” and is a publication series by the Internet Engineering Task Force (IETF), the primary standards organization for the internet. The RFC 2246 document, entitled “The TLS Protocol Version 1.0,” provides a comprehensive description of the TLS-1.0 protocol, including its design goals, cryptographic algorithms, and communication procedures. It serves as the official reference for the TLS-1.0 protocol.

67. The TLS-1.0 Handshake Protocol is the process that establishes a secure communication session between a client and a server. It is a crucial part of the TLS-1.0 protocol and involves several steps to ensure the confidentiality,

integrity, and authenticity of the communication. The steps of the TLS-1.0

Handshake Protocol are as follows:

- Client Hello: The client sends a message to the server indicating its supported cryptographic algorithms and other parameters. Client Hello message include:
 - Protocol Version: This field indicates the version of the TLS protocol that the client supports.
 - Random: This field contains a client-generated random number that is used as part of the key generation process.
 - Session ID: This field is used to identify a previously established session and can be used to resume the session instead of starting a new one.
 - Cipher Suites: This field lists the cryptographic algorithms and protocols that the client supports, such as AES for encryption and SHA-256 for hashing.
 - Compression Methods: This field lists the data compression methods that the client supports, such as Deflate.
 - Extensions: This field is used to negotiate additional parameters, such as maximum packet size and supported elliptic curves for Diffie-Hellman key exchange.

The Client Hello message is used by the server to determine which cryptographic algorithms and parameters to use for the session.

- Server Hello: The server responds with a Server Hello message that includes its own parameters and any additional information needed to complete the handshake process. The Server Hello message includes:
 - Protocol Version: This field indicates the version of the TLS protocol that the server has selected to use for the session.
 - Random: This field contains a server-generated random number that is used as part of the key generation process.

- Session ID: This field is used to identify a previously established session and can be used to resume the session instead of starting a new one.
 - Cipher Suite: This field indicates the cryptographic algorithms and protocols that the server has selected to use for the session, such as AES for encryption and SHA-256 for hashing.
 - Compression Method: This field indicates the data compression method that the server has selected to use for the session, such as Deflate.
 - Extensions: This field is used to negotiate additional parameters, such as maximum packet size and supported elliptic curves for Diffie-Hellman key exchange.
- Certificate: The server sends its certificate to the client, which the client can use to verify the server's identity.
 - Server Key Exchange: The server can send a key exchange message if the selected cipher suite requires it, such as when using Diffie-Hellman key exchange.
 - Certificate Request: The server can request that the client provide its certificate for authentication, if desired.
 - Server Hello Done: The server indicates that it has completed sending information to the client.
 - Client Certificate: If the server requested it, the client sends its certificate for authentication.
 - Client Key Exchange: The client generates a shared secret and sends it to the server in an encrypted form.
 - Change Cipher Spec: Both the client and server send change cipher spec messages to indicate that they will start using the negotiated cryptographic algorithms for the remaining messages.
 - Finished: Both the client and server send a "Finished" message to indicate that they have completed the handshake process and that the secure communication session is established.

- The “Finished” message is created by computing a hash over all the previous handshake messages, including the client hello, server hello, certificate, and key exchange messages. This hash is then encrypted using the newly established shared secret key, producing a message authentication code (MAC). The encrypted MAC is sent in the “Finished” message to the other party.
- The recipient of the “Finished” message decrypts the MAC using the shared secret key, and then recomputes the hash over the previous handshake messages. If the hash values match, the recipient knows that the previous messages have not been tampered with, and the handshake process is considered complete. The “Finished” message is a critical step in the TLS handshake, as it provides a secure end-to-end encryption for all data transmitted during the session.

68. The TLS-1.0 Handshake Protocol uses cryptographic algorithms and digital signatures to establish a secure communication channel between the client and server. The process ensures that the communication is protected from eavesdropping, tampering, and impersonation, and that the authenticity of the client and server can be verified. By the time of the alleged priority date of the '838 Patent, TLS Protocol version 1.2, which expanded on TLS version 1.0, was published by IETF as RFC 5246, in August 2008. *See* Ex-1010 ¶12.

69. There were several open-source programs that implemented the TLS (Transport Layer Security) protocol before 2011. Some of these examples include:

- OpenSSL: OpenSSL is an open-source software library that implements SSL/TLS protocols as well as cryptographic functions. It was first released in 1998 and has been widely used in various applications, including web servers, email servers, and VPNs.
- NSS: NSS (Network Security Services) is a set of open-source libraries

that provide security services, including SSL/TLS protocols. It was first released in 1998 and has been used in various applications, including web browsers and email clients.

- PolarSSL: PolarSSL (now known as mbed TLS) is an open-source software library that implements SSL/TLS protocols as well as cryptographic functions. It was first released in 2009 and has been used in various applications, including embedded systems and IoT devices.
- MatrixSSL: MatrixSSL is an open-source software library that implements SSL/TLS protocols. It was first released in 2004 and has been used in various applications, including web servers, email servers, and VPNs.

70. I teach the TLS Protocol in my *Computer Networking* class, directed to third- and fourth-year computer science and computer engineering students. A POSITA in 2011 would have been familiar with the details of the TLS Handshake Protocol, including the type of messages exchanged in TLS Handshake and their components.

C. Digital Signature

71. In public key cryptography, a “digital signature,” or “signing with a private key” is a mathematical technique used to verify the authenticity and integrity of a digital document or message. It is based on the use of public and private key pairs, which are unique mathematical keys that can encrypt and decrypt messages.

72. To create a digital signature, the sender of the message uses their private key to generate a code that is added to the message. This code is unique to the message and the sender, and it can be used to verify that the message has not

been tampered with and that it was indeed sent by the sender.

73. When the recipient receives the message, they can use the sender's public key to verify the signature. The recipient uses the sender's public key, which is made available to the public, to apply a reverse function to the code received. If the result of this function matches the original message, it proves that the signature was indeed created by the sender using their private key. If the result does not match the original message, it means that the signature is invalid, and the message may have been tampered with or forged.

74. The security of digital signatures in public key cryptography relies on the fact that it is computationally infeasible to create a matching signature without the sender's private key. This ensures that the digital signature is unique to the sender and that it cannot be forged by an unauthorized party.

75. Digital signatures are widely used in many applications, such as online banking, e-commerce, and secure email communication, to provide authentication and non-repudiation services. Non-repudiation means that the sender of a message cannot later deny having sent it, as the digital signature provides irrefutable proof of their identity and intent.

76. I teach digital signature in my *Computer Networking* class, directed to third- and fourth-year computer science and computer engineering students. A POSITA in 2011 would have been familiar with the details of the TLS Handshake

Protocol, including the type of messages exchanged in TLS Handshake and their components.

VIII. THE '838 PATENT AND ITS FILE HISTORY

A. Overview of the '838 Patent

77. The '838 Patent, titled “Method and Device for Secure Communication over a Network Using a Hardware Security Engine,” is directed to “[a] security engine . . . embodied as a security co-processor or processing circuitry separate from the processor core.” Ex-1001, 4:17-19.

78. While much of the specification is spent to describe the various communications between the hardware security engine and a server, those communications steps are not the invention of the '838 Patent. Rather, these communications and information exchanged are the standard description of the SSL/TLS protocol. The description of the various Diffie-Hellman key exchange components are likewise the description of the TLS protocol that implements Diffie-Hellman key exchange algorithm (which the TLS explicitly supports). *See e.g.*, Ex-1006, § 7.4.2 (listing RSA public key and Diffie-Hellman key as type of key exchange algorithms); *see also*, Ex-1005. The same applies to the description of the RSA key exchange contained in the '838 Patent. *See* Ex-1006, § 7.4.2; Ex-1007.

79. The '838 Patent itself acknowledges that the alleged invention

“establish[es] a secure communication session with a server includes initiating a request for a secure communication session, such as a Secure Sockets Layer (SSL) communication session with a server.” Ex-1001, Abstract. Accordingly, much of the descriptions of the communication steps, including the description accompanying Figure 3, reproduced below, recites the well-established standard steps that of the SSL/TLS protocol.

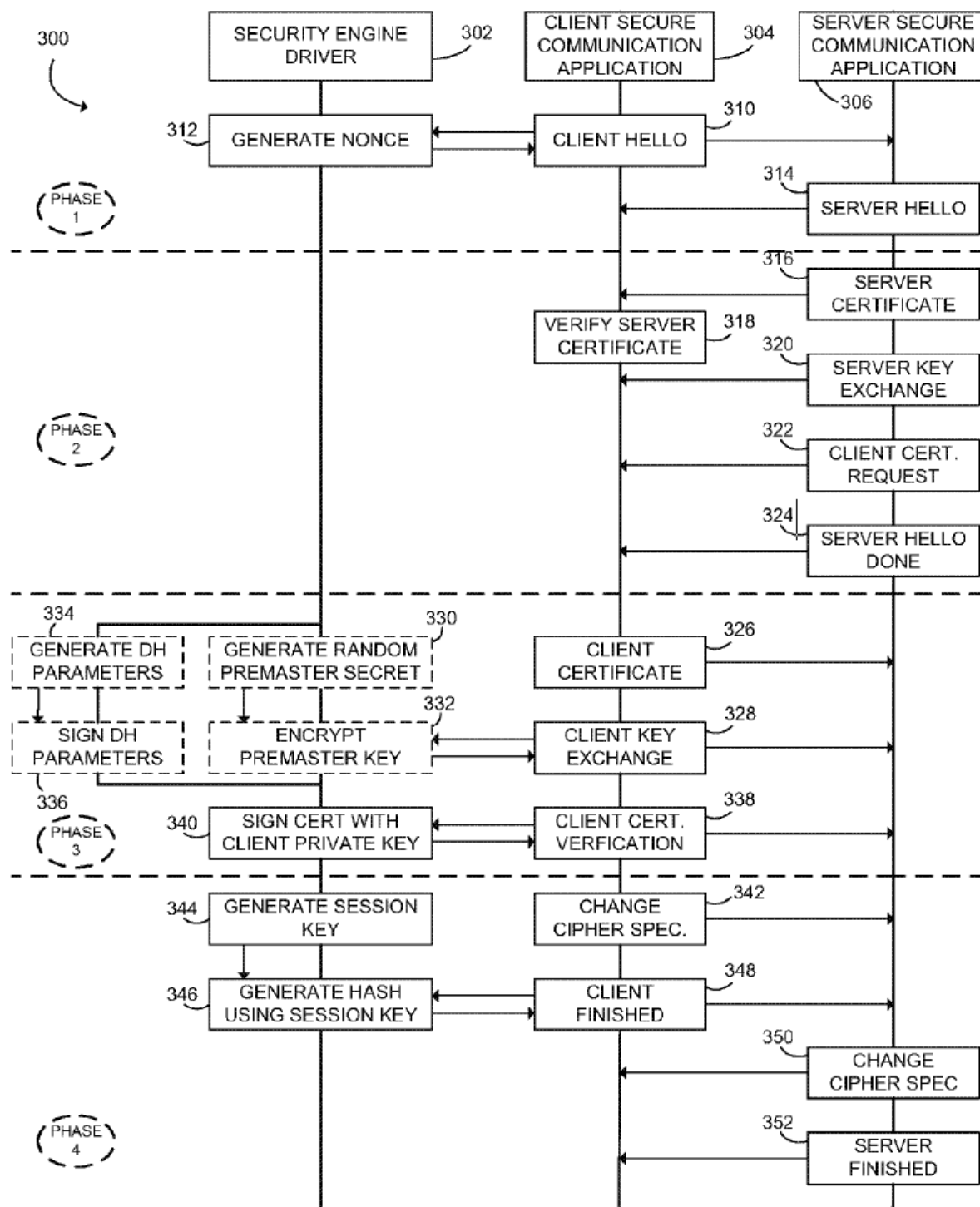


FIG. 3

Ex-1001, Fig. 3.

80. Each of the steps that are described in connection with Figure 3 matches the SSL/TLS handshake protocol I explained above in Section **VII.B** above. These known steps of the SSL/TLS protocol (i.e., Client Hello message, which includes a random nonce, initiating the key exchange; generating a pre-master key; creating a symmetric session key from the pre-master key) were performed by a “security engine” (highlighted in yellow) which includes a “secure memory” (highlighted in green). The security engine also includes a “security key” (highlighted in cyan), which is a device-specific key that is encoded at the time the SoC that includes the security engine was manufactured. The “secure memory” is used to store various data processed by the security engine, such as the session key encrypted by the “security key,” encrypted pre-master key, or other data received during the SSL key exchange procedure for use in creating the pre-master key (e.g., a server nonce received from the server).

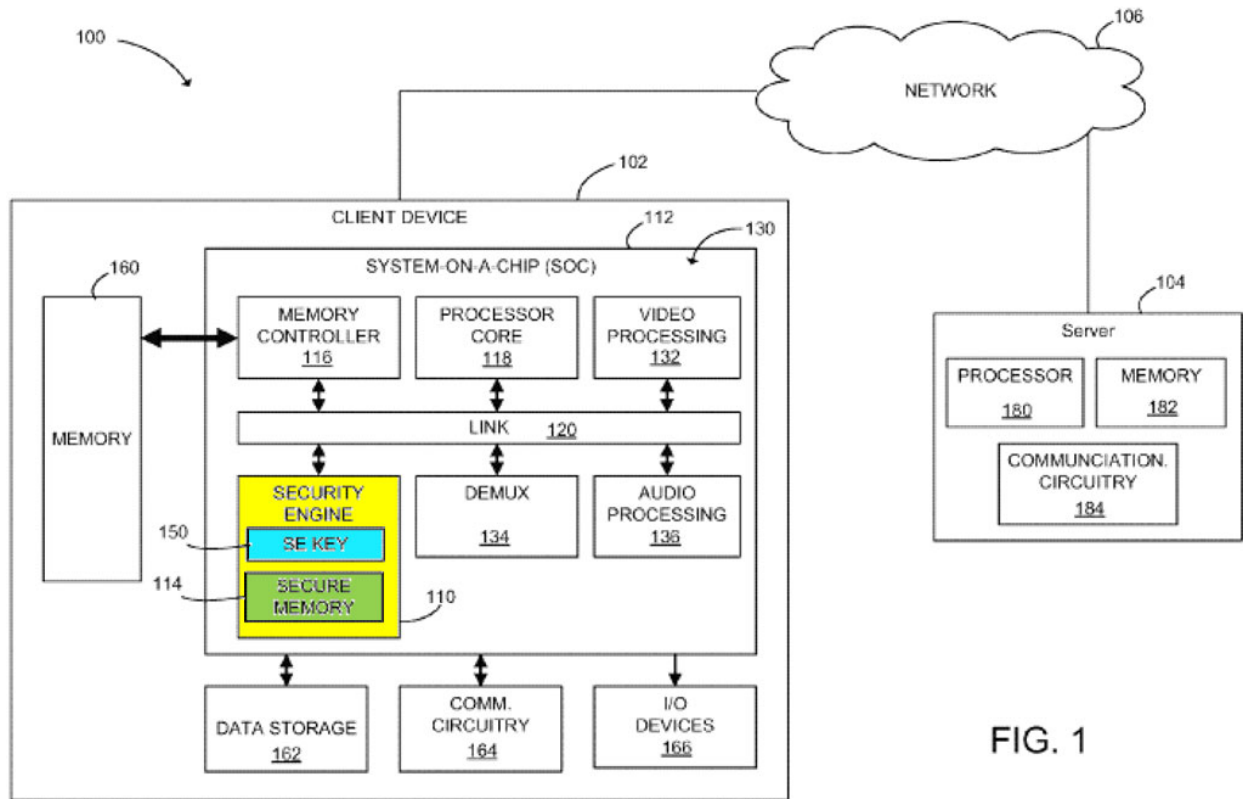


FIG. 1

Ex-1001, Fig. 1 (highlight added).

B. Challenged Claims

The claims at issue are Claims 1, 3-7, 9-15, 17-24 of the '838 Patent.

C. Prosecution History of the '838 Patent

81. I understand that the '838 Patent issued from U.S. Application No. 13/997,412 ("the '412 Application"), filed on July 23, 2010. Ex-1001, Cover.

82. I understand that the claims of the '412 Application were rejected over prior art references several times, and amended each time to overcome such rejections. For example, in Claim 1 of the '838 Patent reproduced below, the underlined limitations were each added to overcome rejections based on prior art:

1. A system-on-a-chip apparatus comprising:

a system-on-a-chip comprising a security engine that is separate from a processor core of the system-on-a-chip and has a secure memory accessible only by the security engine, wherein the secure memory includes a security key that was encoded in the secure memory during a manufacturing process of the system-on-a-chip, the security engine to:

generate a random nonce for initiating a request for a secure communication session with a remote server over a network using the nonce;

perform a cryptographic key exchange with the remote server;

generate a symmetric session key, based on the cryptographic key exchange, to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session;

encrypt the symmetric session key based on the security key; and

store the encrypted session key in the secure memory,

the system-on-a-chip to establish the secure communication session with the remote server over the network using the session key.

Ex-1001, Cl. 1.

83. I also understand that the prior art relied on in this Petition was not considered by the PTO during prosecution of the '838 Patent. Ex-1001, Cover.

84. Thus, I understand that the Petition presents substantially new arguments that were not considered during prosecution.

D. Priority Date

85. For the purposes of this declaration, I have been asked to assume that the priority date of the '838 Patent is December 15, 2011.

E. Claim Construction

86. I do not believe that any term of the challenged claims requires explicit construction.

IX. OVERVIEW OF THE APPLIED PRIOR ART REFERENCES

A. United States Patent Application Publication No. 2011/0191599 (“Chou”) (Ex-1008)

87. I understand that Chou, titled “Apparatus and Methods for Providing Hardware Security” was filed February 26, 2010, published August 4, 2011, and qualifies as prior art under at least pre-AIA 35 U.S.C. §102(a).

88. Chou is directed to a “hardware security module that provides a secure boundary for retention of a secure key within the secure boundary and prevention of unauthorized accesses from external sources outside of the secure boundary to obtain the secure key.” Ex-1008, Abstract.

89. Chou recognizes that, to ensure only authorized parties can access and decrypt transmitted data, keys are used to encrypt it. For example, to perform secure financial transactions over a network, “data content that contains financial information, such as a credit/debit card information, are encrypted and transmitted to a designated recipient through an unsecure network.” Ex-1008 ¶ 6. But the

security of the data can only be guaranteed as long as the keys used to encrypt and decrypt the data remain secure. Thus, Chou sought to “provide a more secure key management scheme where a secure key may be unwrapped by a recipient, but the key not be revealed to the recipient and/or unauthorized resources.” Ex-1008 ¶ 9. To provide such a secure key management scheme, Chou proposes a “hardware security module” (HSM).

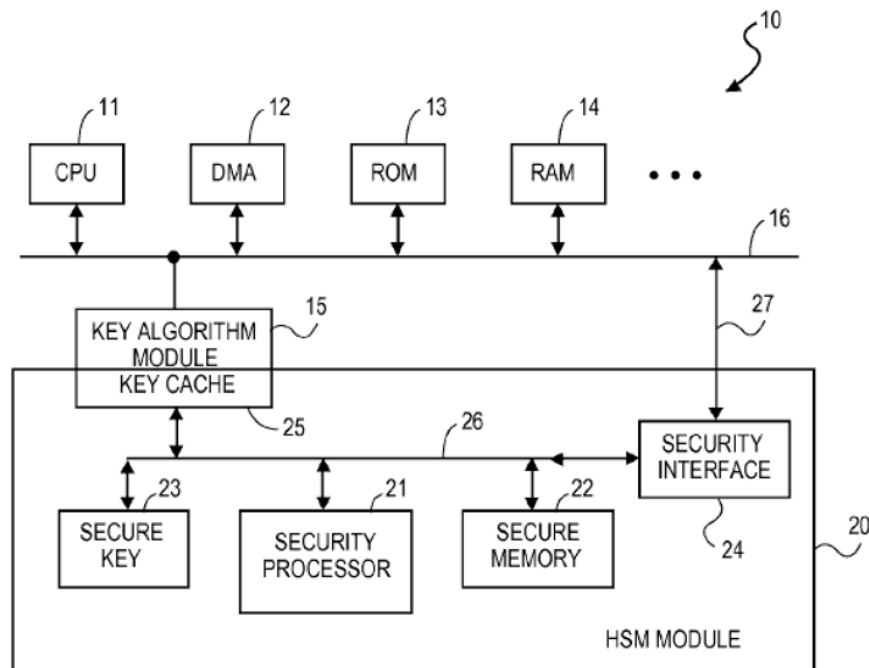


FIG. 1

Ex-1008, Fig. 1.

90. Chou’s HSM module includes its own security processor that is separate from the CPU of the system. Ex-1008 ¶20; Fig. 1.

91. The HSM module also includes a secure memory inside the HSM module, which can only be accessed by the HSM module itself. The secure

memory stores the data processed (or to be processed) by the security processor inside the HSM. Because the secure memory is only accessible by the HSM itself, the sensitive data processed by the HSM is protected from unauthorized access.

92. In addition, to protect the secure key, which may need to be transmitted to outside the HSM, the HSM encrypts the secure key (such as the session key used to encrypt and decrypt the data exchanged during the exemplary secure financial transaction) using a device unique key inside a “secure boundary” (noted with red highlight in Figure 3, reproduced with highlights below), and whenever the secure key is needed outside the secure boundary, only allowing the encrypted version of the secure key to be sent outside the boundary.

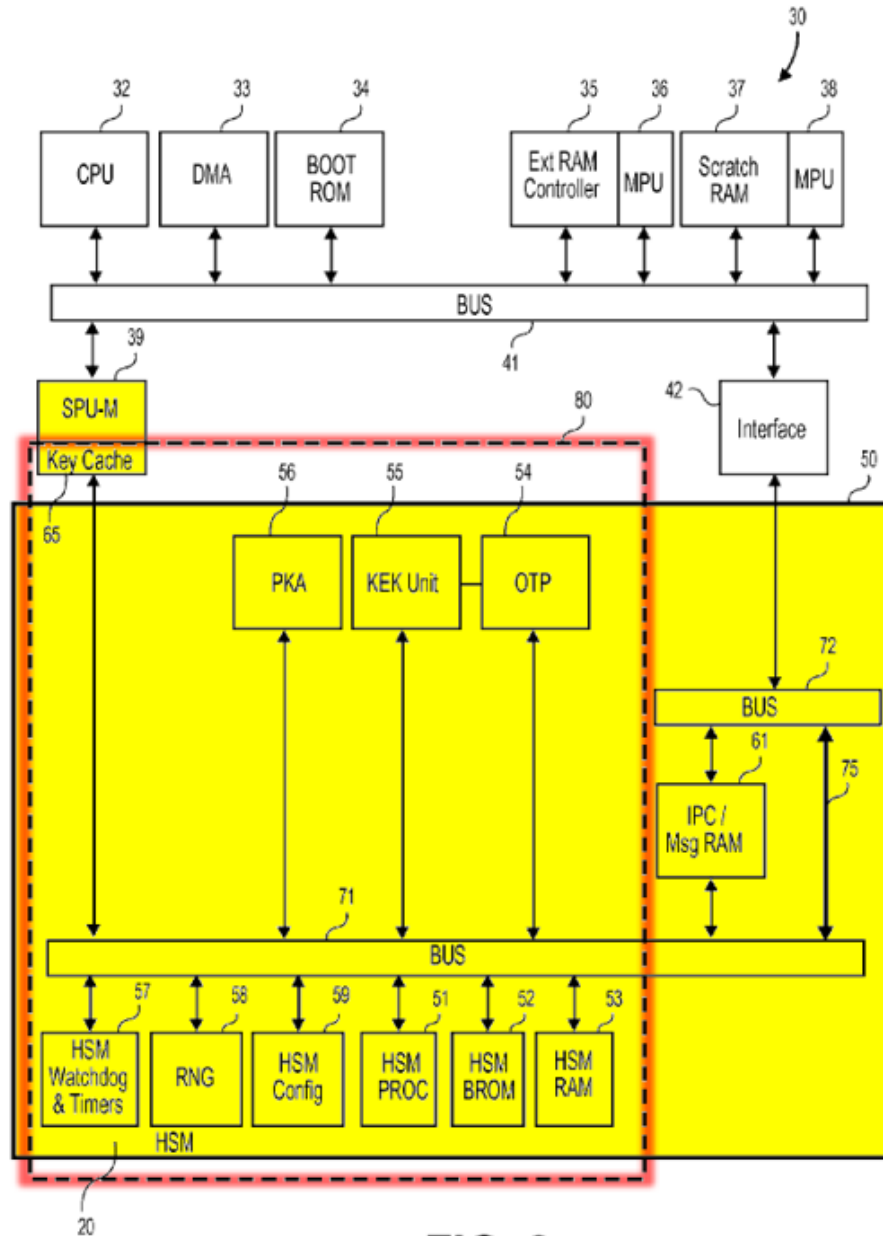


FIG. 3

Ex-1008, Fig. 3.

B. U.S. Patent No. 7,773,754 (“Buer”) (Ex-1009)

93. I understand that Buer, titled “Apparatus and Methods for Providing Hardware Security” was filed July 8, 2002, published August 10, 2010, and qualifies as prior art under at least pre-AIA 35 U.S.C. §102(a) and (b).

94. Buer is directed to addressing the same subject matter as the '838 Patent—i.e., “[m]ethods and systems . . . for providing secured data transmission and for managing cryptographic keys.” Ex-1009, Abstract. Buer discloses in detail the hardware structure for secure key management as well as the communication steps for establishing symmetric keys between two devices over a network.

95. Similar to Chou, Buer also disclose a security module that has its own processor, separate from the host processor of the system. Buer’s security modules can communicate with each other, through the respective host processors, to establish a secure channel of communication: Figure 13 of Buer, reproduced below, shows one exemplary embodiment showing the communication between a main security module (in a host device) and a remotely located satellite security module (a remote server).

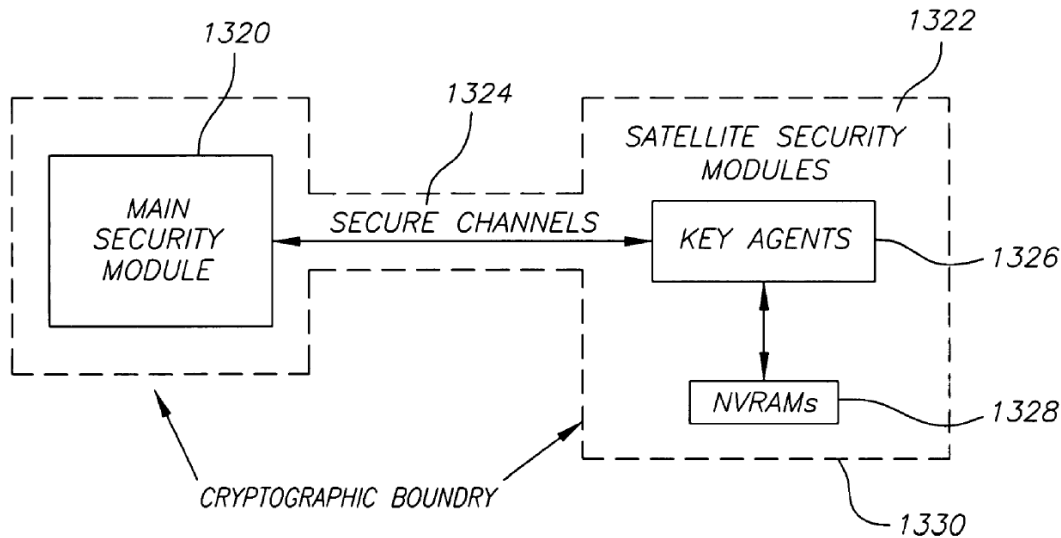


FIG. 13

Ex-1009, Fig. 13.

96. To establish the secure channel between the main security module and the satellite security module, Buer discloses various cryptographic key exchange steps that implement the Diffie-Hellman key exchange algorithm. Buer explains that, once the security module generates the symmetric session key, the session key is encrypted based on the Key Encryption Key (KEK). Ex-1009, 5:43-51 (“In accordance with this embodiment of the invention, these session keys are encrypted using KEK **238** and stored in a database **240** as security associations.”). Buer explains that, to protect the session keys, “the security module **320** encrypts them with a KEK **332** before they are sent to the cryptographic accelerator **326** (via the host processor **322**).” Ex-1009, 6:41-43. *See also*, Ex-1009, 1:9-12. (“To protect [the] private keys [such as the SSL session keys], most of the encryption

and decryption operations are performed inside the security module.”).

C. TLS-1.0

97. T. Dierks and C. Allen, The TLS Protocol Version 1.0 (“TLS-1.0”) was published in January 1999 by Internet Engineering Task Force (IETF), and qualifies as prior art under at least pre-AIA 35 U.S.C. §102(a) and (b).

98. Internet Engineering Task Force is a standard-setting organization well-known to POSITAs. The IETF publications were publicly accessible on the web at <http://IETF.org> since at least the late 90s. IETF publications, such as the TLS-1.0 (published as RFC 2246, “RFC” stands for “request for comments”), have been made available on IETF’s website for review and comments by POSITAs.

99. TLS-1.0 provides a comprehensive description of the TLS-1.0 protocol, including its design goals, cryptographic algorithms, and communication procedures. It serves as the official reference for the TLS-1.0 protocol.

100. TLS-1.0 provides a detailed description of the version 1.0 of the TLS protocol and its features, including:

- Handshake Protocol: The Handshake Protocol is used to negotiate the encryption and authentication algorithms to be used during the secure communication.
- Record Protocol: The Record Protocol is used to encrypt and authenticate the data exchanged between the client and server.
- Alert Protocol: The Alert Protocol is used to convey messages about the status of the connection or errors that occur during the communication.

- Cipher Suites: TLS-1.0 supports various cipher suites that specify the encryption algorithms and key exchange mechanisms used to secure the communication.
- Public Key Infrastructure (PKI): TLS-1.0 uses PKI to authenticate the server and, optionally, the client. This involves the use of digital certificates and a trust model to ensure the authenticity of the parties involved in the communication.
- Message Authentication Code (MAC): TLS-1.0 uses MAC to ensure the integrity of the data transmitted during the communication.

101. TLS-1.0 also provides guidance on how to implement the protocol and addresses various security considerations, such as how to handle certificate validation, how to protect against attacks like man-in-the-middle, and how to prevent downgrade attacks. Overall, the document serves as a comprehensive guide for developers and security professionals looking to implement and use the TLS-1.0 protocol.

102. TLS-1.0 describes each steps the SSL/TLS handshake protocol I explained above in Section **VII.B**.

X. DETAILED EXPLANATION OF THE UNPATENTABILITY GROUNDS

103. I understand from Counsel that the '838 Patent contains three independent claims. Independent Claims 1, 9, and 17 contain substantially similar limitations. Dependent Claims further limit independent Claims 1, 9, and 17. Substantially similar claim elements are discussed together in the below element-by-element analysis. Any meaningful differences between the claims or elements

are individually identified and discussed. As explained below, the challenged claims are disclosed or taught by the prior art.

A. Ground 1: Chou in view of knowledge of a POSITA (Claims 1, 9, 17, 24)

1. Independent Claim 1

a. 1[pre]: A system-on-a-chip apparatus comprising:

104. Chou discloses a hardware security module (HSM) implemented on a system-on-a-chip apparatus, as shown in Figure 4, reproduced below:

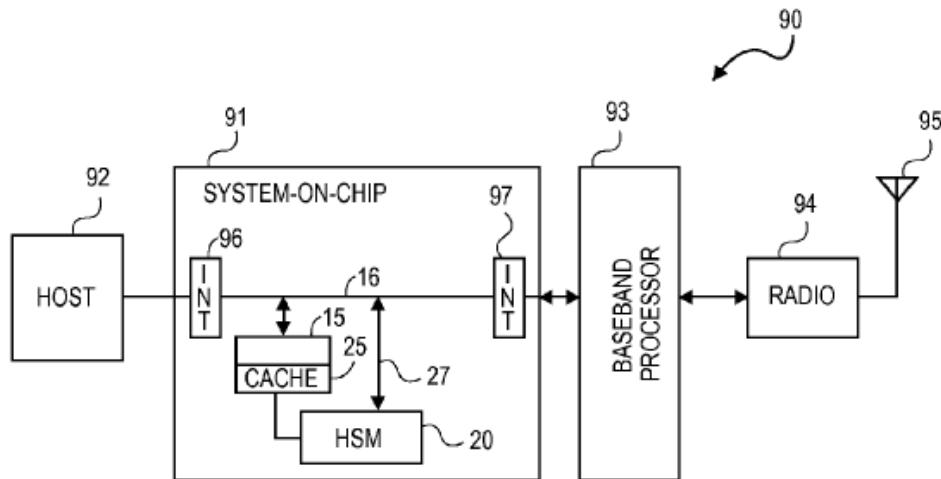


FIG. 4

[0065] HSM 20 may be implemented in a variety of components, circuits, devices, processors, state machines, programmable arrays, etc. It is to be noted that HSM 20 may be constructed on a single integrated circuit (IC) chip or on multiple chips. In other embodiments, components of device 30 is implemented within a single integrated circuit (IC) chip 91 that incorporates a complete system on the IC chip (System-On-Chip or SOC). FIG. 4 shows one such embodiment in which HSM 20 and other components of FIG. 1 or 3 are implemented as SOC 91. It is to be noted

that device **30** of FIG. 3 may be readily implemented as
SOC 91.

Ex-1008 ¶65.

- b. 1[a] a system-on-a-chip comprising a security engine that is separate from a processor core of the system-on-a-chip and has a secure memory accessible only by the security engine, wherein the secure memory includes a security key that was encoded in the secure memory during a manufacturing process of the system-on-a-chip, the security engine to:**

105. **“a system-on-a-chip comprising a security engine that is separate from a processor core of the system-on-a-chip”:** Chou’s system-on-a-chip comprises a hardware security module (“HSM”) and a Secure Processing Unit for Mobile processors (SPU-M, also called “key algorithm module”), together forming a security engine. *See* Ex-1008 ¶54; Fig. 3. Figure 3, reproduced with highlight below, shows the HSM and the SPU-M.

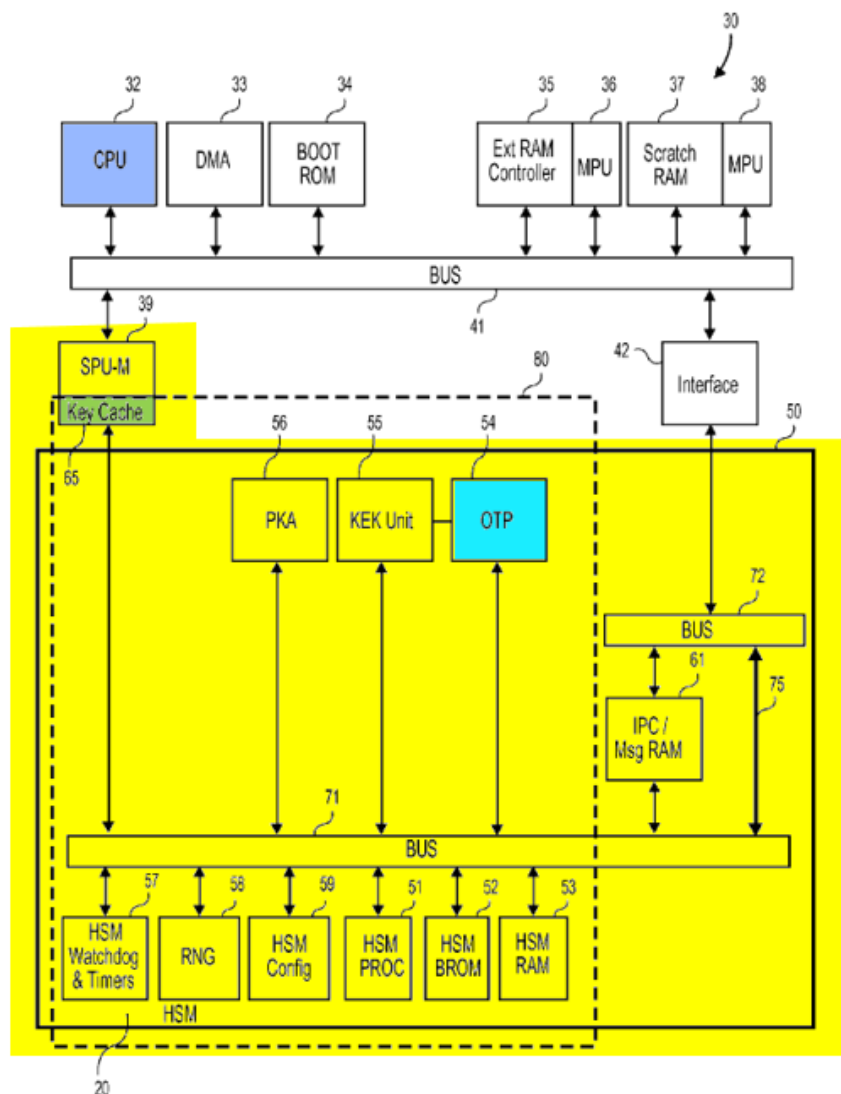


FIG. 3

Ex-1008, Fig. 3. Figure 1, reproduced with highlights showing the security engine below, shows a simplified block diagram of the device incorporating an HSM. Below I will primarily refer to the description of the elements and the numeric designations as shown in Figure 3, but will also refer to descriptions relating to Figure 1, where corresponding elements are discussed.

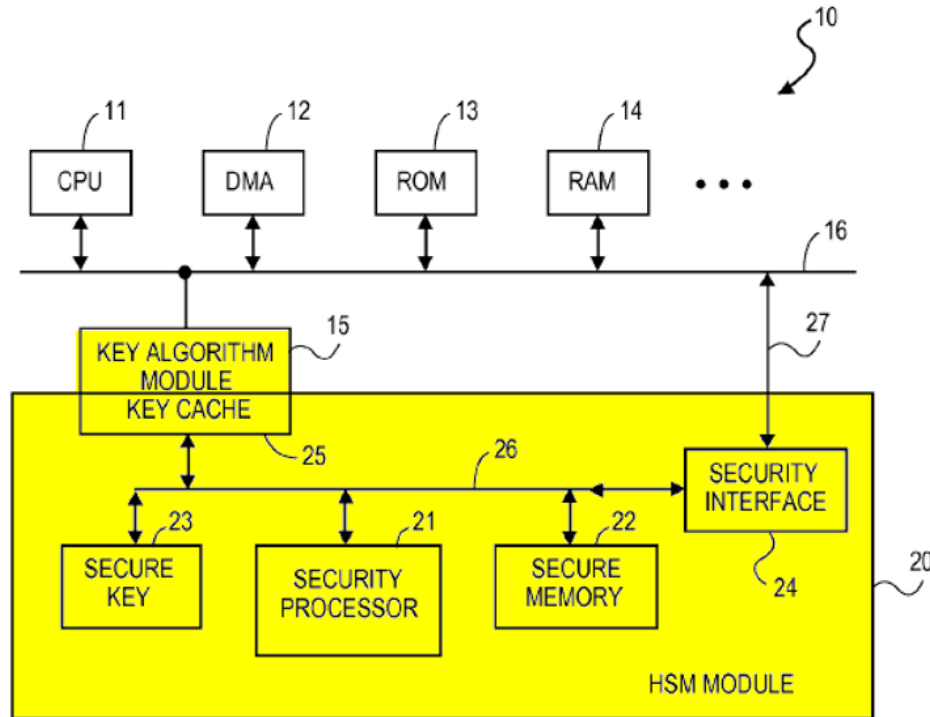


FIG. 1

Ex-1008, Fig. 1.

106. As shown in Figure 3 above, the security engine (HSM and SPU-M, highlighted yellow) are separate from the processor core of the system (the CPU 32, highlighted blue). For example, Chou explains that its device includes a CPU (processor 32 in Figure 3, CPU 11 in Figure 1), separately from the HSM (which has its own processor, such as processor 51 in Figure 3) and the SPU-M (a “Secure Processing Unit for Mobile Processors”):

[0030] FIG. 3 is an embodiment of a device or system 30 that utilizes a more detailed HSM than the HSM of FIG. 1. The various components in the non-secure domain or zone are shown coupled to a bus 41. In the particular embodiment of device 30, the shown components in the

non-secure domain include a processor **32** (labeled as CPU), DMA **33**, Boot ROM (BROM) **34**, external RAM controller **35** for controlling accesses to an external RAM, with accompanying MPU (memory partition unit) **36**, and scratch RAM **37**, with accompanying MPU **38**.

[0031] Processor 32 is equivalent to CPU 11 of FIG. 1 in that processor 32 provides central processing for device 30. In one embodiment, processor **32** is operable to use several processor security levels of operation, including a supervisor mode and a user mode. However, these processor security levels of operation are separate from the secure zone operation of HSM 20 and whether one or more processor security levels are present (or not present) do not affect the described operation of HSM 20.

Ex-1008 ¶¶30-31.

[0034] HSM **20** in this embodiment includes a number of hardware and software components that encompass a secure domain boundary as shown in FIG. 3. A solid-lined box **50** represents the boundary formed by the actual hardware components of HSM **20**, while dashed-line box **80** represents the actual secure domain boundary that is not accessible by non-secure components. HSM 20 of FIG. 3 includes a HSM processor 51, HSM Boot ROM 52, HSM RAM 53, HSM watchdog and timers module 57, RNG (random number generator) 58, HSM configuration module 59, PKA (Public Key Accelerator) module 56, OTP (One Time Programmed memory) 54, KEK (Key-Encryption Key) unit 55, and IPC/Msg RAM (Inter Processor Communication/Message RAM) module 61. Two different buses **71** and **72** are also present within HSM **20**.

Ex-1008 ¶34.

[0054] Furthermore, similar to cache **25** of FIG. 1, HSM 20 also provides a connection to a key cache 65 of a Secure Processing Unit for Mobile processors (SPU-M) module

39, which is equivalent to key algorithm module of 15 of FIG. 1. In the particular embodiment shown, SPU-M module 39 operates as a hardware accelerator for symmetric key algorithms. In other embodiments there may be multiple such SPU-M units, with respective caches. In one embodiment, key cache 65 has capacity to store 512 bytes and cache 65 is organized with a small table to describe the key type, the starting location and the size of each key entry. The table is physically stored in the same memory as the key entries. As shown, HSM software may access the entire key cache memory from bus 71. The algorithm(s) of SPU-M module 39 may access key cache 65, which contains the unwrapped keys, but the contents of key cache 65 are not retrievable by software running on non-secure components outside of secure domain 80. For security protection, reading from the key cache may be disabled in certain specified situations.

Ex-1008 ¶54.

107. **“the security engine . . . has a secure memory accessible only by the security engine”**: The security engine (HSM and SPU-M) has memory areas, including the OTP 54 (highlighted cyan), and Key Cache 65 (highlighted green), together forming the claimed “secure memory.” Each of these memory areas are accessible only by the security engine (HSM and SPU-M). For example, Figure 3 shows both Key Cache 65 and OTP 54, inside the dashed line (highlighted in red below). Chou also explains that “[t]he solid-lined box 50 represents the boundary formed by the actual hardware components of HSM 20, while dashed-line box 80 represents the actual secure domain boundary that is not accessible by non-secure components.” Ex-1008 ¶34. In other words, the Key Cache 65 and OTP 54 (i.e.,

the “secure memory”) are accessible only by the components within the dashed line (portions of HSM and SPU-M, together the claimed “security engine”).

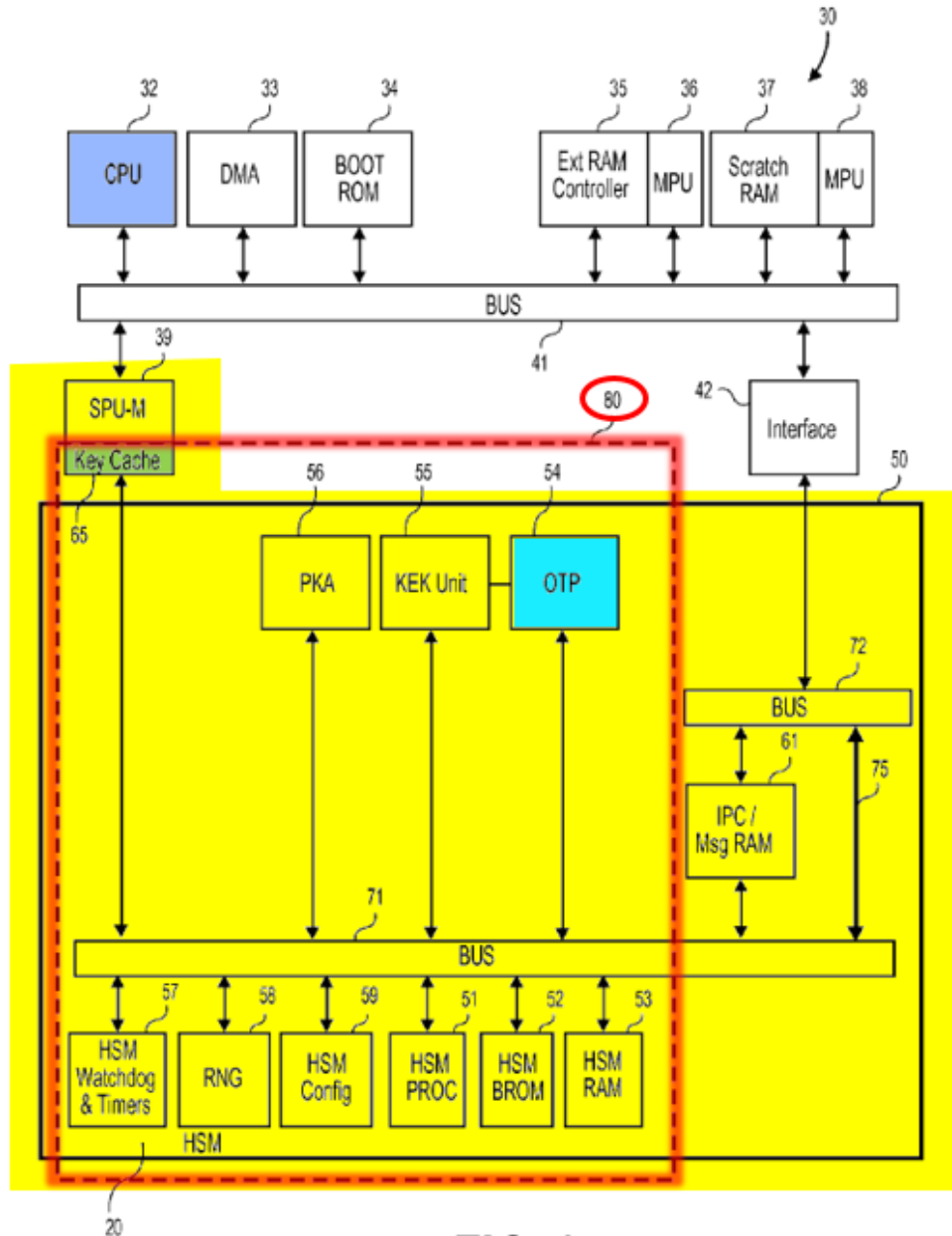


FIG. 3

Ex-1008, Fig. 3.

108. In Figure 1, the secure memory areas inside the security engine are

shown as “secure memory **22**” (highlighted in cyan), key cache **25** (green) and secure key module **23** (orange).

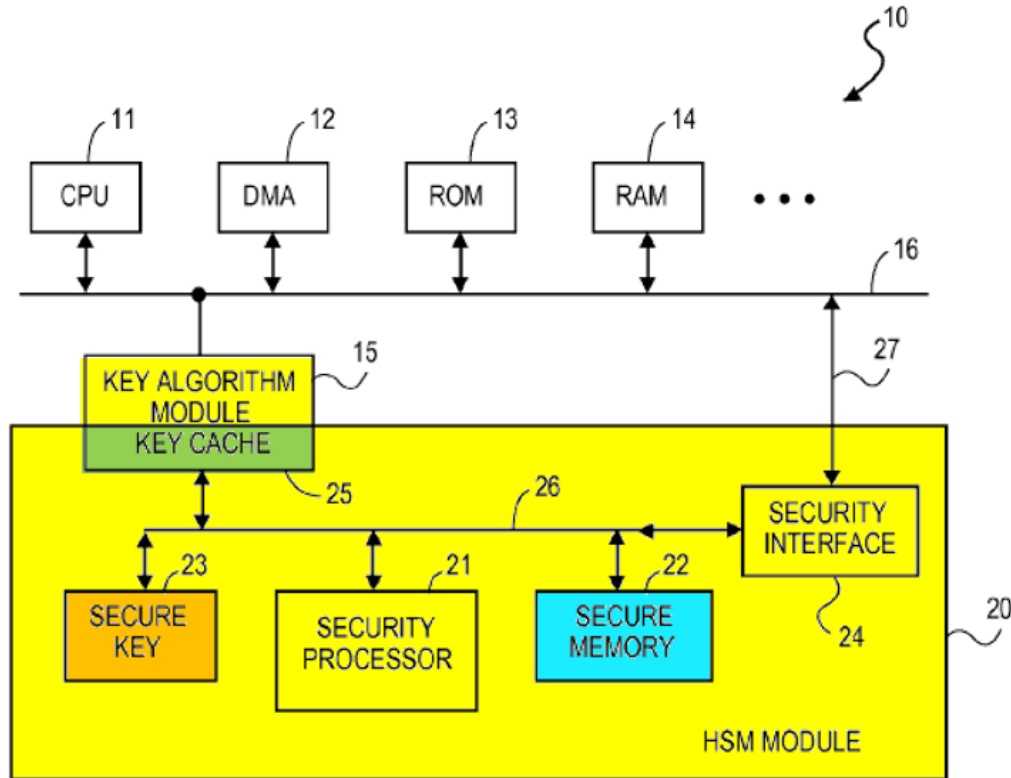


FIG. 1

Ex-1008, Fig. 1.

[0021] HSM **20** includes a security processor **21**, secure memory **22**, secure key module **23**, secure bus **26** and a security interface **24**. The various components **21-24** and **26** reside within a secure boundary established by the hardware. That is, a secure domain or zone is established by and for the components within HSM **20**, in order to set a secure boundary for processing that occurs within this boundary. Security processor **21** provides the various processing within HSM **20**, including key authentication and the encrypting and/or decrypting operation to encrypt and/or decrypt data. Secure key

module 23 is a storage medium, such as a memory, register, circuit, etc. that stores one or more key value(s) that is used by security processor 21 to encrypt and decrypt data. Secure memory 22 is utilized to store data that is processed and/or to be processed by security processor 21.

Ex-1008 ¶21.

[0026] In some embodiments, a key algorithm module 15 may be present, as shown in the example device 10 of FIG. 1. Key algorithm module 15 resides in the non-secure domain and contains one or more algorithms to encrypt/decrypt data. The algorithms, which are typically software routines for providing encryption/decryption, may be resident within key algorithm module 15 or loaded into key algorithm module 15. However, in order to perform the encryption/decryption, a secure key is needed. In order to ensure that an unwrapped key is not accessible by components in the non-secure domain, key algorithm module 15 includes a key cache 25, which resides within the secure domain. When an operating algorithm within key algorithm module 15 requires a secure key, security processor 21 unwraps the secure key and places the unwrapped key contents in key cache 25 for use by an algorithm in key algorithm module 15. Although the content of key cache 25 may be utilized, the content of cache 25 may not be copied or otherwise accessed by components of the non-secure domain. Once the key is no longer needed by key algorithm module 15, the cache content of key cache 25 is invalidated, evicted or erased to prevent unauthorized access.

Ex-1008 ¶26.

109. **“wherein the secure memory includes a security key that was encoded in the secure memory during a manufacturing process of the system-**

on-a-chip”: The OPT **54** of Chou (part of the “secure memory”) includes a “device unique key” that is programmed at the time of manufacturing of the SoC, and stored in OTP 54. Ex-1008 ¶¶34, 38. For example, Chou explains that the OTP (One Time Programmed memory) holds the “device unique key”:

[0038] In respect to secure key processing, HSM configuration module **59** establishes the operating configuration for the processors and/or accelerators that are used to wrap/unwrap and authenticate keys. HSM watchdog and timers module **57** provides the necessary clocks and timers that are utilized for key processing, as well as monitoring of key processing. OTP **54** is a non-volatile memory that is used to one-time program a key value or values for device **30**. For example, the device unique key may be programmed and stored in OTP **54**.

Ex-1008 ¶38. Chou also explains that the OTP is programmed during manufacturing:

[0064] As noted previously, the use of a non-firewall mode allows a same integrated circuit (IC) chip to be two different devices. One where HSM firewall is used and another where HSM firewall is disabled. Connection **75** is disabled when secure operation of HSM is desired. In one embodiment, disabling of connection **75** is achieved by programming a non-volatile bit in OTP **54** during manufacturing. A non-secure mode may be utilized during initial testing and productization. In the non-secure mode, unwrapped key(s) may be accessed by components outside of HSM **20**.

Ex-1008 ¶64.

110. Although the above-quoted description in Chou paragraph 64 relates to programing of a non-volatile bit in the same OTP **54** that controls the mode of

operation (whether to enable or disable the connection **75** shown in Figure 3) rather than the programming of the device unique key itself, a POSITA would have understood Chou to disclose that the OTP **54** would be programmed during manufacturing of the SoC, and in turn, understood that the “device unique key,” which is also stored in the OTP **54**, would also be programmed during manufacturing of the SoC.

- c. **1[b] generate a random nonce for initiating a request for a secure communication session with a remote server over a network using the nonce;**

111. Chou discloses that HSM includes a random number generator (RNG) (highlighted pink).

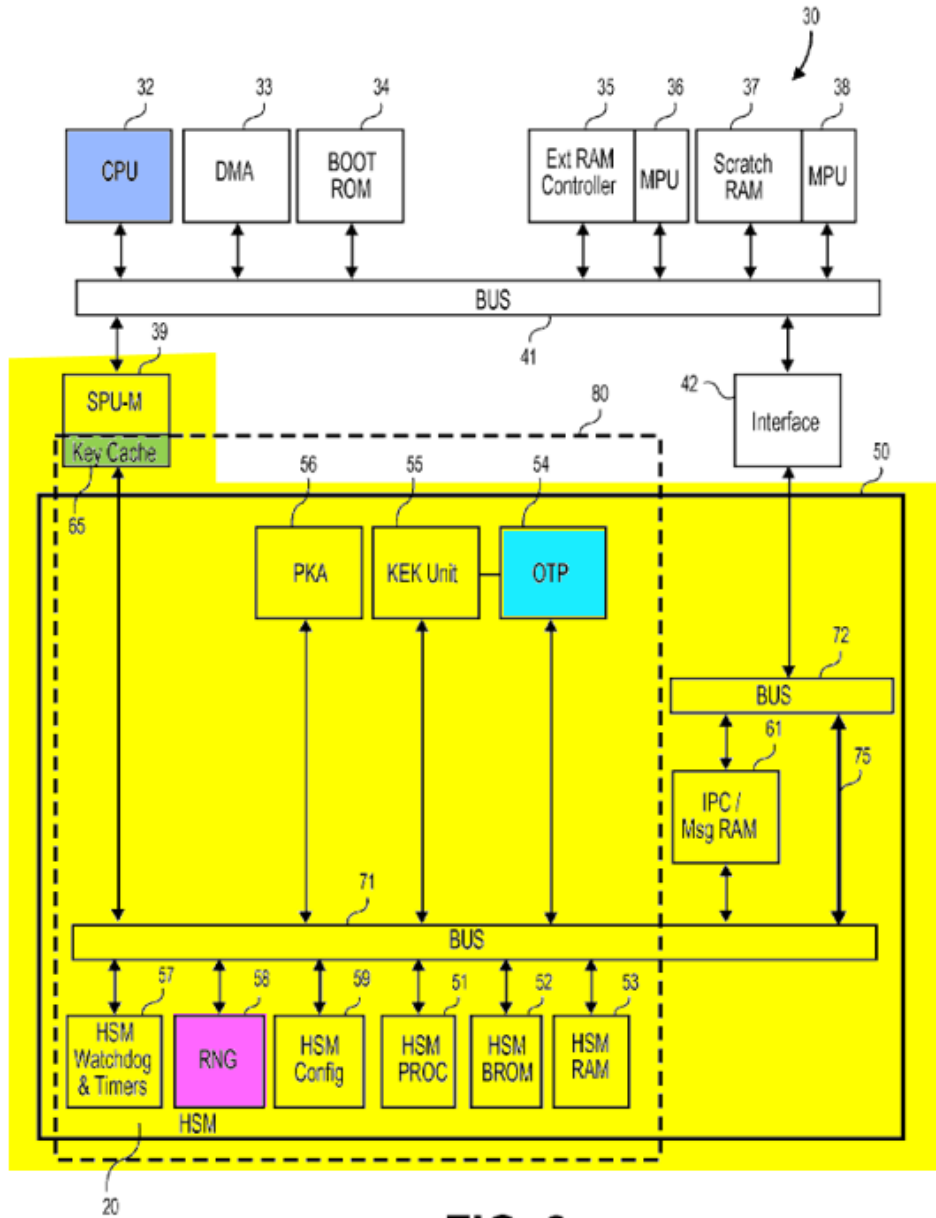


FIG. 3

Ex-1008, Fig. 3.

112. The RNG is used to generate true random number (i.e., a random nonce) that is used as a seed for cryptographic algorithms. For example,

[0034] HSM 20 in this embodiment includes a number of hardware and software components that encompass a secure domain boundary as shown in FIG. 3. A solid-lined

box **50** represents the boundary formed by the actual hardware components of HSM **20**, while dashed-line box **80** represents the actual secure domain boundary that is not accessible by non-secure components. HSM **20** of FIG. 3 includes a HSM processor **51**, HSM Boot ROM **52**, HSM RAM **53**, HSM watchdog and timers module **57**, RNG (random number generator) **58**, HSM configuration module **59**, PKA (Public Key Accelerator) module **56**, OTP (One Time Programmed memory) **54**, KEK (Key-Encryption Key) unit **55**, and IPC/Msg RAM (Inter Processor Communication/Message RAM) module **61**. Two different buses **71** and **72** are also present within HSM **20**.

Ex-1008 ¶34.

[0042] RNG **58** uses a random bit generator to generate true random numbers. In one embodiment, RNG **58** utilizes free running oscillators to capture thermal noise as the source of randomness and is controlled via configuration registers and generates one bit at a time into a **512** bit capture FIFO (First-In-First-Out). The generated random value is typically used as a seed for a cryptographic algorithm and the value from RNG **58** is typically post processed and not used directly. In one embodiment, the random bit rate is a nominal **1** MHz generation rate providing **1** Mbps of random data.

Ex-1008 ¶42.

113. A POSITA would have understood that, based on Chou's disclosure that the generated random number is used as seed for a cryptographic algorithm (Ex-1008 ¶42), the random number would have been thus used during the cryptographic key exchange that requires random seed. For example, Chou explains that the HSM also includes PKA (Public Key Accelerator) module **56**

(Ex-1008 ¶34, Fig. 3), which provides acceleration for a number of algorithms, including Diffie-Hellman public value and shared key computation (Ex-1008 ¶43). As explained above in Section VII.A.2, a POSITA would have understood that Diffie-Hellman is a key exchange algorithm to establish a shared encryption key between two different entities to establish a secure session, and starts with each side generating a random number. Thus, a POSITA would have understood that the random number used for initiating the secure session using the Diffie-Hellman key exchange method (i.e., the random seed for the Diffie-Hellman key exchange) would be generated by the RNG 58 within the HSM.

114. Chou further discloses that the HSM establishes a secure communication session with a remote server using the nonce. For example, Chou explains that its invention is used to securely manage the various session keys, which include the keys used in the context of financial transaction with a server over a network, such that the keys are not revealed to unauthorized resources:

[0006] As an example, secure data transfer is utilized to perform secure financial transactions over a network, such as the Internet. In a typical secure transaction, data content that contains financial information, such as a credit/debit card information, are encrypted and transmitted to a designated recipient through an unsecure network. The recipient of the secure data utilizes a secure key to decrypt the data to retrieve the financial information.

...

[0009] Therefore, a need exists to provide a more secure

key management scheme where a secure key may be unwrapped by a recipient, but the key not be revealed to the recipient and/or unauthorized resources.

Ex-1008 ¶¶6-9. Thus, a POSITA would have understood that the RNG in Chou's HSM is used for the HSM's operation, such as the secure communication with a remote server.

d. 1[c] perform a cryptographic key exchange with the remote server;

115. Chou's HSM includes a Public Key Accelerator (PKA) module. The PKA module also performs a number of symmetric key algorithms, which a POSITA would have known to result in generation of a symmetric session key. Those algorithms include the "Diffie-Hellman public value and shared key computation," "RSA encryption," "RSA decryption . . . using Chinese Remainder Theorem," "RSA key generation with Rabin-Miller number selection," and the "elliptic curve Diffie-Hellman." Ex-1008 ¶43. As explained above in Section VII.A.2, a Diffie-Hellman or RSA are methods of establishing a shared key between two separate entities, such as a client device and a remote server (i.e., they are cryptographic key exchange methods). Thus, Chou's HSM, which includes the PKA, performs the cryptographic exchange with a remote server.

[0043]-[0053] PKA module 56 is a specialized processor that provides acceleration for a number of algorithms. In one embodiment, PKA module 56 is an accelerator that is operable for the following algorithms:

1. Diffie-Hellman public value and shared key computation of modulus sizes up to 4096 bits.
2. 1024-bit DSA signature with SHA-1 hash or 2048-bit DSA signature with SHA-256 hash.
3. RSA encryption of modulus sizes up to 4096-bit key size.
4. RSA decryption of modulus sizes up to 4096-bits using Chinese Remainder Theorem.
5. RSA key generation with Rabin-Miller number selection.
6. Elliptic curve Diffie-Hellman in prime field GF(p).
7. Prime field EC-DSA signature for modulus sizes up to 512-bit.
8. Generic prime field ECC point operations.
9. Generic long integer math operations.
10. NIST prime field ECC curve optimization.

It is to be noted that these algorithms are listed for exemplary purpose and that other embodiments of PKA module 56 may support a different list of algorithms, including algorithms not listed above.

Ex-1008 ¶¶43-53.

116. In addition, Chou explains that the HSM “includes a security processor to unwrap and authenticate a secure key within the secure boundary to decrypt or encrypt data and to provide data through a single interface that communicates with external sources, so that all data transfers between the secure boundary, formed by the hardware security module, and external sources are

transferred only through the interface.” Ex-1008, Abstract. “As an example, secure data transfer is utilized to perform secure financial transactions over a network, such as the Internet.” Ex-1008 ¶6. In another example, Chou also explains that “a content provider may provide encrypted content with a content key to system **10** and the unwrapping of the content key is performed strictly within HSM **20**, such as by using the device unique key stored in secure key module **23**.” Ex-1008 ¶25. Thus, Chou discloses HSM providing secure communication with a remote server, such as for financial transaction through internet or with a remote content provider.

117. In addition, Chou discloses an SPU-M module (part of the “security engine”) that operates as a hardware accelerator for symmetric key algorithms. The SPU-M module is connected to the HSM and can access the contents of the key cache, which is located within the secure domain. Ex-1008 ¶54. Chou discloses that the SPU-M may implement “many symmetric key algorithms such as AES, DES, RC4, SHA1-HMAC, SHA2-HMAC, MD5-HMAC, as well as others.” *Id.* Thus, a POSITA would have further understood that HSM would perform a key exchange to establish a symmetric session key.

[0054] Furthermore, similar to cache **25** of FIG. 1, HSM **20** also provides a connection to a key cache **65** of a Secure Processing Unit for Mobile processors (SPU-M) module **39**, which is equivalent to key algorithm module of **15** of FIG. 1. In the particular embodiment shown, SPU-M module **39** operates as a hardware accelerator for

symmetric key algorithms. In other embodiments there may be multiple such SPU-M units, with respective caches. In one embodiment, key cache 65 has capacity to store 512 bytes and cache 65 is organized with a small table to describe the key type, the starting location and the size of each key entry. The table is physically stored in the same memory as the key entries. As shown, HSM software may access the entire key cache memory from bus 71. The algorithm(s) of SPU-M module 39 may access key cache 65, which contains the unwrapped keys, but the contents of key cache 65 are not retrievable by software running on non-secure components outside of secure domain 80. For security protection, reading from the key cache may be disabled in certain specified situations.

[0055] In one embodiment, SPU-M module 39 may implement many symmetric key algorithms such as AES, DES, RC4, SHA1-HMAC, SHA2-HMAC, MD5-HMAC, as well as others. It is to be noted that these algorithms are listed for exemplary purpose and that other embodiments of SPU-M module 39 may support a different list of algorithms, including algorithms not listed above. Keys for these algorithms may be loaded by HSM 20 into key cache 65. Components outside of the HSM secure zone, such as CPU 32, may then send encrypted/decrypted data along with a pointer to a corresponding key entry in key cache 65. SPU-M module 39 receives the payload and accesses the correct key from key cache 65 and perform decryption/encryption operations.

Ex-1008 ¶¶54-55.

- e. **1[d] generate a symmetric session key, based on the cryptographic key exchange, to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session;**

118. As explained above, Chou discloses a PKA module that performs a

number of symmetric key exchanges, such as Diffie-Hellman key exchange, RSA key generation, and Elliptic curve Diffie-Hellman exchange, among others. See Ex-1008 ¶43 (the HSM of Chou specifically supports “Diffie-Hellman public value and shared key computation of modulus sizes up to 4096 bits.”). As explained above in Section VII.A.2, a Diffie-Hellman (or its variant, Elliptic curve Diffie-Hellman) is a key exchange algorithm that allows the two parties at both ends of the communication to separately generate a common (i.e., symmetric) session key for a secure message exchange. Thus, Chou discloses generating a symmetric session key based on the Diffie-Hellman exchange.

119. In addition, Chou’s SPU-M (also part of the claimed “security engine”) is “a hardware accelerator for *symmetric key algorithms*” that “implement[s] many *symmetric* key algorithms such as AES, DES, RC4, SHA1-HMAC, SHA2-HMAC, MD5-HMAC, as well as others.” Ex-1008 ¶¶54-55.

[0054] Furthermore, similar to cache **25** of FIG. 1, HSM **20** also provides a connection to a key cache **65** of a Secure Processing Unit for Mobile processors (SPU-M) module **39**, which is equivalent to key algorithm module of **15** of FIG. 1. In the particular embodiment shown, SPU-M module **39** operates as a hardware accelerator for symmetric key algorithms. In other embodiments there may be multiple such SPU-M units, with respective caches. In one embodiment, key cache **65** has capacity to store **512** bytes and cache **65** is organized with a small table to describe the key type, the starting location and the size of each key entry. The table is physically stored in the same memory as the key entries. As shown, HSM software may access the entire key cache memory from bus **71**. The

algorithm(s) of SPU-M module **39** may access key cache **65**, which contains the unwrapped keys, but the contents of key cache **65** are not retrievable by software running on non-secure components outside of secure domain **80**. For security protection, reading from the key cache may be disabled in certain specified situations.

[0055] In one embodiment, SPU-M module **39** may implement many symmetric key algorithms such as AES, DES, RC4, SHA1-HMAC, SHA2-HMAC, MD5-HMAC, as well as others. It is to be noted that these algorithms are listed for exemplary purpose and that other embodiments of SPU-M module **39** may support a different list of algorithms, including algorithms not listed above. Keys for these algorithms may be loaded by HSM **20** into key cache **65**. Components outside of the HSM secure zone, such as CPU **32**, may then send encrypted/decrypted data along with a pointer to a corresponding key entry in key cache **65**. SPU-M module **39** receives the payload and accesses the correct key from key cache **65** and perform decryption/encryption operations.

Ex-1008 ¶¶54-55. In other words, the PKA generates the shared keys (for example, through the Diffie-Hellman exchange) to be used in the communication with the remote server, and the SPU-M encrypts and decrypts data exchanged with the remote server using the keys generated by PKA, using *symmetric* key algorithms.

120. In addition, Chou discloses a “secure key,” such as the session key to be established through a Diffie-Hellman exchange, that is used to “encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session” as claimed. For example, Chou

explains:

[0005] Secure key mechanisms are utilized to convey encrypted data, so that only those having the proper key or keys are able to authenticate and decrypt the data. Typically, a sender will encrypt the data that is to be transmitted, in which the data, context data, or content key is “wrapped” to prevent unauthorized access. The recipient of the encrypted data utilizes a content key or a key that is unique to the receiving device to “unwrap” and authenticate to obtain access to the decrypted data. The key may be resident within the recipient or sent by the sender, in which case the key that is sent may also be “wrapped” to protect it from unauthorized access. An authorization key may also be transmitted through a hierarchy of devices and installed in the recipient by traversing a chain of trust through those devices. A variety of techniques are known, such as private key/public key exchanges, for transmitting data securely.

[0006] As an example, secure data transfer is utilized to perform secure financial transactions over a network, such as the Internet. In a typical secure transaction, data content that contains financial information, such as a credit/debit card information, are encrypted and transmitted to a designated recipient through an unsecure network. The recipient of the secure data utilizes a secure key to decrypt the data to retrieve the financial information.

Ex-1008 ¶¶5-6.

121. As quoted above, Chou explains that “a sender will encrypt the data that is to be transmitted” (Ex-1008 ¶5) and “the recipient of the secure data utilizes a secure key to decrypt the data” (¶6) using the secure key mechanism. In other words, the “secure key” is used to encrypt and decrypt the data transmitted between two parties (e.g., in a secure financial transactions), and therefore is a

“symmetric session key” for the transaction session.

f. 1[e] encrypt the symmetric session key based on the security key; and

122. Chou discloses that HSM performs wrapping (i.e., encryption) of the secure keys. For example, Chou explains that “Components **54-59**” shown in Figure 3 (namely, OTP (One Time Programmed memory) **54** (highlighted in cyan), KEK (Key-Encryption Key) unit **55** (light green), PKA (Public Key Accelerator) module **56** (purple), HSM watchdog and timers module **57** (orange), RNG (random number generator) **58** (pink), and HSM configuration module **59** (brown) (*see* Ex-1008 ¶34)) are “operable to provide processing operations that pertain to secure key functions, including the *wrapping and unwrapping of keys*, context data, etc.” Ex-1008 ¶37.

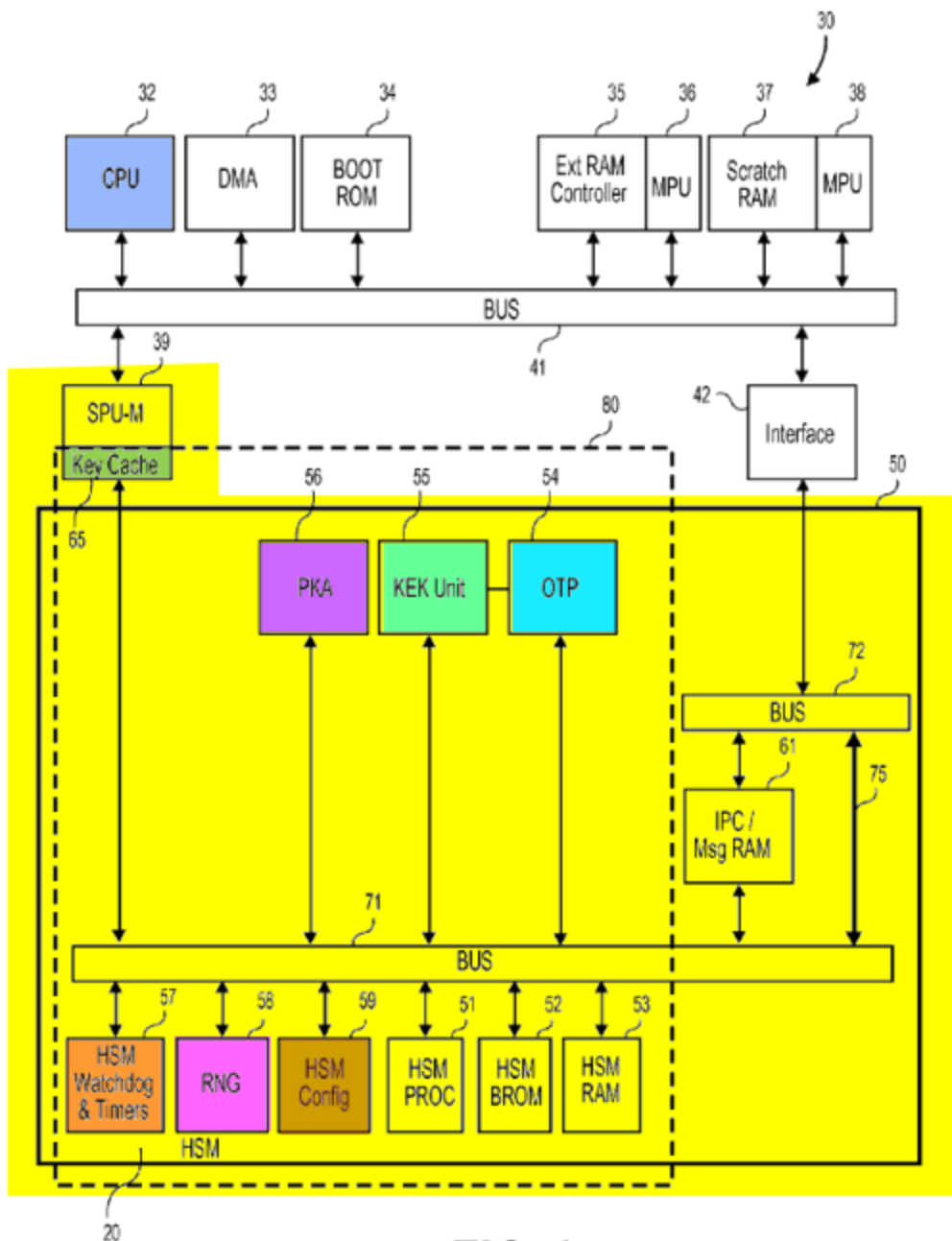


FIG. 3

Ex-1008, Fig. 3.

123. Chou further clarifies that the “secure keys” that are being wrapped and unwrapped would include “symmetric session key” generated by the key exchange procedure. For example, Chou explains that “[t]he embodiments of the

present invention may be practiced in a variety of settings that utilize a secure key to authenticate and unwrap and/or decrypt data, context data and/or a secure key.”).

Ex-1008 ¶17. “A POSITA would have recognized that a “secure key” that unwrap and/or decrypt data is a “session key.”

124. Chou further discloses that the “secure key” is wrapped/unwrapped using a “device unique key” (i.e., the security key) by the HSM, inside the secure zone (identified by the red highlighted box in Figure 3 reproduced above). For example, Chou explains that a “content key”, which is a type of a “secure key”, is unwrapped using the device unique key:

[0025] It is to be noted that since the unwrapping of secure keys is done solely within HSM and since the only route to internal hardware components of HSM **20** is through security interface **24**, any unauthorized attempts from the non-secure domain to access an unwrapped key is prevented. In this manner, a content provider may provide encrypted content with a content key to system **10** and the unwrapping of the content key is performed strictly within HSM **20**, such as by using the device unique key stored in secure key module **23**. Once the content data is decrypted, the content may be provided to components within the non-secure domain, again through the security interface. HSM **20** ensures that a secure key is unwrapped only within HSM **20** and that secure keys and/or secure context data that leave HSM **20** are always in a wrapped protected state. Preventing unauthorized access to secure keys, such as content keys, ensures that the secure key is always protected in the unwrapped state within the secure zone and not accessed or copied by unauthorized entities from outside of the secure zone. Anytime a secure key or content data leaves the secure zone of HSM **20**, the key or content is in a wrapped state.

Ex-1008 ¶25.

[0069] Accordingly, a technique for providing secure key operation, in which a secure zone boundary is defined by components that make up a hardware security module (HSM) is described. Secure keys are wrapped/unwrapped and authenticated within the secure zone and do not leave the HSM in an unwrapped state, nor are the unwrapped keys accessed or copied by unauthorized source external to the secure boundary established by the HSM.

Ex-1008 ¶69.

g. 1[f] store the encrypted session key in the secure memory,

125. Chou further discloses storing the data processed by the security processor, such as the session key wrapped by the HSM, in the secure memory.

For example, Chou discloses a Hardware Security Module (HSM), which includes secure memory used to store the data processed by the security processor.

[0021] HSM 20 includes a security processor 21, secure memory 22, secure key module 23, secure bus 26 and a security interface 24. The various components 21-24 and 26 reside within a secure boundary established by the hardware. That is, a secure domain or zone is established by and for the components within HSM 20, in order to set a secure boundary for processing that occurs within this boundary. Security processor 21 provides the various processing within HSM 20, including key authentication and the encrypting and/or decrypting operation to encrypt and/or decrypt data. Secure key module 23 is a storage medium, such as a memory, register, circuit, etc. that stores one or more key value(s) that is used by security processor 21 to encrypt and decrypt data. Secure memory 22 is utilized to store data that is processed and/or to be processed by security

processor 21.

Ex-1008 ¶ 21. Figure 1, reproduced with highlights showing the security engine below, shows a simplified block diagram of the device incorporating an HSM, with secure memory 22 (cyan) and secure key module 23 (orange) inside the HSM.

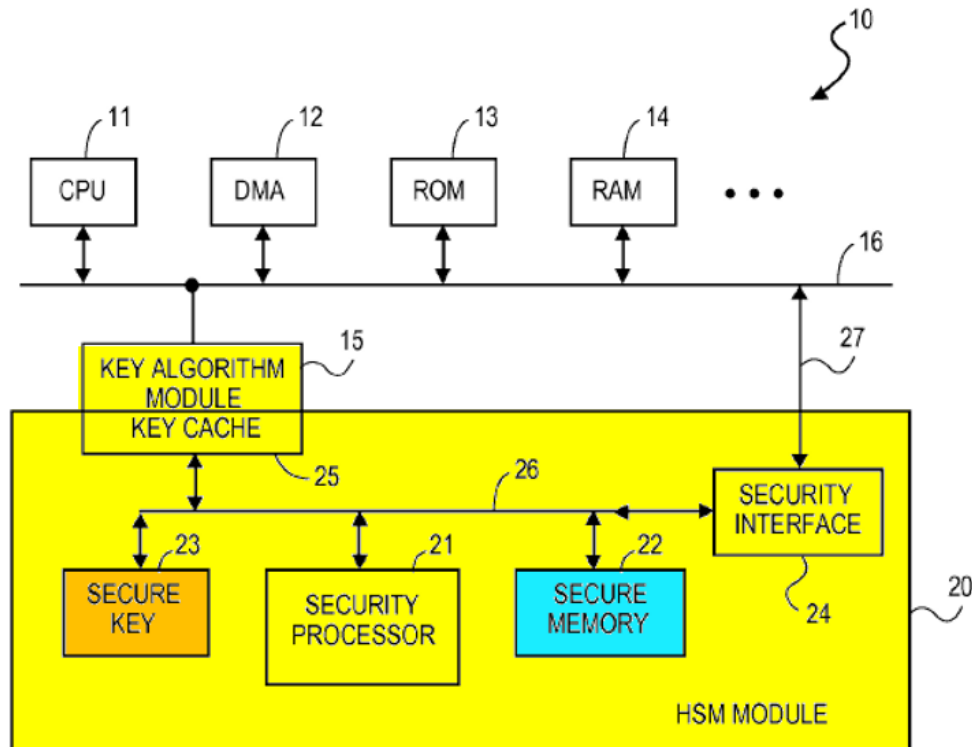


FIG. 1

Ex-1008, Fig. 1.

126. The encrypted session key is one of the data that is processed by the security processor. As explained above, the “secure key” (the session key) is wrapped/unwrapped using a “device unique key” (i.e., the security key) by the HSM, inside the secure zone (identified by the red highlighted box in Figure 3

reproduced above). Ex-1008 ¶25 (“the unwrapping of the content key is performed strictly within HSM **20**, such as by using the device unique key stored in secure key module”); ¶69 (“Secure keys are wrapped/unwrapped and authenticated within the secure zone and do not leave the HSM in an unwrapped state.”) Thus, the encrypted session key is also stored in the secure memory **22**, which is the memory location where the data processed by HSM is stored.

127. In addition, Chou explains “[k]eys for [the symmetric key algorithms] may be loaded by HSM **20** into key cache **65**.” Ex-1008 ¶55. This “key cache” is a volatile memory, as explained below:

[0026] In some embodiments, a key algorithm module **15** may be present, as shown in the example device **10** of FIG. 1. Key algorithm module **15** resides in the non-secure domain and contains one or more algorithms to encrypt/decrypt data. The algorithms, which are typically software routines for providing encryption/decryption, may be resident within key algorithm module **15** or loaded into key algorithm module **15**. However, in order to perform the encryption/decryption, a secure key is needed. In order to ensure that an unwrapped key is not accessible by components in the non-secure domain, key algorithm module **15** includes a key cache **25**, which resides within the secure domain. When an operating algorithm within key algorithm module **15** requires a secure key, security processor **21** unwraps the secure key and places the unwrapped key contents in key cache **25** for use by an algorithm in key algorithm module **15**. Although the content of key cache **25** may be utilized, the content of cache **25** may not be copied or otherwise accessed by components of the non-secure domain. Once the key is no longer needed by key algorithm module **15**, the cache

content of key cache 25 is invalidated, evicted or erased to prevent unauthorized access.

Ex-1008 ¶26. In addition, a POSITA would have readily understood that the term “cache” refers to a type of volatile memory.

128. Because the “key cache” is a volatile memory, a POSITA would have understood that the keys would be stored elsewhere in the HSM, such as the secure key module 23 that stores the key values (such as the secure keys) used to encrypt and decrypt data or the secure memory 22, which is the memory location where the data processed by HSM is stored. Ex-1008 ¶21. Thus, POSITA would understand that the symmetric session key would be stored in a storage area (such as the secure key module 23 or the secure memory 22) and loaded into key cache for the SPU-M to use in symmetric encryption.

h. 1[g] the system-on-a-chip to establish the secure communication session with the remote server over the network using the session key.

129. Chou’s SoC is used to establish secure communication session with a remote server, such as a financial transaction sessions. One of the examples Chou list as utilizing a secure key is a “secure data transfer [] utilized to perform secure financial transactions over a network, such as the Internet.” Ex-1008 ¶ 6. The transaction is protected by a “secure key.” *Id.* Chou’s invention is directed to providing secure key management scheme to manage the handling of such secure keys. Thus, a POSITA would have understood that Chou’s HSM is designed to be

used in SoCs that establish the secure communication session with a remote server over the network using the session key.

[0006] As an example, secure data transfer is utilized to perform secure financial transactions over a network, such as the Internet. In a typical secure transaction, data content that contains financial information, such as a credit/debit card information, are encrypted and transmitted to a designated recipient through an unsecure network. The recipient of the secure data utilizes a secure key to decrypt the data to retrieve the financial information.

...

[0009] Therefore, a need exists to provide a more secure key management scheme where a secure key may be unwrapped by a recipient, but the key not be revealed to the recipient and/or unauthorized resources.

Ex-1008 ¶¶6-9.

2. Claim 24

- a. **24. The system-on-a-chip apparatus of claim 1, wherein the security engine is communicatively coupled to the processor core through an interconnect of the system-on-a-chip.**

130. Chou discloses a connection between the security engine (the HSM **20** and SPU-M **15**) and a CPU through a security interface that connects to a connection or bus (connection **27**, between the security interface and the connection to CPU). The HSM **20** is communicatively coupled to the CPU through the connection that goes through the security interface **24** Ex-1008 ¶24 (“As shown, a connection or bus **27** couples security interface **24** to bus **16**, so that

all accesses to HSM 20 and data transfers into and out of HSM 20 are routed through security interface 24.”); ¶23 (“In operation, when encrypted data is received by device 10, components within the non-secure domain, Such as CPU 11, process the data and transfers the encrypted data to security interface 24. Security interface 24 is utilized as the single interface for data transfers between components of HSM 20 and devices that reside outside of HSM 20.”). Thus, the security engine (HSM 20) is communicatively coupled to the processor core (CPU 11) through an interconnect. This interconnect is an interconnect of the system-on-a-chip, since the entire “device 10, including HSM 20, is constructed on a single integrated circuit chip.” Ex-1008 ¶20.

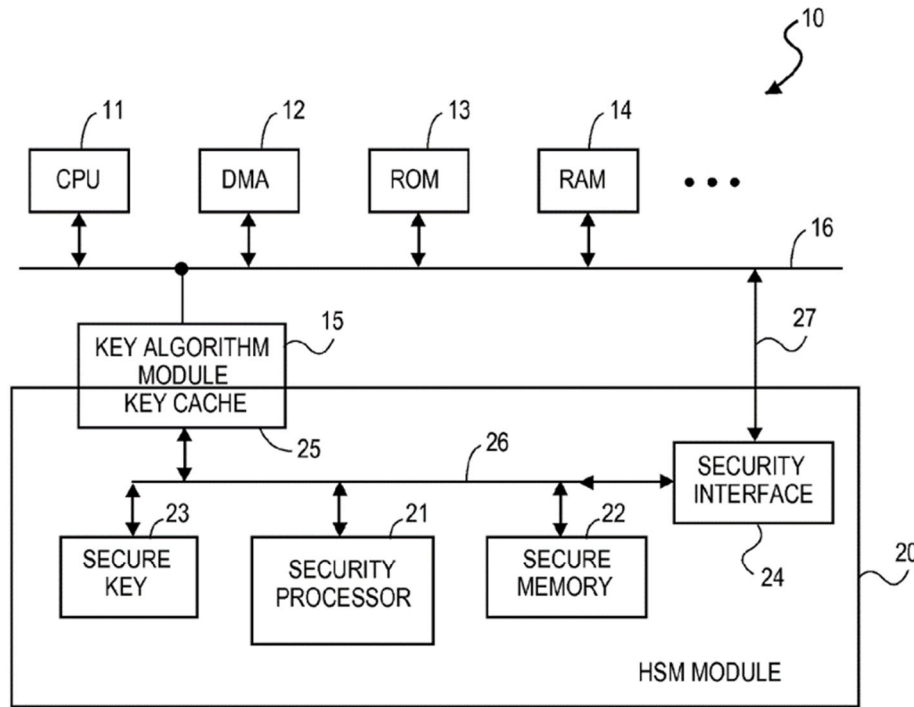


FIG. 1

3. Independent Claims 9, 17

a. 9[pre] A method comprising;

17[pre] One or more non-transitory machine-readable storage media comprising a plurality of instructions stored thereon that, in response to execution by a computing device, cause the computing device to:

131. To the extent limiting, Chou discloses the preamble of Claim 9. For example, Chou states it discloses a method. For example, Chou is titled “Apparatus and Method for Providing Hardware Security.” Ex-1008, Cover. Chou also states:

[0010] The present invention is directed to apparatus and

methods of operation that are further described in the following Brief Description of the Drawings, the Detailed Description of the Invention, and the Claims. Other features and advantages of the present invention will become apparent from the following detailed description of the embodiments of the invention made with reference to the accompanying drawings.

Ex-1008 ¶10.

132. Also to the extent limiting, Buer discloses the preamble of Claim 17.

For example, Chou discloses that the operations may embodied in processors executing appropriate software. A POSITA would recognize that such software would be stored in a non-transitory machine-readable storage media comprising a plurality of instructions that are executed when the device executes:

[0071] The embodiments of the present invention have been described above with the aid of functional building blocks illustrating the performance of certain functions. The boundaries of these functional building blocks have been arbitrarily defined for convenience of description. Alternate boundaries could be defined as long as the certain functions are appropriately performed. One of ordinary skill in the art may also recognize that the functional building blocks, and other illustrative blocks, modules and components herein, may be implemented as illustrated or by discrete components, application specific integrated circuits, processors executing appropriate software and the like or any combination thereof.

Ex-1008 ¶71.

- b. 9[a] generating a random nonce in a security engine that is separate from a processor core of a system-on-a-chip of a client device;**

17[a] generate a random nonce in a security engine that is separate from a processor core of a system-on-a-chip of the computing device;

133. Chou discloses limitations 9[a] and 17[a] for the same reasons I explained in Section **X.A.1.b**, limitation 1[a] (“a system-on-a-chip comprising a security engine that is separate from a processor core of the system-on-a-chip . . .”) and Section **X.A.1.c**, limitation 1[b] (the security engine to “generate a random nonce for initiating a request for a secure communication session with a remote server over a network using the nonce”).

- c. 9[b] initiating, using the client device, a request for a secure communication session with a remote server over a network, the request including the random nonce;**

17[b] initiate a request for a secure communication session with a remote server over a network, the request including the random nonce;

134. Chou discloses limitations 9[b] and 17[b] for the same reasons I explained in Section **X.A.1.c**, limitation 1[b] (“generate a random nonce for initiating a request for a secure communication session with a remote server over a network using the nonce”).

- d. 9[c] performing a cryptographic key exchange , using the security engine of the system-on-a-chip, with the remote server;**

17[c] perform a cryptographic key exchange, using the security engine of the system-on-a-chip, with the remote server;

135. Chou discloses limitations 9[c] and 17[c] for the same reasons I explained in Section **X.A.1.d.** limitation 1[c] (the security engine to “perform a cryptographic key exchange with the remote serve”)

- e. 9[d] generate a symmetric session key to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session;**

17[d] generate, based on the cryptographic key exchange, a symmetric session key to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session;

136. Chou discloses limitations 9[d] and 17[d] for the same reasons I explained in Section **X.A.1.e**, limitation 1[d] (“generate a symmetric session key, based on the cryptographic key exchange, to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session”).

- f. 9[e] encrypting the session key based on a security key that was encoded in a secure memory of the security engine during a manufacturing process of the system-on-a-chip;**

17[e] encrypt the session key based on a security key that was encoded in a secure memory of the security engine during a manufacturing process of the system-on-a-chip;

137. Chou discloses limitations 9[e] and 17[e] for the same reasons I explained in Section **X.A.1.f**, limitation 1[e] (“encrypt the symmetric session key based on the security key”) and Section **X.A.1.b**, limitation 1[a] (“... wherein the secure memory includes a security key that was encoded in the secure memory during a manufacturing process of the system-on-a-chip”).

- g. 9[f] storing the encrypted session key in the secure memory of the security engine of the system-on-a-chip; and**

17[f] store the session key in the secure memory of the security engine of the system-on-a-chip; and

138. Chou discloses limitations 9[f] and 17[f] for the same reasons I explained in Section **X.A.1.g**, limitation 1[f] (“store the encrypted session key in the secure memory”).

- h. 9[g] establishing, using the client device, the secure communication session with the remote server using the session key.**

17[g] establish the secure communication session with the remote server using the session key.

139. Chou discloses limitations 9[f] and 17[f] for the same reasons I explained in Section **X.A.1.h**, limitation 1[g] (“the system-on-a-chip to establish the secure communication session with the remote server over the network using the

B. Ground 2: Chou, in combination with TLS-1.0

140. To the extent Chou, alone or in combination with knowledge of a POSITA (Ground 1) does not disclose any limitations of the challenged claims, those limitations are obvious in further combination with TLS-1.0, as further explained below. The contemplated combination would retain the same disclosed structure of Chou for creating and managing the secure keys, wherein the secure key is created in accordance with SSL/TLS protocol.

1. Motivation to Combine

141. As explained in Section **VII.B** above, SSL protocol and its successor, TLS protocol, specifies a standard protocol to be used by two communicating entities (such as a client device and a web server) to establish an encrypted and authenticated communication channel. The SSL/TLS protocol was widely used by the priority date of the '838 Patent.

142. Chou notes that one of the contexts where a secure key is used is in communication via an unsecure network, such as the internet:

[0006] As an example, secure data transfer is utilized to perform secure financial transactions over a network, such as the Internet. In a typical secure transaction, data content that contains financial information, such as a credit/debit card information, are encrypted and transmitted to a designated recipient through an unsecure network. The recipient of the secure data utilizes a secure key to decrypt the data to retrieve the financial information.

...

[0009] Therefore, a need exists to provide a more secure key management scheme where a secure key may be unwrapped by a recipient, but the key not be revealed to the recipient and/or unauthorized resources.

Ex-1008 ¶¶ 6-9.

143. A POSITA would have recognized that the SSL/TLS protocol was a well-established method of establishing a secure communication transaction through an unsecure network, such as the internet. By the priority date of the '838 Patent, the use of SSL/TLS was a widely-accepted method of securing a communication between two parties over the internet. As such, TLS was widely supported by web browsers, servers, and other applications at the time, making it a standard protocol for secure communication on the internet. A POSITA would have realized that, by adopting the TLS protocol standard for external communication (communication with remote entities), Chou's system could be

ensured to be able to communicate securely with wide variety of servers, including those servers whose design are not explicitly contemplated by Chou. Thus, a POSITA would have been motivated to incorporate TLS protocol as the standard for creating session keys over the internet, in such scenarios where “secure data transfer is utilized to perform secure financial transactions over a network” as contemplated in Chou, Ex-1008 ¶6.

144. Also as explained in Section **VII.B** above, TLS provides mutual authentication of the client and server, ensuring that both parties are who they claim to be. TLS also provides data integrity, ensuring that data exchanged between the client and server has not been tampered with or modified in transit. This helps prevent data corruption and ensures the integrity of the communication. A POSITA would therefore have been motivated to incorporate TLS protocol in the communication scenarios using Chou’s HSM devices, and to use Chou’s HSM device to manage the TLS session key to further secure the session key.

145. A POSITA would have further been motivated to use SSL/TLS protocol, because doing so would allow the implementer to use any one of the many available SSL/TLS open source libraries, rather than writing the software to implement each of the exchange steps from scratch. Utilizing the available SSL/TLS open source libraries would have allowed the implementer to save significant amount of development time, and save development resources and

manpower. Adopting SSL/TLS would have also allowed the implementer to take advantage of the large community of developers who contribute to the code, provide support, and continuously improve the library, which helps to ensure that the library is up-to-date and can adapt to changing technology trends. Open-source libraries are also usually subject to peer review, which can help identify security vulnerabilities and ensure that the library is secure. Thus, a POSITA would have recognized each of these benefits, and would have been motivated to adopt SSL/TLS standard protocol to take advantage of the available open source software. In addition, because open-source libraries are usually well-tested and have gone through multiple revisions to ensure quality and reliability, a POSITA would have had a reasonable expectation of success of implementing the SSL/TLS protocol between the main security module and the satellite security module, based on the widely available open-source community resources and their experiences.

2. Dependent Claim 3

- a. 3. The system-on-a-chip apparatus of claim 1, wherein the security engine is further to generate a pre-master key, encrypt the pre-master key using the security key of the security engine, and send the encrypted pre-master key to the remote server.**

146. The '838 Patent claims encrypting a pre-master key with the security key, but does not describe this subject matter in its specification. Instead, it describes the normal TLS operation, i.e., encrypting a pre-master key with the

server's public key. *See* '838 Patent at 8:28-32; 10:60-64.

147. For purposes of this declaration, I have been asked to assume that the disclosure provided in the '838 Patent, which describes encrypting a pre-master key using the server's public key (rather than the "security key") is sufficient to support Claim 3 as written (i.e., encrypt the pre-master key *using the security key of the security engine*).

148. A POSITA would have found Claim 3 obvious in light of TLS-1.0, which was a well-established and widely-used standard protocol before the priority date of the '838 Patent. A POSITA would have been motivated to combine the HSM device of Chou to manage the session keys created during a communication with a server through internet, as explained above in Section **X.B.1**.

149. TLS-1.0 specifies the use of RSA-encrypted premaster secret message which discloses the client device (such as the SoC device of Chou) creating a premaster secret message, which requires generating a pre-master key, encrypting the pre-master key using the public key of the server (received in a server certificate), and sending the encrypted pre-master key to the server:

7.4.7.1. RSA encrypted premaster secret message

Meaning of this message:

If RSA is being used for key agreement and authentication, the client generates a 48-byte premaster secret, encrypts it using the public key from the server's certificate or the temporary RSA key provided in a server key exchange message, and sends the result in an encrypted premaster secret message. This structure is a variant of the client key exchange message, not a message in itself.

Ex-1006, §7.4.7.1. Thus, TLS-1.0 discloses the same operation as disclosed in the '838 Patent, and renders obvious this claim.

3. Dependent Claim 4

- a. **4. The system-on-a-chip apparatus of claim 3 wherein the security engine is further to wrap the pre-master key using a server public key and store the wrapped pre-master key in the secure memory.**

150. **“the security engine is further to wrap the pre-master key using a server public key”:** As explained in Section X.B.2 (Claim 3, combination with TLS-1.0) above, TLS-1.0 specifies that the premaster secret is encrypted using the server (responder)'s public key. For example TLS-1.0 states:

7.4.7.1. RSA encrypted premaster secret message

Meaning of this message:

If RSA is being used for key agreement and authentication, the client generates a 48-byte premaster secret, encrypts it using the public key from the server's certificate or the temporary RSA key provided in a server key exchange message, and sends the result in an encrypted premaster secret message. This structure is a variant of the client key exchange message, not a message in itself.

Ex-1006, §7.4.7.1.

151. **“store the wrapped pre-master key in the secure memory”**: Chou further discloses storing the data to be processed by the security processor in the secure memory. For example, Chou discloses a Hardware Security Module (HSM), which includes a memory used to store data processed and/or to be processed by the security processor.

[0021] HSM 20 includes a security processor 21, secure memory 22, secure key module 23, secure bus 26 and a security interface 24. The various components 21-24 and 26 reside within a secure boundary established by the hardware. That is, a secure domain or zone is established by and for the components within HSM 20, in order to set a secure boundary for processing that occurs within this boundary. Security processor 21 provides the various processing within HSM 20, including key authentication and the encrypting and/or decrypting operation to encrypt and/or decrypt data. Secure key module 23 is a storage medium, such as a memory, register, circuit, etc. that stores one or more key value(s) that is used by security processor 21 to encrypt and decrypt data. Secure memory 22 is utilized to store data that is processed and/or to be processed by security processor 21.

Ex-1008 ¶21. Figure 1, reproduced with highlights showing the security engine below, shows a simplified block diagram of the device incorporating an HSM.

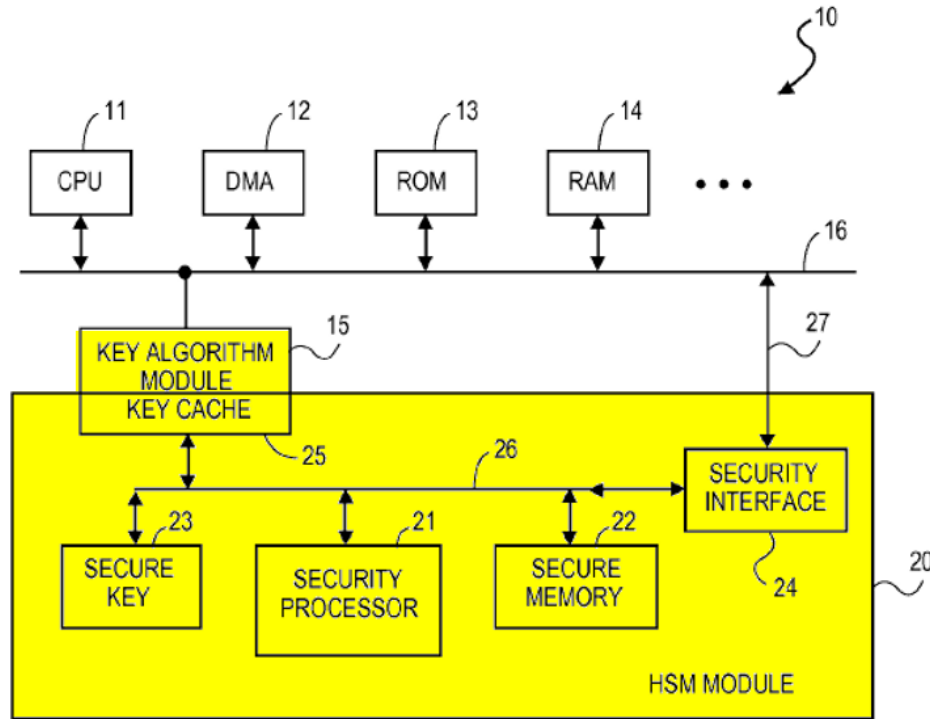


FIG. 1

Ex-1008, Fig. 1.

152. In the combination where Chou's system is used to generate a secure key according to TLS-1.0, the key exchange would be performed by the security processor **21**. In the context of more detailed components shown in Figure 3 (reproduced below), the TLS key exchange would be performed by the PKA module, which performs the relevant calculations such as the "Diffie-Hellman public value and shared key computation" and "RSA key generation." Ex-1008 ¶¶43-44, 48.

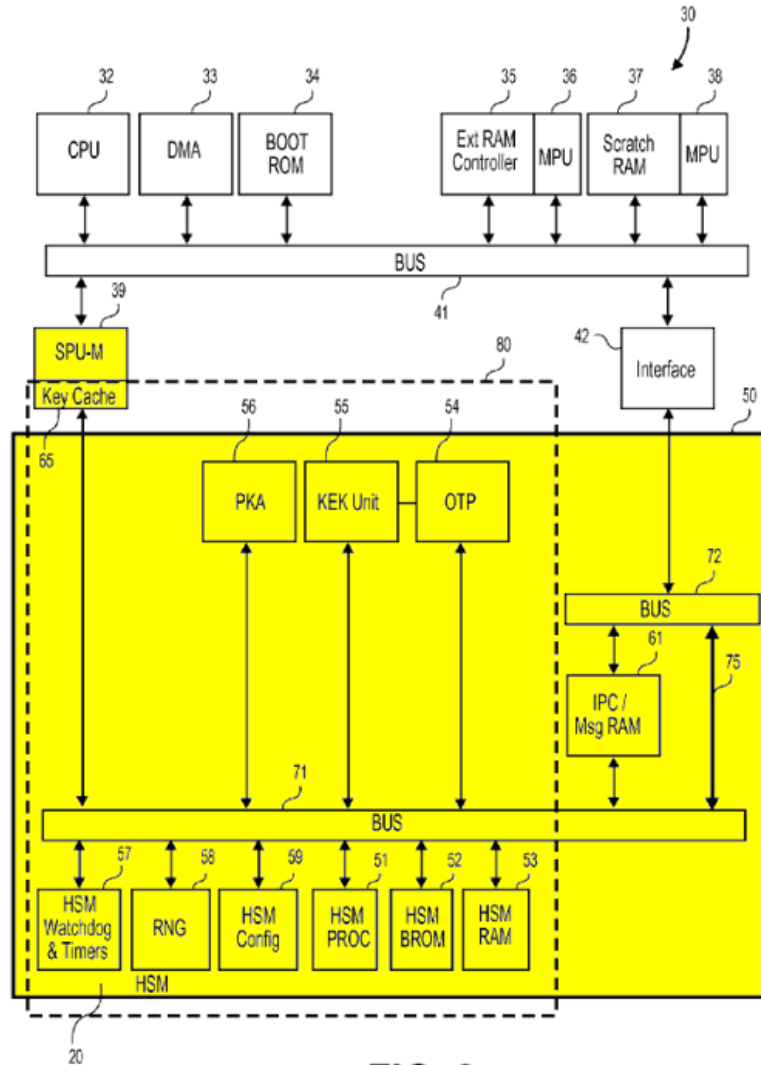


FIG. 3

Ex-1008, Fig. 3.

[0043]-[0053] PKA module 56 is a specialized processor that provides acceleration for a number of algorithms. In one embodiment, PKA module 56 is an accelerator that is operable for the following algorithms:

1. Diffie-Hellman public value and shared key computation of modulus sizes up to 4096 bits.
2. 1024-bit DSA signature with SHA-1 hash or 2048-bit DSA signature with SHA-256 hash.
3. RSA encryption of modulus sizes up to 4096-bit key

size.

4. RSA decryption of modulus sizes up to 4096-bits using Chinese Remainder Theorem.

5. RSA key generation with Rabin-Miller number selection.

6. Elliptic curve Diffie-Hellman in prime field GF(p).

7. Prime field EC-DSA signature for modulus sizes up to 512-bit.

8. Generic prime field ECC point operations.

9. Generic long integer math operations.

10. NIST prime field ECC curve optimization.

It is to be noted that these algorithms are listed for exemplary purpose and that other embodiments of PKA module 56 may support a different list of algorithms, including algorithms not listed above.

Ex-1008 ¶¶43-53.

153. Thus, Chou discloses storing encrypted pre-master key, which is one of the data processed by the security processor inside the HSM, in the secure memory in the HSM.

4. Dependent Claim 5

- a. 5[pre]/5[a]: The system-on-a-chip apparatus of claim 3, wherein the security engine is further to: generate the session key as a function of the pre-master key**

154. TLS-1.0 specifies that “master secret” (session key) is created by following method:

8.1. Computing the master secret

For all key exchange methods, the same algorithm is used to convert the `pre_master_secret` into the `master_secret`. The `pre_master_secret` should be deleted from memory once the `master_secret` has been computed.

$$\text{Master_secret} = \text{PRF}(\text{pre_master_secret}, \text{“master secret”}, \text{ClientHello.random} + \text{ServerHello.random})$$

[0..47];

The master secret is always exactly 48 bytes in length. The length of the premaster secret will vary depending on key exchange method.

Ex-1006, §8.1.

155. The statement “`master_secret = PRF(pre_master_secret, “master secret”, ClientHello.random + ServerHello.random) [0..47]`” refers to the process of generating a shared master secret key for subsequent TLS communications. The master secret key is generated by applying the Pseudo-Random Function (PRF) to the pre-master secret key (which is generated using the Diffie-Hellman key exchange algorithm or other key exchange methods), along with the “master secret” label and a concatenated value of the client and server random values (`ClientHello.random + ServerHello.random`).

156. The PRF is a cryptographic function that takes as input a shared secret key and a label, and generates a pseudo-random output that can be used as a symmetric key for encryption and decryption. The PRF function used in TLS is

defined in the TLS specification, and is designed to ensure that the output is unpredictable and provides sufficient entropy for use as a symmetric key. The [0..47] notation at the end of the statement indicates that the resulting master secret key should be truncated to a maximum length of 48 bytes (384 bits).

157. This key generation method takes the pre_master_secret as input, i.e., generate the session key as a function of the pre-master key.

b. 5[b] calculate a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server

158. Chou discloses that the PKA module (part of the HSM) performs hash calculation:

[0043]-[0053] PKA module **56** is a specialized processor that provides acceleration for a number of algorithms. In one embodiment, PKA module **56** is an accelerator that is operable for the following algorithms:

1. Diffie-Hellman public value and shared key computation of modulus sizes up to 4096 bits.
2. 1024-bit DSA signature with SHA-1 hash or 2048-bit DSA signature with SHA-256 hash.
3. RSA encryption of modulus sizes up to 4096-bit key size.
4. RSA decryption of modulus sizes up to 4096-bits using Chinese Remainder Theorem.
5. RSA key generation with Rabin-Miller number selection.
6. Elliptic curve Diffie-Hellman in prime field GF(p).

7. Prime field EC-DSA signature for modulus sizes up to 512-bit.
8. Generic prime field ECC point operations.
9. Generic long integer math operations.
10. NIST prime field ECC curve optimization.

It is to be noted that these algorithms are listed for exemplary purpose and that other embodiments of PKA module **56** may support a different list of algorithms, including algorithms not listed above.

Chou at [0043].

159. Chou, however, does not disclose that the hash function is calculated as a function of the nonces exchanged between the client-server (i.e., the random nonce generated by the security engine and a nonce received from the remote server). This limitation, however, would have been obvious in view of the proposed combination where Chou's device is used to manage session keys created according to the TLS-1.0.

160. TLS-1.0 explains that the master secret (session key) is created by the following method.

8.1. Computing the master secret

For all key exchange methods, the same algorithm is used to convert the `pre_master_secret` into the `master_secret`. The `pre_master_secret` should be deleted from memory once the `master_secret` has been computed.

`Master_secret = PRF(pre_master_secret, "master`

secret”, ClientHello.random +
ServerHello.random)
[0..47];

The master secret is always exactly 48 bytes in length. The length of the pre-master secret will vary depending on key exchange method.

Ex-1006, §8.1.

161. TLS-1.0 further explains that the PRF, i.e., the calculation that creates the master secret from the pre-master secret and the random numbers exchanged, is made up of two hash functions, MD5 and SHA-1:

The PRF is then defined as the result of mixing the two pseudorandom streams by exclusive-or'ing them together.

$$\text{PRF}(\text{secret}, \text{label}, \text{seed}) = \text{P_MD5}(\text{S1}, \text{label} + \text{seed}) \text{ XOR } \text{P_SHA-1}(\text{S2}, \text{label} + \text{seed});$$

Ex-1006, §5. TLS-1.0 further provides explanation of MD5 and SHA-1.

MD5

MD5 is a secure hashing function that converts an arbitrarily long data stream into a digest of fixed size (16 bytes). [MD5]

SHA

The Secure Hash Algorithm is defined in FIPS PUB 180-1. It produces a 20-byte output. Note that all references to SHA actually use the modified SHA-1 algorithm. [SHA]

Ex-1006, § B (Glossaries).

162. Thus, during the key exchange according to TLS-1.0, the session key is generated as a result of calculating PRF (which includes calculating MD5 and

SHA hash functions) using, among others, the “ClientHello_random” (i.e., the random nonce generated by the security engine) and the “ServerHello_random” (i.e., a nonce received from the remote server) as two of the inputs. In other words, the client calculates a hash function as a function of the random nonce generated by the client and a nonce received from the remote server.

5. Dependent Claim 6

- a. **6[pre]/6[a]. The system-on-a-chip apparatus of claim 1, wherein the security engine is further to generate a public Diffie-Hellman client key and a private Diffie-Hellman client key,**

163. As I explained above in Section VII.A.2, the “private Diffie-Hellman client key” is the number x_i , initially generated by the initiator (the client) at the beginning of the Diffie-Hellman algorithm (and kept private). The “public Diffie-Hellman client key” is the value $y_i = g^{x_i} \bmod p$, which is generated using the private Diffie-Hellman client key (x_i) and the prime number (often referred to as “ p ”) and the primitive root modulo p (often referred to as “ g ”). The value y_i is sent to the responder via an unsecured channel, and is therefore “public.”

164. Chou discloses the PKA (part of the HSM, which is in turn, part of the claimed “security engine”) performing the Diffie-Hellman shared key computation:

[0043]-[0053] PKA module 56 is a specialized processor that provides acceleration for a number of algorithms. In one embodiment, PKA module 56 is an accelerator that is

operable for the following algorithms:

1. Diffie-Hellman public value and shared key computation of modulus sizes up to 4096 bits.
2. 1024-bit DSA signature with SHA-1 hash or 2048-bit DSA signature with SHA-256 hash.
3. RSA encryption of modulus sizes up to 4096-bit key size.
4. RSA decryption of modulus sizes up to 4096-bits using Chinese Remainder Theorem.
5. RSA key generation with Rabin-Miller number selection.
6. Elliptic curve Diffie-Hellman in prime field $GF(p)$.
7. Prime field EC-DSA signature for modulus sizes up to 512-bit.
8. Generic prime field ECC point operations.
9. Generic long integer math operations.
10. NIST prime field ECC curve optimization.

It is to be noted that these algorithms are listed for exemplary purpose and that other embodiments of PKA module 56 may support a different list of algorithms, including algorithms not listed above.

Ex-1008, ¶¶ 43-53.

165. As I explained above, the Diffie-Hellman key exchange algorithm must necessarily generate a “private Diffie-Hellman client key” (x_i) and “public Diffie-Hellman client key” (y_i), in order to calculate the shared key. Thus a POSITA would have understood that Chou’s HSM (in particular, PKA) generates a

public Diffie-Hellman client key and a private Diffie-Hellman client key.

166. In addition, to the extent not already disclosed by Chou's disclosure of Diffie-Hellman public value and shared key computation, this limitation is disclosed by the combination with TLS-1.0. TLS-1.0 discloses a client exchange key message that establishes a premaster secret. In TLS key exchange message using Diffie-Hellman key exchange method, a client device generates a public Diffie-Hellman client key and a private Diffie-Hellman client key as part of the calculations to arrive at the premaster secret. Specifically, the client device generate a private Diffie-Hellman client key x_i , calculates the public Diffie-Hellman client key y_i from x_i , , and send y_i to the server. The server, on its side, generates the private Diffie-Hellman server key, calculates the public Diffie-Hellman server key, and send the latter to the client device. From the private Diffie-Hellman client key (that the client kept private) and the public Diffie-Hellman server key (that the client received from the server), the client generates the shared Diffie-Hellman key (the premaster secret). The server separately calculates the shared Diffie-Hellman key from the private Diffie-Hellman server key (that it kept private) and the public Diffie-Hellman client key (that the server received from the client), to arrive at the same result as I explained above in Section VII.A.2.

7.4.7. Client key exchange message

When this message will be sent:

This message is always sent by the client. It will immediately follow the client certificate message, if it is sent. Otherwise it will be the first message sent by the client after it receives the server hello done message.

Meaning of this message:

With this message, the premaster secret is set, either through direct transmission of the RSA-encrypted secret, or by the transmission of Diffie-Hellman parameters which will allow each side to agree upon the same premaster secret. When the key exchange method is DH_RSA or DH_DSS, client certification has been requested, and the client was able to respond with a certificate which contained a Diffie-Hellman public key whose parameters (group and generator) matched those specified by the server in its certificate, this message will not contain any data.

Ex-1006, §7.4.7.

F.1.1.3. Diffie-Hellman key exchange with authentication

When Diffie-Hellman key exchange is used, the server can either supply a certificate containing fixed Diffie-Hellman parameters or can use the server key exchange message to send a set of temporary Diffie-Hellman parameters signed with a DSS or RSA certificate. Temporary parameters are hashed with the hello.random values before signing to ensure that attackers do not replay old parameters. In either case, the client can verify the certificate or signature to ensure that the parameters belong to the server.

If the client has a certificate containing fixed Diffie-Hellman parameters, its certificate contains the information required to complete the key exchange. Note that in this case the client and server will generate the same

Diffie-Hellman result (i.e., pre_master_secret) every time they communicate. To prevent the pre_master_secret from staying in memory any longer than necessary, it should be converted into the master_secret as soon as possible. Client Diffie-Hellman parameters must be compatible with those supplied by the server for the key exchange to work.

If the client has a standard DSS or RSA certificate or is unauthenticated, it sends a set of temporary parameters to the server in the client key exchange message, then optionally uses a certificate verify message to authenticate itself.

Ex-1006, §F.1.1.3. Thus, a POSITA would have understood that generating the shared Diffie-Hellman result would require the client device generating public Diffie-Hellman client key and a private Diffie-Hellman client key.

b. 6[b] sign the public Diffie-Hellman client key using the security key of the security engine,

167. TLS-1.0 further explains the specific form of the client key exchange message, when Diffie-Hellman key exchange method is used to establish the secure key for the session. This message includes the public Diffie-Hellman client key.

168. The process of signing the public Diffie-Hellman client key and sending it to the server is described in Section 7.4.7.2 of the TLS-1.0, which covers the client key exchange message format. Specifically, the section describes the ClientDiffieHellmanPublic structure, which contains the client's public Diffie-Hellman value and is sent in the "Client Key Exchange" message. TLS-1.0 further

specifies that, if the server has requested a digital signature from the client, the client should sign the ClientDiffieHellmanPublic structure using its private key and include the resulting signature in the message. Upon receiving the “Certificate Verify” message, the server can verify the signature using the client’s public key and use it in combination with its own private key to derive the shared secret key for use in subsequent communication with the client. *See also*, Section VII.C (Digital Signature).

169. In particular, TLS-1.0 states:

7.4.7.2. Client Diffie-Hellman public value

Meaning of this message:

This structure conveys the client’s Diffie-Hellman public value (Yc) if it was not already included in the client’s certificate.

The encoding used for Yc is determined by the enumerated PublicValueEncoding. This structure is a variant of the client key exchange message, not a message in itself.

Structure of this message:

enum { implicit, explicit } PublicValueEncoding;

implicit

If the client certificate already contains a suitable Diffie-Hellman key, then Yc is implicit and does not need to be sent again. In this case, the Client Key Exchange message will be sent, but will be empty.

Explicit

Yc needs to be sent.

```
Struct {  
    select (PublicValueEncoding) {  
        case implicit: struct { };  
        case explicit: opaque dh_Yc<1..2^16-1>;  
        } dh_public;  
    } ClientDiffieHellmanPublic;  
  
    dh_Yc
```

The client's Diffie-Hellman public value (Yc).

Ex-1006, §7.4.7.2.

170. As quoted above, this variant of the ClientKeyExchange message contain “dh_public” field. This field contains the client's Diffie-Hellman public value. The “PublicValueEncoding” field specifies whether the value is sent implicitly or explicitly. If it is sent implicitly, this field is empty. If it is sent explicitly, the “dh_Yc” field contains the client's public value as an opaque byte string. The length of this field is variable, but must be between 1 and $2^{16}-1$ bytes.

171. TLS-1.0 Section 7.4.8 further describes a “Certificate Verify” message. This message is used by the client to send a digital signature over the entire handshake message up to this point (excluding any “Certificate Verify” messages). The signature is generated using the client's private key (a key specific to the client device) and is verified by the server using the client's public key.

7.4.8. Certificate verify

When this message will be sent:

This message is used to provide explicit verification

of a client certificate. This message is only sent following a client certificate that has signing capability (i.e. all certificates except those containing fixed Diffie-Hellman parameters). When sent, it will immediately follow the client key exchange message.

Structure of this message:

```
struct {  
    Signature signature;  
} CertificateVerify;
```

The Signature type is defined in 7.4.3.

CertificateVerify.signature.md5_hash

MD5(handshake_messages);

Certificate.signature.sha_hash

SHA(handshake_messages);

Here handshake_messages refers to all handshake messages sent or received starting at client hello up to but not including this message, including the type and length fields of the handshake messages. This is the concatenation of all the Handshake structures as defined in 7.4 exchanged thus far.

Ex-1006, §7.4.8.

172. The above quoted portion means this: The structure of the “Certificate Verify” message contains a single field of type “Signature”, which is defined in Section 7.4.3 of the TLS-1.0. The “Signature” structure contains two fields: md5_hash and sha_hash, which are used to store the message digest values for the MD5 and SHA-1 hash functions, respectively.

173. The values of md5_hash and sha_hash are computed by hashing all of the handshake messages exchanged between the client and server up to this point, including the “Client Key Exchange” message described above (which includes the public Diffie-Hellman client key), using the corresponding hash function.

174. Once the message digest values have been computed, they are signed using the private key corresponding to the client’s certificate. The resulting signature is then included in the “Certificate Verify” message and sent to the server. Upon receiving this message, the server can verify the signature using the public key from the client’s certificate, and can therefore be sure that the client possesses the private key corresponding to that certificate. This provides the server with assurance that it is communicating with the intended client, rather than an impostor.

c. 6[c] send the signed public Diffie-Hellman key to the remote server.

175. As explained above, the “Certificate Verify” message from the client to the server in TLS-1.0 sends the signed key exchange messages, including signed the “Client Key Exchange” message described above (which includes signed the public Diffie-Hellman client key).

* * * * *

176. Thus, based on the disclosures in Chou and TLS-1.0, Chou in combination with TLS-1.0 renders Claim 6 obvious.

6. Dependent Claim 7

- a. 7[a] The system-on-a-chip apparatus of claim 6, wherein the security engine is further to receive a public Diffie-Hellman server key from the remote server and**

177. Chou in combination with TLS-1.0 renders this limitation obvious.

Chou discloses the PKA (part of the HSM, which is in turn, part of the claimed “security engine”) performing the Diffie-Hellman shared key computation. To compute Diffie-Hellman shared key, the client device (the SoC that includes HSM) must receive public Diffie-Hellman server key (y_r), such that the client device (in particular, the PKA inside the HSM in Chou) can compute the Diffie-Hellman shared key from the public Diffie-Hellman server key and the private Diffie-Hellman client key, as explained in Section VII.A.2 above.

[0043]-[0053] PKA module 56 is a specialized processor that provides acceleration for a number of algorithms. In one embodiment, PKA module 56 is an accelerator that is operable for the following algorithms:

1. Diffie-Hellman public value and shared key computation of modulus sizes up to 4096 bits.
2. 1024-bit DSA signature with SHA-1 hash or 2048-bit DSA signature with SHA-256 hash.
3. RSA encryption of modulus sizes up to 4096-bit key size.
4. RSA decryption of modulus sizes up to 4096-bits using Chinese Remainder Theorem.
5. RSA key generation with Rabin-Miller number

selection.

6. Elliptic curve Diffie-Hellman in prime field $GF(p)$.
7. Prime field EC-DSA signature for modulus sizes up to 512-bit.
8. Generic prime field ECC point operations.
9. Generic long integer math operations.
10. NIST prime field ECC curve optimization.

It is to be noted that these algorithms are listed for exemplary purpose and that other embodiments of PKA module 56 may support a different list of algorithms, including algorithms not listed above.

Ex-1008 ¶¶43-53.

- b. 7[b] generate the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key.**

178. Chou discloses compute the Diffie-Hellman shared key from the public Diffie-Hellman server key and the private Diffie-Hellman client key, as explained above. This Diffie-Hellman shared key is the “pre-master key” as explained in Claim 3.

179. Chou in combination with TLS-1.0 further discloses this limitation. In particular, TLS-1.0 specifies that a “master secret” (session key) is created by following method:

8.1. Computing the master secret

For all key exchange methods, the same algorithm is used to convert the `pre_master_secret` into the `master_secret`.

The `pre_master_secret` should be deleted from memory once the `master_secret` has been computed.

```
Master_secret = PRF(pre_master_secret, "master  
secret", ClientHello.random +  
ServerHello.random)  
  
[0..47];
```

The master secret is always exactly 48 bytes in length. The length of the premaster secret will vary depending on key exchange method.

Ex-1006, §8.1.

180. The statement “`master_secret = PRF(pre_master_secret, “master secret”, ClientHello.random + ServerHello.random) [0..47]`” refers to the process of generating a shared master secret key for subsequent TLS communications. The master secret key is generated by applying the Pseudo-Random Function (PRF) to the pre-master secret key (which is generated using the Diffie-Hellman key exchange algorithm or other key exchange methods), along with the “master secret” label and a concatenated value of the client and server random values (`ClientHello.random` + `ServerHello.random`).

181. The PRF is a cryptographic function that takes as input a shared secret key and a label, and generates a pseudo-random output that can be used as a symmetric key for encryption and decryption. The PRF function used in TLS is defined in the TLS specification, and is designed to ensure that the output is unpredictable and provides sufficient entropy for use as a symmetric key. The

[0..47] notation at the end of the statement indicates that the resulting master secret key should be truncated to a maximum length of 48 bytes (384 bits).

182. This key generation method takes the `pre_master_secret` as input, which, in Diffie-Hellman method, generated using private Diffie-Hellman client key and the public Diffie-Hellman server key. Specifically, the client device generate a private Diffie-Hellman client key x_i , calculates the public Diffie-Hellman client key y_i from x_i , , and send y_i to the server. The server, on its side, generates the private Diffie-Hellman server key, calculates the public Diffie-Hellman server key, and send the latter to the client device. From the private Diffie-Hellman client key (that the client kept private) and the public Diffie-Hellman server key (that the client received from the server), the client generates the shared Diffie-Hellman key (the premaster secret). The server separately calculates the shared Diffie-Hellman key from the private Diffie-Hellman server key (that it kept private) and the public Diffie-Hellman client key (that the server received from the client), to arrive at the same result as I explained above in Section VII.A.2. Thus, the `master_secret` (session key) is generated using private Diffie-Hellman client key and the public Diffie-Hellman server key.

* * * * *

183. Thus, based on the disclosures in Chou and TLS-1.0, Chou in combination with TLS-1.0 renders Claim 7 obvious.

7. Dependent Claims 10, 18

a. 10[pre]. The method of claim 9, wherein initiating a request for a secure communication session comprises sending a ClientHello message, including the nonce, to the remote server; and further comprising:

18[pre]. The one or more non-transitory machine-readable storage media of claim 17, wherein to initiate a request for a secure communication session comprises to send a ClientHello message, including the nonce, to the remote server; and

184. A POSITA would have found this limitation obvious in light of TLS-1.0, which was a well-established and widely-used standard protocol before the priority date of the '838 Patent. A POSITA would have been motivated to combine the HSM device of Chou to manage the session keys created during a communication with a server through internet, as explained above in Section **X.B.1.**

185. TLS-1.0 specifies the various messages exchanged between the initiator and the responder during the handshake to establish a secure session, and their sequences. The initial message sent from the client to the server is "ClientHello." For example, TLS-1.0 discloses:

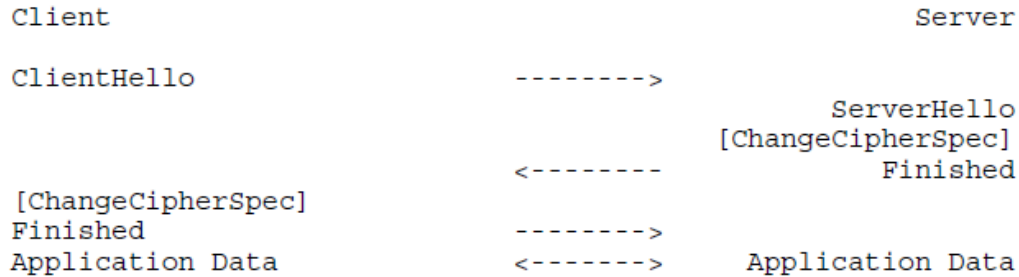


Fig. 2 - Message flow for an abbreviated handshake

Ex-1006, §7.3, Fig. 2.

These goals are achieved by the handshake protocol, which can be summarized as follows: The client sends a client hello message to which the server must respond with a server hello message, or else a fatal error will occur and the connection will fail. The client hello and server hello are used to establish security enhancement capabilities between client and server. The client hello and server hello establish the following attributes: Protocol Version, Session ID, Cipher Suite, and Compression Method. Additionally, two random values are generated and exchanged: ClientHello.random and ServerHello.random.

Ex-1006, §7.3.

186. TLS-1.0 further discloses the client hello message includes

ClientHello.random:

7.4.1.2. Client hello

When this message will be sent:

When a client first connects to a server it is required to send the client hello as its first message. The client can also send a client hello in response to a hello request or on its own initiative in order to renegotiate the security parameters in an existing connection.

Structure of this message:

The client hello message includes a random structure, which is used later in the protocol.

```
Struct {  
    uint32 gmt_unix_time;  
    opaque random_bytes[28];  
} Random;
```

gmt_unix_time

The current time and date in standard UNIX 32-bit format (seconds since the midnight starting Jan 1, 1970, GMT) according to the sender's internal clock. Clocks are not required to be set correctly by the basic TLS Protocol; higher level or application protocols may define additional requirements.

Random_bytes

28 bytes generated by a secure random number generator.

Ex-1006, §7.4.1.2.

b. 10[a] receiving a ServerHello message from the remote server, the ServerHello message including a server nonce; and

18[a] wherein the plurality of instructions further cause the computing device to: receive a ServerHello message from the remote server, the ServerHello message including a server nonce; and

187. A POSITA would have found this limitation obvious in light of TLS-1.0, which was a well-established and widely-used standard protocol before the priority date of the '838 Patent.

188. TLS-1.0 specifies the various messages exchanged between the

initiator and the responder during the handshake to establish a secure session, and their sequences. After the ClientHello message, the next TLS-1.0 message is the ServerHello message from the server to the client. For example, TLS-1.0 discloses:

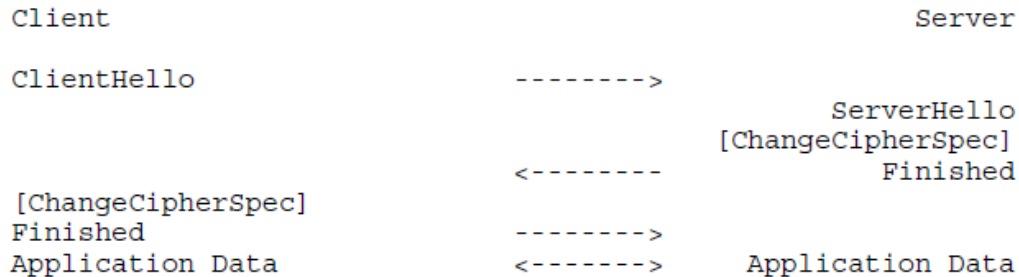


Fig. 2 - Message flow for an abbreviated handshake

Ex-1006, §7.3, Fig. 2.

These goals are achieved by the handshake protocol, which can be summarized as follows: The client sends a client hello message to which the server must respond with a server hello message, or else a fatal error will occur and the connection will fail. The client hello and server hello are used to establish security enhancement capabilities between client and server. The client hello and server hello establish the following attributes: Protocol Version, Session ID, Cipher Suite, and Compression Method. Additionally, two random values are generated and exchanged: ClientHello.random and ServerHello.random.

Ex-1006, §7.3.

189. TLS-1.0 further discloses the server hello message includes

ServerHello.random:

77.4.1.3. Server hello

When this message will be sent:

The server will send this message in response to a client hello message when it was able to find an acceptable set of algorithms. If it cannot find such a match, it will respond with a handshake failure alert.

Structure of this message:

```
struct {  
    ProtocolVersion server_version;  
    Random random;  
    SessionID session_id;  
    CipherSuite cipher_suite;  
    CompressionMethod compression_method;  
} ServerHello;
```

RFC2246 at §7.4.1.3.

random

This structure is generated by the server and must be different from (and independent of) ClientHello.random.

Ex-1006, §7.4.1.3.

c. **10[b] storing the server nonce in the secure memory of the security engine of the system-on-a-chip.**

18[b] store the server nonce in the secure memory of the security engine of the system-on-a-chip.

190. Chou discloses storing the data to be processed by the security processor in the secure memory. For example, Chou discloses a Hardware Security Module (HSM), which includes a memory used to store data that is processed and/or to be processed by the security processor.

[0021] HSM 20 includes a security processor 21, secure memory 22, secure key module 23, secure bus 26 and a

security interface **24**. The various components **21-24** and **26** reside within a secure boundary established by the hardware. That is, a secure domain or zone is established by and for the components within HSM **20**, in order to set a secure boundary for processing that occurs within this boundary. Security processor **21** provides the various processing within HSM **20**, including key authentication and the encrypting and/or decrypting operation to encrypt and/or decrypt data. Secure key module **23** is a storage medium, such as a memory, register, circuit, etc. that stores one or more key value(s) that is used by security processor **21** to encrypt and decrypt data. Secure memory **22** is utilized to store data that is processed and/or to be processed by security processor **21**.

Ex-1008 ¶21. Figure 1, reproduced with highlights showing the security engine below, shows a simplified block diagram of the device incorporating an HSM.

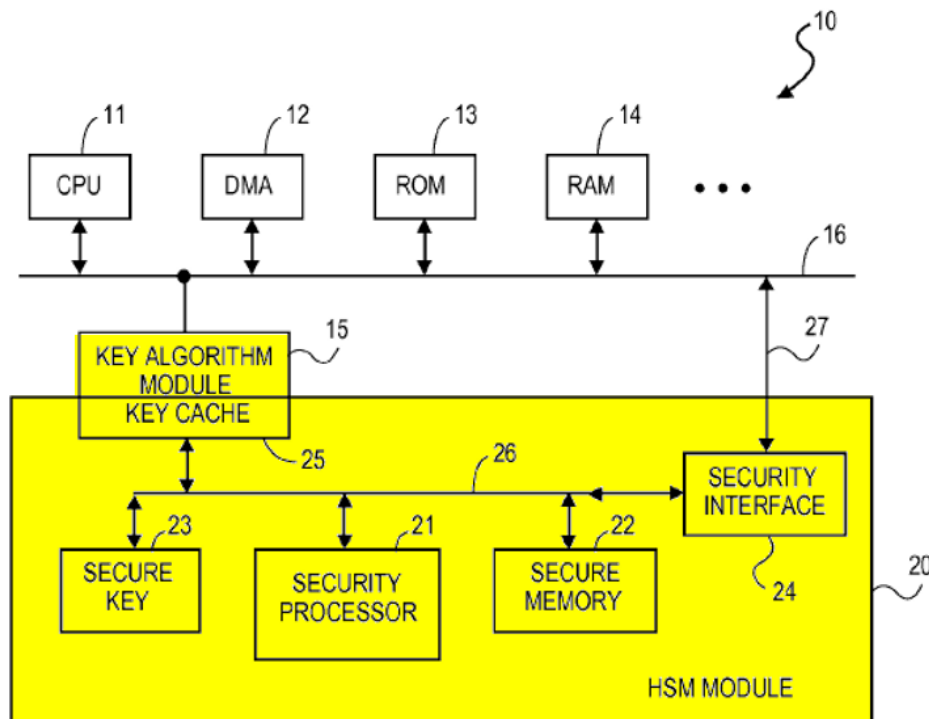


FIG. 1

Ex-1008, Fig. 1.

191. In the combination where Chou's system is used to generate a secure key according to TLS-1.0, the key exchange would be performed by the security processor **21**. In the context of more detailed components shown in Figure 3 (reproduced below), the TLS key exchange would be performed by the PKA module (highlighted purple), which performs the relevant calculations such as the "Diffie-Hellman public value and shared key computation" and "RSA key generation." Ex-1008 ¶¶43-44, 48.

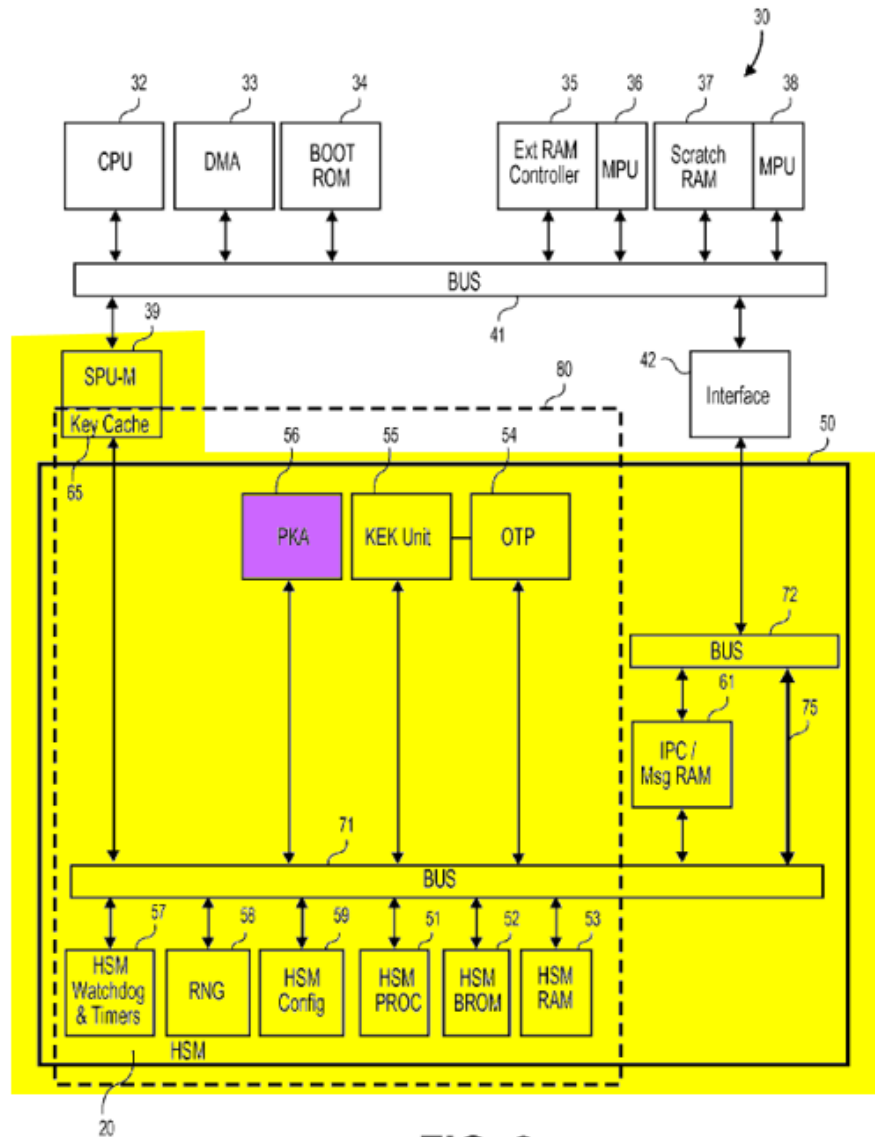


FIG. 3

Ex-1008, Fig. 3.

[0043]-[0053] PKA module **56** is a specialized processor that provides acceleration for a number of algorithms. In one embodiment, PKA module **56** is an accelerator that is operable for the following algorithms:

1. Diffie-Hellman public value and shared key computation of modulus sizes up to 4096 bits.
2. 1024-bit DSA signature with SHA-1 hash or 2048-bit DSA signature with SHA-256 hash.

3. RSA encryption of modulus sizes up to 4096-bit key size.
4. RSA decryption of modulus sizes up to 4096-bits using Chinese Remainder Theorem.
5. RSA key generation with Rabin-Miller number selection.
6. Elliptic curve Diffie-Hellman in prime field GF(p).
7. Prime field EC-DSA signature for modulus sizes up to 512-bit.
8. Generic prime field ECC point operations.
9. Generic long integer math operations.
10. NIST prime field ECC curve optimization.

It is to be noted that these algorithms are listed for exemplary purpose and that other embodiments of PKA module **56** may support a different list of algorithms, including algorithms not listed above.

Ex-1008 ¶¶43-53.

192. Thus, Chou discloses storing the server nonce, which is one of the data processed by the security processor (PKA) during the key computation, in the secure memory in the HSM.

* * * * *

193. Thus, based on the disclosures in Chou and TLS-1.0, Chou in combination with TLS-1.0 renders Claim 10 obvious.

8. Dependent Claims 11, 19

a. 11 The method of claim 9, wherein performing the cryptographic key exchange with the remote server comprises: generating a pre-master key in the security engine; encrypting the pre-master key using the security key of the security engine; and sending the encrypted pre-master key to the remote server.

19 The one or more non-transitory machine-readable storage media of claim 17, wherein to perform the cryptographic key exchange with the remote server comprises to: generate a pre-master key in the security engine encrypt the pre-master key using the security key of the security engine; and send the encrypted pre-master key to the remote server.

194. Chou in combination with TLS-1.0 renders Claims 11 and 19 obvious for the same reasons I explained in Section **X.B.2**, Claim 3 (“The system-on-a-chip apparatus of claim 1, wherein herein the security engine is further to generate a pre-master key, encrypt the pre-master key using the security key of the security engine, and send the encrypted pre-master key to the remote server.”).

9. Dependent Claims 12, 20

- a. 12[pre]/12[a]: The method of claim 11, further comprising: wrapping the pre-master key in the security engine of the system-on-a-chip using a server public key, and
12[b] storing the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip.**

**20[pre]/20[a]: The one or more non-transitory machine-readable storage media of claim 19, wherein the plurality of instructions further cause the computing device to: wrap the pre-master key in the security engine of the system-on-a-chip with a server public key, and
20[b] store the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip.**

195. Chou in combination with TLS-1.0 renders Claims 12 and 20 obvious for the same reasons I explained in Section **X.B.3**, Claim 4 (“The system-on-a-chip apparatus of claim 3 wherein the security engine is further to wrap the pre-master key using a server public key and store the wrapped pre-master key in the secure memory.”). Thus, Chou in combination with TLS-1.0 renders Claims 12 and 20 obvious.

10. Dependent Claims 13, 21

a. 13. The method of claim 11, wherein performing the cryptographic key exchange with the remote server comprises generating the session key in the security engine of the system-on-a-chip as a function of the pre-master key.

21. The one or more non-transitory machine-readable storage media of claim 19, wherein to perform the cryptographic key exchange with the remote server comprises to generate the session key in the security engine of the system-on-a-chip as a function of the pre-master key.

196. Chou in combination with TLS-1.0 renders Claims 12 and 20 obvious for the same reasons I explained in Section **X.B.4.a**, limitation 5[pre] and 5[a] (“The system-on-a-chip apparatus of claim 3, wherein the security engine is further to: generate the session key as a function of the pre-master key”). Thus, Chou in combination with TLS-1.0 renders Claims 13 and 21 obvious.

11. Dependent Claims 14, 22

a. 14. The method of claim 13, wherein generating the session key in the security engine comprises calculating, using the security engine of the system-on-a-chip, a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server.

22. The one or more non-transitory machine-readable storage media of claim 21, wherein to generate the session key in the security engine comprises to calculate, using the security engine of the system-on-a-chip, a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server.

197. Chou in combination with TLS-1.0 renders Claims 13 and 21 obvious, as discussed above. Chou further discloses the additional limitations of dependent Claims 14 and 22 for the same reasons I explained in Section **X.B.4.b**, limitation 5[b] (“calculate a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server”). Thus, Chou in combination with TLS-1.0 renders Claims 14 and 22 obvious.

12. Claims 15, 23

- a. 15[pre]/15[a] The method of claim 9 wherein performing the cryptographic key exchange with the remote server comprises: generating, in the security engine of the system-on-a-chip, a public Diffie-Hellman client key and a private Diffie-Hellman client key;**

23[pre]/23[a] The one or more non-transitory machine-readable storage media of claim 17, wherein to perform the cryptographic key exchange with the remote server comprises to: sign the public Diffie-Hellman client key using the security key of the security engine, and

198. Chou in combination with TLS-1.0 renders 15[a] and 23[a] obvious for the same reasons I explained in Section **X.B.5.a**, limitation 6[a] (“The system-on-a-chip apparatus of claim 1, wherein the security engine is further to generate a public Diffie-Hellman client key and a private Diffie-Hellman client key”).

- b. 15[b] signing the public Diffie-Hellman client key using the security key of the security engine of the system-on-a-chip;**

23[b] sign the public Diffie-Hellman client key using the security key of the security engine of the system-on-a-chip;

199. Chou in combination with TLS-1.0 renders limitations 15[b] and 23[b] obvious for the same reasons I explained in Section **X.B.5.b**, limitation 6[b] (“sign the public Diffie-Hellman client key using the security key of the security engine”).

- c. **15[c] sending the signed public Diffie-Hellman key to the remote server;**

23[c] send the signed public Diffie-Hellman key to the remote server;

200. Chou discloses limitations 15[c] and 23[c] for the same reasons I explained in Section **X.B.5.c**, limitation 6[c] (“send the signed public Diffie-Hellman key to the remote server”).

- d. **15[d] receiving a public Diffie-Hellman server key from the remote server; and**

23[d] receive a public Diffie-Hellman server key from the remote server; and

201. Chou discloses limitations 15[d] and 23[d] for the same reasons I explained in Section **X.B.6.a**, limitation 7[a] (“The system-on-a-chip apparatus of claim 6, wherein the security engine is further to receive a public Diffie-Hellman server key from the remote server”).

- e. **15[e] generating, in the security engine of the system-on-a-chip, the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key.**

23[e] generate, in the security engine of the system-on-a-chip, the session key as a function of the private DiffieHellman client key and the public Diffie-Hellman server key.

202. Chou discloses limitations 15[d] and 23[d] for the same reasons I explained in Section **X.B.6.bX.C.7.b**, limitation 7[b] (“generate the session key as

a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key.”).

* * * * *

203. Thus, based on the disclosures in Chou and TLS-1.0, Chou in combination with TLS-1.0 renders Claims 15 and 23 obvious.

C. Grounds 3 and 4: Buer, in view of knowledge of a POSITA (Ground 3) or in view of Chou (Ground 4)

204. To the extent Buer in view of knowledge of a POSITA (Ground 3) does not disclose any limitations of the challenged claims, those limitations are obvious in further combination with Chou, as further explained below. The contemplated combination would retain the same disclosed structure of Buer, but further incorporates Chou’s secure memory **22** (inside HSM **20**) as the storage area inside Buer’s security module to store data processed or to be processed by the processor of the security module.

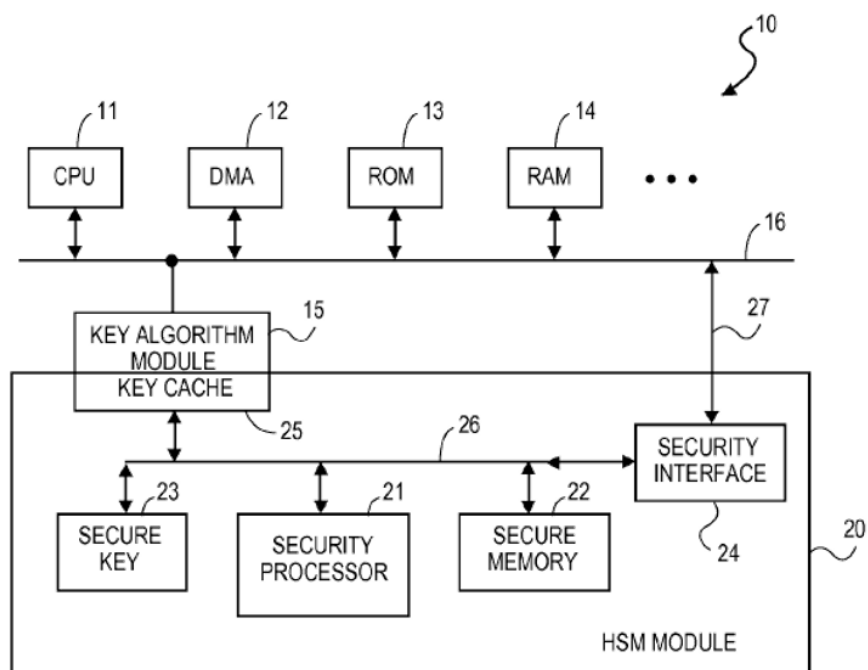


FIG. 1

Ex-1008, Fig. 1.

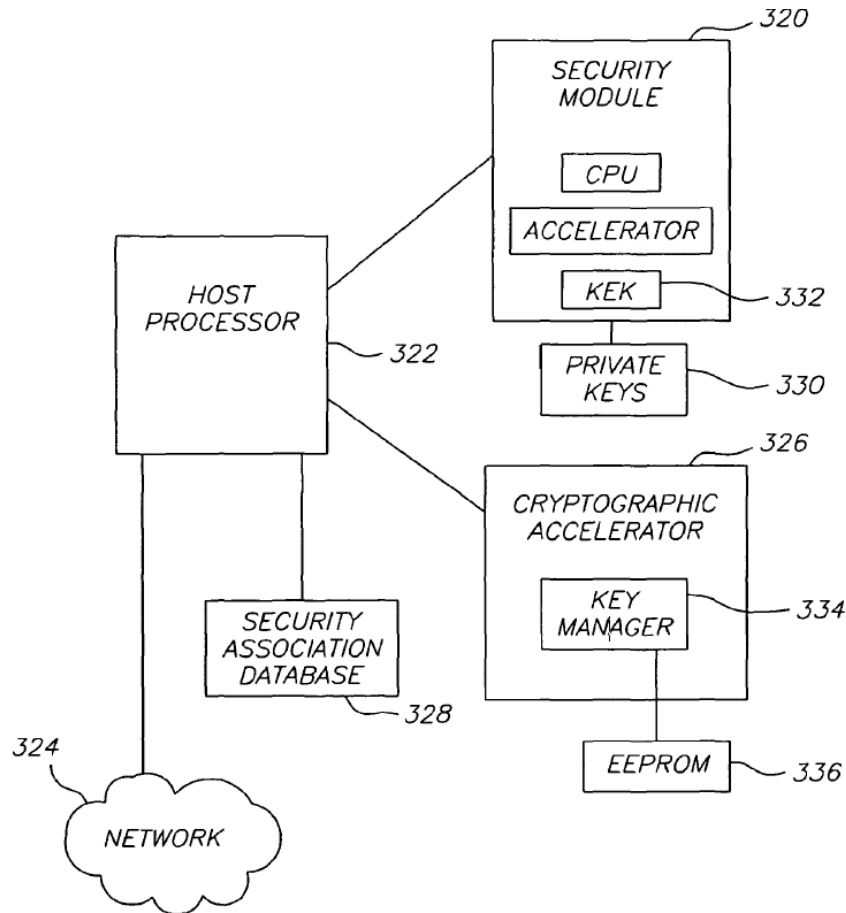


FIG. 3

Ex-1009, Fig. 3.

1. Motivation to Combine Buer with Chou

205. A POSITA would have recognized that Chou's secure storage inside HSM provide several benefits. For example, by integrating a secure storage inside a security module, the system can provide a higher level of security for sensitive data. The security module, such as Buer's security module, can be used to manage cryptographic keys and perform hardware-based encryption and decryption, while the integrated secure storage can be used to store the encrypted data. This can

provide a higher level of security than using software-based encryption or storing data in plain text. This is because the security module would have exclusive access to the data stored in the secure storage integrated into the security module, providing a higher level of security against attacks. By storing sensitive information, such as cryptographic keys or key material, inside the security module's internal memory, it is less likely that this information will be accessed or stolen by unauthorized parties.

206. In addition, a POSITA would have recognized that, by storing the data processed or to be processed by the Buer's security module inside the security module itself (rather than in a separate database outside the security module), overall system performance can be improved. For example, cryptographic operations can be performed more efficiently on a security processor with hardware acceleration for these operations without constantly going through an interface to the external storage, which may involve further access control checks and procedures. Thus, incorporating Chou's secure memory into Buer's security module would provide faster access to the data for processing with the security module.

207. A POSITA would have also realized that, by integrating the secure memory of Chou into Buer's security module, Buer's security module further simplifies the key management process. The security module can handle key

generation, storage, and deletion, as well as key rotation and backup, which can reduce the complexity of managing keys in the outside data storage area in a separate database and reduce the risk of key compromise.

208. Thus, a POSITA would be motivated to incorporate Chou's secure memory into Buer's security module for its potential benefits, including enhanced security, improved performance, and simplified key management.

209. In addition, including an internal storage into a security module was a well-known approach to securing sensitive data. A POSITA would have been familiar with the basic concepts of hardware-based encryption and decryption, as well as key management using a secure memory inside a security module.

210. Furthermore, combining an internal memory is a logical extension of Buer's system, which already has a non-volatile memory to store a Key Encryption Key (KEK) in a protected portion of the system.

In one embodiment, the KEK **130** is stored in a non-volatile data memory device **134**. In a symmetric key system the initial KEK **130** may be programmed into the non-volatile memory **134** when the system is manufactured. In an asymmetric key system, a private key and a certificate may be stored in the non-volatile memory **134** when the system is manufactured. Typically, the non-volatile memory device **134** (e.g., an EEPROM) is protected by epoxy or some other means. Hence, the non-volatile memory **134** and the key manager **132** may be in a "protected portion" of the system. Moreover, the system typically is designed so that only the key manager **132** has access to the portion of the non-volatile memory **134** that contains sensitive data such as

KEK 130. Alternatively, the non-volatile memory 134 may be integrated into the key manager's integrated circuit to control access to the non-volatile memory 134.

Ex-1009, 4:45-60; *see also*, Fig. 3.

211. Thus, expanding the non-volatile memory to further store the data processed or to be processed by the CPU of the security module would have eliminated the need to separately maintain and manage the security association database outside the security module, further simplifying the design of Buer's device and providing processing efficiencies. A POSITA would have realized that these changes would involve existing technology and would not require any significant innovation. Thus, a POSITA would have a reasonable expectation of success in combining Chou's secure memory into Buer's security module to practice a well-established data storage method.

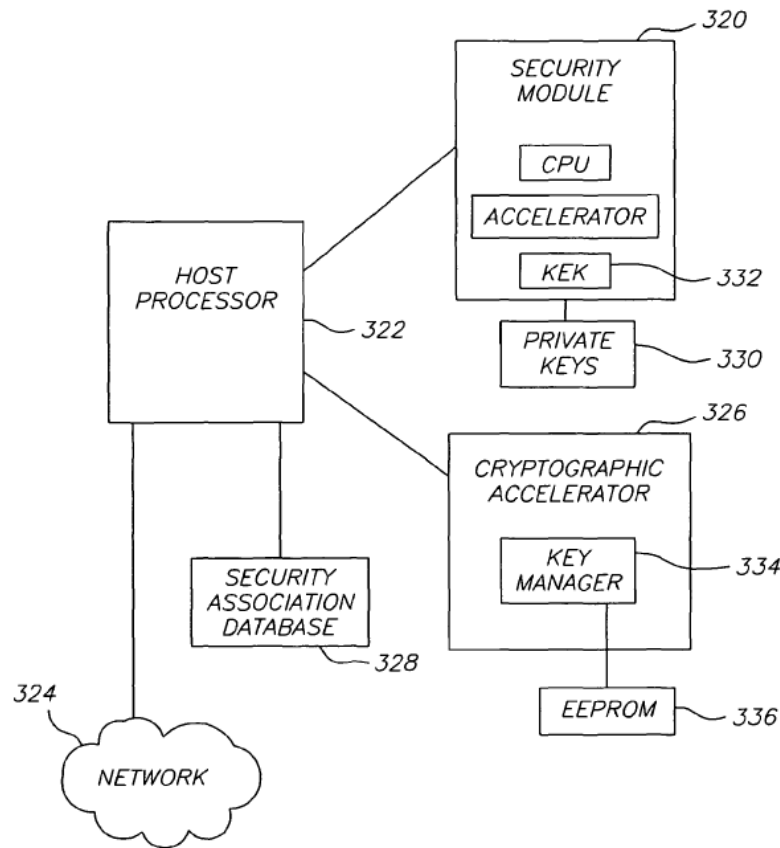
212. Thus, a POSITA would find it obvious to combine Chou's secure memory into Buer's security module because it is a logical extension of existing technology disclosed in Bure, and is well-established practice that will improve the security, efficiency, and manageability of the key management system.

2. Independent Claim 1

a. 1[pre]: A system-on-a-chip apparatus comprising:

213. To the extent the preamble is limiting, Buer discloses multiple processors (a host processor, security module, and cryptographic accelerator), and

states that these can be combined in one integrated circuit such as an SoC.



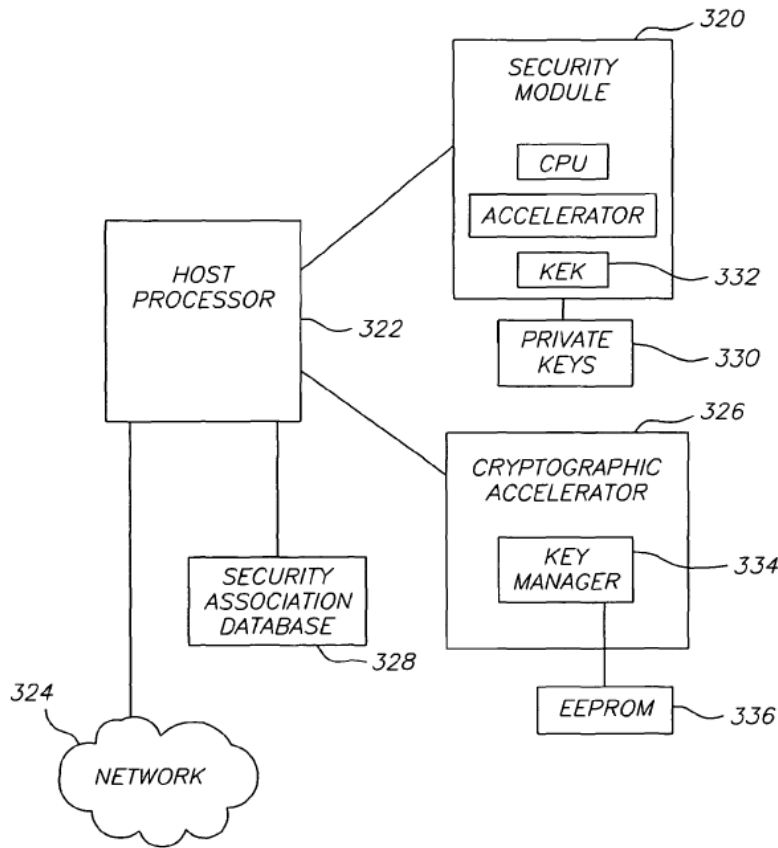
Ex-1009, Fig. 3. Buer further explains:

It should also be appreciated that the inventions described herein may be constructed using a variety of physical components and configurations. For example, a variety of hardware and software processing components may be used to implement the functions of the main security modules, satellite security modules, host processors, cryptographic accelerators, security modules, network controller and the packet processors. Typically, the network controller and packet processing functions may be implemented in a network processor. These components may be combined on one or more integrated circuits.

Ex-1009, 16:36-46.

- b. **1[a] a system-on-a-chip comprising a security engine that is separate from a processor core of the system-on-a-chip and has a secure memory accessible only by the security engine, wherein the secure memory includes a security key that was encoded in the secure memory during a manufacturing process of the system-on-a-chip, the security engine to:**

214. **“a system-on-a-chip comprising a security engine that is separate from a processor core of the system-on-a-chip”:** Buer’s device includes a “security module” and a “cryptographic processor” (together, the claimed “security engine”) that are separate from the host processor of the system-on-a-chip. For example, Buer Figure 3, reproduced below, show the Security Module **320** and Cryptographic Accelerator **326**, separate from the Host Processor **322**.



Ex-1009, Fig. 3.

The host processor **120** cooperates with a cryptographic accelerator **128** to decrypt and encrypt messages received from and sent to the network **122**, respectively. That is, when the host processor **120** receives an encrypted message from the network **122**, the host processor **120** sends this message to the cryptographic accelerator **128**.

Ex-1009, 3:58-63.

215. “a security engine . . . has a secure memory accessible only by the security engine”: The security engine (the security module and cryptographic accelerator) of Buer also includes a secure memory accessible only by the security engine. For example, Buer explains that its system includes a non-volatile

memory, such as an EEPROM, that is accessible only by the key manager, which is part of the cryptographic accelerator, which is, in turn, part of the claimed “security engine.”

In one embodiment, the KEK **130** is stored in a non-volatile data memory device **134**. In a symmetric key system the initial KEK **130** may be programmed into the non-volatile memory **134** when the system is manufactured. In an asymmetric key system, a private key and a certificate may be stored in the non-volatile memory **134** when the system is manufactured. Typically, the non-volatile memory device **134** (e.g., an EEPROM) is protected by epoxy or some other means. Hence, the non-volatile memory **134** and the key manager **132** may be in a “protected portion” of the system. Moreover, the system typically is designed so that only the key manager **132** has access to the portion of the non-volatile memory **134** that contains sensitive data such as KEK **130**. Alternatively, the non-volatile memory **134** may be integrated into the key manager’s integrated circuit to control access to the non-volatile memory **134**.

Ex-1009, 4:45-60.

216. **“wherein the secure memory includes a security key that was encoded in the secure memory during a manufacturing process of the system-on-a-chip”:** The secure memory, such as the non-volatile memory **134**, holds Key Encryption Key (KEK), which is generated at the time of manufacturing of the SoC. *See* Ex-1009, 8:1-8.

FIG. 6 depicts typical operations that may be performed by a cryptographic system according to the invention. This example describes an embodiment that uses a symmetric key.

The flowchart [begins] at block **600** with operations that, in general, relate to initialization operations. Referring, for example, to the system of FIG. 2, when the system is being manufactured the initial value for the KEK is loaded into the EEPROM and the host processor (block 602).

Ex-1009, 8:1-8.

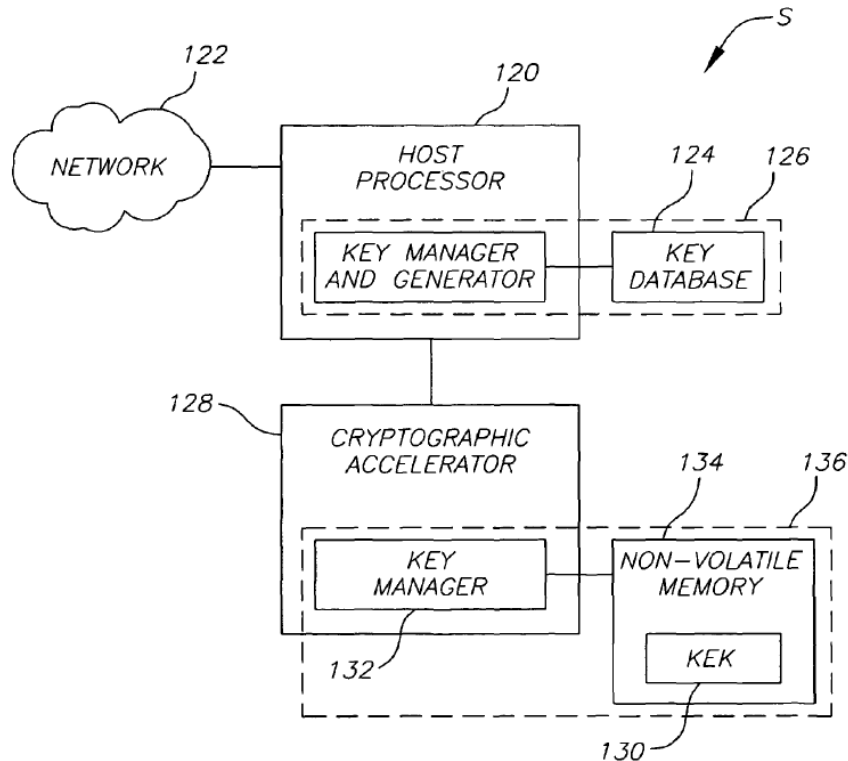


FIG. 1

Ex-1009, Fig. 1.

- c. **1[b] generate a random nonce for initiating a request for a secure communication session with a remote server over a network using the nonce;**

217. Buer discloses methods for establishing a secure channel between a main security module and a remote satellite security module (a remote server) through a network, as shown in Figure 13, reproduced below. The methods

include a step of generating a random nonce.

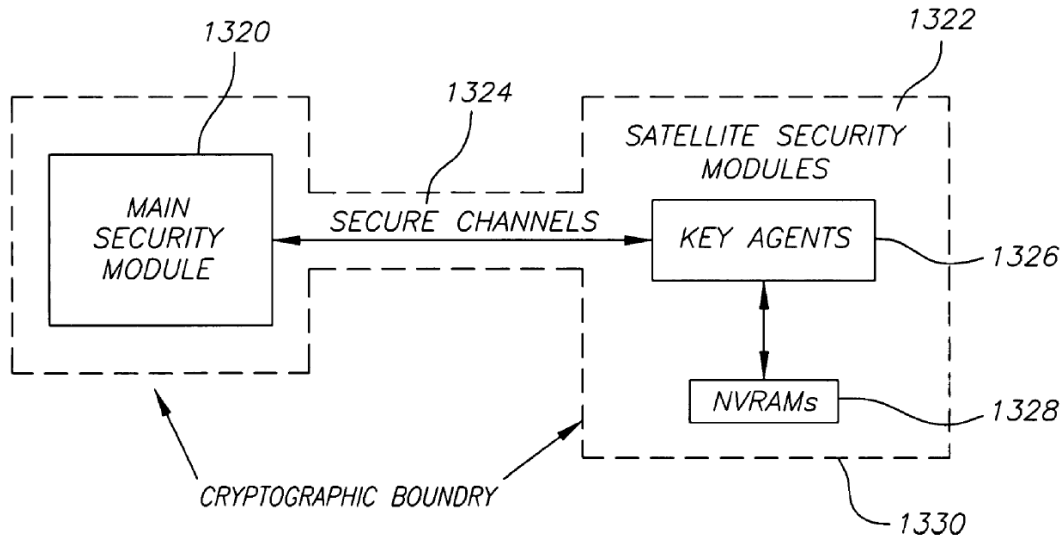


FIG. 13

Ex-1009, Fig. 13.

FIG. 13 is a high-level block diagram of one embodiment of a security system constructed according to the invention. A main security module **1320** communicates with one or more satellite security modules **1322** over one or more secure channels 1324. Each satellite security module **1322** includes processing capability (not shown) and at least one key agent **1326** and non-volatile memory device **1328**. In a typical configuration, the satellite security modules 1322 will be located throughout a data network (e.g., the Internet).

Ex-1009, 12:4-12.

218. Buer further discloses that the main security module and the satellite security module can be separate devices with their own host processors, separate from the security module. For example, Buer explains that “the secure channel

1324 may be established through the host processor (e.g., processor **1024** in Figure 10).” Ex-1009, 12:62-64. Buer further explains that the “the satellite security module **1332** . . . may be physically detached and located some distance away from the main security module **1320**.” Ex-1009, 12:17-21. “[T]he satellite security module **1322** may be implemented in blade servers or in router service blades.” Ex-1009, 12:21-23. Thus, the communication through the secure channel **1334** between the main security module and the satellite security module in Buer is a “a secure communication session with a remote server” as claimed.

219. Buer also explains that, to establish the secure channel, “the main security module **1320** initiates the secure channel exchanges.” Ex-1009, 13:1-4. Buer further clarifies that the “security module” (part of the claimed “security engine” as explained above in Section **X.C.2.b**), rather than the host processor of the client device, performs the various functions described in the secure channel exchange:

Typically, the main security module **1320** initiates the secure channel exchanges. For example, the main security module **1320** will create a message and send it to the host processor. The host processor then sends a command to the satellite security module **1322**. This command is followed by a response from the satellite security module **1322** [i.e., the remote server,] that the host processor returns to the main security module **1320**.

Ex-1009, 13:1-7.

220. Buer further discloses the various protocols that can be used to

establish the secure communication session between the main security module and the satellite security module. For example, Buer explains:

The secure channel 1324 may be established and implemented with many different protocols. In addition, the secure channel 1324 may support a variety of key management messages. Several examples follow. These examples are merely representative and do not limit the protocols that may be used in implementing the invention.

The shared secret may be established, for example, using shared secret protocols such as Unauthenticated Diffie-Hellman, One-Way RSA Authenticated Diffie-Hellman, Mutually Authenticated Diffie-Hellman and Shared Secret ReKey. These protocols have the property that they can be implemented using the request/response message format described above.

Ex-1009, 13:1-20.

221. Each of the above-listed key exchange protocols, “Unauthenticated Diffie-Hellman, One-Way RSA Authenticated Diffie-Hellman, Mutually Authenticated Diffie-Hellman and Shared Secret ReKey,” involves generating a random nonce for initiating the secure communication request. For example, Buer discloses, in the context of Unauthenticated Diffie-Hellman exchange, the initiator (i.e., the main security module) generates random number x_i , to create a public number, and sends it the responder (the satellite security module) with a nonce N_i in the initial steps of the exchange:

In Unauthenticated Diffie-Hellman the main security module and the satellite security module share public knowledge of a Diffie-Hellman instance, consisting of a

prime number p and generator g . This is the basic Diffie-Hellman exchange.

1. Initiator generates random number x_i , and creates public number $y_i = g^{x_i} \bmod p$;
2. Initiator sends y_i to the responder, along with a nonce N_i ;
3. Responder generates a random number x_r , public number $y_r = g^{x_r} \bmod p$;
4. Responder generates the Diffie-Hellman shared secret, $K = y_i^{x_r} \bmod p$, and nonce N_r ;
5. Responder generates as much key material stream as needed using $\text{HMAC}(K, N_i \parallel N_r \parallel i)$, for $i=0, 1, 2, \dots$;
6. Responder returns y_r to the initiator, along with nonce N_r ;
7. Initiator derives shared key material using the same formula.

This is a simple and standard protocol that creates FIPS quality shared key material. Here, the satellite security module only needs access to the Diffie-Hellman instance constants, and these could be included in the integrated circuit logic. However, this approach may be vulnerable to man-in-the-middle (MITM) attacks that could compromise the secure channel and all KEKs exchanged over the secure channel.

Ex-1009, 13:21-44.

222. Buer discloses similar steps of generating a random nonce to initiate the One-Way RSA Authenticated Diffie-Hellman, Mutually Authenticated Diffie-Hellman and Shared Secret ReKey key exchange steps as well. *See* Ex-1009,

13:45-14:10 (One-Way RSA Authenticated Diffie-Hellman beginning with “Initiator generat[ing] random number xi” and “send[ing] yi to the responder, along with a nonce Ni”); 14:11-42 (Mutually Authenticated Diffie-Hellman beginning with “Initiator generat[ing] random number xi” and “send[ing] yi to the responder, along with a nonce Ni”); 14:43-59 (Shared Secret ReKey beginning with “Initiator generat[ing] nonce Ni and send[ing] it to the responder.”).

d. 1[c] perform a cryptographic key exchange with the remote server;

223. As explained above in Section X.C.2.c, Buer discloses the main security module (part of the claimed “security engine”) performing various cryptographic key exchange with the satellite security module (the claimed “remote server”), such as the Unauthenticated Diffie-Hellman, One-Way RSA Authenticated Diffie-Hellman, Mutually Authenticated Diffie-Hellman and Shared Secret ReKey key exchange protocols. *See* Ex-1009, 13:21-44 (describing Unauthenticated Diffie-Hellman key exchange protocol); 13:45-14:10 (One-Way RSA Authenticated Diffie-Hellman exchange); 14:11-42 (Mutually Authenticated Diffie-Hellman); 14:43-59 (Shared Secret ReKey).

Typically, the main security module 1320 initiates the secure channel exchanges. For example, the main security module 1320 will create a message and send it to the host processor. The host processor then sends a command to the satellite security module 1322. This command is followed by a response from the satellite security module 1322 that the host processor returns to the main

security module **1320**.

The secure channel **1324** may be established and implemented with many different protocols. In addition, the secure channel **1324** may support a variety of key management messages. Several examples follow. These examples are merely representative and do not limit the protocols that may be used in implementing the invention

The shared secret may be established, for example, using shared secret protocols such as Unauthenticated Diffie-Hellman, One-Way RSA Authenticated Diffie-Hellman, Mutually Authenticated Diffie-Hellman and Shared Secret ReKey. These protocols have the property that they can be implemented using the request/response message format described above.

Ex-1009, 13:1-20.

- e. **1[d] generate a symmetric session key, based on the cryptographic key exchange, to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session;**

224. The various cryptographic key exchange protocols disclosed in Buer leads to the initiator (the main security module) generating a shared key material. The “shared” key material is shared (i.e., common) between the main security module and the satellite security module (i.e., the claimed “remote server”), and used to encrypt and decrypt messages exchanged between the main and the satellite security modules.

225. For example, Buer discloses, in the context of Unauthenticated Diffie-Hellman exchange, the initiator (i.e., the main security module) generates shared

key material based on the various information exchanged between the main security module and the satellite security module:

In Unauthenticated Diffie-Hellman the main security module and the satellite security module share public knowledge of a Diffie-Hellman instance, consisting of a prime number p and generator g . This is the basic Diffie-Hellman exchange.

1. Initiator generates random number x_i , and creates public number $y_i = g^{x_i} \bmod p$;
2. Initiator sends y_i to the responder, along with a nonce N_i ;
3. Responder generates a random number x_r , public number $y_r = g^{x_r} \bmod p$;
4. Responder generates the Diffie-Hellman shared secret, $K = y_i^{x_r} \bmod p$, and nonce N_r ;
5. Responder generates as much key material stream as needed using $\text{HMAC}(K, N_i \parallel N_r \parallel i)$, for $i=0, 1, 2, \dots$;
6. Responder returns y_r to the initiator, along with nonce N_r ;
7. Initiator derives shared key material using the same formula.

This is a simple and standard protocol that creates FIPS quality shared key material. Here, the satellite security module only needs access to the Diffie-Hellman instance constants, and these could be included in the integrated circuit logic. However, this approach may be vulnerable to man-in-the-middle (MITM) attacks that could compromise the secure channel and all KEKs exchanged over the secure channel.

Ex-1009, 13:21-44.

226. Buer discloses similar steps of generating the shared key material in One-Way RSA Authenticated Diffie-Hellman, Mutually Authenticated Diffie-Hellman and Shared Secret ReKey key exchange steps as well. *See* Ex-1009, 13:45-14:10 (One-Way RSA Authenticated Diffie-Hellman ending with the step of “Initiator deriv[ing] shared key material using the same formula, and verifies the HMAC over the exchange parameters.”); 14:11-42 (Mutually Authenticated Diffie-Hellman ending with “Initiator deriv[ing] shared key material using the same formula, and verifies the HMAC over the exchange parameters.”); 14:43-59 (Shared Secret ReKey ending with “Initiator deriv[ing] the shared key material, and verifies the HMAC.”).

227. A POSITA would have understood that the Diffie-Hellman exchange described in Buer would result in the same session key on both sides of the communication for encryption and decryption, i.e., a symmetric session key. As explained in Section VII.A.2 above, Diffie-Hellman key exchange algorithm is designed for each side to be able to securely generate a common encryption key, based on information exchanged via insecure channel. A POSITA would have understood that understand that the “shared key material” independently generated on both sides of the key exchange disclosed in Buer can then be used either directly as an symmetric encryption/decryption key (i.e., symmetric session key), or used to generate a symmetric encryption key according to the agreed-upon

algorithm. Thus, a POSITA would have understood Buer to disclose the Initiator (the main security engine) to generate a symmetric session key based on the various disclosed key exchange protocols.

228. Buer also discloses generating a symmetric session key in the context of connections between multiple devices (different host processors). For example, Buer states:

FIG. 2 also depicts how the host processor **224** communicates with other host processors **226** and **228** by establishing connections with those host processors to send messages via a packet network **222**, in this case the Internet. In some applications these connections may be referred to as sessions. Thus, one or more sessions may be established between the host processor **224** and the host processor **226**. And one or more sessions may be established between the host processor **224** and the host processor **228**.

After a connection is established, some of the subsequent messages associated with the session may need to be transmitted securely, for example, when the message contains financial information. In this case, the host processors cooperate to establish a set of keys for each session so that each host processor may decrypt the data encrypted by the other host processor. These keys are uniquely associated with a session and are referred to as session keys **230, 232, 234**.

Ex-1009, 5:7-23.

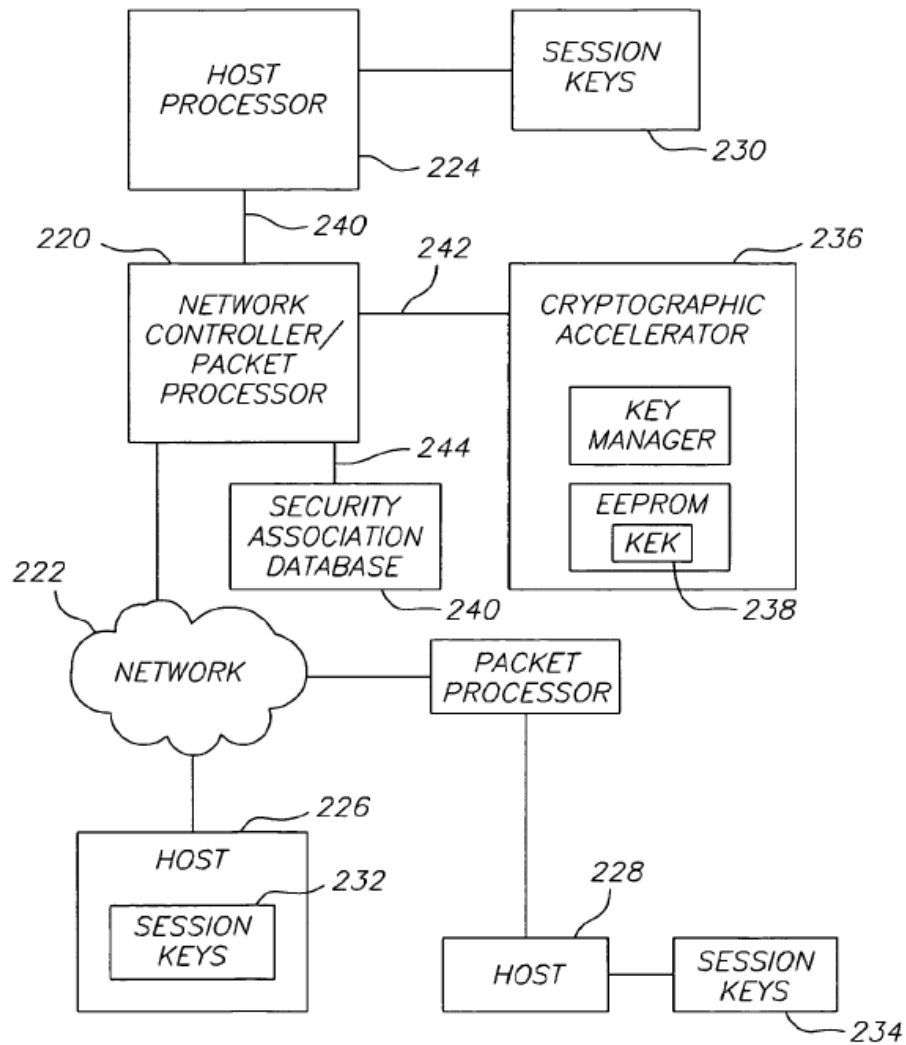


FIG. 2

Ex-1009, Fig. 2.

229. As explained above, Buer's main security module may be a part of a device that also includes a separate host processor, through which the communication with the satellite security module is made. See Ex-1009, 12:62-64 ("the secure channel **1324** may be established through the host processor (e.g., processor **1024** in Fig. 10)."); Section **X.C.2.c**. The host processor coupled to the

main security module would cooperate with the host processor coupled to the satellite security module to “establish a set of keys for each session so that each host processor may decrypt the data encrypted by the other host processor.” Ex-1009, 5:19-21. As Buer explained, these keys, such as the key established between the main security module and the satellite security module are “uniquely associated with a session and are referred to as session keys.” Buer further clarifies that these session keys are symmetric, by explaining that the keys are used such that “each host processor may decrypt the data encrypted by the other host processor.” Ex-1009, 5:20-22.

After a connection is established, some of the subsequent messages associated with the session may need to be transmitted securely, for example, when the message contains financial information. In this case, the host processors cooperate to establish a set of keys for each session so that each host processor may decrypt the data encrypted by the other host processor. These keys are uniquely associated with a session and are referred to as session keys 230, 232, 234.

Ex-1009, 5:16-23.

f. 1[e] encrypt the symmetric session key based on the security key; and

230. Buer explains that, once the security module generates the symmetric session key, the session key is encrypted based on the Key Encryption Key (KEK, the claimed “security key”).

231. For example, Buer states:

In accordance with this embodiment of the invention, these session keys are encrypted using KEK 238 and stored in a database 240 as security associations. Thus, the network controller/packet processor **220** associates each security association with a particular session or sessions. When the network controller/packet processor **220** receives an encrypted packet associated with a session, the network controller/packet processor **220** sends the packet and the security association to the cryptographic accelerator **236**.

Ex-1009, 5:43-51.

232. Buer further disclosed that the security module (part of the claimed security engine), rather than the host processor, encrypts the session key. For example, Buer states that “[t]o protect [the] private keys [such as the SSL session keys], most of the encryption and decryption operations are performed inside the security module.” Ex-1009, 1:9-12. Buer explains that, to protect the session keys, “the security module 320 encrypts them with a KEK **332** before they are sent to the cryptographic accelerator **326** (via the host processor **322**).” Ex-1009, 6:41-43. The KEK is the claimed “security key” as explained above in Section **X.C.2.b**.

233. Buer further clarifies that the above-quoted description made in the context of Figure 3 is also applicable to the communication between the main security module and the satellite security module as shown in Figure 13. For example, Buer states that “in FIG. 3 the security module [**320**] and associated data memory **330** is one embodiment of a main security module **1320**, and the cryptographic accelerator **326** with its key manager **334** and the EEPROM **336** is

one embodiment of a satellite security module **1322**.” Ex-1009, 12:47-52. Thus, Buer discloses the security module (such as the main security module shown in Figure 13) encrypting the symmetric session key, using KEK (the claimed “security key”).

g. 1[f] store the encrypted session key in the secure memory,

(1) Disclosure in Buer (Ground 3)

234. Buer discloses storing encrypted session key in a database. For example, Buer states:

In accordance with this embodiment of the invention, these session keys are encrypted using KEK **238** and stored in a database **240** as security associations. Thus, the network controller/packet processor **220** associates each security association with a particular session or sessions. When the network controller/packet processor **220** receives an encrypted packet associated with a session, the network controller/packet processor **220** sends the packet and the security association to the cryptographic accelerator **236**.

Ex-1009, 5:43-51.

235. Buer does not restrict the security association database to be stored in a secure memory, but acknowledges that “the encrypted security associations may be stored in *any* data memory.” Ex-1009, 5:66-6-1. This would include, for example, the non-volatile memory **134** connected to key manager **132** (see Fig. 1) or private keys store **330** attached to security module **320** (see Fig. 3).

236. A POSITA would have recognized that, because Buer contemplates encrypting and decrypting messages using the security module that communicates with the cryptographic accelerator, it would be most efficient to store the session keys within the cryptographic boundaries of the security module or the cryptographic accelerator, thus eliminating the need for additional communication with otherwise unsecured host processor. This can also improve system performance by reducing the number of memory accesses required to retrieve the encrypted session key. Thus, A POSITA would have been motivated to store the encrypted session keys inside the secure memory that is directly attached to the security module.

237. In addition, a POSITA would have further recognized that, storing the security associations in the already-available non-volatile memory attached to the security module would eliminate the need for a separate storage area for the security association database, and would therefore simplify the design of the system. Because the non-volatile memory is already access controlled by being coupled to the security engine, separate management or access control to the security association database is not needed.

238. To the extent that Patent Owner argues that Buer “teaches away” from storing the encrypted session keys in the secure memory, I disagree. Buer states that the “database **240**” does not need to be protected, i.e., may not be in the

“secure memory.” Ex-1009, 5:63-65 (“since the security associations are encrypted, the database device **240** does not need to be protected with epoxy or other methods. . . Theoretically, the encrypted security associations may be stored in any data memory in the network. As a result, significant cost savings may be realized in a system constructed according to the invention. ”). But Buer does not discourage storing the encrypted security associations (such as the encrypted session keys) in a secure memory location. In fact, Buer states that “the encrypted security associations may be stored in *any* data memory.” Buer also explains that the non-volatile memory can be made more secure by integrating into the same circuit as the key manager.

In one embodiment of the invention, the non-volatile memory may reside on the same integrated circuit as the key manager. This embodiment may provide more security for a KEK and/or a private key.

Ex-1009, 17:1-4.

239. Thus, Buer merely presents alternate design options to store the encrypted session key either in a more secure, but higher-cost location (such as those protected with epoxy or other methods), or in a less secure, but more cost effective data location. A POSITA would have found either of these disclosed options to be possible design options to choose from, and would not have been taught away from storing the encrypted session keys in the memory associated with the security module (the secure memory).

(2) Combination with Chou (Ground 4)

240. To the extent this limitation is not already disclosed in or rendered obvious by Buer as discussed above, Buer in combination with Chou (Ground 4) renders this limitation obvious.

241. As explained in Section **X.A.1.g** (Ground 1, limitation 1[f]) above, Chou discloses a hardware security module (HSM) similar to Buer's security module. Chou's HSM further includes a secure memory, which stores the data processed or to be processed by the processor of the HSM, the equivalent of Buer's CPU inside the security module.

242. Chou further discloses storing the data processed by the security processor, such as the session key wrapped by the HSM, in the secure memory. For example, Chou discloses a Hardware Security Module (HSM), which includes memory used to store the data processed by the security processor.

[0021] HSM 20 includes a security processor 21, secure memory 22, secure key module 23, secure bus 26 and a security interface 24. The various components 21-24 and 26 reside within a secure boundary established by the hardware. That is, a secure domain or zone is established by and for the components within HSM 20, in order to set a secure boundary for processing that occurs within this boundary. Security processor 21 provides the various processing within HSM 20, including key authentication and the encrypting and/or decrypting operation to encrypt and/or decrypt data. Secure key module 23 is a storage medium, such as a memory, register, circuit, etc. that stores one or more key value(s) that is used by security processor 21 to encrypt and

decrypt data. Secure memory 22 is utilized to store data that is processed and/or to be processed by security processor 21.

Ex-1008 ¶21. Figure 1, reproduced with highlights showing Chou's security engine below, shows a simplified block diagram of the device incorporating an HSM.

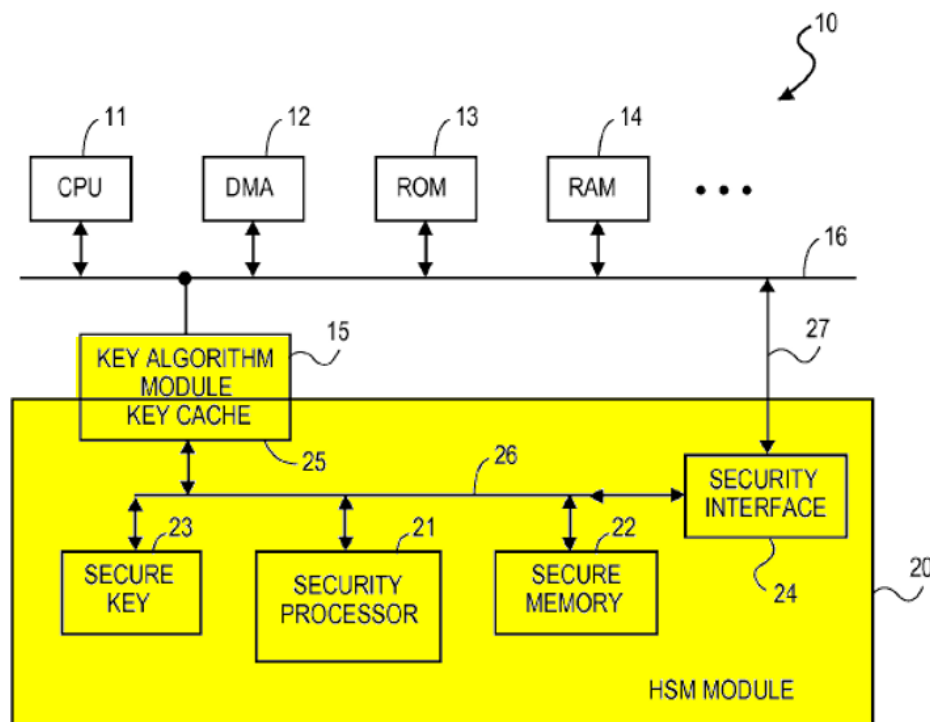


FIG. 1

Ex-1008, Fig. 1.

243. The encrypted session key is one of the data that is processed by the security processor. As explained above, the “secure key” (the session key) is wrapped/unwrapped using a “device unique key” (i.e., the security key) by the HSM, inside the secure zone (identified by the red highlighted box in Figure 3

reproduced above). Ex-1008 ¶25 (“the unwrapping of the content key is performed strictly within HSM **20**, such as by using the device unique key stored in secure key module”); ¶69 (“Secure keys are wrapped/unwrapped and authenticated within the secure zone and do not leave the HSM in an unwrapped state.”) Thus, the encrypted session key is also stored in the secure memory **22**, which is the memory location where the data processed by HSM is stored.

244. A POSITA would have understood that, in Buer’s system modified to incorporate Chou’s secure memory, Buer’s encrypted session keys, which are one of the “one or more key value(s) that is used by security processor **21** to encrypt and decrypt data,” can similarly be stored in the secure memory **22**. Buer already provides for non-volatile memory, used to store KEK and other private keys, and is accessible only to the security module. A POSITA would have found it obvious to expand the non-volatile memory to provide additional storage, such as Chou’s secure memory, for encrypted keys that are used to encrypt and decrypt data.

245. A POSITA would have recognized that Chou’s secure memory inside HSM provide several benefits and would have been motivated to modify Buer’s system to incorporate Chou’s secure memory for the reasons I explained above in Section **X.C.1**.

h. 1[g] the system-on-a-chip to establish the secure communication session with the remote server over the network using the session key.

246. Buer discloses establishing a secure session with a satellite security module over the network using a secure channel secured by the session key. Buer further explains that the satellite security module (the claimed “remote server”) is located remotely from the main security module and “located throughout a data network (e.g., the Internet.”). Ex-1009, 12:11-12.

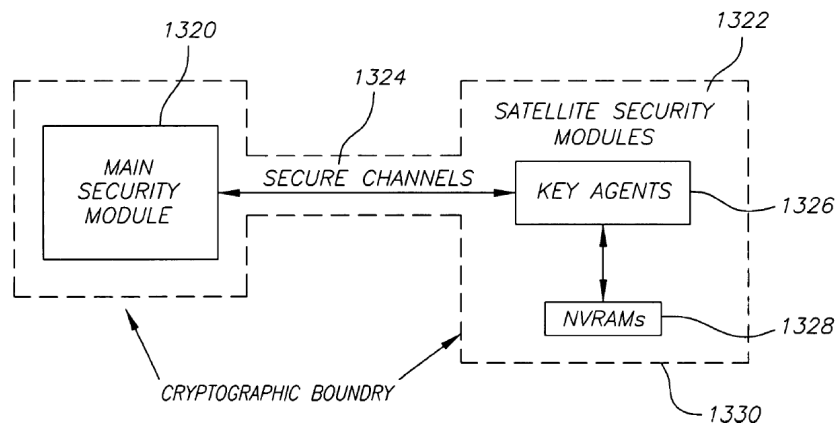


FIG. 13

Ex-1009, Fig. 13.

FIG. 13 is a high-level block diagram of one embodiment of a security system constructed according to the invention. A main security module **1320** communicates with one or more satellite security modules **1322** over one or more secure channels **1324**. Each satellite security module **1322** includes processing capability (not shown) and at least one key agent **1326** and non-volatile memory device **1328**. In a typical configuration, the satellite security modules **1322** will be located throughout a data network (e.g., the Internet).

As represented by dashed line 1330, the main security modules 1320, the secure channel 1324 and the satellite security modules 1322 cooperate to form a cryptographic boundary. For example, in one embodiment the main security module is FIPS-140-2 certified. Application of the teachings of the invention enables the satellite security module 1322 to maintain the cryptographic perimeter of the main security module 1320 even though it may be physically detached and located some distance away from the main security module 1320. For example, the satellite security module 1322 may be implemented in blade servers or in router service blades.

Ex-1009, 12:4-23.

* * * * *

247. For the reasons explained above, it is my opinion that Buer in view of a knowledge of a POSITA, or in view of Chou, discloses or renders obvious each element of the '838 Patent Claim 1.

3. Dependent Claim 3

- a. **3. The system-on-a-chip apparatus of claim 1, wherein the security engine is further to generate a pre-master key, encrypt the pre-master key using the security key of the security engine, and send the encrypted pre-master key to the remote server.**

248. The '838 Patent claims encrypting a pre-master key with the security key, but does not describe this subject matter in its specification. Instead, it describes the normal TLS operation, i.e., encrypting a pre-master key with the server's public key. *See* '838 Patent at 8:28-32; 10:60-64.

249. For purposes of this declaration, I have been asked to assume that the

disclosure provided in the '838 Patent, which describes encrypting a pre-master key using the server's public key (rather than the "security key") is sufficient to support Claim 3 as written (i.e., encrypt the pre-master key *using the security key of the security engine*).

250. Buer discloses that the security module "uses the SSL protocol to provide data security in a network." Ex-1009, 6:13-15. Buer further explains that the security module receives server's public key, and sends "session information to the server encrypted with the server's public key." Ex-1009, 6:26-29.

FIG. 3 depicts an embodiment of the invention where a security module 320 uses the SSL protocol to provide data security in a network. A host processor 322 (e.g., a server) interfaces with a network 324 to communicate with client applications running on other processors (not shown) connected to the network 324. In a similar manner as discussed above, the host processor 322 cooperates with a cryptographic accelerator 326 and a database 328 containing secured security associations to decrypt encrypted packets received from the network 324 and to encrypt packets sent to the network 324.

As discussed above, SSL uses a public/private key scheme. Thus, a secure connection may be established using standard public key techniques. For example, a server sends its signed public key to a client, the client verifies the public key then sends session information to the server encrypted with the server's public key. The keys to be used for the session are then derived from this information and the server's private key.

Ex-1009, 6:13-31.

251. Buer further discloses what type of session information are encrypted

with server's public key. For example, Buer explains that "[f]rom the session information . . . a shared secret is created from which the cryptographic accelerator **1026** creates a set of keys for the session (block 1208)." Ex-1009, 11:21-23. The "cryptographic accelerator **1026**" is the equivalent of the satellite security module (the remote server) of Figure 13. *See* Ex-1009, 12:45-52 ("the security module **320** and associated data memory **330** is one embodiment of a main security module **1320**, and the cryptographic accelerator **326** with its key manager **334** and the EEPROM **336** is one embodiment of a satellite security module **1322**").

In a similar manner as discussed above, the cryptographic accelerator **1026** decrypts the encrypted private key using the KEK_B **1032** (block **1206**). Then, the cryptographic accelerator **1026** decrypts the encrypted session information using the server's private key. From the session information and the decrypted private key a shared secret is created from which the cryptographic accelerator **1026** creates a set of keys for the session (block **1208**). The cryptographic accelerator **1026** then encrypts these keys and sends them to the client. Next, the cryptographic accelerator **1026** encrypts the security association (e.g., the session keys and, in some embodiments, associated information) using KEK_A **1040** (block **1210**). Then, the cryptographic accelerator **1026** sends the security association data to the security association database **1038** (block **1212**).

Ex-1009, 11:17-31. Thus, a POSITA would have understood that the "session information" encrypted with server's public key and sent to the server according to the SSL protocol includes pre-master secret from which the session keys are created.

4. Dependent Claim 4

- a. 4. The system-on-a-chip apparatus of claim 3 wherein the security engine is further to wrap the pre-master key using a server public key and store the wrapped pre-master key in the secure memory.**

(1) Disclosure in Buer (Ground 3)

252. **“security engine is further to wrap the pre-master key using a server public key”**: As explained in Section X.C.3.a above, Buer discloses the main security module performing cryptographic key exchange with a satellite security module (the remote server) SSL protocol, during which the main security module sends “session information” encrypted with server’s public key to the server. Also as explained above, a POSITA would have understood that the “session information” encrypted with server’s public key and sent to the server according to the SSL protocol includes pre-master secret from which the session keys are created.

253. **“store the wrapped pre-master key in the secure memory”**: Buer also discloses that the wrapped pre-master key is stored in the secure memory. For example, Buer discloses that a shared secret (such as the Diffie-Hellman shared secret generated during the cryptographic key exchange steps) can be stored in a non-volatile memory associated with the security module.

In addition, the non-volatile memory may comprise a one-time-programmable circuit for storing, for example, an initial value for KEK, a private key or a shared secret. Examples of one-time-programmable circuits may be found in the following U.S. patent applications which are assigned to the same Assignee as this application: U.S. patent application Ser. No. 10/141,197, filed May 8, 2002 and entitled USING AN ON-CHIP ONE-TIME PROGRAMMABLE NON-VOLATILE MEMORY (OTP NVM) FOR CONFIGURING DEVICE FEATURES; U.S. patent application Ser. No. 10/141,599, filed May 8, 2002 and entitled SYSTEM AND METHOD FOR PROGRAMMING NON-VOLATILE MEMORY. The contents of these applications are hereby incorporated by reference herein.

Ex-1009, 17:5-18.

254. This “shared secret” stored in the non-volatile memory associated with the security module are established through the cryptographic key exchange process:

The shared secret may be established, for example, using shared secret protocols such as Unauthenticated Diffie-Hellman, One-Way RSA Authenticated Diffie-Hellman, Mutually Authenticated Diffie-Hellman and Shared Secret ReKey. These protocols have the property that they can be implemented using the request/response message format described above.

Ex-1009, 13:14-20.

255. In addition, Buer discloses storing encrypted keys in a database as security associations, as explained in Section **X.C.2.g.(1)** (limitation 1[f], “store the encrypted session key in the secure memory”). For example, Buer states:

In accordance with this embodiment of the invention, these session keys are encrypted using KEK 238 and stored in a database 240 as security associations. Thus, the network controller/packet processor 220 associates each security association with a particular session or sessions. When the network controller/packet processor 220 receives an encrypted packet associated with a session, the network controller/packet processor 220 sends the packet and the security association to the cryptographic accelerator 236.

Ex-1009, 5:43-51. A POSITA would have understood that, an encrypted pre-master key is also a “security association” to be stored in the security association database. *See* Ex-1009, 4:21-26 (“For convenience the term ‘security association’ will be used herein to refer to key information. This key information may include, for example, a key or keys, one or more encrypted keys and associated identifiers and other information such as rules relating to how to use the keys and the types of algorithms that may be used to decrypt the keys.”).

256. Buer does not restrict the security association database to be stored in a secure memory, but acknowledges that “the encrypted security associations may be stored in any data memory.” This would include, for example, the non-volatile memory 134 connected to key manager 132 (see Fig. 1) or private keys store 330 attached to security module 320 (see Fig. 3). A POSITA would have recognized that, because Buer contemplates encrypting and decrypting messages using the security module that communicates with the cryptographic accelerator, it would be most efficient to store the session keys within the cryptographic boundaries of the

security module or the cryptographic accelerator, thus eliminating the need for additional communication with otherwise unsecured host processor. Thus, A POSITA would have been motivated to store the encrypted pre-master keys inside the secure memory that is directly attached to the security module.

257. To the extent that Patent Owner argues that Buer “teaches away” from storing the encrypted pre-master keys in the secure memory, I disagree. Buer states that the “database 240” does not need to be protected, i.e., may not be in the “secure memory.” Ex-1009, 5:63-65 (“since the security associations are encrypted, the database device **240** does not need to be protected with epoxy or other methods. . . Theoretically, the encrypted security associations may be stored in any data memory in the network. As a result, significant cost savings may be realized in a system constructed according to the invention.”). But Buer does not discourage storing the encrypted security associations (such as the encrypted session keys) in a secure memory location. In fact, Buer states that “the encrypted security associations may be stored in *any* data memory.” Buer also explains that the a non-volatile memory can be made more secure by integrating into the same circuit as the key manager.

In one embodiment of the invention, the non-volatile memory may reside on the same integrated circuit as the key manager. This embodiment may provide more security for a KEK and/or a private key.

Ex-1009, 17:1-4.

258. Thus, Buer merely presents alternate design options to store the encrypted session key either in a more secure, but higher-cost location (such as those protected with epoxy or other methods), or in a less secure, but more cost effective data location. A POSITA would have found either of these disclosed options to be possible design options to choose from, and would not have been taught away from storing the encrypted session keys in the memory associated with the security module (the secure memory).

259. Thus, a POSITA would have understood that Buer discloses encrypting (i.e., wrapping) the pre-master key using the server public key. Buer further discloses storing the pre-master key (which is wrapped) in the secure memory.

(2) Combination with Chou (Ground 4)

260. To the extent Buer does not disclose “stor[ing] the wrapped pre-master key in the secure memory,” combination with Chou renders this limitation obvious. Chou discloses a Hardware Security Module (HSM), which includes a memory used to store data processed and/or to be processed by the security processor.

[0021] HSM 20 includes a security processor 21, secure memory 22, secure key module 23, secure bus 26 and a security interface 24. The various components 21-24 and 26 reside within a secure boundary established by the hardware. That is, a secure domain or zone is established by and for the components within HSM 20, in

order to set a secure boundary for processing that occurs within this boundary. Security processor 21 provides the various processing within HSM 20, including key authentication and the encrypting and/or decrypting operation to encrypt and/or decrypt data. Secure key module 23 is a storage medium, such as a memory, register, circuit, etc. that stores one or more key value(s) that is used by security processor 21 to encrypt and decrypt data. Secure memory 22 is utilized to store data that is processed and/or to be processed by security processor 21.

Ex-1008, ¶ 21. Figure 1, reproduced with highlights showing the security engine below, shows a simplified block diagram of the device incorporating an HSM.

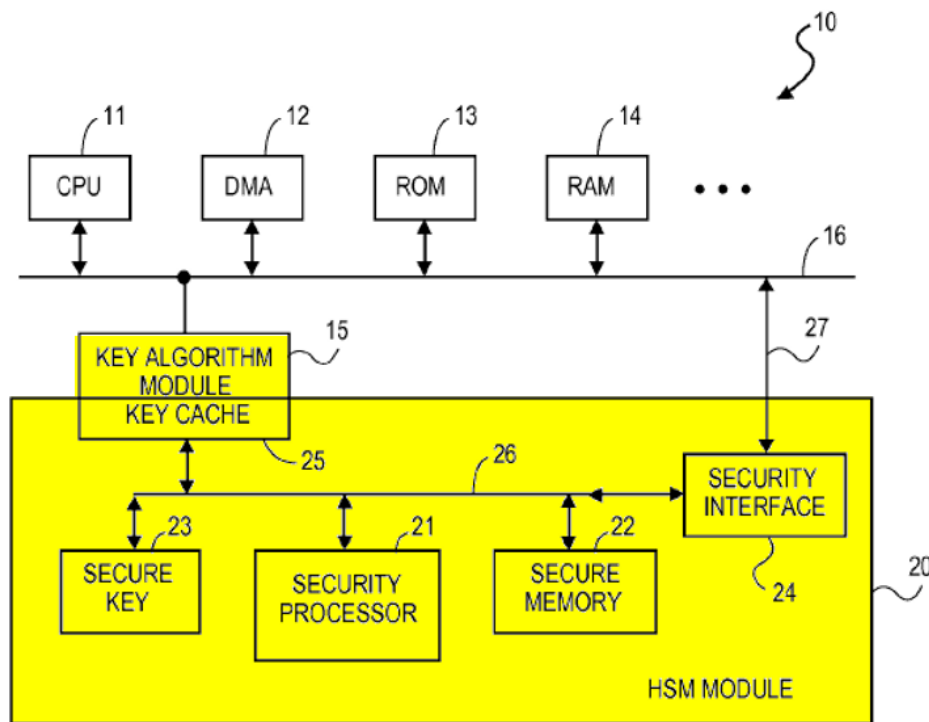


FIG. 1

Ex-1008, Fig. 1.

261. The wrapped pre-master key is one of the data processed by the

security module of Buer. Thus, in the proposed combination (Buer's system modified to include the secure memory of Chou that stores data processed or to be processed by the processor of the security module), the wrapped pre-master key would also be stored in the secure memory inside the security module.

262. A POSITA would have recognized that Chou's secure memory inside HSM provide several benefits and would have been motivated to modify Buer's system to incorporate Chou's secure memory for the reasons I explained above in Section X.C.1.

5. Dependent Claim 5

- a. 5[pre]/5[a]: The system-on-a-chip apparatus of claim 3, wherein the security engine is further to: generate the session key as a function of the pre-master key**

263. Buer discloses a cryptographic key exchange using the Diffie-Hellman algorithm. The "Diffie-Hellman shared secret" (K) created by both the Responder (in step 3 in the example below) and the Initiator (in step 7) is the pre-master key. The session key is generated as a function of the pre-master key according to the agreed-upon key generation method.

In One-Way RSA Authenticated Diffie-Hellman the main security module and satellite security module share public knowledge of a Diffie-Hellman instance, as above. In addition, the main security module has an RSA key, and the SM has a trusted copy of the main security module public key.

1. Initiator generates random number x_i , and creates

public number $y_i = g^{x_i} \bmod p$;

2. Initiator sends y_i to the responder, along with a nonce N_i and its signature over the combination.;
3. Responder verifies the signature and generates a random number x_r , public number $y_r = g^{x_r} \bmod p$;
4. Responder generates the Diffie-Hellman shared secret, $K = y_i^{x_r} \bmod p$, and nonce N_r ;
5. Responder generates as much key material stream as needed using $\text{HMAC}(K, N_i \parallel N_r \parallel i)$, for $i=0, 1, 2, \dots$;
6. Responder returns y_r to the initiator, along with nonce N_r . Responder uses the KEK from the key material stream to compute $\text{HMAC}(\text{KEK}, x_i \parallel x_r \parallel N_i \parallel N_r)$;
7. Initiator derives shared key material using the same formula, and verifies the HMAC over the exchange parameters.

Ex-1009, 13:45-65.

264. A POSITA would have also understood that the Diffie-Hellman shared secret is typically used as a pre-master secret from which a session key is derived according to a pre-agreed key derivation function (as the case would be in SSL/TLS protocol).

- b. 5[b] calculate a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server**

265. Buer's cryptographic key exchange algorithms includes steps of calculating HMAC (Hash-based Message Authentication Code) values to generate

key material stream message, as a function of the nonce generated by the main security module (calculated at the beginning of the exchange) and a nonce received from the satellite security module. HMAC is a cryptographic hash function that is used to verify the authenticity and integrity of a message. The HMAC algorithm takes a secret key and a message as inputs and produces a fixed-size output called a hash. The hash is used to verify that the message has not been tampered with or altered in transit.

266. For example, Buer's key exchange steps includes Mutually Authenticated Diffie-Hellman exchange where "Responder" (i.e., the satellite security module) computes the "HMAC (KEK, xi || xr || Ni || Nr)." The expression "HMAC (KEK, xi || xr || Ni || Nr)" means that the HMAC algorithm is being used with a secret key (KEK) and a message that is composed of four concatenated values: xi, xr, Ni, and Nr. The values xi and xr are random numbers generated by the Initiator (main security module) and the Responder (satellite security module), respectively, and are used to prevent replay attacks. The values Ni and Nr are nonces exchanged between the Initiator and the Responder in the earlier step of the exchange. Once the Initiator receives the HMAC value from the responder, the Initiator calculates the HMAC value using the parameters that it derived from the key exchange steps, and compare the value to the HMAC value received from the responder. If the two values match, the message is considered authentic and has

not been tampered with during transmission.

267. For example, Buer discloses:

In Mutually Authenticated Diffie-Hellman the main security module and satellite security module share public knowledge of a Diffie-Hellman instance, as above. For mutual authentication, both the main security module and the satellite security module can have an RSA public/private key pair. However, the same result can be achieved using a shared secret S. In effect, S is used as a “password” to authenticate the exchange.

1. Initiator generates random number x_i , and creates public number $y_i = g^{x_i} \bmod p$;
2. Initiator sends y_i to the responder, along with a nonce N_i ;
3. Responder generates a random number x_r , public number $y_r = g^{x_r} \bmod p$;
4. Responder generates the Diffie-Hellman shared secret, $K = y_i^{x_r} \bmod p$, and nonce N_r ;
5. Responder generates as much key material stream as needed using HMAC ($K, N_i \parallel N_r \parallel S \parallel i$), as above;
6. Responder returns y_r to the initiator, along with nonce N_r and an HMAC over the exchange parameters. Responder uses the KEK from the key material stream to compute this value as HMAC ($KEK, x_i \parallel x_r \parallel N_i \parallel N_r$);
7. Initiator derives shared key material using the same formula, and verifies the HMAC over the exchange parameters.

This is a much simpler protocol and is secure against MITM and spoofing, but requires a pre-existing shared secret. Although this could be installed at manufacturing

time, in practice an unchangeable secret may be difficult to manage over time. The shared secret is used for authentication, not to provide entropy for key establishment. Nonetheless, it must be protected from disclosure as would a PIN, and must be contained within the cryptographic boundary.

Ex-1009, 14:11-42.

268. The concatenation of these values ($x_i \parallel x_r \parallel N_i \parallel N_r$) forms the message that is passed to the HMAC algorithm along with the secret key (KEK). The output of the HMAC algorithm is a fixed-size hash that is used to verify the authenticity and integrity of the message. The use of HMAC with a secret key provides a strong level of security, as an attacker would need to know the secret key in order to generate a valid hash.

269. A POSITA would have understood that HMAC is an acronym for Hash-Based Message Authentication Codes, and understood that the initiator (the main security engine) would “verify the HMAC over the exchange parameters” by independently calculating the Hash using the same values (including the random nonce (x_i and N_i) generated from by the Initiator and random nonce received from the Responder (N_r)), and comparing the value with the HMAC value received from the Responder.

6. Dependent Claim 6

- a. 6[pre]/6[a]. The system-on-a-chip apparatus of claim 1, wherein the security engine is further to generate a public Diffie-Hellman client key and a private Diffie-Hellman client key,**

270. As I explained above in Section VII.A.2, the “private Diffie-Hellman client key” is the random number x_i , initially generated by the initiator (the client) at the beginning of the Diffie-Hellman algorithm (and kept private). The “public Diffie-Hellman client key” is the value $y_i = g^{x_i} \bmod p$, which is generated using the private Diffie-Hellman client key (x_i) and the prime number (often referred to as “ p ”) and the primitive root modulo p (often referred to as “ g ”). The value y_i is sent to the responder via an unsecured channel, and therefore “public.”

271. Buer discloses generating a public Diffie-Hellman client key (x_i) and a private Diffie-Hellman client key (y_i) during the Diffie-Hellman key exchange between the main security module and the satellite security module. For example, Buer discloses:

In One-Way RSA Authenticated Diffie-Hellman the main security module and satellite security module share public knowledge of a Diffie-Hellman instance, as above. In addition, the main security module has an RSA key, and the SM has a trusted copy of the main security module public key.

1. Initiator generates random number x_i , and creates public number $y_i = g^{x_i} \bmod p$;
2. Initiator sends y_i to the responder, along with a nonce N_i and its signature over the combination.;

3. Responder verifies the signature and generates a random number x_r , public number $y_r = g^{x_r} \bmod p$;
4. Responder generates the Diffie-Hellman shared secret, $K = y_i^{x_r} \bmod p$, and nonce N_r ;
5. Responder generates as much key material stream as needed using $\text{HMAC}(K, N_i \parallel N_r \parallel i)$, for $i=0, 1, 2, \dots$;
6. Responder returns y_r to the initiator, along with nonce N_r . Responder uses the KEK from the key material stream to compute $\text{HMAC}(\text{KEK}, x_i \parallel x_r \parallel N_i \parallel N_r)$;
7. Initiator derives shared key material using the same formula, and verifies the HMAC over the exchange parameters.

This also only requires public constants at the satellite security module. However, the main security module public key must be specific to the main security module, and could either be provided using a write once memory or using two levels of RSA keys and a root key loaded at manufacturing time. The main security module would supply a digital certificate signed by the root key to establish the trusted public key at the satellite security module.

This is secure against MITM attacks, but not against spoofing by a satellite security module imposter. Moreover, the main security module could inadvertently disclose the KEKs for decrypting security associations to an attacker.

Ex-1009, 13:45-14:10; *see also*, 13:25-26; 14:19-20.

b. 6[b] sign the public Diffie-Hellman client key using the security key of the security engine,

272. In step 2 of the above-quoted One-Way RSA Authenticated Diffie-

Hellman described in Buer, the “Initiator sends y_i [i.e., the public Diffie-Hellman client key] to the responder, along with a nonce N_i and its signature over the combination.” Ex-1009, 13:52-53 (emphasis added). This means the Initiator (the main security module) creates a digital signature by applying a cryptographic hash function to the combination of the public Diffie-Hellman client key (y_i) and the nonce (N_i), using the Initiator’s public key (see Section VII.C, explanation of digital signature). As quoted above, Buer further explains that the digital signature is “signed by the root key to establish the trusted public key at the satellite security module” and the “root key” is “loaded at manufacturing time.” Ex-1009, 13:66-14:6.

c. 6[c] send the signed public Diffie-Hellman key to the remote server.

273. As explained above, in step 2 of the above-quoted One-Way RSA Authenticated Diffie-Hellman described in Buer, the “Initiator sends y_i [i.e., the public Diffie-Hellman client key] to the responder, along with a nonce N_i and its signature over the combination.” Ex-1009, 13:52-53 (emphasis added).

7. Dependent Claim 7

a. 7[a] The system-on-a-chip apparatus of claim 6, wherein the security engine is further to receive a public Diffie-Hellman server key from the remote server and

274. As I explained above in Section VII.A.2, the server (the Responder in

Diffie-Hellman exchange) also generates a random number x_r , which is kept private, and calculates $y_r = g^{x_r} \bmod p$. This value, $y_r = g^{x_r} \bmod p$, which is then shared with the client (the Initiator), is referred to as the “public Diffie-Hellman server key.”

275. Buer discloses Responder (the satellite security module) generating a public Diffie-Hellman server key during the Diffie-Hellman key exchange between the main security module and the satellite security module, and sending it to the Initiator (the main security module, i.e., the claimed “remote server”). For example, Buer discloses:

In One-Way RSA Authenticated Diffie-Hellman the main security module and satellite security module share public knowledge of a Diffie-Hellman instance, as above. In addition, the main security module has an RSA key, and the SM has a trusted copy of the main security module public key.

1. Initiator generates random number x_i , and creates public number $y_i = g^{x_i} \bmod p$;
2. Initiator sends y_i to the responder, along with a nonce N_i and its signature over the combination.;
3. Responder verifies the signature and generates a random number x_r , public number $y_r = g^{x_r} \bmod p$;
4. Responder generates the Diffie-Hellman shared secret, $K = y_i^{x_r} \bmod p$, and nonce N_r ;
5. Responder generates as much key material stream as needed using $\text{HMAC}(K, N_i \parallel N_r \parallel i)$, for $i=0, 1, 2, \dots$;

6. Responder returns y_r to the initiator, along with nonce N_r . Responder uses the KEK from the key material stream to compute $\text{HMAC}(\text{KEK}, x_i \parallel x_r \parallel N_i \parallel N_r)$;
7. Initiator derives shared key material using the same formula, and verifies the HMAC over the exchange parameters.

Ex-1009, 13:45-65.

- b. 7[b] generate the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key.**

276. Buer further explains that the Responder generates “the Diffie-Hellman shared secret, $K=y_i^{x_r} \bmod p$ ” (step 3 above), and that “Initiator derives shared key material using the same formula” (step 7 above). A POSITA would understand that the shared key material ($K=y_i^{x_r} \bmod p$) generated using the same formula on the Initiator side ($K=y_r^{x_i} \bmod p$) will yield the same result (thereby establishing the pre-master secret K). In the formula used in the Initiator side, y_r is the public Diffie-Hellman server key, and x_i is the private Diffie-Hellman client key.

277. If the shared secret K is directly used as the session key, it is evident from the formula for the K that the session key is generated as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key.

278. A POSITA would have also understood that, to the extent the shared secret K is not directly used as the session key, but used as a pre-master secret

from which a session key is derived according to a pre-agreed key derivation function (as the case would be in SSL/TLS protocol), the session key is generated using the pre-master key K, which is in turn generated using the private Diffie-Hellman client key and the public Diffie-Hellman server key. Thus, the session key is generated as a function of public DH server key and the private DH client key.

8. Claim 24

- a. 24. The system-on-a-chip apparatus of claim 1, wherein the security engine is communicatively coupled to the processor core through an interconnect of the system-on-a-chip.**

279. Buer's security module and/or cryptographic accelerator is communicatively coupled to the host processor through an interconnect.

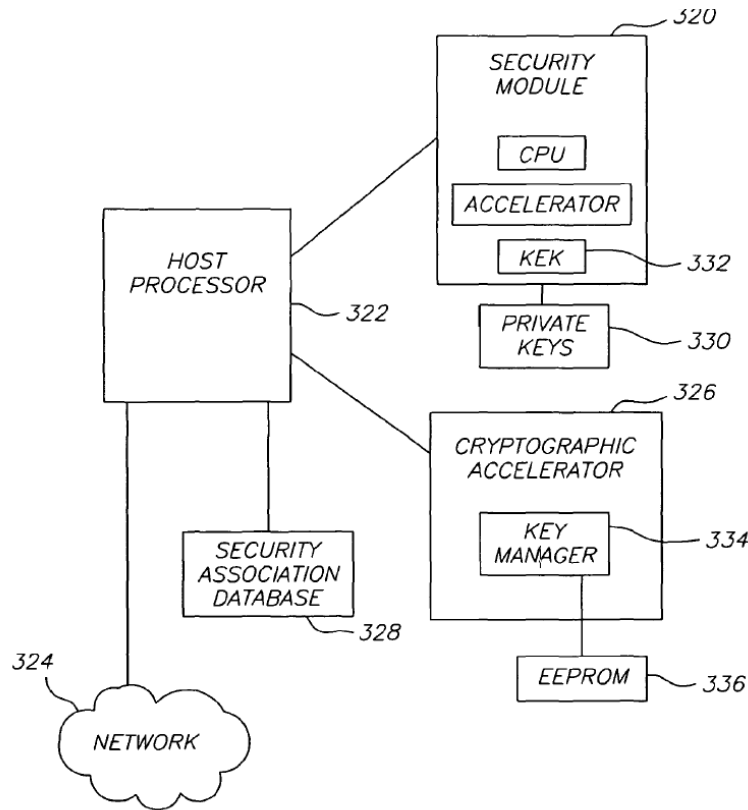


FIG. 3

Ex-1009, Fig. 3; *see also*, Figs. 1, 2, 4, 5, 10, 13. Buer also explains:

In addition, the components and functions described herein may be connected in many different ways. Some of the connections represented by the lead lines in the drawings may be in an integrated circuit, on a circuit board, over a backplane to other circuit boards, over a local network and/or over a wide area network (e.g., the Internet). Thus, some of the components may be located in a remote location with respect to the other components. Typically, one or more of the connections represented by the lead lines in the drawings (e.g., lead lines 240-244 and 1050-1060 in FIGS. 2 and 10) may, for example, comprise a data network. In addition, these connections may be made with physical wire, fiber and/or wireless connections, for example.

Some of the connections between components made

comprise secure connections (e.g., FIPS-140-2 compliant)
while other connections comprise unsecure connections.

Ex-1009, 16:47-62. Buer further explains that the various components, such as the security module and the host processor, may be “combined on one or more integrated circuits,” i.e., forming a system-on-a-chip. Ex-1009, 16:45-46.

9. Independent Claims 9, 17

a. 9[pre] A method comprising;

17[pre] One or more non-transitory machine-readable storage media comprising a plurality of instructions stored thereon that, in response to execution by a computing device, cause the computing device to:

280. To the extent limiting, Buer discloses the preamble of Claim 9. For example, Buer states it discloses a method:

Methods and systems are disclosed for providing secured data transmission and for managing cryptographic keys. One embodiment of the invention provides secure key management when separate devices are used for generating and utilizing the keys. One embodiment of the invention provides secure storage of keys stored in an unsecured database. One embodiment of the invention provides key security in conjunction with high speed decryption and encryption, without degrading the performance of the data network.

Ex-1009, Abstract.

281. Also to the extent limiting, Buer discloses the preamble of Claim 17. For example, Buer states its invention relates to the “field of data communications” and to “to systems and methods for providing secured data transmission and for

managing cryptographic keys.” Ex-1009, 1:6-9. Buer further explains that a variety of hardware and software components may be used to implement the functions of the main security module or the host processors. Ex-1009, 16:36-46. A POSITA would have understood that such software implementation would necessarily involve including one or more non-transitory machine-readable storage media, which stores the instructions that performs the disclosed methods of Buer’s invention when executing the disclosed system of Buer’s invention.

It should also be appreciated that the inventions described herein may be constructed using a variety of physical components and configurations. For example, a variety of hardware and software processing components may be used to implement the functions of the main security modules, satellite security modules, host processors, cryptographic accelerators, security modules, network controller and the packet processors. Typically, the network controller and packet processing functions may be implemented in a network processor. These components may be combined on one or more integrated circuits.

Ex-1009, 16:36-46.

b. 9[a] generating a random nonce in a security engine that is separate from a processor core of a system-on-a-chip of a client device;

17[a] generate a random nonce in a security engine that is separate from a processor core of a system-on-a-chip of the computing device;

282. Buer discloses limitations 9[a] and 17[a] for the same reasons I explained in Section **X.C.2.b**, limitation 1[a] (“a system-on-a-chip comprising a

security engine that is separate from a processor core of the system-on-a-chip . . .’)

and Section **X.C.2.c**, limitation 1[b] (the security engine to “generate a random nonce for initiating a request for a secure communication session with a remote server over a network using the nonce”).

- c. 9[b] initiating, using the client device, a request for a secure communication session with a remote server over a network, the request including the random nonce;**

17[b] initiate a request for a secure communication session with a remote server over a network, the request including the random nonce;

283. Buer discloses limitations 9[b] and 17[b] for the same reasons I explained in Section **X.C.2.c**, limitation 1[b] (“generate a random nonce for initiating a request for a secure communication session with a remote server over a network using the nonce”).

- d. 9[c] performing a cryptographic key exchange, using the security engine of the system-on-a-chip, with the remote server;**

17[c] perform a cryptographic key exchange, using the security engine of the system-on-a-chip, with the remote server;

284. Buer discloses limitations 9[c] and 17[c] for the same reasons I explained in Section **X.C.2.d**, limitation 1[c] (the security engine to “perform a cryptographic key exchange with the remote serve”).

- e. **9[d] generate a symmetric session key to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session;**

17[d] generate, based on the cryptographic key exchange, a symmetric session key to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session;

285. Buer discloses limitations 9[d] and 17[d] for the same reasons I explained in Section **X.C.2.e**, limitation 1[d] (“generate a symmetric session key, based on the cryptographic key exchange, to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session”).

- f. **9[e] encrypting the session key based on a security key that was encoded in a secure memory of the security engine during a manufacturing process of the system-on-a-chip;**

17[e] encrypt the session key based on a security key that was encoded in a secure memory of the security engine during a manufacturing process of the system-on-a-chip;

286. Buer discloses limitations 9[e] and 17[e] for the same reasons I explained in Section **X.C.2.f**, limitation 1[e] (“encrypt the symmetric session key based on the security key”) and Section **X.C.2.b**, limitation 1[a] (“. . . wherein the secure memory includes a security key that was encoded in the secure memory during a manufacturing process of the system-on-a-chip”).

- g. 9[f] storing the encrypted session key in the secure memory of the security engine of the system-on-a-chip; and**

17[f] store the session key in the secure memory of the security engine of the system-on-a-chip; and

287. Buer discloses limitations 9[f] and 17[f] for the same reasons I explained in Section **X.C.2.g**, limitation 1[f] (“store the encrypted session key in the secure memory”).

- h. 9[g] establishing, using the client device, the secure communication session with the remote server using the session key.**

17[g] establish the secure communication session with the remote server using the session key.

288. Buer discloses limitations 9[f] and 17[f] for the same reasons I explained in Section **X.C.2.h**, limitation 1[g] (“the system-on-a-chip to establish the secure communication session with the remote server over the network using the session key”).

10. Dependent Claims 10, 18

a. **10[pre]. The method of claim 9, wherein initiating a request for a secure communication session comprises sending a ClientHello message, including the nonce, to the remote server; and further comprising:**

18[pre]. The one or more non-transitory machine-readable storage media of claim 17, wherein to initiate a request for a secure communication session comprises to send a ClientHello message, including the nonce, to the remote server; and

289. Buer discloses that the security module “uses the SSL protocol to provide data security in a network.” Ex-1009, 6:13-15. As I explained above in Section **VII.B** (SSL/TLS Protocol), the SSL protocol includes standard handshake messages, which begins with a ClientHello message that includes a nonce generated by the client.

290. In the context of Buer, the initial message during the key exchange protocol steps sent by the main security module (called the “Initiator”) to the satellite security module (called the “Responder”) constitutes the ClientHello message:

FIG. 3 depicts an embodiment of the invention where a security module 320 uses the SSL protocol to provide data security in a network. A host processor 322 (e.g., a server) interfaces with a network 324 to communicate with client applications running on other processors (not shown) connected to the network 324. In a similar manner as discussed above, the host processor 322 cooperates with a cryptographic accelerator 326 and a database 328 containing secured security associations to decrypt

encrypted packets received from the network **324** and to encrypt packets sent to the network **324**.

Ex-1009, 6:13-23.

291. For example, Buer discloses, in the context of Unauthenticated Diffie-Hellman exchange, the initiator (i.e., the main security module) sends the first message (a ClientHello message) that includes the nonce generated by the initiator to the responder (the claimed “remote server”):

In Unauthenticated Diffie-Hellman the main security module and the satellite security module share public knowledge of a Diffie-Hellman instance, consisting of a prime number p and generator g . This is the basic Diffie-Hellman exchange.

1. Initiator generates random number x_i , and creates public number $y_i = g^{x_i} \bmod p$;
2. Initiator sends y_i to the responder, along with a nonce N_i ;
3. Responder generates a random number x_r , public number $y_r = g^{x_r} \bmod p$;
4. Responder generates the Diffie-Hellman shared secret, $K = y_i^{x_r} \bmod p$, and nonce N_r ;
5. Responder generates as much key material stream as needed using $\text{HMAC}(K, N_i \parallel N_r \parallel i)$, for $i=0, 1, 2, \dots$;
6. Responder returns y_r to the initiator, along with nonce N_r ;
7. Initiator derives shared key material using the same formula.

This is a simple and standard protocol that creates FIPS

quality shared key material. Here, the satellite security module only needs access to the Diffie-Hellman instance constants, and these could be included in the integrated circuit logic. However, this approach may be vulnerable to man-in-the-middle (MITM) attacks that could compromise the secure channel and all KEKs exchanged over the secure channel.

Ex-1009, 13:21-44.

292. Buer discloses similar steps of sending the initial message including nonce N_i in One-Way RSA Authenticated Diffie-Hellman, Mutually Authenticated Diffie-Hellman and Shared Secret ReKey key exchange steps as well. *See* Ex-1009, 13:45-14:10 (One-Way RSA Authenticated Diffie-Hellman beginning with a message that includes nonce N_i); 14:11-42 (Mutually Authenticated Diffie-Hellman); 14:43-59 (Shared Secret ReKey).

b. 10[a] receiving a ServerHello message from the remote server, the ServerHello message including a server nonce; and

18[a] wherein the plurality of instructions further cause the computing device to: receive a ServerHello message from the remote server, the ServerHello message including a server nonce; and

293. Buer further discloses that, in response to the initial message from the main security module (called the “Initiator”) (the ClientHello message), the satellite security module (called the “Responder”) sends a response message that includes a Responder-generated nonce N_r . This message constitutes the ServerHello message.

294. For example, Buer discloses, in the context of Unauthenticated Diffie-Hellman exchange, the Initiator receives the response from the Responder (a ServerHello message) that includes the nonce generated by the Responder:

In Unauthenticated Diffie-Hellman the main security module and the satellite security module share public knowledge of a Diffie-Hellman instance, consisting of a prime number p and generator g . This is the basic Diffie-Hellman exchange.

1. Initiator generates random number x_i , and creates public number $y_i = g^{x_i} \bmod p$;
2. Initiator sends y_i to the responder, along with a nonce N_i ;
3. Responder generates a random number x_r , public number $y_r = g^{x_r} \bmod p$;
4. Responder generates the Diffie-Hellman shared secret, $K = y_i^{x_r} \bmod p$, and nonce N_r ;
5. Responder generates as much key material stream as needed using $\text{HMAC}(K, N_i \parallel N_r \parallel i)$, for $i=0, 1, 2, \dots$;
6. Responder returns y_r to the initiator, along with nonce N_r ;
7. Initiator derives shared key material using the same formula.

This is a simple and standard protocol that creates FIPS quality shared key material. Here, the satellite security module only needs access to the Diffie-Hellman instance constants, and these could be included in the integrated circuit logic. However, this approach may be vulnerable to man-in-the-middle (MITM) attacks that could compromise the secure channel and all KEKs exchanged over the secure channel.

Ex-1009, 13:21-44.

295. Buer discloses similar steps of the Responder sending a message including nonce Nr in One-Way RSA Authenticated Diffie-Hellman, Mutually Authenticated Diffie-Hellman and Shared Secret ReKey key exchange steps as well. *See* Ex-1009, 13:45-14:10 (One-Way RSA Authenticated Diffie-Hellman); 14:11-42 (Mutually Authenticated Diffie-Hellman); 14:43-59 (Shared Secret ReKey).

c. **10[b] storing the server nonce in the secure memory of the security engine of the system-on-a-chip.**

18[b] store the server nonce in the secure memory of the security engine of the system-on-a-chip.

(1) Disclosure in Buer (Ground 3)

296. Buer discloses data memory locations that can be used by the security engine. These data locations include, for example, the non-volatile memory **134** connected to key manager **132** (see Fig. 1) or private keys store **330** attached to security module **320** (see Fig. 3). A POSITA would have recognized that, because Buer contemplates generating session keys from the data received through the cryptographic key exchange using the security module, it would be most efficient to store the data received through the key exchange within the cryptographic boundaries of the security module, thus eliminating the need for additional communication with otherwise unsecured host processor. This can also improve

system performance by reducing the number of memory accesses required to retrieve the encrypted session key. Thus, A POSITA would have been motivated to store the encrypted session keys inside the secure memory that is directly attached to the security module.

297. In addition, a POSITA would have further recognized that, storing the server nonce, which is further used by the security module to generate the shared key material, would eliminate the need to manage such data in a secure manner outside the security module. Because the non-volatile memory is already access controlled by being coupled to the security module, separate management or access control to the security association database is not needed. Thus a POSITA would have been motivated, from Buer's disclosure of non-volatile memory coupled to the security module, to the non-volatile memory to store the data used by the security module, such as the server nonce.

(2) Combination with Chou (Ground 4)

298. Chou discloses storing the data to be processed by the security processor in the secure memory. For example, Chou discloses a Hardware Security Module (HSM), which includes a memory used to store data that is processed and/or to be processed by the security processor.

[0021] HSM 20 includes a security processor 21, secure memory 22, secure key module 23, secure bus 26 and a security interface 24. The various components **21-24** and **26** reside within a secure boundary established by

the hardware. That is, a secure domain or zone is established by and for the components within HSM 20, in order to set a secure boundary for processing that occurs within this boundary. Security processor 21 provides the various processing within HSM 20, including key authentication and the encrypting and/or decrypting operation to encrypt and/or decrypt data. Secure key module 23 is a storage medium, such as a memory, register, circuit, etc. that stores one or more key value(s) that is used by security processor 21 to encrypt and decrypt data. Secure memory 22 is utilized to store data that is processed and/or to be processed by security processor 21.

Ex-1008 ¶21. Fig. 1, reproduced with highlights showing the security engine below, shows a simplified block diagram of the device incorporating an HSM.

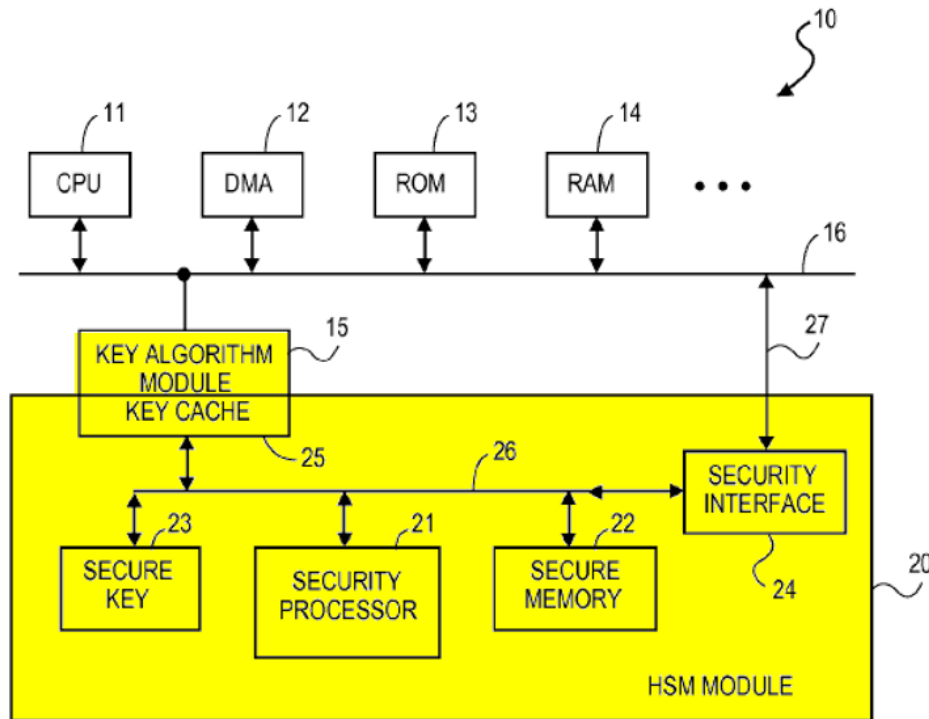


FIG. 1

Ex-1008, Fig. 1.

299. In the Buer's system modified to incorporate Chou's secure memory into the security module, server nonce (the nonce Nr received from the satellite security module) is one of the data processed by the CPU of the main security module. Thus, in view of Chou's disclosure that the data processed by the security module is to be stored in the secure memory, Buer's system modified to incorporate Chou's secure memory would store the server nonce in the secure memory inside the security module.

300. A POSITA would have recognized that Chou's secure memory inside HSM provide several benefits and would have been motivated to modify Buer's system to incorporate Chou's secure memory for the reasons I explained above in Section X.C.1.

11. Dependent Claims 11, 19

a. **11[pre]/11[a]. "The method of claim 9, wherein performing the cryptographic key exchange with the remote server comprises: generating a pre-master key in the security engine"**

19[pre]/19[a]. "The one or more non-transitory machine-readable storage media of claim 17, wherein to perform the cryptographic key exchange with the remote server comprises to: generate a pre-master key in the security engine"

301. Buer discloses limitations 11[a] and 19[a] for the same reasons I explained in Section X.C.3.a, Claim 3 ("The system-on-a-chip apparatus of claim 1, wherein the security engine is further to generate a pre-master key . . .").

- b. 11[b] “encrypting the pre-master key using the security key of the security engine”**

19[b] “encrypt the pre-master key using the security key of the security engine

302. Buer discloses limitations 11[b] and 19[b] for the same reasons I explained in Section **X.C.3.a**, Claim 3 (“encrypt the pre-master key using the security key of the security engine”)

- c. 11[c] “sending the encrypted pre-master key to the remote server”**

19[c] “send the encrypted pre-master key to the remote server”

303. Buer discloses limitations 11[c] and 19[c] for the same reasons I explained in Section **X.C.3.a**, Claim 3 (“send the encrypted pre-master key to the remote server”).

12. Dependent Claims 12, 20

- a. 12[pre]/12[a]: The method of claim 11, further comprising: wrapping the pre-master key in the security engine of the system-on-a-chip using a server public key, and**

20[pre]/20[a]: The one or more non-transitory machine-readable storage media of claim 19, wherein the plurality of instructions further cause the computing device to: wrap the pre-master key in the security engine of the system-on-a-chip with a server public key, and

304. Buer discloses limitations 12[a] and 20[a] for the same reasons I explained in Section **X.C.4.a**, Claim 4 (“The system-on-a-chip apparatus of claim

3 wherein the security engine is further to wrap the pre-master key using a server public key and store the wrapped pre-master key in the secure memory.”).

- b. 12[b] storing the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip.**

20[b] store the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip.

305. Buer discloses limitations 12[b] and 20[b] for the same reasons I explained in Section **X.C.4.a**, Claim 4 (“The system-on-a-chip apparatus of claim 3 wherein the security engine is further to wrap the pre-master key using a server public key and store the wrapped pre-master key in the secure memory.”).

13. Dependent Claims 13, 21

- a. 13. The method of claim 11, wherein performing the cryptographic key exchange with the remote server comprises generating the session key in the security engine of the system-on-a-chip as a function of the pre-master key.**

21. The one or more non-transitory machine-readable storage media of claim 19, wherein to perform the cryptographic key exchange with the remote server comprises to generate the session key in the security engine of the system-on-a-chip as a function of the pre-master key.

306. Buer discloses Claims 13 and 21 for the same reasons I explained in Section **X.C.5.a**, limitation 5[pre] and 5[a] (“The system-on-a-chip apparatus of claim 3, wherein the security engine is further to: generate the session key as a

function of the pre-master key”).

14. Dependent Claims 14, 22

a. 14. The method of claim 13, wherein generating the session key in the security engine comprises calculating, using the security engine of the system-on-a-chip, a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server.

22. The one or more non-transitory machine-readable storage media of claim 21, wherein to generate the session key in the security engine comprises to calculate, using the security engine of the system-on-a-chip, a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server.

307. Buer discloses Claims 13 and 21 for the same reasons I explained in Section **X.C.5.b**, imitation 5[b] (“calculate a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server”)

15. Claims 15, 23

- a. 15[pre]/15[a] The method of claim 9 wherein performing the cryptographic key exchange with the remote server comprises: generating, in the security engine of the system-on-a-chip, a public Diffie-Hellman client key and a private Diffie-Hellman client key;**

23[pre]/23[a] The one or more non-transitory machine-readable storage media of claim 17, wherein to perform the cryptographic key exchange with the remote server comprises to: sign the public Diffie-Hellman client key using the security key of the security engine, and

308. Buer discloses limitations 15[a] and 23[a] for the same reasons I explained in Section **X.C.6.a**, limitation 6[a] (“The system-on-a-chip apparatus of claim 1, wherein the security engine is further to generate a public Diffie-Hellman client key and a private Diffie-Hellman client key”).

- b. 15[b] signing the public Diffie-Hellman client key using the security key of the security engine of the system-on-a-chip;**

23[b] sign the public Diffie-Hellman client key using the security key of the security engine of the system-on-a-chip;

309. Buer discloses limitations 15[b] and 23[b] for the same reasons I explained in Section **X.C.6.b**, limitation 6[b] (“sign the public Diffie-Hellman client key using the security key of the security engine”).

- c. **15[c] sending the signed public Diffie-Hellman key to the remote server;**

23[c] send the signed public Diffie-Hellman key to the remote server;

310. Buer discloses limitations 15[c] and 23[c] for the same reasons I explained in Section **X.C.6.c**, limitation 6[c] (“send the signed public Diffie-Hellman key to the remote server”).

- d. **15[d] receiving a public Diffie-Hellman server key from the remote server; and**

23[d] receive a public Diffie-Hellman server key from the remote server; and

311. Buer discloses limitations 15[d] and 23[d] for the same reasons I explained in Section **X.C.7.a**, limitation 7[a] (“The system-on-a-chip apparatus of claim 6, wherein the security engine is further to receive a public Diffie-Hellman server key from the remote server”).

- e. **15[e] generating, in the security engine of the system-on-a-chip, the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key.**

23[e] generate, in the security engine of the system-on-a-chip, the session key as a function of the private DiffieHellman client key and the public Diffie-Hellman server key.

312. Buer discloses limitations 15[d] and 23[d] for the same reasons I explained in Section **X.C.7.b**, limitation 7[b] (“generate the session key as a

function of the private Diffie-Hellman client key and the public Diffie-Hellman server key.”).

D. Grounds 5 and 6: Grounds 3 and 4, in further combination with TLS-1.0

313. To the extent Buer, alone (Ground 3) or in combination with Chou (Ground 4) does not disclose any limitations of the challenged claims, those limitations are obvious in further combination with TLS-1.0, as further explained below. The contemplated combination would retain the same disclosed structure of Buer, alone or in combination with Chou, while implementing the key exchange steps between the main security module and the satellite security module using SSL/TLS protocol.

1. Motivation to Combine

314. As explained in Section VII.B above, SSL protocol and its successor, TLS protocol, specifies a standard protocol to be used by two communicating entities (such as a client device and a web server) to establish an encrypted and authenticated communication channel. SSL/TLS protocol was widely used by the priority date of the '838 Patent. Buer itself notes that the security module “uses the SSL protocol to provide data security in a network.” Ex-1009, 6:13-15.

FIG. 3 depicts an embodiment of the invention where a security module 320 uses the SSL protocol to provide data security in a network. A host processor 322 (e.g., a server) interfaces with a network 324 to communicate with client applications running on other processors (not shown)

connected to the network 324. In a similar manner as discussed above, the host processor 322 cooperates with a cryptographic accelerator 326 and a database 328 containing secured security associations to decrypt encrypted packets received from the network 324 and to encrypt packets sent to the network 324.

As discussed above, SSL uses a public/private key scheme. Thus, a secure connection may be established using standard public key techniques. For example, a server sends its signed public key to a client, the client verifies the public key then sends session information to the server encrypted with the server's public key. The keys to be used for the session are then derived from this information and the server's private key.

Ex-1009, 6:13-31. Thus, to the extent not already disclosed in Buer itself by the reference to the SSL protocol, a POSITA would have been motivated to incorporate the specific teachings of the SSL protocol in the communication between the main security module and the satellite security module.

315. A POSITA would have recognized that the various key exchange steps disclosed in Buer can be implemented using the standard TLS handshake procedure, and would have been motivated to incorporate TLS handshake protocol into Buer's cryptographic key exchange steps between the main security module and the satellite security module. This is because, as explained above, there were several open-source libraries implementing the SSL/TLS protocol that were readily available for anyone trying to implement a system that supports the SSL/TLS protocol. A POSITA would have realized that Buer's cryptographic key exchange

steps can be easily implemented by adopting the SSL/TLS protocol as the key exchange method between the main security module and the satellite security module.

316. A POSITA would have further been motivated to use SSL/TLS protocol, because doing so would allow the implementer to use any one of the many available SSL/TLS open source libraries, rather than writing the software to implement each of the exchange steps from scratch. Utilizing the available SSL/TLS open source libraries would have allowed the implementer to save significant amount of development time, and save development resources and manpower. Adopting SSL/TLS would have also allowed the implementer to take advantage of the large community of developers who contribute to the code, provide support, and continuously improve the library, which helps to ensure that the library is up-to-date and can adapt to changing technology trends. Open-source libraries are also usually subject to peer review, which can help identify security vulnerabilities and ensure that the library is secure. Thus, a POSITA would have recognized each of these benefits, and would have been motivated to adopt SSL/TLS standard protocol to take advantage of the available open source software.

317. In addition, because open-source libraries are usually well-tested and have gone through multiple revisions to ensure quality and reliability, a POSITA

would have had a reasonable expectation of success of implementing the SSL/TLS protocol between the main security module and the satellite security module, based on the widely available open-source community resources and their experiences.

2. Dependent Claim 3

- a. 3. The system-on-a-chip apparatus of claim 1, wherein the security engine is further to generate a pre-master key, encrypt the pre-master key using the security key of the security engine, and send the encrypted pre-master key to the remote server.**

318. To the extent Buer, alone (Ground 3) or in combination with Chou (Ground 4), does not disclose Claim 3, a POSITA would have found Claim 3 obvious in light of TLS-1.0, which was a well-established and widely-used standard protocol before the priority date of the '838 Patent.

319. TLS-1.0 specifies the use of RSA-encrypted premaster secret message which discloses the client device (such as the main security module of Buer) creating a premaster secret message, which requires generating a pre-master key, encrypting the pre-master key using the public key of the server (received in a server certificate), and sending the encrypted pre-master key to the server:

7.4.7.1. RSA encrypted premaster secret message

Meaning of this message:

If RSA is being used for key agreement and authentication, the client generates a 48-byte premaster secret, encrypts it using the public key from the server's certificate or the temporary RSA key provided in a server key exchange message, and sends the result in an encrypted premaster secret message. This structure is a variant of the client key exchange message, not a message in itself.

Ex-1006, §7.4.7.1. Thus, to the extent Buer (Ground 3) or Buer in combination with Chou (Ground 4) does not disclose this claim, this claim is obvious in further combination with TLS-1.0.

3. Dependent Claim 4

- a. 4. The system-on-a-chip apparatus of claim 3 wherein the security engine is further to wrap the pre-master key using a server public key and store the wrapped pre-master key in the secure memory.**

320. To the extent the security engine “to wrap the pre-master key using a server public key “ is not explicitly disclosed in Buer, a POSITA would have found Claim 4 obvious in further combination with TLS-1.0.

321. As explained in Section **X.D.2** (Claim 3, combination with TLS-1.0) above, TLS-1.0 specifies that the premaster secret is encrypted using the server (responder)’s public key. For example TLS-1.0 states:

7.4.7.1. RSA encrypted premaster secret message

Meaning of this message:

If RSA is being used for key agreement and authentication, the client generates a 48-byte premaster secret, encrypts it using the public key from the server's certificate or the temporary RSA key provided in a server key exchange message, and sends the result in an encrypted premaster secret message. This structure is a variant of the client key exchange message, not a message in itself.

Ex-1006, §7.4.7.1.

322. Buer, alone or in combination with Chou discloses all remaining elements of Claim 4, as explained in Section X.C.4 (Grounds 3 and 4) above.

4. Dependent Claim 5

a. 5[pre]/5[a]: The system-on-a-chip apparatus of claim 3, wherein the security engine is further to: generate the session key as a function of the pre-master key

5[b] calculate a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server

323. To the extent Buer (Grounds 3 and 4) does not disclose “generate the session key as a function of the pre-master key” (limitation 5[a]), SSL/TLS protocol discloses it. A POSITA would have been motivated to implement the communication between the main security module and the satellite security module using SSL/TLS protocol, as explained above in Section X.D.1 (Motivation to combine Ground 3 (Buer) or Ground 4 (Buer in combination with Chou) with TLS-1.0).

324. TLS-1.0 specifies that “master secret” (session key) is created by

following method:

8.1. Computing the master secret

For all key exchange methods, the same algorithm is used to convert the `pre_master_secret` into the `master_secret`. The `pre_master_secret` should be deleted from memory once the `master_secret` has been computed.

```
master_secret = PRF(pre_master_secret, "master  
secret", ClientHello.random + ServerHello.random)
```

```
[0..47];
```

The master secret is always exactly 48 bytes in length. The length of the premaster secret will vary depending on key exchange method.

Ex-1006, §8.1.

325. The statement “`master_secret = PRF(pre_master_secret, "master secret", ClientHello.random + ServerHello.random) [0..47]`” refers to the process of generating a shared master secret key for subsequent TLS communications. The master secret key is generated by applying the Pseudo-Random Function (PRF) to the pre-master secret key (which is generated using the Diffie-Hellman key exchange algorithm or other key exchange methods), along with the “master secret” label and a concatenated value of the client and server random values (`ClientHello.random + ServerHello.random`).

326. The PRF is a cryptographic function that takes as input a shared secret key and a label, and generates a pseudo-random output that can be used as a symmetric key for encryption and decryption. The PRF function used in TLS is

defined in the TLS specification, and is designed to ensure that the output is unpredictable and provides sufficient entropy for use as a symmetric key. The [0..47] notation at the end of the statement indicates that the resulting master secret key should be truncated to a maximum length of 48 bytes (384 bits).

327. This key generation method takes the pre_master_secret as input, i.e., generate the session key as a function of the pre-master key.

328. Buer, alone or in combination with Chou discloses all remaining elements of Claim 5, as explained in Section X.C.5 (Grounds 3 and 4) above.

5. Dependent Claim 7

7[a] The system-on-a-chip apparatus of claim 6, wherein the security engine is further to receive a public Diffie-Hellman server key from the remote server and

7[b] generate the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key.

329. To the extent Buer does not disclose generating a session key from Diffie-Hellman shared secret K, and therefore “generat[ing] the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key” (limitation 7[b]), SSL/TLS protocol discloses it.

330. TLS-1.0 specifies that “master secret” (session key) is created by following method:

8.1. Computing the master secret

For all key exchange methods, the same algorithm is used to convert the `pre_master_secret` into the `master_secret`. The `pre_master_secret` should be deleted from memory once the `master_secret` has been computed.

```
master_secret = PRF(pre_master_secret, "master  
secret", ClientHello.random + ServerHello.random)  
  
[0..47];
```

The master secret is always exactly 48 bytes in length. The length of the premaster secret will vary depending on key exchange method.

Ex-1006, §8.1.

331. The statement “`master_secret = PRF(pre_master_secret, "master secret", ClientHello.random + ServerHello.random) [0..47]`” refers to the process of generating a shared master secret key for subsequent TLS communications. The master secret key is generated by applying the Pseudo-Random Function (PRF) to the pre-master secret key (which is generated using the Diffie-Hellman key exchange algorithm or other key exchange methods), along with the “master secret” label and a concatenated value of the client and server random values (`ClientHello.random + ServerHello.random`).

332. The PRF is a cryptographic function that takes as input a shared secret key and a label, and generates a pseudo-random output that can be used as a symmetric key for encryption and decryption. The PRF function used in TLS is defined in the TLS specification, and is designed to ensure that the output is unpredictable and provides sufficient entropy for use as a symmetric key. The

[0..47] notation at the end of the statement indicates that the resulting master secret key should be truncated to a maximum length of 48 bytes (384 bits).

333. This key generation method takes the `pre_master_secret` as input, which, in Diffie-Hellman method, generated using private Diffie-Hellman client key and the public Diffie-Hellman server key.

334. Buer, alone or in combination with Chou discloses all remaining elements of Claim 7, as explained in Section X.C.7 (Grounds 3 and 4) above.

6. Dependent Claims 10, 18

a. **10[pre]. The method of claim 9, wherein initiating a request for a secure communication session comprises sending a ClientHello message, including the nonce, to the remote server; and further comprising:**

18[pre]. The one or more non-transitory machine-readable storage media of claim 17, wherein to initiate a request for a secure communication session comprises to send a ClientHello message, including the nonce, to the remote server; and

335. To the extent Buer does not explicitly use the term “ClientHello” to refer to the initial message from the Initiator to the Responder, a POSITA would have found this limitation obvious in light of TLS-1.0, which was a well-established and widely-used standard protocol before the priority date of the ’838 Patent.

336. TLS-1.0 specifies the various messages exchanged between the

initiator and the responder during the handshake to establish a secure session, and their sequences. For example, TLS-1.0 discloses:

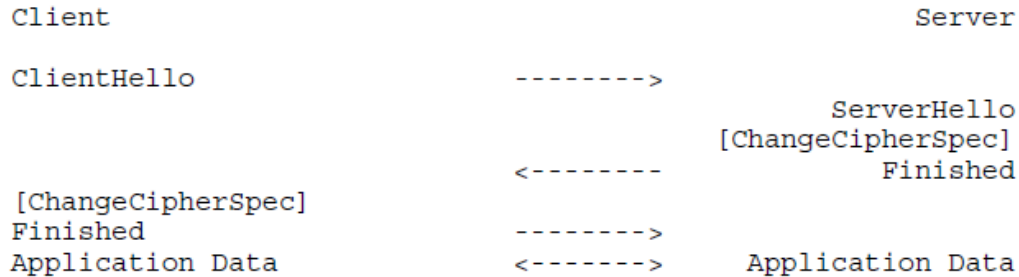


Fig. 2 - Message flow for an abbreviated handshake

Ex-1006, §7.3, Fig. 2.

These goals are achieved by the handshake protocol, which can be summarized as follows: The client sends a client hello message to which the server must respond with a server hello message, or else a fatal error will occur and the connection will fail. The client hello and server hello are used to establish security enhancement capabilities between client and server. The client hello and server hello establish the following attributes: Protocol Version, Session ID, Cipher Suite, and Compression Method. Additionally, two random values are generated and exchanged: ClientHello.random and ServerHello.random.

Ex-1006, §7.3.

337. TLS-1.0 further discloses the client hello message includes

ClientHello.random:

7.4.1.2. Client hello

When this message will be sent:

When a client first connects to a server it is required to send the client hello as its first message. The client can also send a client hello in response to a hello request or on

its own initiative in order to renegotiate the security parameters in an existing connection.

Structure of this message:

The client hello message includes a random structure, which is used later in the protocol.

```
struct {  
    uint32 gmt_unix_time;  
    opaque random_bytes[28];  
} Random;
```

gmt_unix_time

The current time and date in standard UNIX 32-bit format (seconds since the midnight starting Jan 1, 1970, GMT) according to the sender's internal clock. Clocks are not required to be set correctly by the basic TLS Protocol; higher level or application protocols may define additional requirements.

random_bytes

28 bytes generated by a secure random number generator.

Ex-1006, §7.4.1.2.

b. 10[a] receiving a ServerHello message from the remote server, the ServerHello message including a server nonce; and

18[a] wherein the plurality of instructions further cause the computing device to: receive a ServerHello message from the remote server, the ServerHello message including a server nonce; and

338. To the extent Buer does not explicitly use the term “ServerHello” to refer to the Responder’s message sent in response to the ClientHello message, a

POSITA would have found this limitation obvious in light of TLS-1.0, which was a well-established and widely-used standard protocol before the priority date of the '838 Patent.

339. TLS-1.0 specifies the various messages exchanged between the initiator and the responder during the handshake to establish a secure session, and their sequences. For example, TLS-1.0 discloses:

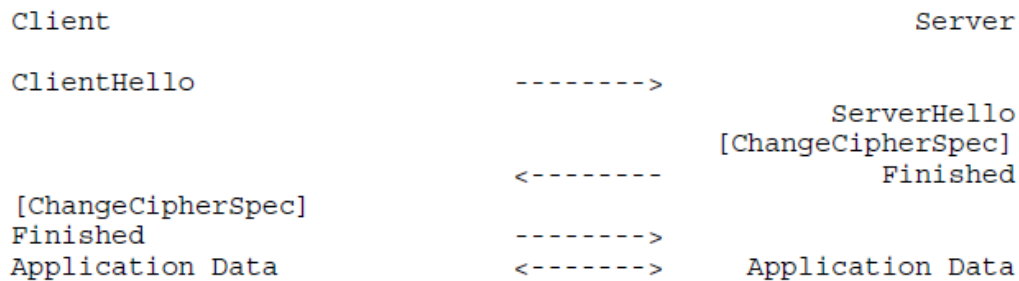


Fig. 2 - Message flow for an abbreviated handshake

Ex-1006, §7.3, Fig. 2.

These goals are achieved by the handshake protocol, which can be summarized as follows: The client sends a client hello message to which the server must respond with a server hello message, or else a fatal error will occur and the connection will fail. The client hello and server hello are used to establish security enhancement capabilities between client and server. The client hello and server hello establish the following attributes: Protocol Version, Session ID, Cipher Suite, and Compression Method. Additionally, two random values are generated and exchanged: ClientHello.random and ServerHello.random.

Ex-1006, §7.3.

340. TLS-1.0 further discloses the server hello message includes

ServerHello.random:

77.4.1.3. Server hello

When this message will be sent:

The server will send this message in response to a client hello message when it was able to find an acceptable set of algorithms. If it cannot find such a match, it will respond with a handshake failure alert.

Structure of this message:

```
struct {  
    ProtocolVersion server_version;  
    Random random;  
    SessionID session_id;  
    CipherSuite cipher_suite;  
    CompressionMethod compression_method;  
} ServerHello;
```

RFC2246 at §7.4.1.3.

random

This structure is generated by the server and must be different from (and independent of) ClientHello.random.

Ex-1006, §7.4.1.3.

341. Thus, to the extent not already disclosed in Buer in view of the knowledge of a POSITA, a POSITA would have been motivated to incorporate the TLS handshake protocol into Buer's cryptographic key exchange steps between the main security module and the satellite security module, for the same reasons explained above.

- c. **10[b] storing the server nonce in the secure memory of the security engine of the system-on-a-chip.**

18[b] store the server nonce in the secure memory of the security engine of the system-on-a-chip.

342. *See* Section **X.C.10.c**, disclosure in Buer and/or Chou.

7. Dependent Claims 11, 19

- a. **11 The method of claim 9, wherein performing the cryptographic key exchange with the remote server comprises: generating a pre-master key in the security engine; encrypting the pre-master key using the security key of the security engine; and sending the encrypted pre-master key to the remote server.**

19 The one or more non-transitory machine-readable storage media of claim 17, wherein to perform the cryptographic key exchange with the remote server comprises to: generate a pre-master key in the security engine encrypt the pre-master key using the security key of the security engine; and send the encrypted pre-master key to the remote server.

343. Buer in combination with TLS-1.0 (Ground 5) or Buer in combination with Chou and TLS-1.0 (Ground 6) renders Claims 11 and 19 obvious for the same reasons I explained in Section **X.C.3.a**, Claim 3 (“The system-on-a-chip apparatus of claim 1, wherein herein the security engine is further to generate a pre-master key, encrypt the pre-master key using the security key of the security engine, and send the encrypted pre-master key to the remote server.”).

8. Dependent Claims 12, 20

- a. 12[pre]/12[a]: The method of claim 11, further comprising: wrapping the pre-master key in the security engine of the system-on-a-chip using a server public key, and
12[b] storing the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip.**

**20[pre]/20[a]: The one or more non-transitory machine-readable storage media of claim 19, wherein the plurality of instructions further cause the computing device to: wrap the pre-master key in the security engine of the system-on-a-chip with a server public key, and
20[b] store the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip.**

344. To the extent “wrapping the pre-master key . . . using a server public key” (limitations 12[a] and 20[a]) is not explicitly disclosed in Buer, a POSITA would have found this limitation obvious in further combination with TLS-1.0.

345. TLS-1.0 discloses limitations 12[a] and 20[a] as explained in Section **X.D.3**, Claim 4 (“The system-on-a-chip apparatus of claim 3 wherein the security engine is further to wrap the pre-master key using a server public key and store the wrapped pre-master key in the secure memory.”). Thus, Buer in combination with TLS-1.0 (Ground 5) or Buer in combination with Chou and TLS-1.0 (Ground 6) renders Claims 12 and 20 obvious.

9. Dependent Claims 13, 21

a. 13. The method of claim 11, wherein performing the cryptographic key exchange with the remote server comprises generating the session key in the security engine of the system-on-a-chip as a function of the pre-master key.

21. The one or more non-transitory machine-readable storage media of claim 19, wherein to perform the cryptographic key exchange with the remote server comprises to generate the session key in the security engine of the system-on-a-chip as a function of the pre-master key.

346. To the extent Buer does not disclose “generating the session key in the security engine of the system-on-a-chip as a function of the pre-master key”, SSL/TLS protocol discloses it for the same reasons I explained in Section **X.D.4.a**, limitation 5[pre] and 5[a] (“The system-on-a-chip apparatus of claim 3, wherein the security engine is further to: generate the session key as a function of the pre-master key”). Thus, Buer in combination with TLS-1.0 (Ground 5) or Buer in combination with Chou and TLS-1.0 (Ground 6) renders Claims 13 and 21 obvious.

10. Dependent Claims 14, 22

a. 14. The method of claim 13, wherein generating the session key in the security engine comprises calculating, using the security engine of the system-on-a-chip, a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server.

22. The one or more non-transitory machine-readable storage media of claim 21, wherein to generate the session key in the security engine comprises to calculate, using the security engine of the system-on-a-chip, a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server.

347. Buer in combination with TLS-1.0 (Ground 5) or Buer in combination with Chou and TLS-1.0 (Ground 6) renders Claims 13 and 21 obvious, as discussed above. Buer further discloses the additional limitations of dependent Claims 14 and 22 for the same reasons I explained in Section **X.C.5.b**, limitation 5[b] (“calculate a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server”). Thus, Buer in combination with TLS-1.0 (Ground 5) or Buer in combination with Chou and TLS-1.0 (Ground 6) renders Claims 14 and 22 obvious.

11. Claims 15, 23

- a. 15[e] generating, in the security engine of the system-on-a-chip, the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key.**

23[e] generate, in the security engine of the system-on-a-chip, the session key as a function of the private DiffieHellman client key and the public Diffie-Hellman server key.

348. To the extent Buer does not disclose generating a session key from Diffie-Hellman shared secret K, and therefore “generating . . . the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key” (limitations 15[e], 23[e]), SSL/TLS protocol discloses it. *See* Section **X.D.5**, limitation 7[b] (“generate the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key.”).

349. Buer, alone or in combination with Chou, discloses all other limitations of Claims 15 and 23, as explained in Sections **X.C.6.a** through **X.C.7.a**. Thus, Buer in combination with TLS-1.0 (Ground 5) or Buer in combination with Chou and TLS-1.0 (Ground 6) renders Claims 15 and 23 obvious.

XI. CONCLUSION

350. I declare that all statements made herein of my knowledge are true, and that all statements made on information and belief are believed to be true, and that these statements were made with the knowledge that willful false statements

and the like so made are punishable by fine or imprisonment, or both, under
Section 1001 of Title 18 of the United States Code.

Date: March 20, 2023

A handwritten signature in black ink, reading "Mark E. Crovella". The signature is written in a cursive, flowing style.

Dr. Mark E. Crovella