

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

MediaTek Inc.,

Petitioner,

v.

Daedalus Prime LLC,

Patent Owner.

Case: IPR2025-00100

U.S. Patent No. 9,887,838

Issue Date: February 6, 2018

Title: Method and Device for Secure Communications Over a Network
Using a Hardware Security Engine

PETITION FOR *INTER PARTES* REVIEW

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	STANDING.....	1
III.	RELIEF REQUESTED.....	1
IV.	THE '838 PATENT	1
	A. Overview	1
	B. Person of Ordinary Skill in the Art (“POSITA”).....	3
V.	CLAIM CONSTRUCTION	4
VI.	PRIOR ART	4
	A. Bugiel	4
	B. Chou	5
	C. TLS-1.2	6
	D. Tkacik.....	7
VII.	Grounds 1A, 1B, and 1C Render Obvious Claims 1-24	8
	A. Motivation to Combine Bugiel, Chou and, optionally, TLS-1.2	8
	B. Motivation to Combine Bugiel, Chou, and Tkacik	11
	C. Claim 1	13
	1. [1p] “system-on-a-chip apparatus”	13
	2. [1a-1] “security engine that is separate from a processor core”	15
	3. [1a-2] “a secure memory accessible only by the security engine,”	20
	4. [1a-3] “a security key ... encoded ... during a manufacturing process,”	23

5.	[1b] “generate a random nonce”	26
6.	[1c] “cryptographic key exchange”	28
7.	[1d] “generate a symmetric session key”	37
8.	[1e] “encrypt the symmetric session key”	42
9.	[1f] “store the encrypted session key”	43
10.	[1g] “establish the secure communication session ... using the session key.”	46
D.	Claim 2: “receive a server nonce”	48
E.	Claim 3: “pre-master key”	50
F.	Claim 4: “wrap the pre-master key”	54
G.	Claim 5	55
1.	[5a]: “generate the session key”	55
2.	[5b]: “calculate a hash function”	56
H.	Claim 6	57
1.	[6a]: “generate ... Diffie-Hellman key[s]”	57
2.	[6b]: “sign ..., and send”	59
I.	Claim 7: “public Diffie-Hellman server key”	60
J.	Claim 8: “Diffie-Hellman global values”	61
K.	Claims 9-15	62
L.	Claim 16	64
1.	[16a]: “hash code of a preceding message”	64
2.	[16b]: “signing the hash code”	64
3.	[16c]: “certificate verification message”	65

M.	Claim 17	65
1.	[17p]	65
2.	[17a]-[17g]	65
N.	Claims 18-23	66
O.	Claim 24	66
VIII.	Ground 2A, 2B: Tkacik and TLS-1.2, and Tkacik, TLS-1.2, and Bugiel Render Obvious Claims 1-24	68
A.	Motivation to Combine Tkacik and TLS-1.2	68
B.	Motivation to Combine Tkacik-TLS-1.2 with Bugiel	69
C.	Claim 1	70
1.	[1p]	70
2.	[1a-1]	72
3.	[1a-2]	74
4.	[1a-3]	76
5.	[1b]	76
6.	[1c]	79
7.	[1d]	81
8.	[1e]	83
9.	[1f]	85
10.	[1g]	88
D.	Claim 2	90
E.	Claim 3	90
F.	Claim 4	91

G.	Claim 5	92
1.	[5a]	92
2.	[5b]	92
H.	Claim 6	93
1.	[6a]	93
2.	[6b]	93
I.	Claim 7	93
J.	Claim 8	93
K.	Claims 9-15	94
L.	Claim 16	96
1.	[16a]	96
2.	[16b]	96
3.	[16c]	96
M.	Claim 17	96
1.	[17p]	96
2.	[17a]-[17g]	96
N.	Claims 18-23	97
O.	Claim 24	98
IX.	The Board Should Institute Review	98
X.	MANDATORY NOTICES UNDER 37 C.F.R. § 42.8	100
A.	Real Party-in-Interest	100
B.	Related Matters	100
C.	Lead and Backup Counsel Information	100

XI. CONCLUSION	101
----------------------	-----

LIST OF EXHIBITS

Exhibit	Description
EX-1001	U.S. Patent No. 9,887,838 (“the ’838 patent”)
EX-1002	S. Bugiel, et al., “TruWalletM: Secure Web Authentication on Mobile Platforms,” Second International Conference on Trusted Systems (INTRUST 2010) (Dec. 13-15, 2010) (“Bugiel”)
EX-1003	U.S. Patent Application Publication No. 2011/0191599 to Chou (“Chou”)
EX-1004	T. Dierks, The TLS Protocol Version 1.2, Internet Engineering Task Force (IETF), Network Working Group Request for Comments No. 5246 (August 2008) (“TLS-1.2”)
EX-1005	U.S. Patent Application Publication No. 2010/0296651 to Tkacik (“Tkacik”)
EX-1006	Stephen Thomas, SSL and TSL Essentials: Securing the Web (2000)
EX-1007	U.S. Patent Application Publication No. 2006/0107032 to Paaske
EX-1008	U.S. Patent No. 7,773,754 to Buer
EX-1009	John Crowe, Introduction to Digital Electronics (1998)
EX-1010	U.S. Patent No. 5,657,390 by Elgamal
EX-1011	R. Zuccherato, Methods for Avoiding the “Small-Subgroup” Attacks on the Diffie-Hellman Key Agreement Method for S/MIMEInternet, Engineering Task Force (IETF), Network Working Group Request for Comments No. 2785 (Mar. 2000)
EX-1012	Prosecution History of the ’838 patent
EX-1013	Declaration of Paul D. Martin, Ph.D.
EX-1014	Docket Control Order, <i>Daedalus Prime LLC v. Mediatek Inc.</i> , No. 2-24-cv-00235, Dkt. 33 (E.D. Tex. Oct. 23, 2024).
EX-1015	Complaint, <i>Daedalus Prime LLC v. Mediatek Inc.</i> , No. 2-24-cv-00235, Dkt. 1 (E.D. Tex. Apr. 8, 2024).
EX-1016	Plaintiff’s Infringement Contentions, <i>Daedalus Prime LLC v. Mediatek Inc.</i> , No. 2-24-cv-00235.
EX-1017	Declaration of Raymon Cruitt
EX-1018	T. Dierks, The TLS Protocol Version 1.0, Internet Engineering Task Force (IETF), Network Working Group Request for Comments No. 2246 (January 1999) (“TLS-1.0”)
EX-1019	Curriculum Vitae of Dr. Paul Martin Ph.D.
EX-1020	U.S. Patent No. 6,643,774 to McGarvey

EX-1021	U.S. Patent No. 7,039,946 to Binding
EX-1022	U.S. Patent Publication No. 2010/0205443 A1 to Yao
EX-1023	U.S. Patent Publication No. 2011/0231653 A1 to Bollay
EX-1024	U.S. Patent Publication No. 2010/0042824 A1 to Lee
EX-1025	U.S. Patent No. 6,996,547 to Tugenerg

LIST OF CLAIMS

[1p]	A system-on-a-chip apparatus comprising:
[1a-1]	a system-on-a-chip comprising a security engine that is separate from a processor core of the system-on-a-chip and
[1a-2]	has a secure memory accessible only by the security engine,
[1a-3]	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,
[1b]	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;
[1c]	perform a cryptographic key exchange with the remote server;
[1d]	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;
[1e]	encrypt the symmetric session key based on the security key; and
[1f]	store the encrypted session key in the secure memory,
[1g]	the system-on-a-chip to establish the secure communication session with the remote server over the network using the session key.
[2]	The system-on-a-chip apparatus of claim 1, wherein the security engine is further to receive a server nonce from the remote server in a response message to the request for the secure communication session, the response message including a server nonce and store the server nonce in the secure memory.
[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.
[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.
[5a]	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; and
[5b]	calculate a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server.
[6a]	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
[6b]	sign the public Diffie-Hellman client key using the security key of the security engine, and send the signed public Diffie-Hellman key to the remote server.
[7]	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 generate the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key.
[8]	The system-on-a-chip apparatus of claim 1, wherein the security engine is further to receive Diffie-Hellman global values and a public Diffie-Hellman server key from the remote server and store at least one of the Diffie-Hellman global values and the public Diffie-Hellman server key in the secure memory.
[9p]	A method comprising:
[9a]	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;

[9b]	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;
[9c]	performing a cryptographic key exchange, using the security engine of the system-on-a-chip, with the remote server;
[9d]	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;
[9e]	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;
[9f]	storing the encrypted session key in the secure memory of the security engine of the system-on-a-chip; and
[9g]	establishing, using the client device, the secure communication session with the remote server using the session key.
[10a]	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
[10b]	further comprising: receiving a ServerHello message from the remote server, the ServerHello message including a server nonce; and storing the server nonce in the secure memory of the security engine of the system-on-a-chip.
[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.
[12]	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 storing the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip.
[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.
[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.
[15a]	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;
[15b]	signing the public Diffie-Hellman client key using the security key of the security engine of the system-on-a-chip; sending the signed public Diffie-Hellman key to the remote server;
[15c]	receiving a public Diffie-Hellman server key from the remote server; and
[15d]	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.
[16a]	The method of claim 9, further comprising: generating, in the security engine of the system-on-a-chip, a hash code of a preceding message sent from the client to the remote server;
[16b]	signing the hash code using the security key stored in the secure memory of the security engine; and

[16c]	sending a client certificate verification message to the remote server, the client certificate verification message including the signed hash code.
[17p]	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:
[17a]	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;
[17b]	initiate a request for a secure communication session with a remote server over a network, the request including the random nonce;
[17c]	perform a cryptographic key exchange, using the security engine of the system-on-a-chip, with the remote server;
[17d]	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;
[17e]	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;
[17f]	store the session key in the secure memory of the security engine of the system-on-a-chip; and
[17g]	establish the secure communication session with the remote server using the session key.
[18a]	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
[18b]	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 store the server nonce in the secure memory of the security engine of the system-on-a-chip.
[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.
[20]	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 store the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip.
[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.
[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.
[23a]	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, in the security engine of the system-on-a-chip, a public Diffie-Hellman client key and a private Diffie-Hellman client key;
[23b]	sign the public Diffie-Hellman client key using the security key of the security engine of the system-on-a-chip; send the signed public Diffie-Hellman key to the remote server;

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

I. INTRODUCTION

Petitioner requests *inter partes* review (“IPR”) of claims 1-24 of U.S. Patent No. 9,887,838 (“’838 patent”).

II. STANDING

Petitioner is not barred or estopped from requesting IPR and challenging the ’838 patent on the proposed grounds.

III. RELIEF REQUESTED

Petitioner requests institution on the following grounds:

Ground	Prior Art	Basis	Claims
1A	Bugiel, Chou	§103	1-24
1B	Bugiel, Chou, Tkacik	§103	1-24
1C	Bugiel, Chou, TLS-1.2, optionally with Tkacik	§103	1-24
2A	Tkacik, TLS-1.2	§103	1-24
2B	Tkacik, TLS-1.2, Bugiel	§103	1-24

IV. THE ’838 PATENT

A. Overview

The ’838 patent discloses establishing secure network communications. EX-1001, 1:5-32, 3:24-59, Fig. 1. A client includes a system-on-a-chip (“SOC”) with a security engine 110 including security key 150 “provisioned during the manufacturing of the SOC 112.” *Id.*, 3:24-4:16, 4:16-34.

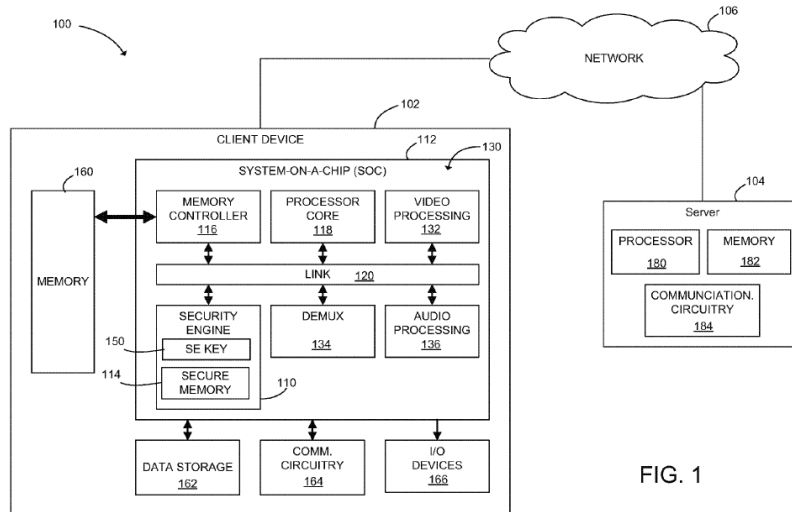


FIG. 1

Id., Fig. 1.

The '838 patent discloses communicating with a server using Secure Sockets Layer ("SSL") and related Transport Layer Security ("TLS") protocols. *Id.*, 1:7-32. Figure 3 matches the known SSL/TLS handshake, including a ClientHello message initiating the key exchange, which includes a random nonce, generating a pre-master key, and creating a symmetric session key from the pre-master key. *Id.*, 6:20-9:49.

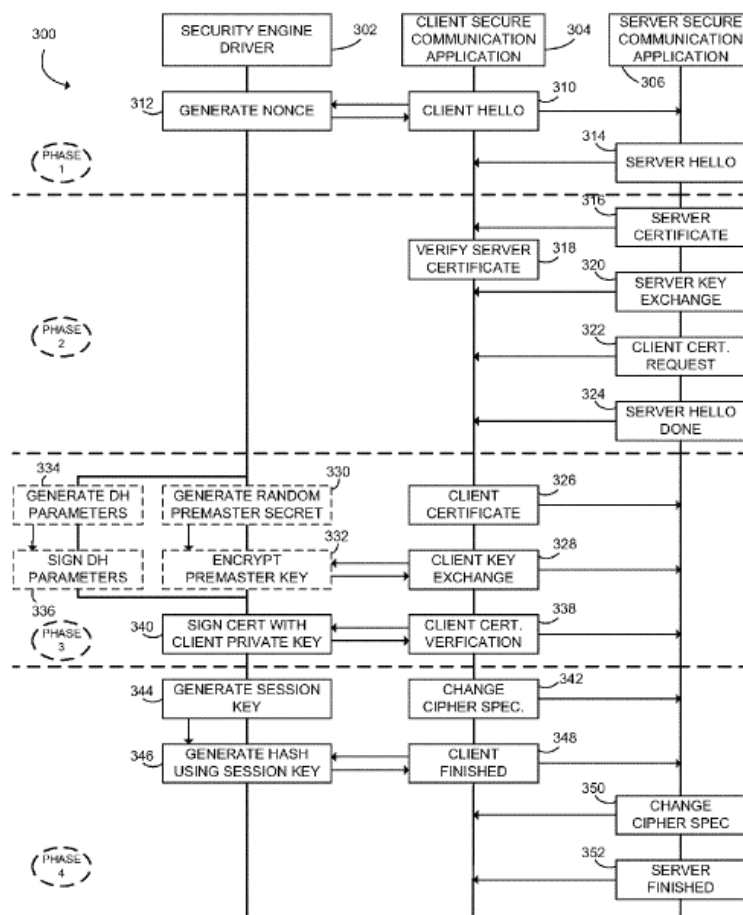


FIG. 3

Id., Fig. 3.

SSL/TLS predates the '838 patent and widely used for secure communications. *See* EX-1006, 6. The patent acknowledges this. EX-1001, 1:5-24.

B. Person of Ordinary Skill in the Art (“POSITA”)

A POSITA at the time of the invention would have had at least a bachelor’s degree in electrical engineering, computer science, computer engineer, or a similar field, and approximately two years of experience in a field related to cryptography and secure communications between network devices. EX-1013 ¶¶66-68. Work

experience can substitute for formal education and additional formal education can substitute for work experience. *Id.*

V. CLAIM CONSTRUCTION

Given the '838 patent's similarities to the prior art, no terms require construction here.

VI. PRIOR ART

A. Bugiel

Bugiel discloses secure authentication between mobile devices and servers using a “hardware-supported secure trusted execution environment (TrEE).” Bugiel, 219-220. Bugiel also uses SSL/TLS. *See id.*, 223, 225-226.

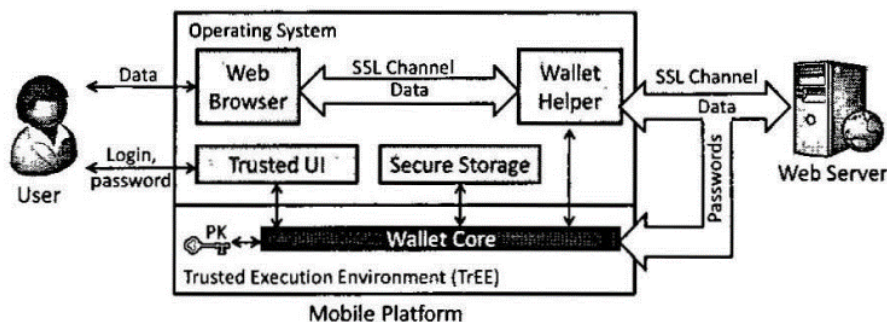


Fig. 1. TruWalletM Architecture

Id., Fig. 1.

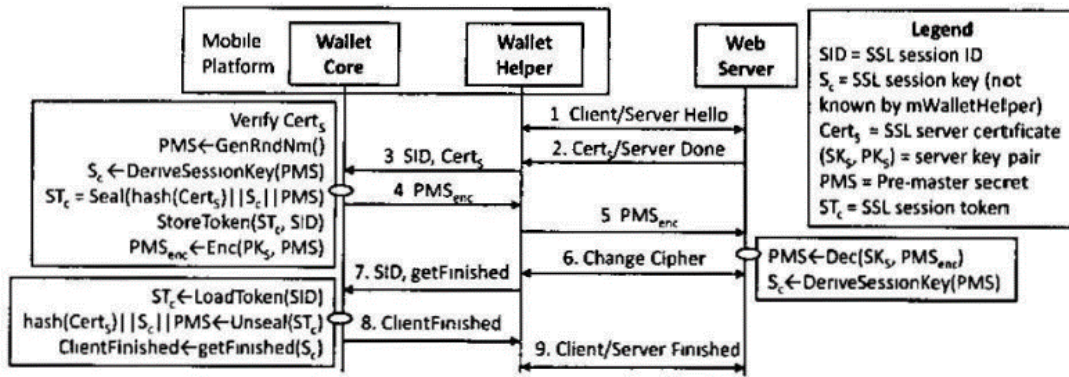


Fig. 2. SSL/TLS handshake

Id., Fig. 2.

Bugiel was publicly available and accessible before December 15, 2011. EX-1017 ¶¶26-31. Bugiel was presented at the INTRUST 2010 conference December 13-15, 2010, published in the proceedings of that conference at least by November 17, 2011, and cited by other publications as early as June 2011. *Id.*

B. Chou

Chou discloses a hardware security module (HSM) 20 that provides a secure boundary for a secure key. Chou, Abstract. A security processor “unwrap[s] and authenticate[s] 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.” *Id.*, Abstract, ¶¶18-23, 30-35.

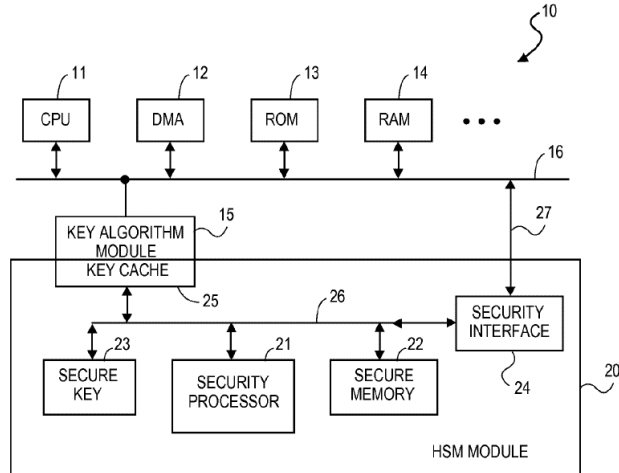


FIG. 1

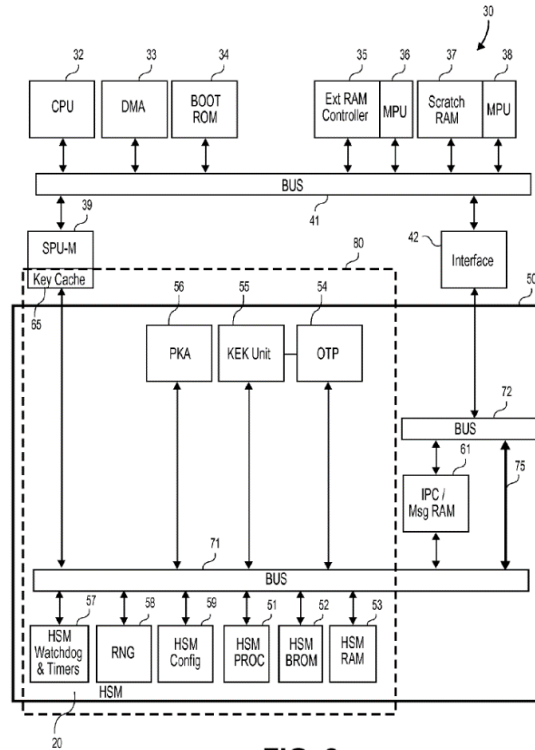


FIG. 3

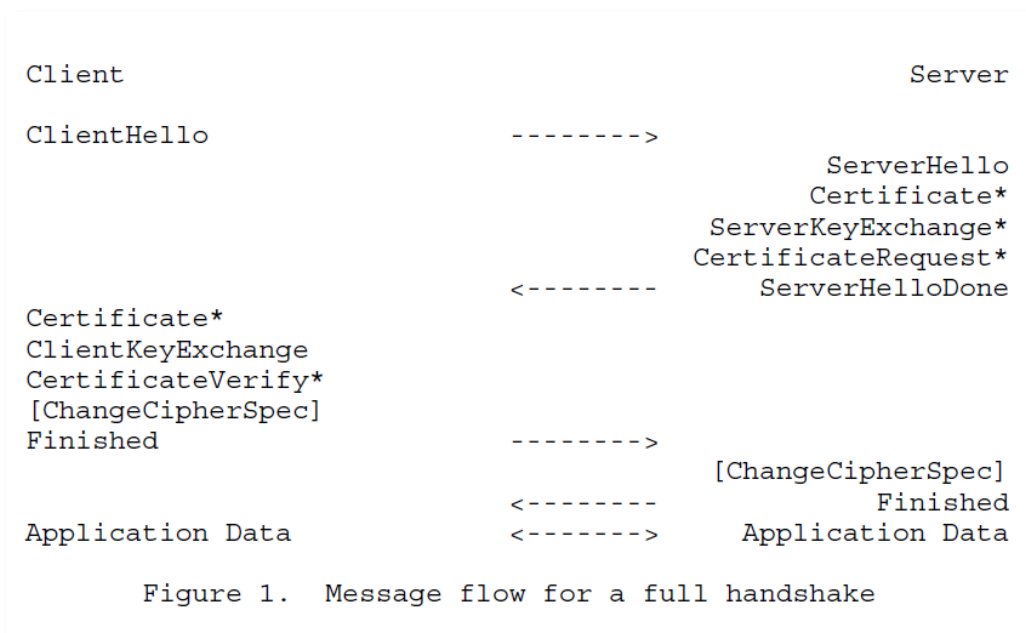
Id., Figs. 1, 3.

C. TLS-1.2

TLS-1.2 discloses the Transport Layer Security (“TLS”) Protocol Version 1.2.

TLS-1.2, 1. TLS-1.2 discloses a protocol “provid[ing] communications security over

the Internet.” *Id.* Two communicating devices negotiate a shared secret and authenticate using “asymmetric, or public key, cryptography” and the handshake below. *Id.*, 4-5.



Id., Fig. 1.

TLS-1.2 was publicly available and accessible before December 15, 2011. EX-1017 ¶¶32-39. It published in August 2008, as indicated on its face, was stored in a repository as early as March 2011, and was cited by other publications as early as November 2009. *Id.*

D. Tkacik

Tkacik discloses a secure processing section that provides a secure execution environment (“SEE”) and stores a plaintext master key 160. Tkacik, Abstract, ¶¶20-26, 44-46. Long-lived key register 60 is programmed with plaintext master key

160 “during the manufacturing process of secure processing system 12 and cannot thereafter be altered.” *Id.* ¶46.

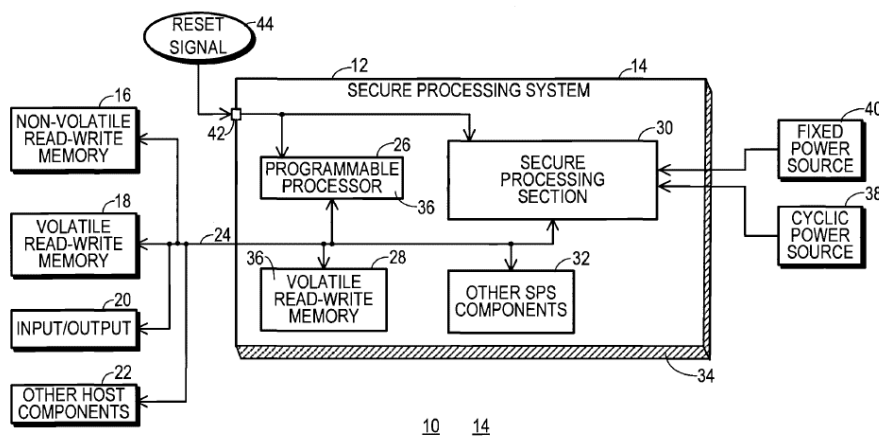


FIG. 1

Id., Fig. 1.

VII. Grounds 1A, 1B, and 1C Render Obvious Claims 1-24

A. Motivation to Combine Bugiel, Chou and, optionally, TLS-1.2

A POSITA would have combined Bugiel and Chou (**Grounds 1A and 1B**) and Bugiel and Chou with TLS-1.2 (**Ground 1C**).

It would have been obvious to implement Bugiel’s system using Chou’s HSM 20. A POSITA would have understood both Chou and Bugiel relate to secure communication using cryptographic keys for encryption and decryption. Bugiel uses SSL/TLS “[f]or secure communication between the device and the remote web server” and explicitly cites TLS-1.2. *See* Bugiel, 225; *see also id.*, 224-235. Chou similarly describes secure communications. Chou ¶¶6, 17. Chou’s hardware may be

deployed in mobile phones, where “secure key operations” use the “secure zone boundary [that] is defined by components that make up a hardware security module (HSM).” *Id.* ¶¶17, 66-69; EX-1013 ¶¶73-76.

SSL/TLS—one of the most popular secure communication protocols—was “built in to nearly all browsers and Web servers.” EX-1006, 6. Given Bugiel’s express teachings and the ubiquity of SSL/TLS, a POSITA would have been motivated to look to the SSL/TLS protocol for implementation details of the TLS Handshake Protocol for generating cryptographic keys based on cryptographic key exchange. EX-1013 ¶77.

A POSITA also would have understood Bugiel’s and TLS-1.2’s SSL/TLS processes require hardware components such as a client (Bugiel, 221); secure storage and isolated secure execution environment (*id.*, 222, 232); and hardware for (i) generating random nonces for TLS ClientHello/ServerHello messages to initiate cryptographic key exchange (*See id.* 222-226; TLS-1.2, 34), implementing Diffie-Hellman and/or RSA key generation for generating TLS session keys (Bugiel, 222-226; TLS-1.2, 65, hashing, and signing algorithms for certifying client certificates (Bugiel, 222-226; TLS-1.2, 46, 52), symmetric encryption using generated encryption keys (Bugiel, 222-226; TLS-1.2), and encrypting a session key with a key never available outside TrEE (Bugiel, 230-231). EX-1013 ¶78.

A POSITA would have known Chou provides exemplary hardware to implement these functions, and would have combined Chou with Bugiel and/or TLS-1.2. Chou's SoC 91 includes components designed to perform the above functions required by Bugiel and TLS-1.2. EX-1013 ¶79; Chou ¶¶18, 40, 42-55, 63, 66, Fig. 5. Because Chou's hardware satisfies the requirements of Bugiel and TLS-1.2, a POSITA would have found it obvious to implement Bugiel's techniques, including TLS-1.2, using Chou's SoC apparatus. Specifically, Chou's HSM 20 and SPU-M 39/Key Algorithm Module 15 would have generated and used keys based on the SSL handshake protocol. *See* Chou ¶¶42-55, 18, 66; EX-1013 ¶80.

The Bugiel-Chou, and Bugiel-TLS-1.2-Chou combinations would have been nothing more than combining known prior art elements according to understood methods to perform the well-known TLS handshake process using a secure hardware environment. EX-1013 ¶81. This would have yielded predictable results in establishing secure, encrypted communication between a client mobile device and a server. *Id.* A POSITA would have expected success in the result, as Bugiel contemplates using TLS-1.2 and secure hardware to store protected keys and to encrypt/decrypt data, as taught by Chou. *Id.*

The combination would also have been nothing more than a known technique (e.g., Bugiel's and TLS-1.2's techniques for secure communications) to improve similar devices (e.g., Chou's flexible security hardware) in the same way (e.g., using

Chou's hardware to implement Bugiel's and TLS-1.2's techniques) to establish secure communications between a client mobile device and a server. *Id.* ¶82. Indeed, the prior art suggests the combination because Bugiel's teaches using secure hardware for mobile devices to protect the execution of critical code from tampering while engaging in TLS-1.2-secured communication that Chou's hardware provides. *Id.*

A POSITA would have looked to TLS-1.2 given Bugiel's use of this protocol. *Id.* ¶83. Given the ubiquity of SSL/TLS, a POSITA would have been motivated to review the SSL/TLS, e.g., to implement **Grounds 1A-1C**. TLS-1.2 was the current version of the TLS/SSL standard at the priority date, and it had been so for more than three years. *Id.* Accordingly, the TLS handshake process described in TLS-1.2 would have been not only mature enough for a POSITA to implement in Chou's system-on-a-chip architecture, but also the most up-to-date version of one of the most popular secure communication standards. *Id.*

B. Motivation to Combine Bugiel, Chou, and Tkacik

A POSITA would have found it obvious to combine Bugiel and Chou with Tkacik (**Ground 1B**) to implement Chou's secure memory with a security key encoded during a manufacturing process of the SOC. EX-1013 ¶¶84-85. Chou discloses programming a device unique key/root key in one-time programmable ("OTP") memory 54, which a POSITA would have found obvious to program during

manufacturing, just as Chou contemplates for connection 75 of OTP 54. *See* Chou ¶¶64, 38, 40; *see* [1a-3]. It also would have been obvious to implement OTP 54 on Chou's SOC as memory with a security key set during manufacturing, as disclosed by Tkacik. Tkacik ¶46.

Tkacik's non-volatile, read-only register is an OTP memory, just like Chou's OTP 54. OTP memory utilizes "fuses" to permanently and irreversibly program the memory. *See* EX-1009, 240-241; *see also id.*, 246-247. A POSITA would have combined Tkacik with Bugiel-Chou by programming Chou's unique key/root on OTP 54 using Tkacik's manufacturing process. A POSITA would have been motivated to do so to ensure Chou's device is programmed for use when manufactured, thereby providing a ready-to-use device for users. EX-1013 ¶¶86-87.

This would have been nothing more than combining known methods of programming memory and applying this process during manufacturing, as taught and suggested by Chou and Tkacik. Chou ¶64; Tkacik ¶46. Other prior art confirms this was well-known. EX-1013 ¶88. Applying this known manufacturing process to Chou would have been merely applying known methods of programming OTP memory, according to understood principles, with predictable results. A POSITA would have had an expectation of success in doing so. *Id.* ¶¶89-90.

If it is disputed that Tkacik's non-volatile, read-only register is an OTP memory like Chou's OTP, it would have been obvious to implement Chou's OTP

using Tkacik’s non-volatile, read-only memory. Doing so would have been nothing more than substitution one known type of permanent memory for another, with predictable results. *Id.* ¶¶91.

C. Claim 1

1. [1p] “system-on-a-chip apparatus”

Chou discloses implementing components of device 30 “within a single integrated circuit (IC) chip 91 that incorporates a complete system on the IC chip (System-On-Chip or SOC).” Chou ¶¶65; *see also id.* ¶¶20, 65, Figs. 4, 5. Chou’s SOC 91 satisfies the apparatus itself as Figures 4 and 5 show:

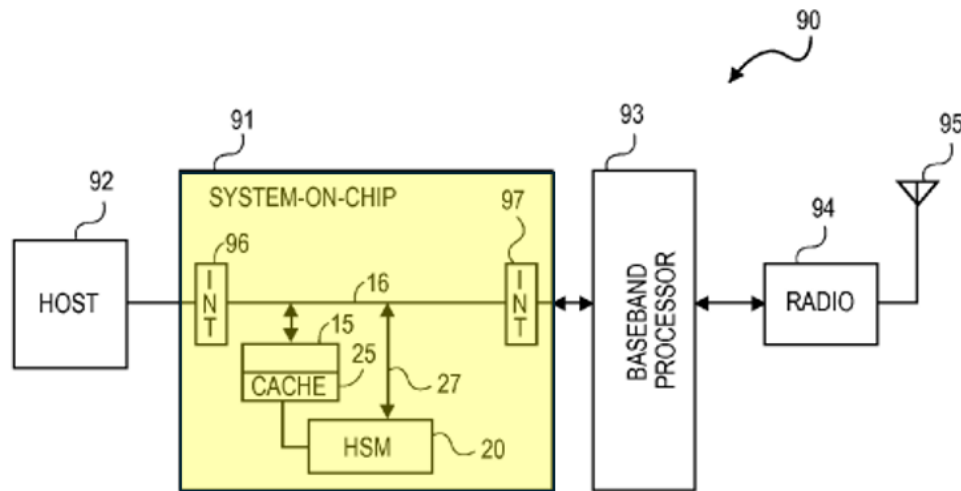


FIG. 4

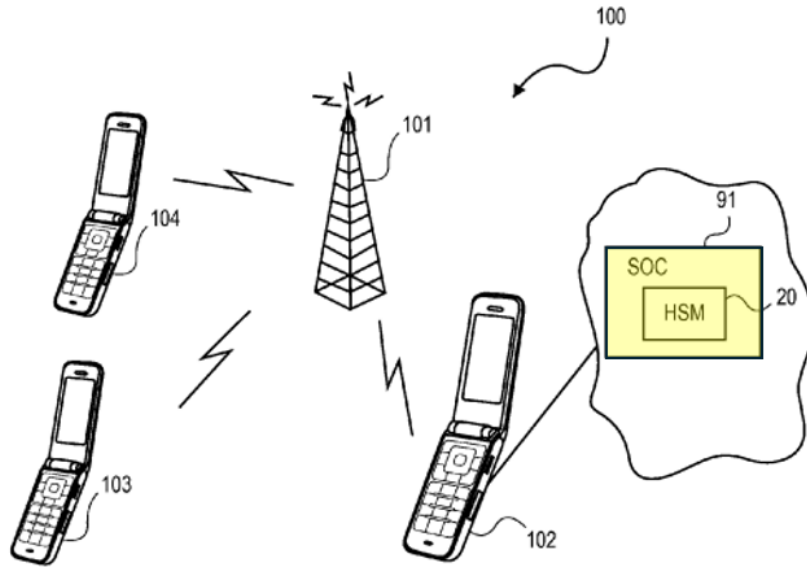


FIG. 5

Chou, Figs. 4, 5 (annotated). This mirrors the '838 patent: “[system-on-a-chip] devices are integrated circuits that incorporate various components, in addition to the processing core, of electronic systems on a single die.” EX-1001, 1:27-32; EX-1013 ¶¶92-93.

Chou’s wireless device 90 also satisfies this feature to the extent the apparatus requires a device *with* an SOC. Chou ¶66; *see also id.* ¶65). Chou’s wireless device 90 or mobile phone 102 having SOC 91 discloses this feature. *Id.* ¶68.

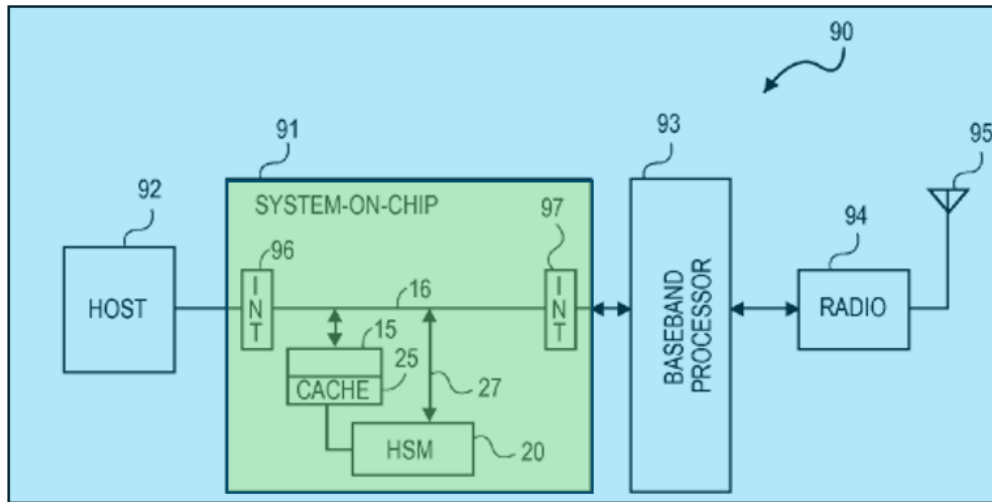


FIG. 4

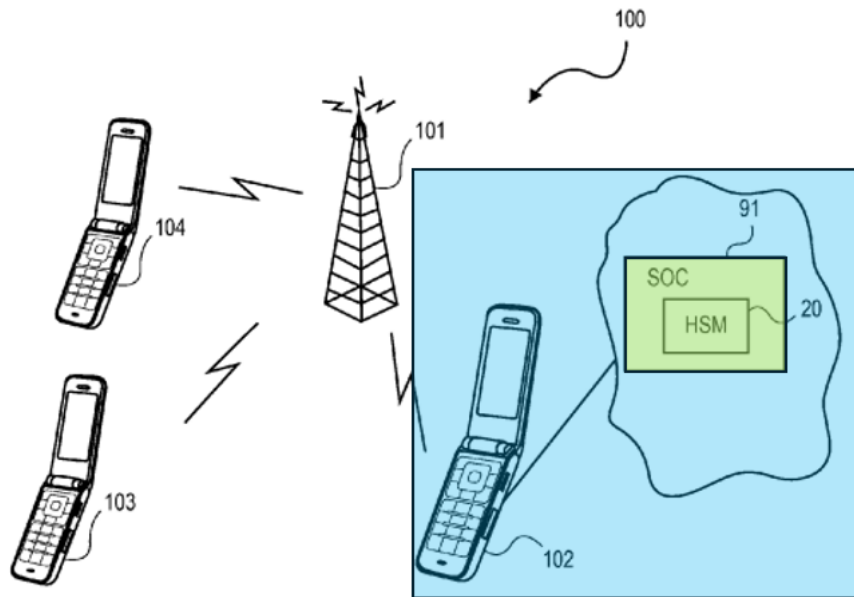


FIG. 5

Id., Figs. 4, 5 (annotated); EX-1013 ¶94.

2. [1a-1] “security engine that is separate from a processor core”

Chou discloses a “security engine” in Figures 1 and 3. Chou’s HSM 20 includes “a security processor 21, secure memory 22, secure key module 23, secure

bus 26 and a security interface 24.” *Id.* ¶21. HSM 20 “set[s] a secure domain, zone or boundary to protect a secure key.” *Id.* ¶18.

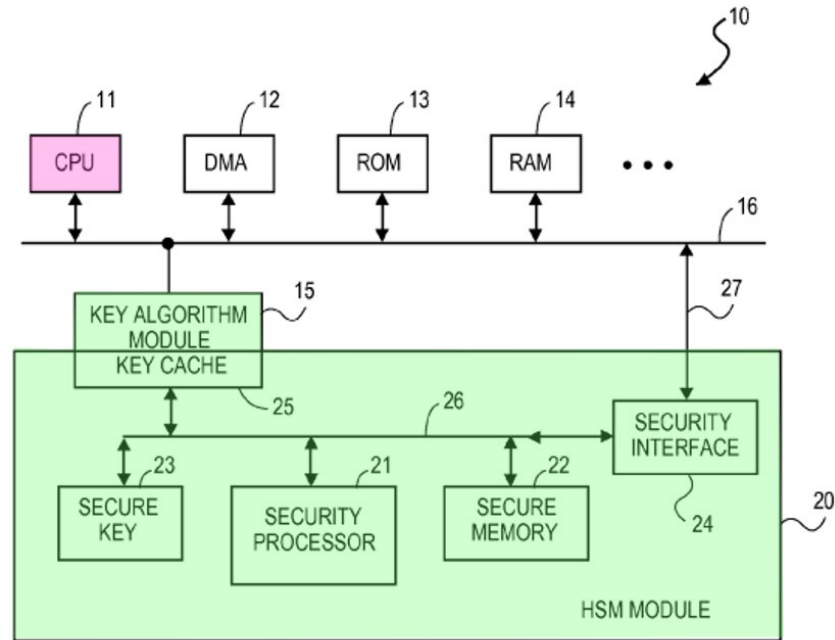


FIG. 1

Id., Fig. 1 (annotated); EX-1013 ¶¶95-97.

Chou discloses “a more detailed HSM,” *Id.* ¶¶30, 14, Fig. 3. including “hardware and software components that encompass a secure domain boundary as shown in FIG. 3.” *Id.* ¶¶ 14, 30, 34, 35, Fig. 3. HSM 20 in Chou’s Figures 1 and 3 describe features of the same HSM, where Figure 3 provides additional details regarding its hardware and operation. *See id.* ¶30, Figs. 1, 3.

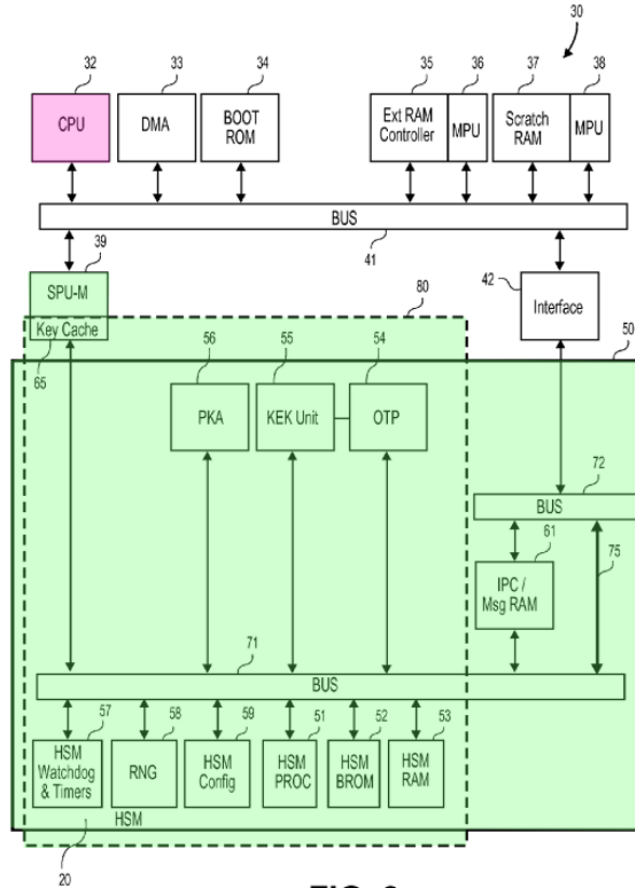


FIG. 3

Id., Fig. 3 (annotated).

Chou’s SPU-M module 39 “is equivalent to key algorithm module 15 of Fig. 1.” Chou ¶54; *see also id.* ¶26. SPU-M module is “a hardware accelerator for symmetric key algorithms.” *Id.* ¶54; *see also id.* ¶¶55-62. Within HSM 20, Chou discloses secure memory for storing keys, including secure key module 23 (Figure 1) and components 54-59 (Figure 3), used for “wrapping and unwrapping of keys.” *Id.* ¶37; *see also id.* ¶¶30, 34-37 (same), ¶¶21-25); EX-1013 ¶¶98-100.

Chou’s HSM 20, including processors 21 and 51 and SPU-M module 39 and key algorithm module 15, discloses the claimed “security engine.” The ’838 patent

discloses security engine 110 may be processing circuitry separate from processor core 118, and “includes the security key 150 and the secure memory 114, which is accessible only by the security engine 110.” EX-1001, 4:17-34. Security engine 110 also “maintains the keys in secrecy within a secure memory 114.” EX-1001, 3:35-41, 4:17-34. EX-1013 ¶101.

Chou’s HSM 20 and key algorithm module 15/SPU-M 39 are separate from the processor core of the SOC. CPUs 11 and 32 “provide[] central processing for device [10 and]30” outside of the secure zone of HSM 20. Chou, ¶¶31-32, 20-21, Figs. 1, 3. CPU 11 “reside[s] *outside* of a boundary established by HSM 20 ... in a *non-secure domain or zone*.” *Id.* ¶20¹, Fig. 1. Chou also discloses that “components in the *non-secure domain or zone*” are coupled to bus 16/41, outside of HSM 20. *Id.* ¶30, Fig. 3; *see also id.* ¶31; EX-1013 ¶¶102-103.

¹ All emphases added unless otherwise indicated.

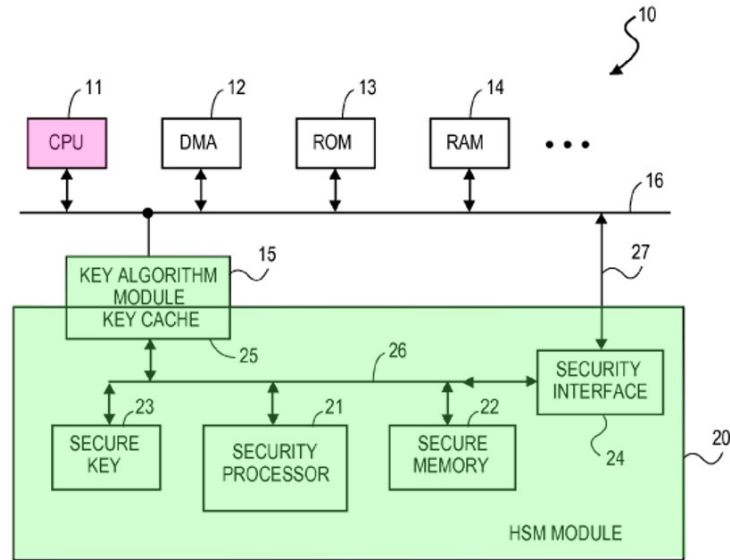


FIG. 1

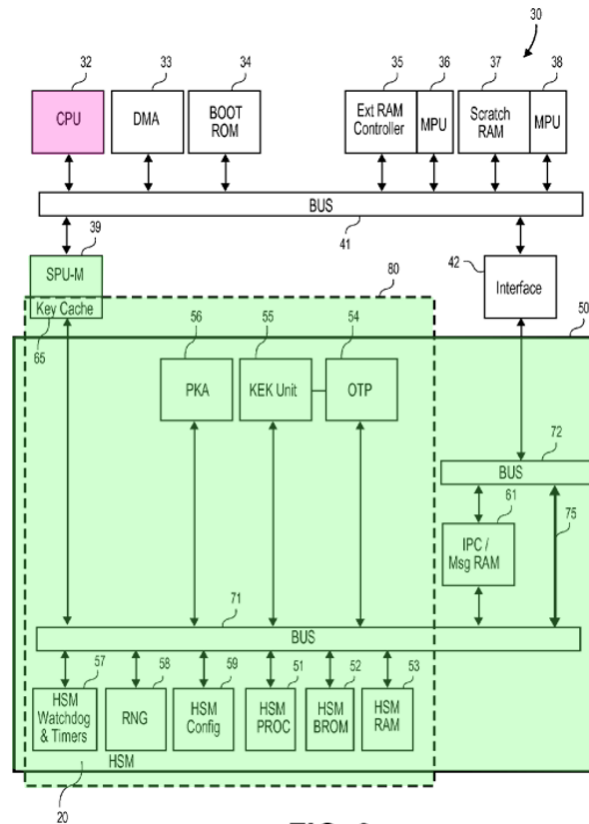


FIG. 3

Chou, Figs. 1, 3 (annotated).

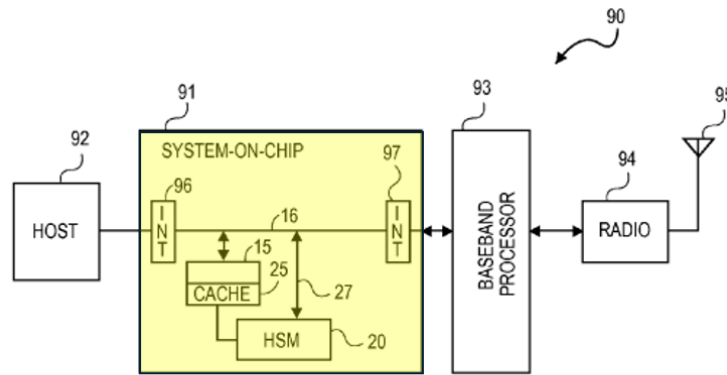


FIG. 4

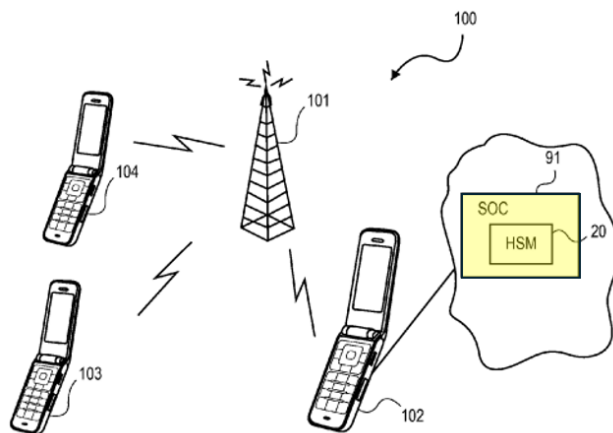


FIG. 5

Id., Figs. 4, 5 (annotated).

3. [1a-2] “a secure memory accessible only by the security engine,”

Chou’s secure memory 22, secure key module 23, key cache 25 in Figure 1, and OTP 54, BROM 52, HSM RAM 53, and key cache 65 in Figure 3, each disclose a memory. The ’838 patent contemplates multiple physical memories, including “machine-readable medium,” memory 160, and memory 182. EX-1001, 2:38-55, 4:41-51, 5:16-33. A POSITA would have also understood a memory may include one

or more memories. Chou's memory is "secured memory" because HSM 20 "set[s] a secure domain, zone or boundary to protect a secure key." Chou ¶¶18, 20. Chou's secure zone contains secure memory 22, secure key module 23, and key cache 25. *Id.* ¶¶21-22, 23-29; EX-1013 ¶¶106-108.

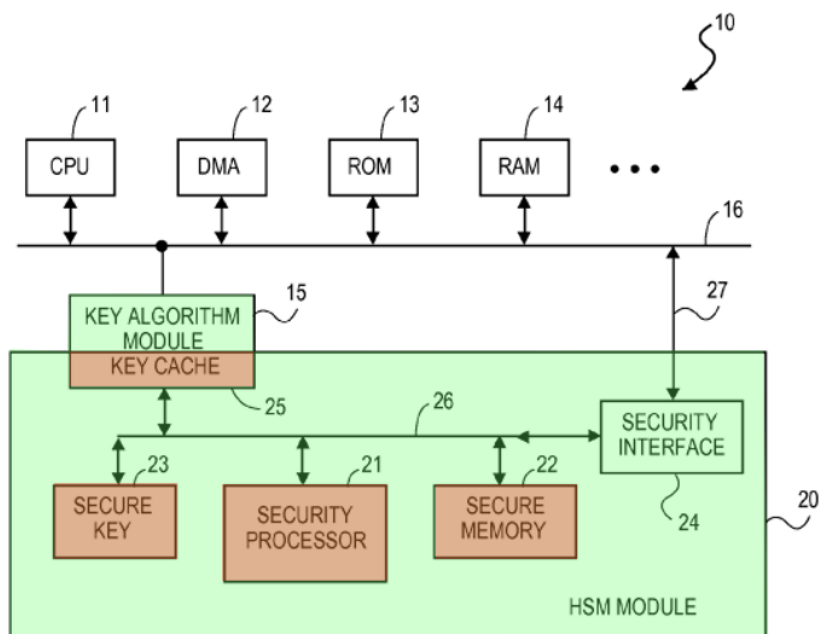


FIG. 1

Chou, Fig. 1 (annotated).

Chou's OTP 54, HSM BROM 52, HSM RAM 53, and key cache 65, are also memories associated with the secure processing inside HSM (e.g., HSM processor 51, PKA 56, KEK 55 and RNG 58) making them "secure memory" in Chou's secure zone. Chou ¶¶30-41. FIG. 3, annotated below, illustrates HSM 20's "hardware and software components that encompass a secure domain boundary." *Id.* ¶34.

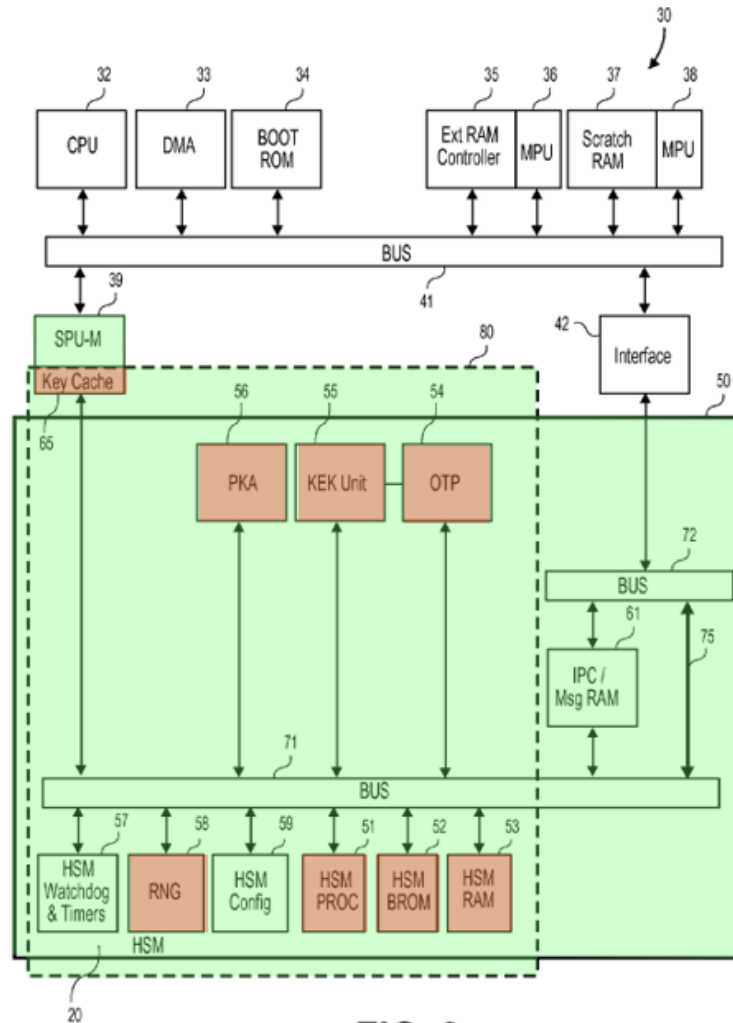


FIG. 3

Id., Fig. 3 (annotated); EX-1013 ¶109.

Chou’s secure memory is “accessible only by the security engine” because Chou discloses that “all accesses to HSM 20 and data transfers into and out of HSM 20 are routed through security interface 24,” and “*[n]o other accesses are permitted* ... , except for the key cache operation.” Chou ¶¶22, 24-27, 34-36, 63, 69; EX-1013 ¶¶110-112.

A POSITA would have combined Bugiel with Chou’s hardware because Bugiel discloses utilizing “secure hardware which provides a Trusted Execution Environment (TrEE).” Bugiel, 222, 232, 224, 234-35. Bugiel therefore discloses or suggests implementing a secure memory accessible only by its TrEE in the secure execution environment, and a POSITA would have found it obvious to implement the secure storage disclosed by Bugiel using Chou’s flexible secure hardware. EX-1013 ¶¶104-105.

4. [1a-3] “a security key ... encoded ... during a manufacturing process,”

Chou discloses security processor 21 encrypting and decrypting data using a “device unique key that is unique to the device,” Chou ¶¶21-22, which is stored in key module 23, *id.* ¶24. Chou also discloses the device unique key or “root key” being stored in OTP 54” *Id.* ¶¶38, 40.

Chou’s device unique key/root key (a “security key”) in Chou’s “secure memory.” *See* Element [1a-2]. This unique key is used to wrap and unwrap other security keys. *Id.* ¶¶22-25, 37, 40, 69; *see also id.* ¶¶7, 23, 25. Chou’s bus 71 communicatively couples PKA module 56 and KEK Unit 55. *See id.* ¶¶35, 39. KEK unit 55 loads the session key generated by PKA module 56 and the device unique key/root key from OTP 54 to encrypt a session key using the device unique key/root key. *See id.* ¶¶39-40; EX-1013 ¶¶113-115.

Chou's unique key/root key is "encoded in the secure memory during a manufacturing process of the system-on-a-chip." Chou's OTP 54 is "used to one-time program a key value or values for device 30," including storing the device unique key. Chou ¶¶34, 38. OTP 54 is non-volatile memory that, once programmed, cannot be changed, EX-1009, 240-241, 246-247, which is also referred to as fuse programmable ROM (or PROM). *Id.*; EX-1013 ¶116.

A POSITA would have understood the unique key/root key would have been encoded either during or after manufacturing. It would have been obvious to program the unique key/root key during manufacturing because Chou contemplates programming OTP 54 during manufacturing. *See* Section VII.A. Chou discloses disabling connection 75 of OTP 54 by "**programming** a non-volatile bit in OTP 54 **during manufacturing**." Chou ¶64 (emphases added). It therefore would have been obvious to program all aspects of OTP 54 during manufacturing, including encoding Chou's device unique key/root. EX-1013 ¶117.

Programming OTP during manufacturing was well-known. EX-1007 ¶¶7, 56; EX-1008, 2:25-40, 4:45-53, 17:5-22. A POSITA therefore would have programmed Chou's OTP 54 during manufactured, just as Chou teaches and suggests. EX-1013 ¶118. The '838 patent does not disclose a particular manufacturing process separate from "provision[ing]" and blown fuses. EX-1001, 4:24-29. The patent therefore contemplates programming its security key by blowing fuses during manufacturing,

which Chou discloses or suggests by programming its OTP 54 “during manufacturing.” *Id.* ¶64; EX-1013 ¶119.

Encoding during manufacturing also would have been obvious in view of Tkacik (**Ground 1B**). Tkacik discloses a plaintext master key 160 stored in a non-volatile, read-only register, where “long-lived key register 60 is desirably programmed with plaintext master key 160 *during the manufacturing process* of secure processing system 12 and cannot thereafter be altered.” *Id.* ¶¶44-46; *see* Section VII.B. Tkacik’s register is one-time programmable memory, just like Chou’s 54. *See* EX-1013. ¶46; EX-1013 ¶¶120-122; EX-1009, 240-241, 246-247. A POSITA would have combined Tkacik with Bugiel and Chou by programming Chou’s OTP 54 using Tkacik’s laser or electronic burnt fuse manufacturing process. *See* Tkacik ¶46. Doing so would have ensured the combined device is programmed for use when manufactured and rather than needing to be later programmed by the user. *See* Section VII.B; EX-1013 ¶123.

Adding Tkacik to the combination would have been no more than combining known elements according to understood principles. EX-1007 ¶¶7, 56; EX-1008, 2:25-40, 4:45-53, 17:5-22. Applying this known teaching would have yielded predictable results and an expectation of success. Indeed, the combination is nothing more than applying known methods of programming one-time programmable

Fig. 2. First, the mobile device exchanges ClientHello and ServerHello messages with the web server. *Id.*, 225-226. This message includes a string of random bytes known as the client random nonce. *See, e.g.*, TLS-1.2, 17, 33-34; EX-1006, 41-43, 42; EX-1013 ¶¶131-132.

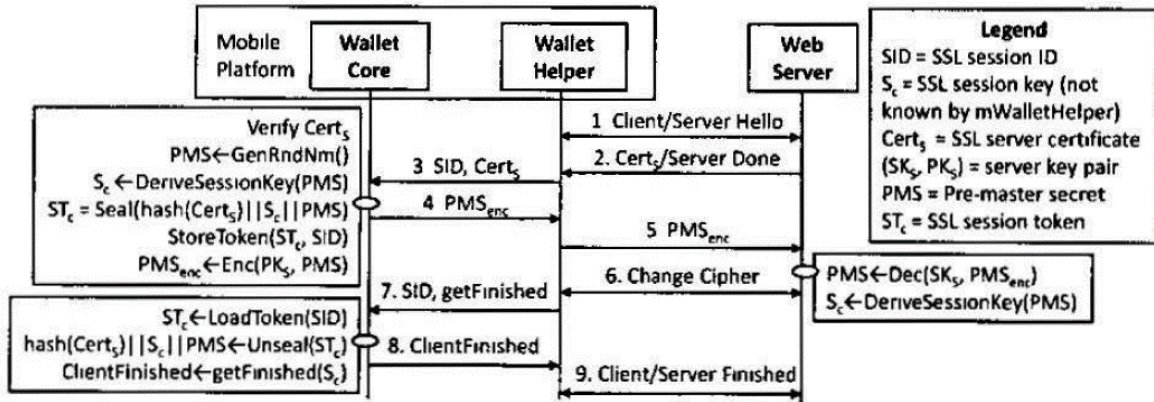


Fig. 2. SSL/TLS handshake

Bugiel, Fig. 2.

Bugiel's ClientHello includes a client-generated random number, which is a nonce for initiating a request for a secure communication session with a remote server over a network using the nonce, as claimed. EX-1013 ¶¶133-135. This is consistent with the '838 patent's use of "any suitable *random number-generating algorithm* to generate the random nonce." EX-1001, 10:14-27, 6:40-50. Bugiel is also consistent with Chou, which uses its hardware for "secure data transfer ... over a network, such as the Internet." Chou ¶6.

In the proposed combination, Bugiel's ClientHello message would utilize a random number generated by, e.g., Chou's random number generator 58 of HSM 20

(e.g., the “security engine,” *see* [1a-1]). Chou, ¶¶34, 42, claim 11, Fig. 3; EX-1013 ¶136. While Chou also discloses its random number could be used for other purposes, Chou ¶42, a POSITA would have found it obvious to utilize random number generator 58 of HSM 20 to generate a random number for the ClientHello message during the TLS handshake in Bugiel. Doing so would have been nothing more than combining known elements in the prior art with expected results. EX-1013 ¶¶136-137 Chou’s use of its random number as a “seed for a cryptographic algorithm” is also consistent with TLS handshake methods, as it was known that using a cryptographically secure pseudorandom number generator seeded with true randomness is a common way to get a high-quality pseudorandom number for use in cryptographic applications. *Id.*

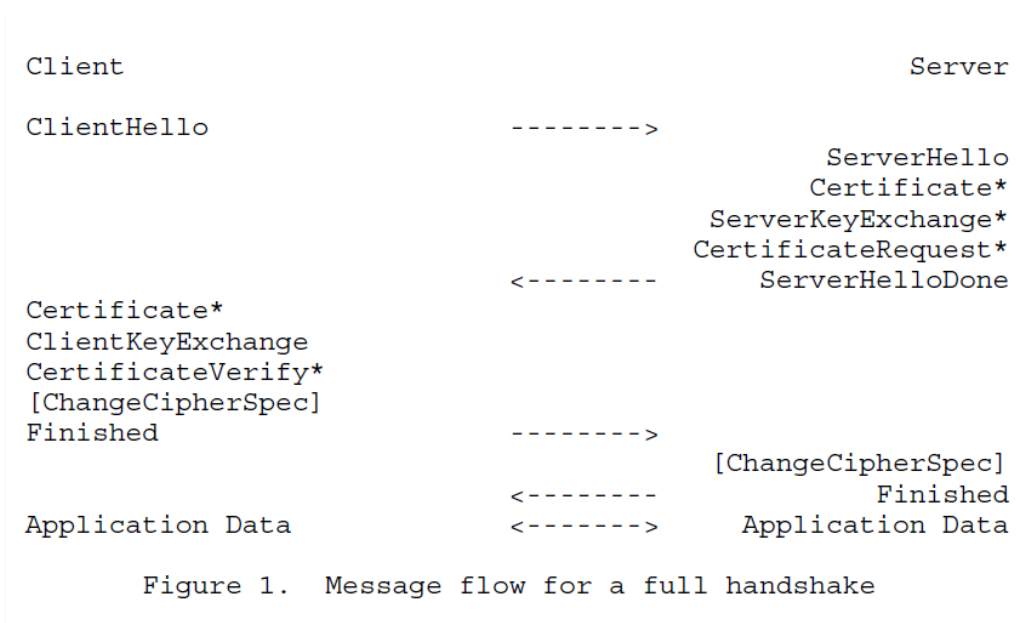
A POSITA thus would have found it obvious to use Chou’s RNG 58 to “generate a random nonce,” as claimed, for use in the ClientHello message, which Bugiel’s TruWalletM method requires to request a secure SSL/TLS communication session with a remote network server. *Id.*

6. [1c] “cryptographic key exchange”

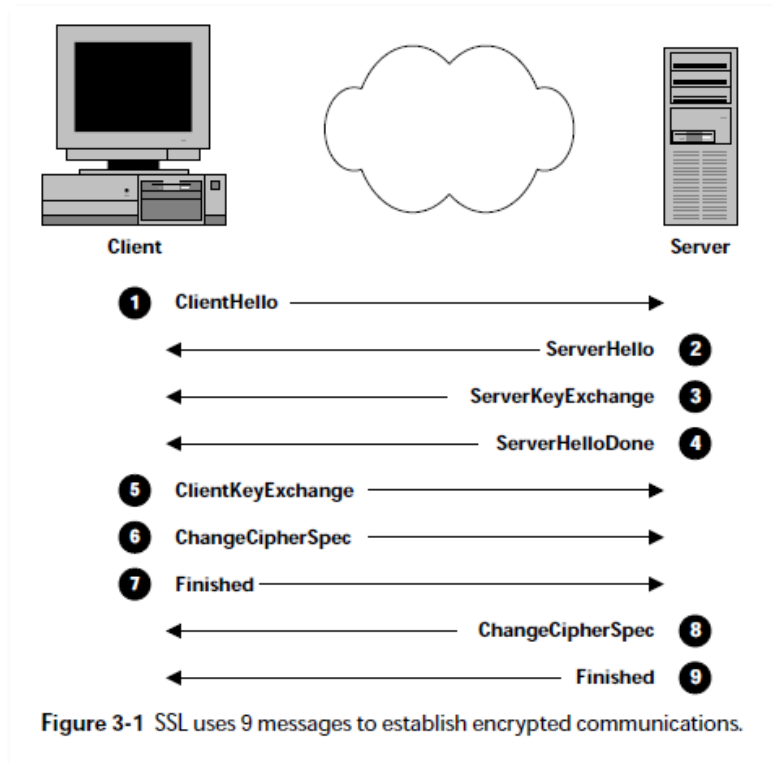
Messages between Bugiel’s device and Web Server “are standard messages of SSL/TLS handshake.” Bugiel, 225; *see also id.*, 226.

As was known and disclosed by Bugiel, the SSL/TLS handshake includes the client/server exchanging Hello messages, followed by a ServerKeyExchange

message, and optionally a public key certificate in a certificate message. TLS-1.2, 47-53; EX-1006, 38-51, 52-59; EX-1013 ¶¶138-139.



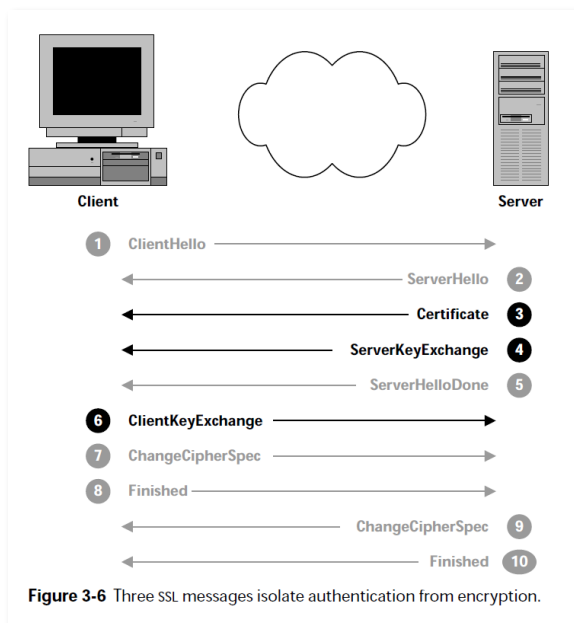
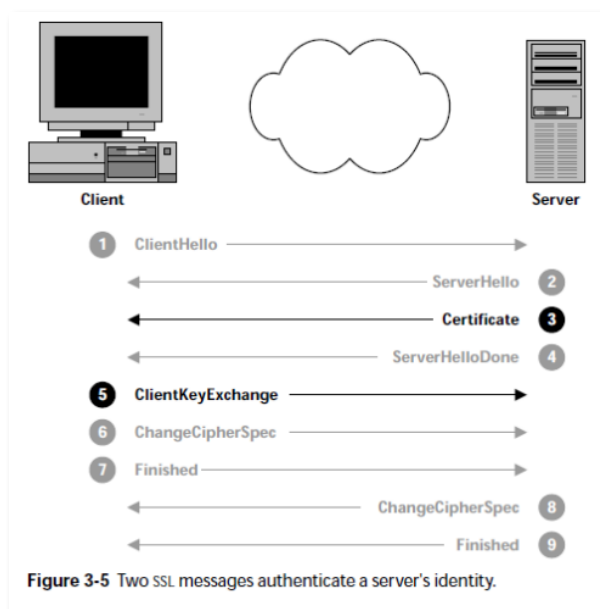
TLS-1.2, Fig. 1.



EX-1006, Fig. 3-1.

Table 3-2 Negotiation of Encrypted Communications	
Step	Action
1	Client sends ClientHello message proposing SSL options.
2	Server responds with ServerHello message selecting the SSL options.
3	Server sends its public key information in ServerKeyExchange message.
4	Server concludes its part of the negotiation with ServerHello-Done message.
5	Client sends session key information (encrypted with server's public key) in ClientKeyExchange message.
6	Client sends ChangeCipherSpec message to activate the negotiated options for all future messages it will send.
7	Client sends Finished message to let the server check the newly activated options.
8	Server sends ChangeCipherSpec message to activate the negotiated options for all future messages it will send.
9	Server sends Finished message to let the client check the newly activated options.

Id., Table 3-2.



Id., Figs. 3-5, 3-6.

The server in a TLS handshake sends a public key in either a ServerKeyExchange message or in the server's Certificate, where the client can use the server public key to generate a symmetric session key. *See* EX-1006, 45, 55-56, 55-56, 59; TLS-1.2, 47-53; EX-1013 ¶140.

Bugiel discloses a cryptographic key exchange in which its server sends a certificate message according to the SSL/TSL handshake, where WalletHelper “acts as an SSL/TLS proxy” when communicating with the web server. Bugiel, 225-226, Fig. 2. After the client and server exchange ClientHello and ServerHello messages, “the server's certificate $Cert_S$ is sent and verified.” *Id.* 225. The certificate $Cert_S$ is an “SSL server certificate,” which, following receipt, the mobile device uses to generate a pre-master secret (“PMS”) and derive a session key in S_C . *Id.*, 225-226.

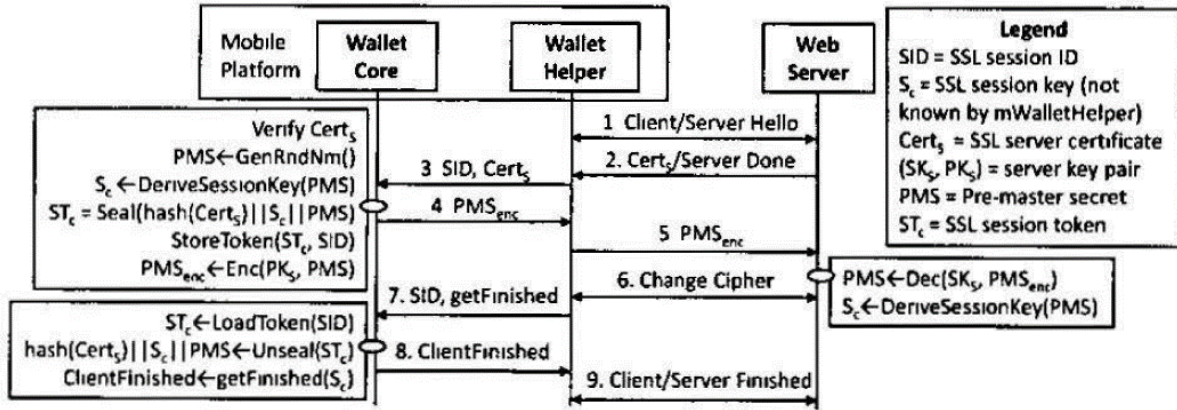


Fig. 2. SSL/TLS handshake

Id., Fig. 2.

A POSITA would have understood that Bugiel sends a public key in its server certificate $Cert_s$ or in a separate ServerKeyExchange message according to the TLS protocol using RSA encryption. EX-1013 ¶¶141-142. Bugiel derives the PMS and session key S_c after receiving and verifying the server certificate $Cert_s$. Bugiel, 225-226, Fig. 2. Thus, according to the known TLS-1.2 specification, Bugiel's server's certificate $Cert_s$ or a separate ServerKeyExchange contains a public key. EX-1013 ¶142; Bugiel, 225; EX-1006, 45, 55-56, 55-56, 59; TLS-1.2, 47-53. Bugiel's server sending certificate $Cert_s$ therefore discloses performing a cryptographic key exchange with [a] remote server. EX-1013 ¶143.

The '838 patent similarly discloses a server sending a message to perform this "exchange," including sending a server certificate in step 316 and a public key in step 320. EX-1001, 7:10-44, Fig. 3. When using RSA key exchange, the server "performs a server RSA *key exchange* in block 320," where "server 104 may

generate a temporary RSA public/private key pair and *send the RSA public key* to the client device 102 in block 320 [(“Server Key Exchange”).]” *Id.*, Fig. 3. The patent therefore contemplates a message from the server with a public key satisfying a “cryptographic key exchange.” EX-1013 ¶143.

To the extent the “cryptographic key exchange” encompasses both the server sending its public key to the client and the client responding with the client’s key information, Bugiel also discloses this in its PMS_{enc} message to the server. Bugiel, 226, Fig. 2. The PMS_{enc} message encloses both the public key (“PK_s”) and the pre-master key (“PMS”). *See id.* The server utilizes these to derive the PMS and the symmetric session key S_C. *See id.* This is consistent with the TLS-1.2 specification. TLS-1.2, 33-37, Fig. 1; EX-1013 ¶¶144; EX-1006, 45-46, 56.

It also would have been obvious to modify Bugiel with TLS-1.2 to send a public key in a separate ServerKeyExchange message, as contemplated by the TLS-1.2 specification (**Ground 1C**). EX-1013 ¶¶145-146. Bugiel incorporates the TLS-1.2 specification into its disclosure and contemplates using it to perform “standard messages of [an] SSL/TLS handshake.” Bugiel, 225. It therefore would have been obvious to send a public key from Bugiel’s server in a separate ServerKeyExchange message instead of including it in Bugiel’s certificate *Cert_s*, consistent with separating server authentication and encryption messages in the TLS-1.2 specification. TLS-1.2, 33-37, Fig. 1; *see* EX-1006, 56-58. This would have

been nothing more than substituting one known feature for another, with predictable results. EX-1013 ¶147. Implementing Diffie-Helman encryption in Bugiel, as taught by TLS-1.2, also would have been obvious. *Id.* ¶148. Instead of using RSA encryption and sending a pre-master secret to the server, as taught by Bugiel, it would have been obvious to perform the TLS handshake process according to a Diffie-Helman key exchange in which the pre-master secret is mutually computed by both the client and server from the Diffie-Helman keys and global values. *See infra* Section VII.H (claim 6); TLS-1.2, 55-58, 61, 65, 74-75, 86, 92-93, 100; EX-1006, 81-84, 40-45, 54-59.

Modifying Bugiel to perform a cryptographic key exchange based on TLS-1.2 would have been nothing more than the simple substitution of one known element for another to obtain predictable results. EX-1013 ¶¶147-149. Sending a public key via a ClientKeyExchange message to perform the RSA or Diffie-Hellman key exchange as described in the TLS-1.2 handshake process as described in the TLS-1.2 handshake process was well known, and a POSITA would have readily substituted one for the other with predictable results and an expectation of success. *Id.*

A POSITA would have implemented Bugiel's WalletHelper and WalletCore using Chou's hardware, *see* Section VII.A, including Chou's security interface 24 or IPC/Msg RAM module, and key algorithm module 15 or equivalent SPU-M 39.

Chou ¶¶54-63, 21-23, 34-37, Figs. 1, 3; EX-1013 ¶150. Both Chou and Bugiel disclose a secure zone in Chou's HSM 20 and Bugiel's TrEE. Chou ¶¶18, 21, 26, 30, 34, Figs. 2, 3; Bugiel, 223-225, Fig. 1. Chou's key algorithm module 15 or equivalent SPU-M 39, like Bugiel's WalletHelper, are outside of the secure zone. Chou ¶¶26, 54. Chou describes the PKA module 56 inside HSM 20 is operable for cryptographic key exchange algorithms, including Diffie-Hellman and RSA key generation. *Id.* ¶¶43-53.

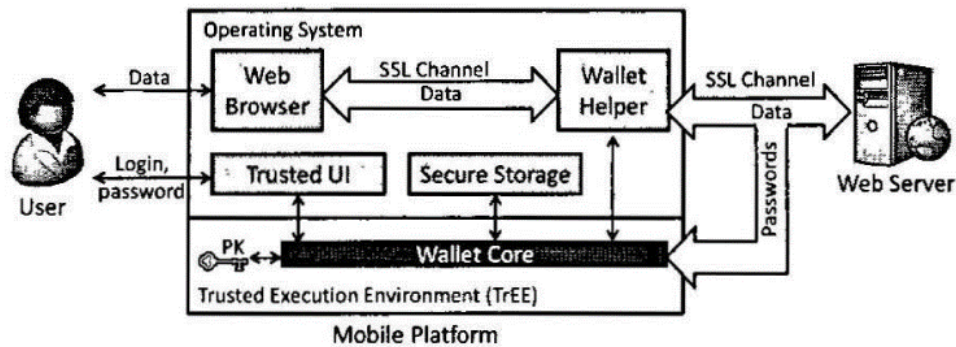


Fig. 1. TruWalletM Architecture

Bugiel, Fig. 1.

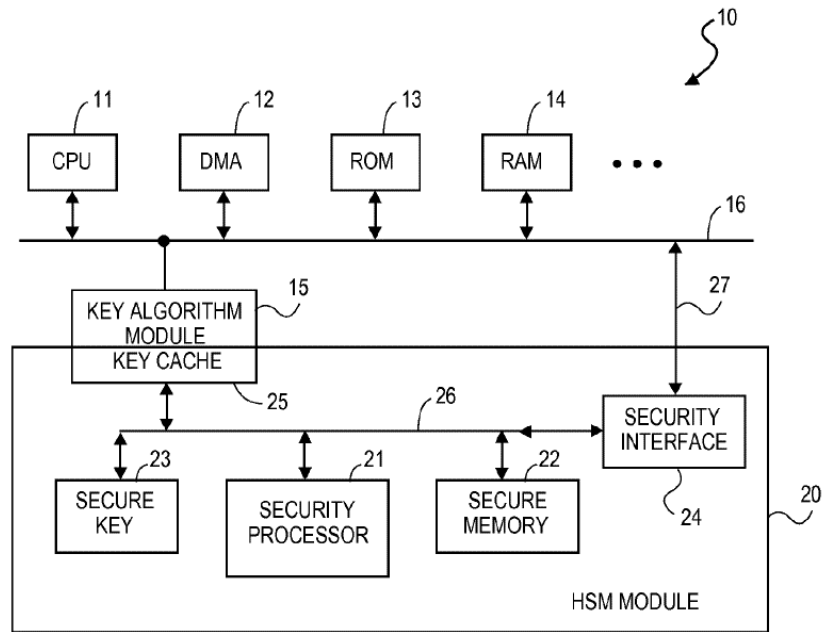


FIG. 1

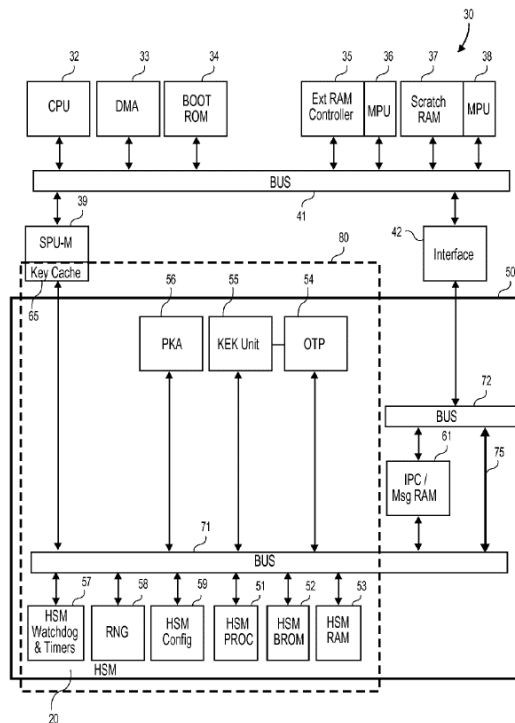


FIG. 3

Chou, Figs. 1, 3.

Bugiel’s WalletHelper acting as a proxy for its WalletCore, implemented in security interface 24 or equivalent IPC/Msg RAM module 61, corresponds with the ’838 patent, which discloses a client secure communication application 304 that is “executed on the client device 102.” EX-1001, 6:20-33, Fig. 3; *see also id.*, 6:34-7:44; EX-1013 ¶151.

7. [1d] “generate a symmetric session key”

Bugiel generates an SSL/TLS session key S_c after receiving and verifying the server certificate $Cert_s$ in the SSL/TLS Handshake. Bugiel, 225. As depicted in Figure 2, Bugiel’s WalletCore generates the session key S_c following verification of the server certificate $Cert_s$:

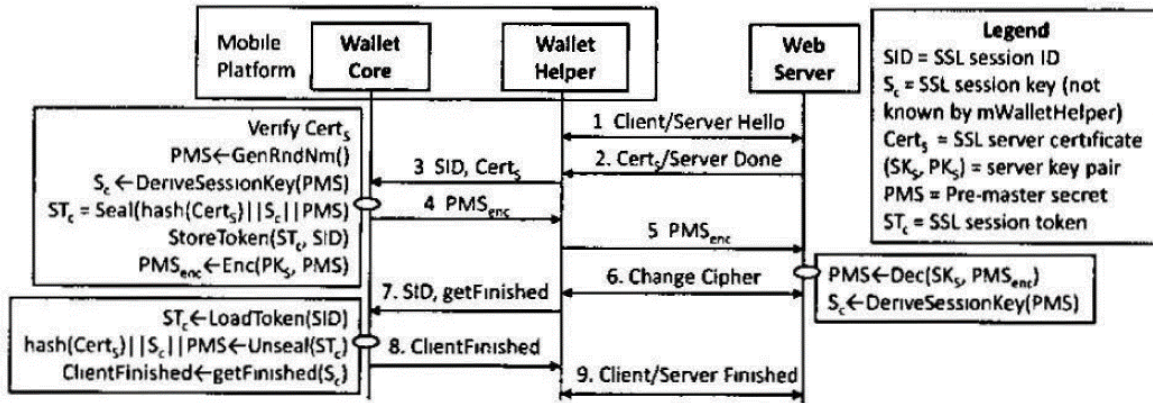


Fig. 2. SSL/TLS handshake

Id., Fig. 2.

As described in [1c], a POSITA would have understood that Bugiel’s server certificate $Cert_s$ includes a public key for generating a symmetric session key, consistent with the TLS-1.2 specification. *See* [1c]. Alternatively, it would have been

obvious for Bugiel’s server to provide the public key for generating a symmetric session key in a separate ClientKeyExchange message, as was also known in the TLS-1.2 specification. *See* [1c]; EX-1013 ¶¶152-153.

Bugiel’s “SSL/TLS session key S_c ” is a symmetric session key generated based on the cryptographic key exchange because Bugiel describes generating the SSL/TLS session key S_c based on the server certificate. *See* [1c]. Bugiel discloses verifying the server certificate $Cert_s$, generating a pre-master key (PMS), and then deriving the session key S_c from the PMS. *See* Bugiel, Fig. 2 (“ $S_c \leftarrow \text{DeriveSessionKey}(\text{PMS})$ ”). Bugiel’s session key S_c is “symmetrical” because the server derives the same session key S_c after Bugiel’s client sends its PMS_{enc} message, consistent with the ClientKeyExchange message in the TLS-1.2 specification. *See* Bugiel, 225-226, Figs. 1, 2; TLS-1.2, 33-37, Fig. 1; *see also id.*, 58-61, 18, 24-26, 64-65, 79-81, 91, 94; EX-1006, 45-46, 56; EX-1013 ¶154.

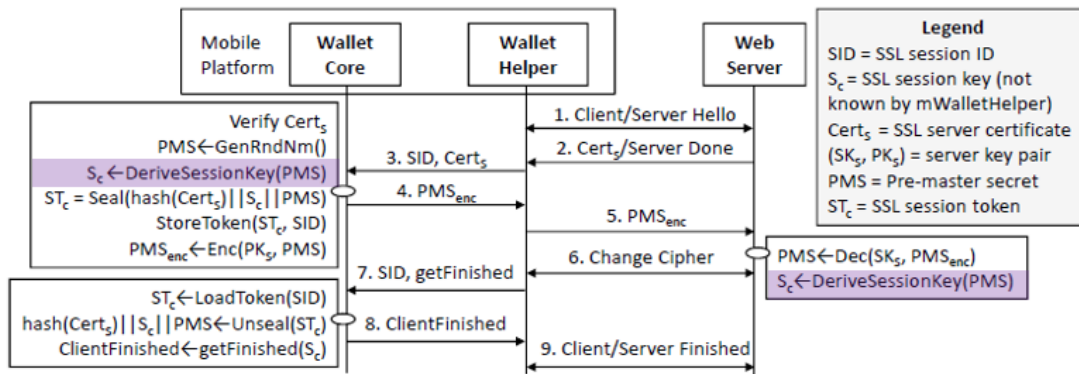


Fig. 2. SSL/TLS handshake

Bugiel, Fig. 2 (annotated).

This is consistent with the '838 patent, which discloses in “block 344, the security engine 110 of the client device 102 generates a session key (i.e., the ‘master’ key),” and “wherein an RSA key exchange is used, the security engine 110 generates the session key as a function of the pre-master key generated in block 330.” EX-1001, 9:7-25 (discussing Fig. 3). The patent discloses the same processes of generating the session key as Bugiel: “[t]o do so, the security engine 110 calculates a hash function of the pre-master key, the client random nonce generated in block 312, and the server random nonce received in block 314.” *Compare id.*, with Bugiel, Fig. 2 (“PMS $\leftarrow$ GenRndNm(),” and “ $S_c \leftarrow$ DeriverSessionKey(PMS)”); *see also* TLS-1.2, 65 (describing RSA for server authentication and key exchange); EX-1013 ¶¶155-156. It also would have been obvious to generate the symmetric session key using a Diffie-Helman exchange based on TLS-1.2. *See* [1c]; *see* Section VII.I (claim 7); EX-1013 ¶158. The master secret (e.g., the symmetric session key) would have been generated based on a ClientKeyExchange message that does not include a pre-master key but one that includes a Diffie-Hellman client public key. TLS-1.2, 57-61; *see* EX-1006, 85-88; *see* [1c].

Bugiel uses “SSL/TLS session key S_c ” to encrypt messages sent to and decrypt messages from the remote server during a secure communication session, as claimed. Bugiel discloses that, after generating the symmetric session key, “[a]ll following messages should be encrypted using S_c .” *Id.*, 225. “All following

messages” includes messages sent and messages received from Bugiel’s remote server. EX-1013 ¶157.

The symmetric session key of Bugiel-Chou would have been implemented by the HSM 20 of Chou. Chou explains that SPU-M 39 “operates as a hardware accelerator for symmetric key algorithm,” Chou ¶¶54-55, and a POSITA would have found it obvious to implement Bugiel’s and/or TLS-1.2’s methods using Chou’s hardware, *See id.* ¶¶54-61; EX-1013 ¶159.

TLS-1.2 also discloses the claimed symmetric session key by generating a master secret based on the cryptographic key exchange in the TLS handshake process as discussed in [1c]. TLS-1.2, 64-65, 16-17, 24-27, 34, 58, 80, 91-94; EX-1006, 85-87, 88-89; EX-1013 ¶160. The ’838 patent states that, “if an RSA key exchange is performed, the pre-master key and the master key (*i.e., the session key*) may be stored in the secure memory 114.” EX-1001, 5:63-66.

TLS-1.2 generates encryption keys (e.g., a “key_block”) for data encryption, based on the master secret for each session. TLS-1.2, 25-26. The key_block is a key with multiple partitions, including a “client_write_key” partition and a “server_write_key” partition. *See id.*, 26. The client uses the client_write_key partition to encrypt data transmitted from the client to a server, and the client uses the server_write_key partition to decrypt data received from the server. *See id.*, 79-81. TLS-1.2 further explains that “TLS secures transmitted application data via

the use of symmetric encryption,” thereby disclosing that the encryption keys are symmetric session keys. *Id.*, 95.

```
key_block = PRF(SecurityParameters.master_secret,  
                "key expansion",  
                SecurityParameters.server_random +  
                SecurityParameters.client_random);
```

Id., 26.

A POSITA would have implemented Bugiel to generate the claimed symmetric session key via TLS-1.2’s “master secret” or “key_block” (**Ground 1C**). EX-1013 ¶¶161-163. The combination would have (1) generated an encryption key, either via the “master secret” or “key_block,” based on cryptographic key exchange according to the TLS handshake protocol; and (2) used the same session key (e.g., the “master secret” or “key_block”) to encrypt/decrypt messages sent/received to/from the remote server. *Id.* This would have been nothing more than substituting one known element in the prior art (e.g., Bugiel’s SSL/TLS session key S_c) for another (e.g., TLS-1.2’s master secret or key_block) to yield predictable results. *Id.* A POSITA would have had a high expectation of success because Bugiel contemplates using the TLS-1.2 specification and it either implements the specification in its SSL/TLS session key S_c , or was compatible with the specification, including its master secret or key_block. The combination would have performed these operations using Chou’s HSM 20 to generate keys based on SSL and Chou’s SPU-M 39 to encrypt/decrypt messages. *Id.*

8. [1e] “encrypt the symmetric session key”

After generating its SSL/TLS session key S_c , the session key S_c is “*sealed*.” Bugiel, 225. Bugiel’s Figure 2 depicts this sealing as “ $ST_C = \text{Seal}(\text{hash}(\text{Cert}_S) \parallel S_c \parallel \text{PMS})$ ”), where ST_C the “SSL session token.” *See id.*, Fig. 2. Bugiel discloses that “the session key S_c is included in the token ST_C , as it is used by WalletCore.” *Id.*, 226.

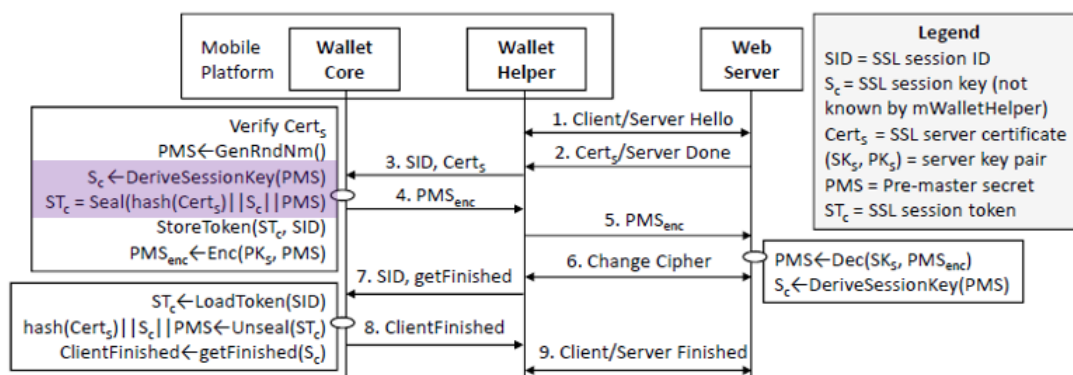


Fig. 2. SSL/TLS handshake

Id., Fig. 2 (annotated); EX-1013 ¶¶164-165.

Sealing in Bugiel means “protecting an object so that only a certain set of entities can access or use it.” *Id.*, 225 n.8, 230-231. Bugiel’s “sealing” and “encrypt[ing] under a key” its session key S_c discloses encrypting a symmetric session key based on a security key. The resulting session token ST_C discloses the encrypted symmetric session key. EX-1013 ¶¶166-167. Indeed, Bugiel discloses encrypting the session key with “a key that is never available outside the TrEE.” *Id.*

Bugiel’s key corresponds to the claimed “security key” and a POSITA would have implemented Bugiel’s key for encrypting its session key S_c via Chou’s device unique key/root key. *See* [1a-3]. Both Chou and Bugiel contemplate secure regions immune from malicious software, and it would have been obvious to implement Bugiel’s similar implementation of a key in its TrEE using Chou’s corresponding unique key/root key encoded on its OTP 54. EX-1013 ¶¶168-170. This would have been nothing more than substituting one known feature in the prior art (i.e., Bugiel’s key in its TrEE) with another (i.e., Chou’s HSM 20 hardware, including its key-encoded OTP 54), with predictable results and an expectation of success. *Id.* Chou’s KEK 55 would encrypt Bugiel’s symmetric session key S_C using Chou’s device unique key/root key stored in OTP 54. *Id.* In the Bugiel-Chou-TLS-1.2 combination, Chou’s KEK 55 would encrypt the master key or key_block (e.g., the symmetric session key) using Chou’s device unique key/root key stored in OTP 54. *Id.*

9. [1f] “store the encrypted session key”

Bugiel discloses storing a session-token ST_C in secure storage. After “the SSL/TLS session key S_c is generated ... and sealed (together with pre-master secret PMS and the hash value $\text{hash}(Cert_s)$),” “[t]he resulting session-token ST_C is ***stored in a secure storage*** and can be loaded and unsealed later using the session identifier SID.” Bugiel, 225-226, Fig. 2. As discussed in [1e], Bugiel’s session-token ST_C

discloses the encrypted symmetric session key. Figure 2, reproduced below, illustrates storing the session-token ST_C . EX-1013 ¶¶171-173.

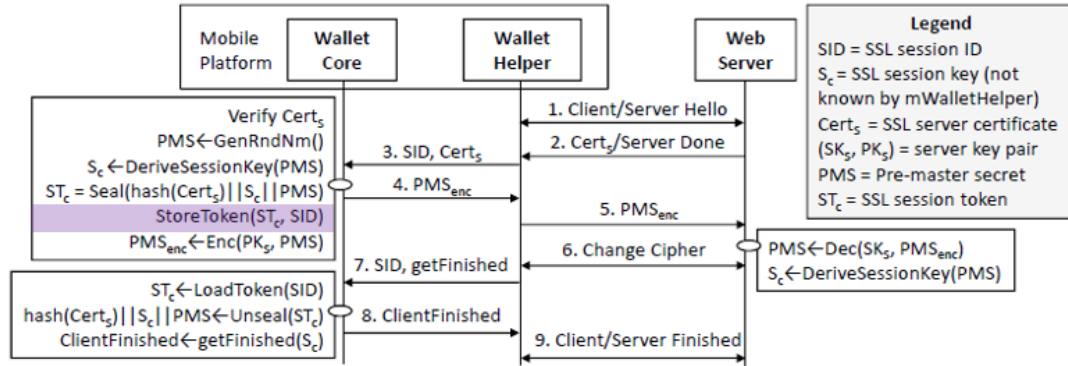


Fig. 2. SSL/TLS handshake

Bugiel, Fig. 2 (annotated).

Bugiel also teaches or suggests [1f] by disclosing storing the encrypted session key ST_C in WalletCore in the TrEE, which “loads the SSL/TLS session token ST_C from the secure storage and unseals it.” Bugiel at 228; *see also id.*, 232, Figs. 4-6; EX-1013 ¶¶173.

A POSITA would have stored Bugiel’s session-token ST_C in one of Chou’s secure memory locations. *See* [1a-2]; EX-1013 ¶¶174. A POSITA would have found it obvious to store the session-token ST_C in Chou’s secure key module 23 or secure memory 22 of Figure 1, or in Chou’s HSM RAM 53 of Figure 3. Chou ¶¶21, 35, 40.

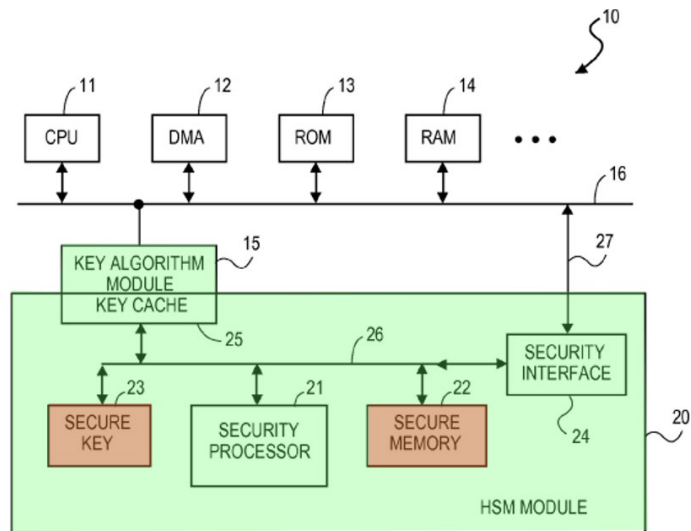


FIG. 1

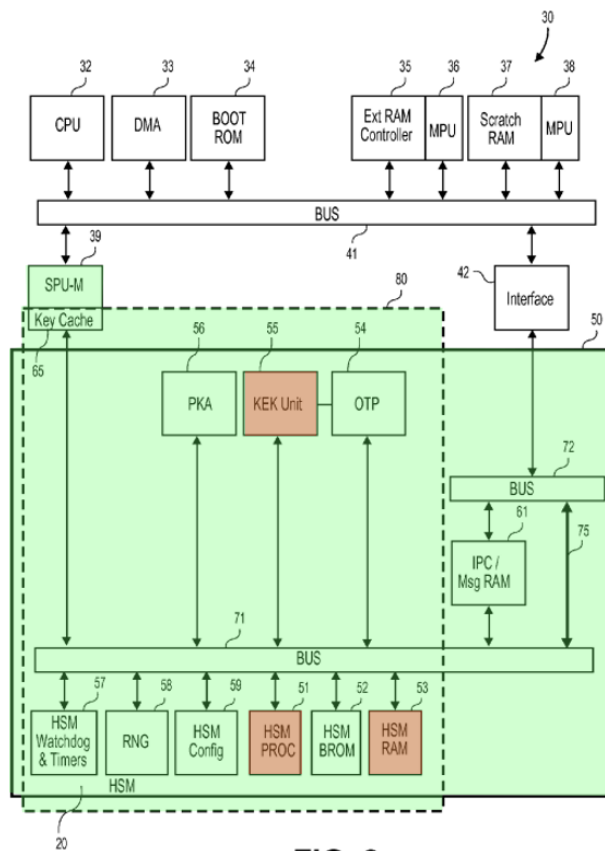


FIG. 3

Id., Figs. 1, 3 (annotated).

10. [1g] “establish the secure communication session ... using the session key.”

Bugiel provides “[a] secure channel between the user and the web server.” Bugiel, 221. Bugiel further discloses “deploy[ing] [a] mobile device to access services provided by remote web servers (over the Internet).” *Id.* After generating symmetric session key S_C , “[a]ll following messages should be encrypted using S_C .” *Id.*, 225; *see also id.*, 226. Bugiel thus teaches establishing a secure communication session with a remote server over a network using a session key. A POSITA would have implemented Bugiel using Chou’s hardware to achieve this. EX-1013 ¶¶175-176.

Alternatively, as discussed in [1d], TLS-1.2 discloses its TLS handshake protocol establishes secure communication between client and server, allowing them “to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data.” TLS-1.2 1, 4; *see also id.*, 6. Application data can be sent only after the handshake process completes. *See id.*, 33-37; EX-1013 ¶177.

TLS-1.2’s establishing secure communication using a session key is consistent with and admitted prior art in the ’838 patent. *See* EX-1001, 1:10-25. **Grounds 1A-1C** utilize Chou’s hardware to implement Bugiel’s and/or TLS-1.2’s communications methods to establish secure communication. Chou designed SOC

91 to manage secure keys used to establish secure communications. *See* Chou ¶¶6, 69; EX-1013 ¶¶178-179.

Consistent Bugiel and TLS-1.2, Chou discloses SOC 91 is “implemented in a mobile phone 102, such as a cell phone ... operat[ing] within a cellular network 100.” Chou ¶68, 65-69, Fig. 5. Chou’s SOC 91 establishes a secure communication session with a remote server over a network using a session key provide by Bugiel and/or TLS-1.2. EX-1013 ¶180. This is consistent with Bugiel, which discloses a mobile device for securely communicating with a web server. *See* [1a-3]. In **Grounds 1A-1C**, Chou’s SOC 91 would implement Bugiel and/or TLS-1.2 to establish secure communications with a remote server, including a web server over the Internet and/or a cellular network.

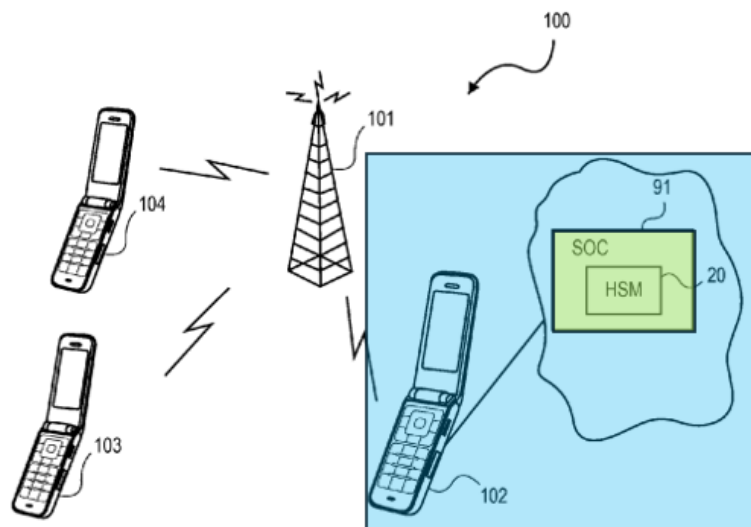
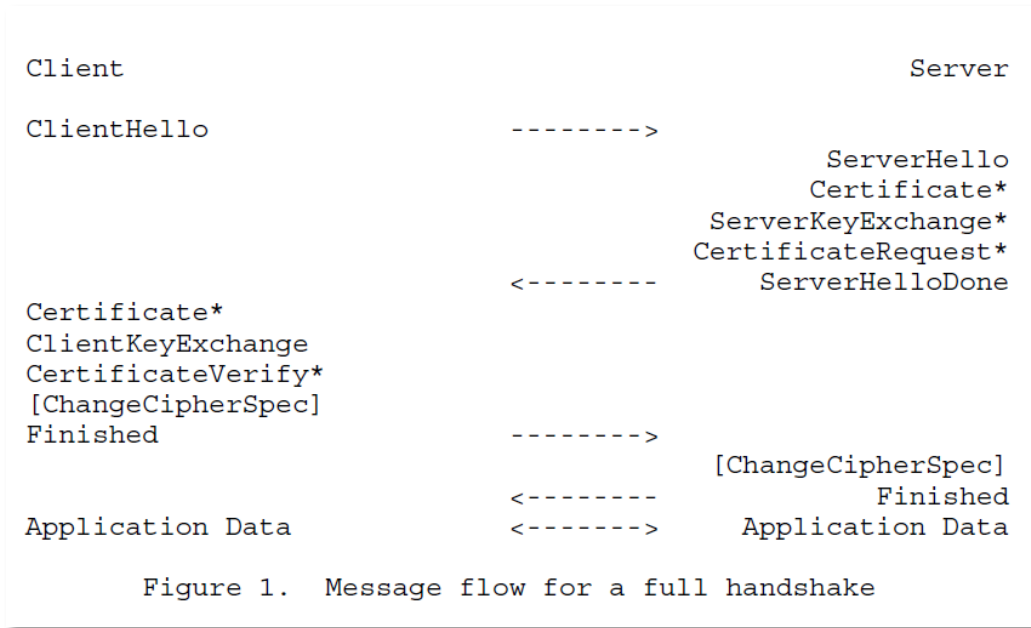


FIG. 5

Id., Fig. 5 (annotated).

D. Claim 2: “receive a server nonce”

The handshake in Bugiel and TLS-1.2 includes a server responding to a ClientHello with a ServerHello, then sending a ServerKeyExchange message, a public key certificate in a certificate, or both. *See* [1c]; TLS-1.2, 47-53; Bugiel, 225, 226, Fig. 2; EX-1006, 38-59, Figs. 3-1, 3-5, 3-6, Tables 3-2, 3-9.



TLS-1.2, Fig. 1.

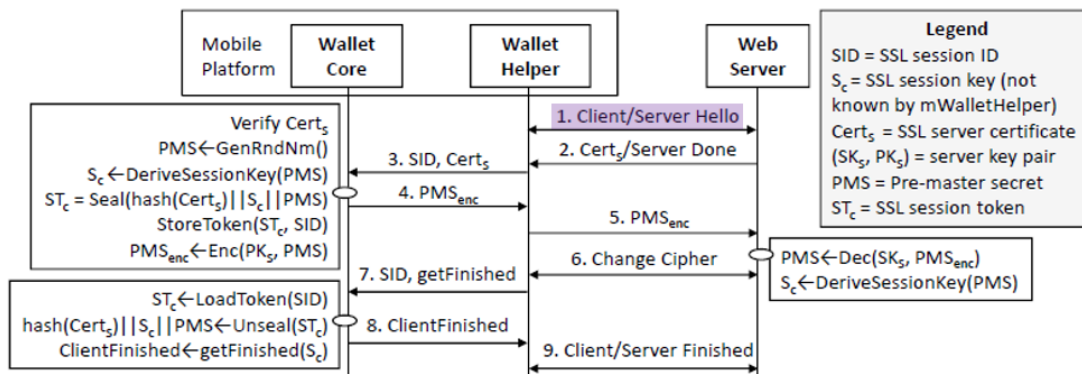


Fig. 2. SSL/TLS handshake

Bugiel, Fig. 2 (annotated).

Bugiel's server, as TLS-1.2 discloses, sends a server nonce in the ServerHello. TLS-1.2, 34, 42-43, 17-18; EX-1006, Table 3-4, 43-44; EX-1013 ¶¶181-188.

Table 3-4 ServerHello Components	
Field	Use
Version	Identifies the version of the SSL protocol to be used for this communication.
RandomNumber	A 32-byte random number used to seed the cryptographic calculations.
SessionID	Identifies the specific SSL session.
CipherSuite	The cryptographic parameters to be used for this communication.
Compression-Method	The data compression method to be used for this communication.

Id., Table 3-4.

Grounds 1A-1C satisfy claim 2 using Chou's hardware. *See* [1p]. TLS-1.2 requires secure keys to provide a secure connection. TLS-1.2, 96. To ensure secure keys and to operate on the server nonce to calculate other shared secrets and keys, the server random nonce must be stored in a secure memory. *See* TLS-1.2, 96, 25-26, 64; Chou ¶¶20-21; EX-1013 ¶¶189-190.

Chou discloses that secure memory 22 is used “to store data that is processed and/or to be processed by security processor 21.” Chou ¶21; *see also id.* ¶35. Chou also discloses that “the unwrapping of secure keys is done solely within HSM.” *Id.*

¶¶25, 22. A POSITA would have understood the Chou-Bugiel combination would have stored the server nonce in secure memory 22. Alternatively, a POSITA would have understood the Chou-Bugiel combination would have stored the server nonce in, for example, RAM 53 or secure memory 22, etc. EX-1013 ¶¶191-192.

E. Claim 3: “pre-master key”

The '838 patent only discloses encrypting a “pre-master key” using a *server public key* from the server before sending the encrypted pre-master key to the server. *See* EX-1001, 8:28-34, 10:60-64. Assuming the patent’s disclosure of encrypting a pre-master key using the server’s public key (rather than the “security key”) supports claim 3 as written, **Grounds 1A-1C** implement TLS-1.2 and thus satisfy this feature. This is consistent with Patent Owner’s allegations, which assert infringement based on the TLS handshake. *See* EX-1016, Ex. F-1 at 23-25 (accusing TLS handshake), Ex. F-2 at 26-28 (same).

As discussed in [1d], Bugiel discloses verifying a server certificate $Cert_S$ and generating a pre-master key (PMS) (e.g., the claimed “pre-master secret”) to derive a session key S_c from the PMS. *See* Bugiel, Fig. 2 (“PMS $\leftarrow$ GenRndNm(),” and “ $S_c \leftarrow$ DeriveSessionKey(PMS)”).

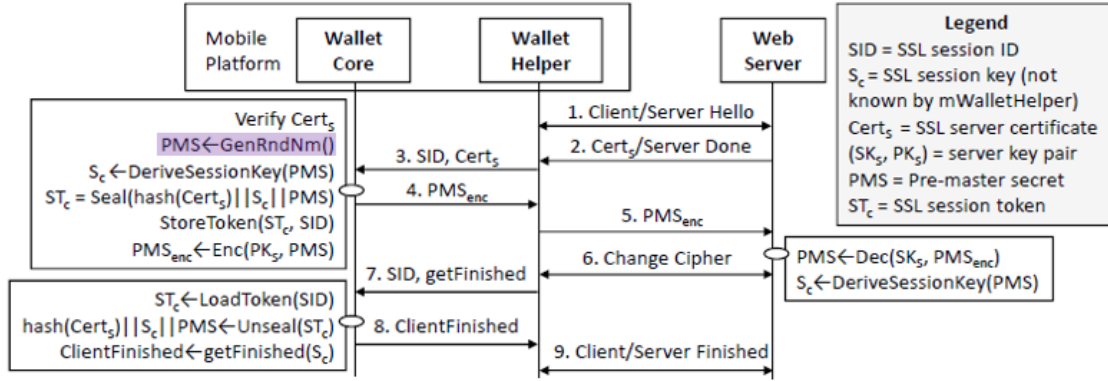


Fig. 2. SSL/TLS handshake

Id. (annotated); EX-1013 ¶¶193-194.

Grounds 1A-1C encrypt the pre-master key before sending it to the server.

Figure 2 discloses encrypting the pre-master key into “PMS_{enc}” and sending the encrypted pre-master key PMS_{enc} to the Web Server in Step 4. *See* Bugiel, Fig. 2 (“PMS_{enc} ← (PK_s, PMS),” where PK_s is the server public key).

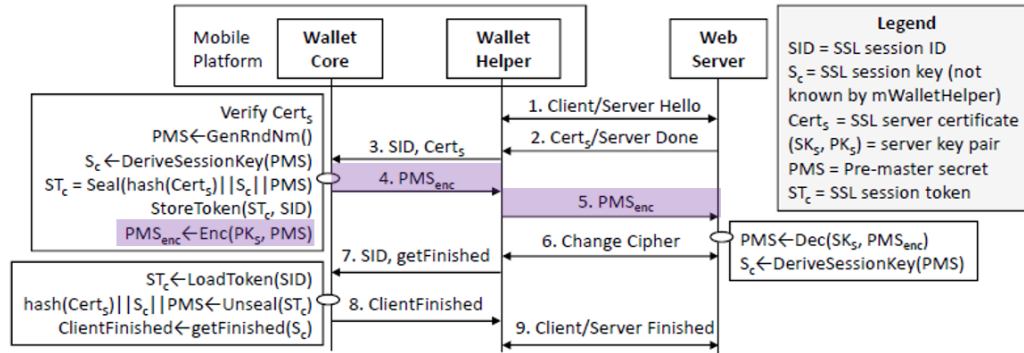


Fig. 2. SSL/TLS handshake

Id. (annotated); EX-1013 ¶195.

Bugiel’s PMS is consistent with the TLS 1.2 protocol, which discloses “generat[ing] a 48 byte premaster secret, encrypt[ing] it using the public key from

the server’s certificate, and send[ing] the result in an encrypted premaster secret message.” TLS-1.2, 58-61; *see also id.*, 50, 57, 91. Bugiel’s encrypting its pre-master secret in PMS_{enc} using a server public key also corresponds to the TLS-1.2 specification. *See* TLS-1.2, 57-61. After the ServerHello, the client sends a ClientKeyExchange message. *See id.*; EX-1006, 40-41, 45-46. This message includes “session key information (encrypted with server’s public key).” EX-1006 40-41; TLS-1.2, 58; EX-1006, 46. In RSA encryption, “the ClientKeyExchange carries a premaster secret.” *Id.*, 85-87, Fig. 4-21; *see also id.*, 86; TLS-1.2 58-59; EX-1013 ¶¶196-198.

Chou’s HSM 20 would have implemented Bugiel’s and/or Bugiel and TLS-1.2’s teaching of generating a pre-master secret. For example, PKA module 56 of HSM 20 is communicatively coupled to bus 71. *See* Chou ¶35, Fig. 3. PKA module 56 communicates with the non-secure zone through interfaces 42, bus 72, and IPC Msg RAM 61. *Id.* ¶¶35-36. Using these connections, Chou’s PKA module 56 would have loaded the pre-master key and the server public key passed through the security interfaces, or the security key from the OTP 54 to encrypt the pre-master key using the server public key or security key with the KEK unit 55. EX-1013 ¶¶199. In this way, a POSITA would have used Chou’s hardware to encrypt the pre-master key using the server public key as Bugiel and TLS-1.2 describe. *Id.*; *see* Chou, 225-226, Fig. 2; TLS-1.2, 57-61.

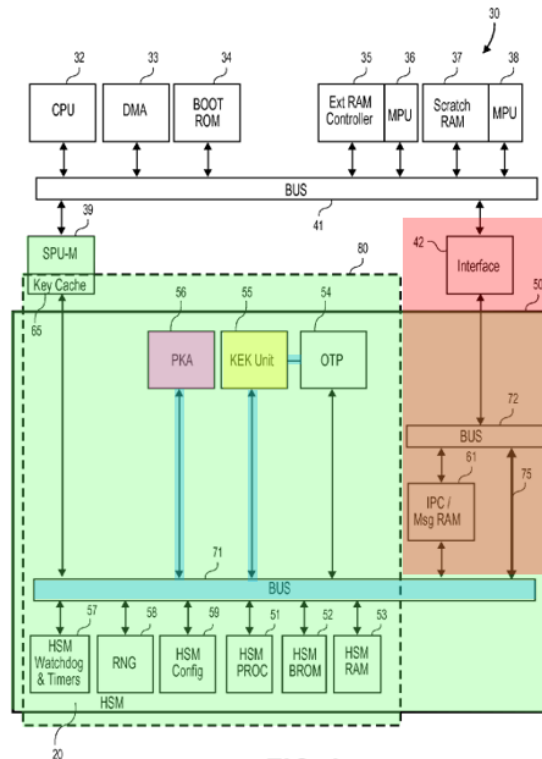


FIG. 3

Chou, Fig. 3 (annotated).

The Bugiel-Chou combinations alternatively disclose claim 3 by storing keys during the client key exchange. *See* EX-1001, 11:6-9; EX-1013 ¶¶200-201. Chou’s HSM 20 would have implemented Bugiel’s teaching to store the pre-master key in the secure storage and in an encrypted state (encrypted using the security key). *Id.*; Bugiel, 225-226, Fig. 2. As explained for claim feature [1a-3], a POSITA would have found it obvious to use Chou’s device unique key/root key as the claimed “security key.” Chou’s KEK 55 would encrypt Bugiel’s PMS (pre-master secret) using Chou’s device unique key/root key stored in OTP 54. EX-1013 ¶¶202-203.

F. Claim 4: “wrap the pre-master key”

The '838 patent uses the terms “encrypted” and “wrapped” synonymously. *See* EX-1001, 5:67-6:1, 6:6-9, 8:29-32. Wrapping a key refers to encrypting a key. **Grounds 1A-1C** therefore satisfy claim 4 for the same reasons the pre-master key is encrypted with the server public key. *See* claim 3; EX-1013 ¶204.

Grounds 1A-1C also “store” the wrapped pre-master. Bugiel discloses sealing the pre-master secret together with the session key using the security key and store the resulting session-token ST_C in a secure storage. *See* Bugiel, 225-226, 231; *see supra* [1a-3] and [1e]-[1f]. TLS-1.2 also teaches this by storing the pre-master key to compute a master secret using the pre-master key. *See* TLS-1.2, 64. A POSITA therefore would have stored the encrypted (e.g., “wrapped”) pre-master key in Chou’s secure memory (e.g., secure memory 22 or HSM RAM 53). EX-1013 ¶205.

Chou’s security processor 21 provides for “key authentication and the encrypting and/or decrypting operation to encrypt and/or decrypt data,” where “[s]ecure memory 22 is utilized to store data that is processed and/or to be processed by security processor 21.” Chou ¶21; *see also id.* ¶¶35, 43-53 (PKA unit 56 operable to perform Diffie-Hellman and RSA encryption). A POSITA would have generated and stored the encrypted pre-master key in Chou’s secure memory because, when implementing the method of Bugiel and TLS-1.2, the encrypted pre-master key is data processed by security processor 21 or PKA unit 56.

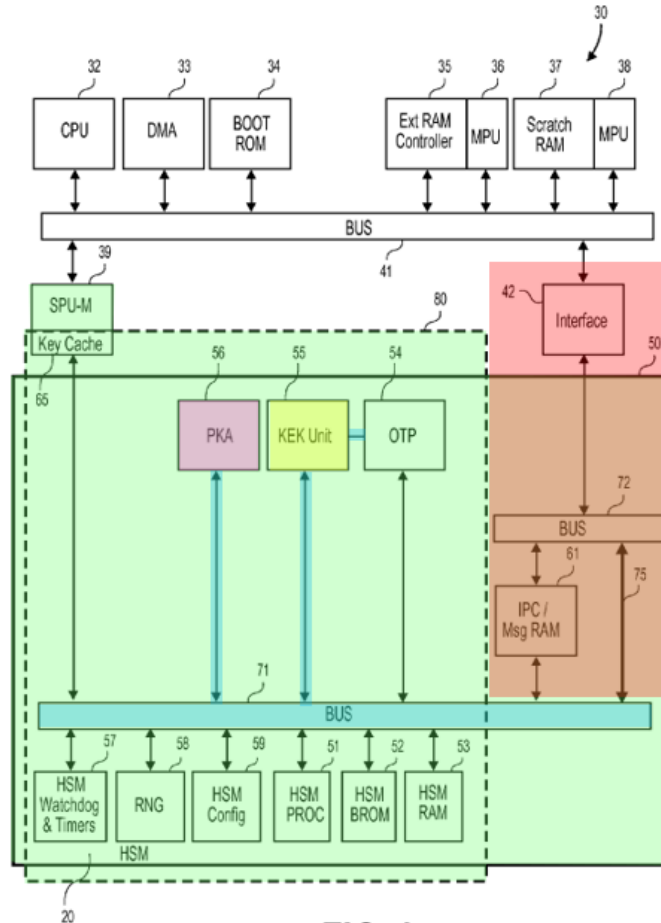


FIG. 3

Id., Fig. 3; EX-1013 ¶206.

G. Claim 5

1. [5a]: “generate the session key”

As discussed in [1d], Bugiel discloses verifying a server certificate $Cert_S$, generating a pre-master key (PMS), and then deriving session key S_c (e.g., the “symmetric session key”) from the PMS. *See* [1d]; Bugiel, Fig. 2. Alternatively, TLS-1.2’s generating the “master key” or “key_block” based on the pre-master key satisfies this feature. *See* [1d]; TLS-1.2, 64-65; EX-1006, 25-26, 79-81, 85-87, 88-89, 25-26; *see also* Chou ¶¶43-53, Fig. 3; EX-1013 ¶207.

2. [5b]: “calculate a hash function”

Bugiel teaches performing a hash algorithm. *See* Bugiel, 225 (“the hash value $hash(Cert_S)$ ”), Fig. 2. TLS-1.2 discloses this feature in several ways. First, TLS-1.2 discloses calculating the master secret by calculating a hash function from the server random nonce: “master_secret = PRF (pre_master_secret, “master secret”, ClientHello.random + ServerHello.random).” TLS-1.2, 64 (Computing the Master Secret), 14 (“PRF with the SHA-256 hash function is used for all cipher suites”). “TLS’s PRF is created by applying ***P-hash to the secret*** as: PRF(secret, label, seed) = P_<hash>(S1, label + seed).” *Id.*, 15. This calculates a hash function (a PRF function) as a function of the random nonce (ClientHello.random) generated by the client’s security engine (*see* [1b]) and random nonce (ServerHello.random) received from the remote server.

Second, TLS-1.2 discloses this by calculating the encryption key_block: “key_block = PRF(SecurityParameters.master_secret, ‘key expansion’, SecurityParameters.server_random + SecurityParameters.client_random.” *Id.*, 26; *see also id.*, 94. Creating encryption key_block discloses [5b] for the same reasons as calculating the master secret.

Third, TLS-1.2 teaches generating a Finished message including a hash of all the handshake messages: “PRF(master_secret, finished_label, Hash(handshake_message) [0 .. verify_data_length-1],” where handshake_message

includes all the handshake messages, including ClientHello and ServerHello messages. *See id.*, 63-64 (Finished). Because ClientHello includes ClientHello.random, and because ServerHello includes ServerHello.random, creating the Finished message discloses [5b].

Fourth, TLS-1.2's client generates a Certificate Verify message after a client key exchange message, which "compute[s] running hashes for all potential hash algorithms" on all handshake messages from the client hello message up to, but not including, the Certificate Verify message. *Id.*, 62 (Certificate Verify). This includes exchanged ClientHello and ServerHello messages. *See id.* Because the ClientHello message includes ClientHello.random, and because the ServerHello message includes ServerHello.random, creating this Certificate Verify message after receiving the ServerHello message discloses [5b]. EX-1013 ¶¶208-212.

Bugiel and TLS-1.2 would have been implemented by Chou, which is designed to calculate a hash function. *See* Chou ¶¶43-53 (PKA module 56 performs algorithms like "1024-bit DSA signature with SHA-1 hash or 2048-bit DSA signature with SHA-256 hash"); EX-1013 ¶213.

H. Claim 6

1. [6a]: "generate ... Diffie-Hellman key[s]"

TLS-1.2's symmetric key and master secret are generated by RSA or Diffie-Hellman key exchange. *See* EX-1006, 81-84, 40-45, 54-59; *see* [1c].

Implementing Diffie-Hellman, **Grounds 1A-1C** would have implemented TLS-1.2 (EX-1004) on Chou’s hardware. In this alternative, the master secret (e.g., the symmetric session key) would have been generated based on a ClientKeyExchange message that includes a Diffie-Hellman client key. *See* TLS-1.2, 57-61; *see also id.*, 61 (“This structure conveys the client’s Diffie-Hellman public value (Yc)”); EX-1006, 85-88; EX-1013 ¶¶214-215.

Using Diffie-Hellman key exchange, the client sends the “client’s Diffie-Hellman public value (Yc),” i.e., the claimed public Diffie-Hellman client key, to the server. TLS-1.2, 61. TLS-1.2 generates a private Diffie-Hellman client key by “generating a fresh DH private key (X) for each handshake.” *Id.*, 93, 87.

TLS-1.2 suggests implementing Diffie-Hellman key exchange according to RFC 2785. *See id.*, 93 (citing [SUBGROUP]), 100 (defining [SUBGROUP] as RFC 2785). RFC 2785 explains that, during Diffie-Hellman Key Exchange, each of parties A and B generates a **private key** (xa and xb, respectively) and then generates a **public key** (ya and yb, respectively) based on its private key:

```
ya is Party A's public key; ya = g ^ xa mod p
yb is Party B's public key; yb = g ^ xb mod p
xa is Party A's private key; xa is in the interval [2, (q - 2)]
xb is Party B's private key; xb is in the interval [2, (q - 2)]
p is a large prime
g = h^((p-1)/q) mod p, where
h is any integer with 1 < h < p-1 such that h^((p-1)/q) mod p > 1
  (g has order q mod p)
q is a large prime
j a large integer such that p=q*j + 1
```

EX-1011, 2.

When Party A is the client and Party B is the server, x_a is the claimed “private Diffie-Hellman client key,” and y_a is the claimed “public Diffie-Hellman client key. **Grounds 1A-1C** implementing Diffie-Hellman, including RFC 2785, thus generates a public Diffie-Hellman client key and a private Diffie-Hellman client key. EX-1013 ¶¶215-218.

This would have been implemented by Chou, for example using PKA module 56. Chou ¶¶43-53; EX-1013 ¶219.

2. [6b]: “sign ..., and send”

The TLS Handshake protocol signs the public Diffie-Hellman client key and sends the signed public Diffie-Hellman key to the remote server. *See* TLS-1.2, 62 (Certificate Verify). The public Diffie-Hellman client key Y_c is sent to the server to authenticate and verify the client has a corresponding private key. *See id.* More specifically, TLS-1.2 teaches sending a Certificate Verify including a “digitally-signed” version of all previous handshake messages, which would include the client key exchange message that contains public Diffie-Hellman client key Y_c . This “provide[s] explicit verification of a client certificate.” *Id.* (Certificate Verify). EX-1013 ¶220.

TLS-1.2 also teaches using a key specific to the client device so the server can verify and authenticate the signed public Diffie-Hellman client key based a device-specific key. *See* TLS-1.2, 33-37. A POSITA would have found it obvious verify and

authenticate the signed public Diffie-Hellman client key using the claimed security key. *See* [1a-3]; *see* Chou ¶38; EX-1013 ¶221.

I. Claim 7: “public Diffie-Hellman server key”

TLS-1.2 discloses claim 7 by receiving a public Diffie-Hellman server key dh_Y to complete a key exchange and calculate the premaster secret. TLS-1.2, 50-52 (ServerKeyExchange).

TLS-1.2 generates the session key as a function of the private Diffie-Hellman client key and public Diffie-Hellman server key:

First, TLS-1.2 generates the master secret (e.g., “session key”) as a function of the pre-master secret. *See id.*, 64. The client generates the pre-master secret via conventional Diffie-Hellman using the public Diffie-Hellman server key. *See id.*, 51, 65; *see also id.*, 93 (using RFC 2785). RFC 2785, in turn, teaches that a shared key (ZZ), i.e., the pre-master key, is generated as follows:

Note that the individual parties actually perform the computations:

$$ZZ = (y_b ^ x_a) \bmod p = (y_a ^ x_b) \bmod p$$

where $^$ denotes exponentiation.

y_a is Party A's public key; $y_a = g ^ x_a \bmod p$
 y_b is Party B's public key; $y_b = g ^ x_b \bmod p$
 x_a is Party A's private key; x_a is in the interval $[2, (q - 2)]$
 x_b is Party B's private key; x_b is in the interval $[2, (q - 2)]$
 p is a large prime
 $g = h^{((p-1)/q)} \bmod p$, where
 h is any integer with $1 < h < p-1$ such that $h^{((p-1)/q)} \bmod p > 1$
(g has order $q \bmod p$)
 q is a large prime
 j a large integer such that $p=q*j + 1$

EX-1011, 2.

When Party A is the client and Party B is the server, RFC 2785 teaches the shared key (ZZ), which TLS-1.2 uses as a premaster key, is generated as a function of public Diffie-Hellman server key y_b and private Diffie-Hellman client key x_a (specifically, $ZZ = (y_b^{x_a}) \bmod p$). TLS-1.2 thus teaches generating the premaster secret as a function of the public Diffie-Hellman server key y_b and private Diffie-Hellman client key x_a .

Second, TLS-1.2 generates the `key_block` (“session key”) in the same way: (1) the `key_block` as a function of master secret; (2) the master secret as a function of pre-master secret; and (3) a premaster secret as a function of the private Diffie-Hellman client key x_a and public Diffie-Hellman server key y_b (discussed above). The `key_block` is generated as a function of the master secret. TLS-1.2, 26. TLS-1.2 therefore discloses claim 7. EX-1013 ¶¶222-229.

J. Claim 8: “Diffie-Hellman global values”

Grounds 1A-1C would have implemented the Diffie-Hellman key exchange of TLS-1.2 on Chou’s hardware. *See* claim 6; TLS-1.2, 55-58, 61, 65, 74-75, 86, 92-93; *see* [1c]. To calculate the premaster secret, TLS-1.2 discloses receiving Diffie-Hellman global values dh_p (prime) and dh_g (primitive root modulo p) (e.g., together forming the “Diffie-Hellman global values”), as well as DH_Ys (e.g., the “public Diffie-Hellman server key”), in the server key exchange message from the server. TLS-1.2, 50-53. This is consistent with the ’838 patent, which describes

Diffie-Hellman global values as, “e.g., a prime number and a primitive root of the prime number.” EX-1001, 7:47-53; EX-1013 ¶¶230-231.

Chou’s HSM 20 would have generated the session key using the TLS/SSL Handshake, such that it must receive the TLS-1.2 server key exchange message, where Chou’s security engine would have received Diffie-Hellman global values (e.g., dh_p and dh_g) and a public Diffie-Hellman server key (e.g., DH_Ys). *See* [1c]-[1d]. PKA module 56 provides acceleration for algorithms like “Diffie-Hellman public value and shared key computation” and “Elliptic curve Diffie-Hellman in prime field $GF(p)$.” Chou ¶¶43-53; EX-1013 ¶232.

The combination would have stored at least one of the Diffie-Hellman global values and public Diffie-Hellman server key in Chou’s secure memory. This includes storing them in, e.g., secure memory 22 or HSM RAM 53, to compute the associated secrets and encryption keys. A POSITA would have done so via security interface 24 or interface 42. Chou ¶¶22-23, 35; *see* claim 2; EX-1013 ¶233.

K. Claims 9-15

<u>Claim Element</u>	<u>Corresponding Disclosure</u>
[9a]	[1p]-[1a-3]
[9b]	[1b]

[9c]	[1c]
[9d]	[1d]
[9e]	[1a-2], [1a-3], [1e]
[9f]	[1a-2], [1f]
[9g]	[1g]
[10a]	[1b]
[10b]	[1b], claim 2
11	[1d], claim 3
12	claim 4
13	[1d], claim 3, [5a]
14	[1d], [5b]
[15a]	[1d], [6a]
[15b]	[6b]
[15c]	claim 7

[15d]	claim 7
-------	---------

L. Claim 16

1. [16a]: “hash code of a preceding message”

Grounds 1A-1C satisfy this feature. *See* claim [5a], [5b].

TLS-1.2 sends a client Certificate Verify message including a hash of prior handshake messages. TLS-1.2, 62; *see* [5b]. The prior handshake messages include all handshake messages sent or received by the client, starting with the ClientHello message. *See id.* The Certificate Verify message sent following the client key exchange message includes a hash of the ClientHello and client key exchange messages (e.g., “a preceding message sent from the client to the remote server”). EX-1013 ¶¶251-252.

2. [16b]: “signing the hash code”

The purpose of sending the Certificate Verify message is to “provide explicit verification of a client certificate.” TLS-1.2, 62. Signing in TLS-1.2 allows the server to verify the client’s identity. Because the device unique key/root key in Chou’s security engine is unique to the client, it would have been obvious to use this as a device-specific key to sign the hash in the combination because it allows the server to verify the identity of the client device. *See* [1a-3]; Chou ¶¶38, 45; EX-1013 ¶¶253-256.

3. [16c]: “certificate verification message”

Grounds 1A-1C satisfy this feature by sending a client Certificate Verify message. TLS-1.2, 62; *see* [5b], [16a]; EX-1013 ¶257.

M. Claim 17

1. [17p]

Grounds 1A-1C satisfy this feature. *See* [1p]-[1a-3]. **Grounds 1A-1C** implement Bugiel and TLS-1.2 using Chou’s hardware, which discloses [17a]. *See* [1p]-[1a-3]; EX-1013 ¶258.

2. [17a]-[17g]

<u>Claim Element</u>	<u>Corresponding Disclosure</u>
[17a]	[1p]-[1a-3], [1b]
[17b]	[1b]
[17c]	[1c]
[17d]	[1d]
[17e]	[1p]-[1a-3], [1e]
[17f]	[1a-2], [1f]
[17g]	[1f]

N. Claims 18-23

<u>Claim Element</u>	<u>Corresponding Disclosure</u>
[18a]	[10a], [1b]
[18b]	[10b], claim 2, [1b]
19	claim 3
20	claim 4
21	[1d], [5a]
22	[1d], [5b]
[23a]	[1d], [6a]
[23b]	[6b]
[23c]	claim 7
[23d]	claim 7

O. Claim 24

Grounds 1A-1C satisfy this feature. *See* [1p]-[1a-3]. HSM 20 and SPU-M 39/Key Algorithm Module 15 are communicatively coupled to CPU 11/32 (the claimed “processor core”) through a Security Interface 24/42 and busses 16/41 and

27/72 (e.g., the “interconnect”). *See* Chou ¶¶22, 30, 33, 35-36. This is consistent with the ’838 patent. *See* EX-1001, 3:67-4:3; EX-1013 ¶¶276-277.

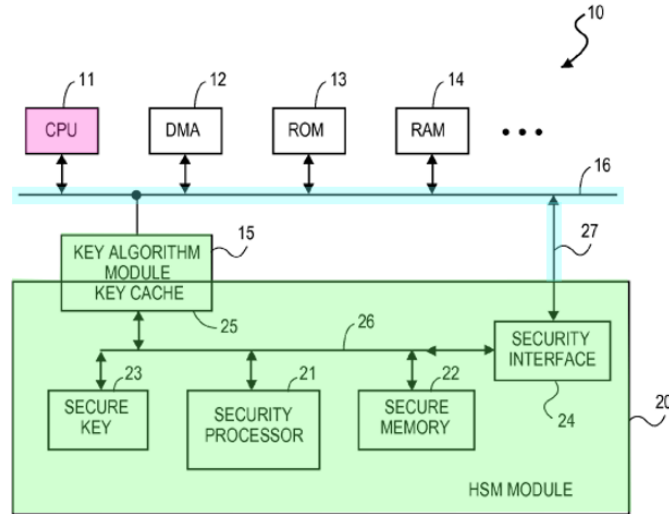


FIG. 1

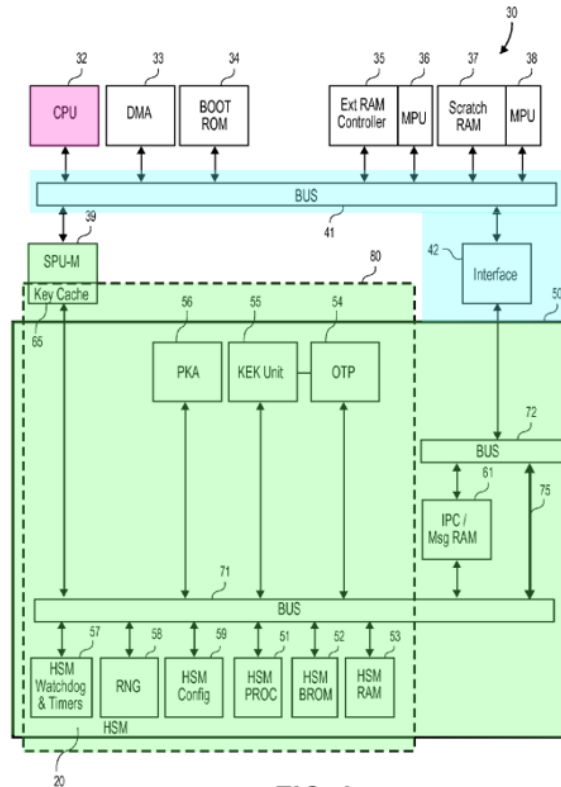


FIG. 3

Chou, Figs. 1, 3 (annotated).

VIII. Ground 2A, 2B: Tkacik and TLS-1.2, and Tkacik, TLS-1.2, and Bugiel Render Obvious Claims 1-24

A. Motivation to Combine Tkacik and TLS-1.2

A POSITA would have implemented TLS-1.2's Handshake using Tkacik's architecture, including using Tkacik's secure processing section to generate SSL keys. *See* Tkacik ¶¶79, 80; TLS-1.2, 6, 33. A POSITA would have combined Tkacik-TLS-1.2 to follow an "Internet standard[] track protocol" that "prevent[s] eavesdropping, tampering, or message forgery." TLS-1.2, 1. A POSITA would have been motivated to utilize Tkacik's hardware to protect keys used for encrypted communication sessions on its host device from access from malicious processes that may try to steal these keys for malicious purposes. Tkacik ¶21; EX-1013 ¶¶278-281.

Tkacik-TLS-1.2 would have been nothing more than combining prior art elements according to known methods to yield predictable results. A POSITA would have had an expectation of success because TLS-1.2 was a standard communication protocol and Tkacik expressly contemplates generating keys for SSL. Tkacik ¶80; EX-1013 ¶282.

This also would have been applying a known technique of communicating over the Internet to a known device, which would have improved Tkacik by enabling secure communication according to the latest version of SSL/TLS. EX-1013 ¶¶283-284.

B. Motivation to Combine Tkacik-TLS-1.2 with Bugiel

A POSITA would have found it obvious to store Tkacik's encrypted session key in secure memory based on Bugiel. Bugiel discloses storing an encrypted session key in SecureStorage, which is ideally implemented within a hardware TrEE. Bugiel, 224-226, 235-235, Figs. 1, 2. A POSITA would implemented this teaching to store the encrypted session key generated in Tkacik-TLS-1.2 in one of Tkacik's secure memory. *See* Grounds 1A-1C, [1a-2]; EX-1013 ¶¶285-287.

Bugiel also discloses storing the encrypted session key in isolated memory as it is being encrypted/decrypted during run-time. Bugiel, 232, 228, Figs. 4-6. A POSITA would have applied these teachings to store Tkacik's ciphertext key 372 (e.g., an encrypted session key) in secure memories of Tkacik as the session key is being encrypted/decrypted. *See* Tkacik ¶¶37, 54, 73-74, 84; EX-1013 ¶288.

This would have been nothing more than combining known elements in the prior art according to known principles, and applying a known storage technique to Tkacik, with predictable results. A POSITA would have had a reasonable expectation of success in applying Bugiel's storing keys in secure storage to Tkacik because both disclose similar hardware for encrypting/storing keys. A POSITA would have applied Bugiel to Tkacik to store keys in persistent storage to protect user credentials. Bugiel, 232; EX-1013 ¶289.

C. Claim 1

1. [1p]

Tkacik's secure processing system 12 discloses a "system-on-a-chip apparatus" in which the apparatus is a system-on-a-chip. Tkacik ¶¶20-25, Fig. 1.

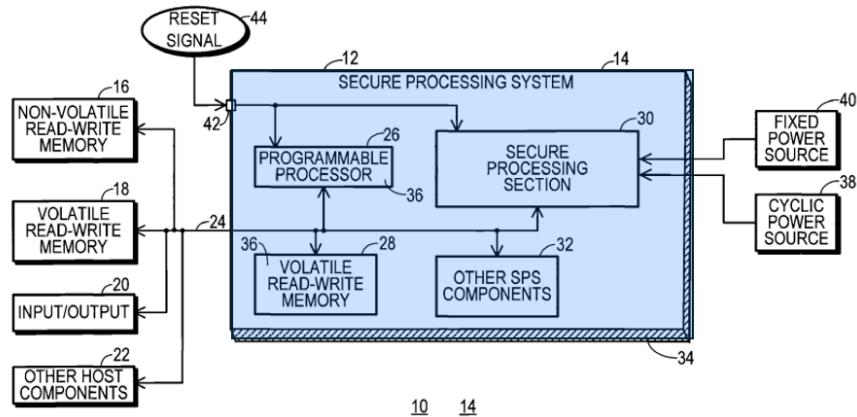


FIG. 1

Id., Fig. 1.

Secure processing system 12 includes "processor 26, memory 28, secure processing section 30, and other SPS components 32" "formed together on a *common semiconductor substrate* and are packaged together and reside within a *single integrated circuit 34*." *Id.* ¶25 (emphases added). Volatile read-write memory 28 of secure processing system 12 is "on-chip memory 28," as opposed to "off-chip memories 16 and 18." *Id.* ¶26. Secure processing system 12 is therefore a "system-on-a-chip apparatus." EX-1013 ¶¶290-291.

Secure processing system 12 is consistent with the '838 patent, which discloses system-on-a-chip devices that “are integrated circuits that incorporate various components, in addition to the processing core, of electronic systems on a single die,” and SOC having “[multiple] components on a single chip.” EX-1001, 1:27-32; EX-1013 ¶292.

Tkacik also discloses an apparatus that has a system-on-a-chip. Tkacik’s host device 10 and/or encryption apparatus 14—described synonymously—*include* secure processing system 12. Tkacik ¶¶20-25. Host device 10 “utilize[s]” and “includes” secure processing system 12.” *Id.* ¶¶20-22. Because host device 10 includes system-on-a-chip 12, host device 10 and/or encryption apparatus 14 discloses a “system-on-a-chip apparatus.” *Id.* ¶¶20-25, Fig. 1.

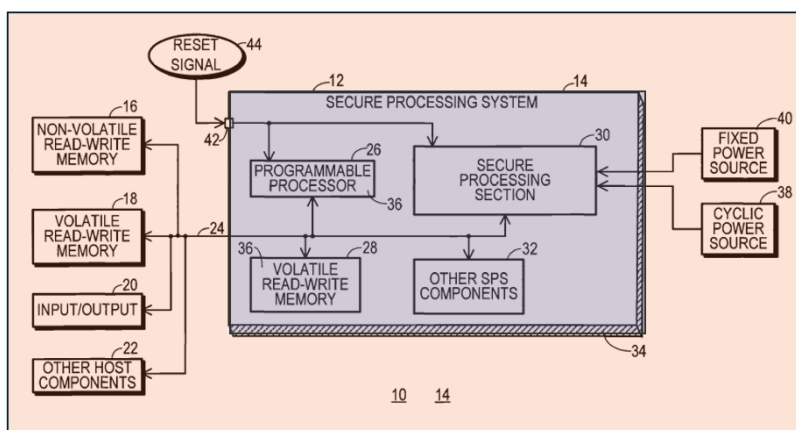


FIG. 1

Id., Fig. 1; EX-1013 ¶293.

2. [1a-1]

Tkacik's secure processing section 30 and programmable processor 26 disclose this feature. Tkacik ¶¶25-26, Fig. 1. Secure processing section 30 is a “security engine,” consistent with the '838 patent, which discloses the security engine “maintains the keys in secrecy within a secure memory 114.” EX-1001, 3:35-41, 4:17-34.

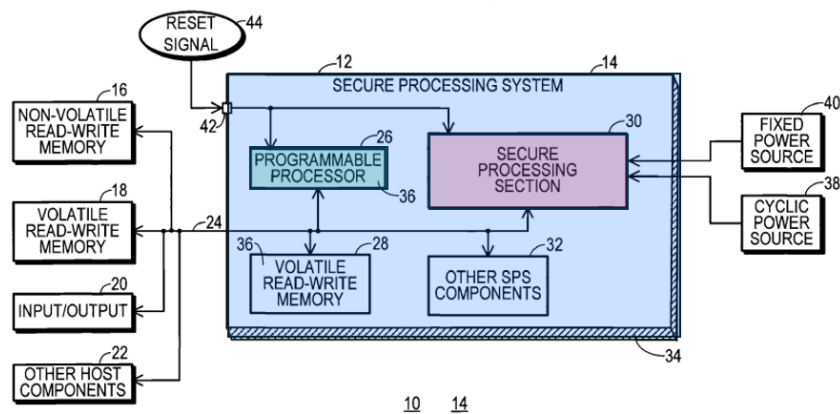


FIG. 1

Tkacik, Fig. 1 (annotated); EX-1013 ¶¶294-295.

Secure processing section 30 “provides security services ... including cryptographic operations which use keys stored in unsecured memory 16, 18 and/or 28 located outside of section 30.” Tkacik ¶26. Secure processing section 30 includes volatile key register 54 and long-lived, non-volatile key register 60. *Id.* ¶¶38, 44-46. Volatile key register 54 “retain[s] secret plaintext cryptographic key” that “has a limited lifetime and is preferably valid for no longer than the duration between

resets.” *Id.* ¶¶38, 88. Long-lived, non-volatile key register 60 “stores one or more plaintext master keys 160.” *Id.* ¶¶44-46; EX-1013 ¶296.

Secure processing section 30 also includes zeroizable read write memory 52 that “provides secure memory for secure processing system 12,” where key register 54 and zeroizable read-write memory 52 may be separate or the same volatile memory. Tkacik ¶38. Key register 54, active key register 72, control register 74, and data register 76 are included as part of read-write memory 52. *Id.* ¶¶38, 54.

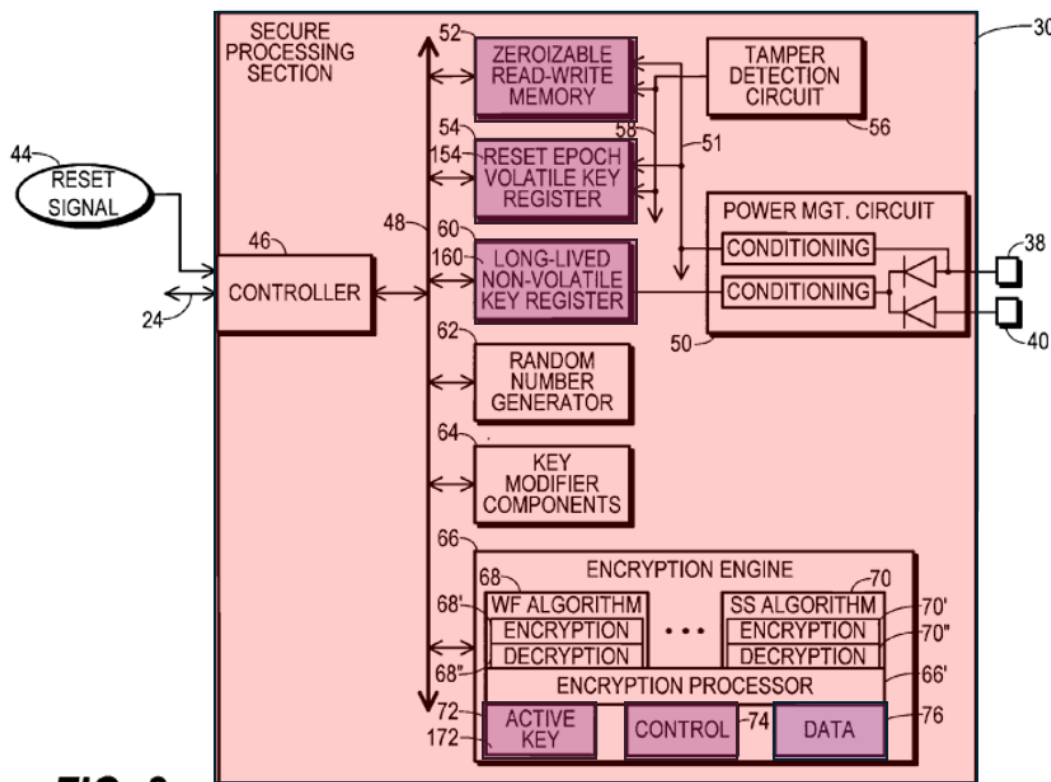


FIG. 2

Id., Fig. 2 (annotated); EX-1013 ¶297.

Secure processing section 30 is a “security engine.” *See* Tkacik ¶26. Tkacik also discloses a separate “processor core” on the system-on-a-chip. This includes

“programmable processor 26,” *id.* ¶¶23-24, which controls secure processing system 12. *Id.* ¶30. Programmable processor 26 and secure processing section 30 are separate structures on secure processing system 30 that are “couple[d] to bus 24.” *Id.* ¶25.

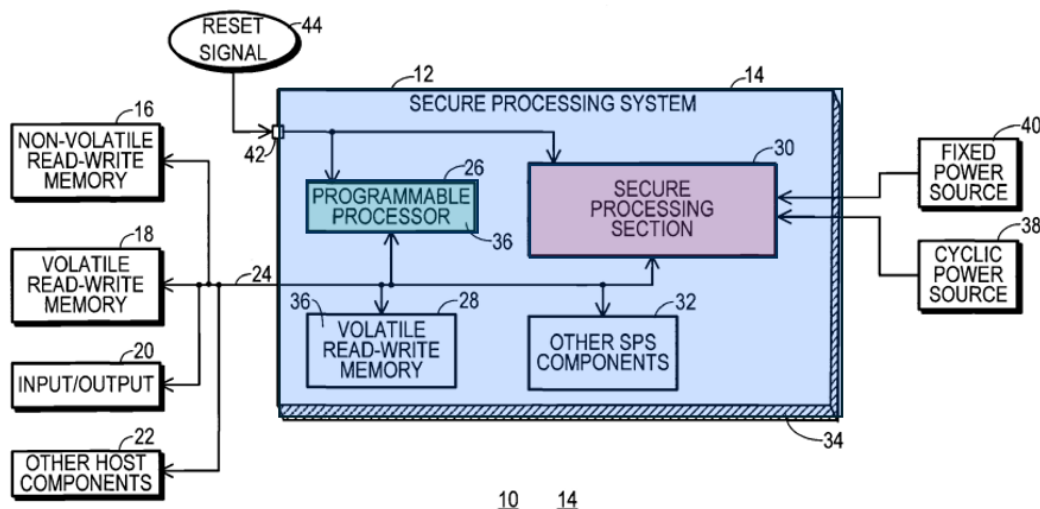


FIG. 1

Id., Fig. 1 (annotated); EX-1013 ¶¶298-299.

3. [1a-2]

Tkacik discloses memories within secure processing section 30: zeroizable read-write memory 52, key register 54, long-lived non-volatile key register 60, active key register 72, control register 74, data register 76. Tkacik ¶¶36-38, 44-46, 54-55. Tkacik’s key register 54, active key register 72, control register 74, and data register 76 may be included as part of read-write memory 52. *See id.* ¶¶38, 54. These

are “secure memory” because Tkacik’s “memory located within secure processing section 30 ... is secured memory.” Tkacik ¶26; *see also id.* ¶37.

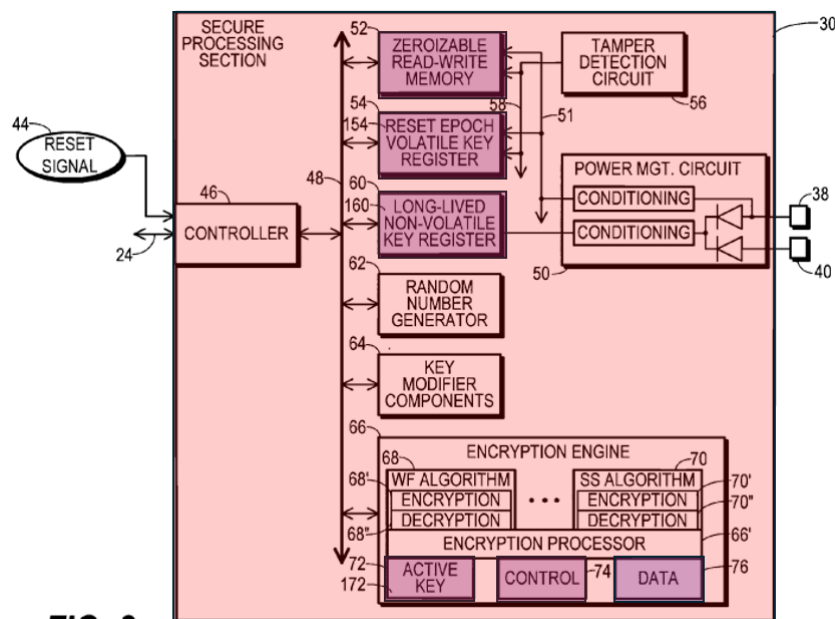


FIG. 2

Id., Fig. 2 (annotated); EX-1013 ¶300.

Zeroizable read-write memory 52, key register 54, long-lived non-volatile key register 60, active key register 72, control register 74, data register 76 all disclose memory “accessible only by the security engine” because these are within secure processing section 30 that “provides a secure execution environment (SEE).” Tkacik ¶26; *see also id.* ¶5. Zeroizable read-write memory 52’s “contents are protected against the execution of malicious software within host device 10,” and its storage capacity may be intentionally limited to maintaining keys, “in addition to other data storage and buffering tasks ... in connection with moving blocks of payload data *into and out* from secure processing section 30.” *Id.* ¶37; EX-1013 ¶¶301-302.

4. [1a-3]

Tkacik discloses long-lived key register 60, part of the “secure memory” of secure processing section 30, includes a plaintext master key 160. Tkacik ¶¶44-46, Fig. 2. Plaintext master key 160 is “long-lived” and “has a life which extends across power cycles.” *Id.* ¶44. Plaintext master key 16 is a “security key” because it is used for cryptographic operations, including a SS algorithm 70, for “symmetric cryptographic algorithms.” *Id.* ¶¶49-50, 57, 69-70, 74, 83-84; EX-1013 ¶¶303-304.

Tkacik’s plaintext master key 160 is “encoded in the secure memory during a manufacturing process of the system-on-a-chip” because it is “programmed ... during the manufacturing process of secure processing system 12 and cannot thereafter be altered.” Tkacik ¶46; *see* Section VIII.A; EX-1013 ¶305.

5. [1b]

Tkacik generates a random nonce using random number generator 62, Tkacik ¶47, Fig. 2, which “generates random numbers ... [that] may then be moved and stored in other components of secure processing section 30, or more generally host device 10, for subsequent use,” *Id.* The random numbers are used to “generate cryptographic keys,” including “master key 160.” *Id.* ¶48, 59, 80, claims 1, 9; EX-1013 ¶306.

Tkacik’s random number generator 62 is part of secure processing section 30. Tkacik ¶¶47-48, Fig. 2. Random number generator 62’s random number is a random

nonce. *See* Section VII.C.5 (**Grounds 1A-1C**, [1b]). Indeed, the '838 patent discloses “using “any suitable random number-generating algorithm to generate the random nonce.” EX-1001, 10:14-27, 6:40-50.

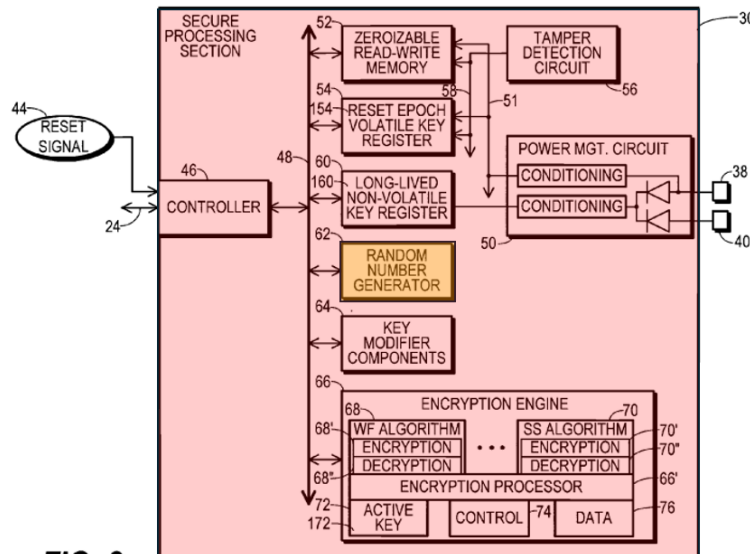


FIG. 2

Tkacik, Fig. 2 (annotated); EX-1013 ¶307.

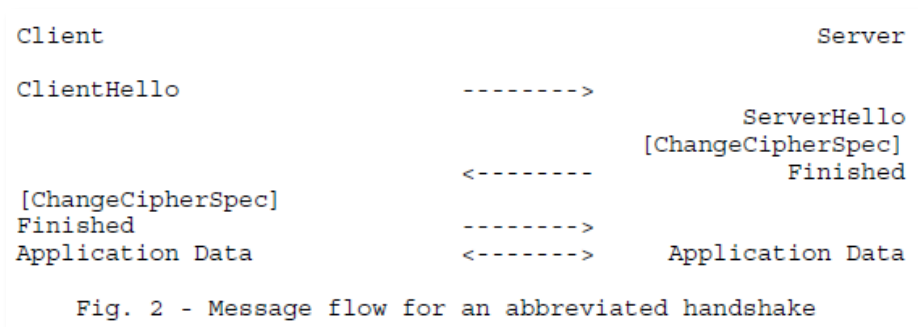
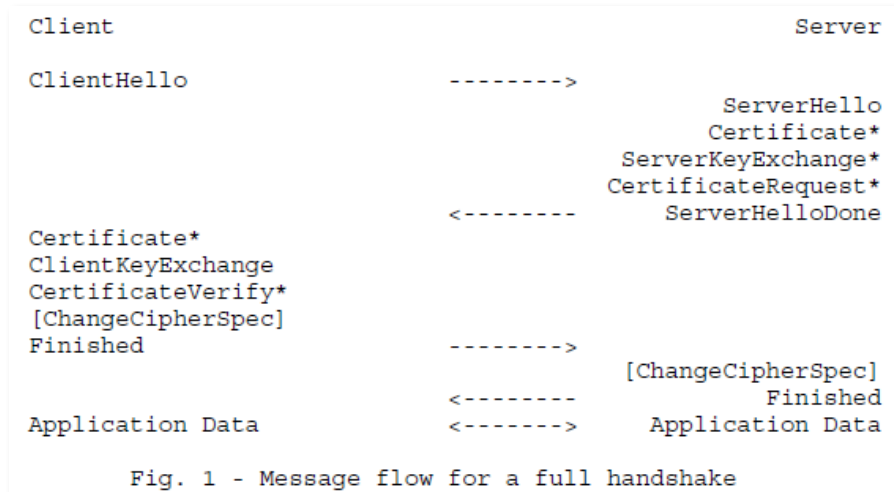
TLS-1.2 ClientHello.random and ServerHello.random were random nonces exchanged between client and server. TLS-1.2, 91-92 (incorporating EX-1018, 74); *see* Section VII.C.5 (**Grounds 1A-1C**, [1b]); EX-1013 ¶308.

Tkacik’s host device 10 is “an electronic device,” which “includes provisions for maintaining the security of data that may be ... communicated by host device 10.” Tkacik ¶21; *see also id.* ¶8. Therefore, Tkacik discloses or at least suggests initiating a request for a secure communication session with a remote server over a network using the nonce. EX-1013 ¶309.

Tkacik generates keys “using any number of protocols,” including SSL. Tkacik ¶80. SSL keys are utilized for secure communication between different devices, including client host devices and servers. TLS-1.2, 1. EX-1006, 2-6, 37-38. The patent discloses this as well. EX-1001, 1:10-24; EX-1013 ¶310.

A POSITA would have combined Tkacik with TLS-1.2 to initiate the claimed request using a generated nonce. The combination would have incorporated Tkacik’s architecture to implement TLS-1.2’s communication protocol, and would have utilized secure processing section 30 to generate keys based on the SSL protocol. *See* Tkacik ¶79; EX-1013 ¶311. Indeed, “TLS should be used to establish a secure connection between two parties.” TLS-1.2, 6. The first step is for the client to generate ClientHello.random and send a client hello message including ClientHello.random to the server. *Id.*, 34, 36-37, 39. TLS-1.2’s ClientHello.random corresponds to Tkacik’s random number. *Id.*, 17, 33-34; EX-1006, 41-43; EX-1013 ¶¶311-312.

The Tkacik-TLS-1.2 combination would have generated a ClientHello.random and sent the server a ClientHello message, including ClientHello.random, to initiate the TLS secure connection, according to the TLS-1.2 handshake. TLS-1.2, 3-4, 36-37, 39-42; EX-1013 ¶313. TLS-1.2 depicts this in Figures 1 and 2:



Id., Figs. 1, 2.

In Tkacik-TLS-1.2, random number generator 62 would have generated the ClientHello.random nonce to initiate communication with a server over a network using a ClientHello message as part of the TLS handshake. EX-1013 ¶314.

6. [1c]

Tkacik-TLS-1.2 satisfies this feature for the reasons explained in Section VII.C.6 (**Grounds 1A-1C**, [1c]). Tkacik-TLS-1.2 would have implemented TLS 1.2’s cryptographic key exchange using Tkacik’s hardware to generate keys according to the TLS 1.2 protocol. Tkacik discloses a “make new key process 140”

in Figure 8 that can be used to ciphertext keys 272 “as a result of a single invocation of process 140.” Tkacik ¶80, Fig. 8. Tkacik generates plaintext active key 172 using any one of many protocols known to a POSITA, including “IKE, DH, RSA key gen., SSL, random key, and the like.” Tkacik ¶80; EX-1013 ¶¶315-316.

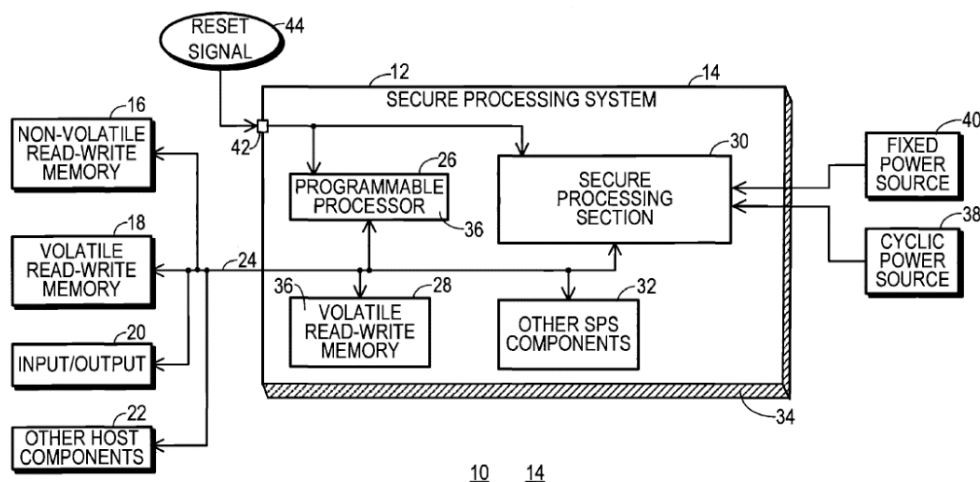


FIG. 1

Tkacik, Fig. 1.

Tkacik-TLS-1.2 would have implemented TLS 1.2’s cryptographic key exchange using Tkacik’s flexible security hardware, including secure processing section 30. See Tkacik ¶¶79-81. A POSITA would have known this would have included performing the handshake process with either RSA or Diffie-Hellman key exchange. See TLS-1.2, 26-65, 80, 90; see Section VII.C.6 ([1c]); EX-1013 ¶¶317-320.

7. [1d]

Tkacik’s secure processing section 30 perform make-new-key process 140 to generate new cryptographic keys. Tkacik ¶79, Fig. 8. This includes plaintext active key 172, which is a “symmetric session key.” *Id.* ¶¶54-56, 80-82, 87.

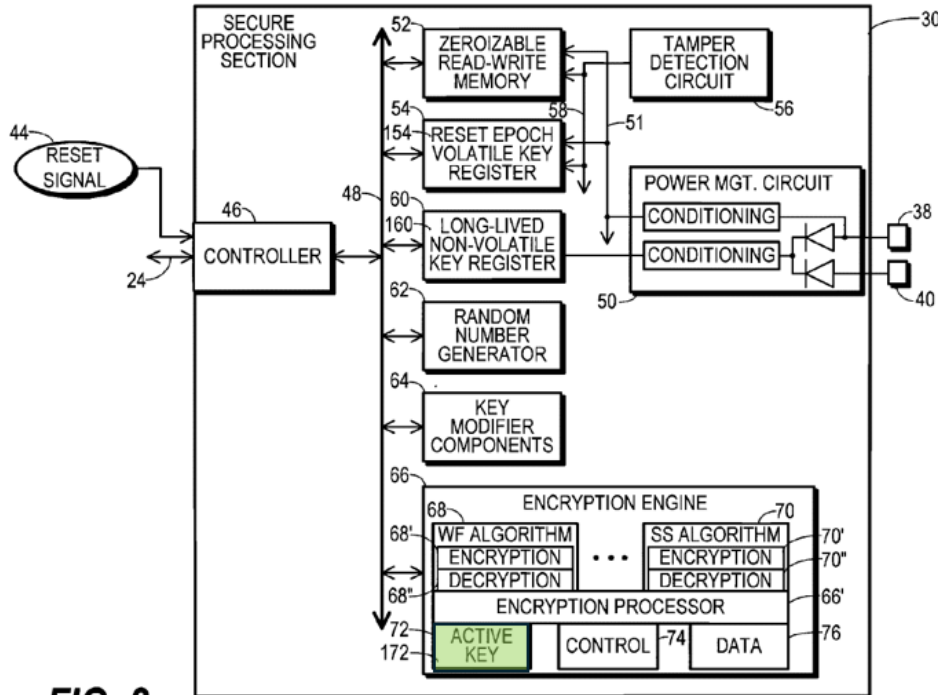


FIG. 2

Id., Fig. 2 (annotated); EX-1013 ¶321.

Plain text master key 172 performs encryption and decryption in the encryption engine 66. Tkacik ¶54. Once generated, Plaintext active key 172 is used “[i]n accordance with symmetrical cryptographic principles” to encrypt and decrypt ciphertext data at a later time. *Id.* ¶56. Tkacik’s encryption engine 66 “implement[s] a variety of *symmetric cryptographic algorithms* for the encryption of plaintext data

into ciphertext data and for the decryption of the ciphertext data back into the plaintext data.” *Id.* ¶49.

With reference to Figure 8, plaintext active key 172 may be generated based on a random number, together with “any of a number of protocols known to those skilled in the art, including ... [Diffie-Helman], RSA key gen[eration], [and] SSL.” *Id.* ¶¶79-80, Fig. 8. The plaintext active key 172 is then “saved for future use and moves the plaintext active key 172 to data register 76.” *Id.*

Plaintext active key 172 is a “symmetric session key” because it is used for symmetric cryptographic algorithms and performs encryption and decryption in the encryption engine. *See id.* ¶¶54, 56. It is also generated using known symmetric key methods, including Diffie-Helman, RSA and SSL. *Id.* ¶¶79-80, Fig. 8. This corresponds with the symmetric session key used in the ’838 patent. *Compare* EX-1001, 9:7-25, *with* Tkacik, ¶80; *see also* TLS-1.2, 65; EX-1013 ¶¶322-325.

Plaintext active key 172 is also used to encrypt and decrypt messages, as claimed. Tkacik ¶¶54, 56, 49. To the extent plaintext active key 172 alone does not disclose a symmetric session key, it would have been obvious to implement Tkacik’s plaintext active key 172 as the “master key” or “key_block” in TLS-1.2. *See* Section VII.C.7 (**Grounds 1A-1C**, [1d]); EX-1013 ¶¶326-327. The combination would have been nothing more than substituting one known element (e.g., Tkacik’s plaintext active key 172) for another (e.g., TLS-1.2’s master secret or key_block) to yield

predictable results. EX-1013 ¶¶328-329. A POSITA would have had a high expectation of success because Tkacik contemplates using SSL encryption methods and Tkacik's methods are compatible with the TLS specification, including its master secret or key_block. *Id.* It would have been obvious to perform these tasks using Tkacik's secure processing section 30 to generate keys based on SSL to encrypt/decrypt messages. *Id.* Tkacik's plaintext active key 172 would have been implemented to be the "master secret" or "key_block." *Id.*; see Tkacik ¶80; TLS-1.2, 25-26, 64.

8. [1e]

Tkacik uses its master key 160 ("security key") to encrypt the plaintext key 172 ("symmetric session key") to form a long-lived ciphertext key 372 ("encrypted session key"). Tkacik ¶¶83-84, 46; see [1a-3], [1d]. Plain master key 160 can "directly serve[] as the BKEK," if desired. *Id.* ¶¶83, 69 (BKEK meaning "blob key encryption key"). The BKEK "is a cryptographic key to be used informing a blob," and is "a self-contained object which includes a cryptographic key in an encrypted form and a message authentication code (MAC)." *Id.* ¶70; EX-1013 ¶¶330-331.

Tkacik generates the plaintext active key 172 using known protocols. See [1d]; Tkacik ¶80. Tkacik also moves the generated plaintext active key 172 "to data register 76 and encrypt[ing] this key 172 using ... the BKEK" to form long-lived

alternate ciphertext key 372. *Id.* ¶84, Fig. 8. The BKEK is the plaintext master key 160.

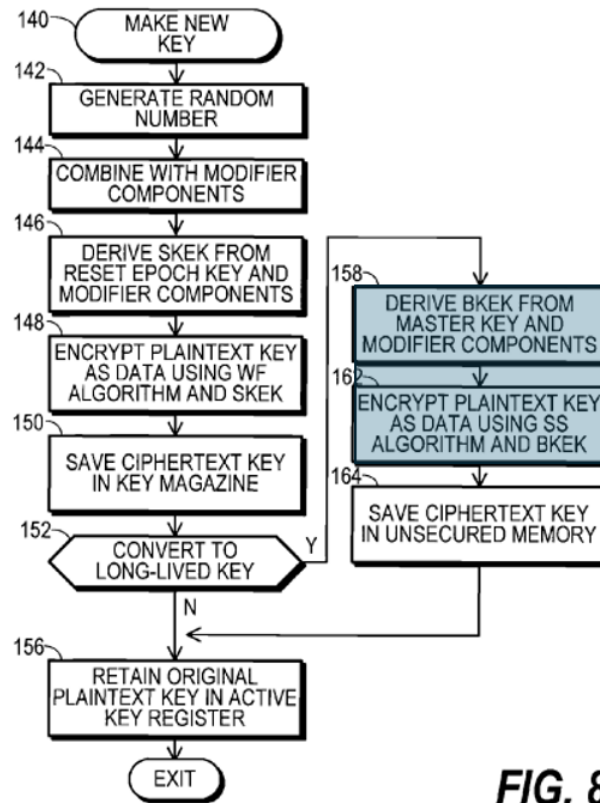


FIG. 8

Id., Fig. 8 (annotated); EX-1013 ¶372.

Tkacik discloses [1e] by disclosing that secure processing section 30 encrypts the plaintext active key 172 into long-lived alternative ciphertext key 372 based on the plaintext master key 160 (e.g., BKEK, the claimed “security key.”). It would have been obvious to implement Tkacik’s plaintext active key 172 as either the master secret or the key_block in the TLS-1.2 protocol and encrypt this as the long-lived alternative ciphertext key 372. TLS-1.2, 25-26, 64; *see* [1d]; EX-1013 ¶334.

9. [1f]

Tkacik stores alternative ciphertext key 372 (e.g., the encrypted session key) in data register 76. Tkacik ¶84. Tkacik’s task 162 “forms a long-lived alternate ciphertext key 372.” *Id.* Data register 76 discloses a secure memory that is part of the encryption engine 66 within the secure processing section 30. *Id.* ¶¶54-55, 25-26. Data register 76 prevents intrusion by disclosing that its contents may be “destroyed upon a tamper event.” *Id.* ¶54. By moving plaintext active key 172 to data register 76 and encrypting it to form long-lived alternate ciphertext key 372, the long-lived alternate ciphertext key 372 is encrypted and stored in data register 76. EX-1013 ¶¶335-336.

Alternatively, Tkacik stores the encrypted session key (e.g., the long-lived alternate ciphertext key 372) as part of a blob that is later stored to data register 76. Tkacik’s “task 164 stores long-lived alternate ciphertext key 372 in unsecured memory 16, 18, and/or 28 (FIG. 1) as part of a blob 110 (FIG. 4).” Tkacik ¶83. When the unencrypted session key 172 is needed, a blob is retrieved that *includes* the encrypted session key 372 from unsecure memory 16, 18, or 28 and *moves* it to secure data register 76, then decrypts the blob. *See id.* ¶73. Task 122 “retrieves the identified blob 110 from unsecure memory to encryption engine 66, *where it is saved in data register 76.*” *Id.* ¶74. Tkacik’s task 126 “forms a recovered plaintext active key 172 from the long-lived alternate ciphertext key 372 portion of the blob 110,

which may be *moved* to active key register 72 or *otherwise saved* within secure processing section 30 for subsequent use.” *Id.* ¶67. In this way, Tkacik’s security engine stores encrypted session key (e.g., long-lived alternate ciphertext key 372) in secure memory (e.g., data register 76).

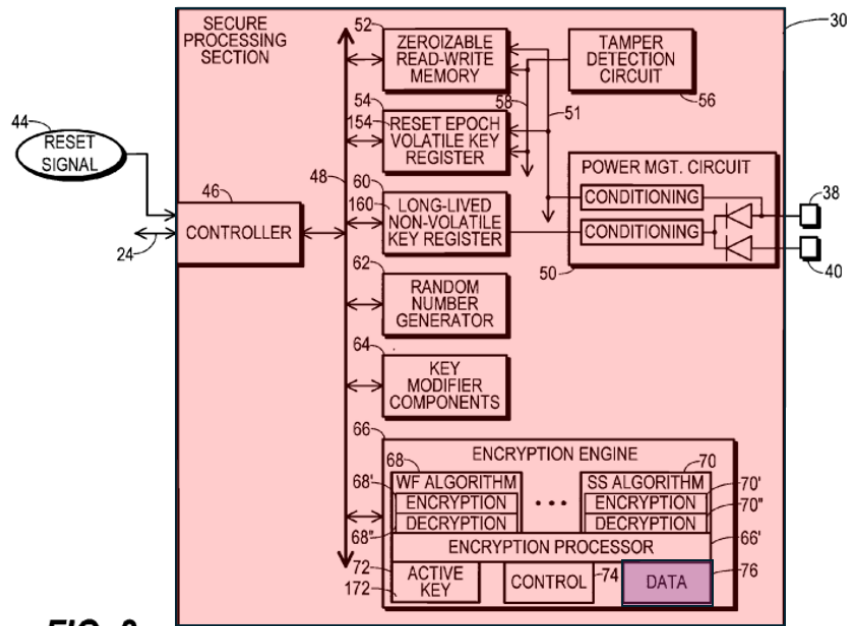


FIG. 2

Tkacik, Fig. 2 (annotated); EX-1013 ¶337.

As a second alternative, Tkacik stores the long-lived alternate ciphertext key 372 in secure memory by storing it in volatile memory. Data register 76 “may also be included as a part of zeroizable memory 52,” which performs “data storage and buffering tasks ... in connection with moving blocks of payload data into and out from secure processing section.” *Id.* ¶¶37, 54. After generating the long-lived alternate ciphertext key 372 (i.e., the claimed “encrypted session key”) in the security engine, the encrypted session key 372 is moved from the security engine 30

to the unsecure memories 16, 18, and/or 28 outside the security engine 30. *Id.* ¶84. In doing so, encrypted session key 372 is stored, at least temporarily, in RAM of the secure processing section 372, for example in zeroizable read-write memory 52 or data register 76, which can itself be part of zeroizable read-write memory 52. *See id.* ¶54. This understanding corresponds with the prior art. *See* EX-1013 ¶¶338-340.

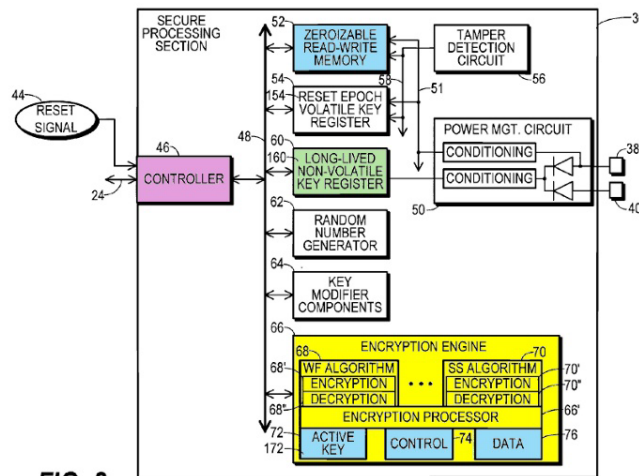


FIG. 2

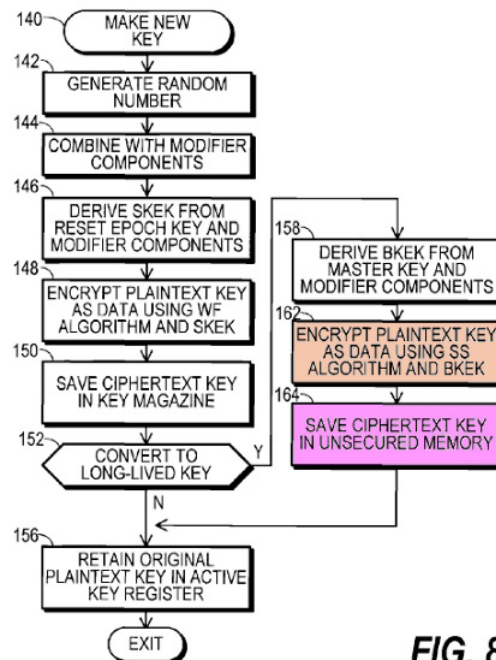


FIG. 8

Tkacik, Figs. 2, 8 (annotated).

In the Tkacik-Bugiel or Tkacik-Bugiel-TLS-1.2 combinations (**Grounds 2A, 2B**), a POSITA would have found it obvious to store the encrypted session key generated in Tkacik in the secure memory as described in Bugiel. EX-1013 ¶341. Bugiel teaches storing a session-token ST_C (the claimed “the encrypted session key”) in secure memory. *See* Section VIII.B; Bugiel, 224, 225-226, 235-235, Figs. 1, 2. The resulting session-token ST_C is stored in a secure storage and is loaded and unsealed later using the session identifier SID. Bugiel, 225-226, Fig. 2.

Tkacik’s symmetric session key 172 would have been implemented as be either the “master secret” or “key_block” according to TLS-1.2. *See* [1d]; TLS-1.2, 25-26, 64. Accordingly, the encrypted session key 372, which is encrypted based on Tkacik’s master key 160, would have been implemented for future SSL communications as either to the encrypted “master secret” or encrypted “key_block.” The encrypted “master secret” or “key_block” would have been passed from secure processing section 30 to the unsecure world, as taught by Tkacik. Tkacik ¶84. The encrypted master secret or encrypted key_block would also have been retrieved from data register 76. *See id.* ¶¶73-74; EX-1013 ¶342.

10. [1g]

After making or retrieving cryptographic keys (e.g., when process 94, 118 or 140 completes), secure processing section 30 proceeds performs “cryptographic

operation ... using the plaintext active key 172 as the encryption or decryption key.”

Tkacik ¶87.

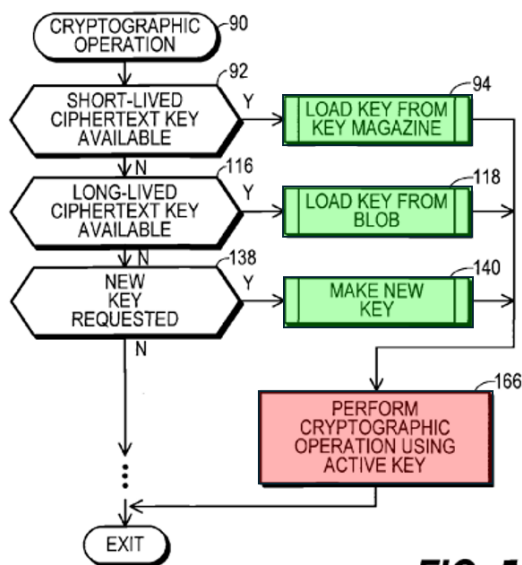


FIG. 5

Id., Fig. 5 (annotated); EX-1013 ¶¶343-344.

As explained in [1b], Tkacik-TLS-1.2 securely communicates with a remote server over a network. TLS-1.2, 4, 6. Application data can be sent only after the handshake process completes, which establishes the encryption algorithm and keys used for the communication. *See id.*, 33-37. A POSITA would have implemented TLS-1.2’s secure communication between a client with a remote server using the hardware of Tkacik, including using the plaintext active key 172 of secure processing section 30. EX-1013 ¶¶345-346. Indeed, Tkacik’s secure processing system 12 of host device 10 includes secure processing section 30, where secure processing system 12 is a system-on-a-chip. *See* [1p]. By utilizing plaintext active key 172 to initiate secure communication via TLS-1.2, Tkacik’s secure processing

system 12 (e.g., system-on-a-chip) establishes a secure communication session with the remote server over the network using the session key. EX-1013 ¶¶346. Tkacik’s secure processing system 12 may be configured for various applications and maintains the security of data that may be “communicated by host device 10.” Tkacik ¶21. Tkacik therefore suggests employing its SOC encryption features to engage in communication with remote servers.

D. Claim 2

TLS-1.2 satisfies claim 2 for the reasons in Section VII.D (**Grounds 1A-1C**). Tkacik-TLS-1.2 further satisfies storing the server nonce in the secure memory. *See* TLS 1.2, 91, 25-26, 64; Tkacik ¶¶26, 28.

Because Tkacik teaches generating keys using components of secure processing section 30, the server nonce would have been stored in memory within this section for calculation. *See* Tkacik ¶¶79, 26. This includes “[z]eroizable read-write memory 52 [that] provides secure memory,” whose contents protected from “malicious software within host device 10.” Tkacik ¶37; EX-1013 ¶¶347-350.

E. Claim 3

TLS-1.2 discloses claim 3 for the reasons in Section VII.E (**Grounds 1A-1C**).

A POSITA would have implemented Tkacik’s secure processing section 30 to generate and encrypt keys, including the pre-master key, using the server public key. EX-1013 ¶¶351-354; Tkacik ¶¶79, 80. TLS-1.2 teaches generating the pre-master

key, master key, and encryption keys during the handshake. *See* TLS-1.2, 91. TLS-1.2 also explains that the result of RSA key generation in SSL/TLS protocol is the pre-master secret. *See id.*, 92-93, 58. Tkacik-TLS-1.2 would have generated a pre-master key via Tkacik's plaintext active key 172 based on Diffie-Hellman or RSA. EX-1013 ¶355.

Tkacik's secure processing section 30 would have encrypted the pre-master key, e.g., plaintext active key 172, with a server public key received in the TLS Handshake (e.g., via the ServerKeyExchange message). *See* Tkacik ¶¶79-80. TLS pre_master_secrets, master_secrets, and key_blocks are all examples of a plaintext active key 172. *See, e.g., id.* ¶55; EX-1013 ¶356.

Alternatively, Tkacik teaches encrypting the plaintext active key 172 using the master key 160 (e.g., "security key"). *See* Tkacik ¶¶83-84. In Tkacik-TLS-1.2, the keys and related keying data generated during the client key exchange would have been stored in the secure memory (e.g., zeroizable read-write memory 52 or data register 76 (*see* Tkacik ¶54)) in an encrypted state using the security key. *See* [1f]; EX-1013 ¶¶357-359.

F. Claim 4

Tkacik-TLS-1.2 discloses claim 4 for the reasons in claim 3 and Section VII.F (**Grounds 1A-1C**). Encrypted pre-master secrets must be stored to be processed. *See* [1a-1]; Section VII.F; TLS-1.2, 64. Tkacik-TLS-1.2 would have stored the encrypted

(e.g., “wrapped”) pre-master key in Tkacik’s secure memory. *See* [1a-1]; EX-1013 ¶¶360-361.

Tkacik’s secure memory (e.g., zeroizable read-write memory 52, active key register 72, or data register 76, etc.) would have stored the encrypted pre-master key when calculating it, when including it in the encrypted premaster key message, and when it is sent to the server during the TLS handshake. *See* Tkacik ¶¶7, 26; *see* [1f]; EX-1013 ¶362.

Tkacik also teaches or suggests storing wrapped keys in the secure memory when transferring encrypted keys out and into secure memory. Tkacik uses secure memory 52 for data storage and buffering tasks when “moving blocks of payload data into and out from secure processing section 30.” Tkacik ¶37; *see also id.* ¶¶84, 74; EX-1013 ¶363.

G. Claim 5

1. [5a]

Tkacik-TLS-1.2 satisfies this feature. *See* [1d]; Tkacik ¶¶79-80, 56; EX-1013 ¶¶364-365. Tkacik-TLS-1.2, as discussed in [1d], would have used Tkacik’s secure processing section 30 to implement TLS-1.2’s generating the session key as a function of the pre-master key according to the TLS Handshake. *See* [1d].

2. [5b]

TLS-1.2 discloses [5b] for the reasons in Section VII.G.2 (**Grounds 1A-1C**, [5b]); EX-1013 ¶¶366-370. Tkacik’s encryption engine 66 performs “cryptographic

algorithms, such as hashing algorithms.” Tkacik ¶52, Fig. 2. In Tkacik-TLS-1.2 during the TLS Handshake, it would have been necessary to calculate a hash function as a function of the client-generated and server random nonces. As discussed, TLS-1.2 discloses this feature in several ways. *See* Section VII.G.2.

H. Claim 6

1. [6a]

TLS-1.2 discloses [6a] for the reasons in Section VII.H.1 (**Grounds 1A-1C**, [6a]); EX-1013 ¶¶371-377. TLS-1.2’s methods would have been implemented by Tkacik. *See* Tkacik ¶¶79-80.

2. [6b]

TLS-1.2 discloses [6b] for the reasons in Section VII.H.2 (**Grounds 1A-1C**, [6b]); EX-1013 ¶¶378-381. Tkacik’s security key, a device-specific key, would have been used for signing the public Diffie-Hellman client key so the server could verify the client device. *See* Tkacik ¶46.

I. Claim 7

TLS-1.2 discloses claim 7 for the reasons in Section VII.I (**Grounds 1A-1C**, claim 7); EX-1013 ¶¶382-388.

J. Claim 8

TLS-1.2 discloses claim 8 for the reasons in Section VII.J (**Grounds 1A-1C**, claim 8). Tkacik-TLS-1.2 would have implemented the Diffie-Hellman key exchange. EX-1013 ¶¶390-394. In the combination, Tkacik’s secure processing

section 30 would generate the session key using the TLS/SSL Handshake, such that it must receive the server key exchange message, which would have involved using Tkacik's security engine to receive Diffie-Hellman global values (e.g., dh_p and dh_g) and a public Diffie-Hellman server key (e.g., DH_Ys). *See* [1d]; Section VII.J; *see* Tkacik ¶¶79-80. A POSITA would have found it obvious to store at least one of the Diffie-Hellman global values and the public Diffie-Hellman server key in Tkacik's secure memory. *See* TLS-1.2, 1, 4, 34, 59, 92-93; Section VII.J; Tkacik ¶7; *see also id.* ¶26. Storing these essential security parameters in Tkacik's secure memory (including, e.g., memory 52) would have prevented attackers from obtaining them and for computing the secrets and encryption keys. EX-1013 ¶394.

K. Claims 9-15

<u>Claim Element</u>	<u>Corresponding Disclosure</u>
[9a]	[1p]-[1a-3]
[9b]	[1b]
[9c]	[1c]
[9d]	[1d]
[9e]	[1a-2], [1a-3], [1e]

[9f]	[1a-2], [1f]
[9g]	[1g]
[10a]	[1b]
[10b]	[1b], claim 2
11	[1d], claim 3
12	Claim 4
13	[1d], claim 3, [5a]
14	[1d], [5b]
[15a]	[1d], [6a]
[15b]	[6b]
[15c]	Claim 7
[15d]	Claim 7

L. Claim 16

1. [16a]

Tkacik-TLS-1.2 satisfies this feature. *See* claim [5a], [5b]; Section VII.L.1; EX-1013 ¶¶412-413.

2. [16b]

Tkacik-TLS-1.2 satisfies this feature. TLS-1.2, 62; Section VII.L.2; Tkacik ¶46; EX-1013 ¶¶414-417.

3. [16c]

Tkacik-TLS-1.2 satisfies this feature. TLS-1.2, 62; *see* [5b], [16a]; EX-1013 ¶¶418.

M. Claim 17

1. [17p]

Tkacik-TLS-1.2 satisfies this feature. *See* [1p]-[1a-3]; EX-1013 ¶419.

2. [17a]-[17g]

<u>Claim Element</u>	<u>Corresponding Disclosure</u>
[17a]	[1p]-[1a-3], [1b]
[17b]	[1b]
[17c]	[1c]
[17d]	[1d]

[17e]	[1p]-[1a-3], [1e]
[17f]	[1a-2], [1f]
[17g]	[1f]

N. Claims 18-23

<u>Claim Element</u>	<u>Corresponding Disclosure</u>
[18a]	[10a], [1b]
[18b]	[10b], claim 2, [1b]
19	Claim 3
20	Claim 4
21	[1d], [5a]
22	[1d], [5b]
[23a]	[1d], [6a]
[23b]	[6b]
[23c]	Claim 7

[23d]	Claim 7
-------	---------

O. Claim 24

Tkacik-TLS-1.2 satisfies this feature. *See* [1p]-[1a-3]; EX-1013 ¶¶437-438. Bus 24 (an “interconnect”) of the secure processing system 12 (the claimed “system-on-a-chip”) communicatively couples secure processing section 30 to the programmable processor 26, consistent with the ’838 patent. EX-1001, 3:67-4:3.

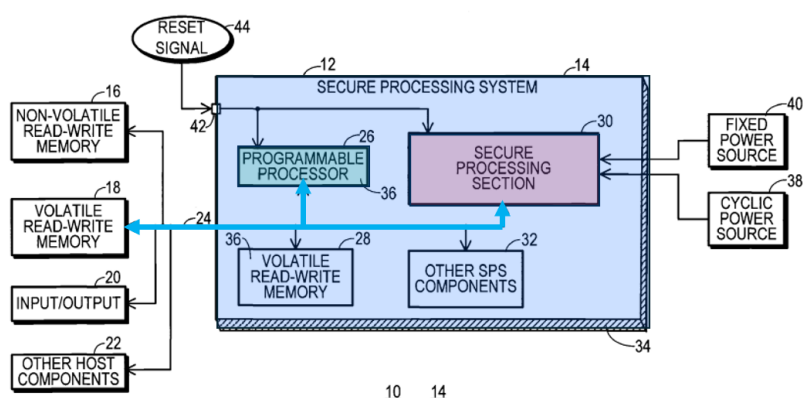


FIG. 1

Tkacik, Fig. 1 (annotated).

IX. The Board Should Institute Review

The *Fintiv* Factors support institution. Petitioner presents a compelling unpatentability challenge under Factor 6, Factors 3 and 4 favor institution, and Factors 1, 2, and 5 are, at worst, neutral.

Under Factor 6, Petitioner relies on materially different and noncumulative references not applied during prosecution of the ’838 patent. Factor 3 favors

institution because Petitioner diligently filed the Petition before any significant parallel investment. Patent Owner served infringement contentions two months ago (September 4, 2024), fact discovery just opened (October 9, 2024) but no discovery requests have been served, and Petitioner has not yet served invalidity contentions (November 20, 2024). A *Markman* hearing will not be held until July 31, 2025. EX-1024, 2; *Apple Inc. v. Fintiv Inc.*, IPR2020-00019, Paper 11, 11 (P.T.A.B. Mar. 20, 2020) (precedential) (“[I]t is often reasonable for a petitioner to wait to file its petition until it learns which claims are being asserted against it in the parallel proceeding.”); *Coolit Sys., Inc. v. Asetek Danmark A/S*, IPR2021-01195, Paper 10 at 11, 14 (P.T.A.B. Dec. 28, 2021) (petitioner acted diligently where it petition filed within five months after PO provided infringement contentions). Factor 4 favors institution because the Petition challenges claims 2, 7, 8, and 16, which are not involved in the litigation. *See* EX-1016, 29.

For Factors 1, 2, and 5, Patent Owner filed a Complaint against Petitioner on April 8, 2024. EX-1015. No stay has been requested and trial is scheduled for January 2026. While a Final Written Decision is anticipated by May 2026, balancing all factors, the Board should institute review because the date of trial is but one factor that should be balanced against the other factors. *See e.g., Netnut Ltd., v. Bright Data Ltd.*, IPR2021-01492, Paper 12 at 9-16 (P.T.A.B. March 21, 2022) (instituting review with trial date six months before FWD deadline; no stipulation); *Coolit*, Paper 10 at

11, 14 (same by five months; *Sand II* stipulation); *Micron Technology, Inc., v. Vervain, LLC*, IPR2021-01550, Paper 11 at 9-13 (P.T.A.B. April 11, 2022) (same by approximately three months). The trial date is also not determinative here because, in EDTX, multiple trials are commonly scheduled on the same date and trial dates are therefore subject to change.

X. MANDATORY NOTICES UNDER 37 C.F.R. § 42.8

A. Real Party-in-Interest

MediaTek Inc. is the real party-in-interest for this petition.

B. Related Matters

The '838 patent is, or has been, involved in the following proceedings:

Name	Number	Forum	Filed
<i>Daedalus Prime LLC v. Samsung Electronics Co., Ltd. et al.</i>	2-22-cv-00354	E.D. Tex.	Sept. 12, 2022
<i>Samsung Electronics Co., Ltd. et al v. Daedalus Prime LLC</i>	IPR2023-00727	P.T.A.B.	Mar. 20, 2023
<i>Daedalus Prime LLC v. MediaTek Inc.</i>	2:24-cv-00235	E.D. Tex.	Apr. 8, 2024

C. Lead and Backup Counsel Information

Petitioner provides the following designation of counsel:

Lead Counsel	Back-Up Counsel
Cory C. Bell (Reg. No. 75,096) cory.bell@finnegan.com Finnegan, Henderson, Farabow, Garrett & Dunner, LLP Two Seaport Lane, 6th Floor Boston, MA 02210-2001	J. Preston Long, Ph.D. (Reg. No. 65,125) jp.long@finnegan.com Finnegan, Henderson, Farabow, Garrett & Dunner, LLP 901 New York Avenue, NW Washington, D.C. 20001

Tel: 617-646-1641 Fax: 617-646-1666	Tel: 202-408-4347 Fax: 202-408-4400 Alexander M. Boyer (Reg. No. 66,599) alexander.boyer@finnegan.com Finnegan, Henderson, Farabow, Garrett & Dunner, LLP 1875 Explorer Street, Suite 800 Reston, VA 20190-6023 Tel: 571-203-2700 Fax: 202-408-4400
--	--

MediaTek Inc. has granted Power of Attorney to Practitioners at Finnegan, Henderson, Farabow, Garrett & Dunner, LLP. Petitioner consents to service by email at the addresses listed above.

XI. CONCLUSION

The Board should grant IPR and find all challenged claims unpatentable.

Dated: November 15, 2024

By: /Cory Bell/
Cory Bell (Reg. No. 75,096)
Lead Counsel

CERTIFICATE OF COMPLIANCE

Pursuant to 37 C.F.R. § 42.24(a)(1)(i), the undersigned hereby certifies that the foregoing **Petition for *Inter Partes* Review** contains 13,917 words, excluding parts of this Petition exempted under § 42.24(a), as measured by the word-processing system used to prepare this paper.

Dated: November 15, 2024

By: /Cory Bell/
Cory Bell (Reg. No. 75,096)
Counsel for Petitioner

CERTIFICATE OF SERVICE

Pursuant to 37 C.F.R. §§ 42.6(e) and 42.105(a), the undersigned certifies that on November 15, 2024, a copy of the foregoing **Petition for *Inter Partes* Review, the associated power of attorney, and Exhibits 1001-1025** were served by FedEx Priority Overnight on the correspondence address of record indicated in the Patent Office's Patent Center website for U.S. Patent No. 9,887,838:

Brian L. Burdick
Barnes & Thornburg LLP (Intel)
11 S. Meridian Street
Indianapolis, IN 46204

Date: November 15, 2024

By: /William Esper/
William Esper
Legal Assistant and PTAB Coordinator
FINNEGAN, HENDERSON, FARABOW,
GARRETT & DUNNER, LLP