

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

**DECLARATION OF PAUL D. MARTIN, PH.D.
IN SUPPORT OF PETITION FOR INTER PARTES REVIEW OF
U.S. PATENT NO. 9,887,838**

TABLE OF CONTENTS

I.	Introduction	1
II.	Qualifications and Background	2
	A. Teaching	2
	B. Research	3
	C. Industry Experience	7
III.	Materials Considered.....	10
IV.	Legal Standards	12
	A. Claim Construction	12
	B. Obviousness Under 35 U.S.C. § 103.....	13
	C. Secondary Considerations of Non-Obviousness	15
V.	The '838 Patent.....	17
	A. Overview of the '838 patent.....	17
	B. Technical Background	19
	C. Summary of the Prosecution History of the '838 patent.....	25
	D. Claim Construction of Terms in the '838 patent.....	25
VI.	Person of Ordinary Skill in the Art.....	26
VII.	The Prior Art.....	27
	A. Overview of Bugiel (EX-1002).....	27
	B. Overview of Chou (EX-1003).....	28
	C. Overview of TLS-1.2 (EX-1004).....	30
	D. Overview of Tkacik (EX-1005)	31
VIII.	Opinions on Unpatentability.....	32

A.	Grounds 1A, 1B, and 1C Render Obvious Claims 1-24	33
1.	Motivation to Combine Bugiel, Chou, and TLS-1.2	33
2.	Motivation to Combine Bugiel-Chou with Tkacik	41
3.	Claim 1	45
4.	Claim 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.”	105
5.	Claim 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.”	111
6.	Claim 4: “The system-on-a-chip apparatus of claim 3 wherein the security engine is further to wrap the pre-master key using a server public key and store the wrapped pre-master key in the secure memory.”	117
7.	Claim 5	120
8.	Claim 6	124
9.	Claim 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.”	128
10.	Claim 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.”	131
11.	Claim 9.....	133
12.	Claim 10.....	134
13.	Claim 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.”	134
14.	Claim 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-n-a-chip.”	135
15.	Claim 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.”	135
16.	Claim 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 nonce received from the remote server.”	135
17.	Claim 15.....	135
18.	Claim 16.....	136
19.	Claim 17.....	138
20.	Claim 18.....	140
21.	Claim 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.”	140
22.	Claim 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 publickey, and store the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip.”	140
23.	Claim 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.”	141
24.	Claim 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.”	141
25.	Claim 23.....	141
26.	Claim 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.”	142
B.	Grounds 2A, 2B: Tkacik (EX-1002) and TLS-1.2, and Tkacik, TLS-1.2, and Bugiel (EX-1002) Render Obvious Claims 1-24.....	143
1.	Motivation to Combine Tkacik and TLS-1.2.....	143
2.	Motivation to Combine Tkacik and TLS-1.2 with Bugiel.....	147

3.	Claim 1.....	151
4.	Claim 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.”	186
5.	Claim 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”	187
6.	Claim 4: “The system-on-a-chip apparatus of claim 3 wherein the security engine is further to wrap the pre-master key using a server public key and store the wrapped pre-master key in the secure memory.”	192
7.	Claim 5.....	194
8.	Claim 6.....	197
9.	Claim 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 pblic Diffie-Hellman server key.”	201
10.	Claim 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.”	204
11.	Claim 9.....	207
12.	Claim 10.....	208

13.	Claim 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.”	208
14.	Claim 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.”	209
15.	Claim 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.”	209
16.	Claim 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 nonce received from the remote server.”	209
17.	Claim 15.....	209
18.	Claim 16.....	210
19.	Claim 17.....	212
20.	Claim 18.....	213
21.	Claim 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.”	214

22.	Claim 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 publickey, and store the wrapped pre-master key in the secure memory of the security engine of the system-on-a-chip.”	214
23.	Claim 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.”	214
24.	Claim 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.”	215
25.	Claim 23	215
26.	Claim 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.”	216
IX.	Secondary Considerations	217
X.	Conclusion	217

I. INTRODUCTION

1. I, Paul Martin, submit this declaration to state my opinions on the matters described below.

2. I have been retained by MediaTek Inc. (“Petitioner”) as an independent expert in this proceeding before the United States Patent and Trademark Office. The company for which I work, Harbor Experts, is being compensated at the rate of \$675 per hour for time that I spend on this case. Harbor Experts is also being reimbursed for my reasonable and customary expenses. No part of my compensation depends on the outcome of this proceeding, and I have no other interest in this proceeding.

3. I understand that this proceeding involves U.S. Patent No. 9,887,838 (“the ’838 patent,” EX-1001), and I have been asked to provide my opinions as to the patentability of the claims of the ’838 patent.

4. I have been asked to consider the validity of certain claims of the ’838 patent based on certain prior art references. I have also been asked to consider the state of the art and prior art available as of December 15, 2011, which is the earliest claimed priority date on the face of the ’838 patent. Based on the combination of prior art references discussed in this declaration, it is my opinion that claims 1-24 of the ’838 patent are unpatentable for the reasons provided below.

II. QUALIFICATIONS AND BACKGROUND

5. I am well qualified to serve as a technical expert in this matter based upon my educational and work experience, which I summarize below.

6. I am the Chief Scientist at Harbor Experts, Inc. and a part-time Lecturer in the Department of Computer Science at Johns Hopkins University. I hold B.S., M.S.E., and Ph.D. degrees from Johns Hopkins University, with all degrees, including my doctorate, being in computer science. My current curriculum vitae (CV) is attached to this declaration as EX-1019.

A. Teaching

7. I am a part-time lecturer in the Department of Computer Science at Johns Hopkins University. I currently teach classes on computer and network security and applied cryptography. In the past I have taught classes on the topic of hardware hacking, including analyzing, modifying, and repairing computer hardware and embedded devices. I also occasionally substitute teach classes on network security and cryptography.

8. My classes cover a diverse array of topics including hardware and software security design, analysis and reverse engineering for vulnerability assessment; hardware and software-based attacks on computer components including RAM and CPUs; applied cryptography; computer architecture; computer networking; and component-level repair and modification. My classes also utilize

hardware and software emulation and virtualization techniques in order to present students with a variety of real-world environments in which to learn and perform hands-on projects.

B. Research

9. In 2011, I completed my Bachelor's degree in computer science at Johns Hopkins University. That year I began working towards my Ph.D. in computer science, also at Johns Hopkins University, and I began researching computer security and privacy systems in order to both design novel technologies for securing computing systems and to also bridge the interface gap between users and their technology. As such, my research focused not only on developing technical solutions to novel security and privacy systems but to also presenting these solutions in a way that non-technical users could understand.

10. Part of my research also focused on non-deterministically generating virtualized computer networks in order to provide malware education environments for students. I developed an emulator for analyzing malware code segments as part of a student project with the NSA. This emulator was integrated into an IDA Pro plugin and was able to observe how binary instruction snippets taken from malware samples modified their computing environment.

11. During my tenure as a Ph.D. student at Johns Hopkins, I was fully funded as a research assistant and/or teaching assistant throughout all of my

semesters. I was a member of the Upsilon Pi Epsilon computer science honor society, as well as its treasurer for the 2014-2015 school year. I was also a member of the Johns Hopkins chapter of the Association for Computing Machinery. I earned an award for outstanding teaching assistant in the Computer Science department for the year 2014-2015. I co-instructed a short course called, "Introduction to Hardware Hacking," with a colleague, Dr. Michael Rushanan, which was the highest-rated course in the computer science department (based on student reviews) during the winter session in which it was offered. In this course, we offered lessons on a variety of topics including modifying game consoles and device firmware; electronics repair; binary analysis and modification, network traffic analysis; and web-based vulnerability assessment and exploitation.

12. At Johns Hopkins University, I served the department as the faculty liaison czar for several semesters. I also mentored and supervised a high school student on a year-long student project in software design and I supervised a variety of undergraduate research assistants on some of my research projects.

13. I finished my Ph.D. by successfully defending my dissertation, entitled "Securing Medical Devices and Protecting Patient Privacy in the Technological Age of Healthcare," on February 12, 2016. My doctoral thesis tells the story of a secure healthcare practice of the future in which technology is seamless to use for healthcare providers while providing a much higher standard of security than is typical today.

It focuses on the juxtaposition of usability and security with an emphasis on simplicity, automation, and error-proofing of security controls.

14. I have produced, peer-reviewed, and published research in areas such as fingerprinting, anomaly detection, multifactor authentication, and embedded systems security. I focused not only on designing novel security systems but also on designing usable web-based security systems for controlling them and understanding their data output. Some of my research has formed the basis for commercial products and services. For example, in my work on integrated audit and access control, which was funded in part by Accenture Labs, I developed a Hadoop-based application to perform large-scale statistical analysis of audit logs from an electronic medical record system. As part of this work, I also designed a web application to automatically produce human-readable reports and graphs for use in consulting. Accenture subsequently patented this technology.

15. Similarly, in my research at Applied Communication Sciences in 2013, I designed and implemented a web-based traffic visualization dashboard and analysis system for field area smart grid networks that could be used to quickly gain an understanding of the current state of a SmartGrid network as well as to detect unexpected anomalies in the network. Applied Communication Sciences subsequently patented this work and continued to build on the project. To my

knowledge, they sold and/or still sell this product as part of their SecureSmart Managed Security Service product offering.

16. I have worked on and published two research projects related to improving the quality and ease-of-use of authentication technologies, especially in healthcare settings. In one project, I and my co-authors designed an indoor location tracking system consisting of unspoofable Bluetooth Low Energy beacons. These beacons were placed around a building and mapped to a backend such that a user could report which beacons he or she was within range of in order to be used as a secondary authentication mechanism for accessing patient medical records. In the way that we designed and envisioned the system, a doctor would log into a mobile device which would connect to a web-based backend. As the doctor walked past patient rooms, he or she would be automatically presented with medical records of nearby patients. In another project, I and my co-authors designed an authentication bracelet that would receive a Kerberos ticket upon login to a modified computer terminal through use of low-energy electrical signals transmitted over the wearer's skin. Upon use of other terminals throughout the hospital, this Kerberos ticket could be sent back over the wearer's skin to the custom contact on the terminal in order to allow the user to login without a password. The bracelet was also designed to immediately lose the cryptographic secret upon being removed from the user

17. My recent research has focused primarily on embedded systems security with an emphasis on binary analysis, anomaly detection and automated security enforcement. In one case, I designed a security system that can be soldered directly to the CPU of embedded devices in order to perform control-flow integrity for purposes of profile building and enforcement. In another case, I designed a system to automatically discern name and version information from binaries on embedded devices in order to build a software profile of the platform configuration of the device which can then be cross-referenced with a vulnerability database. I have also written a patent on this specific technique.

18. I am a member of the program committee for the IEEE Symposium on Security and Privacy. As a member of this committee, I peer review research that has been submitted for publication at the Symposium. I have recently reviewed research on memory forensics, side channel attacks, password vaults, differential privacy, computational theory, memory module design, and hardware security.

19. I am a named inventor on six patents. A full listing of these patents can be found on my CV.

C. Industry Experience

20. I began working in the software industry in 2008, when I obtained my first independent consulting client—Brandeis University. I created a program for their computer repair shop's ticketing system to interface with a label printer. This

enabled the computer repair shop to automatically print labels associated with tickets in order to easily identify customers' devices.

21. Shortly after finishing my engagement with Brandeis University, I began working for the Johns Hopkins University Digital Research and Curation Center as a student programmer. There, I created a batch importer for a digital archiving system called DSpace in order to enable hundreds of digitally archived works per week to automatically be added to the repository. I next ported a web interface for statistics tracking and visualization to the same digital archiving system in order to allow library staff to more easily visualize and understand data about how the digital archiving system was being used through a common web-based interface.

22. In 2009, I began working for a security consulting company called Independent Security Evaluators in Baltimore, Maryland. I worked on a wide variety of security and privacy projects including projects to test and break the DRM scheme of a magazine distribution application for IOS and Android (at the behest of the developer of said application), projects to test implementation code for cryptographic secret splitting, projects to assist with fuzz testing for security vulnerabilities of a variety of software applications, software development projects to automate testing of antivirus and antimalware solutions, and a variety of other security-related projects. I designed automation systems to create and restore virtualized snapshots of Windows machines on which I tested antivirus software on

actual malware samples as part of a project for Consumer Reports. I worked for ISE as an intern throughout the semesters and summers until August 2011. In 2010, I also designed a security architecture for a large-scale digital curation system meant to be a major inter-university initiative to create a successor to existing digital curation systems that could be used for decades.

23. In spring 2011, I completed my bachelor's in computer science at Johns Hopkins University, and I also retained an independent consulting client, the University of Michigan, for whom I performed a penetration test and security assessment of a cloud-based research system that they planned to deploy.

24. In February 2016, I joined Harbor Labs full-time as a research scientist. In January 2019, I was promoted to Director of Firmware Analysis. In December 2023, I was promoted to Vice President of Applied Research and Technology. In October 2024, the division of Harbor Labs that I managed became Harbor Experts. I became Chief Scientist at Harbor Experts during this time. At Harbor Experts, I manage client engagements related to all of the analysis work that Harbor Labs performs. I am also lead Harbor Experts' research projects.

25. A substantial portion of my role at Harbor Experts is in managing source code review engagements. As part of this work, I have reviewed software systems of varying sizes, often totaling in the millions or billions of lines of code. I have reviewed products in the security space, television-based set top boxes,

network appliances, numerous web-based enterprise systems, email management systems, telephony products, embedded system bootloaders, social network platforms, virtualization platforms and countless other products.

26. Prior to joining Harbor Experts, I was the technical and development lead for a firmware security analysis engine for a product developed at Harbor Labs called Firmware IQ. Firmware IQ is designed to analyze the firmware of embedded devices for security vulnerabilities in an automated fashion. In one configuration of the product, a developer uploads a firmware image to a web-based portal which uses a broker to forward the firmware to an engine for analysis. The engine unpacks the firmware, breaks it into its constituent components, and then performs more than a hundred automated security checks in order to find vulnerabilities in the firmware image. The results are then reported through JSON to a web-based presentation system which displays them in an interactive fashion.

27. I believe that I am well qualified to serve as a technical expert in this matter based upon my educational and work experience. My CV contains further details about my education, experience, publications, patents, and other qualifications to render an expert opinion, which I understand is being submitted as EX-1019.

III. MATERIALS CONSIDERED

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

Exhibit	Description
EX-1001	U.S. Patent No. 9,887,838 (“the ’838 patent”)
EX-1002	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-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-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

IV. LEGAL STANDARDS

29. In considering the subject matter of the '838 patent and its claims in light of the prior art and forming my opinions, I am relying on certain legal principles that counsel in this case explained to me. My understanding of these concepts is summarized below.

30. I understand that the claims define the invention. I also understand that an unpatentability analysis is a two-step process. First, the claims of the patent are construed to determine their meaning and scope. Second, after the claims are construed, the content of the prior art is compared to the construed claims.

31. I understand that an invention, as defined by the claims of the patent, must not be anticipated or rendered obvious by the prior art.

A. Claim Construction

32. I understand that the United States Patent and Trademark Office interprets claim terms in an *inter partes* review proceeding under the same claim construction standard that is used in a United States federal court. I understand that under this standard, the meaning of claim terms is considered from the viewpoint of a person of ordinary skill in the art ("POSITA") at the time of the alleged invention.

33. I have been informed that claim terms are generally given their ordinary and customary meaning as understood by one of ordinary skill in the art in light of the specification and the prosecution history pertaining to the patent. I understand,

however, that claim terms are generally not limited by the embodiments described in the specification.

34. I understand that in addition to the claims, specification, and prosecution history, other evidence may be considered to ascertain the meaning of claim terms, including textbooks, encyclopedias, articles, and dictionaries. I have been informed that this other evidence is often less significant and less reliable than the claims, specification, and prosecution history.

B. Obviousness Under 35 U.S.C. § 103

35. I understand that a patent claim is invalid as obvious if the claimed invention would have been obvious to a person of ordinary skill in the art at the time the claimed invention was made. This means that even if all of the limitations of the claim cannot be found in a single prior art reference that would anticipate the claim, a person of ordinary skill in the art who knows about all the prior art would have come up with the claimed invention. I understand that in an obviousness determination, the person of ordinary skill in the art is presumed to have knowledge of all material prior art. I understand that whether a claim is obvious is based upon the determination of several factual issues.

36. In considering obviousness, I understand that one must determine the scope and content of the prior art. I understand that, in order to be considered prior art to a patent, a reference must be reasonably related to the claimed invention of

that patent. A reference is reasonably related if it is in the same field as the claimed invention or is from another field to which a person of ordinary skill in the art would look to solve a known problem.

37. I understand that one must determine what differences, if any, existed between the claimed invention and the prior art.

38. I understand that a patent claim composed of several limitations is not proved obvious merely by demonstrating that each of its limitations was independently known in the prior art. In evaluating whether such a claim would have been obvious, one may consider whether a reason has been identified that would have prompted a person of ordinary skill in the art to combine the elements or concepts from the prior art in the same way as in the claimed invention.

39. I understand that one may consider whether (1) the change is merely the predictable result of using prior art elements according to their known functions or the result of true inventiveness; (2) there is some teaching or suggestion in the prior art to make the modification or combination of limitations claimed in the patent; (3) the innovation applies a known technique that had been used to improve a similar device or method in a similar way; (4) the claimed invention would have been obvious to try, meaning that the claimed innovation is one of a relatively small number of possible approaches to the problem that could have been used by those of ordinary skill in the art with a reasonable expectation of success; (5) the invention

merely substitutes one known element for another known element in order to obtain predictable results; (6) the invention merely applies a known technique to a known device, method, or product to yield predictable results; or (7) known work in the field may have prompted variations of use of the same inventions in the same or different fields due to market forces or design incentives that would have been predictable to a person of ordinary skill in the art.

40. I understand that one must be careful not to determine obviousness using the benefit of hindsight; many true inventions might seem obvious after the fact. I understand that the determination should be based on the position of a person of ordinary skill in the art at the time the claimed invention was made and should not reflect what is known today or what is learned from the teaching of the patent.

C. Secondary Considerations of Non-Obviousness

41. I understand that certain secondary considerations may be considered in evaluating obviousness in order to prevent hindsight bias. These secondary considerations include the commercial success of products that practice the patent, long-felt need for the patented technology, failure by others to solve the problem addressed by the patent, initial skepticism by others in the industry, industry recognition and praise of the patented products, and efforts by others to copy the patented technology.

42. I have been informed that these secondary considerations of non-obviousness must be balanced against the strength of the *prima facie* case of obviousness. Where the invention represents no more than the predictable use of prior art elements according to their established functions, secondary considerations are inadequate to establish non-obviousness.

43. I have also been informed that the patentee must establish a nexus between any secondary considerations and the claimed invention. Where a secondary consideration results from something other than what is both claimed and novel in the patent, there is no nexus to the merits of the claimed invention.

44. I have been informed that, for example, the nexus requirement as it applies to evidence of commercial success requires that the patentee must show that the sales of a product that allegedly practices the claimed invention must be a direct result of the unique characteristics of the claimed invention, as opposed to other economic and commercial factors that are unrelated to the patented invention. I understand this to mean that if the product's commercial success is due to an element that existed in the prior art or that is not claimed by the patent, then no nexus exists.

45. I have been further informed that, for example, the nexus requirement as it applies to the long-felt need element must likewise be supported by evidence. Where the differences between the prior art and the claimed invention are minimal,

it cannot be said that any long-felt need was met by the patented invention or that the patent solved any technological problems that were unaddressed by others.

46. I have been informed that evidence of industry praise must also show a nexus with the claimed invention, and that self-serving statements by the patentee regarding the merits of the claimed invention are generally not given weight.

V. THE '838 PATENT

A. Overview of the '838 patent

47. The '838 patent discloses an apparatus and method for secure communication over a network using a client device. EX-1001, 1:5-32, 3:24-59, Fig. 1. The patent discloses that the client device with a system-on-a-chip (SOC) that includes a security engine 110. *Id.*, 3:24-4:16. The security engine 110 may include a security key 150 that is “provisioned during the manufacturing of the SOC 112.” *Id.*, 4:16-34.

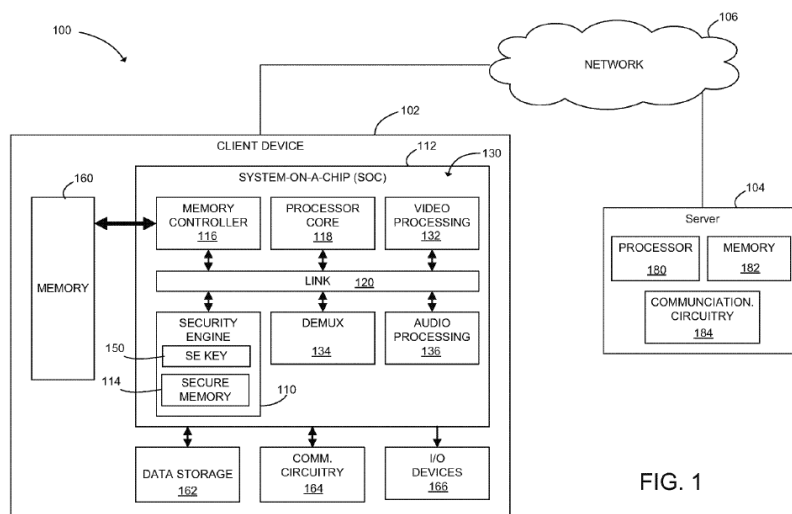


FIG. 1

Id., Fig. 1.

48. The '838 patent discloses the above hardware for conducting secure communications with a remote server using Secure Sockets Layer (SSL) protocol and the related Transport Layer Security (TLS) protocol. *Id.*, 1:7-32. Each of the steps of Figure 3 matches the known SSL/TLS handshake. *See id.*, Fig. 3. This includes a Client Hello 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.

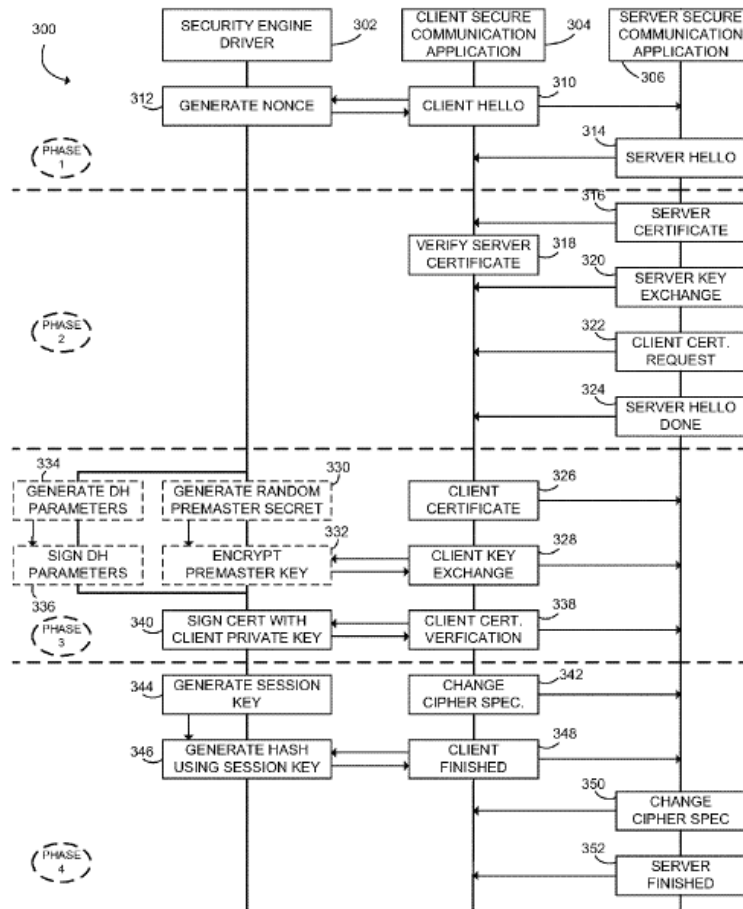


FIG. 3

Id., Fig. 3.

49. The SSL/TLS protocol was not new as of the '838 patent. Instead, it was a well-known and widely used protocol for secure communications. A POSITA would have understood that SSL/TLS was one of the most popular secure communication protocols at the relevant time. As one textbook described before the '838 patent, “[s]upport for SSL is now built in to nearly all browsers and Web servers.” EX-1006, 6. Indeed, the patent acknowledges this protocol was commonly used in “many different types of secure communication sessions including, for example, secure web-browsing, ecommerce, secure upgrades, and other secure communication sessions between two computing devices on an untrusted network such as the Internet.” EX-1001, 1:5-32.

B. Technical Background

50. The '838 patent relates to encryption using the SSL/TLS protocol. The following is a technical discussion of secure key encryption used in SSL/TLS.

51. Cryptography is a process that transforms information or data into a coded format to protect it from unauthorized access. The goal is to enable the encrypted contents to be deciphered only by those with the appropriate key. Cryptography is widely used in communication over networks and is commonly used to protect sensitive information. This is particularly important in networks like the Internet because messages passing over such networks are subject to potential inspection, interception, and modification by third parties.

52. Encryption works by using algorithms to transform the original message, making it unreadable to anyone who does not have the appropriate key. When the recipient receives the encrypted message, they use the proper key to decrypt the encrypted message back to the original message and read its contents. There are two forms of cryptography: symmetric and asymmetric. Symmetric cryptography, also known as symmetric key cryptography and shared secret cryptography, uses the same key for both encryption and decryption. The sender and receiver of the encrypted message both have access to the same key, and was one of the earliest forms of encryption used to protect sensitive information. Asymmetric cryptography, also known as public key cryptography, in contrast to symmetric cryptography, uses two different (yet complimentary) keys, one for encryption and the other for decryption.

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

54. Asymmetric, or public key, cryptography, in contrast to symmetric cryptography, uses two different yet complimentary keys for encryption and decryption. One key is used to encrypt the message and the other key is used to decrypt the message. This eliminates the need for a shared secret, solving the key exchange problem of symmetric key cryptography. On the other hand, public key cryptography algorithms are much more computationally difficult than symmetric key cryptography algorithms. As a result, public key cryptography is also used to distribute symmetric keys which are then used to encrypt the rest of the communication. This process is called hybrid encryption.

55. RSA is a common public key cryptographic algorithm that is widely recognized as one of the most important contributions to the development of secure communication methods over the Internet. RSA uses mathematical algorithms to generate two keys: a public key and a private key. The public key is used to encrypt the message, and the private key is used to decrypt the message. This allows secure communication without the need for a shared secret, as the private key is kept secret by the recipient of the message, thus a party can ensure that a message can only be decrypted and read by specific entity by encrypting the message with the party's public key.

56. The Diffie-Hellman key exchange algorithm was another early algorithm designed to solve the key distribution problem in cryptography, which is

the challenge of securely sharing secret keys between two parties who have not met.

The Diffie-Hellman algorithm eliminated the need for parties to exchange a common key by physical means by allowing two parties to securely exchange keys over an insecure channel, such as the Internet. The Diffie-Hellman algorithm uses a public key to establish a shared secret between the two parties, which can then be used as a symmetric key to encrypt and decrypt messages.

57. The basic idea of Diffie-Hellman key exchange is that two parties can each generate public-private key pairs, and then exchange their public keys. They can then use their private keys and the other party's public key to derive a shared secret key. Because only the public keys are exchanged over the insecure channel and the private keys are never exchanged, an eavesdropper cannot determine the shared secret key, even if they could intercept the public keys.

58. Transport Layer Security (TLS) is a protocol that provides secure communication over the Internet. The TLS protocol is not itself a cryptographic algorithm. Instead, the TLS protocol provides the framework and mechanisms for secure communication, including the negotiation of cryptographic algorithms, the exchange of cryptographic keys, and the verification of the authenticity of the communicating parties. The actual encryption of data is performed using one or more cryptographic algorithms that are agreed upon during the TLS handshake. The encryption keys used for this cryptographic process are generated and exchanged

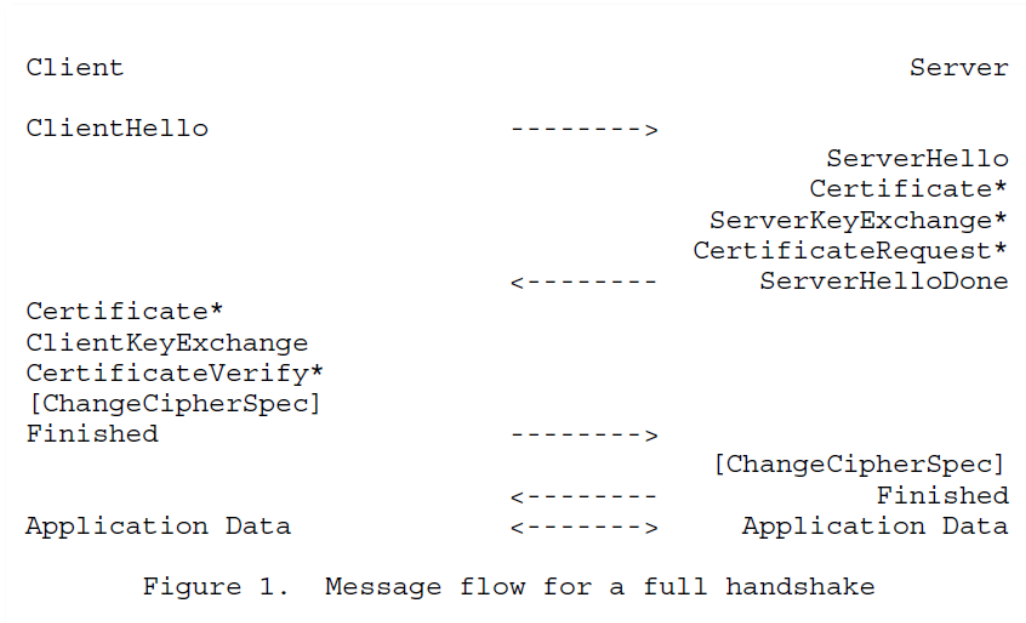
during the TLS handshake and are used to encrypt and decrypt data transmitted between the communicating parties.

59. The predecessor to TLS is called the Secure Sockets Layer (SSL) protocol. The SSL protocol was designed to provide a secure, encrypted connection between a client and a server, allowing sensitive information to be transmitted securely over the Internet. The primary use case for SSL was to secure online transactions, such as e-commerce, online banking, and other activities that involved the transmission of sensitive information, such as credit card numbers, social security numbers, and other personal data.

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

61. In 1999, the Internet Engineering Task Force (IETF) took over the development of SSL and renamed it TLS (Transport Layer Security). Since then, several versions of TLS have been released, each with increased security and improved performance. (IETF) “renamed SSL to Transport Layer Security (TLS),” but “[d]espite the change of names, TLS is nothing more than a new version of SSL.” *See* EX-1006, 6. The TLS Handshake Protocol is the process that establishes a secure

communication session between a client and a server. The steps of the TLS handshake include the following:



TLS-1.2, Fig. 1.

62. The TLS Handshake Protocol uses cryptographic algorithms and digital signatures to establish a secure communication channel between the client and server. The process ensures that the communication is protected from eavesdropping, tampering, and impersonation, and that the authenticity of the client and server can be verified.

63. As discussed above, the '838 patent discloses and claims performing the steps of the TLS Handshake Protocol. Therefore, I will discuss the particular steps of the Protocol in my analysis of the claims.

C. Summary of the Prosecution History of the '838 patent

64. I have reviewed the prosecution history of the application that led to the '838 patent. *See* EX-1012. During prosecution, the claims were rejected based on obviousness in four office actions. *Id.*, 272-290, 214-232, 144-161, 86-107. In response, the claims were amended to recite, among other things, that the security engine is “separate from a processor core of the system-on-a-chip,” *id.*, 70-76, the secure communication session is with a “remote” server, *id.*, 200-206, and “the secure memory includes a security key that was encoded in the secure memory during a manufacturing process of the system-on-a-chip,” *id.*, 256-262.

D. Claim Construction of Terms in the '838 patent

65. As discussed above, I have been informed that, for purposes of *inter partes* review, the standard for construction of terms in a patent's claims is the same as that applied in federal district court litigation. I understand that a district court has yet to construe the claim terms of this patent. At this time, I do not see the need to construe any terms of the challenged claims for the purposes of this proceeding. Accordingly, in preparing my analysis I have applied the ordinary and customary meaning that each claim term would have to a person of ordinary skill in the art in light of the '838 patent's specification and prosecution history. If any claim terms are differently construed, I reserve the right to reassess my opinions under those constructions.

VI. PERSON OF ORDINARY SKILL IN THE ART

66. I have been informed that patentability must be analyzed from the perspective of a “person of ordinary skill in the art” in the same field as the patent-in-suit at the time of the invention. I also understand that several factors are considered in assessing the level of ordinary skill in the art, including (1) the types of problems encountered in the art; (2) the prior art solutions to those problems; (3) the rapidity with which innovations are made; (4) the sophistication of the technology; and (5) the education level of active workers in the field.

67. I refer to a person of ordinary skill in the art here as a “POSITA.” A POSITA at the time of the invention would have had a minimum of a bachelor’s degree in electrical engineering, computer science, computer engineer, or a similar field, and approximately two years of industry or academic experience in a field related to cryptography and secure communications between network devices. Work experience can substitute for formal education and additional formal education can substitute for work experience. I at least meet this definition of a POSITA, now and as of the priority date of the ’838 patent.

68. I am also familiar with the knowledge of a POSITA as of the priority date of the ’838 patent, as I have taught and interacted with hundreds of such engineers both in industry and academia. I am able to opine on how a POSITA would have understood the disclosure and claims of the ’838 patent, the disclosures of the

prior art, the various motivations to combine the prior art, and what combinations would have been obvious to a POSITA.

VII. THE PRIOR ART

A. Overview of Bugiel (EX-1002)

69. Bugiel discloses methods of secure web authentication between mobile devices and servers. Bugiel, 219. Bugiel describes connecting to web server with mobile phones and describes the need for a “hardware-supported secure trusted execution environment (TrEE)” that “allows the secure and isolated execution of certain code, and which can also protect certain amount of data persistently.” *Id.*, 220. In one application, Bugiel discloses “TruWalletM,” a “hardware-assisted wallet-based web authentication as a mechanism to protect user credentials from malware, phishing and physical attacks on mobile platforms.” *Id.*, 220. For secure communication, Bugiel discloses utilizing the standard SSL/TLS protocol. *See id.*, 223, 225-226 (“All messages which are exchanged between WalletHelper and WebServer are standard messages of SSL/TLS handshake and can be found in the corresponding specification [16].”) (citing TLS-1.2).

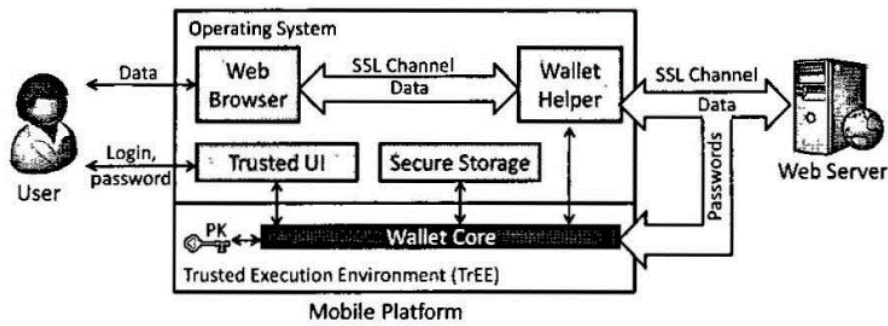


Fig. 1. TruWalletM Architecture

Id., Fig. 1.

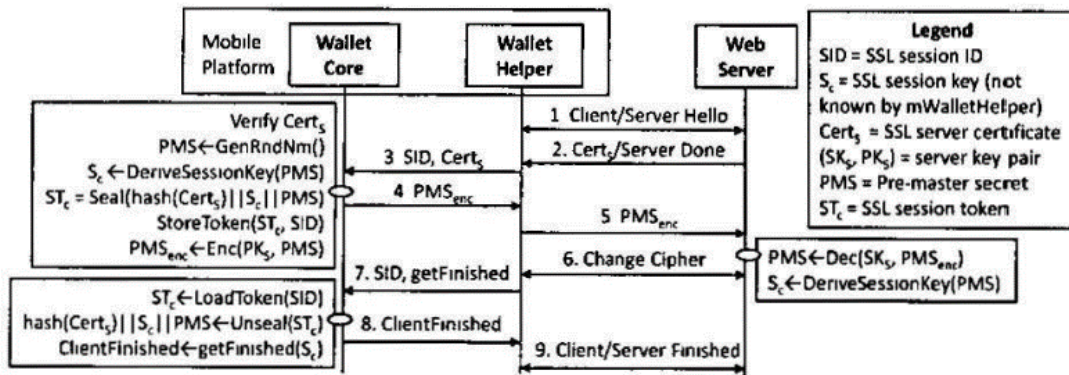


Fig. 2. SSL/TLS handshake

Id., Fig. 2 (Bugiel implementing the SSL/TLS handshake with a remote server).

B. Overview of Chou (EX-1003)

70. Chou discloses “[a] technique to provide a hardware security module that provides a secure boundary for retention of a secure key within the secure boundary and prevention of unauthorized accesses from external sources outside of the secure boundary to obtain the secure key.” Chou, Abstract. Chou’s hardware includes a security processor “to unwrap and authenticate a secure key within the secure boundary to decrypt or encrypt data and to provide data through a single

interface that communicates with external sources, so that all data transfers between the secure boundary, formed by the hardware security module, and external sources are transferred only through the interface.” *Id.* Chou hardware includes a hardware security module (HSM) 20 forming the secure domain. *Id.* ¶¶18-23, 30-35, Figs. 1, 3.

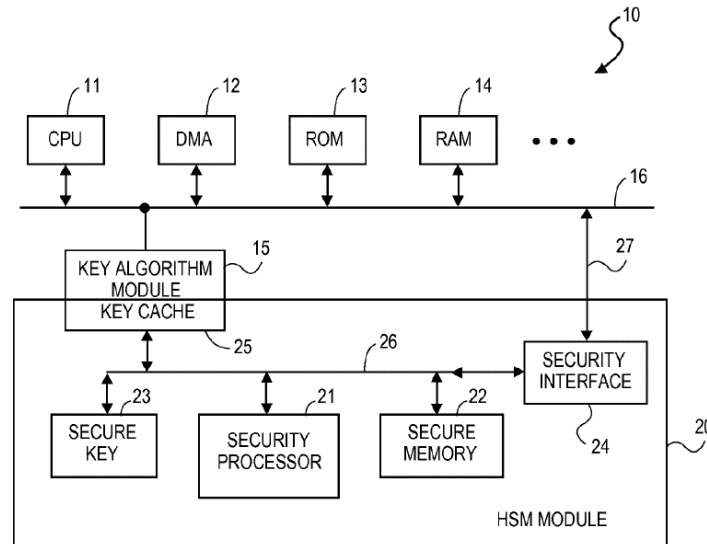


FIG. 1

Id., Fig. 1.

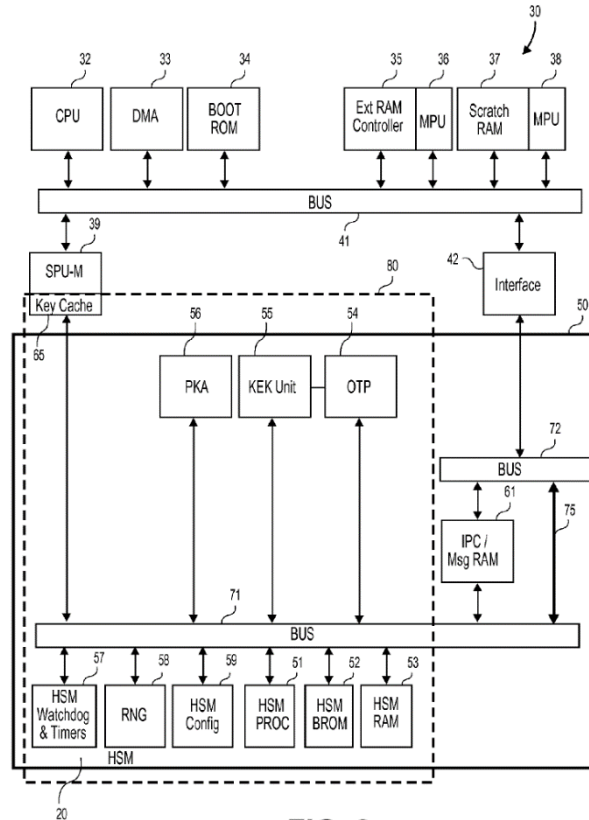


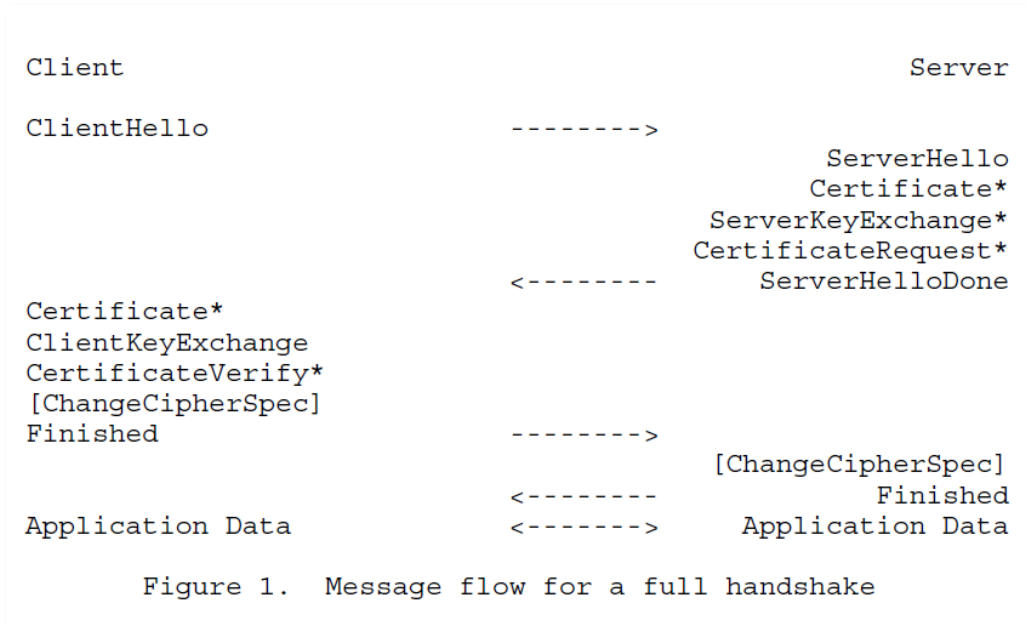
FIG. 3

Id., Fig. 3.

C. Overview of TLS-1.2 (EX-1004)

71. TLS-1.2 discloses the Transport Layer Security (TLS) Protocol Version 1.2. TLS-1.2, 1. TLS-1.2 discloses a protocol that “provide[] communications security over the Internet,” where the protocol “allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.” *Id.* The protocol includes two layers, the TLS Record Protocol (e.g., TCP [TCP]), and the TLS Handshake. *Id.*, 4. Two parties are authenticated using “asymmetric, or public key, cryptography.” After

authentication, the parties negotiate a shared secret. *Id.*, 4-5. The TLS Handshake is shown below in Figure 1.



TLS-1.2, Fig. 1.

D. Overview of Tkacik (EX-1005)

72. Tkacik discloses an encryption apparatus that includes a secure processing section 30. Tkacik, Abstract. The secure processing section 30 provides a secure execution environment (SEE) and stores a plaintext master key that is “configured to have a long life.” *Id.*, Abstract, ¶¶20-26, 44-46. Tkacik discloses that “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.* ¶46.

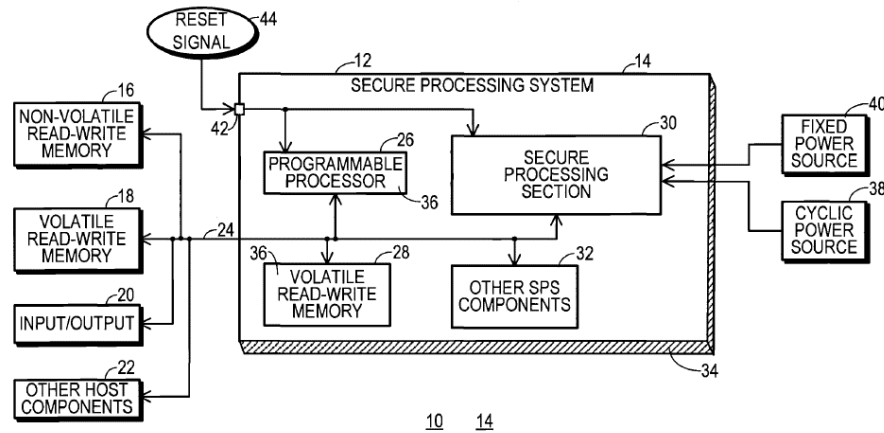


FIG. 1

Id., Fig. 1.

VIII. OPINIONS ON UNPATENTABILITY

In the analysis that follows, I identify the following prior art and combinations of prior art that, in my opinion, render obvious the claims of the '838 patent:

Ground	Prior Art	Basis	Claims
1A	Bugiel (EX-1002) and Chou (EX-1003)	§ 103	1-24
1B	Bugiel (EX-1002), Chou (EX-1003), and Tkacik (EX-1005)	§ 103	1-24
1C	Bugiel (EX-1002), Chou (EX-1003), TLS-1.2 (EX-1004), optionally with Tkacik (EX-1005)	§ 103	1-24
2A	Tkacik (EX-1005) and TLS-1.2 (EX-1004)	§ 103	1-24

2B	Tkacik (EX-1005), TLS-1.2 (EX-1004), and Bugiel (EX-1002)	§ 103	1-24
----	--	-------	------

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

1. Motivation to Combine Bugiel, Chou, and TLS-1.2

73. A POSITA would have found it obvious to combine Bugiel (EX-1002) and Chou (EX-1003) (**Ground 1A**) and Bugiel and Chou with TLS-1.2 (EX-1004) (**Ground 1C**).

74. Bugiel discloses a method of secure communication between a client device and a server using any “secure hardware for mobile devices” that provides “a Trusted Execution Environment (TrEE) to protect the execution of critical code from tampering.” Bugiel, 220; *see also id.*, 234 (“Our solution does not rely on a secure operating system, but exploits hardware security features, in particular Trusted Execution Environment (TrEE) available on many modern mobile phones.”). Bugiel contemplates and expressly incorporates by reference the SSL/TLS handshake for encrypted communications with the server. *Id.*, 225 (“All messages which are exchanged between WalletHelper and WebServer are standard messages of SSL/TLS handshake and can be found in the corresponding [TLS-1.2] specification [16].”) (citing TLS-1.2); *see also id.*, 224-227, 230-231, 234-235.

75. Chou describes secure hardware for mobile devices that are completed by Bugiel. For example, Chou disclose HSM (Hardware Security Module) 20 that

establishes a secure area within a system-on-a-chip SOC 91, providing a secure zone for a mobile device. *See* Chou ¶18 (“[A] HSM 20 to set a secure domain, zone or boundary to protect a secure key. The term ‘secure’ (such as secure domain, zone or boundary) is used herein to designate a zone controlled by HSM 20. Thus, components outside of the zone established by HSM 20 reside in a non-secure zone relative to HSM 20.”); *see also id.* ¶¶27, 36, 68, Figs. 1, 3. Chou’s hardware and secure HSM is the type Bugiel contemplates using as its “Trusted Execution Environment (TrEE).” *See* Bugiel, 220 (“[M]any modern mobile phones are already equipped with a hardware-supported secure trusted execution environment (TrEE) which allows the secure and isolated execution of certain code, and which can also protect certain amount of data persistently.”); *see also id.*, 222-225. A POSITA thus would have found it obvious to implement Bugiel’s system using hardware that includes Chou’s HSM 20.

76. A POSITA would have understood that both Chou and Bugiel relate to secure communication using cryptographic keys for encryption and decryption. As explained above, Bugiel discloses utilizing SSL/TLS protocol “[f]or secure communication between the device and the remote web server” and explicitly cites TLS-1.2 in describing its use of the SSL/TLS handshake. *See* Bugiel, 222, 225; *see also id.*, 224-227, 230-231, 234-235. Chou similarly describes secure communications in which “secure data transfer is utilized to perform secure financial

transactions over a network, such as the Internet.” Chou ¶6. Chou explains that, “[i]n a typical secure transaction, data content that contains financial information, such as a credit/debit card information, are encrypted and transmitted to a designated recipient through an unsecure network,” and “[t]he recipient of the secure data utilizes a secure key to decrypt the data to retrieve the financial information.” *Id.*; *see also id.* ¶17 (“The embodiments of the present invention may be practiced in a variety of settings that utilize a secure key to authenticate and unwrap and/or decrypt data, context data and/or a secure key.”). Chou explains that implementations of its hardware may be deployed in mobile phones, where “secure key operation[s]” use the “secure zone boundary [that] is defined by components that make up a hardware security module (HSM).” *Id.* ¶¶66-69.

77. SSL/TLS was one of the most popular secure communication protocols at the relevant time. *See* TLS-1.2 at 1 (“The TLS protocol provides communications security over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.”). As was known before the ’838 patent, “[s]upport for SSL is now built in to nearly all browsers and Web servers.” EX-1006, 6.¹ Given the express

¹While EX-1006 describes TLS 1.0, where relevant I cite this text to describe certain features of the TLS Handshake in TLS-1.2.

teachings in Bugiel and the ubiquity of the SSL/TLS communication protocol, a POSITA would have been motivated to review the relevant SSL/TLS protocol for more details regarding how to implement the TLS Handshake Protocol for generating cryptographic keys based on cryptographic key exchange.

78. A POSITA also would have understood the SSL/TLS processes contemplated by Bugiel and TLS-1.2 require hardware components. This includes, for example, the following hardware:

- a client device with baseband and transceivers for communication with a remote server, *see* Bugiel, 221 (“We consider application scenarios where a user deploys his mobile device to access services provided by remote web servers (over the Internet).”);
- secure storage and isolated secure execution environment, *see* Bugiel, 222 (“Assumptions. We rely on availability of a secure hardware which provides a Trusted Execution Environment (TrEE) with following features: (i) isolated secure code execution, (ii) secure storage, (iii) integrity protection of secure execution environment”), 232 (“[T]he TrEE is isolated from the rest of the system in terms of processing and memory.”);
- hardware for generating random nonces for ClientHello/ServerHello messages to initiate the cryptographic key

exchange, *see* TLS-1.2, 33-37 (generating client and server random values used as seed for SSL/TLS handshake protocol);

- hardware for implementing Diffie-Hellman and/or RSA key generation algorithms, which are the two designated cryptographic key exchanges algorithms for generating encryption session keys in SSL/TLS handshake protocol, *see* TLS-1.2, 64-67 (sections 8.1.1 and 8.1.2);
- hardware for performing hashing and signing algorithms for certifying client certificates, *see* TLS-1.2, 45-47, 50-53 (hash and signature algorithms, such as DSA with SHA-1);
- hardware for performing symmetric encryption using the generated encryption keys, *see* TLS-1.2, 4 (“Symmetric cryptography is used for data encryption (e.g., AES [AES], RC4 [SCH], etc.).”); and
- hardware for encrypting a session key with a key that is never available outside TrEE; *see* Bugiel, 230-231.

79. A POSITA would have known Chou provides exemplary hardware for carrying out these functions and would have combined Chou with Bugiel and/or TLS-1.2. For example, Chou explains that its SOC 91 includes components that are capable of performing the above functions required by Bugiel and TLS-1.2, including:

- SOC 91 that can be implemented in wireless device 90 used to transmit and receive wireless communication, where the “wireless device 90 includes a baseband processor 93 and radio 94” (which includes transceivers), Chou ¶66, Fig. 5;
- an HSM 20 of SOC 91 that sets a secure zone that separates HSM processing from CPU 11/32, *see id.* ¶¶18, 34, Figs. 1, 3;
- secure memories, such as secure key module 21, secure memory 22, OTP 54, key cache 65, HSM ROM 52, HSM RAM 53, etc., that are not accessible by components outside the secure zone, *see id.* ¶¶54-55, 63, Figs. 1, 3;
- a RNG (random number generator) module 58 within HSM 20 to generate random values “as a seed for a cryptographic algorithm,” *id.* ¶¶34, 41-42;
- a PKA (Public Key Accelerator) module 56, which is an accelerator operable for the cryptographic key exchange algorithms in TLS version 1.2, such as Diffie-Hellman public value and shared key computation and RSA key generation, *see id.* ¶¶43-53;
- a PKA module that is also operable for signing and hashing algorithms like “1024-bit DSA signature with SHA-1 hash or 2048-bit DSA signature with SHA-256 hash,” *see id.* ¶45;

- a Key Algorithm module/SPU-M (Secure Processing Unit for Mobile processors) 39 that operates as a hardware accelerator for symmetric key algorithms, “such as AES, DES, RC4, SHA1-HMAC, SHA2-HMAC, MD5-HMAC.” etc., for encrypting/decrypting messages using key from key cache 65, *see id.* ¶¶54-55; and
- a KEK (key-encryption key) unit 55, which wraps and unwraps keys using a root key programmed in one-time-programmable memory OTP 54, *see id.* ¶40.

80. Because Chou teaches an apparatus with hardware components that satisfy the requirements specified by Bugiel and TLS-1.2, a POSITA would have found it obvious to implement the techniques described in Bugiel, including TLS-1.2 as required by Bugiel, using Chou’s SOC apparatus. The combination of Bugiel, Chou, and optionally TLS-1.2 alleged here incorporates Chou’s hardware to implement Bugiel’s methods, which includes implementing the SSL/TLS Handshake Protocol to establish a secure TLS communication session. Specifically, Chou’s HSM 20 and SPU-M 39/Key Algorithm Module 15 generate and use keys based on the SSL handshake protocol. *See* Chou ¶¶42-55, 18, 66.

81. A POSITA would have understood that the combination of Bugiel and Chou, and Bugiel and TLS-1.2 with Chou, 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. This would have yielded predictable results in establishing secure, encrypted communication between a client mobile device and a server. A POSITA would have had an expectation of success in the result, as Bugiel contemplates using TLS-1.2 communication protocols and secure hardware to store protected keys and to encrypt and decrypt data, as taught by Chou.

82. The combination also would have been nothing more than using a known technique (e.g., Bugiel's and TLS-1.2's techniques for secure communications using mobile devices) to improve similar devices (e.g., Chou's flexible security hardware for mobile devices) in the same way (e.g., by 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. Indeed, the prior art suggests the combination by Bugiel's teaching the use of secure hardware for mobile devices to protect the execution of critical code from tampering while engaging in TLS-1.2-secured communication, thereby suggesting its combination with Chou's hardware.

83. A POSITA also would have looked to TLS-1.2 given Bugiel's teaching and suggestion to use this communication protocol. Given the ubiquity of the SSL/TLS protocol, a POSITA would have been motivated to review the relevant

SSL/TLS protocol, for example, to implement the Bugiel-Chou combinations using the protocol. This includes implementing the details of the TLS Handshake Protocol for generating cryptographic keys based on cryptographic key exchange. TLS-1.2 was the current version of the TLS/SSL standard at the time the '838 patent was filed, and it had been so for more than three years. 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.

2. Motivation to Combine Bugiel-Chou with Tkacik

84. A POSITA would have found it obvious to combine Bugiel (EX-1002) and Chou (EX-1003) with Tkacik (EX-1005) to implement Chou's secure memory with a security key that was encoded during a manufacturing process of the system-on-a-chip. As explained in [1a-3] below, Chou discloses programming a device unique key/root key in a one-time programmable memory OTP 54 and a POSITA would have found it obvious to program OTP 54 as a step in a manufacturing process, where Chou contemplates doing so for its connection 75. *See* Chou ¶¶64, 38, 40; *see* [1a-3]. It also would have been obvious to implement OTP 54 as memory on Chou's system-on-a-chip to be encoded with a security key during manufacturing, as disclosed by Tkacik.

85. For example, Tkacik discloses a long-lived key register 60 that is programmed with a plaintext master key 160 “during the manufacturing process of secure processing system 12 and cannot thereafter be altered.” Tkacik ¶46. Tkacik discloses exemplary processes for this, including burning “laser-scribed fuses . . . during the manufacture of integrated circuit 34 (FIG. 1) to permanently program plaintext master key 160 into long-lived key register,” *id.*, or alternatively, “electronically burn[ing]” electronic fuses to program plaintext master key 160,” *id.*

86. A POSITA would have understood that Tkacik’s non-volatile, read-only register storing long-lived key register 60 may be one-time programmable memory, just like Chou’s 54. As is known in the art, one-time programmable memory utilizes “fuses” to program the memory, which is permanent and irreversible. *See* EX-1009, 240-241 (“The original true ROM types are mask programmable ROM and ***fuse*** programmable ROM (or PROM),” where “fuse programmed ROM devices are programmed by ***blowing small fuses*** and hence are sometimes called One Time Programmable ROM or OTPROM”) (emphases added); *see also id.*, 246-247 (blowing “small fusible links which are made of nichrome or polycrystalline silicon”).

87. A POSITA would have combined Tkacik with Bugiel-Chou by programming the device unique key/root key in Chou’s OTP 54 using Tkacik’s burned laser-scribed fuse or electronically burned fuse manufacturing process. *See*

Chou ¶¶38, 40; Tkacik ¶46. For example, a POSITA would have understood that that Chou's OTP 54 is one-time programmable memory, just like Tkacik's long-lived key register 60, and that Chou's OTP 54 requires programming the unique key/root by blowing fuses. A POSITA would have found it obvious to program OTP 54 by burning "laser-scribed fuses" or electronically burning "electronic fuses" during manufacturing, as Tkacik teaches. Tkacik ¶46. Doing so would have programmed the Bugiel-Chou device from the factory with a security key that is unique to the device and cannot be changed. A POSITA would have been motivated to do so to ensure Chou's device is programmed for use when manufactured, thereby eliminating the later step of programming the OTP by the user and providing the user a ready-to-use device.

88. Combining Bugiel-Chou with Tkacik would have been nothing more than combining known methods of programming one-time programmable memory using blown fuses and applying this process during manufacturing of the memory, as taught by the prior art. For example, it was known to program one-time programmable memory in a device with keys during manufacturing, as taught by Tkacik and at least suggested by Chou. For example, Chou contemplates disabling connection 75 of its OTP 54 "by programming a non-volatile bit in OTP 54 *during manufacturing.*" Chou ¶64 (emphasis added). Chou thus teaches and suggests

programming its OTP 54 during manufacturing. Tkacik disclose this expressly. Tkacik ¶46.

89. Other prior art confirms programming OTP memory during manufacturing was well-known. *See* EX-1007 ¶¶7, 56 (“[W]hen the [trusted platform module] 302 is manufactured security routines may be used to generate secret keys that are unique to each TPM. These keys may then be stored in the OTP memory 332.”); EX-1008, 2:25-40, 4:45-53 (“KEK130 is stored in a non-volatile data memory device 134. In a symmetric key system the initial KEK 130 may be programmed into the non-volatile memory 134 when the system is manufactured. In an asymmetric key system, a private key and a certificate may be stored in the non-volatile memory 134 when the system is manufactured.”), 10:34-35 (“When the cryptographic accelerator 1026 is manufactured, a private key 1028 is stored in the EEPROM 1030.”), 17:5-22.

90. Applying this known manufacturing process to program Chou’s OTP 54 with a unique key/root key that is a “security key” (*see* [1a-3]) would have been nothing more than applying known methods of programming OTP memory to Chou, according to understood principles. The result would have been predictable in an OTP memory that is encoded, either by laser burning or electronic burning, during the manufacturing of the OTP memory of Chou. A POSITA would have had an expectation of success. Indeed, doing so would be nothing more than applying

known methods of programming one-time programmable memory to a step in the manufacturing process, using for example laser burning or electronic burning, as opposed to programming the device after is manufactured.

91. To the extent it is argued that Tkacik's non-volatile, read-only register is not one-time programmable 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.

3. Claim 1

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

92. The Bugiel-Chou combinations (**Grounds 1A-1C**) satisfy this feature. For example, Chou discloses embodiments in which "components of device 30 [are] implemented 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 ("HSM 20 is constructed on a single integrated circuit chip. In another embodiment, device 10, including HSM 20, is constructed on a single integrated circuit chip."). Chou further discloses that "FIG. 4 shows one such embodiment in which HSM 20 and other components of FIG. 1 or 3 are implemented as SOC 91. *Id.* ¶65, Figs. 4, 5. To the extent the claimed "system-on-a-chip apparatus" is understood to mean the apparatus itself needs to be a system-on-a-chip, Chou's

system-on-chip 91 discloses this feature as reflected in annotated Figures 4 and 5, below.

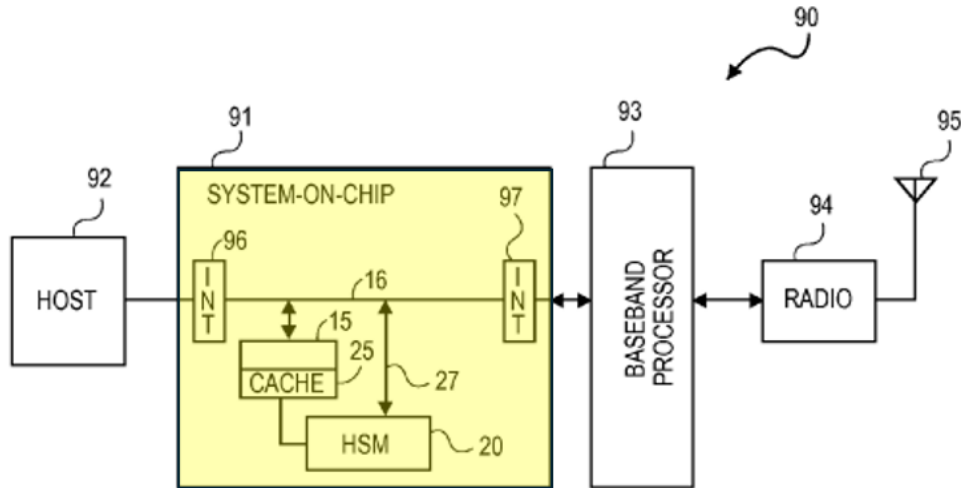


FIG. 4

Id., Fig. 4 (annotated).

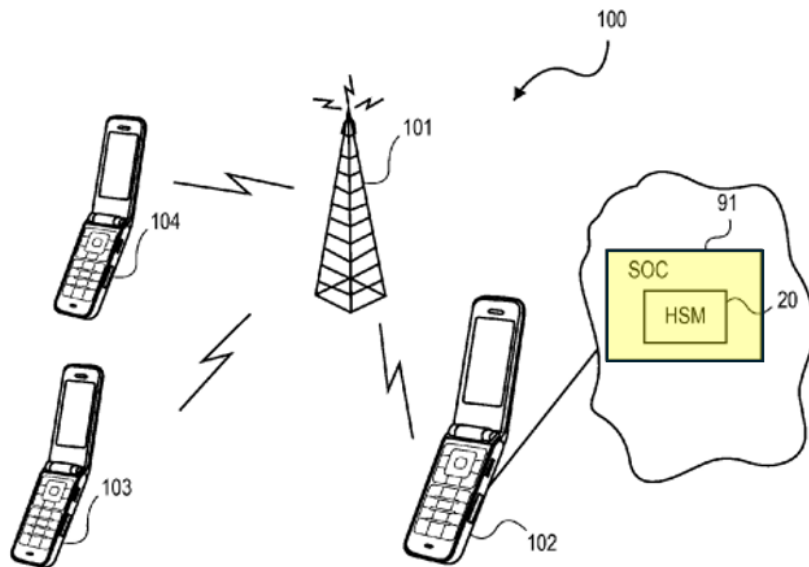


FIG. 5

Id., Fig. 5 (annotated).

93. Chou's "single integrated circuit (IC) chip 91 that incorporates a complete system on the IC chip (System-On-Chip or SOC)" is consistent with the '838 patent's description of a system-on-a-chip, which states that "[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. For example, an SOC may include a processor core, memory controller, video components, audio components, and/or communication components on a single chip." EX-1001, 1:27-32.

94. To the extent "system-on-a-chip apparatus" is understood to require a device *with* a system-on-a-chip, where the apparatus itself is not the system-on-a-chip, Chou discloses this too in "wireless device 90," as illustrated in Chou's Figure 4. Chou discloses that "[t]he particular implementation of SOC 91 in FIG. 4 below is a wireless device 90 that is used to transmit and receive wireless communication." Chou ¶66; *see also id.* ¶65 ("FIG. 4 shows one such embodiment in which HSM 20 and other components of FIG. 1 or 3 are implemented as SOC 91."). Wireless device 90 having SOC 91, as described and depicted with respect to Figure 4, discloses the claimed "system-on-a-chip apparatus." Mobile phone 102, which is also disclosed as including SOC 91 in Figure 5, also discloses the "system-on-a-chip apparatus." *Id.* ¶68 ("As shown in FIG. 5, device 90 [of Fig. 4] is implemented in a mobile phone 102.").

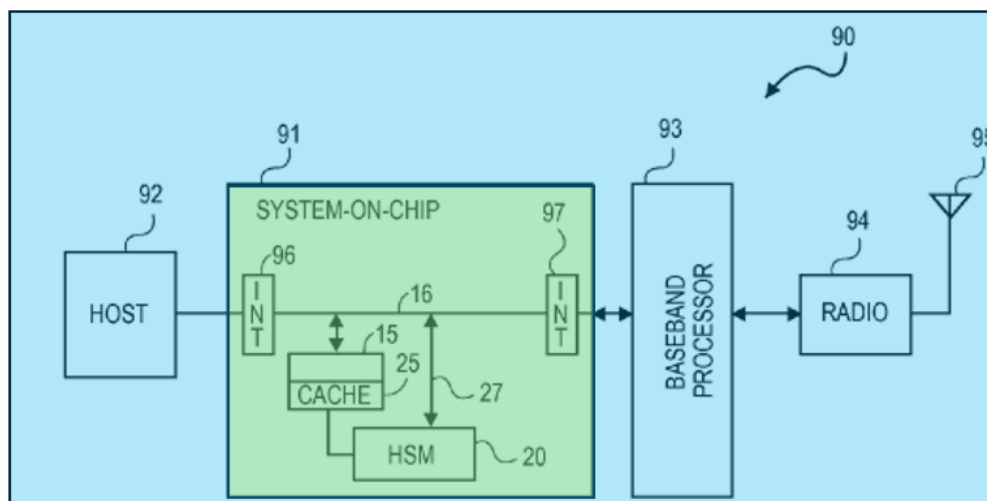


FIG. 4

Id., Fig. 4 (annotated).

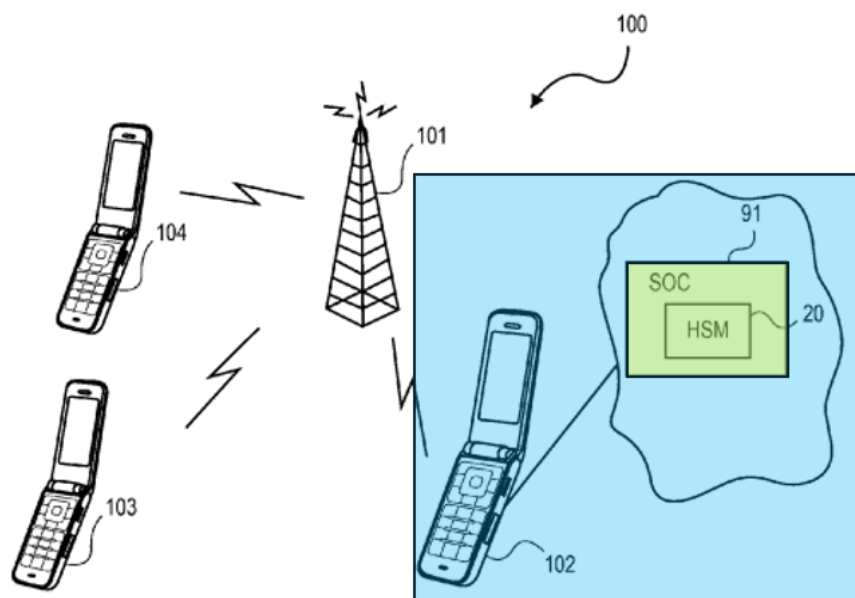


FIG. 5

Id., Fig. 5 (annotated).

b. [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”

95. The Bugiel-Chou combinations (**Grounds 1A-1C**) satisfy this feature. For example, Chou discloses that, with respect to SOC 91 of Figures 4 and 5, the system of Figure 4 “incorporates the HSM of FIG. 1 or 3 in a device that includes a radio portion for wireless communication.” Chou ¶15. Figure 5 “is an example that illustrates the use of the device of FIG. 4 in a mobile phone.” *Id.* ¶16. Figures 4 and 5 therefore include the features of HSM 20, as disclosed with respect to Figures 1 and 3.

96. In Figures 1 and 3, Chou discloses a “security engine.” For example, Chou discloses a hardware security module (HSM) 20. *Id.* ¶21, Fig. 1. HSM 20 includes “a security processor 21, secure memory 22, secure key module 23, secure bus 26 and a security interface 24.” *Id.* ¶21. Chou discloses that HSM 20 “set[s] a secure domain, zone or boundary to protect a secure key.” Chou ¶18 (“The term ‘secure’ (such as secure domain, zone or boundary) is used herein to designate a zone controlled by HSM 20.”); *see also id.* (“The term ‘key’ is used herein to describe any value, data, context, etc., which is used to change the format of the information (such as data) to provide security from unauthorized access.”).

97. The security processor 21 of HSM 20 “provides the various processing within HSM 20, including key authentication and the encrypting and/or decrypting

operation to encrypt and/or decrypt data.” *Id.* ¶21. This includes “secure operations on the decrypted data or, alternatively, provide secure operations on data obtained from the non-secure domain.” *Id.* ¶24. Secure key module 23 is “a storage medium, such as a memory, register, circuit, etc. that stores one or more key value(s) that is used by security processor 21 to encrypt and decrypt data.” *Id.* ¶21; *see also id.* ¶¶25-29 (describing functions of HSM 20).

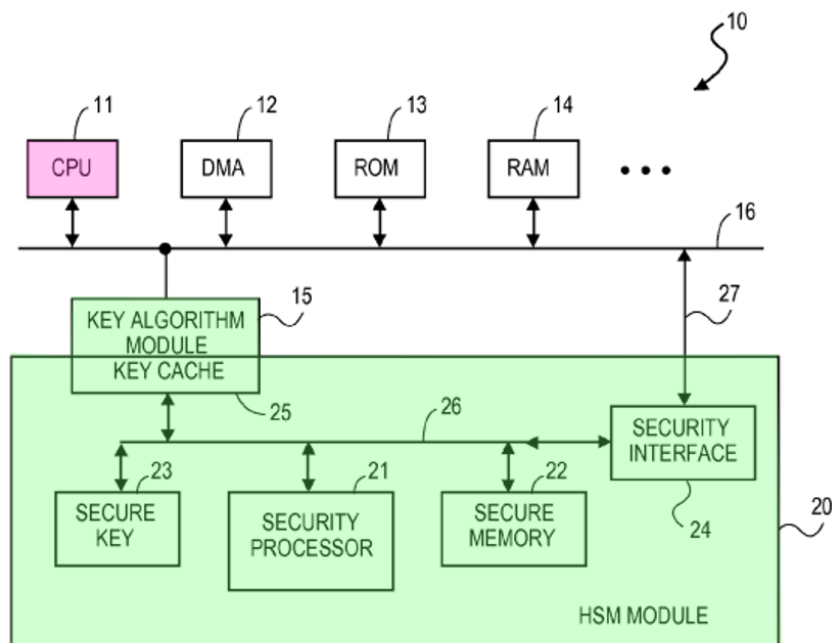
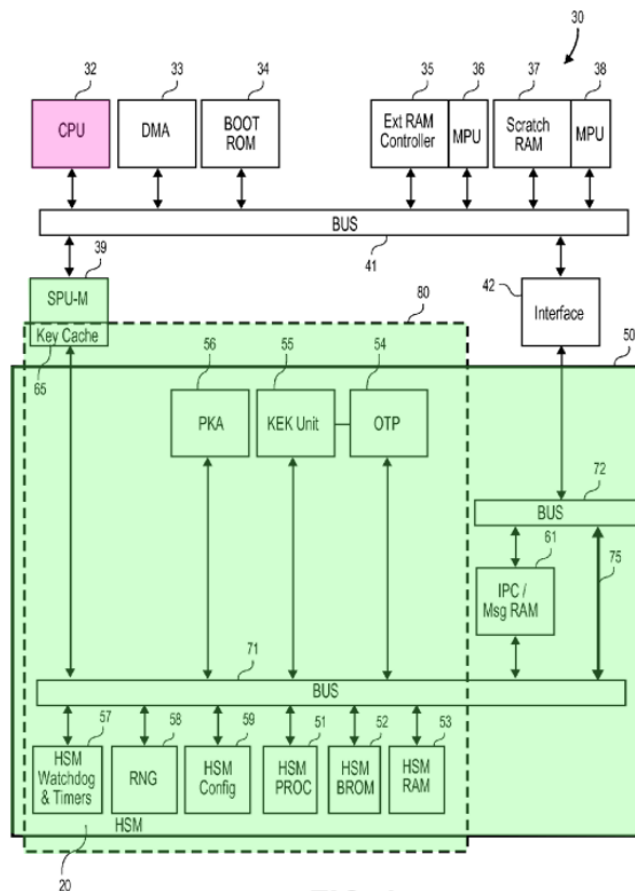


FIG. 1

Id., Fig. 1 (annotated).

98. With respect to Figure 3, Chou discloses a system that “utilizes a more detailed HSM than the HSM of FIG. 1.” *Id.* ¶¶30, 14, Fig. 3. In Figure 3, HSM 20 “includes a number of hardware and software components that encompass a secure domain boundary as shown in FIG. 3,” where “solid-lined box 50 represents the

boundary formed by the actual hardware components of HSM 20, while dashed-line box 80 represents the actual secure domain boundary that is not accessible by nonsecure components.” *Id.* ¶34, Fig. 3. Chou discloses that “[t]he main processing operations for HSM 20 [in Figure 3] are provided by HSM processor 51, which operates with its BROM 52 and RAM 53.” *Id.* ¶35. A POSITA would have understood that HSM 20 in Figures 1 and 3 of Chou describe features of the same hardware security module (HSM), where Figure 3 provides additional details regarding its hardware and operation. *See id.* ¶30, Figs. 1, 3.

**FIG. 3**

Id., Fig. 3 (annotated).

99. Chou also discloses SPU-M module 39 in Figure 3, which Chou explains “is equivalent to key algorithm module 15 of Fig. 1.” Chou ¶54; *see also id.* ¶26. Key algorithm module 15, and thus SPU-M module 39, “resides in the non-secure domain and contains one or more algorithms to encrypt/decrypt data.” *Id.* ¶26 (disclosing that the algorithms “are typically software routines for providing encryption/decryption.”). Equivalent SPU-M module 39 “operates as a hardware accelerator for symmetric key algorithms.” *Id.* ¶54; *see also id.* ¶¶55-62 (“SPU-M module 39 may implement many symmetric key algorithms such as AES, DES, RC4, SHA1-HMAC, SHA2-HMAC, MD5-HMAC, as well as others.”).

100. Within HSM 20, Chou further discloses secure memory for storing keys, including secure key module 23, secure memory 22, key cache 25 (Figure 1) and components 51-59, 65 (Figure 3), which Chou discloses as used for “wrapping and unwrapping of keys.” *Id.* ¶37 (describing components 51-59); *see also id.* ¶¶30, 34-37 (same); *id.* ¶¶21-22 (“Secure key module 23 is a storage medium, such as a memory, register, circuit, etc. that stores one or more key value(s) that is used by security processor 21 to encrypt and decrypt data. Secure memory 22 is utilized to store data that is processed and/or to be processed by security processor 21”), ¶24 (“secure keys stored in secure key module 23”), ¶25 (“unique key stored in secure key module 23”), ¶26 (“key algorithm module 15 includes a key cache 25, which resides within the secure domain”), ¶¶54-55 (“The algorithm(s) of SPU-M module

39 may access key cache 65, which contains the unwrapped keys, but the contents of key cache 65 are not retrievable by software running on non-secure components outside of secure domain 80”).

101. Chou’s HSM 20 and its SPU-M module 39 and key algorithm module 15 each disclose the claimed “security engine.” This is consistent with the description of this term in the ’838 patent. For example, the ’838 patent discloses that the “security engine 110 may be embodied as a security co-processor or processing circuitry separate from the 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.” *Id.*, 3:35-41, 4:17-34.

102. In addition to disclosing a “security engine,” Chou discloses that its HSM 20 and key algorithm module 15/SPU-M 39 (e.g., “security engine”) is separate from a processor core of the system-on-a-chip, as claimed. For example, Chou discloses CPU 11 and CPU 32 that “provide[] central processing for device [10 and]30” outside of the secure zone of HSM 20. *Id.* ¶31 (“Processor 32 is equivalent to CPU 11 of FIG. 1 in that processor 32 provides central processing for device 30.”), ¶20. CPU 11 “reside[s] **outside** of a boundary established by HSM 20 and, therefore, are regarded as residing in a **non-secure domain or zone**.” *Id.* ¶20 (emphasis added), Fig. 1. Chou discloses that the “components in the **non-secure**

domain or zone” are coupled to bus 16/41, and thus are separate from HSM 20. *Id.*

¶30 (emphasis added), Fig. 3. Indeed, Chou discloses that “processor 32 is operable to use several processor security levels of operation, . . . [h]owever, these processor security levels of operation *are separate* from the secure zone operation of HSM 20.” *Id.* ¶31.

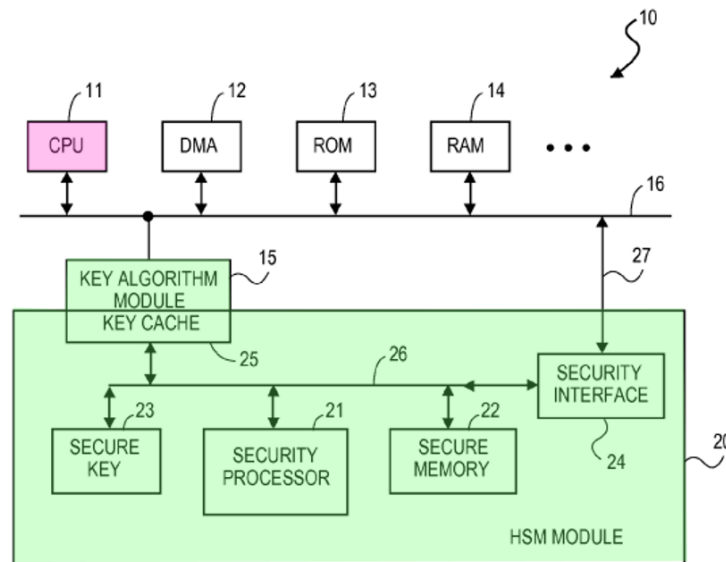


FIG. 1

Id., Fig. 1 (annotated).

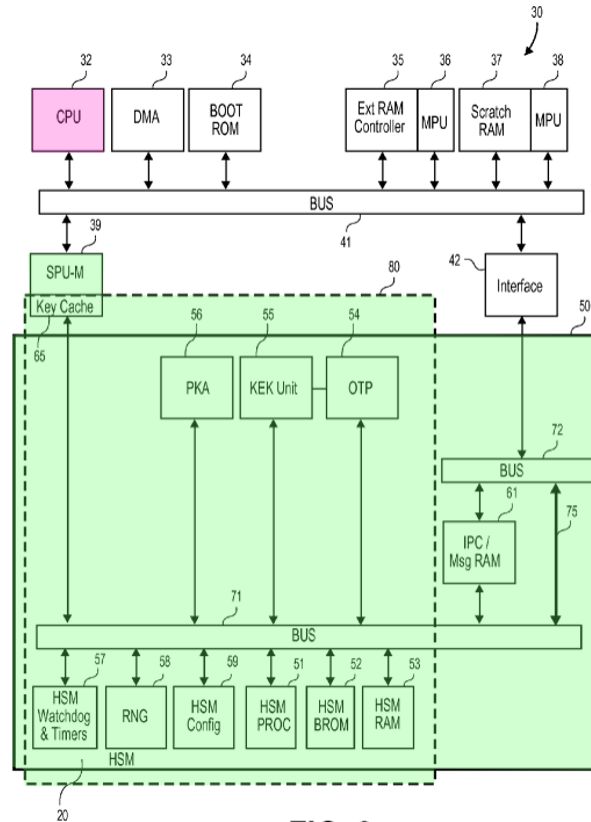


FIG. 3

Id., Fig. 3 (annotated).

103. By disclosing CPU 11 and CPU 32 in a non-secure zone, outside of the secure zone of HSM 20, where CPU 11/32 and HSM 20 reside on SOC 91, Chou discloses a security engine that is separate from a processor core of the system-on-a-chip. *See id.* ¶¶20-21, 31-32, Figs. 1, 3; *see also id.* ¶65 (“FIG. 4 shows one such embodiment in which HSM 20 and other components of FIG. 1 or 3 are implemented as SOC 91”), ¶68 (“As shown in FIG. 5, device 90 is implemented in a mobile phone 102, such as a cell phone.”).

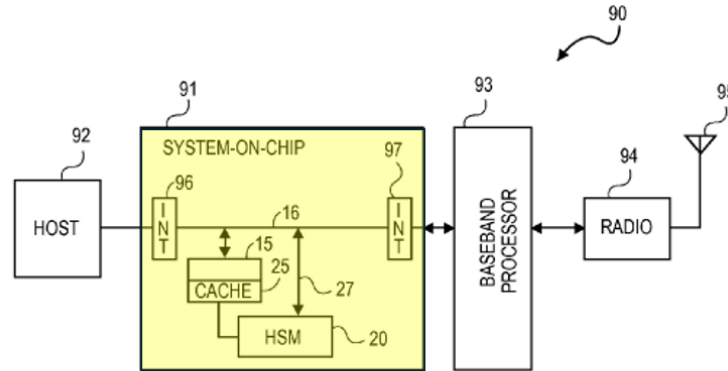


FIG. 4

Id., Fig. 4 (annotated).

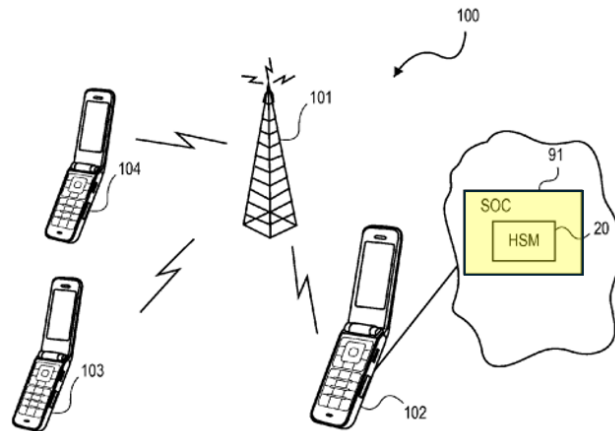


FIG. 5

Id., Fig. 5 (annotated).

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

104. The Bugiel-Chou combinations (**Grounds 1A-1C**) satisfy this feature. One of the assumptions underlying Bugiel’s TruWalletM method is “availability of a secure hardware which provides a Trusted Execution Environment (TrEE) with . . . (ii) secure storage.” Bugiel, 222, 232 (“(i) run-time isolation. All operations on

user credentials (except user input) are performed within the TrEE. As the TrEE is isolated from the rest of the system in terms of processing and memory, that guarantees protection of the user's credentials at run-time from potentially malicious OS and other OS-side components. (ii) secure storage.”). Bugiel discloses that “SecureStorage is a database where passwords and other security sensitive data are stored persistently and securely.” *Id.*, 224; *see also id.*, 224 (“Ideally, TrustedUI should reside within the TrEE”), 232-233 (“credentials associated with the SSL/TLS channel (such as a pre-master key and a session key) are protected by the TrEE”), 234-235 (“recognizing the trend of device manufacturers to further extend their devices’ TrEE capabilities, we argue that our proposed architecture is fundamentally a step into the right direction towards hardware secured user credentials on mobile end-user devices.”), 231 (“isolating the SecureStorage from other, unauthorized (and potentially malicious) script”), Fig. 1 (illustrating secure storage is only accessible by the WalletCore in the TrEE).

105. Therefore, Bugiel discloses or at least suggests implementing a secure memory accessible only by the TrEE in the secure execution environment. A POSITA would have found it obvious to implement the secure storage desired by Bugiel using Chou's flexible secure hardware. *See* Section VIII.A.1.

106. For example, Chou discloses a memory in secure memory 22, secure key module 23, key cache 25, as well as memories associated with the processing

inside HSM 20 (e.g., security processor 21) with respect to Figure 1. *See* Chou ¶¶20-26, Fig. 1. Chou also discloses a memory in One Time Programmed Memory (“OTP”) 54, HSM Boot ROM (“BROM”) 52, HSM RAM 53, and key cache 65, as well as memories associated with the secure processing inside HSM (e.g., HSM processor 51, PKA 56, KEK 55 and RNG 58) with respect to Figure 3. *Id.* ¶¶30-62, Fig. 3. 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,” because this is consistent with the ’838 patent. For example, the ’838 patent contemplates more than one physical memory when describing other memory, including “machine-readable medium,” memory 160, and memory 182. EX-1001, 2:35-46, 4:35-51.

107. For example, the ’838 patent discloses that “*a* machine-readable medium may be embodied as read only memory (ROM); random access memory (RAM); magnetic disk storage media; optical storage media; flash memory devices; mini- or micro-SD cards, memory sticks, electrical signals, *and* others.” *Id.*, 2:38-55 (emphases added). The patent further discloses that “[t]he system memory 160 may be embodied as any type of main memory or data storage *locations* including, for example, dynamic random access memory devices (DRAM), synchronous dynamic random access memory devices (SD RAM), double-data rate synchronous dynamic random access memory device (DDR SDRAM), mask read-only memory (ROM)

devices, erasable programmable ROM (EPROM), electrically erasable programmable ROM (EEPROM) devices, flash memory devices, and/or other volatile *and/or* non-volatile memory devices.” *Id.*, 4:41-51. The patent also discloses that “[t]he memory 182 may be embodied as any type of memory or data storage *locations* including, for example, dynamic random access memory devices (DRAM), synchronous dynamic random access memory devices (SDRAM), double-data rate synchronous dynamic random access memory device (DDR SDRAM), mask read-only memory (ROM) devices, erasable programmable ROM (EPROM), electrically erasable programmable ROM (EEPROM) devices, flash memory devices, *and/or* other volatile *and/or* non-volatile memory devices. The communication circuitry.” *Id.*, 5:16-33 (emphases added). A POSITA would have also understood from claim 1 that “a . . . memory” may include one or more memories based on the plain meaning of this term.

108. Chou’s memory, as described above, is “secure memory.” With respect to Figure 1, Chou discloses that “HSM 20 [] set[s] a secure domain, zone or boundary to protect a secure key.” Chou ¶18 (“The term ‘secure’ (such as secure domain, zone or boundary) is used herein to designate a zone controlled by HSM 20. Thus, components outside of the zone established by HSM 20 reside in a nonsecure zone relative to HSM 20.”); *see also id.* ¶20 (“[C]omponents 11-14 reside outside of a boundary established by HSM 20 and, therefore, are regarded as residing in a non-

secure domain or zone.”). Secure memory 22, secure key module 23, key cache 25/65, as well as memories associated with the secure processing within HSM 20 (e.g., security processor 21) are within Chou’s secure zone. *Id.* ¶¶21-22, Fig. 1; *see also id.* ¶¶23-29 (discussing operations within the “secure domain”).

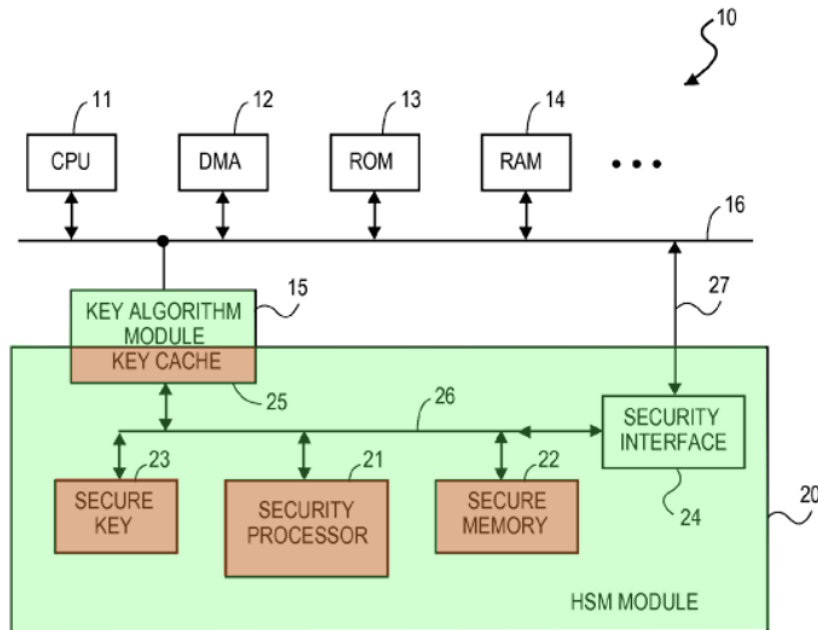


FIG. 1

Id., Fig. 1 (annotated).

109. Chou’s OTP 54, HSM BROM 52, HSM RAM 53, and key cache 65, as well as memories associated with the secure processing inside HSM (e.g., HSM processor 51, PKA 56, KEK 55 and RNG 58) are also “secure memory” because Chou similarly discloses these as being in the secure zone. *Id.* ¶¶30-41, Fig. 3. Chou discloses that HSM 20 of Figure 3 “includes a number of hardware and software components that encompass a secure domain boundary as shown in FIG. 3,” where

“solid-lined box 50 represents the boundary formed by the actual hardware components of HSM 20, while dashed-line box 80 represents the actual secure domain boundary that is not accessible by nonsecure components.” *Id.* ¶34, Fig. 3.

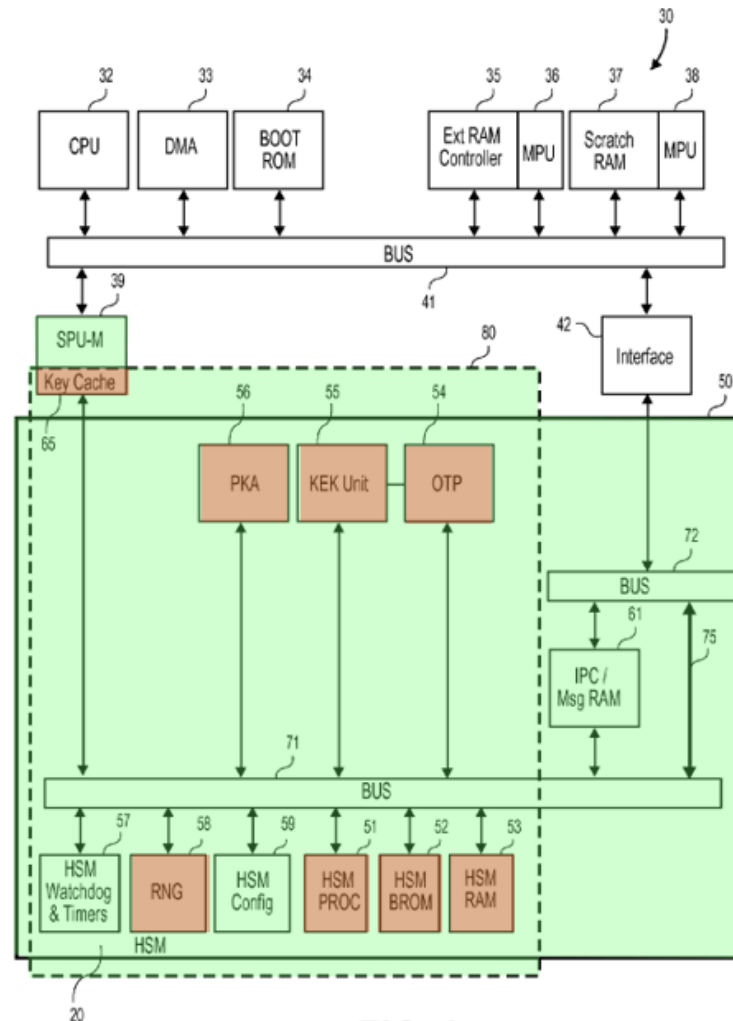


FIG. 3

Id., Fig. 3 (annotated).

110. Chou’s secure memory is “accessible only by the security engine,” as claimed. With respect to HSM 20 and Chou’s secure memory, Chou discloses that “a connection or bus 27 couples security interface 24 to bus 16, so that all accesses

to HSM 20 and data transfers into and out of HSM 20 are routed through security interface 24,” and “[n]o other accesses are permitted from the non-secure domain to HSM 20, except for the key cache operation described later in the description.” *Id.* ¶22. Chou also discloses that “since the unwrapping of secure keys is done solely within HSM and since the only route to internal hardware components of HSM 20 is through security interface 24, any unauthorized attempts from the non-secure domain to access an unwrapped key is prevented.” *Id.*, ¶25; *see also id.* ¶24 (“context data is protected from any unauthorized access from devices resident in the non-secure”), ¶26 (“Although the content of key cache 25 may be utilized, the content of cache 25 may not be copied or otherwise accessed by components of the non-secure domain.”), ¶27 (“any unauthorized access or copying from outside of HSM 20 is prevented”).

111. The secure memory of Figure 3 includes the same protections. *Id.* ¶34 (“[S]olid-lined box 50 represents the boundary formed by the actual hardware components of HSM 20, while dashed-line box 80 represents the actual secure domain boundary that is **not accessible by nonsecure components.**”) (emphasis added); *see also id.* ¶¶35-36 (“[S]imilar to security interface 24 of FIG. 1, IPC/Msg RAM module 61 controls the ingress and egress of instructions and/or data between components within HSM 20 and components outside of HSM 20.”); ¶63 (same), ¶69 (same).

112. Bugiel-Chou therefore satisfies “a secure memory accessible only by the security engine,” as claimed.

d. [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,”

113. The Bugiel-Chou combinations (**Grounds 1A-1C**) satisfy this feature.

114. For example, Chou discloses storing “one or more key value(s) that is used by security processor 21 to encrypt and decrypt data.” Chou ¶21, Fig. 1. Chou also discloses using a “device unique key that is unique to the device,” *id.* ¶23, and that the device unique key is “stored in secure key module 23,” *id.* ¶25. Chou also discloses the device unique key with respect to Figure 3. For example, Chou discloses that “OTP 54 is a non-volatile memory that is used to one-time program a key value or values for device 30,” and that “the device unique key may be programmed and stored in OTP 54.” *Id.* ¶38. Chou also describes the device unique key from OTP 54 as the “root key.” *Id.* ¶40.

115. A POSITA would have understood that Chou’s device unique key/root key is a “security key” that is in Chou’s “secure memory.” *See* Element [1a-2]. For example, Chou discloses using the device unique key to wrap and unwrap other security keys. *Id.* ¶¶22-25, 37, 69. Chou discloses using the device unique key/root key to wrap other keys via the AES key wrap specification. *Id.* ¶40; *see also id.* ¶¶7, 23, 25. Chou’s PKA module 56 and KEK Unit 55 of Figure 3 are communicatively

coupled via bus 71. *See id.* ¶35 (“Other units 54-69 are also coupled to bus 71”). A “direct link exists between KEK Unit 55 and OTP so that the key stored in OTP 54 may be loaded into KEK unit 55 without transitioning onto bus 71.” *Id.* ¶39. Using these connections, Chou’s KEK unit 55 loads the session key generated by PKA module 56 and the device unique key/root key from OTP 54 to encrypt the session key using the device unique key/root key. *See id.* ¶¶39-40, Fig. 3.

116. A POSITA would have understood or found it obvious that Chou’s device unique key/root is “encoded in the secure memory during a manufacturing process of the system-on-a-chip,” as claimed. For example, Chou’s OTP 54 is “One Time Programmed memory” that is “used to one-time program a key value or values for device 30,” including storing the device unique key. *Id.* ¶¶34, 38. One Time Programmed memory, such as OTP 54, is non-volatile memory that once programmed, cannot be changed. EX-1009, 240-241, 246-247. This type of memory is also referred to as fuse programmable ROM (or PROM). *Id.* A POSITA would have understood that PROM memory can be programmed using a program that blows small fusible links that are made of, for example, nichrome or polycrystalline silicon, and the process is irreversible. *Id.*

117. As discussed in Section VIII.A.2, a POSITA would have understood that OTP encoding may be accomplished by a user or by a manufacturer as part of a manufacturing process. For example, it would have been obvious to program

OTP 54 as a step in a manufacturing process because Chou contemplates programming aspects of its OTP 54 during manufacturing. With respect to connection 75 of OTP 54, Chou discloses disabling this feature by “*programming* a non-volatile bit in OTP 54 *during manufacturing*.” Chou ¶64 (emphases added). Because Chou contemplates programming its OTP 54 “during manufacturing,” and because programming bits of OTP is a one-time, irreversible process to set its permanent memory, it would have been obvious to program all aspects of OTP 54. This includes, for example, encoding OTP 54 with Chou’s device unique key/root.

118. Programming one-time programming memory like Chou’s OTP 54 during manufacturing was well-known in the art. For example, Paaske discloses a trusted platform module (TPM) that “provide[s] a set of cryptographic capabilities that enable certain computer functions to be securely executed within the TPM environment (e.g., hardware).” EX-1007 ¶7. Paaske discloses “one-time programmable (‘OTP’) memory 332 [that] may be used to store security information such as key material for security routines,” and that, “when the TPM 302 *is manufactured*[,] security routines may be used to generate secret keys that are unique to each TPM . . . [and] these keys may then be *stored in the OTP memory 332*.” *Id.* ¶56 (emphases added). Buer discloses the same with respect to a key encryption key (“KEK”). EX-1008, 2:25-40. Buer discloses that “KEK 130 is stored in a non-volatile data memory device 134,” including one-time-programmable

memory, and “[i]n a symmetric key system[,] the initial KEK 130 may be *programmed* into the non-volatile memory 134 *when the system is manufactured*.” *Id.*, 4:45-53 (emphases added); *see also id.* (“Typically, the non-volatile memory device 134 (e.g., an EEPROM)”), 17:5-22 (“the non-volatile memory may comprise a one time-programmable circuit”). A POSITA therefore would have known from the prior art to program one-time programmable memory, such as Chou’s OTP 54, when the memory is manufactured, just as Chou teaches and suggests.

119. Programming Chou’s OTP 54 during manufacturing is consistent with the ’838 patent, which discloses either “provision[ing]” the security key 150 during manufacturing of the SOC 112” or doing so “during operation.” EX-1001, 4:24-29 (programming security key 150 “based on blown fuses within the security engine 110 itself”). The patent does not disclose a particular manufacturing process separate from “provision[ing]” and blown fuses. A POSITA therefore would have understood that ’838 patent contemplates programming its security key by blowing fuses during manufacturing, and that Chou discloses or at least suggests doing the same by programming its OTP 54 “during manufacturing.” *Id.* ¶64. Chou’s programming therefore discloses “provision[ing]” a security key during manufacturing, consistent with the ’838 patent.

120. To the extent Chou alone does not satisfy encoding the security key by a manufacturing process during manufacture of the system-on-a-chip, doing so was

well-known in the art and would have been obvious in view of Bugiel-Chou and Tkacik (**Ground 1B**).

121. For example, as discussed in Section VIII.A.2, Tkacik discloses a long-lived key register 60, part of the “secure memory” of secure processing section 30, that includes a plaintext master key 160. Tkacik ¶¶44-46, Fig. 2. Tkacik discloses an embodiment in which plaintext master key 160 is stored in a non-volatile, read-only register, and that, “[i]n this embodiment, 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.* ¶46 (emphasis added). For example, Tkacik discloses that “laser-scribed fuses may be burned during the manufacture of integrated circuit 34 (FIG. 1) to permanently program plaintext master key 160 into long-lived key register.” *Id.* Tkacik also discloses that, alternatively, “electronic fuses may be electronically burned to program plaintext master key 160 and then blocked against further programming.” *Id.* This discloses a security key that is “encoded in [a] secure memory during a manufacturing process of the system-on-a-chip,” as claimed.

122. Tkacik’s non-volatile, read-only register storing long-lived key register 60 may include one-time programmable memory, just like Chou’s 54. In both embodiments of Tkacik’s “program[ming]” long-lived key register 60, Tkacik discloses burning fuses as part of a manufacturing process. *See id.* ¶46. This includes

burning “laser-scribed fuses” and “electronic fuses may be electronically burned” to program long-lived key register 60. *Id.* As is known in the art, one-time programmable memory utilizes fuses to program the memory, which is permanent and irreversible. *See* EX-1009, 240-241 (“The original true ROM types are mask programmable ROM and ***fuse*** programmable ROM (or PROM),” where “fuse programmed ROM devices are programmed by ***blowing small fuses*** and hence are sometimes called One Time Programmable ROM or OTPROM”) (emphases added); *see also id.*, 246-247 (blowing “small fusible links which are made of nichrome or polycrystalline silicon”).

123. A POSITA would have combined Tkacik with Bugiel-Chou by programming Chou’s OTP 54 using Tkacik’s laser or electronic burnt fuse process during manufacturing of Chou’s SoC. For example, a POSITA would have understood that that Chou’s OTP 54 is one-time programmable memory, just like Tkacik’s long-lived key register 60, and that Chou’s OTP 54 requires programming the unique key/root by blowing fuses. A POSITA would have found it obvious to program OTP 54 by burning “laser-scribed fuses” or electronically burning “electronic fuses” during manufacturing of the OTP 54 on Chou’s SoC, as Tkacik discloses. Doing so would have programmed the Bugiel-Chou device from the factory with a security key that is unique to the device and cannot be changed. Doing so by a process during manufacturing of the OTP would have ensured that the device

is programmed for use when manufactured and eliminated the later step of programming the OTP by a user, thus providing a ready-to-use device. *See* Section VIII.A.2.

124. Combining Bugiel-Chou with Tkacik would have been nothing more than combining known elements in the prior art according to understood principles. For example, it was known to program one-time programmable memory in a device with keys during manufacturing, as taught by Tkacik and at least suggested by Chou. Other prior art confirms this, including Paaske and Buer. *See* EX-1007 ¶¶7, 56; EX-1008, 2:25-40, 4:45-53, 17:5-22. Applying this known teaching in the art to Bugiel-Chou's hardware would have yielded predictable results and a POSITA would have had an expectation of success because it is nothing more than applying known methods of programming one-time programmable memory to a step in the manufacturing process of Chou's SoC, as opposed to programming the device after it is manufactured. *See* Section VIII.A.2.

125. The Bugiel-Chou or Bugiel-Chou-Tkacik combinations therefore satisfy encoding the secure memory in the secure memory during a manufacturing process of the system-on-a-chip, as claimed.

- e. **[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;”**

126. The Bugiel-Chou combinations (**Grounds 1A-1C**) satisfy this feature.

127. For example, Bugiel discloses methods for the “protection of login credentials when accessing web services” using, for example, smartphone mobile devices. Bugiel, 219-220 (“The mobile web browser on the phone enables users to access standard Internet web sites, including those that require an authentication of the user (typically username/password).”). This includes protecting communications between the mobile devices and a “remote web server.” *Id.* (discussing “wallet-like authentication agents” to “automatically manage mutual authentication between the user’s computing device and a remote web server” and “existing web-based authentication on most servers”).

128. Bugiel discloses that its solution “split[s] a single SSL/TLS connection *between [a] user device and [a] server* into two logically separated channels, where one is protected by TrEE and is used to transmit passwords, and another one is intended for conventional data.” *Id.*, 220; *see also id.*, 221 (disclosing communications between a user device and a “web server”), 222 (utilizing SSL/TLS protocol “[f]or secure communication between the device and the remote web server”), 225 (“WebBrowser is typically used by the user as a tool to login to remote web servers.”).

129. Bugiel therefore discloses “a secure communication session with a remote server over a network,” as claimed.

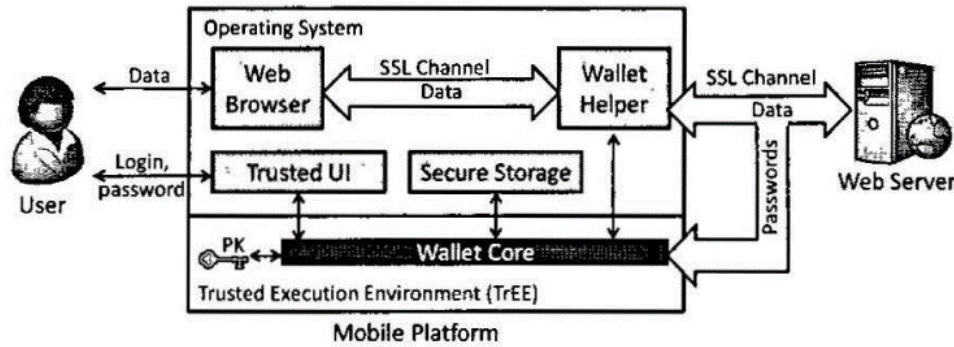


Fig. 1. TruWalletM Architecture

Id., Fig. 1 (illustrating the mobile platform, SSL channel over a network, and a remote web server).

130. Bugiel’s description matches the ’838 patent’s description of using SSL to enable communication with a remote server. *See* EX-1001, 1:10-24 (“One commonly used cryptographic communication protocol is the Secure Sockets Layer (SSL) protocol. The SSL protocol, and the related Transport Layer Security (TLS) protocol, is used for many different types of secure communication sessions including, for example, secure web-browsing, ecommerce, secure upgrades, and other secure communication sessions between two computing devices on an untrusted network such as the Internet . . .”).

131. Bugiel also discloses generating a random nonce for initiating a request for a secure communication session with the remote server using the nonce, as claimed. For example, Bugiel discloses with reference to Figure 2 that the mobile device establishes an “SSL/TLS Connection” with a web server using “the SSL handshake protocol.” Bugiel, 225-226, Fig. 2. Bugiel also discloses “[a]ll messages

which are exchanged between WalletHelper and WebServer are standard messages of SSL/TLS handshake and can be found in the corresponding specification [16].” *Id.*, 225 (citing TLS-1.2).

132. As the first step of the SSL/TLS handshake, the mobile device (e.g., “client”) exchanges ClientHello and ServerHello messages with the web server. *Id.*, 225-226. A POSITA would have understood that a ClientHello message in a TLS handshake includes the client sending a “hello” message to a server that includes, among other things, a string of random bytes known as the client random. *See, e.g.*, TLS-1.2, 17 (defining “client random” as “[a] 32-byte value provided by the client”), 33-34 (“The TLS Handshake Protocol involves the . . . [e]xchange hello messages to agree on algorithms, exchange *random values*, and check for session resumption.”) (emphasis added), 34 (“The ClientHello and ServerHello establish the following attributes: Protocol Version, Session ID, Cipher Suite, and Compression Method. Additionally, *two random values* are generated and exchanged: ClientHello.random and ServerHello.random.”) (emphasis added); *see* EX-1006, 41-43 (“ClientHello Components” include a “RandomNumber” field that includes a 32-byte random number used to seed the cryptographic calculations), 42 (using a “pseudorandom number generation to create random numbers”).

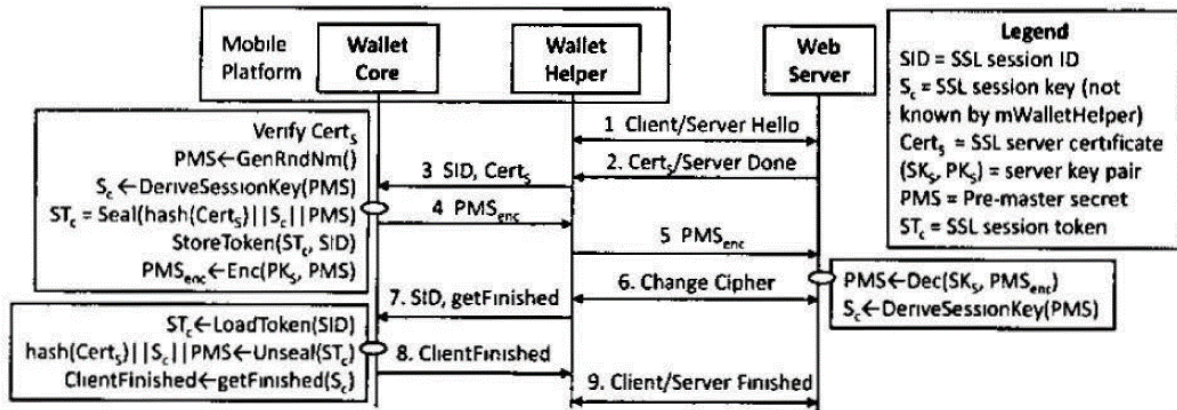


Fig. 2. SSL/TLS handshake

Id., Fig. 2.

133. Indeed, before the filing of the '838 patent, it was common knowledge that ClientHello.random and ServerHello.random were random nonces exchanged between client and remote server to avoid “replay” attack in SSL/TLS protocols. *See* TLS-1.2, 91-92). TLS-1.2 incorporates TLS-1.0 by reference (*see* TLS-1.2, 101), which incorporates by reference the original SSL patent in U.S. Patent No. 5,657,390 by Elgamal. *See* EX-1018, 74 (citing Elgamal). Elgamal teaches including “a random number used to ensure channel integrity” in the client-hello message, which the server responds with a server-hello message including a server random number. EX-1010, 7:13-40 (emphasis added); *see also id.*, Figs. 4-7. Elgamal further describes the random number as “nonce” that is “a randomly generated value used to defeat ‘playback’ attacks.” *Id.*, 33:24-31.

134. Many other patent publications before the '838 patent describe the random numbers exchanged between the server and client in SSL/TLS protocols as

random nonce. *See, e.g.*, EX-1020 (McGarvey), 7:16-8:3; EX-1021 (Binding), 10:41-57, 11:52-12:4, 14:52-15:6; EX-1022 (Yao) ¶¶114, 175-177, 186; EX-1023 (Bollay) ¶¶62, 90, 137; EX-1024 (Lee) ¶¶74, 81, 87-88, 118, 130. Bugiel's ClientHello, which includes a random number as contemplated by the TLS 1.2 specification, therefore discloses generating a "random nonce," as claimed.

135. Bugiel's ClientHello—a message sent by the client device and includes a random number generated by the client—is a nonce for initiating a request for a secure communication session with a remote server over a network, as claimed. This is consistent with the '838 patent, which discloses that "security engine 110 generates a random nonce value" and "may use any suitable *random number-generating algorithm* to generate the random nonce." EX-1001, 10:14-27 (emphasis added); *see also id.*, 6:40-50 ("security engine 110 may use any suitable random number-generating algorithm"). Bugiel's TLS handshake to initiate communication with a remote web server is also consistent with Chou, which explains that its invention is used for "secure data transfer . . . to perform secure financial transactions over a network, such as the Internet." Chou ¶6.

136. In the Chou-Bugiel combination, Bugiel's ClientHello message would be utilize a random number generated by, for example, Chou's random number generator 58 of HSM 20 (e.g., the "security engine," *see* [1a-1]). Chou, ¶¶34, 42, claim 11, Fig. 3. Consistent with the '838 patent and Bugiel's TLS handshake,

random number generator 58 of HSM 20 can generate a random number using, for example, a random bit generator to generate true random numbers.” *Id.* ¶42. While Chou also discloses that its random number “is typically used as a seed for a cryptographic algorithm and the value from RNG 58 is typically post processed and not used directly,” a POSITA would have found it obvious to utilize random number generator 58 of HSM 20 to generate a random number used to form the ClientHello message during the TLS handshake in Bugiel. *Id.* Doing so would have been nothing more than combining known elements in the prior art with expected results. Chou’s disclosure of using its random number as a “seed for a cryptographic algorithm” is also consistent with TLS handshake methods, as it was known in the art 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.

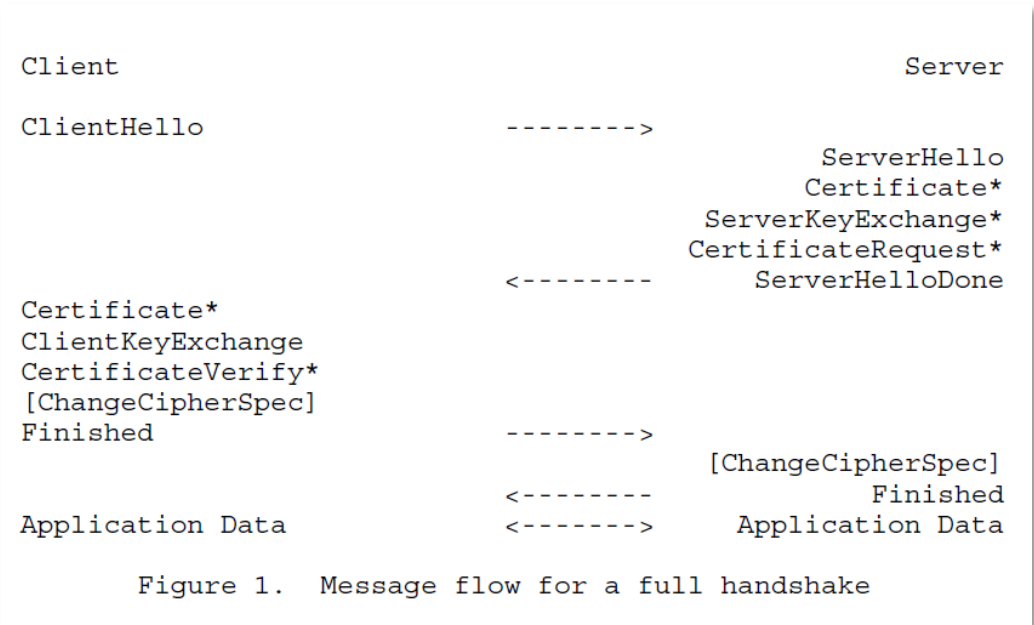
137. A POSITA thus would have found it obvious to use Chou’s RNG 58 to “generate a random nonce,” as claimed, for use in the ClientHello message, which Bugiel requires to request a secure SSL/TLS communication session with a remote network server.

f. [1c] “perform a cryptographic key exchange with the remote server;”

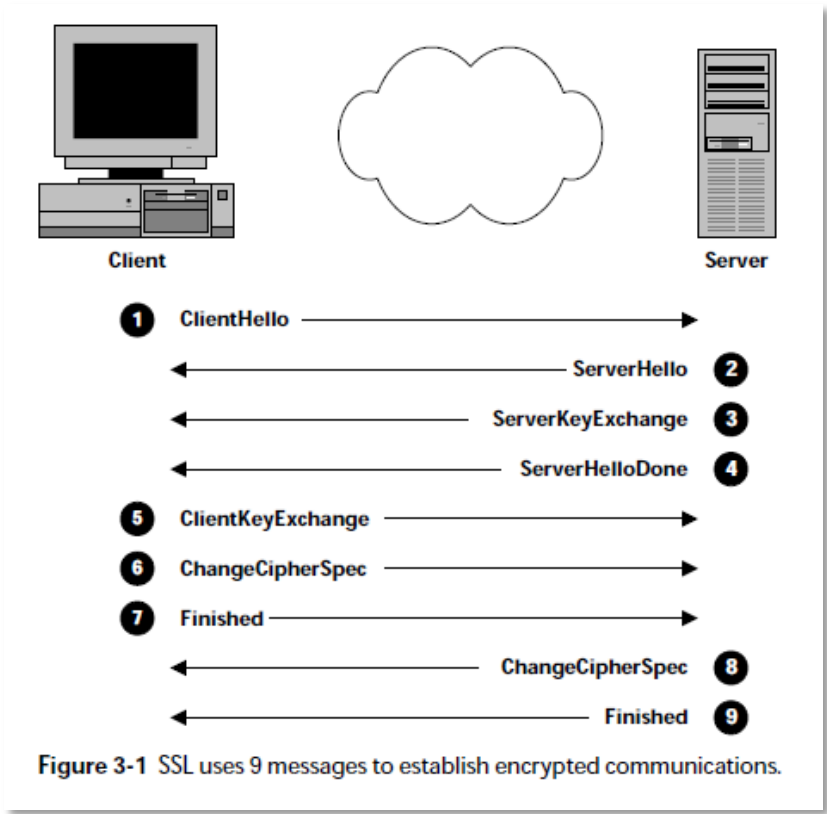
138. The Bugiel-Chou combinations (**Grounds 1A-1C**) satisfy this feature. For example, Bugiel discloses that its message exchange between its device and Web

Server “are standard messages of SSL/TLS handshake and can be found in the corresponding [TLS-1.2] specification.” Bugiel, 225 (referencing TLS-1.2 specification); *see also id.*, 226 (“all protocol messages between WalletHelper and WebServer follow the flow of the SSL resume protocol”).

139. As is known in the art and disclosed by Bugiel, the SSL/TLS handshake includes a client device and server exchanging Hello messages, followed by the server and the client exchanging ServerKeyExchange/ClientKeyExchange messages, and optionally certificates, where the key exchange/certificate messages include key information based on RSA or Diffie-Hellman key exchange. TLS-1.2, 26-64 (TLS Handshake protocol, including exchanging hello messages, certificates, server/client key exchange message, based on RSA or Diffie-Hellman key exchanges, etc.), 47-53 (discussing “Server Certificate” and “Server Key Exchange Message” based on RSA or Diffie-Hellman key exchange, where the “ServerKeyExchange message is sent by the server only when the server Certificate message (if sent) does not contain enough data to allow the client to exchange a premaster secret.”), 64-65 (computing master secret based on RSA or Diffie-Hellman key exchange), 70-75 (Handshake protocol), 92-92 (RSA and Diffie-Hellman Key Exchange); *see* EX-1006, 38-51 (describing ServerKeyExchange message from server), 52-59 (describing certificate message or certificate message and ServerKeyExchange).



TLS-1.2, Fig. 1.

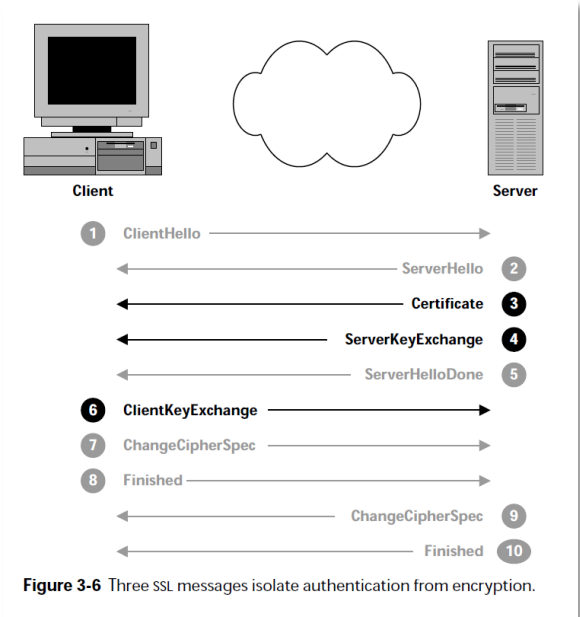
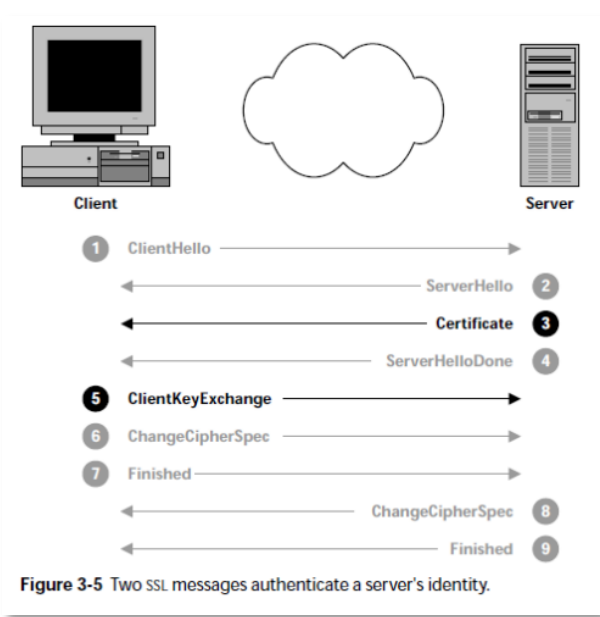


EX-1006, Fig. 3-1 (describing messages exchanged between client and server during

an SSL/TLS handshake process (including e.g., ServerKeyExchange, ClientKeyExchange, etc.)).

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 (same).



Id., Fig. 3-5, (describing server sending Certificate to client, followed by

ClientKeyExchange), Fig. 3-6 (describing server sending Certificate and a ServerKeyExchange message to client, followed by ClientKeyExchange).

140. The server in an SSL/TLS handshake process sends a public key in either a ServerKeyExchange message or in the server's Certificate based on RSA or Diffie-Hellman key exchange methods. TLS-1.2, 47-53 (the ServerKeyExchange message is sent separately from the server Certificate message on if the Certificate "does not contain enough data [(e.g., the public key)] to allow the client to exchange a premaster secret."), 26-64 (the TLS handshaking protocols); *see* EX-1006, 45 (ServerKeyExchange message "contains the public key information itself"), 55-56 ("the server sends a *Certificate* message instead of the ServerKeyExchange" and "clients . . . know[] the public key of trust certificate authorities in advance, through some trusted means"), 55-56 (server sending only its Certificate, and "[i]n this case, of course, the server is authenticating itself and, thus has sent a Certificate message instead of a ServerKeyExchange. The client, therefore, encrypts its ClientKeyExchange information using *the public key contained in the server's certificate.*") (emphasis added), 59 (separating server authentication from encryption by the server sending a separate Certificate and ServerKeyExchange message, where the ServerKeyExchange message "contains the public key the client should use to encrypt session key information."). The server and client therefore

exchange key information in the TLS/SSL handshake messages to generate a symmetric session key.

141. Bugiel discloses a cryptographic key exchange in which all messages exchanged between the client and the sever are standard SSL/TLS handshake messages (including, e.g., its server sends a certificate message) according to TLS-1.2 protocol. Bugiel, 225-226. For example, Bugiel discloses the SSL/TSL handshake with respect to Figure 2. *Id.*, 225-226, Fig. 2. Bugiel discloses that its WalletHelper of its mobile device “acts as an SSL/TLS proxy” when communicating with the web server. *Id.* After the client and server exchange ClientHello and ServerHello messages, “the server’s certificate $Cert_s$ is sent and verified.” *Id.* 225. Bugiel discloses that the certificate $Cert_s$ is an “SSL server certificate,” and following receipt, the mobile device generates a pre-master secret (“PMS”) and derives a session key in S_c . *Id.*, 225-226, Fig. 2.

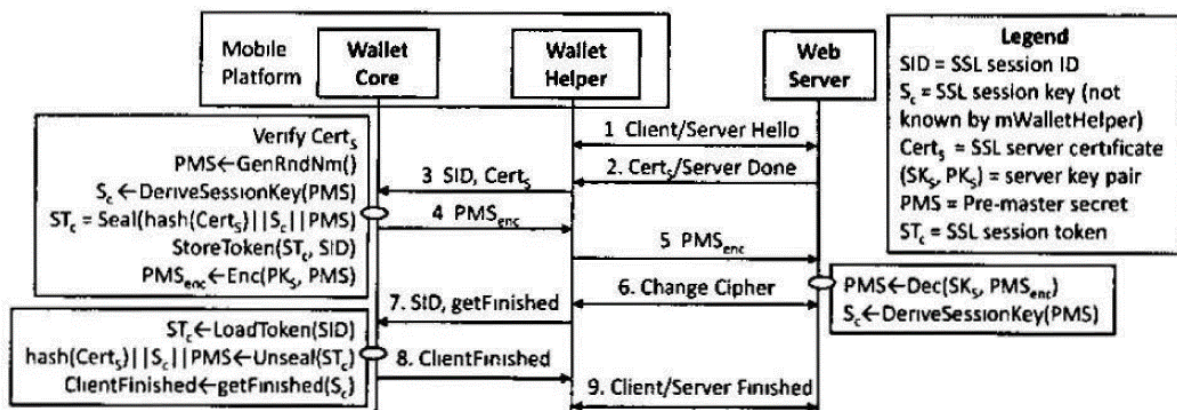


Fig. 2. SSL/TLS handshake

Id., Fig. 2.

142. A POSITA would have understood Bugiel discloses performing the cryptographic key exchange by performing the SSL/TLS handshake process as described in TLS-1.2 using RSA encryption (including e.g., exchanging the server certificate $Cert_S$, and a follow-on key exchange messages according to the TLS handshake). Bugiel also discloses that the PMS and session key S_C are derived based on the key received from these messages, including the server certificate $Cert_S$. *See id.*, 225-226, Fig. 2. According to the known TLS-1.2 specification, Bugiel's server's certificate $Cert_S$ would contain a public key. TLS-1.2, 26-65 (describing TLS handshake protocol, including e.g. exchanging certificates and/or key exchange message to derive master secret and symmetric session keys based on RSA or Diffie-Hellman, etc.); EX-1006, 45, 55-56, 55-56 ("The client, therefore, encrypts its ClientKeyExchange information using *the public key contained in the server's certificate.*") (emphasis added), 59.

143. Bugiel discloses performing a cryptographic key exchange with a remote server, as claimed, because the certificate $Cert_S$ received from the server includes a public key used to derive a session key as part of the TLS handshake process. Indeed, the '838 patent similarly discloses the server sending a message to perform this "exchange." For example, with respect to Figure 3, the '838 patent discloses sending a server certificate in step 316 and sending a public key in step 320. EX-1001, 7:10-44, Fig. 3. The patent explains that, "[i]n embodiments wherein

a RSA key exchange is used, the server secure communication application 306 performs a server RSA *key exchange* in block 320,” where “the 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.” *Id.* (emphases added). Block 320 of Figure 3 is moreover entitled “Server Key Exchange.” *Id.*, Fig. 3.

144. To the extent the claimed “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 as part of the standard TLS handshake process, which involves the client exchanging messages with a remote server in response to the server certificate/server key exchange message. For example, Bugiel discloses sending a PMS_{enc} message to the server. Bugiel, 226, Fig. 2. Bugiel discloses that the PMS_{enc} message encloses both the public key (“PKs”) 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, which provides for a ClientKeyExchange message based on RSA or Diffie-Hellman key exchange. TLS-1.2, 55-65 (TLS Handshake protocol, including a client key exchange message based on RSA or Diffie-Hellman key exchange and optional client certificate), Fig. 1. This “tells the server the client’s key information” for “the symmetric encryption algorithm both parties will use for the session,” where the message is encrypted “using the public key of the server.”

See EX-1006, 45-46, 56. Therefore, the Bugiel of the Bugiel-Chou combinations (**Grounds 1A-1C**), and optionally TLS-1.2 of **Ground 1C**, satisfy this feature.

145. Similarly, to the extent it is argued the cryptographic key exchange requires sending a public key in a separate ServerKeyExchange message, as contemplated as an option in the TLS-1.2 specification, it would have been obvious to modify Bugiel to perform this step (**Ground 1C**). As discussed above, Bugiel incorporates the TLS-1.2 specification into its disclosure and contemplates using the specification to perform “standard messages of [an] SSL/TLS handshake.” Bugiel, 225 (citing TLS-1.2 in [16]). It therefore would have been obvious to receive a public key from Bugiel’s server in a separate ServerKeyExchange message instead of including it in Bugiel’s certificate *Certs*, consistent with separating server authentication and encryption messages according to the TLS-1.2 specification. TLS-1.2, 33-37, Fig. 1. This would have been nothing more than substituting one known feature for another, with predictable results, where a programmer would have known to separate the key used to authenticate a server in one message, from a key used for encryption techniques in another message.

146. Indeed, separately sending a server Certificate message and ServerKeyExchange message was well-known and desirable. TLS-1.2, 33-37, Fig. 1; see EX-1006, 56-58. For example, “some public key algorithms (such as the Digital Signature Algorithm) can only be used for signing,” and thus “cannot be used

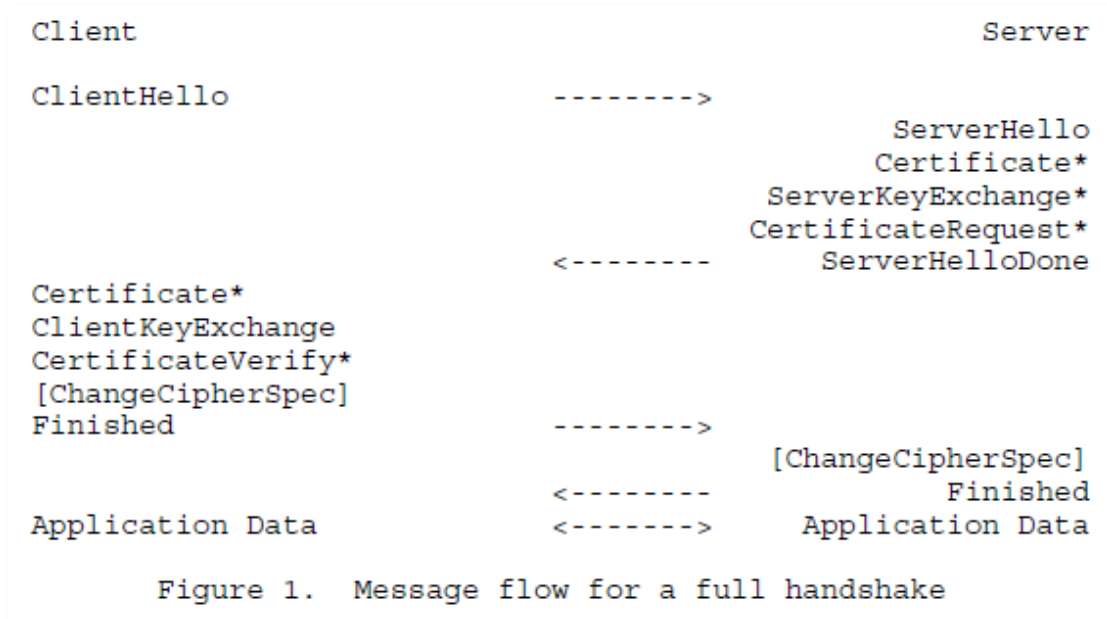
for encryption.” EX-1006, 56-58 (in this case “it will be impossible for the client to encrypt its ClientKeyExchange information using the server’s public key.”).

147. Modifying Bugiel to perform a cryptographic key exchange based on TLS-1.2—to send a public key in a separate ServerKeyExchange message—would have been nothing more than the simple substitution of one known element (e.g., sending a public key via a certificate message) for another (e.g., sending the public key via a ClientKeyExchange message to perform the key exchange as described in the TLS-1.2 handshake process) to obtain predictable results.

148. The same is true of implementing Diffie-Helman encryption, as taught by TLS-1.2, in Bugiel. 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 VIII.A.8 (claim 6) (describing Diffie-Helman key generation in Bugiel-TLS-1.2).

149. Performing TLS-1.2 handshake process using these methods was well known in the art and a POSITA would have readily substituted one for the other with predictable results and an expectation of success. This would have only required (1) the server in Bugiel-Chou to send a send message, after its certificate, with the public key for use by the client device in Bugiel-Chou to generate a session key; or

(2) sending parameters to derive the pre-master secret using Diffie-Helman instead of RSA encryption.



TLS-1.2, Fig. 1.

150. In the Bugiel-Chou combinations (**Grounds 1A, 1B, and 1C**) a POSITA would have implemented Bugiel's WalletHelper and WalletCore using Chou's hardware. *See* Sections VIII.A.1, VII.A.2, VII.A.3. For example, a POSITA would have implemented Bugiel's WalletHelper using, for example, Chou's security interface 24 or IPC/Msg RAM module 61. *See* Chou, ¶21 ("all accesses to HSM 20 and data transfers into and out of HSM 20 are routed through security interface"), ¶36 ("similar to security interface 24 of FIG. 1, IPC/Msg RAM module 61 controls the ingress and egress of instructions and/or data between components within HSM 20 and components outside of HSM 20"), ¶¶21-23, 25, 34-37, 63, Figs. 1 and 3; *see*

also Bugiel, 223-227 (“WalletHelper acts as an SSL/TLS proxy”). A POSITA would have implemented Bugiel’s WalletCore using, for example, Chou’s HSM 20 and SPU-M 39/key algorithm module 15. For example, Chou describes the PKA module 56 inside HSM 20 is operable for cryptographic key exchange algorithms, including Diffie-Hellman and RSA key generation. *See* Chou ¶¶43-53.

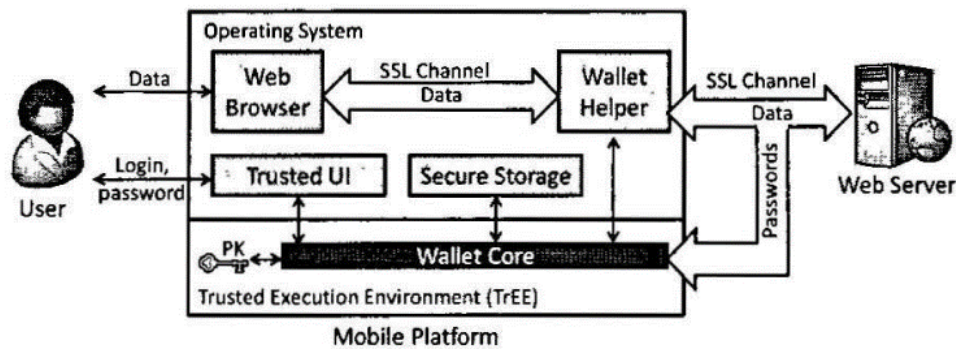


Fig. 1. TruWalletM Architecture

Bugiel, Fig. 1.

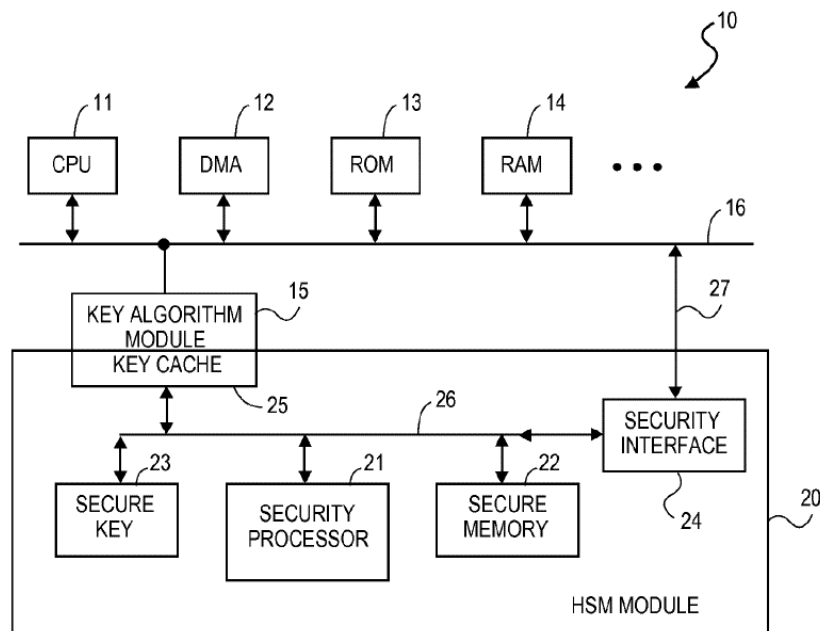


FIG. 1

Chou, Fig. 1.

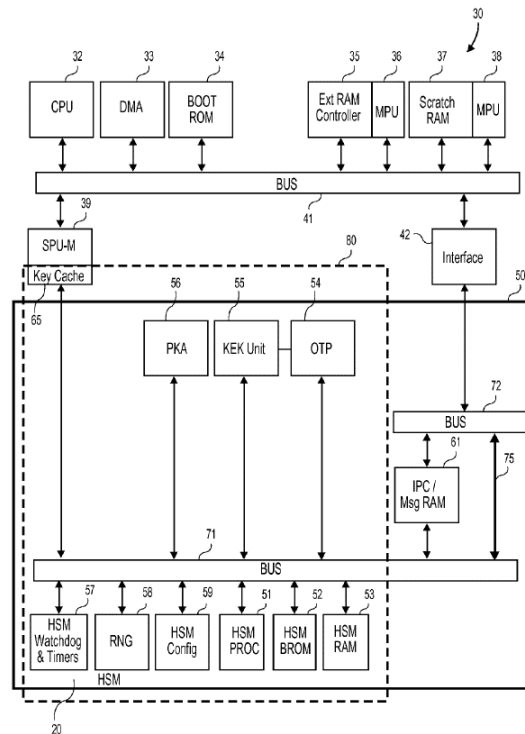


FIG. 3

Id., Fig. 3.

151. Bugiel's WalletHelper acting as a proxy for its WalletCore, implemented in Chou's security interface 24 or equivalent IPC/Msg RAM module 61, corresponds with the description of the '838 patent. The '838 patent discloses, for example, a client secure communication application 304 that is "executed on the client device 102." EX-1001, 6:20-33 ("To establish the secure communication session, a security engine driver 302 of the security engine 110 communicates with a firmware of the security engine 110 and a client secure communication application 304 executed on the client device 102. The client secure communication application 304 also communicates with a corresponding server secure communication

application 306 executed on the server 104.”), Fig. 3; *see also id.*, 6:34-7:44 (secure communication application 304 performing TLS handshake with server, including a key exchange).

- g. [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;”

152. The Bugiel-Chou combinations (**Grounds 1A-1C**) satisfy this feature. For example, Bugiel discloses generating 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, the session key S_c is generated in Bugiel’s WalletCore following verification of the server certificate $Cert_s$. *See id.*, Fig. 2.

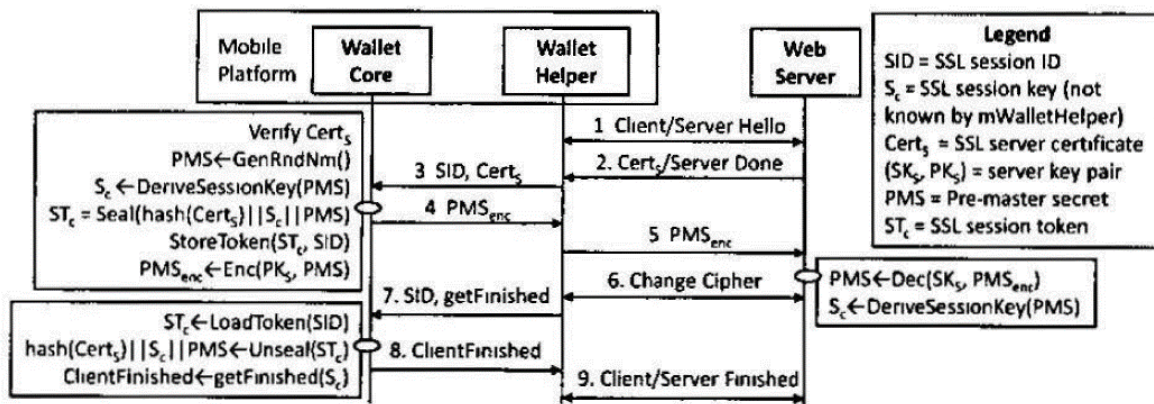


Fig. 2. SSL/TLS handshake

Id.

153. 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]. It also would have been obvious to implement Diffie-Helman key agreement protocol instead of RSA to allow a client and server to perform computations to agree on a shared symmetric key. *See* [1c]; *see infra* Section VIII.A.8 (claim 6) (describing generating the symmetric key as a function of the Diffie-Helman protocol), Section VIII.A.9 (claim 7).

154. 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 session key S_C based on the server certificate. *See* [1c]. For example, Bugiel discloses verifying the server certificate *Certs*, generating a pre-master key (PMS), and then deriving the session key S_C from the PMS. *See* Bugiel, Fig. 2 (" $S_C \leftarrow \text{DeriveSessionKey(PMS)}$ "). 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, Fig. 2; TLS-1.2, 33-37, Figs. 1, 2; *see also id.*, 58-61 (describing the ClientKeyExchange and stating that, "[i]f RSA is being used for key agreement and authentication, the client generates a 48-byte premaster secret, encrypts it using the public key from the server's certificate, and sends the

result in an encrypted premaster secret message.”), 18, 24-26, 64-65, 79-81, 91, 94 (“master_secret = PRF (pre_master_secret, “master secret”, ClientHello.random + SeverHello.random) [0. .47]”), 26, 58, 94; EX-1006, 45-46, 56.

155. Bugiel’s “SSL/TLS session key S_C ” is consistent with the symmetric session key of the ’838 patent. For example, the ’838 patent discloses with respect to Figure 3 that, 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. The patent further 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 (“When RSA is used for server authentication and key exchange, a 48-byte pre_master_secret is generated by the client, encrypted under the server’s public key, and sent to the server. The server uses its private key to decrypt the pre_master_secret. Both parties then convert the pre_master_secret into the master_secret.”).

156. Bugiel’s session key S_C is a “symmetric” session key because, as indicated in Bugiel’s Figure 2, Bugiel’s remote server derives the same session key

S_C as the client device. See Bugiel, Fig. 2 (“ $S_C \leftarrow \text{DeriveSessionKey}(PMS)$ ”). This again is consistent with the TLS-1.2 specification and its encryption of a pre-master secret that is encrypted with the server’s public key and shared back with the server to derive the session key. TLS-1.2, 64-65 (“Both parties ... convert the pre_master_secret into the master_secret”); see EX-1006, 45-46 (session key encrypted by “the server’s public key” and the “ClientKeyExchange tells the server the client’s key information” for decryption using the public key).

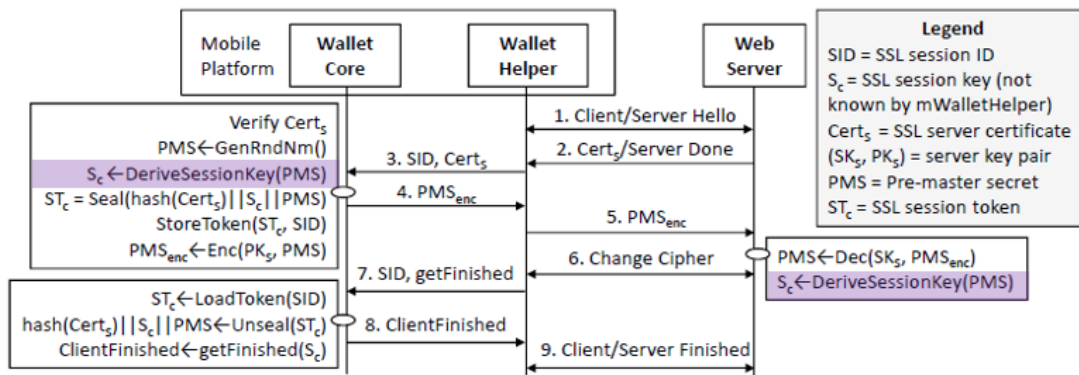


Fig. 2. SSL/TLS handshake

Bugiel, Fig. 2 (annotated).

157. Bugiel’s “SSL/TLS session key S_C ” is used to encrypt messages sent to the remote server and decrypt messages received from the remote server during a secure communication session, as claimed. Indeed, Bugiel expressly discloses this by stating that, after generating the symmetric session key, “[a]ll following messages should be encrypted using S_C .” *Id.*, 225. A POSITA would have understood that

“[a]ll following messages” includes “messages sent to the remote server” and “messages received from the remote server,” as claimed.

158. As discussed above and below in claims 6 and 7, it also would have been obvious to generate the symmetric session key based on a Diffie-Hellman exchange. *See* [1c]; *see infra* Section VIII.A.9 (claim 7). Specifically, it would have been obvious to implement a Diffie-Hellman key exchange in the Bugiel-Chou combinations based on TLS-1.2. *See* Bugiel, 225-226. The master secret (e.g., the symmetric session key) would be 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 (“With this [ClientKeyExchange] message, the premaster secret is set, either by direct transmission of the RSA-encrypted secret or by the transmission of Diffie-Hellman parameters”); *see also id.*, 61 (“This structure conveys the client’s Diffie-Hellman public value (Yc)”); *see* EX-1006, 85-88 (providing contents of ClientKeyExchange message for RSA and Diffie-Hellman); *see* [1c]; *see infra* Section VIII.A.9 (claim 7).

159. As disclosed above, the symmetric session key of the Bugiel-Chou combination would have been implemented by the HSM 20 of Chou. Chou also explains that SPU-M 39 “operates as a hardware accelerator for symmetric key algorithm,” Chou ¶¶54-55 (“SPU-M module 39 may implement many symmetric key algorithms such as AES, DES, RC4, SHA1-HMAC, SHA2-HMAC, MD5-

HMAC, as well as others.”), and a POSITA would have found it obvious to implement Bugiel’s and/or TLS-1.2’s methods using the hardware features in Chou. *See id.* ¶¶54-61.

160. To the extent it is argued that Bugiel’s SSL/TLS session key S_C alone does not disclose the claimed symmetric session key, TLS-1.2 alternatively discloses this feature by generating a “master key” or “key_block.” The 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 of the security engine 110 of the SOC 11.” EX-1001, 5:63-66 (emphasis added). TLS-1.2 thus 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]. *See* TLS-1.2, 64-65 (computing the Master Secret), 16-17, 24-27, 34, 58, 80, 91-94; EX-1006, 85-87 (describing that “the premaster secret is a preliminary step in derive the master secrete for the session.”), 88-89 (client computing the “master secret” based on the premaster secret); *see also* [1c].

161. TLS-1.2 also discloses generating encryption keys (e.g., a “key_block”) for data encryption, based on the master secret for each session. TLS-1.2, 25-26 (Key Calculation). 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 (defining the master_secret, client_write_key partition, and server_write_key partition). 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; *see also id.*, 4 (“The connection is private. Symmetric cryptography is used for data encryption (e.g., AES [AES], RC4 [SCH], etc.). The keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by another protocol (such as the TLS Handshake Protocol).”), 6, 16-17, 22-25, 27, 40, 65, 75-79, 82-84, 97, 100.

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

Id., 26; *see also id.*, 16, 18, 24-26, 80, 84, 91, 94-95.

162. A POSITA would have found it obvious to implement Bugiel to generate the claimed symmetric session key via TLS-1.2’s “master secret” or “key_block” (**Ground 1C**). The combination would have (1) generated an encryption key, either via the “master secret” or “key_block,” based on cryptographic key exchange with a remote server according to the TLS handshake protocol; and (2) used the same session key (e.g., the “master secret” or “key_block”) to encrypt messages sent to the remote server and decrypt messages

received from the remote server during the secure communication session. The combination 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.

163. Indeed, 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 a POSITA would have understood Bugiel is compatible with the specification, including its master secret or key_block. As discussed, a POSITA would have found it obvious to perform these tasks using Chou's HSM 20 (part of the claimed "security engine") to generate keys based on SSL and Chou's SPU-M 39 (also part of the claimed "security engine") to encrypt/decrypt messages.

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

164. The Bugiel-Chou combinations (**Grounds 1A-1C**) satisfy this feature. For example, Bugiel discloses that, after generating its SSL/TLS session key S_C , the session key S_C is "*sealed*." Bugiel, 225 ("In the following steps the server's certificate $Certs$ is sent and verified, the SSL/TLS session key S_C is generated (which is never disclosed to WalletHelper) and sealed (together with pre-master secret PMS and the hash value $\text{hash}(Certs)$). The resulting ST_C is stored in a secure storage and

can be loaded and unsealed later using the session identifier SID.”) (emphasis added).

165. 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.

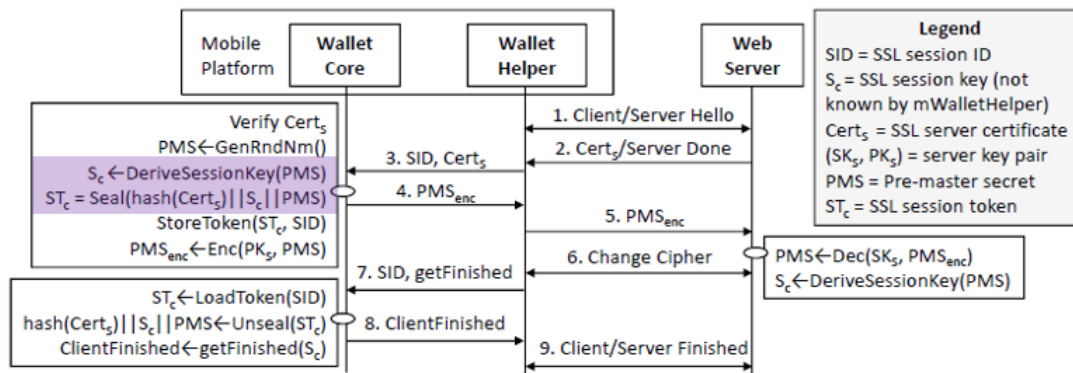


Fig. 2. SSL/TLS handshake

Id. (annotated).

166. Bugiel explains that “sealing” means “protecting an object so that only a certain set of entities can access or use it.” *Id.*, 225 n.8. Bugiel further discloses, when discussing implementations of its TruWalletM system, that “[w]hen sealed, data is **encrypted under a key** that is never available outside the TrEE [(Trusted Execution Environment)].” *Id.*, 230-231 (emphasis added).

167. A POSITA would have understood that, by “sealing” and “encrypt[ing] under a key” its session key S_C , Bugiel discloses encrypting a symmetric session key

based on a security key, as claimed. The resulting session token ST_C discloses the encrypted symmetric session key. Indeed, Bugiel discloses encrypting the session key with “a key that is never available outside the TrEE,”

168. Bugiel’s key corresponds to the claimed “security key.” A POSITA would have implemented Bugiel’s key for encrypting its session key S_C via Chou’s device unique key/root key that is encoded on its OTP 54, as discussed in [1a-3]. Both Chou and Bugiel contemplate secure regions that are 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 device unique key/root key that is encoded on its OTP 54. 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.

169. In the combination, 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. 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. *See also* Chou, ¶37 (“Components 54-59 are operable to provide processing operations that pertain to secure key functions, including the wrapping and unwrapping of keys, context data, etc.”), ¶18 (“The term ‘key’ is used

herein to describe any value, data, context, etc., which is used to change the format of the information (such as data) to provide security from unauthorized access.”), ¶¶24-27, 38-41, 63, 69, claims 4, 8, 16, 19.

170. Accordingly, the combination Bugiel-Chou combinations utilize Chou’s HSM 20 (e.g., part of the claimed “security engine”) to encrypt a session key based on Chou’s device unique key/root key (e.g., the claimed “security key.”).

i. [1f] “store the encrypted session key in the secure memory,”

171. The Bugiel-Chou combinations (**Grounds 1A-1C**) satisfy this feature.

172. For example, Bugiel discloses the need to store a session-token ST_C in secure storage. For example, Bugiel explains that, after “the SSL/TLS session key S_C is generated . . . and sealed (together with pre-master secret PMS and the hash value $\text{hash}(Certs)$),” “[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.

173. 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 . In addition, Bugiel also teaches or suggests this claim limitation 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 (“(i) run-time isolation. All operations on user

credentials (except user input) are performed within the TrEE. As the TrEE is isolated from the rest of the system in terms of processing and memory, that guarantees protection of the user's credentials at run-time from potentially malicious OS and other OS-side components"), Figs. 4-6.

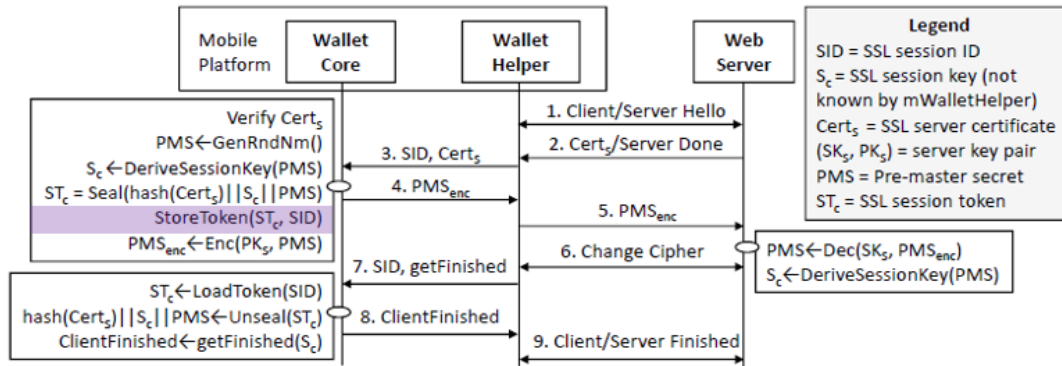
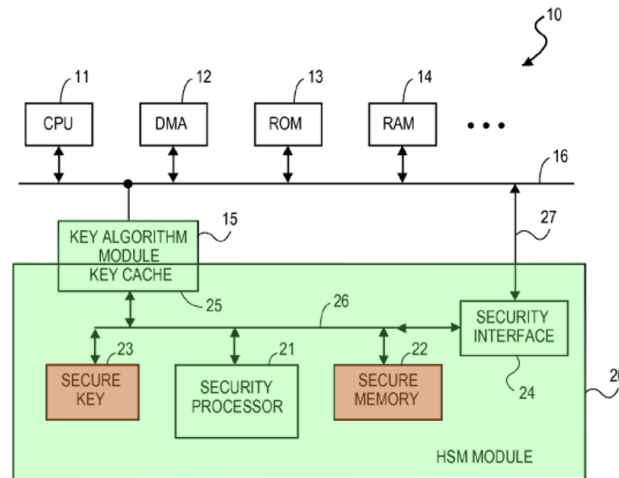
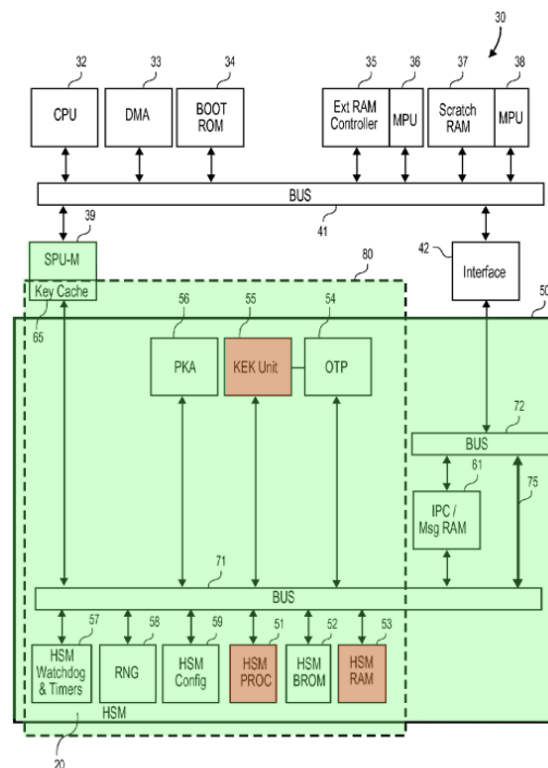


Fig. 2. SSL/TLS handshake

Id., Fig. 2 (annotated).

174. Based on Bugiel's teachings, a POSITA would have stored Bugiel's session-token ST_C in one of Chou's secure memory locations. *See* [1a-2]. For example, 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 or KEK unit 55 of Figure 3. Chou ¶21 ("Secure key module 23 is a storage medium, such as a memory, register, circuit, etc. that stores one or more key value(s) that is used by security processor 21 to encrypt and decrypt data. Secure memory 22 is utilized to store data that is processed and/or to be processed by security processor 21."); ¶35 ("The main processing operations for HSM 20 are provided by HSM

processor 51, which operates with its BROM 52 and RAM 53.”); ¶40 (“KEK unit 55 allows confidential operating context data to be wrapped and, subsequently, stored in a storage component outside of secure zone 80.”).

**FIG. 1****FIG. 3**

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

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

175. The Bugiel-Chou combinations (**Grounds 1A-1C**) satisfy this feature.

176. Bugiel discloses that one of its objectives is to provide “[a] secure channel between the user and the web server.” Bugiel, 221. Bugiel further discloses applications in which “a user deploys his mobile device to access services provided by remote web servers (over the Internet).” *Id.* According to Bugiel, after generating symmetric session key S_C , “[a]ll following messages should be encrypted using S_C .” *Id.*, 225; *see also id.*, 226 (“TruWalletM establishes the SSL/TLS connection and uses the first negotiated session key for authentication, i.e., the logical SSL/TLS channel is directly established between the WalletCore and WebServer. . . . the logical SSL/TLS channel between WalletCore and WebServer for a secure transmission of the credentials.”). Bugiel thus teaches establishing a secure communication session with a remote server over a network using a session key. A POSITA would have found it obvious to implement Bugiel using Chou’s hardware to achieve this.

177. Alternatively, as discussed in [1d], TLS-1.2 discloses that its TLS handshake protocol serves to establish secure communication between the client and server. *See* TLS-1.2, 1 (“This document specifies Version 1.2 of the Transport Layer Security (TLS) protocol. The TLS protocol provides communications security over

the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.”). For example, TLS-1.2 states that “the TLS Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data.” *Id.*, 4; *see also id.*, 6 (“TLS should be used to establish a secure connection between parties.”). TLS-1.2 emphasizes that 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 (providing overview of Handshake Protocol); *see also id.*, 4 (“The connection is private. Symmetric cryptography is used for data encryption The keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by another protocol (such as the TLS Handshake Protocol).”), 16-18, 25-27, 63-65, 77, 79, 81, 91; *see supra* [1d].

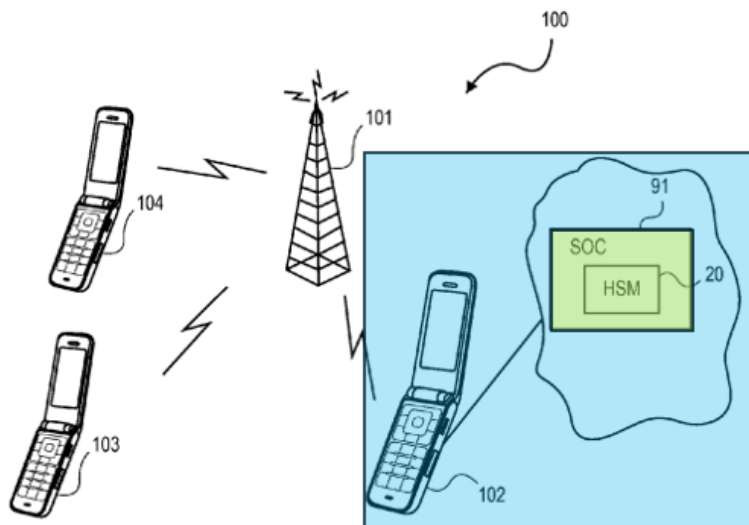
178. TLS-1.2’s disclosure of establishing secure communication with a remote server using a session key (e.g., its “master secret” or “key_block”) is consistent with the ’838 patent, which acknowledges in its background section that SSL protocols, including TLS, are used for establishing secure communication. *See* EX-1001, 1:10-25 (“One commonly used cryptographic communication protocol is the Secure Sockets Layer (SSL) protocol. The SSL protocol, and the related

Transport Layer Security (TLS) protocol, is used for many different types of secure communication sessions including, for example, secure web-browsing, ecommerce, secure upgrades, and other secure communication sessions between two computing devices on an untrusted network such as the Internet.”).

179. As discussed, the Bugiel-Chou combinations utilize Chou’s hardware to implement Bugiel’s and/or TLS-1.2’s communications methods, including implementing SSL/TLS Handshake Protocol messages, to establish a secure communication. Chou also describes performing symmetric encryption algorithms using the key algorithm module 15 or SPU-M 39 for encryption and decryption. *See* Chou, ¶¶26, 53-62, Figs. 1, 3. Chou’s hardware, including its system-on-a-chip, SOC 91, is designed to manage secure keys used to establish secure communications for various applications. *See* Chou ¶6 (“[S]ecure data transfer is utilized to perform secure financial transactions over a network, such as the Internet. . . . The recipient of the secure data utilizes a secure key to decrypt the data to retrieve the financial information.”), ¶69 (“[A] technique for providing secure key operation, in which a secure zone boundary is defined by components that make up a hardware security module (HSM) is described. Secure keys are wrapped/unwrapped and authenticated within the secure zone and do not leave the HSM in an unwrapped state.”).

180. Consistent with the applications Bugiel and TLS-1.2 envision, Chou discloses applications of its SOC 91 (the claimed “system-on-a-chip”) “implemented

in a mobile phone 102, such as a cell phone . . . operat[ing] within a cellular network 100.” Chou ¶68; *see also id.*, ¶¶65-69, Fig. 5. Chou explains that “[t]he various functional blocks of HSM 20, as well as the non-secure components as described above allow HSM 20 to operate within a mobile phone in one embodiment for practicing the invention.” *Id.* ¶¶65-69. In these embodiments, 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. This is consistent with Bugiel, which discloses a mobile device for securely communicating with a web server. *See* [1a-3]. In the Bugiel-Chou combinations, Chou’s SOC 91 would implement Bugiel’s and/or TLS-1.2’s teaching to establish secure communications with a remote server over a network, including a web server over the Internet and/or a cellular network.

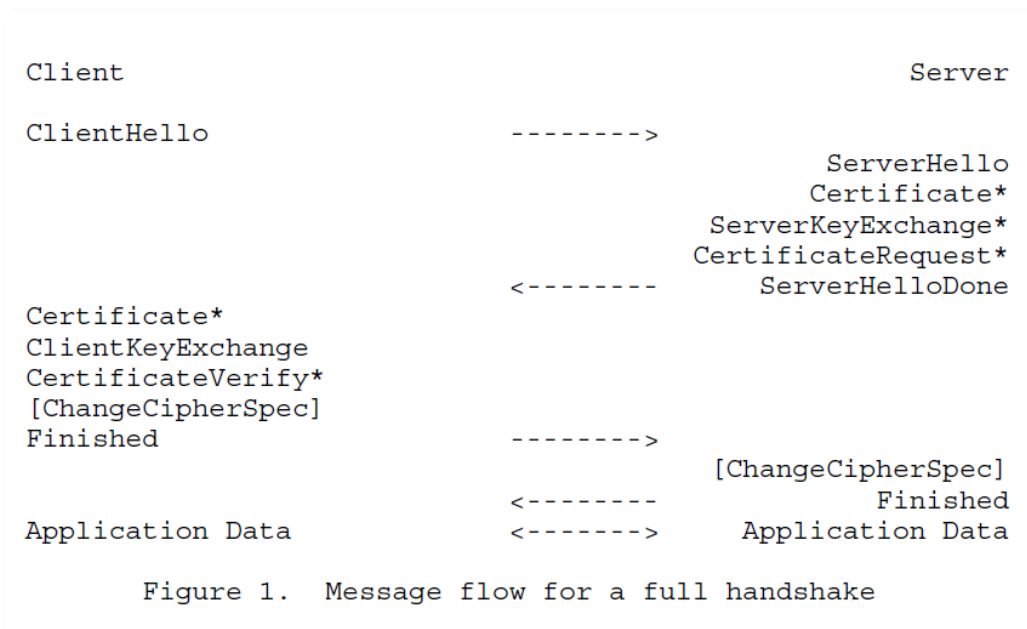
**FIG. 5**

Id., Fig. 5 (annotated).

4. **Claim 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.”**

181. The Bugiel-Chou combinations satisfy this feature.

182. As discussed above in [1c], the SSL/TLS handshake disclosed in Bugiel and TLS-1.2 includes a sever responding to a ClientHello message with a SeverHello message, followed by the server sending either a ServerKeyExchange message, a public key certificate in a certificate message, or both. TLS-1.2, 47-53; Bugiel, 226, Fig. 2; EX-1006, 38-51 (describing ServerKeyExchange message from server), 52-59 (describing certificate message or certificate message and ServerKeyExchange), Figs. 3-1, 3-5, 3-6, Tables 3-2, 3-9.



TLS-1.2, Fig. 1.

183. Figure 2 of Bugiel illustrates exchanging Client/Server Hello required in SSL/TLS handshake and specifically refers to TLS-1.2 for more details regarding TLS/SSL handshake messages, including the ServerHello message. *See* Bugiel, 225 (“All messages which are exchanged between WalletHelper and WebServer are standard messages of SSL/TLS handshake and can be found in the corresponding specification [16].” (citing TLS-1.2)).

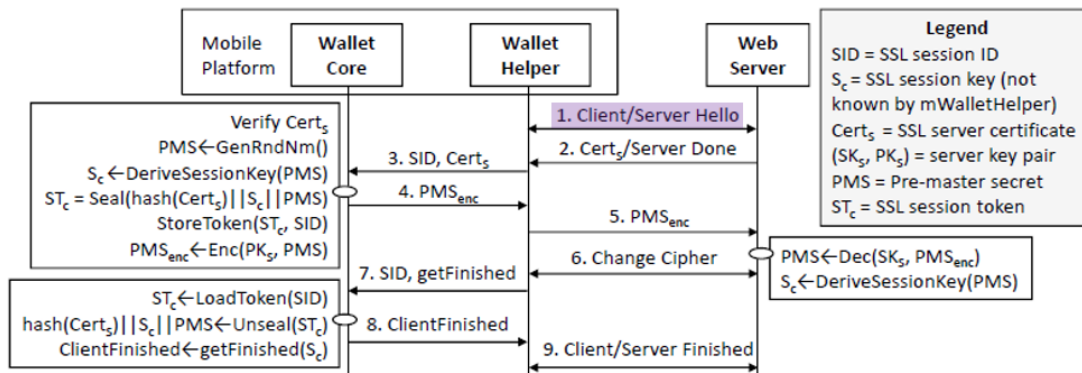


Fig. 2. SSL/TLS handshake

Id., Fig. 2 (annotated).

184. A POSITA would have understood, and TLS-1.2 expressly discloses, that the server in a TLS handshake sends a server nonce in the ServerHello message in response to the ClientHello request for a secure communication session from the client. As discussed in [1b], the client and server randoms are both random nonces to avoid “replay” attack. TLS-1.2 at 91-92. *See supra* [1b]. TLS-1.2 further explains that “[t]he client sends a ClientHello message to which the server must respond with a ServerHello message, or else a fatal error will occur and the connection will fail,”

and that “[t]he ClientHello and ServerHello establish the following attributes: Protocol Version, Session ID, Cipher Suite, and Compression Method,” and “two *random values* are generated and exchanged: ClientHello.random and *ServerHello.random*.” TLS-1.2, 34 (emphasis added), 42-43 (“The server will send this message [ServerHello message] in response to a client hello message”); *see also* EX-1006, Table 3-4 (ServerHello Components include “RandomNumber,” which is a “32-byte random number used to seed the cryptographic calculations.”), 43-44 (“this random value is chosen by the server”).

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.

185. TLS-1.2 discloses that a ServerHello.random (or server nonce) is one of the essential security parameters for calculating shared secrets and keys. TLS-1.2 explains that the variable server_random (i.e., the server random nonce) is part of

the SecurityParameters structure. TLS-1.2, 17-18 (defining “server random” as including “server_random” as one of the SecurityParameters); EX-1006, 44 (random number “seeds important cryptographic calculations”). TLS-1.2 also teaches using the server nonce to generate a master key and an encryption key_block. TLS-1.2, 26, 64; EX-1006, 88-90. TLS-1.2 explains: “[f]or TLS to be able to provide a secure connection, . . . keys . . . must be secure.” TLS-1.2, 96.

186. Bugiel-Chou and Bugiel-Chou-TLS-1.2 combinations therefore disclose receiving a server nonce from a remote server in a response message to the request for the secure communication session, the response message including a server nonce, as claimed.

187. The Bugiel-Chou combinations further satisfy the security engine receiving the server nonce and storing the server nonce in the secure memory. For example, the Bugiel-Chou combinations use Chou’s hardware to implement Bugiel’s communications method, including implementing the SSL/TLS Handshake Protocol messages disclosed in Bugiel and TLS-1.2. *See* [1p].

188. The client and server random nonces are exchanged in unencrypted client and server hello messages transmitted over a network. However, because the server and client random nonces are used to derive the master secret and the security key is secret and must be kept in secure memory, the random nonces must be kept

in secure memory as well because secure memory is separate from open memory and the trust boundary cannot or should not be crossed by the secure processor.

189. A POSITA therefore would have understood and found it obvious in the Chou-Bugiel combinations that a server nonce received from a remote server would be stored in Chou's secure memory (e.g., secure memory 22, HSM BROM 52, HSM RAM 53, and/or memories associated with the secure processing of the sever nonce by e.g., the security processor 21, HSM processing 51, and/or PKA 56, etc.). A POSITA would have found it obvious based on Chou to do so by communicating data, including the nonce, via security interface 24/interface 42/IPC/Msg RAM module 61 to the HSM 20. *See id.* ¶22, ("For HSM 20, all data transfers between the secure domain and non-secure (unsecure) domain [are] routed through security interface 24. As shown, a connection or bus 27 couples security interface 24 to bus 16, so that all accesses to HSM 20 and data transfers into and out of HSM 20 are routed through security interface 24."), ¶23 ("Security interface 24 . . . may operate under control of security processor 21 [of HSM 20]."), ¶25 ("the only route to internal hardware components of HSM 20 is through security interface 24"), ¶35 ("In one embodiment, bus 72 is an AHB bus. IPC/Msg RAM module 61 operates as an interface/bridge between bus 71 and bus 72, as well as the only access pathway to exchange data between HSM 20 and the non-secure domain. Bus 72 is coupled to bus 41 via interface or bridge 42. . . . Thus, similar to security interface 24 of Fig. 1,

IPC/Msg RAM module 61 controls the ingress and egress of instructions and/or data between components within HSM 20 and components outside of HSM 20”). Chou therefore discloses exchanging data, including data for encryption and decryption purposes, from its unsecure zone to its secure zone for processing and storage on HSM 20.

190. This is consistent with Bugiel’s TrEE WalletCore operations and TLS-1.2’s client-side processing, which store and process server-side information to engage in an TLS handshake. *See* Bugiel, 225-226, Fig. 1 (depicting connections between WalletHelper and WalletCore), Fig. 2 (depicting WalletCore processing a SeverHello message and Server certificate in the TLS handshake).

191. Chou also 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.* (“Security processor 21 provides the various processing within HSM 20, including key authentication and the encrypting and/or decrypting operation to encrypt and/or decrypt data.”), ¶35 (“The main processing operations for HSM 20 are provided by HSM processor 51, which operates with its BROM 52 and RAM 53”). Chou also discloses that “the unwrapping of secure keys is done solely within HSM.” *Id.* ¶25; *see also id.* ¶22. In one aspect, a POSITA would have understood the Chou-Bugiel combination would have stored the server nonce in secure memory 22.

192. 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.

5. Claim 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.”

193. The '838 patent only discloses encrypting a “pre-master key” using a *server public key* provided by the server before sending the encrypted pre-master key to the server. *See* EX-1001, 8:28-34, 10:60-64. For purposes of this declaration, I have been asked to assume that the disclosure in the '838 patent of encrypting a pre-master key using the server's public key (rather than the “security key”) is sufficient to support claim 3 as written (i.e., encrypting the pre-master key using the security key of the security engine). The Bugiel-Chou combinations implement TLS-1.2 and thus satisfy this feature.

194. Bugiel discloses generating a “PMS,” or a “Pre-master secret” during the TLS 1.2 cryptographic key exchange. Bugiel, 225-226, Fig. 2. As discussed in [1d], Bugiel discloses verifying a server certificate *Certs* and generating a pre-master key (PMS) to derive a session key S_C from the PMS. *See* Bugiel, Fig. 2 (“ $PMS \leftarrow \text{GenRndNm}()$,” and “ $S_c \leftarrow \text{DeriveSessionKey}(PMS)$ ”).

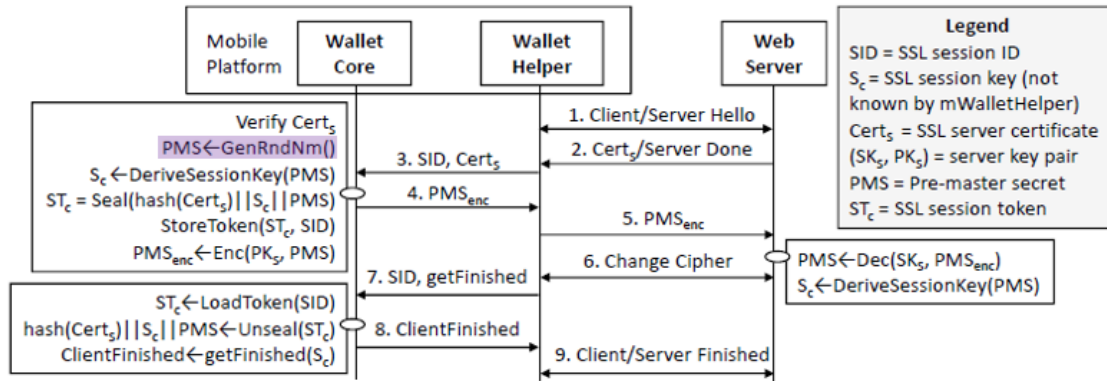


Fig. 2. SSL/TLS handshake

Id., Fig. 2 (annotated).

195. The Bugiel-Chou combinations also encrypt the pre-master key before sending the encrypted pre-master key to the server. For example, Bugiel's Figure 2 teaches 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).

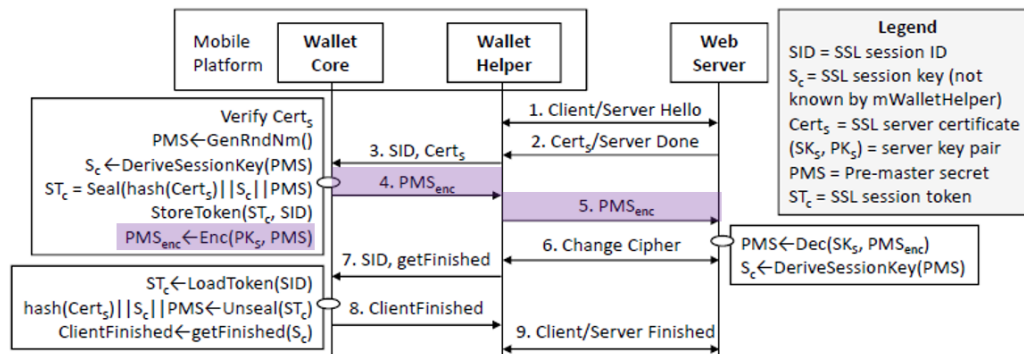


Fig. 2. SSL/TLS handshake

Id., Fig. 2 (annotated).

196. 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 (ServerKeyExchange message that "conveys cryptographic information to allow the client to communicate the premaster secret: a Diffie-Hellman public key with which the client can complete a key exchange (with the result being the premaster secret) or a public key for some other algorithm."), 57 (ClientKeyExchange message sent when "the premaster secret is set, either by direct transmission of the RSA-encrypted secret or by the transmission of Diffie-Hellman parameters that will allow each side to agree upon the same premaster secret."), 58 (the Pre-Master key being a "random value is generated by the client and is used to generate the master secret."), 91-93 (same), 34, 50-51.

197. 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 (RSA encrypted premaster secret message). After the ServerHello is completed, 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)." TLS-1.2, 57-58 ("If RSA is being used for key agreement and authentication, the client generates a 48-byte premaster secret, encrypts it using the public key from the

server's certificate, and sends the result in an encrypted premaster secret message"), 74; *see* EX-1006 40-41. The "encryption protects the key information as it traverses the network, and it allows the client to verify that the server truly possess the private key corresponding to its public key. Otherwise, the server won't be able decrypt the message." EX-1006, 46. If RSA encryption is used, "the ClientKeyExchange carries a premaster secret." TLS-1.2 58-59 (providing the structure of the ClientKeyExchange message); EX-1006, 85-87, Fig. 4-21; *see also id.*, 86 ("The message body itself consists solely of the encrypted premaster secret" using the public key of the server).

198. Therefore, the Bugiel-Chou combinations, including Grounds 1A-1C, satisfy generating a pre-master key, encrypting the pre-master key according to TLS-1.2, and sending the encrypted pre-master key to a remote server.

199. A POSITA would have understood that, in the Bugiel-Chou combinations, Chou's HSM 20 (e.g., part of the claimed "security engine") would have implemented Bugiel's and/or Bugiel and TLS-1.2's teaching of generating a pre-master secret. For example, Chou's Figure 3 (below) illustrates that PKA module 56 of HSM 20 is communicatively coupled with KEK unit 55 via bus 71. *See* Chou ¶35 ("Other units 54-69 are also coupled to bus 71."). Additionally, Chou's PKA module 56 communicates with the non-secure zone through interfaces 42, bus 72 and IPC Msg RAM 61. *See* Chou ¶¶35-36 ("IPC/Msg RAM module 61

operates as an interface/bridge between bus 71 and bus 72, as well as the only access pathway to exchange data between HSM 20 and the non-secure domain.”). Using these connections, Chou’s PKA module 56 would load 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. In this way, a POSITA would have found it obvious to use Chou’s hardware to encrypt the pre-master key using the server public key as Bugiel and TLS-1.2 describe. *See* Bugiel, 225-226, Fig. 2; TLS-1.2, 57-61.

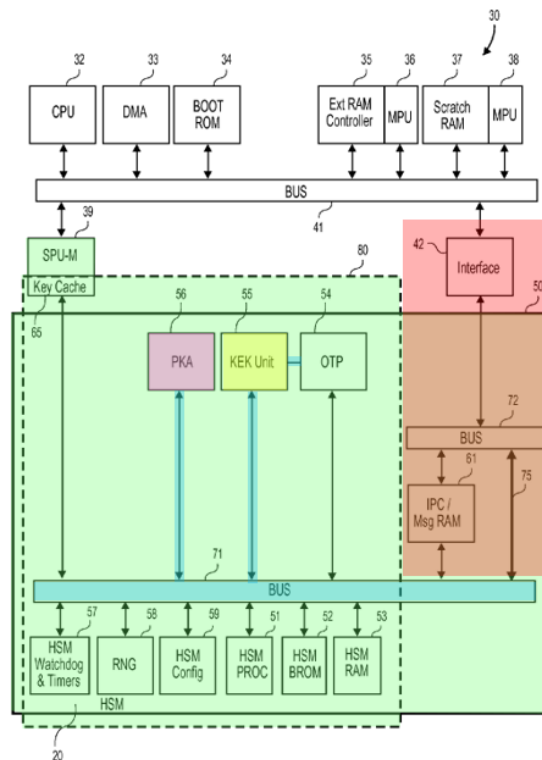


FIG. 3

Id., Fig. 3 (annotated).

200. The '838 patent discloses that “[t]he keys and related keying data generated during the client key exchange may be stored in the secure memory 114

in an encrypted state (using the security key 150) or an unencrypted state.” EX-1001, 11:6-9. To the extent it is argued this supports claim 3, the Bugiel-Chou combinations provide the same disclosure and thus would satisfy claim 3.

201. In the Bugiel-Chou combinations, Chou’s HSM 20 (e.g., the “security engine”) 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). Bugiel explains that “the SSL/TLS session key S_C is generated, never disclosed to WalletHelper, and sealed together with pre-master secret PMS and the hash value $\text{hash}(\text{Certs})$), resulting in session-token ST_C that is “stored in a secure storage.” Bugiel, 225-226; *see also id.*, Fig. 2. Bugiel explains that “[s]ealing means protecting an object so that only a certain set of entities can access or use it,” *id.*, 225 & n.8, and that “[w]hen sealed, data is encrypted under a key that is never available outside the TrEE,” *id.*, 231; *see also id.*, 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.”

202. A POSITA would have found it obvious to use Chou’s HSM 20 to implement this aspect of Bugiel’s method, where KEK 55 would encrypt the PMS (pre-master secret) using Chou’s device unique key/root key stored in OTP 54. As discussed in [1a-3], PKA module 56 and KEK Unit 55 of the HSM are communicatively coupled by bus 71. *See* Chou ¶35. Additionally, “a direct link

exists between KEK Unit 55 and OTP so that the key stored in OTP 54 may be loaded into KEK unit 55 without transitioning onto bus 71.” *Id.* ¶39. In this way, KEK unit 55 loads the pre-master secret generated by PKA module 56 and the device unique key/root key encoded in OTP 54 to encrypt the pre-master key using the device unique key/root key.

203. A POSITA thus would have found it obvious to use Chou’s security hardware (which provides the TrEE Bugiel requires) to encrypt the claimed “pre-master key” (Bugiel’s pre-master secret, PMS) using the claimed “security key” (Chou’s device unique key/root key, which is an encryption key never available outside the TrEE, *see* Chou ¶¶25, 40, as Bugiel teaches, *see* Bugiel, 225-226, n.8, 231. Accordingly, in the combination of Bugiel, Chou, and optionally TLS-1.2, HSM 20 encrypts the pre-master secret using a key that is never available outside the security engine, and HSM 20 also stores the encrypted pre-master key in the secure storage (e.g., Chou’s secure memory 22), which “store[s] data that is processed and/or to be processed by security processor 21.” Chou ¶21); *see also* [1f].

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

204. The ’838 patent uses the terms “encrypted” and “wrapped” synonymously. *See* EX-1001, 5:67-6:1 (“master key may be stored in an encrypted or wrapped form (e.g., encrypted using the security key 150).”), 6:6-9 (same), 8:29-

32 (“encrypted or otherwise wrapped using the server public key provided by the server in server authentication and key exchange phase”). A POSITA also would have understand that wrapping a key is another way to refer to encrypting a key. The Bugiel-Chou combinations therefore satisfy the security engine wrapping the pre-master key using a server public key for the same reasons the pre-master key is encrypted with the server public key, as discussed in claim 3. *See* claim 3.

205. The Bugiel-Chou combinations also “store” the wrapped pre-master key in the secure memory. For example, 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]. In addition, the encrypted pre-master secret is data that must be stored (even if temporarily) to be processed. TLS-1.2 teaches this expressly by storing the pre-master key to compute a master secret using the pre-master key. *See* TLS-1.2, 64 (explaining the pre-master secret should be deleted from memory once the master secret has been computed). In the Bugiel-Chou combinations, a POSITA would have stored the encrypted (e.g., “wrapped”) pre-master key in Chou’s secure memory (e.g., in secure memory 22 or, for example, in HSM RAM 53).

206. Chou explains that “[s]ecurity processor 21 provides the various processing within HSM 20, including key authentication and the encrypting and/or decrypting operation to encrypt and/or decrypt data,” and that “[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 (“HSM RAM 53 may be a scratch RAM in one embodiment, but other types of RAM or memory may be used.”), ¶¶43-53 (PKA unit 56 operable to perform “Diffie-Hellman public value and shared key computation” and “RSA encryption”). A POSITA thus would have found it obvious to generate and store 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 the security processor 21 or PKA unit 56). *See also* Chou ¶35 (“The main processing operations for HSM 20 are provided by HSM processor 51, which operates with its BROM 52 and RAM 53”), Figs. 1, 3.

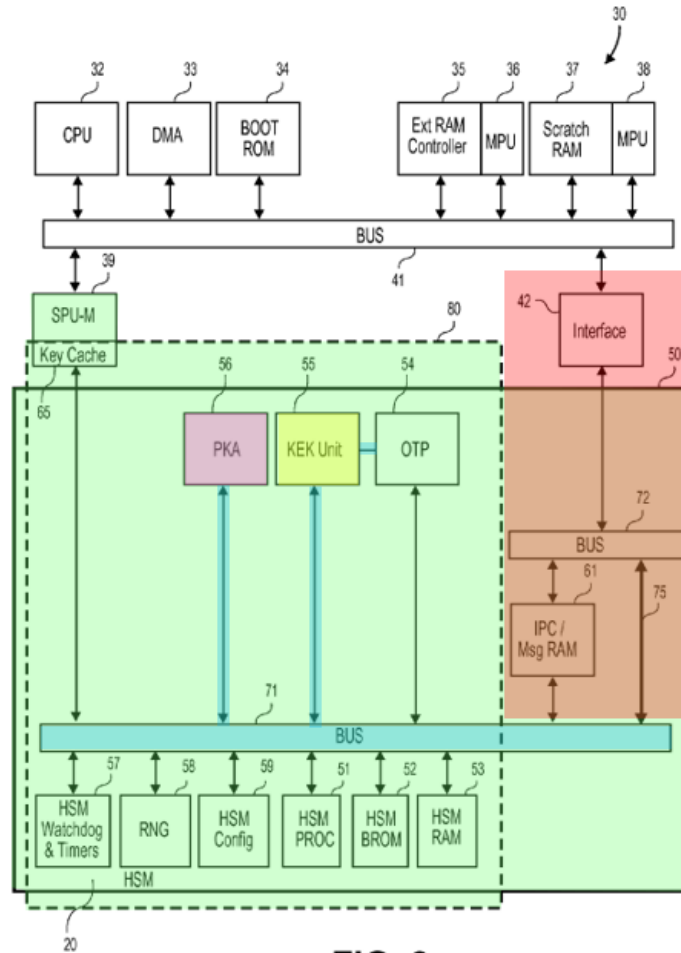


FIG. 3

Id., Fig. 3.

7. **Claim 5**

- a. [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”

207. As discussed in [1d], Bugiel discloses verifying a server certificate *Certs*, generating a pre-master key (PMS), and then deriving a session key S_C (e.g., the “symmetric session key”) from the PMS. *See* Bugiel, Fig. 2 (“PMS $\leftarrow$ GenRndNm(),” and “ $S_C \leftarrow$ DeriveSessionKey(PMS)”). 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 (Key Calculation). A POSITA would have found it obvious to implement Bugiel's and/or TLS-1.2's methods using the hardware features in Chou. *See* [1d]; Chou ¶¶43-53, Fig. 3.

b. [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.”

208. The Bugiel-Chou combinations satisfy this feature. For example, Bugiel teaches the need to perform a hash algorithm. *See* Bugiel, 225 (“the hash value *hash(Certs)*”), Fig. 2.

209. TLS-1.2 discloses this feature in several ways. First, TLS-1.2 teaches calculating the master secret by calculating a hash function as a function of the random nonce generated by the client's security engine and a nonce received from the remote server. TLS-1.2 states: “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-1.2 states: “TLS's PRF is created by applying ***P-hash to the secret*** as: PRF(secret, label, seed) = P_<hash>(S1, label + seed).” *Id.*, 15 (HMAC and the Pseudorandom Function) (emphasis added). As these disclosures demonstrate, this involves calculating 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 a random nonce (ServerHello.random) received from the remote server (claim 2).

210. Second, TLS-1.2 discloses calculating the encryption key_block by calculating a hash function as a function of the random nonce generated by the client's security engine and a random nonce received from the remote server. TLS-1.2 states: "key_block = PRF(SecurityParameters.master_secret, 'key expansion', SecurityParameters.server_random + SecurityParameters.client_random." TLS-1.2., 26; *see also id.*, 94 ("The master secret is hashed with the ClientHello.random and ServerHello.random to produce unique data encryption keys . . ."). Just like the master secret, creating encryption key_block therefore involves calculating a hash function (the PRF function) as a function of the random nonce (ClientHello.random) generated by the client's security engine (*see* [1b]) and a random nonce (ServerHello.random) received from the remote server (claim 2).

211. Third, TLS-1.2 teaches generating a Finished message by calculating a hash function as a function of the random nonce generated by the client's security engine and a random nonce received from the remote server, where the Finished message includes a hash of all the handshake messages. For instance, TLS version 1.2 calculates the following: "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 also involves calculating a hash function (the PRF function) as a function of the random nonce (ClientHello.random) generated by the client's security engine (*see* [1b]) and a random nonce (ServerHello.random) received from the remote server (claim 2).

212. Fourth, TLS-1.2 explains that the client generates a Certificate Verify message after a client key exchange message, whereby the Certificate Verify message "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). After receiving the ServerHello message, the client thus generates a Certificate Verify message that "compute[s] running hashes for all potential hash algorithms" on the 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 involves calculating a hash function (the hash algorithm) as a function of the random nonce (ClientHello.random) generated by the security engine (*see* [1b]) and a random nonce (ServerHello.random) received from the remote server (claim 2).

213. As discussed, Bugiel's and TLS-1.2's methods would have been implemented by Chou. Indeed, Chou is designed to calculate a hash function. *See* Chou ¶61 ("MD5/SHA1/SHA224/SHA256 combo engine to support pure hash or HMAC operations.").

8. Claim 6

- a. [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,"**

214. The Bugiel-Chou combinations satisfy this feature. As contemplated by the TLS-1.2 protocol, a POSITA would have understood that the symmetric key or master secret may be generated based on either (RSA) key exchange or a Diffie-Hellman key exchange. *See* TLS-1.2, 55-57 (client certificate including RSA or Diffie-Hellman parameters), 57-58 (client key exchange message based on RSA or Diffie-Hellman), 61, 65, 74-75, 86, 92-93, 100; EX-1006, 81-84 (describing different ServerKeyExchange message formats for Diffie-Hellman, RSA, and Fortezza key exchange protocols), 40-45 (ServerKeyExchange message), 54-59 (separate ServerKeyExchange message and server certificate); *see* [1c] (describing ServerKeyExchange message with or without certificate and using Diffie-Hellman or RSA encryption techniques).

215. To implement a Diffie-Hellman key exchange, the Bugiel-Chou combinations would have implemented the Diffie-Hellman key exchange teachings

of TLS-1.2 incorporated by reference in Bugiel on Chou's hardware. *See* Bugiel, 225-226; *see* Sections VIII.A.1, VIII.A.3.f ([1c]). That is, the master secret (e.g., the symmetric session key) would be 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 ("With this [ClientKeyExchange] message, the premaster secret is set, either by direct transmission of the RSA-encrypted secret or by the transmission of Diffie-Hellman parameters"); *see also id.*, 61 ("This structure conveys the client's Diffie-Hellman public value (Yc)"); *see* EX-1006, 85-88 (providing contents of ClientKeyExchange message for RSA and Diffie-Hellman).

216. 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. *See* TLS-1.2, 57-61 (Client Diffie-Hellman Public Value). TLS-1.2 also generates a private Diffie-Hellman client key by "generating a fresh DH private key (X) for each handshake." *See id.*, 93, 87 (cautioning to use a strong and properly seeded random number generator for generating Diffie-Hellman private values).

217. TLS-1.2 suggests implementing Diffie_Hellman key exchange according to RFC 2785 ("Methods for Avoiding the 'Small-Subgroup' Attacks on the Diffie-Hellman Key Agreement Method for S/MIME"). *See id.*, 93 (citing [SUBGROUP]), 100 (defining [SUBGROUP] as RFC 2785). RFC 2785 explains

that, during Diffie-Hellman Key Exchange between Party A and Party B, each of parties A and B generates a **private key** (x_a and x_b , respectively) and then generates a **public key** (y_a and y_b , respectively) based on its private key:

```

ya is Party A's public key;  $y_a = g^{x_a} \bmod p$ 
yb is Party B's public key;  $y_b = g^{x_b} \bmod p$ 
xa is Party A's private key;  $x_a$  is in the interval  $[2, (q - 2)]$ 
xb 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 mod p)
q is a large prime
j a large integer such that  $p = q*j + 1$ 

```

EX-1011, 2

218. 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. The Bugiel-Chou combinations implementing Diffie-Hellman, including RFC 2785, generates a public Diffie-Hellman client key and a private Diffie-Hellman client key.

219. As discussed, Bugiel’s and TLS-1.2’s methods would have been implemented by Chou. Indeed, Chou teaches that HSM 20 (part of the claimed “security engine”) includes a PKA module 56 that 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.

- b. **[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.”**

220. The Bugiel-Chou combinations satisfy this feature. For example, during the TLS Handshake protocol, it is necessary to sign the public Diffie-Hellman client key and send the signed public Diffie-Hellman key to the remote server. *See* TLS-1.2, 62 (Certificate Verify). TLS-1.2 teaches that the public Diffie-Hellman client key Yc (the claimed “public Diffie-Hellman client key”) needs to be sent to the server so the server can authenticate the client and verify the client has corresponding private key. *See id.* (showing the client key exchange message includes the public Diffie-Hellman client key Yc). More specifically, TLS-1.2 teaches sending a Certificate Verify message after the ClientKeyExchange message. The Certificate Verify message includes “digitally-signed” version of all previous handshake messages, which would include the client key exchange message that contains public Diffie-Hellman client key Yc. According to TLS-1.2, the purpose of sending the Certificate Verify message is to “provide explicit verification of a client certificate.” *Id.* (Certificate Verify).

221. TLS-1.2 also suggests using a key specific to the client device so the server could verify and authenticate the signed public Diffie-Hellman client key based a device-specific key. *See* TLS-1.2, 33-37 (“Exchange certificates and cryptographic information to allow the client and server to authenticate themselves

... and to explicitly verify possession of the private key in the certificate.”). A POSITA would have found it obvious to have verified and authenticated the signed public Diffie-Hellman client key using the claimed security key. *See* [1a-3]; *See* Chou ¶38 (explaining “OTP 54 is a non-volatile memory that is used to one-time program a key value or values for device 30. For example, the device unique key may be programmed and stored in OTP 54.”).

9. Claim 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.”

222. The Bugiel-Chou combinations satisfy this feature.

223. For example, TLS-1.2 discloses that the client receives a public Diffie-Hellman server key from the remote server by receiving a public Diffie-Hellman server key `dh_Y` to complete a key exchange and to calculate the premaster secret. *See* TLS-1.2, 50-52 (ServerKeyExchange message).

224. TLS-1.2 discloses generating the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key in two ways.

225. First, TLS-1.2 generates the master secret (e.g., the “session key”) as a function of the pre-master secret. *See id.*, 64 (“`master_secret = PRF(pre_master_secret, “master secret”, ClientHello.random + ServerHello.random)`”). The client device generates the premaster secret by performing conventional Diffie-

Hellman using the public Diffie-Hellman server key received from the server. *See id.*, 51 (explaining the result of Diffie-Hellman key exchange is the “premaster secret”), 65 (Diffie-Hellman); *see also id.*, 93 (using RFC 2785 to implement Diffie-Hellman when calculating the premaster secret). RFC 2785, in turn, teaches that a shared key (ZZ), i.e., the pre-master key, is generated based on the calculation below.

Note that the individual parties actually perform the computations:

$$ZZ = (yb ^ xa) \bmod p = (ya ^ xb) \bmod p$$

where ^ denotes exponentiation.

ya is Party A's public key; $ya = g ^ xa \bmod p$

yb is Party B's public key; $yb = g ^ xb \bmod p$

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

$g = h^{(p-1)/q} \bmod p$, where

h is any integer with $1 < h < p-1$ such that $h^{(p-1)/q} \bmod p > 1$
(g has order q mod p)

q is a large prime

j a large integer such that $p=q*j + 1$

EX-1011, 2.

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

227. The pre-master secret is therefore a function of the private Diffie-Hellman client key xa and the public Diffie-Hellman server key yb. Because

TLS-1.2 teaches generating the master secret as a function of the pre-master secret, and the premaster secret is a function of the private Diffie-Hellman client key and public Diffie-Hellman server keys, the master secret (the claimed “session key”) is generated as a function of the public Diffie-Hellman server key and private Diffie-Hellman client key.

228. Second, TLS-1.2 teaches generating the `key_block` (the claimed “session key”) as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key. TLS-1.2 discloses (1) generating the `key_block` (the claimed “session key”) as a function of master secret; (2) generating the master secret as a function of pre-master secret; and (3) generating a premaster secret as a function of the private Diffie-Hellman client key x_a and the public Diffie-Hellman server key y_b (as discussed above).

229. TLS-1.2 explains that the `key_block` is generated as a function of the master secret. See TLS-1.2, 26 (“`key_block = PRF(SecurityParameters.master_secret, ‘key expansion’, SecurityParameters.server_random + SecurityParameters.client_random)’”). Because the master secret is generated as a function of the pre-master secret (as discussed above), and because the pre-master secret generated as a function of the private Diffie-Hellman client key and public Diffie-Hellman server key (as also`

discussed above), the `key_block` is generated as a function of the private Diffie-Hellman client key and public Diffie-Hellman server key. *See* [1d].

- 10. Claim 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.”**

230. The Bugiel-Chou combinations satisfy this feature. As discussed in claim 6, a POSITA would have understood that the symmetric key or master secret in the TLS-1.2 protocol may be generated by either (RSA) key exchange or a Diffie-Hellman key exchange. *See* TLS-1.2, 55-57 (client certificate including RSA or Diffie-Hellman parameters), 57-58 (client key exchange message based on RSA or Diffie-Hellman), 61, 65, 74-75, 86, 92-93, 100; *see* EX-1006, 81-84 (describing different `ServerKeyExchange` message formats for Diffie-Hellman, RSA, and Fortezza key exchange protocols), 40-45 (`ServerKeyExchange` message), 54-59 (separate `ServerKeyExchange` message and server certificate); *see* [1c] (describing `ServerKeyExchange` message with or with certificate). To implement a Diffie-Hellman key exchange, the Bugiel-Chou combinations would have implemented the Diffie-Hellman key exchange teachings of TLS-1.2 on Chou’s hardware.

231. In doing so, 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 (ServerKeyExchange Message). 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.

232. In the combination, Chou’s HSM 20 (part of the claimed “security engine”) generates the session key using the TLS/SSL Handshake protocol, such that it must receive the TLS-1.2 server key exchange message, which would have involved using Chou’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* [1c]-[1d]. For example, Chou’s HSM 20 includes a PKA module 56 that 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.

233. 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 Chou’s secure memory so the global values and public Diffie-Hellman server key computation can take place to compute the shared symmetric key. A POSITA would have found it obvious to store them in Chou’s secure memory (e.g., secure memory 22, HSM RAM 53, etc.) to compute the associated secrets and encryption keys. As explained in claim 2, a POSITA would have done so by communicating data via

security interface 24 or interface 42/IPC/Msg RAM module 61 to the HSM 20. *See id.* ¶¶22-23, 35-36.

11. Claim 9

- a. **[9a] “A method comprising: 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”**

234. The Bugiel-Chou combinations satisfy this feature. *See* [1p]-[1a-3].

- b. **[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;”**

235. The Bugiel-Chou combinations satisfy this feature. *See* [1b].

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

236. The Bugiel-Chou combinations satisfy this feature. *See* [1c].

- d. **[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;”**

237. The Bugiel-Chou combinations satisfy this feature. *See* [1d].

- e. **[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;”**

238. The Bugiel-Chou combinations satisfy this feature. *See* [1a-2], [1a-3], [1e].

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

239. The Bugiel-Chou combinations satisfy this feature. *See* [1a-2], [1f].

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

240. The Bugiel-Chou combinations satisfy this feature. *See* [1g].

12. Claim 10

- a. **[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”**

241. The Bugiel-Chou combinations satisfy this feature. *See* [1b].

- b. **[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.”**

242. The Bugiel-Chou combinations satisfy this feature. *See* [1b], claim 2.

- 13. **Claim 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.”**

243. The Bugiel-Chou combinations satisfy this feature. *See* [1d], claim 3.

14. **Claim 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.”**

244. The Bugiel-Chou combinations satisfy this feature. *See* claim 4.

15. **Claim 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.”**

245. The Bugiel-Chou combinations satisfy this feature. *See* [1d], claim 3, [5a].

16. **Claim 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.”**

246. The Bugiel-Chou combinations satisfy this feature. *See* [1d], [5b].

17. **Claim 15**

- a. **[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;”**

247. The Bugiel-Chou combinations satisfy this feature. *See* [1d], [6a].

- b. **[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;”**

248. The Bugiel-Chou combinations satisfy this feature. *See* [6b].

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

249. The Bugiel-Chou combinations satisfy this feature. *See* claim 7.

- d. **[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.”**

250. The Bugiel-Chou combinations satisfy this feature. *See* claim 7.

18. Claim 16

- a. **[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;”**

251. The Bugiel-Chou combinations satisfy this feature. *See* claim [5a], [5b].

252. TLS-1.2 teaches, for example, sending a client Certificate Verify message includes a hash of prior handshake messages. TLS-1.2, 62 (Certificate Verify); *see* [5b]. Those prior handshake messages include all handshake messages sent or received by the client, starting with the ClientHello message. *See id.* The client Certificate Verify message sent following the client key exchange message includes a hash code of the ClientHello and client key exchange messages (the claimed “a preceding message sent from the client to the remote server”).

b. [16b] “signing the hash code using the security key stored in the secure memory of the security engine; and;”

253. The Bugiel-Chou combinations satisfy this feature.

254. For example, TLS-1.2 explains the purpose of sending the Certificate Verify message is to “provide explicit verification of a client certificate.” TLS-1.2, 62 (Certificate Verify). Although TLS-1.2 may not specify a specific key to be used for signing the hash code, the purpose of signing in the TLS protocol is to allow the server to verify the identity of the client device. A POSITA therefore would have found it obvious to use a key that is specific to the client to sign the hash code.

255. As discussed, Chou discloses such a security key stored in the secure memory of the security engine. *See* [1a-3]. For example, Chou explains that a device unique key/root key (the claimed “security key”) “may be programmed and stored” in OTP 54. *See* Chou ¶38. Further, Chou explains that PKA module 56 (part of the claimed “security engine”) performs digital signing by using a 1024-bit DSA (Digital Signature Algorithm) signature. *See id.* ¶45.

256. Because the device unique key/root key in Chou’s security engine is unique to the client, it would have been an obvious choice to use as a device-specific key to sign the hash code in the combination because it allows the server to verify the identity of the client device.

- c. **[16c] “sending a client certificate verification message to the remote server, the client certificate verification message including the signed hash code.”**

257. The Bugiel-Chou combinations satisfy this feature by sending a client Certificate Verify message. TLS-1.2, 62 (Certificate Verify); *see* [5b], [16a].

19. Claim 17

- a. **[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:”**

258. The Bugiel-Chou combinations satisfy this feature. *See* [1p]-[1a-3]. A POSITA would have understood that implementing the Bugiel-Chou combinations includes implementing Bugiel and TLS-1.2’s functionality using a computing device comprising Chou’s hardware. Chou’s hardware includes 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 perform functions. *See* [1p]-[1a-3].

- b. **[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;”**

259. The Bugiel-Chou combinations satisfy this feature. *See* [1p]-[1a-3], [1b].

- c. **[17b] “initiate a request for a secure communication session with a remote server over a network, the request including the random nonce;”**

260. The Bugiel-Chou combinations satisfy this feature. *See* [1b].

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

261. The Bugiel-Chou combinations satisfy this feature. *See* [1c].

- e. **[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;”**

262. The Bugiel-Chou combinations satisfy this feature. *See* [1d].

- f. **[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;”**

263. The Bugiel-Chou combinations satisfy this feature. *See* [1p]-[1a-3],
[1e].

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

264. The Bugiel-Chou combinations satisfy this feature. *See* [1a-2], [1f].

- h. **[17g] “establish the secure communication session with the remote server using the session key”**

265. The Bugiel-Chou combinations satisfy this feature. *See* [1f].

20. Claim 18

- a. **[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”**

266. The Bugiel-Chou combinations satisfy this feature. *See* [10a], [1b].

- b. **[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.”**

267. The Bugiel-Chou combinations satisfy this feature. *See* [10b], claim 2, [1b].

- 21. Claim 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.”**

268. The Bugiel-Chou combinations satisfy this feature. *See* claim 3.

- 22. Claim 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.”**

269. The Bugiel-Chou combinations satisfy this feature. *See* claim 4.

23. **Claim 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.”

270. The Bugiel-Chou combinations satisfy this feature. *See* [1d], [5a].

24. **Claim 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.”

271. The Bugiel-Chou combinations satisfy this feature. *See* [1d], [5b].

25. **Claim 23**

- a. [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”

272. The Bugiel-Chou combinations satisfy this feature. *See* [1d], [6a].

- b. [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;”

273. The Bugiel-Chou combinations satisfy this feature. *See* [6b].

- c. [23c] “receive a public Diffie-Hellman server key from the remote server; and”

274. The Bugiel-Chou combinations satisfy this feature. *See* claim 7.

- d. [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.”

275. The Bugiel-Chou combinations satisfy this feature. *See* claim 7.

26. **Claim 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.”

276. The Bugiel-Chou combinations satisfy this feature. *See* [1p]-[1a-3]. For example, Chou’s Figures 1 and 3 show that 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, Figs. 1, 3, ¶¶22, 30, 33, 35-36.

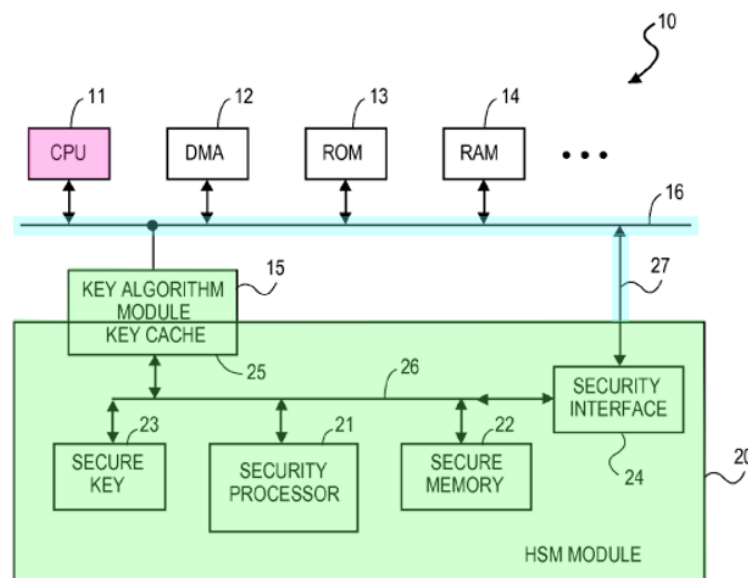


FIG. 1

Id., Fig. 1 (annotated).

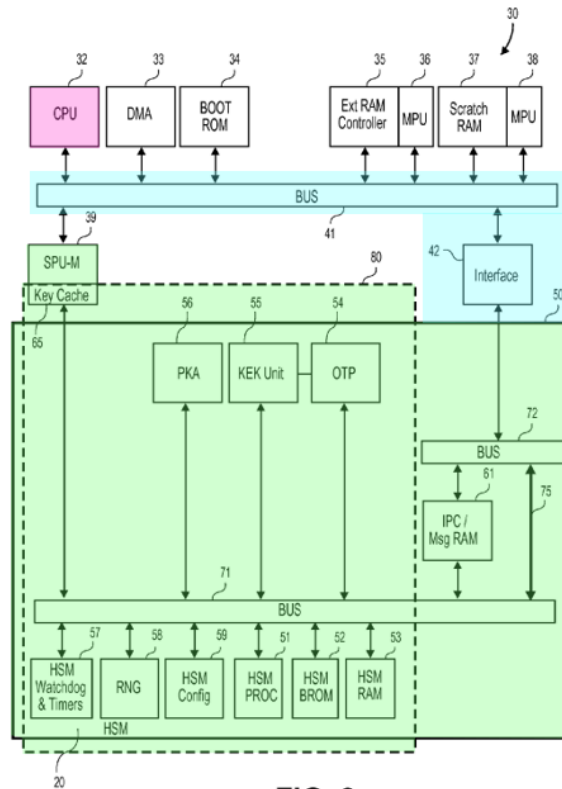


FIG. 3

Id., Fig. 3 (annotated).

277. This is consistent with the '838 patent, which describes an “interconnect such as a bus, point-to-point, or other interconnect capable of facilitating communication between the various components of the SOC.” EX-1001, 3:67-4:3.

B. Grounds 2A, 2B: Tkacik (EX-1002) and TLS-1.2, and Tkacik, TLS-1.2, and Bugiel (EX-1002) Render Obvious Claims 1-24

1. Motivation to Combine Tkacik and TLS-1.2

278. A POSITA would have found it obvious to combine Tkacik and TLS-1.2.

279. Tkacik discloses that “keys may be generated using any of a number of protocols known to those skilled in the art, including IKE, DH, RSA key gen., SSL, random key, and the like.” Tkacik ¶80. SSL/TLS was also one of the most popular secure communication protocols at the relevant time. *See* EX-1006, 6. TLS-1.2 was published and in use nearly three years before the filing of the ’838 patent and therefore would have been mature enough for a POSITA to implement in Tkacik’s system-on-a-chip architecture, which Tkacik expressly contemplates. Tkacik ¶80. Given the express teachings in Tkacik and the ubiquity of the SSL/TLS communication protocol, a POSITA would have been motivated to review the relevant SSL/TLS protocol for more details regarding how to implement the TLS Handshake Protocol for generating cryptographic keys based on cryptographic key exchange.

280. A POSITA would have combined Tkacik and TLS-1.2 to implement TLS-1.2’s TLS Handshake Protocol using Tkacik’s architecture to establish the TLS secure communication. This would have included using Tkacik’s secure processing section to generate keys needed for use in the SSL protocol. *See* Tkacik ¶79 (“FIG. 8 shows a flow chart of an exemplary make new key process 140 performed in secure processing system 12. Process 140 may be performed by controller 46 [within the secure processing section 30] in response to the execution of software programming instructions using components of secure processing section 30, and possibly other

components of host device 10.”); TLS-1.2, 6 (“TLS should be used to establish a secure connection between parties”), 33 (“The cryptographic parameters of the session state are produced by the TLS Handshake Protocol, which operates on top of the TLS record layer. When a TLS client and server first start communicating, they agree on a protocol version, select cryptographic algorithms, optionally authenticate each other, and use public-key encryption techniques to generate shared secrets.”).

281. A POSITA would have been motivated to combine Tkacik with TLS-1.2 for the reasons TLS-1.2 explains: to follow an “Internet standard[] track protocol for the Internet community” that “allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.” TLS-1.2, 1. Indeed, Tkacik contemplates a host device that may include a number of Internet-connected devices, including a “server, point-of-sale terminal, wireline or wireless telephony, radio, personal computer, laptop, handheld computer, workstation, digital media player, router, modem, industrial controller, and the like.” Tkacik ¶21. 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, such as malware running on the host device, that may try to steal these keys for malicious purposes.

282. A POSITA implementing TLS-1.2 on Tkacik's hardware would have understood the combination would have been nothing more than combining known prior art elements (e.g., Tkacik's security engine in a system-on-a-chip apparatus with TLS-1.2's secure handshake process) according to known methods (e.g., performing the well-known TLS handshake process) to yield predictable results (e.g., establishing secure communication between a client and a server). A POSITA would have had a reasonable expectation of success in the combination because TLS-1.2 was a standard protocol for communicating over the Internet, it was well-known to those in the art, and Tkacik expressly contemplates generating keys for use in SSL communications. Tkacik ¶80.

283. The combination also would have been applying a known technique of communicating over the Internet using TLS-1.2's ubiquitous handshake process to generate encryption keys for secure communication between a client and a server, to a known device in Tkacik's flexible security platform in a system-on-a-chip apparatus. A POSITA would have understood that this would have improved Tkacik by enabling secure communication from host device 10 by performing the TLS handshake according to the latest version of the well-known SSL/TLS protocol.

284. Therefore, a POSITA would have found it obvious to implement the TLS 1.2 handshake protocol disclosed in TLS-1.2 using Tkacik's system-on-a-chip apparatus.

2. Motivation to Combine Tkacik and TLS-1.2 with Bugiel

285. In the Tkacik-Bugiel or Tkacik-Bugiel-TLS-1.2 combinations, a POSITA would have found it obvious to store the encrypted session key generated in Tkacik in the secure memory as described in Bugiel. Bugiel discloses storing a session-token ST_C (e.g., “the encrypted session key”) in secure memory: “the SSL/TLS session key S_C is generated . . . and sealed (together with pre-master secret PMS and the hash value $\text{hash}(\text{Cert}_s)$). The 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 (emphases added), Fig. 2.

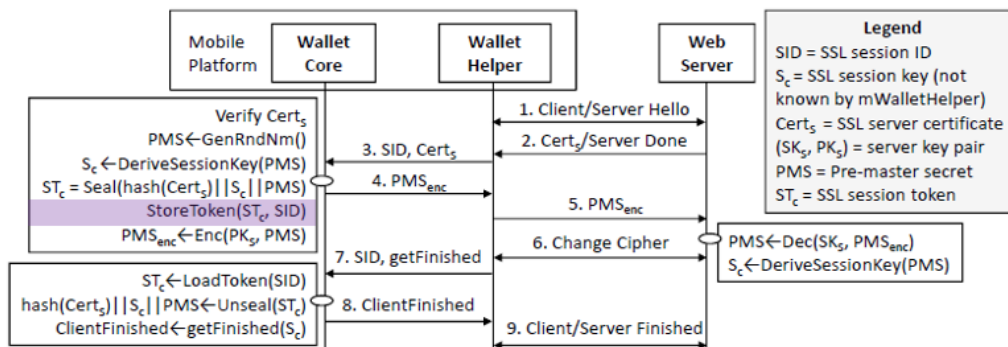


Fig. 2. SSL/TLS handshake

Id., Fig. 2 (annotated).

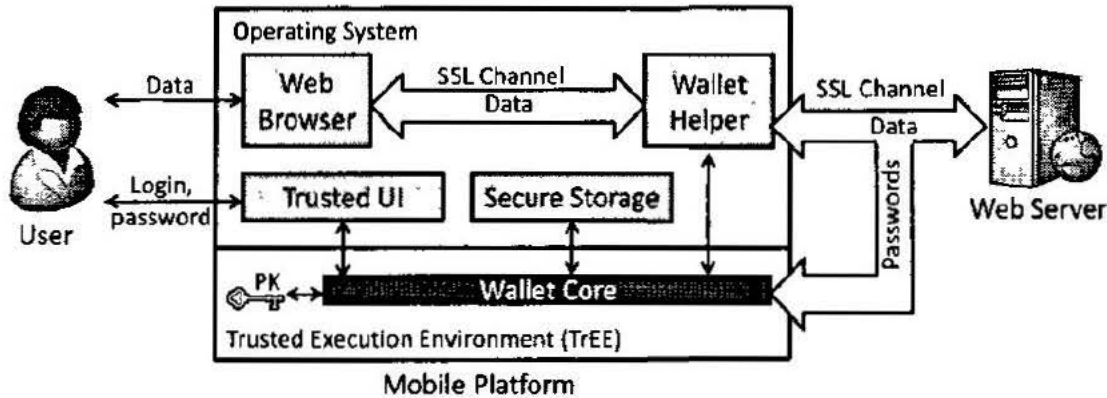


Fig. 1. TruWalletM Architecture

Id., Fig. 1.

286. As shown above, Figure 1 of Bugiel illustrates SecureStorage that is accessible only by the WalletCore in the TrEE. *See also id.*, 224 (“SecureStorage is a database where passwords and other security sensitive data are stored persistently and securely.”). Bugiel further discloses elements “stored within the secure storage . . . are accessible only by the TrustUI component,” and that “[i]deally, Trusted UI should reside within the TrEE.” *Id.*, 224. While Bugiel depicts Trusted UI and SecureStorage outside of the TrEE in Figure 1, Bugiel explains that limited “resources of contemporary trusted execution environments e.g., 20 kB of secure RAM for M-Shield, prohibit such an implementation at the present time. *See id.* Bugiel discloses that “future generations of the TrEE may provide the required capabilities to implement a fully trusted path between the user and the TrEE, because hardware vendors continue to extend their TrEE’s capabilities.” *Id.*, 224, Fig. 1; *see also id.*, 234-235 (“[R]ecognizing the trend of device manufacturers to further

extend their devices' TrEE capabilities, we argue that our proposed architecture is fundamentally a step into the right direction towards hardware secured user credentials on mobile end-user devices.”), 232-233 (“credentials associated with the SSL/TLS channel (such as a pre-master secret key and a session key) are protected by the TrEE”), 231 (“isolating the SecureStorage from other, unauthorized (and potentially malicious) script”).

287. Therefore, Bugiel discloses storing the encrypted session key in SecureStorage, which is ideally implemented within a hardware TrEE. In the Tkacik-Bugiel or Tkacik-Bugiel-TLS-1.2 combinations, a POSITA would have understood and found it obvious to implement Bugiel's teachings to store the encrypted session key generated in Tkacik ideally in one of Tkacik's secure memory, such as zeroizable read-write memory 52, key register 54, active key 72, data register 76, as taught and suggested by Bugiel. *See* [1a-2] in Grounds 1A-1C.

288. Further, Bugiel also teaches storing the encrypted session key in the secure memory (e.g., [1f]) by disclosing run-time isolation: “[a]ll operations on user credentials (except user input) are performed within the TrEE. As the TrEE is isolated from the rest of the system in terms of processing and memory, that guarantees protection of the user's credentials at run-time from potentially malicious OS and other OS-side components.” Bugiel, 232, 228 (describing WalletCore in the TrEE “loads the SSL/TLS session token ST_C from the secure storage and unseals

it”), Figs. 4-6. Therefore, Bugiel discloses storing the encrypted session key in the isolated memory within the secure execution environment as the session key is being encrypted/decrypted during run-time. A POSITA would have found it obvious to store Tkacik’s ciphertext key 372 (which could be the encrypted session key) in the secure memories within secure processing section 30 (e.g., zeroizable read-write memory 52, key register 54, active key 72, data register 76, etc.) of Tkacik, as the session key is being encrypted/decrypted during run-time as taught by Bugiel’s method. *See* Tkacik ¶¶37, 54, 73-74, 84.

289. Combining Tkacik-Bugiel or Tkacik-Bugiel-TLS-1.2 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 methods of storing keys in secure storage within a TrEE to Tkacik because both Bugiel and Tkacik disclose similar hardware for encrypting and storing keys. A POSITA would have been motivated to apply Bugiel to Tkacik to store keys in persistent storage for the reason Bugiel discloses: to protect user credentials and preventing “unseal[ing] by malware or an adversary performing physical attacks.” Bugiel, 232.

3. Claim 1

a. [1p] “A system-on-a-chip apparatus comprising:”

290. Tkacik-TLS-1.2 satisfies this feature. To the extent “system-on-a-chip apparatus” means the apparatus itself is a system-on-a-chip, Tkacik’s secure processing system 12 discloses this, as annotated below. Tkacik ¶¶20-25, Fig. 1.

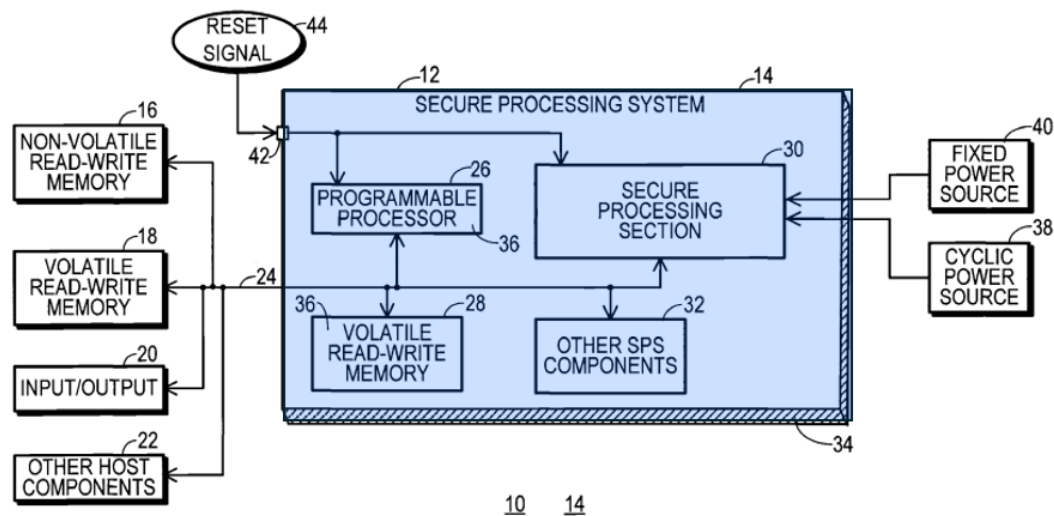


FIG. 1

Id., Fig. 1 (annotating secure processing system 12).

291. For example, Tkacik discloses that its secure processing system 12 includes “processor 26, memory 28, secure processing section 30, and other SPS components 32,” which are “[d]esirably ... all formed together on a *common semiconductor substrate* and are packaged together and reside within a *single integrated circuit 34*.” *Id.* ¶25 (emphases added). Tkacik also refers to volatile read-write memory 28 of secure processing system 12 as “on-chip memory 28,” as

opposed to “off-chip memories 16 and 18.” *Id.* ¶26. A POSITA would have understood from this that secure processing system 12 is a “system-on-a-chip apparatus.”

292. Tkacik’s disclosure of secure processing system 12 is consistent with the description of a system-on-a-chip in the ’838 patent, where the patent discloses that “[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,” and “an SOC may include ... [multiple] components on a single chip.” EX-1001, 1:27-32.

293. To the extent “system-on-a-chip apparatus” requires an apparatus *to have* a system-on-a-chip (as opposed to the apparatus itself being the system-on-a-chip), Tkacik discloses this feature too. For example, Tkacik discloses host device 10 and/or encryption apparatus 14, which are described synonymously, whereby host device 10 and/or encryption apparatus 14 *includes* secure processing system 12. Tkacik ¶¶20-25; *see also id.* ¶20 (“[H]ost device 10 is an encryption apparatus 14 because it performs data encryption and/or decryption operations.”). Specifically, Tkacik discloses that host device 10 “utilize[s]” and “includes” secure processing system 12.” *Id.* ¶¶20-22. Because host device 10 is a device that includes system-on-a-chip 12, as shown and described with respect to annotated Figure 1

below, host device 10 and/or encryption apparatus 14 discloses a “system-on-a-chip apparatus.” *Id.* ¶¶20-25, Fig. 1.

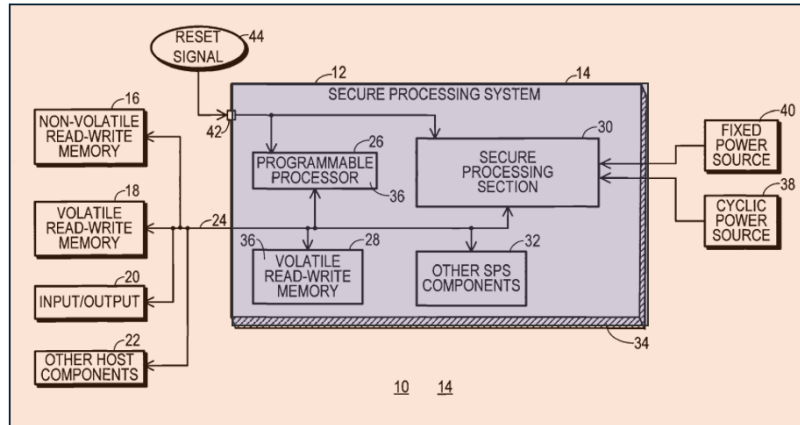


FIG. 1

Id., Fig. 1 (annotating host device 10, encryption apparatus 14, and secure processing system 12).

- b. [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”**

294. Tkacik-TLS-1.2 satisfies this feature. For example, Tkacik’s secure processing section 30 and programmable processor 26 disclose this feature. Tkacik ¶¶25-26, Fig. 1.

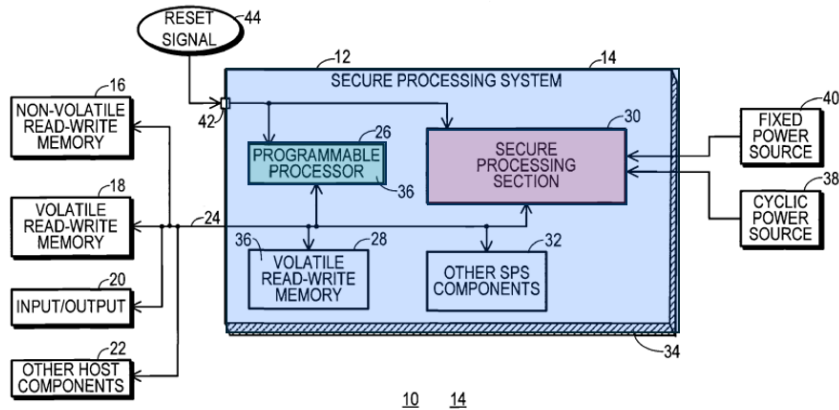


FIG. 1

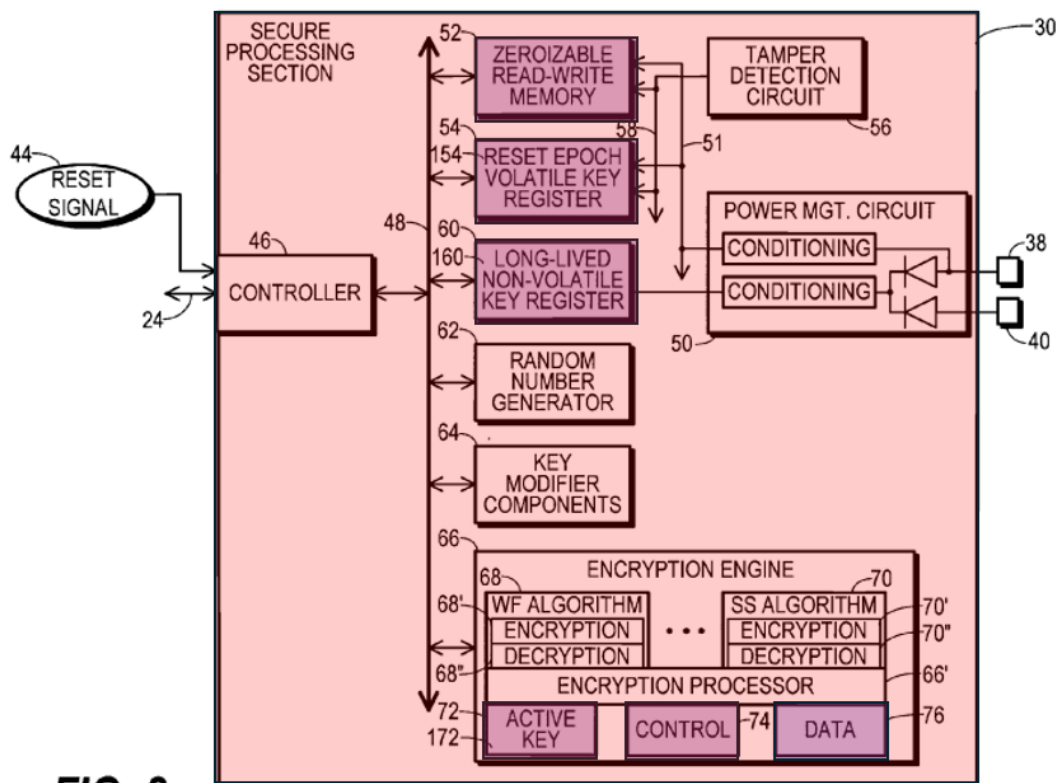
Id., Fig. 1 (annotated).

295. Tkacik’s secure processing section 30 discloses a “security engine,” consistent with the description of this feature in the ’838 patent. For example, the ’838 patent discloses that the security engine “maintains the keys in secrecy within a secure memory 114.” EX-1001, 3:35-41, 4:17-34.

296. 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. Tkacik’s 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, by contrast, “stores one or more plaintext master keys 160 [which] ha[ve] a life which

extends across power cycles for host device 10 and does not end at a power off event.” *Id.* ¶¶44-46.

297. 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 configured as separate or sections of the same volatile memory. *Id.* ¶38. Tkacik also explains that key register 54, active key register 72, control register 74, and data register 76 can be included as part of read-write memory 52. *Id.* ¶¶38, 54.



Id., Fig. 2 (annotated).

298. Tkacik's secure processing section 30 discloses the claimed "security engine." *See* Tkacik ¶26 ("Secure processing section 30 provides a secure execution environment (SEE) for secure processing system 12 and host device 10. . . . memory located within secure processing section 30 (discussed below) is secure memory. In large part, secure processing system 12 provides security services through the operation of secure processing section 30, including cryptographic operations ...")."

299. Tkacik also discloses a separate "processor core" on the system-on-a-chip, as claimed. For example, Tkacik discloses that secure processing system 12 includes "programmable processor 26, which may be viewed as a central processing unit (CPU), processor, controller, microcontroller, microprocessor, or the like." *Id.* ¶¶23-24. Programmable processor 26 controls secure processing system 12 by programmable code 36. *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.

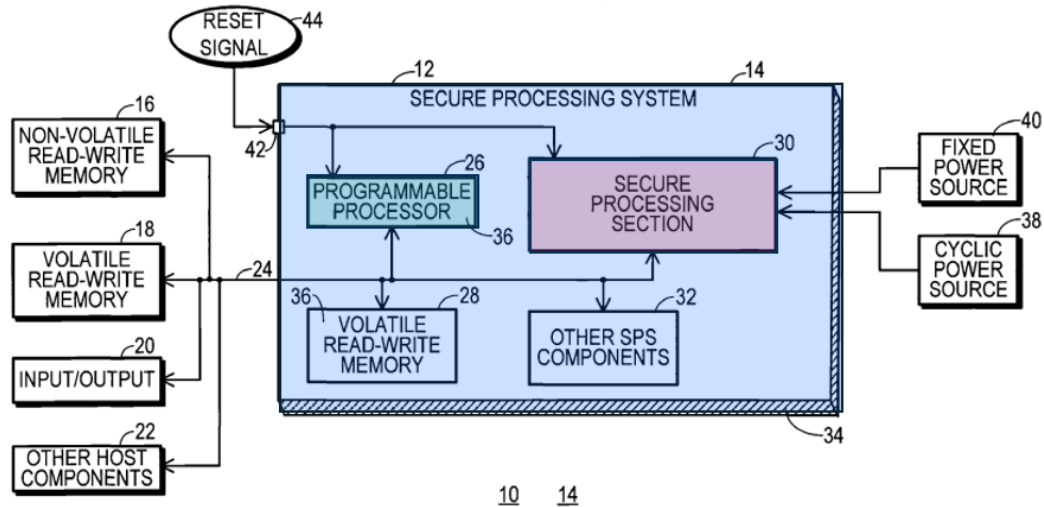


FIG. 1

Id., Fig. 1 (annotated).

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

300. Tkacik-TLS-1.2 satisfies this feature. As discussed in [1a-1], Tkacik discloses secure memory accessible only by the security engine in secure processing section 30. As depicted and described with reference to Figure 2, Tkacik discloses the following 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 discloses that 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,” as claimed, because Tkacik discloses that “memory

located within secure processing section 30 ... is secured memory.” Tkacik ¶26; *see also id.* ¶37 (“[z]eroizable read-write memory 52 provides secure memory”).

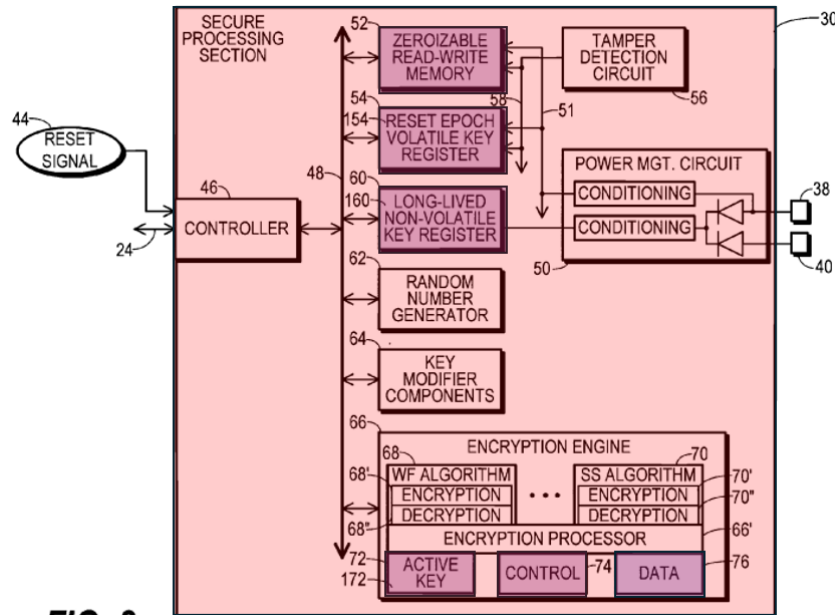


FIG. 2

Id., Fig. 2 (annotated).

301. A POSITA would have understood that each of 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 discloses memory “accessible only by the security engine,” as claimed, because these are within secure processing section 30 that “provides a secure execution environment (SEE).” *Id.* ¶26; *see also id.* ¶5 (“The SEE, also called a trusted platform, security zone, and the like, may include software and/or hardware features which promote a high level of trust in the cryptographic operations the device may undertake, making the security features of the device nearly immune to malicious software that the device may accidentally or

intentionally execute from time to time.”). Tkacik also discloses with respect to zeroizable read-write memory 52 that “[i]ts contents are protected against the execution of malicious software within host device 10,” and that its storage capacity may be intentionally limited to be inadequate for maintaining keys “in addition to other data storage and buffering tasks which may be asked of memory 52 in connection with moving blocks of payload data *into and out* from secure processing section 30 in support of cryptographic operations.” *Id.* ¶37 (emphasis added).

302. By protecting 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 in a “secure processing section 30” that “provides a secure execution environment (SEE),” including embodiments in which memory capacity is intentionally limited to prevent moving data in and out of secure processing section 30, these memories disclose secure memory accessible only by the security engine, as claimed.

d. [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,”

303. Tkacik-TLS-1.2 satisfies this feature. For example, Tkacik discloses that a 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 for host device 10 and does not end at a power off event.” *Id.* ¶44.

304. A POSITA would have understood plaintext master key 16 is a “security key” because it is a key that is used for cryptographic operations, including a SS algorithm 70, for “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.* ¶¶49-50, 57, 69-70, 74, 83-84.

305. Tkacik discloses that plaintext master key 160 is “encoded in the secure memory during a manufacturing process of the system-on-a-chip,” as claimed, by disclosing that “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.* ¶46. For example, Tkacik discloses that “laser-scribed fuses may be burned during the manufacture of integrated circuit 34 (FIG. 1) to permanently program plaintext master key 160 into long-lived key register,” or “electronic fuses may be electronically burned to program plaintext master key 160 and then blocked against further programming.” *Id.*

- e. **[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;”**

306. Tkacik-TLS-1.2 satisfies this feature. For example, Tkacik discloses the generating a random nonce by disclosing that “[s]ecure processing section 30

includes a random number generator 62.” Tkacik ¶47, Fig. 2. The random number generator 62 “generates random numbers when requested to do so by controller 46, and such random numbers may then be moved and stored in other components of secure processing section 30, or more generally host device 10, for subsequent use.” *Id.* For example, the random numbers of random number generator 62 are used to “generate cryptographic keys.” *Id.* ¶48; *see also id.* ¶¶59, 79-80, claims 1, 9.

307. Tkacik’s random number generator 62 is part of Tkacik’s secure processing section 30 (e.g., Tkacik’s “security engine”). *Id.* ¶¶47-48, Fig. 2. A POSITA would have understood the random number generated by the random number generator 62 for SSL protocol is a “random nonce,” as claimed. *See* (Grounds 1A-1C, [1b]). Indeed, the ’838 patent confirms this by disclosing that “security engine 110 generates a random nonce value” and “may use any suitable **random number-generating algorithm** to generate the random nonce.” EX-1001, 10:14-27 (emphasis added); *see also id.*, 6:40-50 (“security engine 110 may use any suitable random number-generating algorithm”).

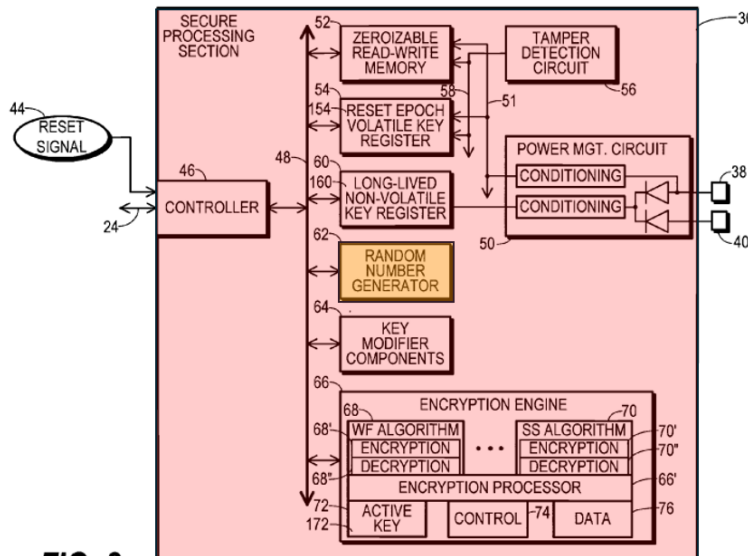


FIG. 2

Id., Fig. 2 (annotated).

308. Indeed, before the filing of the '838 patent, it was common knowledge that ClientHello.random and ServerHello.random were random nonces exchanged between client and remote server to avoid “replay” attack in SSL/TLS protocols. *See* TLS-1.2, 91-92; *see* Section VIII.A.3.e ([1b], **Grounds 1A-1C**) (citing prior art references). Tkacik’s random number therefore discloses generating a “random nonce,” as claimed and contemplated by the TLS-1.2 specification.

309. Tkacik discloses that its host device 10 “represents an electronic device” that may include “[a] server, point-of-sale terminal, wireline or wireless telephony, radio, personal computer, laptop, handheld computer, workstation, digital media player, router, modem, industrial controller, and the like.” Tkacik ¶21. Tkacik further discloses that the host device “includes provisions for maintaining the security of data that may be stored in, processed by, and/or *communicated by* host

device 10.” *Id.* (emphasis added). Tkacik further contemplates improvements where “a vast number of communication sessions or other data object domains for a vast number of clients” are involved. *See id.* ¶8. Therefore, while Tkacik may not expressly disclose that its security engine generates a random nonce “for initiating a request for a secure communication session with a remote server over a network using the nonce,” a POSITA would have understood that Tkacik at least suggests doing so.

310. Indeed, Tkacik contemplates generating keys “using any number of protocols known to those skilled in the art,” including SSL. *Id.* ¶80. A POSITA would have understood that SSL keys are utilized for secure communication between different devices, including client host devices and servers. *See* EX-1006, 2-6 (describing the use of SSL for clients to communicate with web servers), 37-38 (“The client is the system that initiates the secure communications; the server responds to the client’s request.”). The patent discloses this as well. *See* EX-1001, 1:10-24 (“One commonly used cryptographic communication protocol is the Secure Sockets Layer (SSL) protocol. The SSL protocol, and the related Transport Layer Security (TLS) protocol, is used for many different types of secure communication sessions including, for example, secure web-browsing, ecommerce, secure upgrades, and other secure communication sessions between two computing devices on an untrusted network such as the Internet.”).

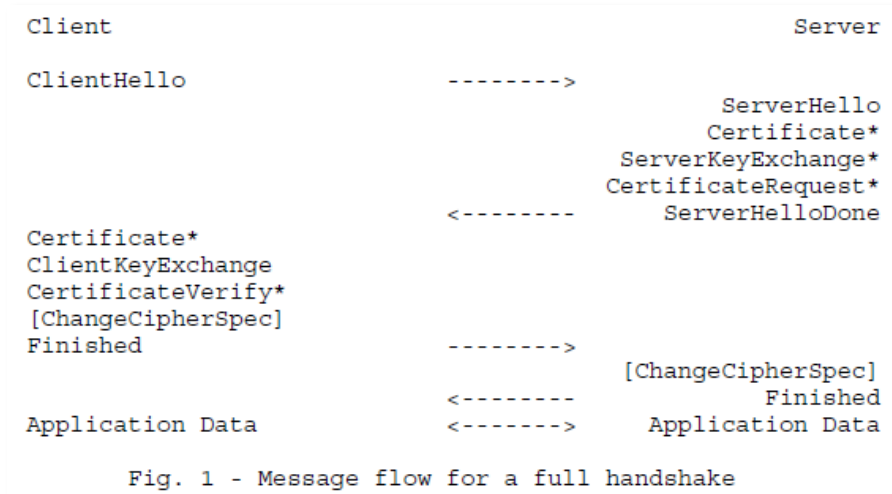
311. A POSITA would have found it obvious to combine Tkacik with TLS-1.2 to initiate a request for a secure communication session with a remote server over a network using a generated nonce. The combination would have incorporated Tkacik's architecture to implement, for example, TLS-1.2's secure communication specification. The combination would have done so by utilizing the secure processing section 30 to Tkacik (e.g., its "security engine") to generate keys based on the SSL protocol. *See* Tkacik ¶79 ("[A]n exemplary make new key process 140 performed in secure processing system 12 may be performed by controller 46 [within the secure processing section 30] in response to the execution of software programming instructions using components of secure processing section 30.").

312. TLS-1.2 explains that "TLS should be used to establish a secure connection between two parties." TLS-1.2, 6. TLS-1.2 discloses that the first step of the TLS Handshake Protocol is for the client to generate ClientHello.random (e.g., the claimed "generate a random nonce") and send a client hello message including ClientHello.random to the server. *Id.* at 34 ("[T]wo random values are generated and exchanged: ClientHello.random and ServerHello.random"); *id.*, 36-37 (Figures 1-2 illustrating initiating a TLS handshake process by sending a ClientHello message); *id.*, 39 ("When a client first connects to a server, it is required to send the ClientHello as its first message. . . The ClientHello message includes a random structure."). A POSITA would have understood that the ClientHello.random corresponds to

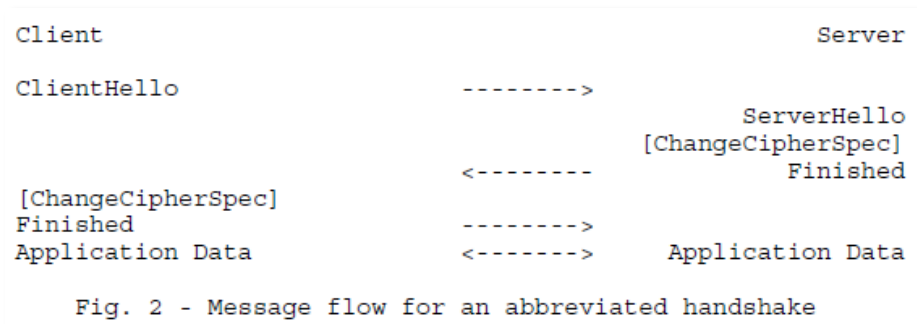
Tkacik's random number. TLS-1.2, 17 (defining "client random" as "[a] 32-byte value provided by the client"); *id.*, 33-34 ("The TLS Handshake Protocol involves the . . . [e]xchange hello messages to agree on algorithms, exchange **random values**, and check for session resumption.") (emphasis added); *id.*, 34 ("The ClientHello and ServerHello establish the following attributes: Protocol Version, Session ID, Cipher Suite, and Compression Method. Additionally, **two random values** are generated and exchanged: ClientHello.random and ServerHello.random.") (emphasis added); *see also* EX-1006, 41-43 ("ClientHello Components" include a "RandomNumber" field that includes a 32-byte random number used to seed the cryptographic calculations); *id.*, 42 (using a "pseudorandom number generation to create random numbers").

313. In initiating a secure communication using TLS-1.2, the Tkacik-TLS-1.2 combination would have generated a ClientHello.random and sent a ClientHello message to a server, including ClientHello.random, to initiate the TLS secure connection, according to the TLS-1.2 handshake. *See, e.g.*, TLS-1.2, 3-4 ("[T]he TLS Handshake Protocol[] allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. The TLS Handshake Protocol provides connection security . . ."); *id.*, 36-37 ("The TLS Handshake Protocol involves the following steps: Exchange hello messages to agree on algorithms, exchange random values, and check for session resumption."), 39-42

(“When a client first connects to a server it is required to send the ClientHello as its first message.”). TLS-1.2 depicts this in Figures 1 and 2, reproduced below.



TLS-1.2, Fig. 1.



Id., Fig. 2; *see also* EX-1006, 40-41 (Table 3-2 and Figure 3-1), 53-54 (Table 3-8 and Figure 3-5).

314. In the Tkacik-TLS-1.2 combination, Tkacik’s random number generator 62 in the secure processing section 30 would have been used to generate the ClientHello.random nonce, and to use that nonce to initiate communication with

a server over a network using a ClientHello message as part of the TLS handshake, as taught by TLS-1.2.

f. [1c] “perform a cryptographic key exchange with the remote server;”

315. Tkacik-TLS-1.2 satisfies this feature.

316. Tkacik teaches that components of the secure processing section perform a make-new-key process 140. *See* Tkacik ¶79. Tkacik also teaches generating a plaintext active key 172 based on random number generated using any one of many protocols known to a POSITA, including “IKE, DH, RSA key gen., SSL, random key, and the like.” Tkacik ¶80. As is known in the art and disclosed by TLS-1.2, the SSL/TLS handshake includes a client device and server exchanging Hello messages, followed by the client and the server exchanging messages based on RSA or Diffie-Hellman key exchange. TLS-1.2, 26-65 (discussing TLS handshake protocol, including, e.g., exchange hello messages, server/client certificates, server/client key exchange messages based on RSA or Diffie-Hellman key exchange, etc.); *see* Section VIII.A.3.f (**Grounds 1A-1C**, [1c]).

317. The combination of Tkacik and TLS-1.2 alleged here implements the TLS 1.2 protocol’s cryptographic key exchange with a remote server using Tkacik’s flexible security hardware, which includes secure processing section 30 (e.g., the “security engine”), to generate keys according to “any of a number of protocol known to those skilled in the art, including IKE, DH, RSA key gen., SSL, random

key, and the like.” Tkacik ¶¶79-81. A POSITA would have known this would have included the TLS 1.2 protocol, which includes performing handshake process with either RSA or Diffie-Hellman key exchange. *See* TLS-1.2, 26-65 (TLS Handshake Protocol), 80, 90.

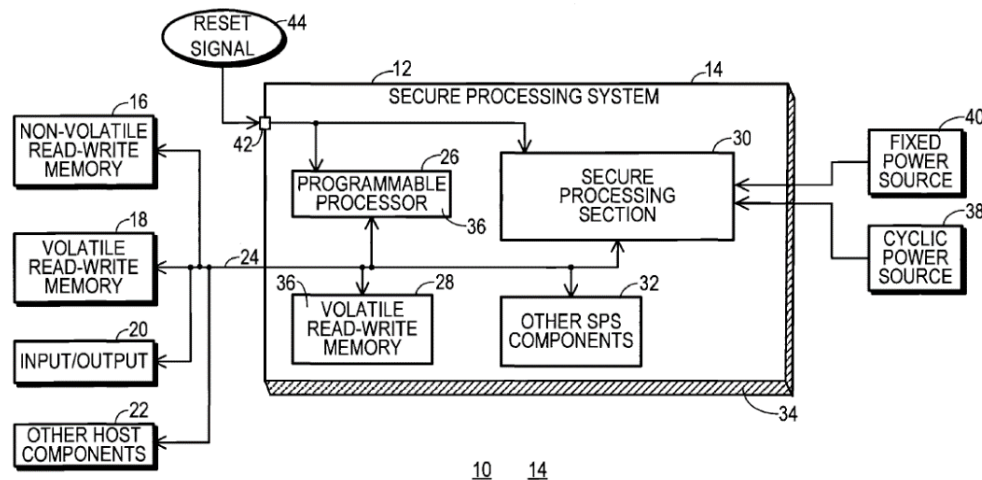


FIG. 1

Id., Fig. 1.

318. The Tkacik-TLS-1.2 combination would have performed cryptographic key exchange with a server based on TLS-1.2 handshake (including e.g., RSA or Diffie-Hellman key exchange, etc.) using Tkacik’s secure processing section. *See* TLS-1.2, 33-37 (Handshake Protocol overview), 37-64 (Handshake Protocol), 64-65 (Computing the Master Secret), Figs. 1-2.

319. Indeed, TLS-1.2 was a well-known protocol for secure communications between clients and servers, well before the ’838 patent, and it would have been

obvious to implement Tkacik in this manner to perform the TLS handshake. *See, e.g.,* Tkacik ¶80 (disclosing that “keys may be generated using any of a number of protocols known to those skilled in the art, including IKE, DH, RSA key gen, SSL, random key, and the like”); EX-1006, 38-51, 52-59; TLS-1.2, 26-65.

320. Tkacik-TLS-1.2 therefore satisfy performing a cryptographic key exchange with the remote server because both Tkacik’s host device and a remote server exchange keys in a TLS handshake.

- g. [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;”**

321. Tkacik-TLS-1.2 satisfies this feature. For example, Tkacik discloses that components of its secure processing section 30 (e.g., the “security engine”) perform a make-new-key process 140 to generate new cryptographic keys. *See* Tkacik ¶79, Fig. 8. One such key Tkacik generates is a plaintext active key 172. *Id.* ¶¶54-56, 80-82, 87, Fig. 2. Tkacik’s plaintext active key 172 discloses the claimed “symmetric session key.”

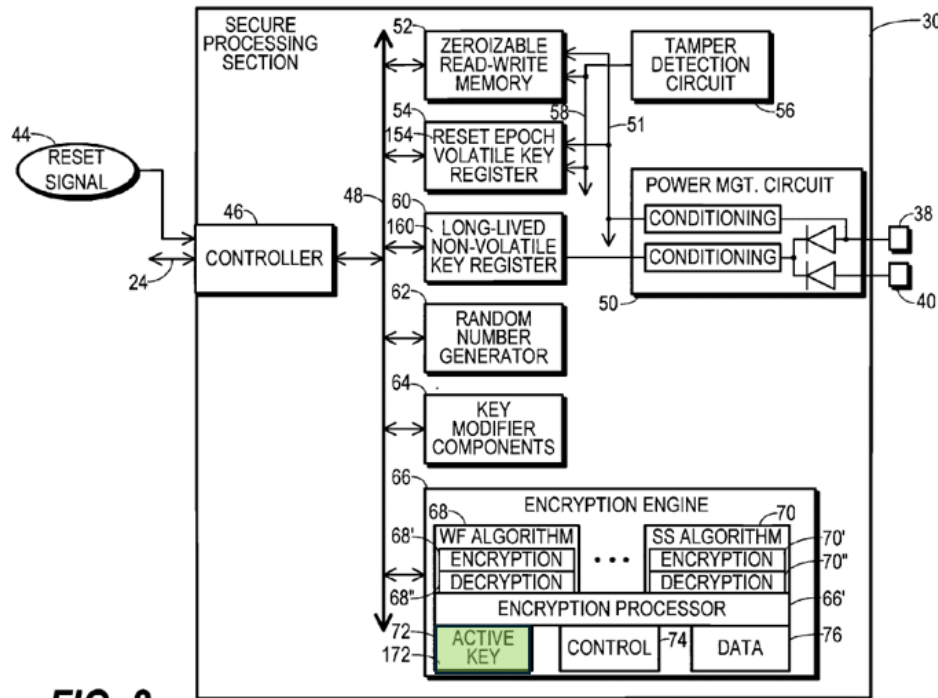


FIG. 2

Id., Fig. 2 (annotating plain text master key 172 in active key register 72).

322. Tkacik discloses that plain text master key 172 may perform encryption and decryption in the encryption engine 66. *Id.* ¶54 (“[E]ncryption processor 66’ performs algorithms 68 and 70 using a plaintext active key 172 stored in an active key register 72, control data stored in a control register 74, and payload data stored in a data register 76.”), Fig. 2. For example, Plaintext active key 172, once generated, is used “[i]n accordance with symmetrical cryptographic principles” to encrypt and decrypt ciphertext data at a later time. *Id.* ¶56 (“In accordance with *symmetrical cryptographic principles*, the same plaintext active key 172 that was used to encrypt plaintext data into ciphertext data is used at a later time to successfully decrypt the ciphertext data back into the plaintext data. Otherwise, a different plaintext active

key 172 will cause the ciphertext data to decrypt unsuccessfully.”) (emphasis added).

Specifically, Tkacik’s encryption engine 66 “provides encryption logic circuitry in the form of an encryption processor 66’ capable of implementing 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 (emphasis added).

323. 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.*

324. A POSITA would have understood that Tkacik’s plaintext active key 172 could be a “symmetric session key,” as claimed, because Tkacik discloses expressly that plaintext active key 172 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 methods, including Diffie-Helman, RSA and SSL. *Id.* ¶¶79-80, Fig. 8. This corresponds with a POSITA’s understanding of a symmetric session key used in TLS/SSL authentication and the ’838 patent.

325. For example, the ’838 patent discloses with respect to Figure 3 that, 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. The patent further discloses the same processes of generating the session key as Tkacik: “[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 Tkacik, ¶80 (“[I]n one embodiment the random number generated in task 142 is combined with key modifier components from section 64 in a task 144, and possibly arithmetically manipulated further by hashing or the like, to form a suitable plaintext active key 172.”); *see also* TLS-1.2, 65 (“When RSA is used for server authentication and key exchange, a 48-byte pre_master_secret is generated by the client, encrypted under the server’s public key, and sent to the server. The server uses its private key to decrypt the pre_master_secret. Both parties then convert the pre_master_secret into the master_secret.”).

326. A POSITA also would have understood that Tkacik’s plaintext active key 172 could be used to encrypt and decrypt messages, as claimed. *See* Tkacik ¶54 (perform encryption algorithms 68 and 70), ¶56 (encrypting and decrypting ciphertext data “at a later time”), ¶49 (performing 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.”).

327. To the extent Tkacik's plaintext active key 172 alone does not disclose the claimed symmetric session key, a POSITA would have found it obvious to implement Tkacik's plaintext active key 172 as the "master key" or "key_block" in the TLS-1.2 specification. TLS-1.2 generates a master secret based on the cryptographic key exchange in the TLS handshake process (i.e., the claimed "cryptographic key exchange"). *See* TLS-1.2, 64 (computing the Master Secret). TLS-1.2 also generates encryption keys (e.g., a "key_block") for data encryption, based on the master secret for each session. *Id.*, 25-26 (Key Calculation). 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 (defining the master_secret, client_write_key partition, and server_write_key partition). 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.

328. A POSITA would have found it obvious to implement Tkacik to generate the claimed symmetric session key via TLS-1.2's "master secret" or "key_block." The combination would have (1) generated an encryption key, either via the "master secret" or "key_block," based on cryptographic key exchange with a remote server according to the TLS handshake protocol; and (2) used the same session key (e.g., the "master secret" or "key_block") to encrypt messages sent to the remote server and decrypt messages received from the remote server during the secure communication session. Tkacik's the plaintext active key 172 would be implemented to be (1) the "master secret" disclosed in TLS-1.2 or (2) the "key_block" disclosed in TLS-1.2, as Tkacik discloses the plaintext active key 172 could be generated based on SSL protocol, and both "master secret" and "key_block" are generated by the SSL/TLS protocol as disclosed in TLS-1.2. *See* Tkacik ¶80; TLS-1.2, 25-26 (Key Calculation), 64 (Computing the Master Secret). The combination would have been nothing more than substituting one known element in the prior art (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.

329. Indeed, a POSITA would have had a high expectation of success because Tkacik contemplates using SSL encryption methods and a POSITA would have understood Tkacik's methods are compatible with the specification, including its master secret or key_block. As discussed, a POSITA would have found it obvious

to perform these tasks using Tkacik's secure processing section 30 (e.g., the "security engine") to generate keys based on SSL to encrypt/decrypt messages.

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

330. Tkacik-TLS-1.2 satisfies this feature. For example, Tkacik discloses using the master key 160 (e.g., the "security key") to encrypt the plaintext key 172 (e.g., the "symmetric session key") to form a long-lived ciphertext key 372 (e.g., the claimed "encrypted session key"). Tkacik ¶¶83-84, 46.

331. As explained, Tkacik discloses a plain master key 160 that is a "security key," by disclosing that "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.* ¶46; *see* [1a-3]. Plain master key 160 can "directly serve[] as the BKEK" if desired. *Id.* ¶83. Here, BKEK means "blob key encryption key." *Id.* ¶69. The BKEK "is a cryptographic key to be used in forming 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.

332. Tkacik further discloses generating the plaintext active key 172 (e.g., the "symmetric session key") using the secure processing section based on known protocols. *See* [1d]; Tkacik ¶¶79-80. Referring to Figure 8, Tkacik then describes moving the generated plaintext active key 172 (e.g., the "symmetric session key") "to data register 76 and encrypt[ing] this key 172 using . . . the BKEK" to form long-

lived alternate ciphertext key 372. *Id.* ¶84. The BKEK is the plaintext master key 160 (e.g., the “security key”).

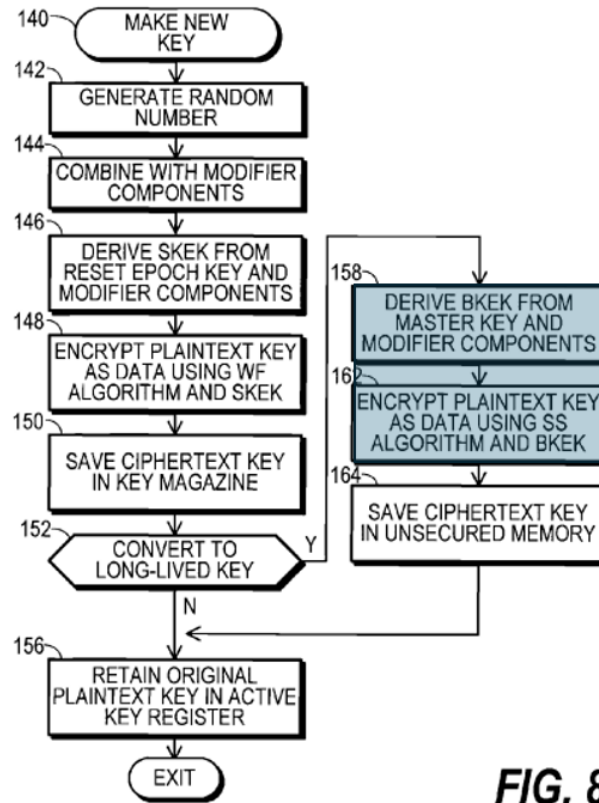


FIG. 8

Id., Fig. 8 (annotated).

333. Tkacik therefore discloses encrypting the symmetric session key based on the security key by disclosing that its secure processing section 30 (e.g., the “security engine”) encrypts the plaintext active key 172 (e.g., “symmetric session key”) into long-lived alternative ciphertext key 372 based on the plaintext master key 160 (e.g., BKEK, the claimed “security key.”).

334. In the Tkacik-TLS-1.2 combination, a POSITA would have found it 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. *See* TLS-1.2 at 25-26 (Key Calculation), 64 (Computing the Master Secret); *see* [1d].

i. [1f] “store the encrypted session key in the secure memory,”

335. Tkacik-TLS-1.2 satisfies this feature. For example, Tkacik discloses storing the alternative ciphertext key 372 (e.g., the encrypted session key) in the data register 76. Tkacik ¶84 (“After task 158, a task 162 moves the plaintext active key 172 generated above in task 144 to data register 76 and encrypts this key 172 using encryption process 70’ of SS algorithm 70 and the BKEK.”). Tkacik discloses that task 162 “forms a long-lived alternate ciphertext key 372.” *Id.*

336. A POSITA would have understood that data register 76 is part of the secure memory. Tkacik discloses that data register 76 is part of the encryption engine 66 within the secure processing section 30. *Id.* ¶¶54-55, 25-26. Secure processing section “provides a secure execution environment (SEE) for secure processing system.” *Id.* ¶¶25-26. Tkacik further discloses that data register 76 prevents intrusion by disclosing that its contents may be “destroyed upon a tamper event.” *Id.* ¶54. A POSITA would have further understood that, 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.

337. Alternatively, a POSITA would have understood 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 discloses that “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).” *Id.* ¶83. Tkacik explains that when the unencrypted session key 172 is needed, a task retrieves a blob 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* Tkacik ¶73 (“[P]rocess 118 may be invoked to recover a plaintext active key 172 which no longer exists within host device 10 but which has an *encrypted long-lived alternate ciphertext key 372 counterpart in a blob 110* stored in unsecure memory 16, 18, or 28”) (emphasis added). Tkacik explains that “task 122 retrieves the identified blob 110 from unsecure memory to encryption engine 66, *where it is saved in data register 76.*” *Id.* ¶74 (emphasis added). Tkacik further discloses that 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, a POSITA would have understood Tkacik’s security engine stores encrypted session key (e.g., long-lived alternate ciphertext key 372) in secure memory (e.g., data register 76), as Figure 2 illustrates below.

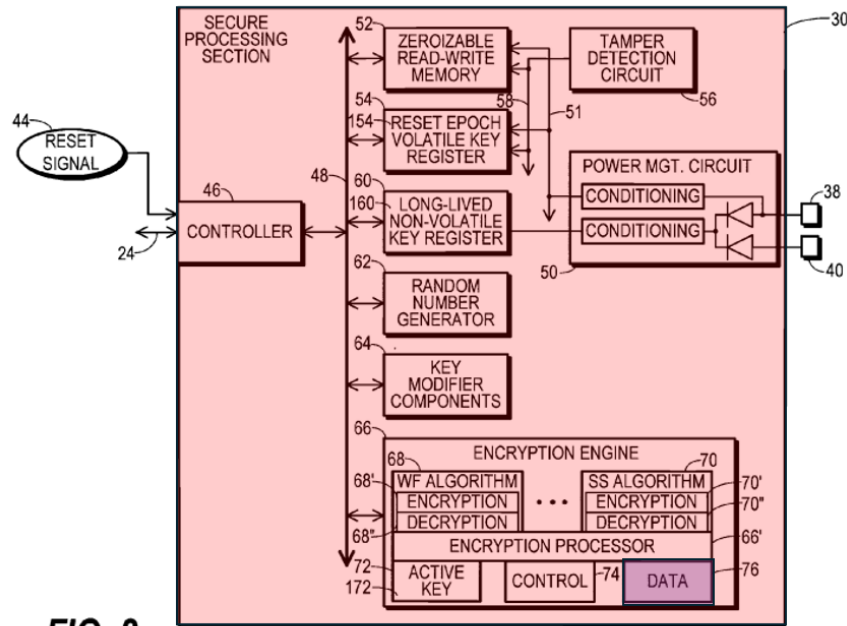


FIG. 2

Id., Fig. 2 (annotated).

338. As a second alternative, Tkacik also discloses storing the long-lived alternate ciphertext key 372 in secure memory by storing it in volatile memory. Tkacik discloses that data register 76 “may also be included as a part of zeroizable memory 52” and that zeroizable read-write memory 52 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, a POSITA would have understood that encrypted session key 372 is stored, at least temporarily, in a RAM of the secure processing section 372, for

example in the zeroizable read-write memory 52 or data register 76, which can itself be part of zeroizable read-write memory 52. *See id.* ¶154.

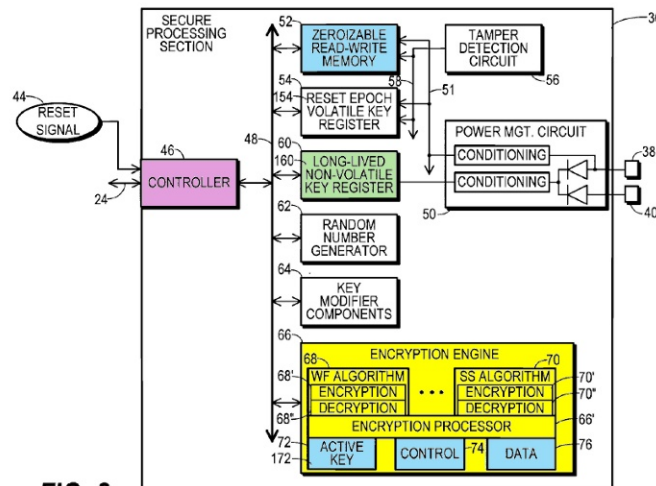


FIG. 2

Id., Fig. 2 (annotated).

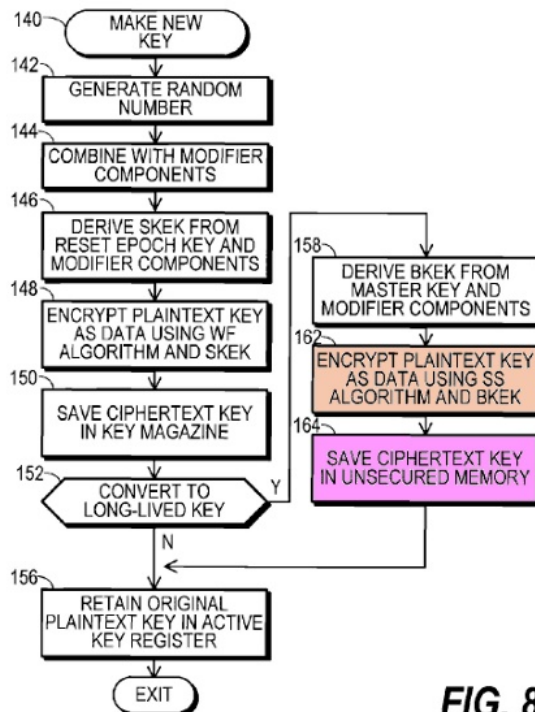


FIG. 8

Id., Fig. 8 (annotated).

339. This understanding corresponds with the prior art. For example, Tugenberg lists Tkacik as a co-inventor and describes the same basic hardware and operation as Tkacik. Tugenberg discloses a communication device 10 “having secret keys stored in non-secure external memory 12 in encrypted form, each secret key having been encrypted with laser-scribed encryption key 21 using secure processing system 14.” Tugenberg, 4:20-25. Tugenberg explains that sensitive information may be “encrypted with the laser-scribed encryption key by encryption logic circuitry and stored in a memory internal to the secure memory device, and preferably stored in a zeroizable RAM.” *Id.*, 6:50-55; *see also id.*, 2:64-66, 3:34-37. The encrypted information is then transferred to external memory 12: “host processor reads the encrypted sensitive information *from the zeroizable memory* and stores this information in a memory that is preferably external to the secure memory such as external memory 12 (FIG. 1).” *Id.*, 6:59-67 (emphasis added); *see also id.* at Abstract (“A user’s sensitive data is encrypted within the secure memory using the laser-scribed encryption key and stored as encrypted data in the non-secure memory.”), 5:4-5 (“After encryption, the encrypted sensitive information is stored in a non-secure memory.”), claim 12 (stating that “sensitive data is encrypted within the secure memory using the laser-scribed encryption key and stored as encrypted data in the non-secure memory”). Thus, although Tugenberg—like Tkacik—disclose “having secret keys stored in non-secure external memory 12 in encrypted form,”

Tugenberg, 4:20-25, Tugenberg also teaches that the encryption process itself involves storing the encrypted secret keys “in a memory internal to the secure memory device,” at least temporarily, before transferring it to external memory 12. *See id.* at 6:50-55, 4:20-25, 6:50-67, Figs. 1, 2.

340. A POSITA thus would have understood that Tkacik operates in the same manner using its zeroizable memory as a volatile store in secure processing section 30 before alternate ciphertext key 372 is stored outside of secure processing section 30. More specifically, a POSITA would have understood Tkacik’s zeroizable memory, like Tugenberg’s, performs “data storage and buffering tasks . . . in connection with moving blocks of payload data into and out from secure processing section 30 in support of cryptographic operations.” Tkacik ¶37; *see also id.* ¶84, Figs. 2, 8. This satisfies storing the encrypted session key in secure memory, as claimed.

341. Alternatively, 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. As discussed above, Bugiel teaches storing a session-token ST_C (the claimed “the encrypted session key”) in secure memory. For example, Bugiel explains “the SSL/TLS session key S_C is generated . . . and sealed (together with pre-master secret PMS and the hash value $\text{hash}(Certs)$). The resulting session-token ST_C is stored in a

secure storage and can be loaded and unsealed later using the session identifier SID.”

Bugiel at 225-226 (emphases added), Fig. 2.

342. As discussed, the symmetric session key in the Tkacik-TLS-1.2 or Tkacik-Bugiel-TLS-1.2 combinations may be implemented as be either (1) a “master secret” or (2) a “key_block,” according to TLS-1.2. *See* [1d]; TLS-1.2, 25-26 (Key Calculation), 64 (Computing the Master Secret). Accordingly, the encrypted session key 372, which is encrypted based on Tkacik’s master key 160, would be implemented for future SSL communications as either to (1) the encrypted “master secret,” or (2) the encrypted “key_block.” In the combination, the encrypted “master secret” or “key_block” is passed from secure processing section 30 to the unsecure world, as taught by Tkacik. Tkacik ¶84. The encrypted master secret or encrypted key_block is also retrieved from the data register 76. *See id.* ¶¶73-74 (storing long-lived alternate ciphertext key 372 in data register 76).

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

343. Tkacik-TLS-1.2 satisfies this feature.

344. For example, referring to Fig. 5, Tkacik explains that after making or retrieving cryptographic keys (e.g., when process 94, 118 or 140 completes), the secure processing section 30 of the secure processing systems proceeds to step 166 and performs “cryptographic operation . . . using the plaintext active key 172 as the

encryption or decryption key.” Tkacik ¶87. Here, the plaintext active key 172 is the claimed “session key.”

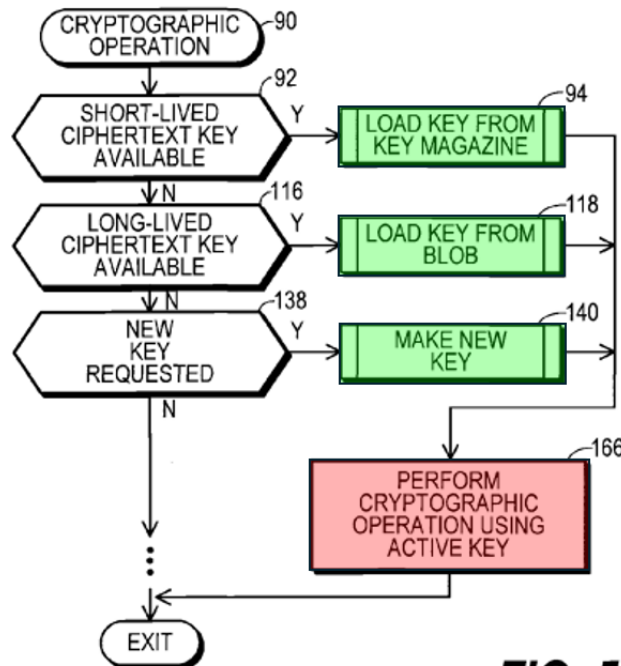


FIG. 5

Id., Fig. 5 (annotated).

345. As explained in [1b], the Tkacik-TLS-1.2 securely communicates with a remote server over a network. TLS-1.2 discloses the TLS handshake protocol that serves to establish secure communication between a client and server, “allow[ing] the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data.” TLS-1.2, 4; *see also id.*, 6 (“TLS should be used to establish a secure connection between parties.”). TLS-1.2 emphasizes that 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 (Handshake Protocol overview).

346. In the combined Tkacik-TLS-1.2 system, a POSITA would have implemented TLS-1.2's secure communication between a client with a remote server using the hardware of host device 10, including using the plaintext active key 172 of secure processing section 30. Indeed, Tkacik explains that 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]. A POSITA therefore would have understood that, in the combined system, by utilizing plaintext active key 172 of secure processing section 30 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, as claimed. Indeed, Tkacik discloses that its secure processing system 12 may be configured for a wide variety applications, including "wireless telephone, radio, personal computer, laptop, handheld computer," etc., to maintain the security of data that may be "communicated by host device 10." *Id.* ¶21. Tkacik therefore suggests employing its SOC encryption features to engage in communication with remote servers. *See also* Grounds 1A-1C, [1g] for additional disclosures of TLS-1.2.

4. **Claim 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.”**

347. Tkacik-TLS-1.2 satisfies this feature for the reasons in Section VIII.A.4 (Grounds 1A-1C, claim 2).

348. Tkacik-TLS-1.2 further satisfies the security engine receiving the server nonce and storing the server nonce in the secure memory. Tkacik-TLS-1.2 implements TLS-1.2’s protocol for the client to receive a ServerHello message (e.g., the “response message”) including a ServerHello.random (e.g., the “server nonce”) according to the TLS Handshake protocol using Tkacik’s security platform, in which secure processing section 30 (e.g., the “security engine”) generates keys based on the SSL protocol. *See* Section VIII.B.1.

349. Keeping the random nonce in secure memory is necessary to operate on the server nonce to calculate other shared secrets and keys securely. *See* TLS-1.2 at 25-26 (Key Calculation), 64 (Computing the Master Secret), 18 (describing server_random as part of the SecurityParameters), 96-97; Tkacik ¶26 (“Secure processing section 30 provides a secure execution environment (SEE) for secure processing system 12 and host device 10. Read-write memories 16, 18, and 28 are unsecure memories because they are all located outside of secure processing section 30.”).

350. Thus, to operate on the server nonce to calculate other shared secrets and keys securely, the server random nonce must be stored in a secure memory. *See* TLS-1.2, 25-26 (Key Calculation), 64 (Computing the Master Secret). Because Tkacik teaches generating keys using components of secure processing section 30, the server nonce also needs to be stored in the secure memory within the secure processing section for calculation. *See* Tkacik ¶¶79, 26. For example, Tkacik explains: “[z]eroizable read-write memory 52 provides secure memory for secure processing system 12 and host device 10 . . . due to the trust provided by secure processing section 30. Its contents are protected against the execution of malicious software within host device 10.” Tkacik ¶37.

5. Claim 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.”

351. Tkacik-TLS-1.2 satisfies this feature. A POSITA would have implemented Tkacik-TLS-1.2 to perform TLS 1.2’s TLS Handshake Protocol, including first generating a pre-master key based on Diffie-Hellman or RSA. TLS-1.2, 91; *see* Section VIII.A.5 (claim 3, **Grounds 1A-1C**).

352. More specifically, TLS-1.2 explains that one of the keys generated using SSL/TLS protocols is pre-master secret (the claimed “pre-master key”). *Id.* (disclosing that the goal of the key exchange is to generate a pre_master_secret,

which will be used to generate the master_secret to generate the encryption keys), 58-61 (“generat[ing] a 48-byte premaster secret, encrypts it using the public key from the server’s certificate, and sends the result in an encrypted premaster secret message.”); *see also id.*, 50 (ServerKeyExchange message that “conveys cryptographic information to allow the client to communicate the premaster secret: a Diffie-Hellman public key with which the client can complete a key exchange (with the result being the premaster secret) or a public key for some other algorithm.”), 57 (ClientKeyExchange message sent when “the premaster secret is set, either by direct transmission of the RSA-encrypted secret or by the transmission of Diffie-Hellman parameters that will allow each side to agree upon the same premaster secret.”), 58 (the Pre-Master key being a “random value is generated by the client and is used to generate the master secret.”), 91 (same).

353. In the TLS/SSL handshake, the client sends a ClientKeyExchange message. TLS-1.2, 57-61, 74; EX-1006, 40-41, 45-46. This message includes “session key information (encrypted with server’s public key).” TLS-1.2, 57-58 (“If RSA is being used for key agreement and authentication, the client generates a 48-byte premaster secret, encrypts it using the public key from the server’s certificate, and sends the result in an encrypted premaster secret message”); *see* EX-1006 40-41. The “encryption protects the key information as it traverses the network, and it allows the client to verify that the server truly possess the private key corresponding

to its public key. Otherwise, the server won't be able decrypt the message." *Id.*, 46. If RSA encryption is used, "the ClientKeyExchange carries a premaster secret." *Id.*, 85-87, Fig. 4-21; *see also id.*, 86 ("The message body itself consists solely of the encrypted premaster secret" using the public key of the server); TLS-1.2 58-59 (providing the structure of the ClientKeyExchange message), 74 (same).

354. A POSITA would have implemented Tkacik-TLS-1.2 such that, consistent with TLS-1.2, Tkacik's secure processing section 30 encrypts keys, such as the pre-master key, using the server public key as contemplated in TLS-1.2. For example, Tkacik discloses that the secure processing section 30 generates keys in accordance with the SSL protocol. *See* Tkacik ¶79 ("Process 140 [i.e., make new key process] may be performed by controller 46 in response to the execution of software programming instructions using components of secure processing section 30.") ¶80 ("keys may be generated using any of a number of protocols known to those skilled in the art, including . . . DH, RSA key gen., SSL, random key, and the like. In each of such embodiments, task 144 saves plaintext active key 172 for future use and moves the plaintext active key 172 to data register 76.").

355. TLS-1.2 similarly teaches generating the pre-master key, master key, and encryption keys during the handshake. *See* TLS-1.2, 91 (stating that the goal of the key exchange is to generate a pre_master_secret to generate the master_secret, which is required to generate the encryption keys). TLS-1.2 also explains that the

result of RSA key gen in SSL/TLS protocol is the pre-master secret. *See id.*, 57-58 (“If RSA is being used for key agreement and authentication, the client generates a 48-byte premaster secret”), 34, 50-51. Therefore, a POSITA would have implemented Tkacik-TLS-1.2 to generate a pre-master key via Tkacik’s plaintext active key 172, which is generated based by RSA in the context of TLS/SSL.

356. A POSITA would have implemented the Tkacik-TLS-1.2 combination to implement TLS-1.2’s encrypting the pre-master key with the server’s public key, as required by the TLS Handshake Protocol. TLS-1.2 explains that the client encrypts a premaster secret using the public key from the server and then sends the result in an encrypted premaster secret message to the server. *See* TLS-1.2, 58-61 (RSA encrypted premaster secret message), 74. In the combination, 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 (“[K]eys may be generated using . . . DH, SSL, RSA key gen [T]ask 144 saves plaintext active key 172 for future use.”). Indeed, a POSITA would have understood that, because Tkacik’s plaintext active key 172 may be any key used in an applicable cryptographic algorithm, TLS pre_master_secrets, master_secrets, and key_blocks are all examples of a plaintext active key 172. *See, e.g.*, Tkacik ¶55 (“The selected

algorithm uses plaintext active key 172 stored in active key register 72 to perform the selected cryptographic algorithm on the payload data stored in data register 76.”).

357. Alternatively, Tkacik teaches encrypting the plaintext active key 172, may include the pre-master key by using the master key 160 (the claimed “security key”). *See* Tkacik ¶¶83-84 (explaining “plaintext master key 160 more directly serves as the BKEK . . . a task 162 moves the plaintext active key 172 generated above in task 144 to data register 76 and encrypts this key 172 using encryption process 70’ of SS algorithm 70 and the BKEK”).

358. Because Tkacik teaches encrypting plaintext active key 172 using plaintext master key 160, the combination of Tkacik-TLS-1.2 includes encrypting the TLS-1.2 pre-master key using Tkacik’s plaintext master key 160. Thus, in the alleged combination, 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, which could be part of zeroizable read-write memory 52 (*see* Tkacik ¶54)) in an encrypted state (using the security key), like what the ’838 patent’s specification describes at 11:6-9. *See also* [1f].

359. In sum, the Tkacik-TLS-1.2 combination provides the same disclosures as the specification of the ’838 patent as it relates to claim 3.

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

360. The '838 patent uses the terms “encrypted” and “wrapped” synonymously. *See* EX-1001, 5:67-6:1 (“master key may be stored in an encrypted or wrapped form (e.g., encrypted using the security key 150).”), 6:6-9 (same), 8:29-32 (“encrypted or otherwise wrapped using the server public key provided by the server in server authentication and key exchange phase”). A POSITA also would have understand that wrapping a key is another way to refer to encrypting a key. The Tkacik-TLS-1.2 combination therefore satisfies the security engine wrapping the pre-master key using a server public key for the same reasons the pre-master key is encrypted with the server public key, as discussed in claim 3. *See* claim 3.

361. Tkacik-TLS-1.2 also “store[s]” the wrapped pre-master key in the secure memory. For example, the encrypted pre-master secret is data that must be stored (even if temporarily) to be processed. TLS-1.2 teaches this expressly by storing the pre-master key to compute a master secret using the pre-master key. *See* TLS-1.2, 64 (explaining the pre-master secret should be deleted from memory once the master secret has been computed). In the Tkacik-TLS-1.2 combination, a POSITA would have stored the encrypted (e.g., “wrapped”) pre-master key in Tkacik’s secure memory. *See* [1a-1].

362. For example, Tkacik discloses that it is conventional to “maintain the privacy is to save the keys in a secure memory device under the control of a SEE [secure execution environment].” Tkacik ¶7. A POSITA thus would have found it obvious to store the encrypted pre-master key in Tkacik’s secure memory (e.g., any memory within the secure processing section 30, including, e.g., zeroizable read-write memory 52, active key register 72, data register 76, etc.) when calculating the encrypted pre-master key, when including the encrypted pre-master key in the encrypted premaster key message, and when the encrypted pre-master key is sent to the server during the TLS handshake protocol. *See* Tkacik ¶¶7, 26; *see* [1f].

363. Tkacik also teaches or suggests storing wrapped keys in the secure memory when the encrypted keys are transferred from secure memory to outside the secure memory, and vice versa. For example, Tkacik teaches using secure memory 52 within the security engine 30 for data storage and buffering tasks “in connection with moving blocks of payload data into and out from secure processing section 30 in support of cryptographic operations.” Tkacik ¶37; *see also id.* ¶84 (“Task 162 forms a long-lived alternate ciphertext key 372. Then a task 164 stores long-lived alternate ciphertext key 372 in unsecure memory 16, 18, and/or 28 (FIG. 1) as part of a blob 110.”), ¶74 (“retrieves the identified blob 110 [which includes encrypted keys] from unsecure memory to encryption engine 66, where it is saved in data register 76”).

7. Claim 5

- a. [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”**

364. Tkacik-TLS-1.2 satisfies this feature. *See* [1d]. As discussed, Tkacik discloses that the secure processing section 30 generates plaintext active key 172. *See* Tkacik ¶¶79-80 (explaining the secure processing section is invoked to generate new plaintext active key 172 based on e.g., SSL protocol), ¶56 (“In accordance with symmetrical cryptographic principles, the same plaintext active key 172 that was used to encrypt plaintext data into ciphertext data is used at a later time to successfully decrypt the ciphertext data back into the plaintext data.”).

365. The Tkacik-TLS-1.2 combination, as discussed in [1d], would have used Tkacik’s secure processing section 30 to implement TLS version 1.2, including TLS-1.2’s generating the session key as a function of the pre-master key according to the TLS Handshake protocol. *See* [1d] (discussing the “master key” or “key_block” in the TLS-1.2 specification).

- b. [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.”**

366. Tkacik-TLS-1.2 satisfies this feature. For example, Tkacik explains that encryption engine 66 within secure processing section “may also perform other cryptographic algorithms, such as hashing algorithms.” Tkacik ¶52, Fig. 2. A POSITA would have understood that, in the Tkacik-TLS-1.2 combination during the

TLS Handshake, it is necessary to calculate a hash function as a function of the random nonce generated by the security engine and a nonce received from the remote server. Indeed, TLS-1.2 discloses this feature in several ways.

367. First, TLS-1.2 teaches calculating the master secret by calculating a hash function as a function of the random nonce generated by the client's security engine and a nonce received from the remote server. TLS-1.2 states: "master_secret = PRF (pre_master_secret, "master secret", ClientHello.random + ServerHello.random)." TLS-1.2, 64 (Computing the Master Secret). TLS-1.2 states: "TLS's PRF is created by applying *P-hash to the secret* as: PRF(secret, label, seed) = P_<hash>(S1, label + seed)." *Id.*, 15 (HMAC and the Pseudorandom Function) (emphasis added). As these disclosures demonstrate, this involves calculating 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 a random nonce (ServerHello.random) received from the remote server (claim 2).

368. Second, TLS-1.2 discloses calculating the encryption key_block by calculating a hash function as a function of the random nonce generated by the client's security engine and a random nonce received from the remote server. TLS-1.2 states: "key_block = PRF(SecurityParameters.master_secret, 'key expansion', SecurityParameters.server_random + SecurityParameters.client_random)." *Id.*, 26; *see also id.*, 94 ("The master secret is hashed with the ClientHello.random and

ServerHello.random to produce unique data encryption keys . . .”). Just like the master secret, creating encryption key_block therefore involves calculating a hash function (the PRF function) as a function of the random nonce (ClientHello.random) generated by the client’s security engine (*see* [1b]) and a random nonce (ServerHello.random) received from the remote server (claim 2).

369. Third, TLS-1.2 teaches generating a Finished message by calculating a hash function as a function of the random nonce generated by the client’s security engine and a random nonce received from the remote server, where the Finished message includes a hash of all the handshake messages. For instance, TLS version 1.2 calculates the following: “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 also involves calculating a hash function (the PRF function) as a function of the random nonce (ClientHello.random) generated by the client’s security engine (*see* [1b]) and a random nonce (ServerHello.random) received from the remote server (claim 2).

370. Fourth, TLS-1.2 explains that the client generates a Certificate Verify message after a client key exchange message, whereby the Certificate Verify

message “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). After receiving the ServerHello message, the client thus generates a Certificate Verify message that “compute[s] running hashes for all potential hash algorithms” on the 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 involves calculating a hash function (the hash algorithm) as a function of the random nonce (ClientHello.random) generated by the security engine (*see* [1b]) and a random nonce (ServerHello.random) received from the remote server (claim 2).

8. Claim 6

- a. **[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,”**

371. Tkacik-TLS-1.2 satisfies this feature.

372. As contemplated by the TLS-1.2 protocol, a POSITA would have understood that the symmetric key or master secret can be generated by either (RSA) key exchange or a Diffie-Hellman key exchange. *See* TLS-1.2, 55-57 (client certificate including RSA or Diffie-Hellman parameters), 57-58 (client key

exchange message based on RSA or Diffie-Hellman), 61, 65, 74-75, 86, 92-93, 100; EX-1006, 81-84 (describing different ServerKeyExchange message formats for Diffie-Hellman, RSA, and Fortezza key exchange protocols), 40-45 (ServerKeyExchange message), 54-59 (separate ServerKeyExchange message and server certificate); *see* [1c] (describing ServerKeyExchange message with or with certificate).

373. To implement a Diffie-Hellman key exchange, the Tkacik-TLS-1.2 combination would have implemented the Diffie-Hellman key exchange teachings of TLS-1.2 on Tkacik's hardware such that the master secret (e.g., the symmetric session key) would be generated based on a ClientKeyExchange message that includes a Diffie-Hellman client key. TLS-1.2, 57-61 ("With this [ClientKeyExchange] message, the premaster secret is set, either by direct transmission of the RSA-encrypted secret or by the transmission of Diffie-Hellman parameters"); *see also id.*, 61 ("This structure conveys the client's Diffie-Hellman public value (Y_c)"); *see* EX-1006, 85-88 (providing contents of ClientKeyExchange message for RSA and Diffie-Hellman).

374. Using Diffie-Hellman key exchange, the client sends the "client's Diffie-Hellman public value (Y_c)," i.e., the claimed public Diffie-Hellman client key, to the server. TLS-1.2, 57-61 (Client Diffie-Hellman Public Value). TLS-1.2 also generates a private Diffie-Hellman client key by "generating a fresh DH private

key (X) for each handshake.” *Id.*, 93, 87 (cautioning to use a strong and properly seeded random number generator for generating Diffie-Hellman private values).

375. TLS-1.2 suggests implementing Diffie_Hellman key exchange according to RFC 2785 (“Methods for Avoiding the ‘Small-Subgroup’ Attacks on the Diffie-Hellman Key Agreement Method for S/MIME”). *See id.*, 93 (citing [SUBGROUP]), 100 (defining [SUBGROUP] as RFC 2785). RFC 2785 explains that, during Diffie-Hellman Key Exchange between Party A and Party B, each of parties A and B generates a **private key** (x_a and x_b , respectively) and then generates a **public key** (y_a and y_b , respectively) based on its private key:

```

ya is Party A's public key;  $y_a = g^{x_a} \bmod p$ 
yb is Party B's public key;  $y_b = g^{x_b} \bmod p$ 
xa is Party A's private key;  $x_a$  is in the interval  $[2, (q - 2)]$ 
xb 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 mod p)
q is a large prime
j a large integer such that  $p = q*j + 1$ 

```

EX-1011, 2

376. 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. The Tkacik-TLS-1.2 combinations implementing Diffie-Hellman, including RFC 2785, generates a public Diffie-Hellman client key and a private Diffie-Hellman client key.

377. As discussed, TLS-1.2's methods would have been implemented by Tkacik. For example, Tkacik teaches that secure processing section 30 is used to generate keys using any of a number of protocols known to those skilled in the art, including . . . DH, RSA key gen . . . SSL, . . . and the like." Tkacik ¶¶79-80. Tkacik-TLS-1.2 therefore satisfies the security engine generating a private Diffie-Hellman client key and generating a public Diffie-Hellman client key based on the private Diffie-Hellman client key, as claimed.

- b. [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."**

378. Tkacik-TLS-1.2 satisfies this feature.

379. For example, during the TLS Handshake protocol, it is necessary to sign the public Diffie-Hellman client key and send the signed public Diffie-Hellman key to the remote server. *See* TLS-1.2, 62 (Certificate Verify). TLS-1.2 teaches that the public Diffie-Hellman client key Y_c (the claimed "public Diffie-Hellman client key") needs to be sent to the server so the server can authenticate the client and verify the client has corresponding private key. *See id.* (showing the client key exchange message includes the public Diffie-Hellman client key Y_c). More specifically, TLS-1.2 teaches sending a Certificate Verify message after the ClientKeyExchange message. The Certificate Verify message includes "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 . According to TLS-1.2, the purpose of sending the Certificate Verify message is to “provide explicit verification of a client certificate.” *Id.* (Certificate Verify).

380. TLS-1.2 also suggests using a key specific to the client device so the server could verify and authenticate the signed public Diffie-Hellman client key based a device-specific key. *See* TLS-1.2, 33-37 (“Exchange certificates and cryptographic information to allow the client and server to authenticate themselves . . . and to explicitly verify possession of the private key in the certificate.”). A POSITA would have found it obvious to have verified and authenticated the signed public Diffie-Hellman client key using the claimed security key. *See* [1a-3].

381. Because Tkacik teaches the security key is encoded in the secure memory during manufacturing of the client device, the security key is a device-specific key that a POSITA would have found obvious to use for signing the public Diffie-Hellman client key so that the server could verify the identity of the client device based on the security key. *See* Tkacik ¶46. A POSITA thus would have found claim feature [6b] obvious based on the combined teachings of Tkacik and TLS-1.2.

9. **Claim 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.”**

382. Tkacik-TLS-1.2 satisfies this feature.

383. For example, TLS-1.2 discloses that the client receives a public Diffie-Hellman server key from the remote server by receiving a public Diffie-Hellman server key `dh_Y` to complete a key exchange and to calculate the premaster secret. *See* TLS-1.2, 50-52 (ServerKeyExchange message).

384. TLS-1.2 discloses generating the session key as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key in two ways.

385. First, TLS-1.2 generates the master secret (e.g., the “session key”) as a function of the pre-master secret. *See id.*, 64 (“`master_secret = PRF(pre_master_secret, “master secret”, ClientHello.random + ServerHello.random)`”). The client device generates the premaster secret by performing conventional Diffie-Hellman using the public Diffie-Hellman server key received from the server. *See id.*, 51 (explaining the result of Diffie-Hellman key exchange is the “premaster secret”), 65 (Diffie-Hellman); *see also id.*, 93 (using RFC 2785 to implement Diffie-Hellman when calculating the premaster secret). RFC 2785, in turn, teaches that a shared key (ZZ), i.e., the pre-master key, is generated based on the calculation below.

Note that the individual parties actually perform the computations:

$$ZZ = (yb \wedge xa) \bmod p = (ya \wedge xb) \bmod p$$

where $\wedge$ denotes exponentiation.

ya is Party A's public key; $ya = g \wedge xa \bmod p$
yb is Party B's public key; $yb = g \wedge xb \bmod p$
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
 $g = h^{((p-1)/q)} \bmod p$, where
h is any integer with $1 < h < p-1$ such that $h^{((p-1)/q)} \bmod p > 1$
(g has order q mod p)
q is a large prime
j a large integer such that $p=q*j + 1$

EX-1011, 2.

386. When Party A is the client device and Party B is the server, RFC 2785 teaches that the shared key (ZZ), which TLS-1.2 uses as premaster key, is generated as a function of public Diffie-Hellman server key yb and private Diffie-Hellman client key xa (specifically, $ZZ = (yb \wedge xa) \bmod p$). TLS-1.2 thus teaches generating the premaster secret as a function of public Diffie-Hellman server key yb and private Diffie-Hellman client key xa.

387. The premaster secret is therefore a function of the private Diffie-Hellman client key xa and the public Diffie-Hellman server key yb. Because TLS-1.2 teaches generating the master secret as a function of the pre-master secret, and the premaster secret is a function of the private Diffie-Hellman client key and public Diffie-Hellman server keys, the master secret (the claimed “session key”) is generated as a function of the public Diffie-Hellman server key and private Diffie-Hellman client key.

388. Second, TLS-1.2 teaches generating the `key_block` (the claimed “session key”) as a function of the private Diffie-Hellman client key and the public Diffie-Hellman server key. TLS-1.2 discloses (1) generating the `key_block` (the claimed “session key”) as a function of master secret; (2) generating the master secret as a function of pre-master secret; and (3) generating a premaster secret as a function of the private Diffie-Hellman client key x_a and the public Diffie-Hellman server key y_b (as discussed above).

389. TLS-1.2 explains that the `key_block` is generated as a function of the master secret. See TLS-1.2, 26 (“`key_block = PRF(SecurityParameters.master_secret, ‘key expansion’, SecurityParameters.server_random + SecurityParameters.client_random)’`”). Because the master secret is generated as a function of the pre-master secret (as discussed above), and because the pre-master secret is generated as a function of the private Diffie-Hellman client key and public Diffie-Hellman server key (as also discussed above), the `key_block` is generated as a function of the private Diffie-Hellman client key and public Diffie-Hellman server key. *See supra* [1d].

- 10. Claim 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.”**

390. Tkacik-TLS-1.2 satisfies this feature.

391. As discussed in claim 6, a POSITA would have understood that the symmetric key or master secret in the TLS-1.2 protocol may be generated by either (RSA) key exchange or a Diffie-Hellman key exchange. *See* TLS-1.2, 55-57 (client certificate including RSA or Diffie-Hellman parameters), 57-58 (client key exchange message based on RSA or Diffie-Hellman), 61, 65, 74-75, 86, 92-93, 100; EX-1006, 81-84 (describing different ServerKeyExchange message formats for Diffie-Hellman, RSA, and Fortezza key exchange protocols), 40-45 (ServerKeyExchange message), 54-59 (separate ServerKeyExchange message and server certificate); *see* [1c] (describing ServerKeyExchange message with or with certificate). To implement a Diffie-Hellman key exchange, the Tkacik-TLS-1.2 combination would have implemented the Diffie-Hellman key exchange teachings of TLS-1.2 on Tkacik's hardware.

392. In doing so, 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 (ServerKeyExchange Message). 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.

393. In the combination, Tkacik's secure processing section 30 generates the session key using the TLS/SSL Handshake protocol, such that it must receive the TLS-1.2 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* [1c]-[1d]. For example, Tkacik's secure processing section 30 is used to generate keys using any of a number of protocols known to those skilled in the art, including . . . [Diffie-Hellman], . . . SSL, . . . and the like." Tkacik ¶¶79-80.

394. 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 to derive the symmetric key. Tkacik explains that "the secret keys [which would at least include the public Diffie-Hellman server key] should be saved for subsequent decryption operations, and they should be kept secret. One conventional way to maintain the privacy is to save the keys in a secure memory device under the control of a SEE." Tkacik ¶7; *see also id.* ¶26 ("memory located within secure processing section 30 (discussed below) is secured memory"). Because both TLS-1.2 and Tkacik teach or suggest the Diffie-Hellman global values and the public Diffie-Hellman server key are essential security parameters for generating encryption keys in the SSL protocol, a POSITA would have found it obvious to store them in Tkacik's secure memory within the security engine

(including, e.g., memory 52) to prevent attackers from obtaining them and for computing the secrets and encryption keys.

11. Claim 9

- a. **[9a] “A method comprising: 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”**

395. Tkacik-TLS-1.2 satisfies this feature. *See* [1p]-[1a-3].

- b. **[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;”**

396. Tkacik-TLS-1.2 satisfies this feature. *See* [1b].

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

397. Tkacik-TLS-1.2 satisfies this feature. *See* [1c].

- d. **[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;”**

398. Tkacik-TLS-1.2 satisfies this feature. *See* [1d].

- e. **[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;”**

399. Tkacik-TLS-1.2 satisfies this feature. *See* [1a-2], [1a-3], [1e].

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

400. Tkacik-TLS-1.2 satisfies this feature. *See* [1a-2], [1f].

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

401. Tkacik-TLS-1.2 satisfies this feature. *See* [1g].

12. Claim 10

- a. **[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”**

402. Tkacik-TLS-1.2 satisfies this feature. *See* [1b].

- b. **[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.”**

403. Tkacik-TLS-1.2 satisfies this feature. *See* [1b], claim 2.

- 13. Claim 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.”**

404. Tkacik-TLS-1.2 satisfies this feature. *See* [1d], claim 3.

14. **Claim 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.”**

405. Tkacik-TLS-1.2 satisfies this feature. *See* claim 4.

15. **Claim 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.”**

406. Tkacik-TLS-1.2 satisfies this feature. *See* [1d], claim 3, [5a].

16. **Claim 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.”**

407. Tkacik-TLS-1.2 satisfies this feature. *See* [1d], [5b].

17. **Claim 15**

- a. **[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;”**

408. Tkacik-TLS-1.2 satisfies this feature. *See* [1d], [6a].

- b. **[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;”**

409. Tkacik-TLS-1.2 satisfies this feature. *See* [6b].

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

410. Tkacik-TLS-1.2 satisfies this feature. *See* claim 7.

- d. **[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.”**

411. Tkacik-TLS-1.2 satisfies this feature. *See* claim 7.

18. Claim 16

- a. **[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;”**

412. Tkacik-TLS-1.2 satisfies this feature. *See* claim [5a], [5b].

413. TLS-1.2 teaches, for example, sending a client Certificate Verify message includes a hash of prior handshake messages. TLS-1.2, 62 (Certificate Verify); *see* [5b]. Those prior handshake messages include all handshake messages sent or received by the client, starting with the ClientHello message. *See id.* The client Certificate Verify message sent following the client key exchange message includes a hash code of the ClientHello and client key exchange messages (the claimed “a preceding message sent from the client to the remote server”).

- b. **[16b] “signing the hash code using the security key stored in the secure memory of the security engine; and;”**

414. Tkacik-TLS-1.2 satisfies this feature.

415. For example, TLS-1.2 explains the purpose of sending the Certificate Verify message is to “provide explicit verification of a client certificate.” TLS-1.2, 62 (Certificate Verify). Although TLS-1.2 may not specify a specific key to be used for signing the hash code, the purpose of signing in the TLS protocol is to allow the server to verify the identity of the client device. A POSITA therefore would have found it obvious to use a key that is specific to the client to sign the hash code.

416. As discussed, Tkacik discloses such a security key stored in the secure memory of the security engine. *See* [1a-3]. For example, Tkacik teaches a long-lived key register 60 (i.e., the claimed “secure memory”) that includes a plaintext master key 160 (i.e., the claimed “security key”) encoded in the long-lived key register 60 (part of the claimed “secure memory”) when manufacturing secure processing system 12 (the claimed “system-on-a-chip”). *See* Tkacik ¶46.

417. A POSITA would have found it obvious to combine the teachings in Tkacik and TLS 1.2 such that the client device implementing TLS version 1.2 using Tkacik’s security engine would have signed the hash code with Tkacik’s security key in the secure memory.

c. [16c] “sending a client certificate verification message to the remote server, the client certificate verification message including the signed hash code.”

418. Tkacik-TLS-1.2 satisfies this feature. TLS-1.2, 62 (Certificate Verify); *see* [5b], [16a].

19. Claim 17

- a. **[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:”**

419. Tkacik-TLS-1.2 satisfies this feature. *See* [1p]-[1a-3]. A POSITA would have understood that implementing the Tkacik-TLS-1.2 combination includes implementing TLS-1.2’s functionality using a computing device comprising Tkacik’s hardware. Tkacik’s hardware includes 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 perform functions. *See* [1p]-[1a-3].

- b. **[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;”**

420. Tkacik-TLS-1.2 satisfies this feature. *See* [1p]-[1a-3], [1b].

- c. **[17b] “initiate a request for a secure communication session with a remote server over a network, the request including the random nonce;”**

421. Tkacik-TLS-1.2 satisfies this feature. *See* [1b].

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

422. Tkacik-TLS-1.2 satisfies this feature. *See* [1c].

- e. **[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;”**

423. Tkacik-TLS-1.2 satisfies this feature. *See* [1d].

- f. **[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;”**

424. Tkacik-TLS-1.2 satisfies this feature. *See* [1p]-[1a-3], [1e].

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

425. Tkacik-TLS-1.2 satisfies this feature. *See* [1a-2], [1f].

- h. **[17g] “establish the secure communication session with the remote server using the session key”**

426. Tkacik-TLS-1.2 satisfies this feature. *See* [1f].

20. Claim 18

- a. **[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”**

427. Tkacik-TLS-1.2 satisfies this feature. *See* [10a], [1b].

- b. **[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.”**

428. Tkacik-TLS-1.2 satisfies this feature. *See* [10b], claim 2, [1b].

- 21. **Claim 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.”**

429. Tkacik-TLS-1.2 satisfies this feature. *See* claim 3.

- 22. **Claim 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.”**

430. Tkacik-TLS-1.2 satisfies this feature. *See* claim 4.

- 23. **Claim 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.”**

431. Tkacik-TLS-1.2 satisfies this feature. *See* [1d], [5a].

24. **Claim 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.”**

432. Tkacik-TLS-1.2 satisfies this feature. *See* [1d], [5b].

25. **Claim 23**

- a. **[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”**

433. Tkacik-TLS-1.2 satisfies this feature. *See* [1d], [6a].

- b. **[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;”**

434. Tkacik-TLS-1.2 satisfies this feature. *See* [6b].

- c. **[23c] “receive a public Diffie-Hellman server key from the remote server; and”**

435. Tkacik-TLS-1.2 satisfies this feature. *See* claim 7.

- d. **[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.”**

436. Tkacik-TLS-1.2 satisfies this feature. *See* claim 7.

26. Claim 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.”

437. Tkacik-TLS-1.2 satisfies this feature. *See* [1p]-[1a-3]. For example, Tkacik’s Figure 1 shows that secure processing section 30 (the claimed “security engine”) is communicatively coupled to the programmable processor 26 (the claimed “processor core”) through the bus 24 (the claimed “interconnect”) of the secure processing system 12 (the claimed “system-on-a-chip”).

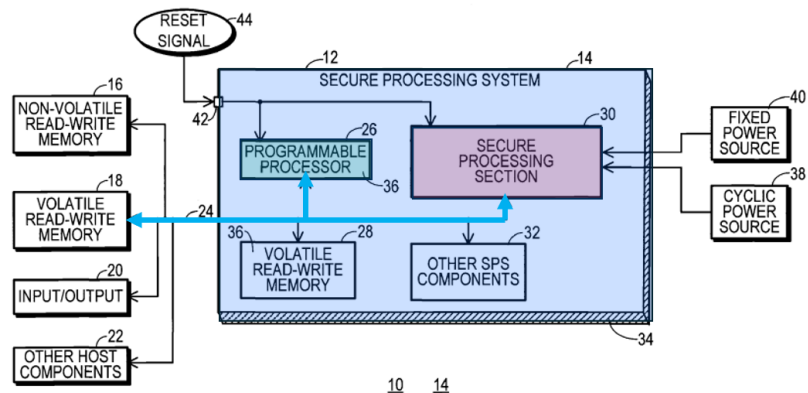


FIG. 1

Id., Fig. 1 (annotated).

438. This is consistent with the description of the ’838 patent’s specification, which defines “interconnect such as a bus, point-to-point, or other interconnect capable of facilitating communication between the various components of the SOC.” EX-1001, 3:67-4:3.

IX. SECONDARY CONSIDERATIONS

439. I have considered the issue of secondary considerations and am currently unaware of any evidence of secondary considerations that would support a finding of non-obviousness or otherwise impact my opinions herein.

X. CONCLUSION

440. For the reasons set forth above, it is my opinion that claims 1-24 of the '838 patent are unpatentable in view of the prior art.

441. In signing this declaration, I understand that the declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I acknowledge that I may be subject to cross-examination in this case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

442. I declare that all statements made herein of my knowledge are true, that all statements made on information and belief are believed to be true, and that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Declaration of Paul D. Martin, Ph.D.
U.S. Patent No. 9,887,838

Dated: November 15, 2024

By: _____

Paul D. Martin, Ph.D.