IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

CRYPTOPEAK SECURITY, LLC, Plaintiff, v. JET.COM, INC., Defendant.	**Civil Action No. _____** **JURY TRIAL DEMANDED**

COMPLAINT FOR PATENT INFRINGEMENT

Plaintiff CryptoPeak Security, LLC ("Plaintiff"), for its Complaint against Defendant Jet.com, Inc. ("Jet.com" or "Defendant") alleges the following:

NATURE OF THE ACTION

1. This is an action for patent infringement arising under the Patent Laws of the United States, 35 U.S.C. § 1 *et seq*.

THE PARTIES

2. Plaintiff is a limited liability company organized under the laws of the State of Delaware with a place of business at 717 N. Union Street, Suite 106, Wilmington, DE 19805.

3. Upon information and belief, Jet.com is a Corporation organized and existing under the laws of Delaware, with a place of business at 353 Bloomfield Ave., Montclair, NJ 07042, and can be served through its registered agent, Corporation Trust Company at 1209 Orange Street, Wilmington, DE 19801. Upon information and belief, Jet.com sold and offered to sell products and services throughout the United States, including in this judicial district, and introduced products and services that into the stream of commerce and that incorporate infringing

technology knowing that they would be sold in this judicial district and elsewhere in the United States.

JURISDICTION AND VENUE

4. This is an action for patent infringement arising under the Patent Laws of the United States, Title 35 of the United States Code.

5. This Court has subject matter jurisdiction under 28 U.S.C. §§ 1331 and 1338(a).

6. Venue is proper in this judicial district under 28 U.S.C. §1400(b). On information and belief Jet.com is incorporated in the State of Delaware.

7. On information and belief, Defendant is subject to this Court's general and specific personal jurisdiction because Defendant has sufficient minimum contacts within the State of Delaware and this District, pursuant to due process and/or the Delaware Long Arm Statute because Defendant purposefully availed itself of the privileges of conducting business in the State of Delaware and in this District, because Defendant regularly conducts and solicits business within the State of Delaware and within this District, and because Plaintiff's causes of action arise directly from Defendant's business contacts and other activities in the State of Delaware and this District. Further, this Court has personal jurisdiction over Defendant because it is incorporated in Delaware and has purposely availed itself of the privileges and benefits of the laws of the State of Delaware.

BACKGROUND

8. This action relates to United States Patent No. 6,202,150 B1 (the "'150 patent"). The application leading to the '150 patent was filed on May 28, 1997. The inventors are Dr. Adam L. Young and Dr. M. M. ("Moti") Yung. Both Dr. Yung and Dr. Young are noted and accomplished experts in cryptology.

9. Dr. Moti Yung obtained his Ph.D. in Computer Science in 1988 at Columbia University. His professional career includes research and technical work for IBM, RSA Security (now a division of EMC), and Google. He has been an adjunct professor for many years at Columbia University, serving on Ph.D. committees and advising more than 60 Ph.D. students. He is an author or co-author of more than 300 refereed abstracts and journal papers, including several in collaboration with Dr. Young. He is an inventor on dozens of issued U.S. patents. He is a Fellow of the ACM (Association for Computing Machinery), the IACR (International Association for Cryptologic Research), and the IEEE (Institute of Electrical and Electronics Engineers).

10. Dr. Adam Young obtained his Ph.D. in Computer Science in 2002 at Columbia University. His professional career includes research and technical work for Lucent, Lockheed Martin, MITRE Corporation, and Bloomberg. He has been a guest lecturer at NYU and Rensselaer Polytechnic Institute. He is an author or co-author of more than three dozen papers and journal articles, including several with Dr. Yung. He is an inventor on at least eight issued U.S. patents.

11. Dr. Yung and Dr. Young also co-authored a book published in 2004, entitled "Malicious Cryptography: Exposing Cryptovirology."

12. The '150 patent has been forward-cited in connection with the examination of at least twenty subsequently-issued U.S. patents, including patents originally assigned to technology companies including Microsoft, HP, General Instruments, Ricoh and Sungard.

13. The invention of the '150 patent was sufficiently prominent that an article, entitled "Auto-Recoverable Auto-Certifiable Cryptosystems," which is related to the subject matter of the '150 patent, was published and presented by Drs. Yung and Young in connection

with the prestigious EUROCRYPT ’98 conference in Espoo, Finland. EUROCRYPT is an annual conference that has been held since 1982, and it is one of the IACR’s three flagship conferences, along with CRYPTO and ASIACRYPT.

14. In 2015, plaintiff commenced a series of legal actions against multiple defendants asserting infringement of ’150 patent in the Eastern District of Texas. In 2016, several defendants filed a Motion to Dismiss For Failure to State a Claim, on the grounds that the claims were invalid under 35 U.S.C. §112 as reciting both a method and apparatus, invalid under §101, and indefinite under §112. Plaintiff contested those motions.

15. In a Report and Recommendation issued by the Magistrate Judge Hon. Roy S. Payne, Judge Payne recommended denying each basis of invalidity. (Exhibit 2.) Defendants filed objections to the Report and Recommendations.

16. In a subsequent Order, District Judge Hon. Robert W. Schroeder III confirmed the Magistrate Judge’s rulings and denied Defendants' Motion to Dismiss. (Exhibit 3.) Specifically, the Court held the claim to recite a method, rather than an apparatus or a mixture of method and apparatus. Thus the claims were not indefinite despite the preamble’s reference to a “method and apparatus.” (Ex. 3 at 3-4.) The Court further adopted the Report and Recommendations and denied the §101 and indefiniteness challenges, at the pleadings stage. (Ex. 3 at 5.) Accordingly, each of Defendants’ bases was denied without prejudice, subject to later claim construction. (Ex. 3 at 5.)

COUNT I – INFRINGEMENT OF U.S. PATENT NO. 6,202,150 B1

17. The allegations set forth in the foregoing paragraphs 1 through 16 are incorporated into this First Claim for Relief.

18. On March 13, 2001, U.S. Patent No. 6,202,150 B1 ("the '150 patent"), entitled "Auto-Escrowable And Auto-Certifiable Cryptosystems" was duly and legally issued by the United States Patent and Trademark Office. A true and correct copy of the '150 patent is attached as Exhibit 1.

19. The inventions of the '150 patent resolve technical problems related to computerized cryptography, and the secure transmission of information over computer networks.

20. The claims of the '150 patent do not merely recite the performance of some business practice known from the pre-Internet world along with the requirement to perform it on the Internet. Instead, the claims of the '150 patent recite one or more inventive concepts that are rooted in computerized cryptographic technology, and overcome problems specifically arising in that technological realm.

21. The claims of the '150 patent recite an invention that is not merely the routine or conventional use of networking techniques. Instead, the invention provides a new and unconventional solution to specific problems related to the security of information within computer network transmissions.

22. The technology claimed in the '150 patent does not preempt all ways of using cryptographic techniques to protect information, nor preempt the use of all public/private key technologies, nor preempt any other well-known or prior art technology.

23. Accordingly, each claim of the '150 patent recites a combination of elements sufficient to ensure that the claim in practice amounts to significantly more than a patent on an ineligible concept.

24. Plaintiff is the assignee and owner of the right, title and interest in and to the ’150 patent, including the right to assert all causes of action arising under said patents and the right to any remedies for infringement of them.

25. Upon information and belief, Defendant has directly infringed at least claims 1 and 3 of the ’150 patent by making, using, selling, importing and/or providing and causing to be used one or more computers or computer server that transmit information utilizing one or more of the following Ecliptic Curve Cryptography ciphersuites (those that begin with TLS_ECDHE):

Server 1

# TLS 1.2 (suites in server-preferred order)			
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d)			
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)			
TLS_RSA_WITH_AES_256_CBC_SHA256 (0x3d)			
TLS_RSA_WITH_AES_128_CBC_SHA256 (0x3c)			
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)			
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)			
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa)	WEAK		

Server 2

# TLS 1.2 (suites in server-preferred order)			
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d)			
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)			
TLS_RSA_WITH_AES_256_CBC_SHA256 (0x3d)			
TLS_RSA_WITH_AES_128_CBC_SHA256 (0x3c)			
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)			
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)			
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa)	WEAK		

Server 3

Collectively, computers or servers communicating in accordance with such ciphersuites are referred to as the "Accused Instrumentalities".

26. In particular, claim 1 of the '150 patent recites a method for generating public keys and a proof that the keys are generated by a specific algorithm comprising the steps of: the user's system generating a random string of bits based on system parameters; the user running a key generation algorithm to get a secret key and public key using the random string and public parameters; the user constructing a proof being a string of bits whose public availability does not compromise the secret key and wherein said constructing of said proof requires access to said secret key, but at the same time said proof provides confidence to at least one of a plurality of other entities that said public key was generated properly by the specified algorithm, and wherein said confidence is gained without having access to any portion of said secret key.

27. The Accused Instrumentalities infringed claim 1 of the '150 patent during the pendency of the '150 patent.

28. In particular, the Accused Instrumentalities practiced a method for generating public keys and a proof that the keys were generated by a specific algorithm.

29. For example, the Accused Instrumentalities included the server(s) corresponding to jet.com, which performs secure communication using Transport Layer Security (TLS) version 1.2 and supports ciphersuite(s) that are based on elliptic curve (EC) cryptography with Diffie-Hellman (DH) key exchange. For example, Qualys, Inc.'s SSL Server Test tool (available at https://www.ssllabs.com/ssltest/) indicates that the above-mentioned Jet.com server(s) support the following ciphersuites for TLS 1.2 in order of preference, including several that are based on ECDH (i.e., start with "TLS_ECDH"):

 # TLS 1.2 (suites in server-preferred order)

TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	0xc030	ECDH secp256r1 (eq. 3072 bits RSA)	FS
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	0xc02f	ECDH secp256r1 (eq. 3072 bits RSA)	FS
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	0xcca8	ECDH secp256r1 (eq. 3072 bits RSA)	FS
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	0xc028	ECDH secp256r1 (eq. 3072 bits RSA)	FS
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	0xc027	ECDH secp256r1 (eq. 3072 bits RSA)	FS
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	0xc014	ECDH secp256r1 (eq. 3072 bits RSA)	FS
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	0xc013	ECDH secp256r1 (eq. 3072 bits RSA)	FS
TLS_RSA_WITH_AES_256_GCM_SHA384	0x9d		
TLS_RSA_WITH_AES_128_GCM_SHA256	0x9c		
TLS_RSA_WITH_AES_256_CBC_SHA256	0x3d		
TLS_RSA_WITH_AES_128_CBC_SHA256	0x3c		
TLS_RSA_WITH_AES_256_CBC_SHA	0x35		
TLS_RSA_WITH_AES_128_CBC_SHA	0x2f		
TLS_RSA_WITH_3DES_EDE_CBC_SHA	0xa	**WEAK**	

Server 1

# TLS 1.2 (suites in server-preferred order)			
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d)			
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)			
TLS_RSA_WITH_AES_256_CBC_SHA256 (0x3d)			
TLS_RSA_WITH_AES_128_CBC_SHA256 (0x3c)			
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)			
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)			
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa)	WEAK		

Server 2

# TLS 1.2 (suites in server-preferred order)			
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d)			
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)			
TLS_RSA_WITH_AES_256_CBC_SHA256 (0x3d)			
TLS_RSA_WITH_AES_128_CBC_SHA256 (0x3c)			
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)			
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)			
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa)	WEAK		

Server 3

30. As further explained below, the Accused Instrumentalities, at least by performing secure communication using ECDH, performed a method for generating public keys and a proof that the keys were generated by a specific algorithm.

31. The Accused Instrumentalities comprised the step of the user's system generating a random string of bits based on system parameters. (It is noted that "user's system" and "user" here refers to the Accused Instrumentalities.)

32. TLS 1.2 is described in Request for Comments (RFC) 5246, entitled "The Transport Layer Security (TLS) Protocol Version 1.2," dated August 2008, and available at https://tools.ietf.org/html/rfc5246 (hereinafter "RFC 5246"). RFC 5246 references RFC 4492, entitled "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)," dated May 2006, and available at https://tools.ietf.org/html/rfc4492 (hereinafter "RFC 4492").

33. RFC 5246 and RFC 4492 state the following. (Fig. 1 to Fig. 4).

RFC 5246 TLS August 2008

The TLS Handshake Protocol involves the following steps:

- Exchange hello messages to agree on algorithms, exchange random values, and check for session resumption.

- Exchange the necessary cryptographic parameters to allow the client and server to agree on a premaster secret.

- Exchange certificates and cryptographic information to allow the client and server to authenticate themselves.

- Generate a master secret from the premaster secret and exchanged random values.

- Provide security parameters to the record layer.

- Allow the client and server to verify that their peer has calculated the same security parameters and that the handshake occurred without tampering by an attacker.

Fig. 1 (RFC 5246, page 34, describing that the server exchanges random values [random string of bits] with the client)

These goals are achieved by the handshake protocol, which can be summarized as follows: The 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. The ClientHello and ServerHello are used to establish security enhancement capabilities between client and server. 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.

Fig. 2 (RFC 5246, page 34 describing that the server generates ServerHello.random [random string of bits])

D.1. Random Number Generation and Seeding

TLS requires a cryptographically secure pseudorandom number generator (PRNG). Care must be taken in designing and seeding PRNGs. PRNGs based on secure hash operations, most notably SHA-1, are acceptable, but cannot provide more security than the size of the random number generator state.

To estimate the amount of seed material being produced, add the number of bits of unpredictable information in each seed byte. For example, keystroke timing values taken from a PC compatible's 18.2 Hz timer provide 1 or 2 secure bits each, even though the total size of the counter value is 16 bits or more. Seeding a 128-bit PRNG would thus require approximately 100 such timer values.

[RANDOM] provides guidance on the generation of random values.

Fig. 3 (RFC 5246, page 85 describing that a random number generation can include utilizing a hardware timer [system parameters])

3.2. Existing Hardware Can Be Used For Randomness

As described below, many computers come with hardware that can, with care, be used to generate truly random quantities.

Fig. 4 (RFC 4086, referenced in Fig. 3 as "[RANDOM"], dated June 2005 and available at

https://tools.ietf.org/html/rfc4086, page 8, describing that hardware [associated with system parameters] of the server can be used to generate random values)

34. The Accused Instrumentalities comprised the step of the user running a key generation algorithm to get a secret key and public key using the random string and public parameters.

35. RFC 5246 states the following. (Fig. 5, to Fig. 10)

The TLS Handshake Protocol involves the following steps:

- Exchange hello messages to agree on algorithms, exchange random values, and check for session resumption.
- Exchange the necessary cryptographic parameters to allow the client and server to agree on a premaster secret.
- Exchange certificates and cryptographic information to allow the client and server to authenticate themselves.
- Generate a master secret from the premaster secret and exchanged random values.
- Provide security parameters to the record layer.
- Allow the client and server to verify that their peer has calculated the same security parameters and that the handshake occurred without tampering by an attacker.

Fig. 5 (RFC 5246, page 34 describing the generation of a master secret [secret key] using random values [random string])

7.4.3. Server Key Exchange Message

When this message will be sent:

This message will be sent immediately after the server Certificate message (or the ServerHello message, if this is an anonymous negotiation).

Fig. 6

```
Meaning of this message:

   This message 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.
```

Fig. 7 (RFC 5246, pages 50-51, describing the use of a premaster secret [public key] during the TLS handshake)

```
8.1.  Computing the Master Secret

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

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

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

Fig. 8 (RFC 5246, page 64, describing that the master secret [secret key] is generated using the ClientHello.random and ServerHello.random values [random string])

```
Actions of the receiver:

A server that receives a ClientHello containing one or both of these
extensions MUST use the client's enumerated capabilities to guide its
selection of an appropriate cipher suite.  One of the proposed ECC
cipher suites must be negotiated only if the server can successfully
complete the handshake while using the curves and point formats
supported by the client (cf. Sections 5.3 and 5.4).
```

Fig. 9 (RFC 4492, page 11, describing the use of a known elliptic curve and point formats [public parameters])

5.10. ECDH, ECDSA, and RSA Computations

All ECDH calculations (including parameter and key generation as well as the shared secret calculation) are performed according to [6] using the ECKAS-DH1 scheme with the identity map as key derivation function (KDF), so that the premaster secret is the x-coordinate of the ECDH shared secret elliptic curve point represented as an octet string. Note that this octet string (Z in IEEE 1363 terminology) as output by FE2OSP, the Field Element to Octet String Conversion Primitive, has constant length for any given field; leading zeros found in this octet string MUST NOT be truncated.

Fig. 10 (RFC 4492, page 26, describing that the premaster secret [public key] is generated using the elliptic curve [public parameters])

36. The Accused Instrumentalities comprised the step of the user constructing a proof being a string of bits whose public availability does not compromise the secret key and wherein said constructing of said proof requires access to said secret key, but at the same time said proof provides confidence to at least one of a plurality of other entities that said public key was generated properly by the specified algorithm, and wherein said confidence is gained without having access to any portion of said secret key.

37. RFC 5246 states the following. (Fig. 11)

```
7.4.9.  Finished

   When this message will be sent:

      A Finished message is always sent immediately after a change
      cipher spec message to verify that the key exchange and
      authentication processes were successful.  It is essential that a
      change cipher spec message be received between the other handshake
      messages and the Finished message.

   Meaning of this message:

      The Finished message is the first one protected with the just
      negotiated algorithms, keys, and secrets.  Recipients of Finished
      messages MUST verify that the contents are correct.  Once a side
      has sent its Finished message and received and validated the
      Finished message from its peer, it may begin to send and receive
      application data over the connection.

   Structure of this message:

      struct {
          opaque verify_data[verify_data_length];
      } Finished;

      verify_data
         PRF(master_secret, finished_label, Hash(handshake_messages))
            [0..verify_data_length-1];

      finished_label
         For Finished messages sent by the client, the string
         "client finished".  For Finished messages sent by the server,
         the string "server finished".
```

Fig. 11 (RFC 5246, page 63, describing the use of a Finished message [proof] in the TLS handshake)

38. As shown in ¶¶ 37, the Finished message from the server must be validated by the client before application data is sent over the established TLS 1.2 connection. Because the contents of the Finished message [proof] is a function of the master_secret [secret key] (which is derived from the Diffie-Hellman premaster secret) and a function of the preceding handshake messages, validating the Finished message verifies that the relevant keys, including the premaster secret and the master secret, were generated properly by the specified algorithm in use (e.g., TLS 1.2 ECDH).

39. Instead of receiving the server's generated master_secret, the client generates a master_secret independently based on the premaster secret and based on the previously exchanged client/server random values to validate the Finished message from the server.

40. RFC 5246 states the following. (Fig. 12)

F.1.3. Detecting Attacks Against the Handshake Protocol

An attacker might try to influence the handshake exchange to make the parties select different encryption algorithms than they would normally choose.

For this attack, an attacker must actively change one or more handshake messages. If this occurs, the client and server will compute different values for the handshake message hashes. As a result, the parties will not accept each others' Finished messages. Without the master_secret, the attacker cannot repair the Finished messages, so the attack will be discovered.

Fig. 12 (RFC 5246, page 94, describing that because the server does not send its generated master secret [secret key] to the client, and vice versa, an attacker does not possess the master secret [secret key])

41. Claim 3 of the '150 patent recites a method for publishing public keys and a proof that the keys and a proof that the keys were generated by a specific algorithm comprising the steps of: the user's system reading the system parameters; the user's system running a key generation algorithm to get a private key and public key; the user's system constructing a proof that the private key was generated properly using the system parameters, and where said constructing of said proof requires access to said private key and verification of said proof can be performed by any other entity with no access to any portion of said private key.

42. The Accused Instrumentalities infringed claim 3 of the '150 patent during the pendency of '150 patent.

43. In particular, the Accused Instrumentalities included a method for publishing public keys and a proof that the keys were generated by a specific algorithm.

44. For example, the Accused Instrumentalities included the server(s) corresponding to jet.com, which performs secure communication using Transport Layer Security (TLS) version 1.2 and supports ciphersuite(s) that are based on elliptic curve (EC) cryptography with Diffie-Hellman (DH) key exchange. For example, Qualys, Inc.'s SSL Server Test tool (available at https://www.ssllabs.com/ssltest/) indicates that the above-mentioned Jet.com server(s) support the following ciphersuites for TLS 1.2 in order of preference, including several that are based on ECDH (i.e., start with "TLS_ECDH"):

Server 1

# TLS 1.2 (suites in server-preferred order)			
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d)			
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)			
TLS_RSA_WITH_AES_256_CBC_SHA256 (0x3d)			
TLS_RSA_WITH_AES_128_CBC_SHA256 (0x3c)			
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)			
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)			
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa)	WEAK		

Server 2

# TLS 1.2 (suites in server-preferred order)			
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d)			
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)			
TLS_RSA_WITH_AES_256_CBC_SHA256 (0x3d)			
TLS_RSA_WITH_AES_128_CBC_SHA256 (0x3c)			
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)			
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)			
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa)	WEAK		

Server 3

45. As further explained below, the Accused Instrumentalities, at least by performing secure communication using ECDH, performed a method for publishing public keys and a proof that the keys were generated by a specific algorithm.

46. The Accused Instrumentalities comprised the step of the user's system reading the system parameters. (It is noted that "user's system" here refers to the Accused Instrumentalities.)

47. TLS 1.2 is described in Request for Comments (RFC) 5246, entitled "The Transport Layer Security (TLS) Protocol Version 1.2," dated August 2008, and available at https://tools.ietf.org/html/rfc5246 (hereinafter "RFC 5246"). RFC 5246 references RFC 4492, entitled "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)," dated May 2006, and available at https://tools.ietf.org/html/rfc4492 (hereinafter "RFC 4492").

48. RFC 5246 states the following. (Fig. 13, Fig. 14)

RFC 5246 TLS August 2008

The TLS Handshake Protocol involves the following steps:

- Exchange hello messages to agree on algorithms, exchange random values, and check for session resumption.

- Exchange the necessary cryptographic parameters to allow the client and server to agree on a premaster secret.

- Exchange certificates and cryptographic information to allow the client and server to authenticate themselves.

- Generate a master secret from the premaster secret and exchanged random values.

- Provide security parameters to the record layer.

- Allow the client and server to verify that their peer has calculated the same security parameters and that the handshake occurred without tampering by an attacker.

Fig. 13 (RFC 5246, page 34, describing exchange of cryptographic parameters [system parameters])

These goals are achieved by the handshake protocol, which can be summarized as follows: The 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. The ClientHello and ServerHello are used to establish security enhancement capabilities between client and server. 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.

Fig. 14 (RFC 5246, page 34, describing establishment of protocol version, session id, cipher suite, and compression method [comprising, inter alia*, system parameters])*

49. The Accused Instrumentalities comprised the user's system running a key generation algorithm to get a private key and public key.

50. RFC 5246 states the following. (Fig. 15 to Fig. 19)

RFC 5246 TLS August 2008

The TLS Handshake Protocol involves the following steps:

- Exchange hello messages to agree on algorithms, exchange random values, and check for session resumption.
- Exchange the necessary cryptographic parameters to allow the client and server to agree on a premaster secret.
- Exchange certificates and cryptographic information to allow the client and server to authenticate themselves.
- Generate a master secret from the premaster secret and exchanged random values.
- Provide security parameters to the record layer.
- Allow the client and server to verify that their peer has calculated the same security parameters and that the handshake occurred without tampering by an attacker.

Fig. 15 (RFC 5246, page 34, describing that the server generates a master secret [private key] from a premaster secret [public key] and exchanged random values)

7.4.3. Server Key Exchange Message

When this message will be sent:

This message will be sent immediately after the server Certificate message (or the ServerHello message, if this is an anonymous negotiation).

Fig. 16

Meaning of this message:

This message 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.

Fig. 17 (RFC 5246, pages 50-51, describing the use of a premaster secret [public key] during the TLS handshake)

8.1. Computing the Master Secret

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

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

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

Fig. 18 (RFC 5246, page 64, describing how the server generates the master secret [private key])

5.10. ECDH, ECDSA, and RSA Computations

All ECDH calculations (including parameter and key generation as well as the shared secret calculation) are performed according to [6] using the ECKAS-DH1 scheme with the identity map as key derivation function (KDF), so that the premaster secret is the x-coordinate of the ECDH shared secret elliptic curve point represented as an octet string. Note that this octet string (Z in IEEE 1363 terminology) as output by FE2OSP, the Field Element to Octet String Conversion Primitive, has constant length for any given field; leading zeros found in this octet string MUST NOT be truncated.

Fig. 19 (RFC 4492, page 26, describing that the premaster secret [public key] is generated using a negotiated elliptic curve)

51. The Accused Instrumentalities comprised the step of the user's system constructing a proof that the private key was generated properly using the system parameters, and where said constructing of said proof requires access to said private key and verification of said proof can be performed by any other entity with no access to any portion of said private key.

52. RFC 5246 states the following. (Fig. 20)

```
7.4.9.  Finished

   When this message will be sent:

      A Finished message is always sent immediately after a change
      cipher spec message to verify that the key exchange and
      authentication processes were successful.  It is essential that a
      change cipher spec message be received between the other handshake
      messages and the Finished message.

   Meaning of this message:

      The Finished message is the first one protected with the just
      negotiated algorithms, keys, and secrets.  Recipients of Finished
      messages MUST verify that the contents are correct.  Once a side
      has sent its Finished message and received and validated the
      Finished message from its peer, it may begin to send and receive
      application data over the connection.

   Structure of this message:

      struct {
          opaque verify_data[verify_data_length];
      } Finished;

      verify_data
         PRF(master_secret, finished_label, Hash(handshake_messages))
            [0..verify_data_length-1];

      finished_label
         For Finished messages sent by the client, the string
         "client finished".  For Finished messages sent by the server,
         the string "server finished".
```

Fig. 20 (RFC 5246, page 63, describing the use of a Finished message [proof] in the TLS handshake, where the contents of Finished message is generated using the master secret [private key])

53. As shown in ¶¶ 52, the Finished message from the server must be validated by the client before application data is sent over the established TLS 1.2 connection. Because the contents of the Finished message[proof] is a function of the master_secret [private key](which is derived from the Diffie-Hellman premaster secret) and a function of the preceding handshake messages, validating the Finished message verifies that the relevant keys, including the premaster secret and the master secret, were generated properly by the specified algorithm in use

(e.g., TLS 1.2 ECDH). Instead of receiving the server's generated master_secret, the client generates a master_secret independently based on the premaster secret and based on the previously exchanged client/server random values to validate the Finished message from the server.

54. RFC 5246 states the following. (Fig. 21, Fig. 22)

7.4.9. Finished

When this message will be sent:

A Finished message is always sent immediately after a change cipher spec message to verify that the key exchange and authentication processes were successful. It is essential that a change cipher spec message be received between the other handshake messages and the Finished message.

Meaning of this message:

The Finished message is the first one protected with the just negotiated algorithms, keys, and secrets. Recipients of Finished messages MUST verify that the contents are correct. Once a side has sent its Finished message and received and validated the Finished message from its peer, it may begin to send and receive application data over the connection.

Fig. 21 (RFC 5246, page 63, describing verification of the Finished message [proof] before communicating application data over the connection)

F.1.3. Detecting Attacks Against the Handshake Protocol

An attacker might try to influence the handshake exchange to make the parties select different encryption algorithms than they would normally choose.

For this attack, an attacker must actively change one or more handshake messages. If this occurs, the client and server will compute different values for the handshake message hashes. As a result, the parties will not accept each others' Finished messages. Without the master_secret, the attacker cannot repair the Finished messages, so the attack will be discovered.

Fig. 22 (RFC 5246, page 94, describing that because the server does not send its generated master secret [private key] to the client, and vice versa, an attacker does not possess the master secret [private key])

55. Plaintiff has been harmed by Defendant's infringing activities.

JURY DEMAND

Pursuant to Rule 38 of the Federal Rules of Civil Procedure, Plaintiff demands a trial by jury on all issues triable as such.

PRAYER FOR RELIEF

WHEREFORE, Plaintiff demands judgment for itself and against each Defendant as follows:

A. An adjudication that Defendant has infringed the '150 patent.

B. An award of damages to be paid by Defendant adequate to compensate Plaintiff for Defendant's past infringement of the '150 patent, including interest, costs, expenses and an accounting of all infringing acts including, but not limited to, those acts not presented at trial;

C. A declaration that this case is exceptional under 35 U.S.C. § 285, and an award of Plaintiff's reasonable attorneys' fees; and

D. An award to Plaintiff of such further relief at law or in equity as the Court deems just and proper.

Dated: June 13, 2017

DEVLIN LAW FIRM LLC

/s/ Timothy Devlin

Timothy Devlin (#4241)
tdevlin@devlinlawfirm.com
1306 N. Broom St., 1st Floor
Wilmington, Delaware 19806

Telephone: (302) 449-9010
Facsimile: (302) 353-4251

Attorneys for Plaintiff CryptoPeak Security, LLC