SSL and TLS

An Overview of A Secure Communications Protocol

Simon Horman aka Horms

horms@valinux.co.jp
horms@verge.net.au
horms@debian.org

Presented at the Security Mini-Conf at Linux.Conf.Au
Canberra, ACT, Australia. April 2005

Abstract

SSL/TLS is widely used to securely send data over the Internet, however it is not a magic solution, and without an understanding of how the protocol works and how the underlying technologies it is at best difficult to fully utilise SSL/TLS and at worst easy to use SSL/TLS in an insecure manner.

This presentation will explain how SSL/TLS work, from a high level protocol discussion of data integrity, confidentiality and endpoint verification, to a low level discussion of the different messages that make up the SSL/TLS protocol and the encryption techniques that ultimately secure the connection.

The intended audience will be interested in using or developing applications that make use of SSL/TLS to secure data transfers.

Contents

1 Secure Communication

The purpose of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols is to provide a mechanism for secure communications between two parties over a network which neither party has end-to-end control over and thus has the potential for third parties to intercept communication. The Internet is a good example of such a network. Fundamentally there are two aspects that need to be addressed, data integrity and end-point verification.

1.1 Data Integrity

When communication is made between two parties in a secure manner it is important that the data is received in its entirety, unmodified and without other parties being able to inspect or modify the communication. To provide data integrity SSL/TLS employs a variety of cryptographic techniques. Asymmetric and symmetric encryption is used to provide privacy by preventing third-parties from being able to access the contents of a message even if it is intercepted. This also provides protection against messages being removed and inserted. Message digests are used to protect against messages being modified.

1.1.1 Asymmetric Encryption

Asymmetric Encryption, also commonly referred to as public key encryption, allows encrypted communication between two parties without the need for prior negotiation of secret keys. It is referred to as asymmetric encryption as different keys are used for encryption and decryption. Arguably the most well known Asymmetric Encryption algorithm is RSA[11],[24].

RSA keys have two parts, a public key and a private key. As the names imply, the public key is freely made available to any interested parties. While the private key is kept secure. The private key can be used to sign messages, which can be verified using the public key. That is only the holder of the private key can sign messages, however anyone who has access to the public key can verify these messages. Conversely, the public key can be used to encrypt messages that can only be decrypted by the private key.

The strength of the RSA algorithm lies in the product of two very long prime numbers which is thought to be very difficult to factorise. The longer the primes, the stronger the key. However, the length of these primes, and the resulting key leads to very slow processing of operations. Other asymmetric encryption algorithms are similarly slow, and thus it is not suitable for bulk data transfer. Because of this, SSL and TLS make use of asymmetric encryption for verification and to negotiate a secret key that will be used for symmetric encryption of bulk data transfers. In this way, the convenience of asymmetric encryption is used to allow communication without prior negotiation of keys, but its slowness is largely mitigated.

1.1.2 Symmetric Encryption

Symmetric Encryption allows encrypted communication between two endpoints using a shared key. It is called symmetric encryption because the same key is used for both encryption and decryption. Commonly used symmetric encryption algorithms are DES, its successor DES3 and more recently AES.

Because the same key is used for both encryption and decryption is important that this key is shared between the two endpoints in a secure manner. If it becomes known to a third-party, then that party can both create bogus encrypted messages, and decrypt intercepted messages. On private networks where end-points are well controlled, for instance an ATM or POS unit which communicates with a bank, this is not much of a problem as long as the key is supplied with the equipment. However for more ad-hoc communication, as provided by SSL/TLS, trust relationships need to be set up on the fly, and thus keys for symmetric encryption are negotiated using asymmetric communication.

Symmetric encryption algorithms tend to make use of much shorter keys that asymmetric encryption that is thought to offer similar levels of security. This lends itself to much faster implementation in software, which is important as at this time most SSL/TLS processing is done in software.

1.1.3 Blocking

Symmetric encryption algorithms tend to be block-based. That is they take a fixed amount of data and encrypt or decrypt it. However SSL/TLS provides stream based data transfer, and the amount of data usually does not match the blocking size of the encryption algorithm being used. In any case, the data being transfered is independent of the algorithms being used, so blocking is needed to divide up data into blocks and pad the data as necessary.

A naive blocking implementation is to simply divide the data into block-size blocks, pad as necessary and encrypt the resulting blocks. This approach is called Electronic Code Book (ECB) mode[7]. However its main drawback is that if the plain-text for two blocks is the same, the cipher text will be the same. This information may be useful for an attacker trying to decrypt the stream.

For this reason SSL and TLS make use of Cipher Block Chaining (CBC) mode[7]. When a stream of data is transfered, the the plain-text of the first block is exclusive-ored with an Initialisation Vector (IV) before encryption. In SSL/TLS, the IV is produced as part of key negotiation during the handshake and is different for each connection. Subsequent blocks are exclusive-ored with the previous block's cipher-text before being encrypted. In this way each block after the first block is dependant on the previous block. So even if the plain-text of two blocks in a stream is the same, the cipher-text will differ.

1.1.4 Message Authentication Codes

A Message Authentication Codes (MAC) are used to ensure that messages are not tampered with or otherwise corrupted during transit. This can be thought of as a digest of the message which includes a secret key. It is constructed when data is sent, and verified when it is received. It is not possible to reproduce the digest without knowing both the input text and the key, and thus a would-be attacker needs to know the secret in order to construct a valid MAC for a message that has been altered. For protection against replay attacks SSL and TLS include a monotonically increasing serial number is included in the input to the MAC.

1.2 Endpoint Verification

It is also important that when communication is made between two endpoints the endpoints are indeed who they claim that they are. In SSL and TLS this is achieved using certificates. During the course of establishing an SSL/TLS connection a message signed with the end-point's certificate is sent along with the certificate. The certificate itself is signed by a certificate authority, and it is in the certificate authority that the web of trust lies.

SSL/TLS is a client-server protocol. This maps quite well to its use with higher-level client-server protocols such as HTTPS, POP3S and IMAP4S. When used in this manner typically only the authenticity of the server is verified by the client. That is, the web or mail server will have a certificate but the client typically will not. However, the SSL and TLS allow for verification of the client and at the request of the server. Which can be useful for controlling access to services. It is also possible for anonymous key exchange to be used, in which case the server does not send a certificate to identify itself. This offers no security against an attacker setting up a bogus server.

To verify the validity of a certificate several checks are made. More rudimentary checks include checking that the certificate has not expired and checking that it's common name matches the hostname connected to. The certificate is also verified cryptographicly by checking to make sure it is signed by a known certificate authority. It is up to the software making the check to supply a list of known certificate authorities. In the case of HTTPS, a list of know certificate authorities is typically included with the web-browser, and may be added or removed after installation. It is also reasonably common for servers to use self-signed certificates, in which case it is up to the individual accessing the site to assess its validity manually, for instance by checking the fingerprint off-line. Though more commonly by just hopping that it is ok.

2 Versions

The first released version of SSL was SSLv2[9], released by Netscape Communications. However it has a number of weaknesses as summarised below.

- MAC is weak and relies on MD5
- Same keys are used for authentication and encryption. This makes the MAC unnecessarily weak with export ciphers.
- Uses TCP for closure which means a truncation attack can be effected if a TCP SYN packet can be spoofed.
- Handshake is not protected from man in the middle attack.

The issues with SSLv2 were addressed by SSLv3[1] which was also released by Netscape Communications. This was subsequently adopted by the IETF and standardised as TLSv1[4]. The main differences between SSLv2 and TLSv1 are as follows.

- Improvement of expansion of keys from the master secret which is calculated using data exchanged during the handshake.
- SSLv3 is based on an early revision of HMAC[8], TLSv1 makes use of HMAC itself.
- Implementations are required to support DH/DSS key exchange and Triple-DES.

3 Record Layer

The record layer encapsulates messages for transmission over the underlying communications protocol, usually TCP/IP. Each record can include up to 2^{14} bytes of data and messages are fragmented as necessary to meet this size limit. A record may also contain multiple messages, as long as they are of the same type. This frequently occurs during the handshake, where multiple messages are included in a single record, in the hope that they will be transmitted in a single packet, and increase the speed of the handshake.

content type	version	length

Record Header

A record begins with a header which includes the version of the protocol, the length of the data in bytes and the type of the message, change cipher spec, alert, handshake or application data.

After the header comes the message data. This is compressed by the compression algorithm that has been negotiated for the connection. However as no compression algorithms are specified for use with TLS or SSL, this is usually a null operation.

The MAC is then calculated for the compressed data and appended to the record. Calculation of the MAC is discussed in the following section.

pad bytes	pad length

Record Pad

If a block cipher is in effect for the connection then then a pad is added in order for the message size to be a multiple of the block size of the cipher. The pad consists of padding followed by a byte containing the length of the padding. The padding itself is also set to the value of the length of the padding. The padding may be any length up to 255 bytes, so long as the length of the resulting record is a multiple of the block length of the cipher.

As an example, if a record is 123 bytes long and the block length is 8 bytes, then a pad length of 4 would make the combined length of the record, padding and pad length 128 bytes, a multiple of the block length. 12, 20 and 252 are examples of other valid pad lengths. The intention of allowing a range of pad values is to allow the length of the record to be obscured to some extent.

4 pad byte 1	4 pad byte 2	4 pad byte 3	4 pad byte 4	4 pad length

Record Pad

If the NULL cipher or a stream cipher is in effect then padding is not needed. In practice this occurs during the initial handshake which is conducted in plain-text and if RC4[23, 12] are used.

3.1 Message Authentication Code

The Message Authentication Code (MAC) used for TLS is HMAC and for SSLv3 it is an earlier draft of HMAC, hence-forth referred to as as SSL3-MAC[10].

HMAC is expressed by the following equation,

$$H(K \oplus opad, H(K \oplus ipad, text))$$

Where:

$\oplus$:	is concatenation
text:	is the plain-text to be encrypted
H:	is the hashing function
B:	is the block length of H in bytes
K:	is a key up to B bytes long
ipad:	is the byte 0x36 repeated B times
opad:	is the byte 0x5C repeated B times

The hashing function is used as a black box, and in this way a wide range of message digest algorithms may be used. However, all non-NULL cipher-suites use MD5[20] or SHA1[3, 16].

The key used is computed as part of the key negotiation that take place during the handshake, and as such is secret. This is important, because it is thought that as long as the key is kept secret, the integrity of the MAC is maintained, even if the underlying hash algorithm is relatively weak[14, 15]. In a nutshell this is because any attack on the hash would have to be permuted over the possible keys. This is particularly important as both MD5 and SHA1 are thought to be broken[5, 22, 25]. Meaning that collisions are more frequent than the $2^{B/2}$ that the birthday principle[13, 21], where a collision occurs when the digest of two messages is the same.

It is thought that SSL3-MAC has much the same properties as HMAC, as is is an earlier version of HMAC, with the difference that the pad strings, ipad and opad are appended to the key, rather than being exclusive-ored with it. However it has not received nearly as much cryptographic analysis as HMAC.

In addition to the key, SSL and TLS includes a monotonically increasing sequence number in the text that the MAC runs over, this is used to prevent replay attacks. There is a separate sequence number for records sent in each direction over the SSL connection. And the sequence numbers are updated by the sender and receiver independently as messages are sent and received.

The MAC is applied to both the head and the data of the record, after the data is compressed, though as no compression algorithms are specified in the SSL or TLS standards, compression is usually skipped. The MAC is then encrypted along with the rest of the record.

4 Messages

Messages are the basic unit of communication in SSL and TLS. There are four different types of messages. Handshake, change cipher suite and alert messages are control messages and application data messages transfer data between applications running at either end of the connection.

4.1 Handshake

type	length

Handshake Message Header

Handshake messages start with a common body that denotes the type of handshake and the length of the message. The body contains one of the following handshake messages: hello request, client hello, server hello, certificate, server key exchange, certificate request, server hello done, certificate verify, client key exchange, finished. The use of handshake messages to negotiate the parameters for a connection is discussed in the section on connection sequences.

4.1.1 Hello Request

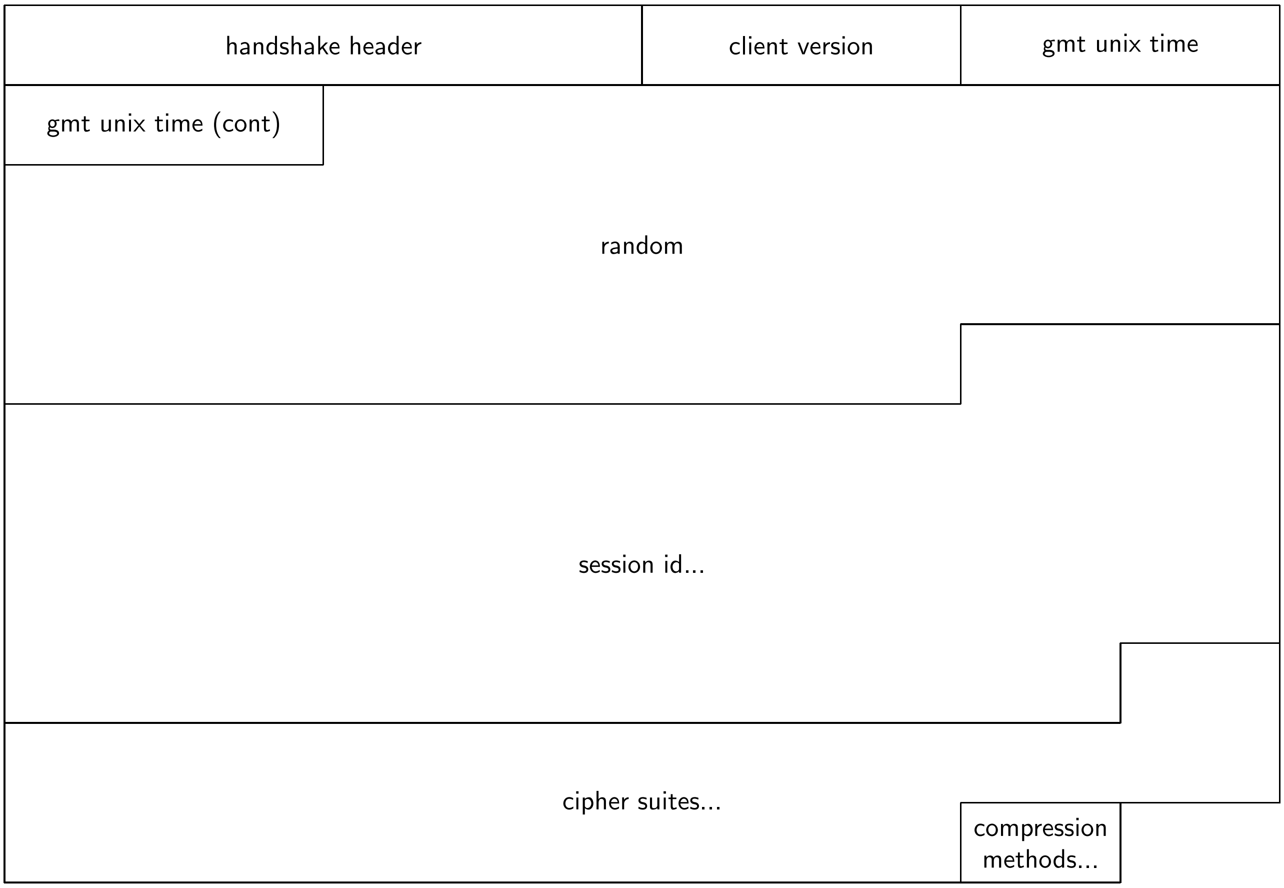
The body of this message is empty. It may be sent by the server at any time to request that the client begin a handshake negotiation sequence. The client will ignore this message if negotiation is already in progress, and may also ignore this message if it does not want to renegotiate the session for any reason. If negotiation is not desired by the client it may optionally send a no renegotiation alert.

This message is not included in the message hash which is updated for all other handshake messages and included in the finished handshake message.

This message is not required when a client initially connects to SSL or TLS as a handshake is always initiated. Rather, it is used to allow the server to request renegotiation of a session. This is typically done to renegotiate keys for long-lived connections.

If the client wishes to initiate renegotiation it sends a client hello message, which like the hello request may be ignored by the server or responded to with a *no_renegotiation* alert.

4.1.2 Client Hello



Client Hello Message

The client hello message is sent when a client connects to a server to initiate the handshake. After the common handshake header is the client version, which specifies the highest version of the protocol that the client supports. SSL and TLS implementations need to support all previous versions of the protocol as well. Then comes a random number which consists of the current unix GMT time-stamp and 28 bytes generated by a cryptographicly secure pseudo number generator. Together these values are used in the generation of the keying material. There is no requirement in the SSL or TLS specifications for the clocks of the client and server to be synchronised.

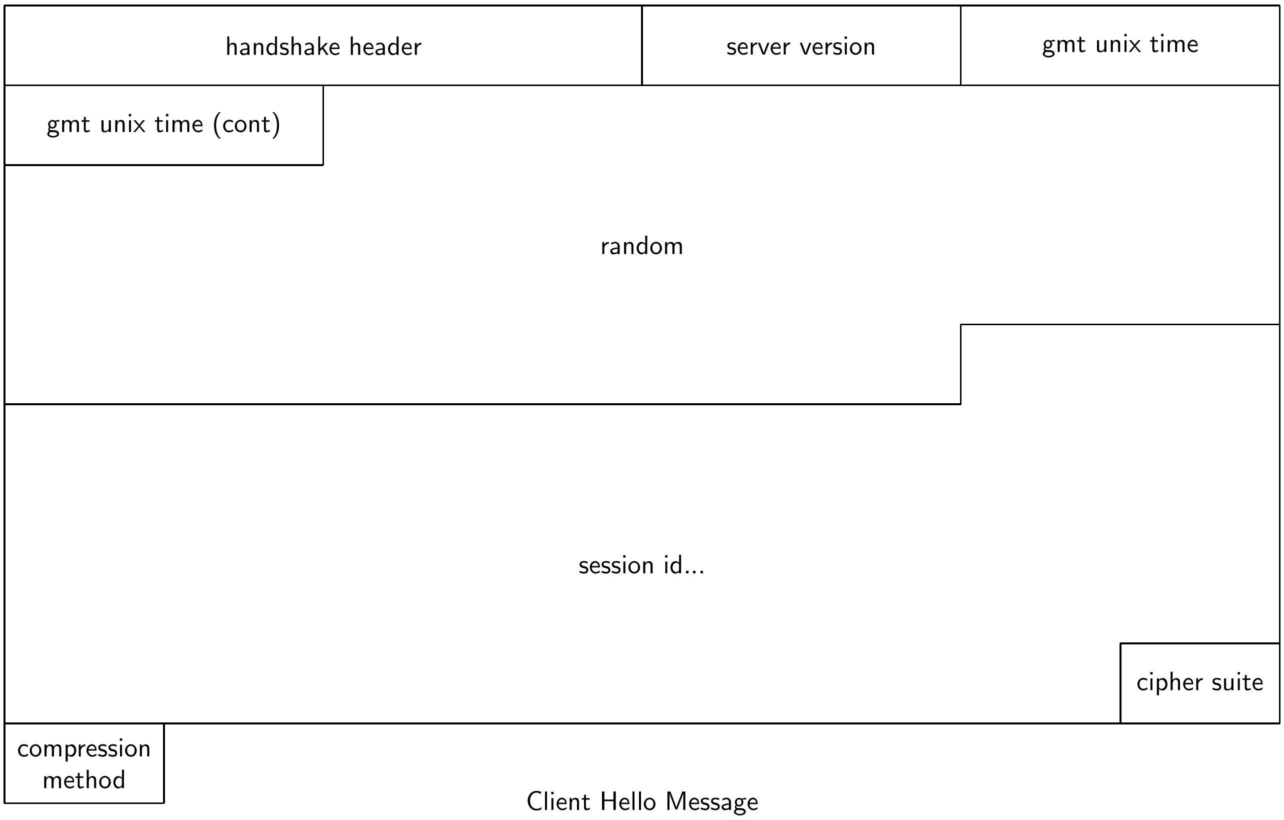
After the random number comes the session id. If supplied, this can be used to perform an abbreviated handshake sequence by reusing key material that was previously negotiated between the client and the server.

Next comes the list of cipher suites that the client is prepared to use. Cipher suites define the encryption and hashing functions that will be used by the connection once the handshake concludes as well as the key-exchange method used during the handshake. They are represented by two 8bit numbers which is documented in the SSL or TLS specification or by some additional specification. For example TLS_RSA_WITH_RC4_128_SHA is a cipher suite that uses RSA for key exchange, RC4 as the bulk encryption algorithm, SHA1 as the hashing function for the MAC and is identified as 0x00,0x05. Different cipher suites offer different levels of security, and thus the cipher suite used has a critical effect on the level of security for a given connection.

The server is free to choose any of the listed cipher suites, and if the server does not support any off the offered cipher suites then it send a handshake failure alert and closes the connection. Although implementations typically order them from most preferred to least preferred, as the server may choose any of the listed cipher suites, it is advisable only to list ciphers that the user feels offer a satisfactory level of security.

Compression methods are represented by a single 8bit number, and as per the cipher suites, the client lists all the compression methods it is willing to use. However, the only compression method defined is NULL, identified as 0x00, so this is usually the only compression method offered by clients.

4.1.3 Server Hello



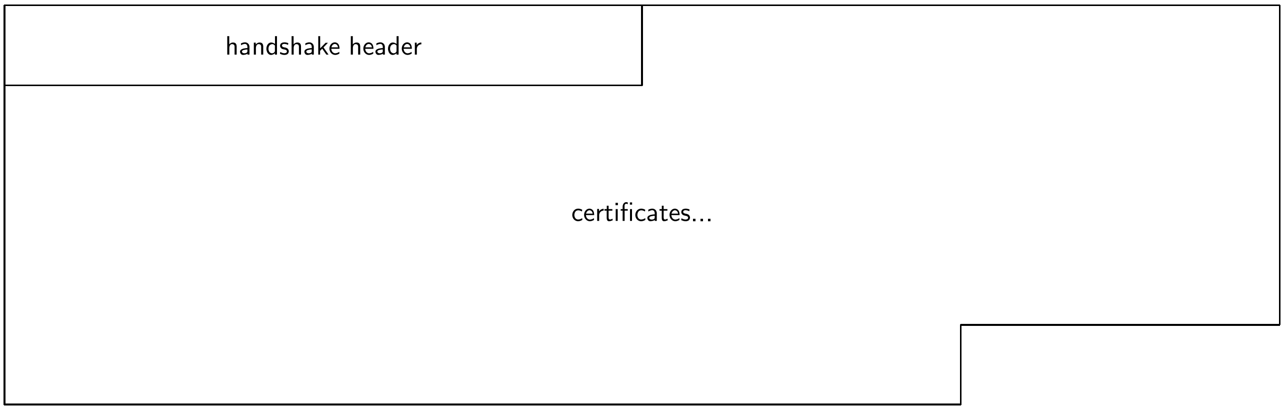
Client Hello Message

The server hello is sent by the server in response to a client hello. The server version is the client version if it is supported by the server, else the highest version supported by the server. Then there is a random number consisting of the unix GMT time-stamp and 28 bytes generated by a cryptographic random number generator.

Next comes the optional session id. If this is the same as the session id sent by the client in the client hello then an abbreviated handshake will be performed using key material cached by both the client and the server. If empty it indicates that the server is not willing to perform an abbreviated key handshake. Otherwise it is the id that the client should use to request an abbreviated key handshake in the future, the client may choose to ignore this. And the server may choose to ignore the session id sent by the client, for instance because the key material is no longer cached.

Finally the message includes the cipher suite and compression method selected by the server from the lists provided in the client hello.

4.1.4 Server Certificate



Server Certificate and Client Certificate Message

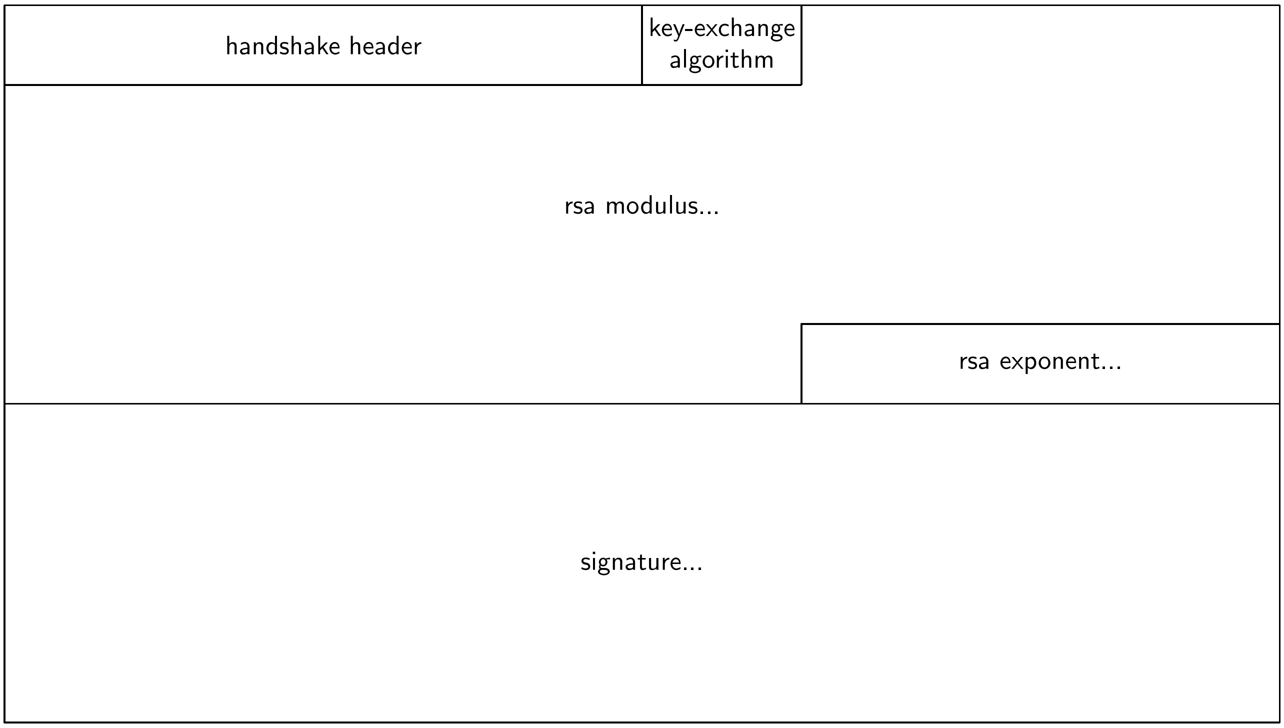
This is sent by the server after sending a server hello if the cipher suite selected specifies a non-anonymous key exchange method, which is typically the case. It consists of an ASN.1[6] encoded sequence of X.509v3[17] certificates starting with the server's certificate.

4.1.5 Server Key Exchange

The server key exchange message is sent after the server certificate message if it does not contain enough information for the client to exchange the pre-master secret, or after the server hello if anonymous key exchange is in use. More specifically it is used for anonymous Diffie-Hellman, Ephemeral Diffie-Hellman and Ephemeral RSA key-exchange methods.

Ephemeral and anonymous Diffie-Hellman[18] is determined by the chosen cipher suite. Ephemeral RSA is used when an RSA export cipher suite is used and the public key is longer than 512 bits. Export cipher suites[2] are designed to satisfy the requirements of now relaxed US export restrictions that amongst other things restricted RSA keys to 512 bits. The idea of Ephemeral RSA is to allow a server to have a strong key for key-exchange with non-export cipher suites, while a temporary key of 512 bits or less is used for key-exchange with export cipher suites. It also allows the server to have a certificate that is somewhat less vulnerable to attack. This certificate is used for both export and non-export cipher suites.

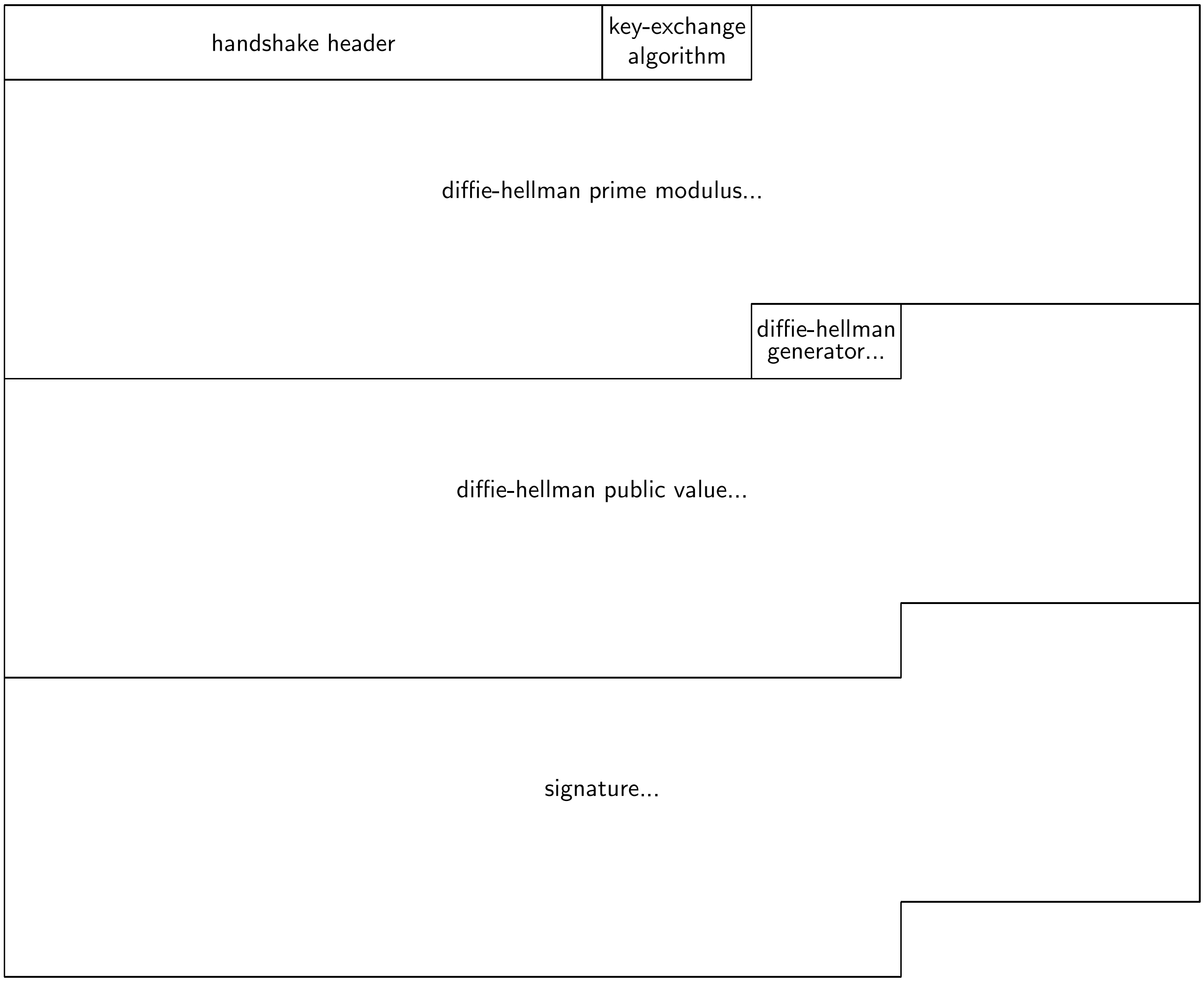
This message has two forms, one for RSA and one for Diffie-Hellman. Both begin with the handshake header followed by the key-exchange algorithm, RSA or Diffie-Hellman.



RSA Server Key Exchange Message

The RSA version of this message includes the RSA modulus and exponent. The signature depends on the type of the server's certificate and the type of key-exchange being performed. For RSA the signature consists of the concatenation of an MD5 and an SHA1 hash, signed by the server's key. For DSS it consists of an SHA1 hash signed by the server's key. The input for the hash is the concatenation of the client random, the server random, and the message itself from the end of the handshake to the beginning of the signature.

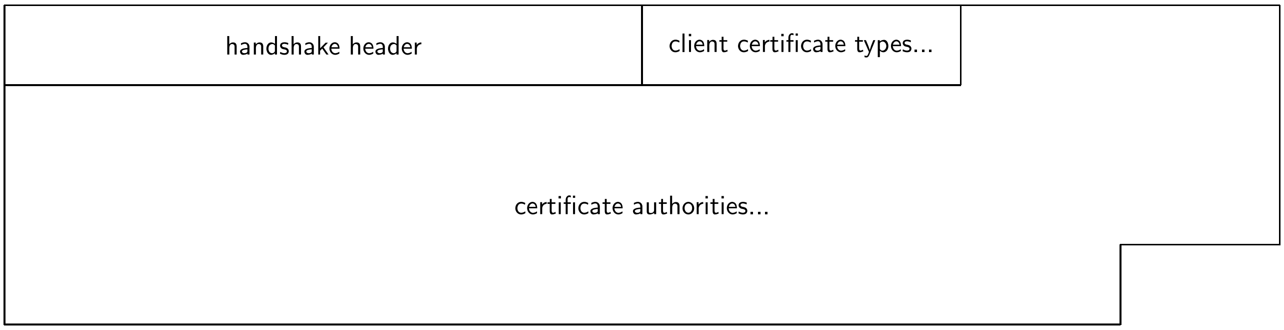
The signature is omitted if anonymous key exchange is being performed. It should be noted that in this case there is no way to verify the identity of the server and thus an attack involving establishing a bogus server may easily be performed.



Diffie-Hellman Server Key Exchange Message

The Diffie-Hellman version of this message includes the Diffie-Hellman prime modulus, generator and public value. This is followed by a signature as described above.

4.1.6 Certificate Request



RSA Client Key Exchange Message

This message is sent by the server to request that the client identify itself. It starts with the handshake header, followed by a list of acceptable certificate types. The defined certificate types are rsa sign, dsa sign, rsa fixed dh and dss fixed dh Finally there is a list of of acceptable certificate authorities in X.509 format. This list is not used for Diffie-Hellman (dh) certificates, as the verification lies in the client's ability to generate the same pre-master secret.

This message is sent after the server key exchange message if it is sent, otherwise after the server certificate message. It is not legal for this message to be sent if anonymous key-exchange is being used.

4.1.7 Server Hello Done

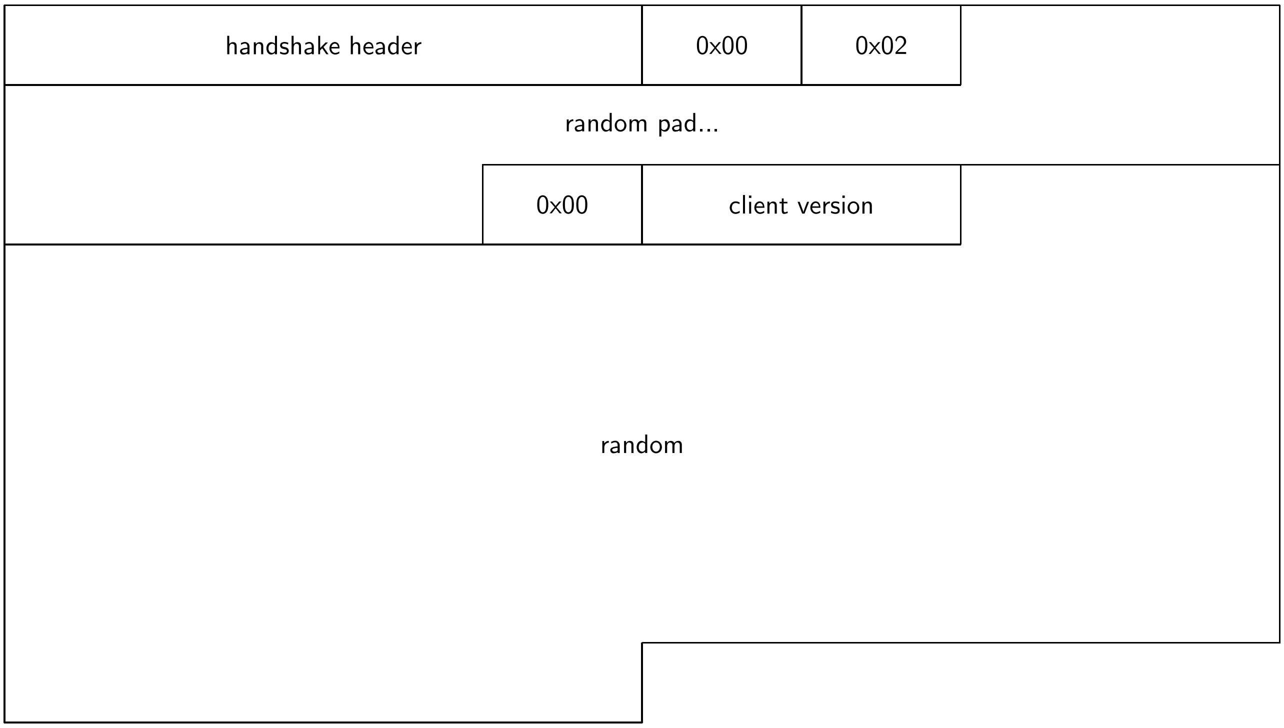
This message is sent by the server to indicate that that it has sending its hello and associated messages. That is, it does not intend to send any of the following messages: server hello, server certificate, certificate request. The message does not contain any fields other than the handshake header.

4.1.8 Client Certificate

This is sent immediately after receipt of a server hello done message if a certificate request message was received. Like the server certificate it consists of an ASN.1 encoded sequence of sequence of X.509v3 certificates starting with the client's certificate.

4.1.9 Client Key Exchange

This message is send immediately after the client certificate if it is sent, otherwise immediately after the server hello done message. It has two variants, one for when RSA is used as the key-exchange algorithm, and one for when Diffie-Hellman is used.



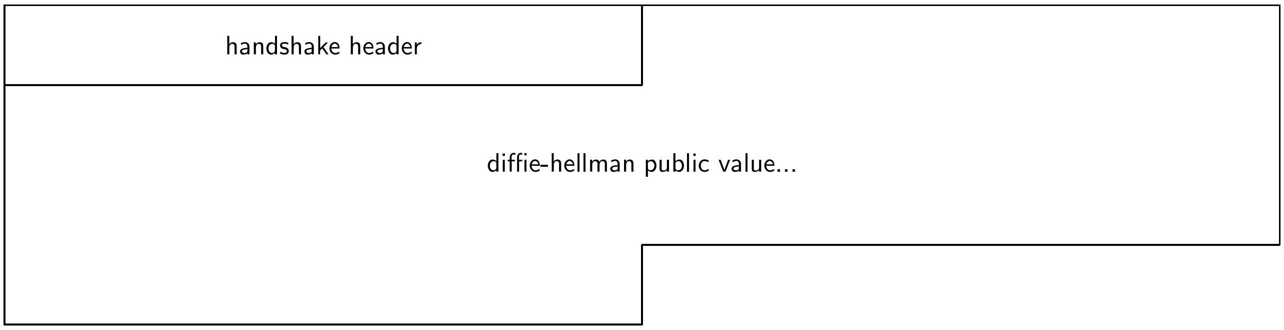
RSA Client Key Exchange Message

When RSA is used the client key exchange message consists of the header handshake followed by PKCS #1[11] encoded pre-master secret that is encrypted using the server's key, as sent in the server certificate message. The handshake header is not encrypted.

The pre-master secret consists of the maximum version supported by the client followed by 48 random bytes generated by a cryptographic random number generator. Combined these 48 bytes provide the client's input into the master secret from which the session keys will be derived. The inclusion of the version is intended to prevent version roll-back by a man-in-the-middle attack. This may indeed work for TLS but in SSLv3 implementations use of the negotiated version is common and thus checking this value creates incompatibility problems[19].

The PKCS #1 encoding is needed to pad the pre-master secret to the length of the RSA key. It consists of 0x00, 0x02, randomly generated pad bytes, 0x00 and finally the message, in this case the pre-master secret. The number of pad bytes is chosen to make the resulting message the same length as the key. The number of pad bytes must be at least 8, so the minimum pad length is 11 bytes. In theory fragmentation of the message may be necessary, but in practice RSA keys are longer than 59 bytes – the length of the pre-master secret combined with the minimum pad – so this is not done.

As the message is encrypted with the server's public key, only the holder of the private key can decrypt the message. This means that although the certificate may be sent by any party, only servers that hold the private key can successfully complete this part of the handshake. This also protects the version number from a man-in-the-middle attack, as the random bytes cannot be recovered from the message for inclusion in a modified message, and if they don't match the key material will not match on both sides and the handshake will fail when when the finished messages are processed. However as discussed above, this protection is largely useless for SSLv3.

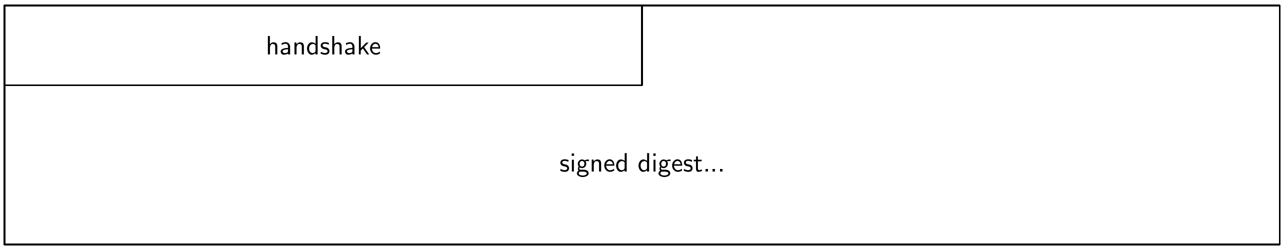


Diffie-Hellman Client Key Exchange Message

When Diffie-Hellman is used the client key exchange message consists of the handshake header followed by the Diffie-Hellman public value. This allows both the client and server to calculate the same pre-master secret.

In the case where client authentication is used, and the client certificate that was sent contains a Diffie-Hellman public that uses the parameters specified by the server, then the public value is omitted from this message as it is already known. In this case the handshake header comprises the message entirely.

4.1.10 Certificate Verify



Certificate Verify

The client verify message is used to indicate that the client is indeed the holder of the client certificate, and it is sent immediately after the client certificate message if that message was sent. It consists of a signed digest of all the handshake messages sent and received thus far. Only the holder of the certificate's private key can sign a message, and it is in this that lies the client's verification of its certificate.

For TLSv1, the format of the signature is as per the server key exchange message. For RSA keys it is the signed concatenation of MD5 and SHA1 hashes of the digest. For DSA[**?**] keys it is a signed SHA1 hash of the digest. For anonymous key exchange, the message is not sent.

For SSLv3 the the hash calculations is expressed as follows.

$$H(master\_secret \oplus pad2 \oplus H(digest \oplus master\_secret \oplus pad1))$$

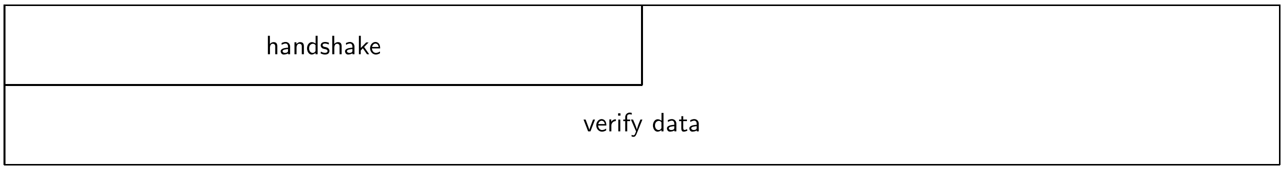
Where:

$\oplus$:	is concatenation
H:	is the hashing function, SHA1 or MD5
pad1:	is the byte 0x36 repeated 48 times for MD5 or 40 times for SHA1
pad2:	is the byte 0x5C repeated 48 times for MD5 or 40 times for SHA1

Like TLS, the MD5 and SHA1 hashes are concatenated for RSA keys, only an SHA1 hash is used for DSS keys and the message is not used for anonymous key-exchange.

4.1.11 Finished

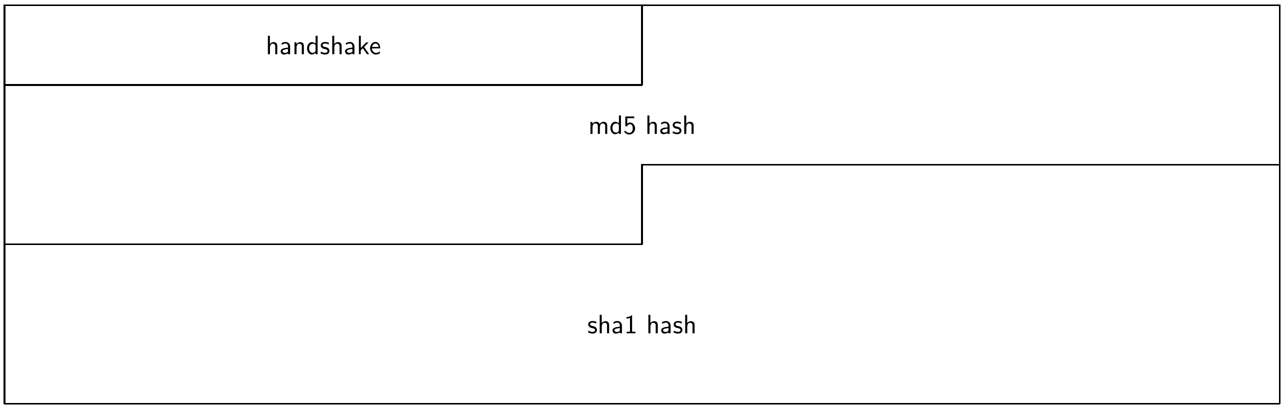
The finished message is sent by both the client and server after a change cipher spec. It is used to verify that the handshake was not tampered with. On receipt it is recalculated locally and if the received and local values do not match, then the handshake has been somehow interfered with. It is the first message to be encrypted with the negotiated cipher suite and keys.



TLSv1 Finished Message

For TLSv1, the verify data is generated by the PRF is used to produce 12 bytes of output. The PRF is discussed in detailed in the section on key material generation.

The call to the PRF uses the master secret as the secret and the text is the concatenation of a label — "server" for the server and "client" for the client — and, an MD5 and SHA1 sum of the handshake messages received up to, but not including this message since the beginning of the connection. This includes any handshake messages used in prior handshakes if renegotiation is occurring.



SSLv3 Finished Message

For SSLv3 a nested hash more closely resembling HMAC is used. It also uses the master secret, the handshake messages and sender and client identifiers. A separate HMAC-like hash is produced using both MD5 and SHA1 and these are concatenated to form the verify data. The client label is 0x434C4E54 and the server label is 0x53535652.

The purpose of the labels used in both SSLv3 and TLSv1, are so that the client and server finished messages will differ, preventing a a man-in-the-middle attach from simply reflecting a finished message back to the sender.

4.2 Change Cipher Spec

change_cipher_spec

Change Cipher Spec

A change cipher spec is issued to notify the other end of the connection that the connection parameters negotiated thus for will take effect from the message immediately following the change cipher spec. In other words, it brings the cipher suite into effect. It is sent immediately after a handshake concludes and before any application data messages are sent. It only effects the direction of the connection in which it was sent and thus must be sent by both ends of the connection.

4.3 Alert

level	description

Alert Message

Alerts may be issued at any time by either end of the connection when the connection is to be closed or an error occurs. The alert consists of a level, warning or fatal, and description. Both of these values are 8bit integer entities. The description is either close notify or the description of some error state, such as unexpected message or bad record mac. A full list of error descriptions is included in appendix A.

4.4 Application Data

Application data messages are used to transfer data between the two endpoints of the connection. It does not have a header, rather the message consists solely of the data itself, which is subsequently encapsulated in a record.

5 Key Material Generation

Key material generation is used to generate the secret material that will be used as keys and input vectors to encrypt and verify records. The inputs to key generation are the client and server random, and the pre-master secret, sent in the client and server hello, and client key exchange messages respectively. As the pre-master secret is sent to the server, encrypted with the key in its certificate, it should only be known by the client and server.

5.1 TLSv1

5.1.1 Pseudo Random Function (PRF)

In TLS, a Pseudo Random Function (PRF) is at the heart of key material generation. It is also used in the finished handshake message. The PRF begins with an expansion function that takes a key and some text and produces arbitrary amounts of out put. It is defined as follows:

$$
\begin{aligned}
P\_H(secret, text) &= HMAC\_H(K, A(1), T) \oplus \\
&\quad HMAC\_H(K, A(2), T) \oplus \\
&\quad HMAC\_H(K, A(3), T) \oplus ...
\end{aligned}
$$

Where:

$\oplus$:	concatenation
A():	$A(0) = text$
	$A(n) = HMAC\_H(secret, A(n-1))$
H:	hashing function, SHA1 or MD5
K:	key
T:	text

This is run as many times as desired. So to produce 48 bytes of output, it would be run 3 times for MD5, and for SHA1 it would also be run 3 times and the last 12 bytes of output would be discarded.

The PRF is comprised of the PRF run with MD5 exclusive-ored with the PRF run with SHA1 as follows:

$$PRF(secret, text) = P\_MD5(S1, text) \otimes P\_SHA1(S2, text)$$

Where:

$\otimes$:	exclusive or
S1:	is the first half of the secret bitwise
S2:	is the second half of the secret bitwise

5.1.2 Master Secret

The client and server independently use the the PRF is used to calculate the 48 byte master secret. Which is in turn converted into the key block. The key block is as many bytes long as is needed and is divided up into MAC secrets, symmetric encryption keys, and input vectors for blocking used with symmetric encryption. These are, in order, the client write MAC secret, server write MAC secret, client write key, server write key, client write IV and server write IV. Unneeded values are omitted. The master secret and key block are computed using the PRF as follows:

$$
\begin{aligned}
X(secret, label) &= PRF(secret, label \oplus client\_random \oplus server\_random) \\
master\_secret &= X(pre\_master\_secret, \text{“master secret''}) \\
key\_block &= X(master\_secret, \text{“key block''})
\end{aligned}
$$

The key block is as many bytes long as is needed and is divided up into MAC secrets, symmetric encryption keys, and input vectors for blocking used with symmetric encryption. These are, in order, the client write MAC secret, server write MAC secret, client write key, server write key, client write IV and server write IV. Unneeded values are omitted.

client write MAC secret	server write MAC secret	client write key	server write key	client write IV	server write IV

Key Block

For export cipher suites, in order to comply with US export restrictions the input vectors are not derived from the master secret as it must be non-secret, and the encryption keys expanded from the 40bit limit dictated by the export restrictions. The key is expanded rather than used verbatim so that the server and client random act as a salt for the key that is used, preventing attackers from creating a lookup-table attack on a 40bit key space.

The calculations are as follows, though it should be noted that US export restrictions have been relaxed and as a result export cipher suites should not really be used by anyone ever.

$$
\begin{aligned}
final\_client\_write\_key &= X(client\_write\_key, \text{“client write key''}) \\
final\_server\_write\_key &= X(server\_write\_key, \text{“server write key''}) \\
iv\_block &= X(0, \text{“IV block''})
\end{aligned}
$$

The first-half of the iv block is used by the client and the second-half by the server.

5.2 SSLv3

SSLv3 master secret and key block calculations use a slightly different construction to TLS. Like TLS, the length of the master secret is 48 bytes, and key block is as long as it needs to be to produce all the keying material required. And the key block calculation below below is run for as many iterations as are needed, with excess data being discarded.

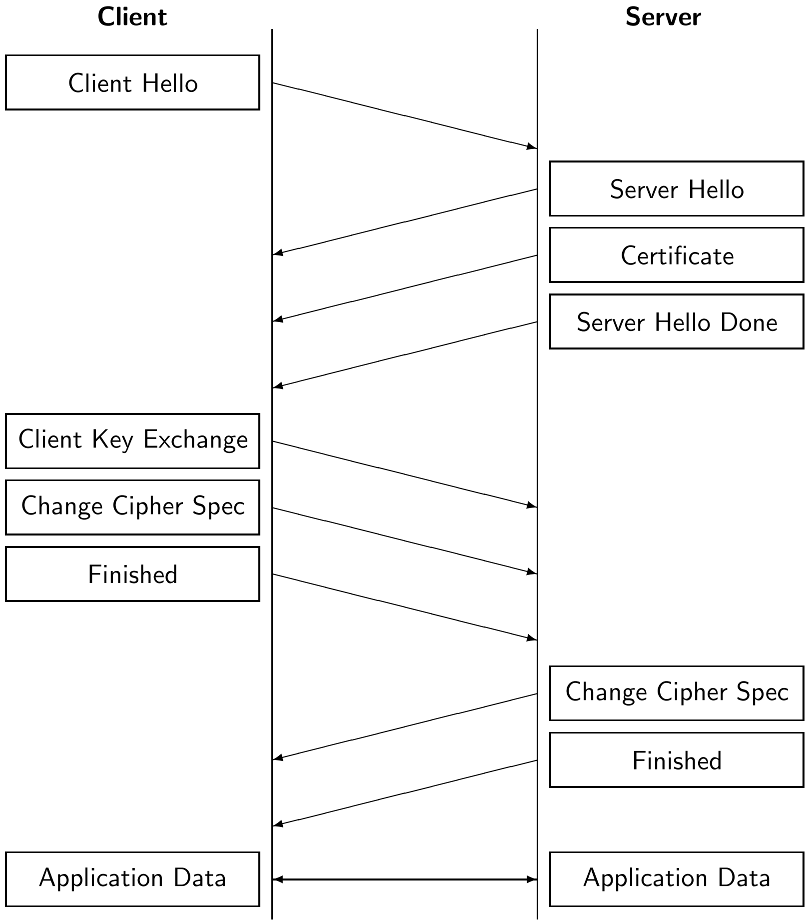
$$
\begin{aligned}
master\_secret &= MD5(pre\_master\_secret \oplus SHA1(\text{“A''} \oplus \\
&\quad pre\_master\_secret \oplus client\_random \oplus server\_random)) \oplus \\
&\quad MD5(pre\_master\_secret \oplus SHA1(\text{“BB''} \oplus \\
&\quad pre\_master\_secret \oplus client\_random \oplus server\_random)) \oplus \\
&\quad MD5(pre\_master\_secret \oplus SHA1(\text{“CCC''} \oplus \\
&\quad pre\_master\_secret \oplus client\_random \oplus server\_random)) \\
key\_block &= MD5(master\_secret \oplus SHA1(\text{“A''} \oplus \\
&\quad master\_secret \oplus server\_random \oplus client\_random)) \oplus \\
&\quad MD5(master\_secret \oplus SHA1(\text{“BB''} \oplus \\
&\quad secret \oplus server\_random \oplus client\_random)) \oplus \\
&\quad MD5(master\_secret \oplus SHA1(\text{“CCC''} \oplus \\
&\quad master\_secret \oplus server\_random \oplus client\_random)) \oplus ...
\end{aligned}
$$

As per TLS, when an export cipher is used, the IV must be non-secret and the asymmetric encryption keys are expanded from 40bit. This is done as follows:

$$
\begin{aligned}
final\_client\_write\_key &= MD5(client\_write\_key \oplus client\_random \oplus server\_random)\\
final\_server\_write\_key &= MD5(server\_write\_key \oplus server\_random \oplus client\_random)\\
client\_write\_iv &= MD5(client\_random \oplus server\_random)\\
server\_write\_iv &= MD5(server\_random \oplus client\_random)
\end{aligned}
$$

6 Handshake Sequences

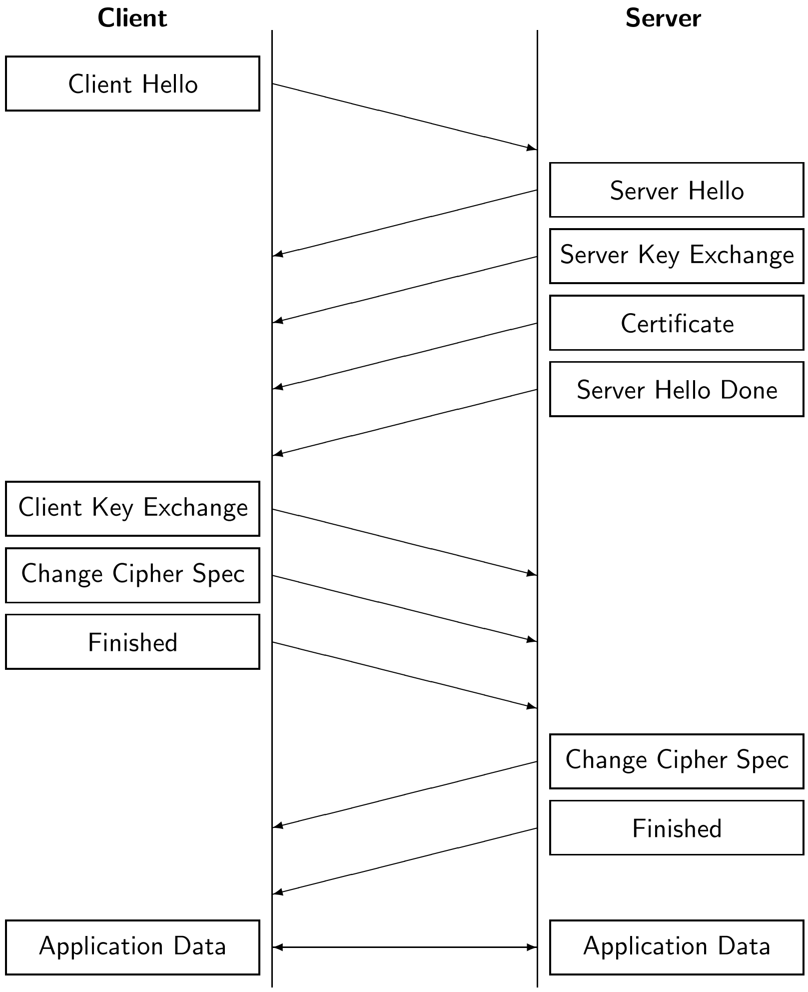
The message protocol described in the previous section affords a number of handshake sequences. This section will cover three of the most common ones, RSA, ephemeral RSA and Diffie-Hellman, and session resumption. Other handshake sequences that are not covered include client authentication, fixed Diffie-Hellman and renegotiation.



RSA Handshake Sequence

The RSA handshake sequence is used for a session with a cipher suite that uses non-ephemeral RSA for key exchange. In this handshake sequence the master-secret is calculated by both sides using the server and client random which are included in the client and server hello messages respectively. The master-secret calculation also includes the pre-master secret which is sent by the client in the client key, encrypted with key supplied by the server in the certificate message.

The client and server both begin encrypting records using the negotiated keys and cipher suite immediately after sending the change cipher spec message. The encrypted finished message includes a digest of all handshake messages sent and received, this verifies that the handshake was not tampered with a third party. It is verified by the recipient independently calculating the digest.

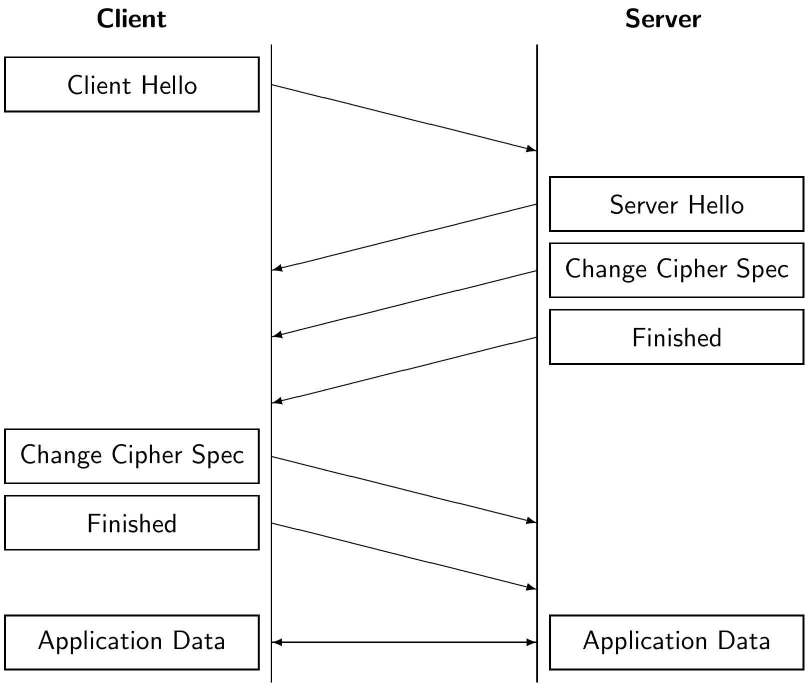


Ephemeral Handshake Sequence

The Ephemeral handshake sequence is used if ephemeral Diffie-Hellman or ephemeral RSA is used for key-exchange. It differs from the RSA handshake sequence above in that a server key exchange message is sent by the server immediately after the server hello message. This message includes the key that is used encrypt the pre-master secret in the client key exchange message.

Ephemeral RSA is used to allow a certificate longer than 512bits to be used with an export cipher. This is done by using the certificate only to verify the server, while temporary keys that are 512 bits or shorter are used to encrypt the client key exchange message. This is because export ciphers may not use an RSA key longer than 512 bits for encryption. However certificates of this length are are somewhat vulnerable to attack.

Ephemeral Diffie-Hellman by definition uses temporary keys which are not included in the certificate, and these are sent in the server key exchange message.



Resumed Session Handshake Sequence

An abbreviated handshake sequence is used if session resumption takes place. This typically occurs if an end-user reconnects to a server withing a short time frame, such as accessing a number of different pages from the same web site in the course of online shopping. In the client hello, the client identifies itself with the session-id previously issued to it by the server, and by returning this session-id in the server hello, the server implicitly agrees to use an abbreviated handshake. The master secret from the original connection is cached by both sides. This is used along with the server and client random for this connection, sent in the server hello and client hello respectively. The abbreviated handshake skips the need for the server to decrypt the client key exchange message to obtain the pre-master secret. This decryption is typically the slowest part of the handshake unless hardware acceleration is used. It also skips the need to calculate the master secret using the PRF, in general has less message processing and the the handshake only takes 1.5 network round-trips instead of 2. All this adds up to make this abbreviated handshake significantly faster in most cases, and it is generally regarded as essential for performance of an SSL/TLS server.

A Alert Values

This appendix details the defined alert severities and descriptions as defined in the TLSv1 specification[4].

Alert Severities

Severity	Numeric Value
warning	1
fatal	2

Alert Descriptions

Description	Numeric Value
close_notify	0
unexpected_message	10
bad_record_mac	20
decryption_failed	21
record_overflow	22
decompression_failure	30
handshake_failure	40
bad_certificate	42
unsupported_certificate	43
certificate_revoked	44
certificate_expired	45
certificate_unknown	46
illegal_parameter	47
unknown_ca	48
access_denied	49
decode_error	50
decrypt_error	51
export_restriction	60
protocol_version	70
insufficient_security	71
internal_error	80
user_cancelled	90
no_renegotiation	100

References

[1] Philip L. Karlton Alan O. Freier, Paul C. Kocher. Ssl 3.0 specification. http://wp.netscape.com/eng/ssl3/, November 1996.

[2] John Banes and Richard Harrington. 56-bit export cipher suites for tls. http://www.ietf.org/proceedings/02mar/I-D/draft-ietf-tls-56-bit-ciphersuites-01.txt, July 2001.

[3] P. Jones D. Eastlake 3rd. Us secure hash algorithm 1 (sha1). http://www.ietf.org/rfc/rfc3174.txt, September 2001.

[4] T. Dierks and C. Allen. The tls protocol version 1.0. http://www.ietf.org/rfc/rfc2459.txt, January 1999.

[5] Hans Dobbertin. The status of md5 after a recent attack. *RSA Laboratories' Crypto Bytes*, 2(2):1,3–6, Summer 1996. http://www.rsasecurity.com/rsalabs/cryptobytes/.

[6] Olivier Dubuisson. *ASN.1 — Communication between heterogeneous systems*. Morgan Kaufmann, October 2000. http://www.oss.com/asn1/dubuisson.html.

[7] M. Dworkin. Recommendation for block cipher modes of operation: Methods and techniques. *NIST Special Publication 800-38A*, pages 9–11, 2001. http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.

[8] M. Bellare H. Krawczyk and R. Canetti. Hmac: Keyed-hashing for message authentication. http://www.ietf.org/rfc/rfc2104.txt, February 1997.

[9] Kipp E. B. Hickman. The ssl protocol. http://wp.netscape.com/eng/ssl3/, November 1994.

[10] David Hopwood. Standard cryptographic algorithm naming. http://www.users.zetnet.co.uk/hopwood/crypto/scan/, 2001.

[11] J. Jonsson and B. Kaliski. Public-key cryptography standards (pkcs) #1: Rsa cryptography specifications version 2.1. http://www.ietf.org/rfc/rfc3447.txt, February 2003.

[12] K. Kaukonen and R. Thayer. A stream cipher encryption algorithm "arcfour". http://www.mozilla.org/projects/security/pki/nss/draft-kaukonen-cipher-arcfour-03.txt, July 1997.

[13] Tom Leighton and Eric Lehman. Independence. http://theory.lcs.mit.edu/classes/6.042/fall04/prob_indep.pdf, November 2004.

[14] R. Canetti M. Bellare and H. Krawczyk. The hmac construction. *RSA Laboratories' Crypto Bytes*, 2(1):12–15, Spring 1996. http://www.rsasecurity.com/rsalabs/cryptobytes/.

[15] R. Canetti M. Bellare and H. Krawczyk. Keying hashing functions for message authentication. http://www-cse.ucsd.edu/users/mihir/papers/hmac.html, June 1996.

[16] National Institute of Standards and U.S. Department of Commerce Technology. Fips 180-1 — secure hash standard. http://www.itl.nist.gov/fipspubs/fip180-1.htm, April 1995.

[17] W. Ford R. Housley, W. Polk and D. Solo. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. http://www.ietf.org/rfc/rfc2459.txt, April 2002.

[18] E. Rescorla. Diffie-hellman key agreement method. http://www.ietf.org/rfc/rfc2631.txt, June 1999.

[19] Eric Rescorla. *SSL and TLS — Designing and Building Secure Systems*. Addison Wesley, 2001.

[20] R. Rivest. The md5 message-digest algorithm. http://www.ietf.org/rfc/rfc1321.txt, April 1992.

[21] Bruce Schneier. *Applied Cryptography*. Wiley & Sons, second edition edition, 1995.

[22] Bruce Schneier. Cryptanalysis of sha-1. http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html, February 2005.

[23] RSA Security. What is rc4? http://www.rsasecurity.com/rsalabs/faq/3-6-3.html.

[24] RSA Security. Pkcs #1: Rsa cryptography standard. http://www.rsasecurity.com/rsalabs/pkcs, February 2003.

[25] Hongbo Yu Xiaoyun Wang, Yiqun Lisa Yin. Collision search attacks on sha1. http://theory.csail.mit.edu/~yiqun/shanote.pdf, February 2005. This is a note of the findings, the full paper is not available yet.