Security at the Transport Layer: SSL and TLS

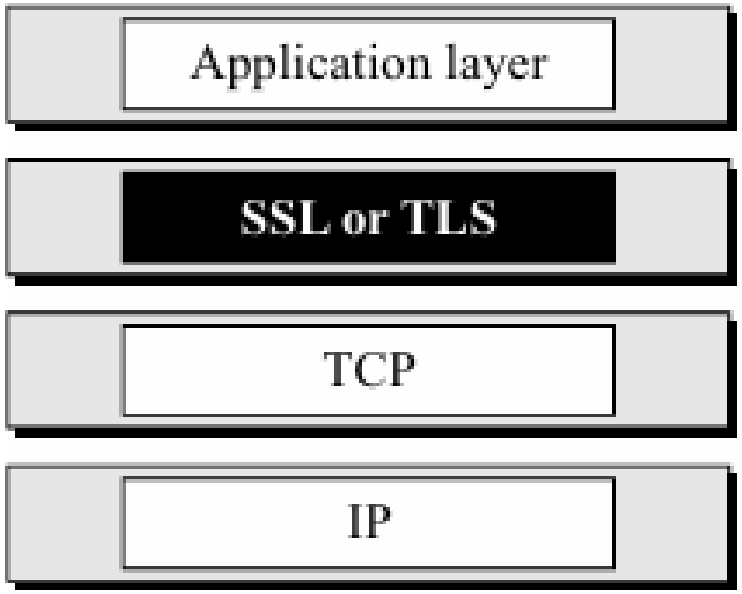
Sachin Tripathi

IIT(ISM), Dhanbad

Introduction

- Transport layer security provides end-to-end security services for applications that use a reliable transport layer protocol such as TCP.
- The idea is to provide security services for transactions on the Internet. For example, when a customer shops online, the following security services are desired:
  - The customer needs to be sure that the server belongs to the actual vendor, not an impostor. The customer does not want to give an impostor her credit card number (entity authentication).
  - The customer and the vendor need to be sure that the contents of the message are not modified during transmission (message integrity).
  - The customer and the vendor need to be sure that an impostor does not intercept sensitive information such as a credit card number (confidentiality).

Location of SSL and TLS in the Internet model



- One of the goals of these protocols is to provide server and client authentication, data confidentiality, and data integrity.
- Application-layer client/server programs, such as Hypertext Transfer Protocol (HTTP), that use the services of TCP can encapsulate their data in SSL packets.

- If the server and client are capable of running SSL (or TLS) programs then the client can use the URL https://… instead of http://… to allow HTTP messages to be encapsulated in SSL (or TLS) packets.

Secure Socket Layer (SSL)

- SSL is designed to provide security and compression services to data generated from the application layer.
- Typically, SSL can receive data from any application layer protocol, but usually the protocol is HTTP.
- The data received from the application is compressed (optional), signed, and encrypted. The data is then passed to a reliable transport layer protocol such as TCP.

SSL ARCHITECTURE

Services

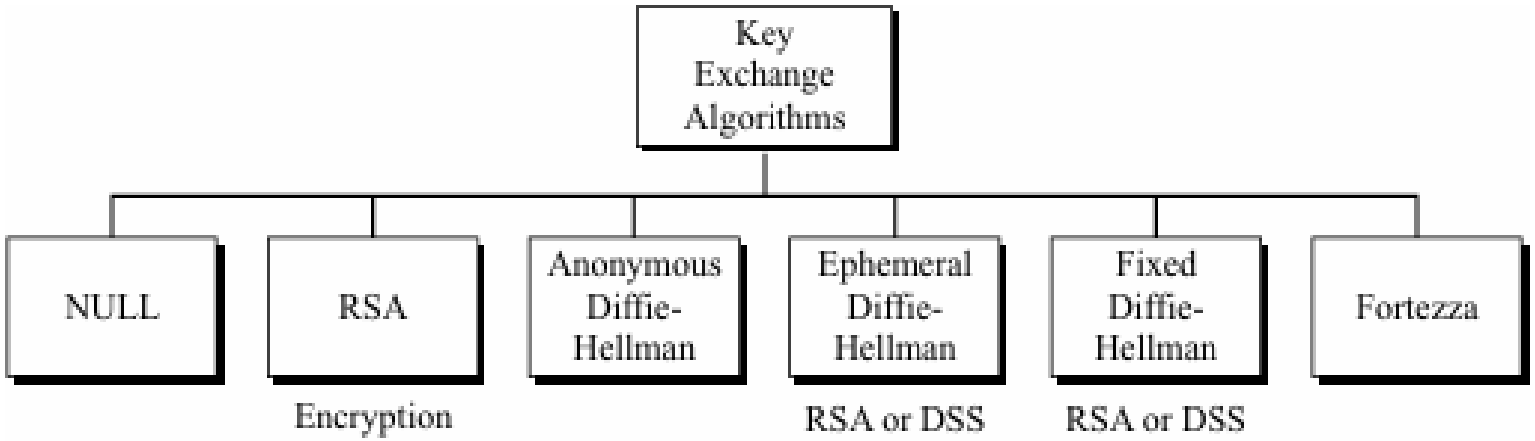
SSL provides several services on data received from the application layer.

- **Fragmentation**
  First, SSL divides the data into blocks of 2^14 bytes or less.
- **Compression**
  Each fragment of data is compressed using one of the lossless compression methods negotiated between the client and server. This service is optional.
- **Message Integrity**
  To preserve the integrity of data, SSL uses a keyed-hash function to create a MAC.
- **Confidentiality** To provide confidentiality, the original data and the MAC are encrypted using symmetric key cryptography.
- **Framing** A header is added to the encrypted payload. The payload is then passed to a reliable transport layer protocol.

Key Exchange Algorithms

- To exchange an authenticated and confidential message, the client and the server each need six cryptographic secrets (four keys and two initialization vectors). However, to create these secrets, one pre-master secret must be established between the two parties. SSL defines six key-exchange methods to establish this pre master secret:

Key-exchange methods



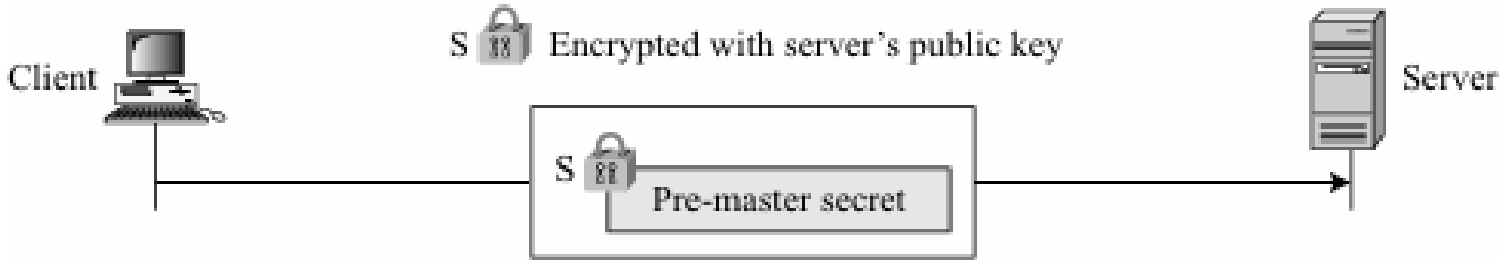
NULL

There is no key exchange in this method. No pre-master secret is established between the client and the server

Note:-Both client and server need to know the value of the pre-master secret.

RSA

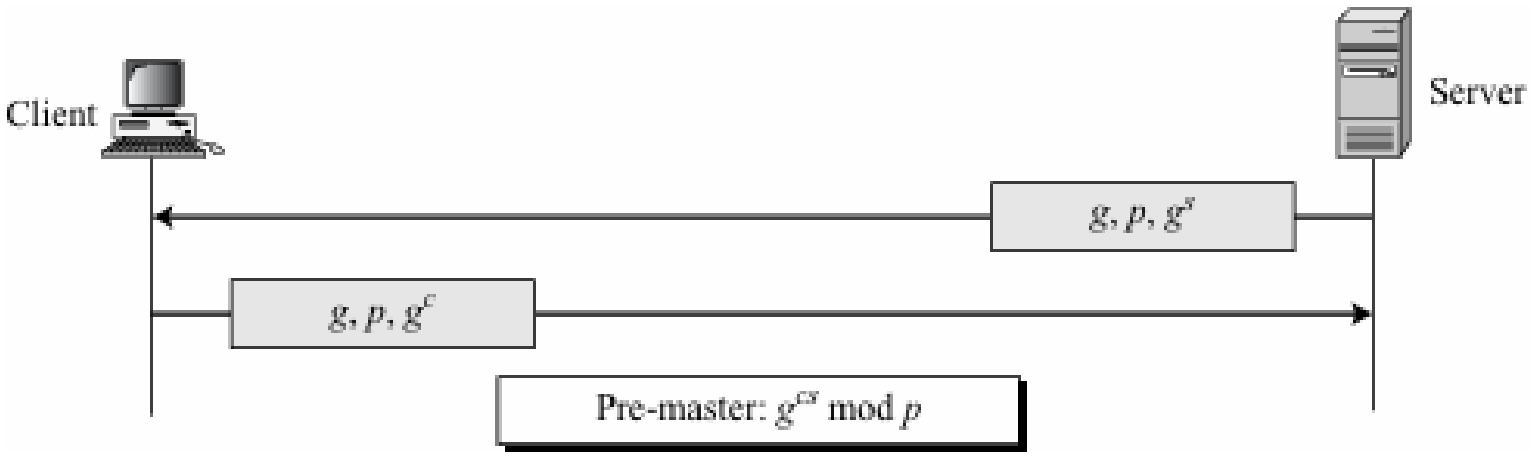
In this method, the pre-master secret is a 48-byte random number created by the client, encrypted with the server's RSA public key, and sent to the server. The server needs to send its RSA encryption/decryption certificate



RSA key exchange; server public key

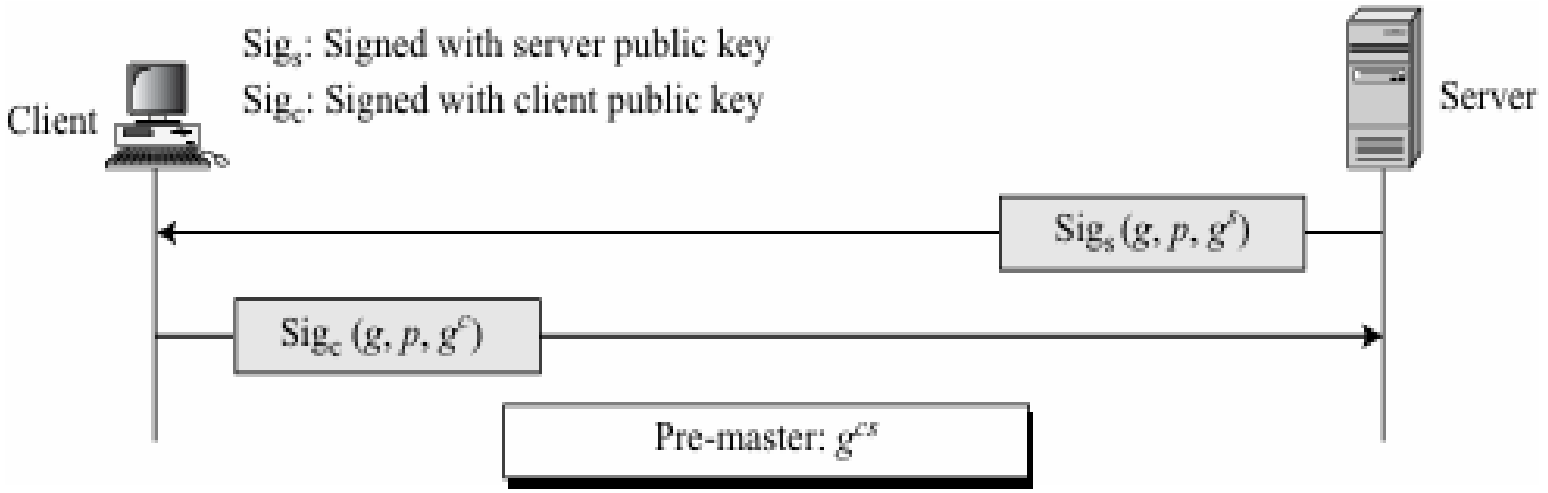
Anonymous Diffie-Hellman

- This is the simplest and most insecure method.
- The pre-master secret is established between the client and server using the Diffie-Hellman (DH) protocol.
- The Diffie Hellman half-keys are sent in plaintext. It is called anonymous Diffie-Hellman because neither party is known to the other.
- Disadvantage of this method is the man-in-the-middle attack.



Ephemeral Diffie-Hellman

- To thwart the man-in-the-middle attack, the ephemeral Diffie-Hellman key exchange can be used.
- Each party sends a Diffie-Hellman key signed by its private key.
- The receiving party needs to verify the signature using the public key of the sender.
- The public keys for verification are exchanged using either RSA or DSS digital signature certificates.



Fixed Diffie-Hellman

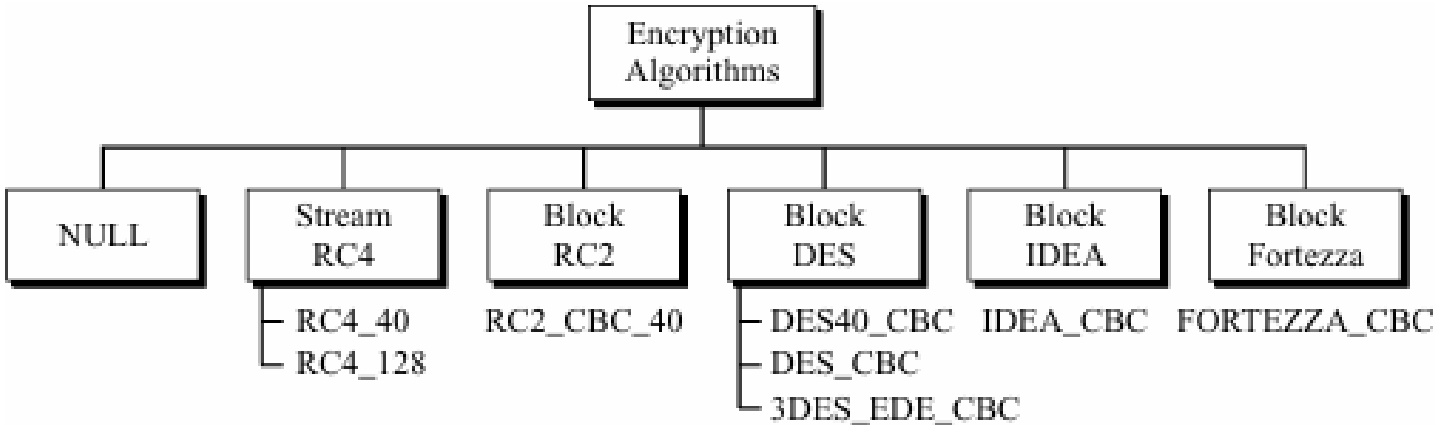
- Another solution is the fixed Diffie-Hellman method.
- All entities in a group can prepare fixed Diffie-Hellman parameters (g and p).
- Then each entity can create a fixed Diffie-Hellman half-key (g^x).
- For additional security, each individual half-key is inserted into a certificate verified by a certification authority (CA).
- In other words, the two parties do not directly exchange the half-keys; the CA sends the half-keys in an RSA or DSS special certificate.
- When the client needs to calculate the pre-master, it uses its own fixed half-key and the server half-key received in a certificate.
- The server does the same, but in the reverse order. Note that no key-exchange messages are passed in this method; only certificates are exchanged.

Fortezza

Fortezza (derived from the Italian word for fortress) is a registered trademark of the U.S. National Security Agency (NSA). It is a family of security protocols developed for the Defense Department

Encryption/Decryption Algorithms

- There are several choices for the encryption/decryption algorithm.
- We can divide the algorithms into 6 groups
- All block protocols use an 8-byte initialization vector (IV) except for Fortezza, which uses a 20-byte IV.



- **NULL**

The NULL category simply defines the lack of an encryption/decryption algorithm.

- **Stream RC**

Two RC algorithms are defined in stream mode: **RC4-40 (40-bit key) and RC4-128 (128-bit key).**

- **Stream RC**

One RC algorithm is defined in block mode: **RC2_CBC_40 (40-bit key).**

DES

All DES algorithms are defined in block mode.

DES40_CBC uses a 40-bit key.

Standard DES is defined as DES_CBC.

3DES_EDE_CBC uses a 168-bit key.

IDEA

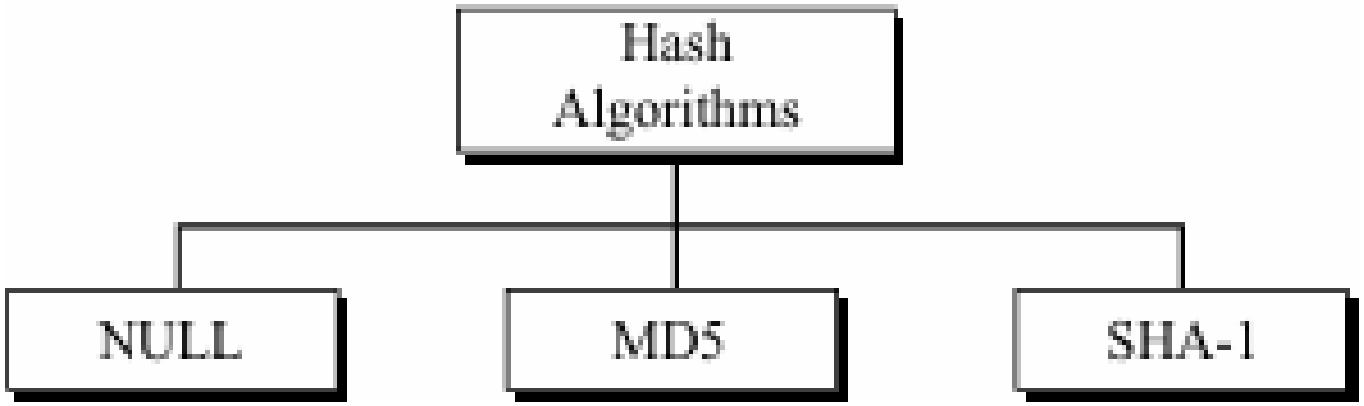
The one IDEA algorithm defined in block mode is **IDEA_CBC, with a 128-bit key.**

Fortezza

The one Fortezza algorithm defined in block mode is **FORTEZZA_CBC, with a 96-bit key.**

Hash Algorithms

SSL uses hash algorithms to provide message integrity (message authentication). Three hash functions are defined, as shown below



Null:-The two parties may decline to use an algorithm. In this case, there is no hash function and the message is not authenticated.

MD5:-The two parties may choose MD5 as the hash algorithm. In this case, a 128-key MD5 hash algorithm is used.

SHA-1:-The two parties may choose SHA as the hash algorithm. In this case, a 160-bit SHA-1 hash algorithm is used.

Cipher Suite

The combination of key exchange, hash, and encryption algorithms defines a cipher suite for each SSL session.

SSL cipher suite list

Cipher suite	*Key Exchange*	*Encryption*	*Hash*
SSL_NULL_WITH_NULL_NULL	NULL	NULL	NULL
SSL_RSA_WITH_NULL_MD5	RSA	NULL	MD5
SSL_RSA_WITH_NULL_SHA	RSA	NULL	SHA-1
SSL_RSA_WITH_RC4_128_MD5	RSA	RC4	MD5
SSL_RSA_WITH_RC4_128_SHA	RSA	RC4	SHA-1
SSL_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA	SHA-1
SSL_RSA_WITH_DES_CBC_SHA	RSA	DES	SHA-1
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES	SHA-1
SSL_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4	MD5
SSL_DH_anon_WITH_DES_CBC_SHA	DH_anon	DES	SHA-1
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES	SHA-1
SSL_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES	SHA-1
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES	SHA-1
SSL_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES	SHA-1
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES	SHA-1
SSL_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES	SHA-1
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES	SHA-1
SSL_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES	SHA-1
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES	SHA-1
SSL_FORTEZZA_DMS_WITH_NULL_SHA	Fortezza	NULL	SHA-1
SSL_FORTEZZA_DMS_WITH_FORTEZZA_CBC_SHA	Fortezza	Fortezza	SHA-1
SSL_FORTEZZA_DMS_WITH_RC4_128_SHA	Fortezza	RC4	SHA-1

Each suite starts with the term “SSL” followed by the key exchange algorithm. The word “WITH” separates the key exchange algorithm from the encryption and hash algorithms.

Compression Algorithms

- Compression is optional in SSLv3. No specific compression algorithm is defined for SSLv3. Therefore, the default compression method is **NULL**. However, a system can use whatever compression algorithm it desires.

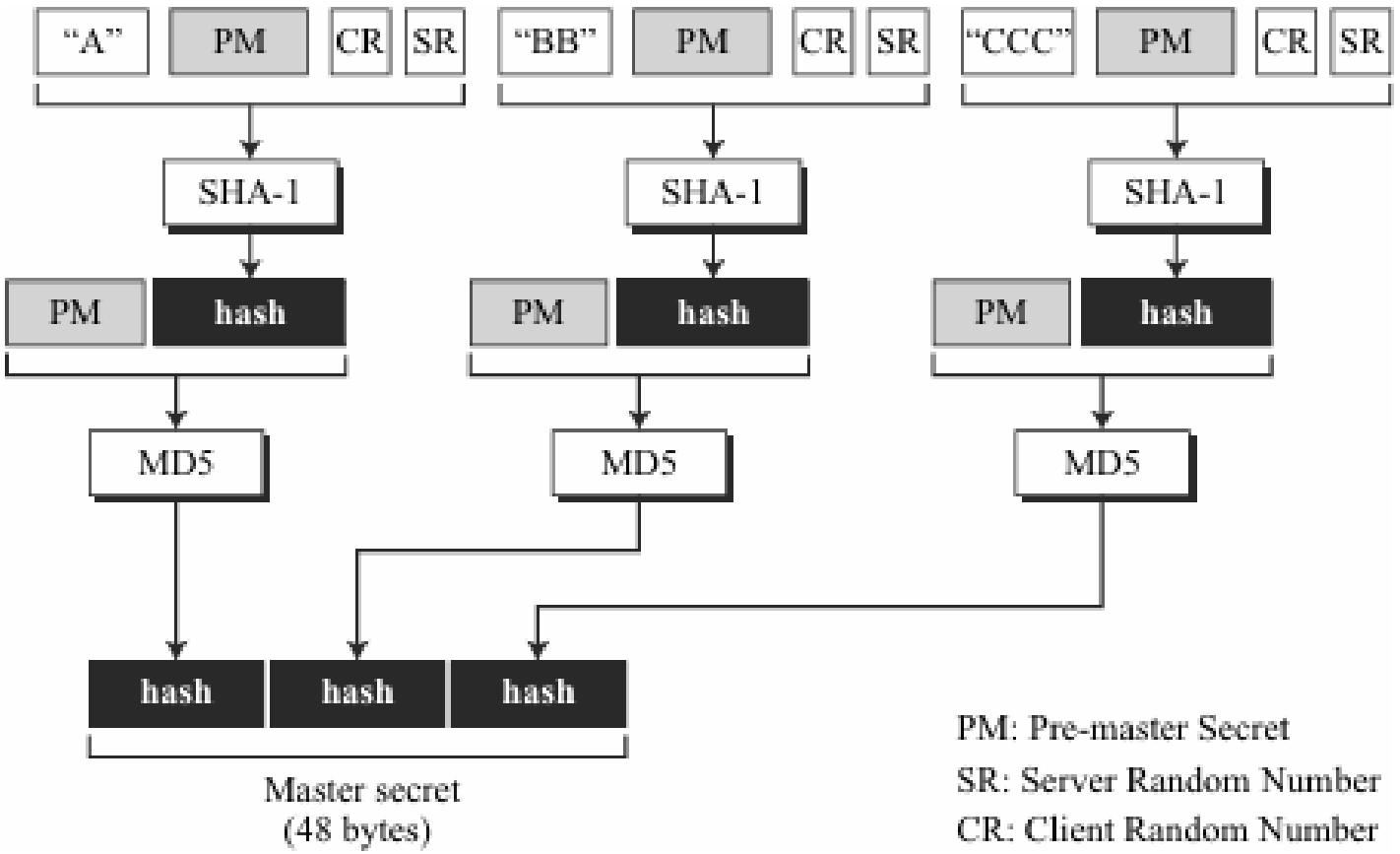
Cryptographic Parameter Generation

- To achieve message integrity and confidentiality, SSL needs six cryptographic secrets, four keys and two IVs.
- The client needs one key for message authentication (HMAC), one key for encryption, and one IV for block encryption.
- The server needs the same.
- SSL requires that the keys for one direction be different from those for the other direction. If there is an attack in one direction, the other direction is not affected. The parameters are generated using the following procedure:
  - The client and server exchange two random numbers; one is created by the client and the other by the server.
  - The client and server exchange one pre-master secret using one of the key exchange algorithms.
  - A 48-byte master secret is created from the pre-master secret by applying two hash functions (SHA-1 and MD5)

- The master secret is used to create variable-length key material by applying the same set of hash functions and prepending with different constants as shown in Figure below. The module is repeated until key material of adequate size is created.

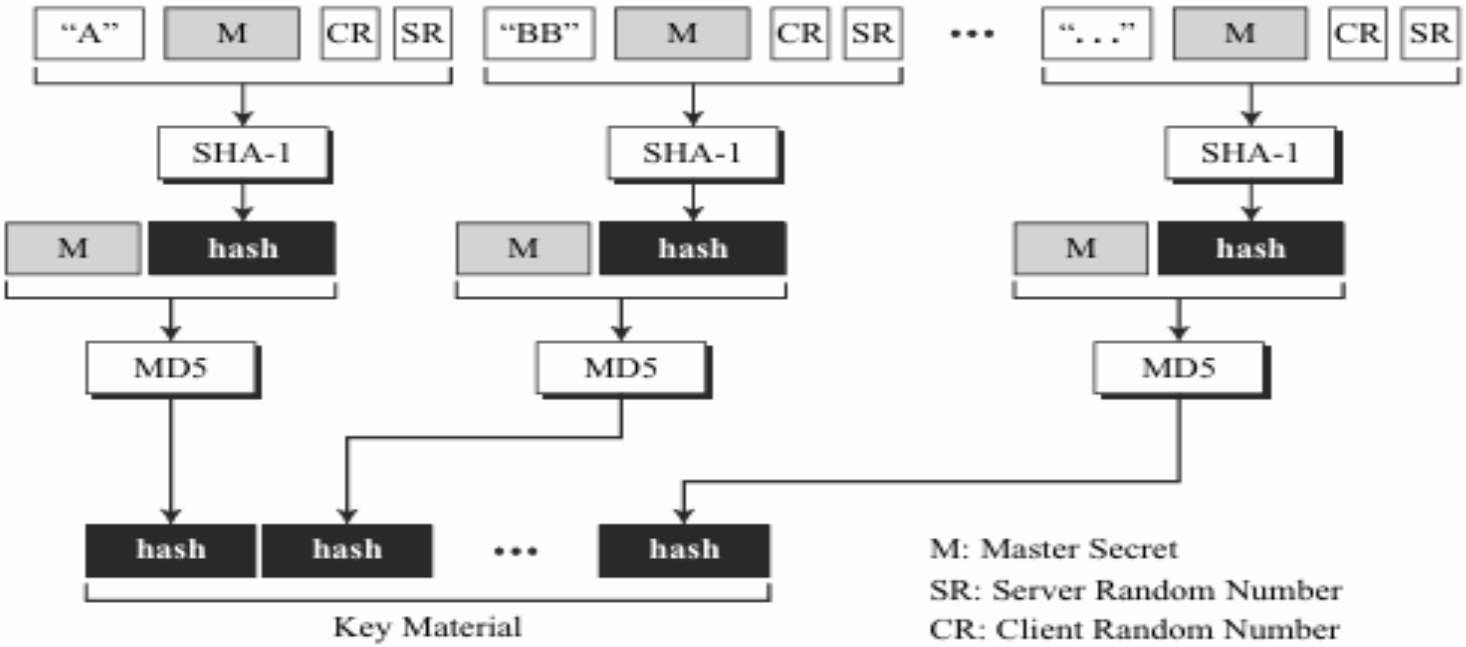
- Six different keys are extracted from the key material, as shown in Figure below

Calculation of master secret from pre-master secret



The module is repeated until key material of adequate size is created

Calculation of key material from master secret

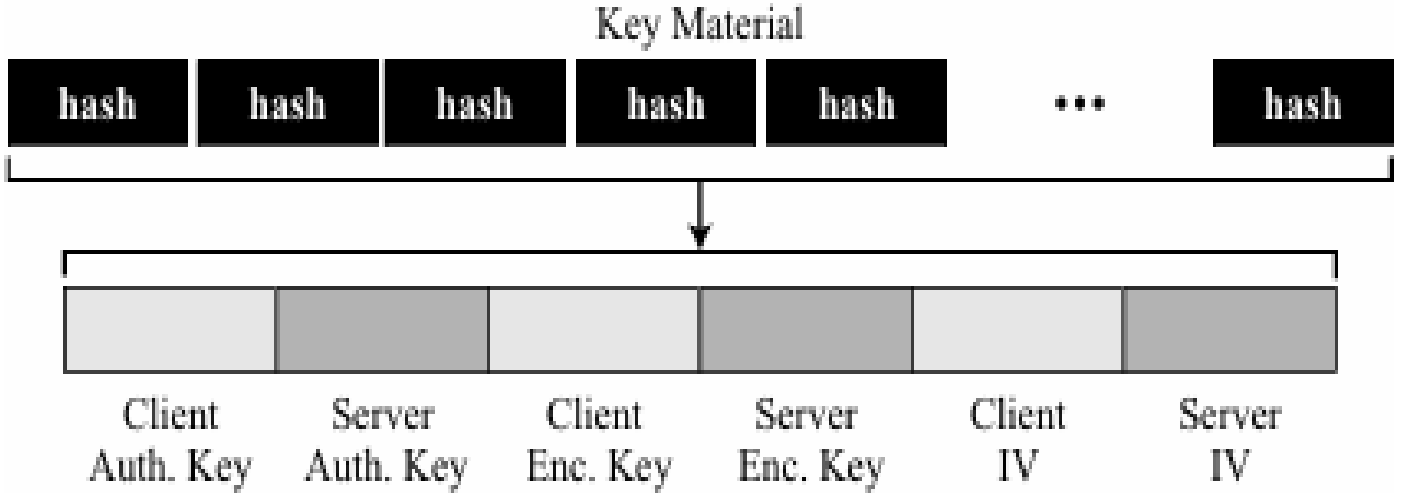


Note that the length of the key material block depends on the cipher suite selected and the size of keys needed for this suite.

Auth. Key: Authentication Key

Enc. Key: Encryption Key

IV: Initialization Vector



Sessions and Connections

- SSL differentiates a connection from a session.

Let us elaborate on these two terms here.

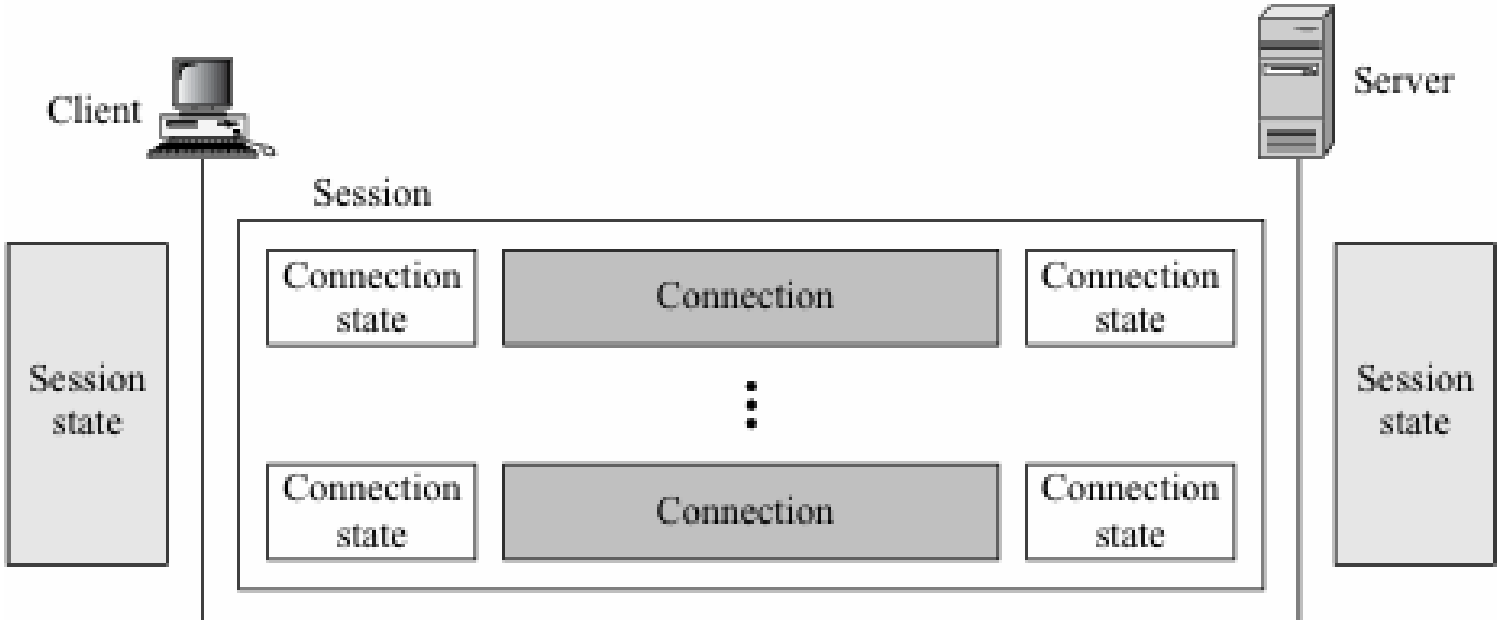
- A session is an association between a client and a server. After a session is established, the two parties have common information such as the session identifier, the certificate authenticating each of them (if necessary), the compression method (if needed), the cipher suite, and a master secret that is used to create keys for message authentication encryption. For two entities to exchange data, the establishment of a session is necessary, but not sufficient; they need to create a connection between themselves.

- The two entities exchange two random numbers and create, using the master secret, the keys and parameters needed for exchanging messages involving authentication and privacy. By allowing a session to be suspended and resumed, the process of the master secret calculation can be eliminated.

- A session can consist of many connections.
- A connection between two parties can be terminated and reestablished within the same session.
- When a connection is terminated, the two parties can also terminate the session, but it is not mandatory.
- A session can be suspended and resumed again.
- To create a new session, the two parties need to go through a negotiation process.
- To resume an old session and create only a new connection, the two parties can skip part of the negotiation process and go through a shorter one.
- There is no need to create a master secret when a session is resumed. The separation of a session from a connection prevents the high cost of creating a master secret

In a session, one party has the role of a client and the other the role of a server; in a connection, both parties have equal roles, they are peers. .

A session and connections



Session State

A session is defined by a session state, a set of parameters established between the server and the client.

Session state parameters

Parameter	*Description*
Session ID	A server-chosen 8-bit number defining a session.
Peer Certificate	A certificate of type X509.v3. This parameter may by empty (null).
Compression Method	The compression method.
Cipher Suite	The agreed-upon cipher suite.
Master Secret	The 48-byte secret.
Is resumable	A yes-no flag that allows new connections in an old session.

Connection State

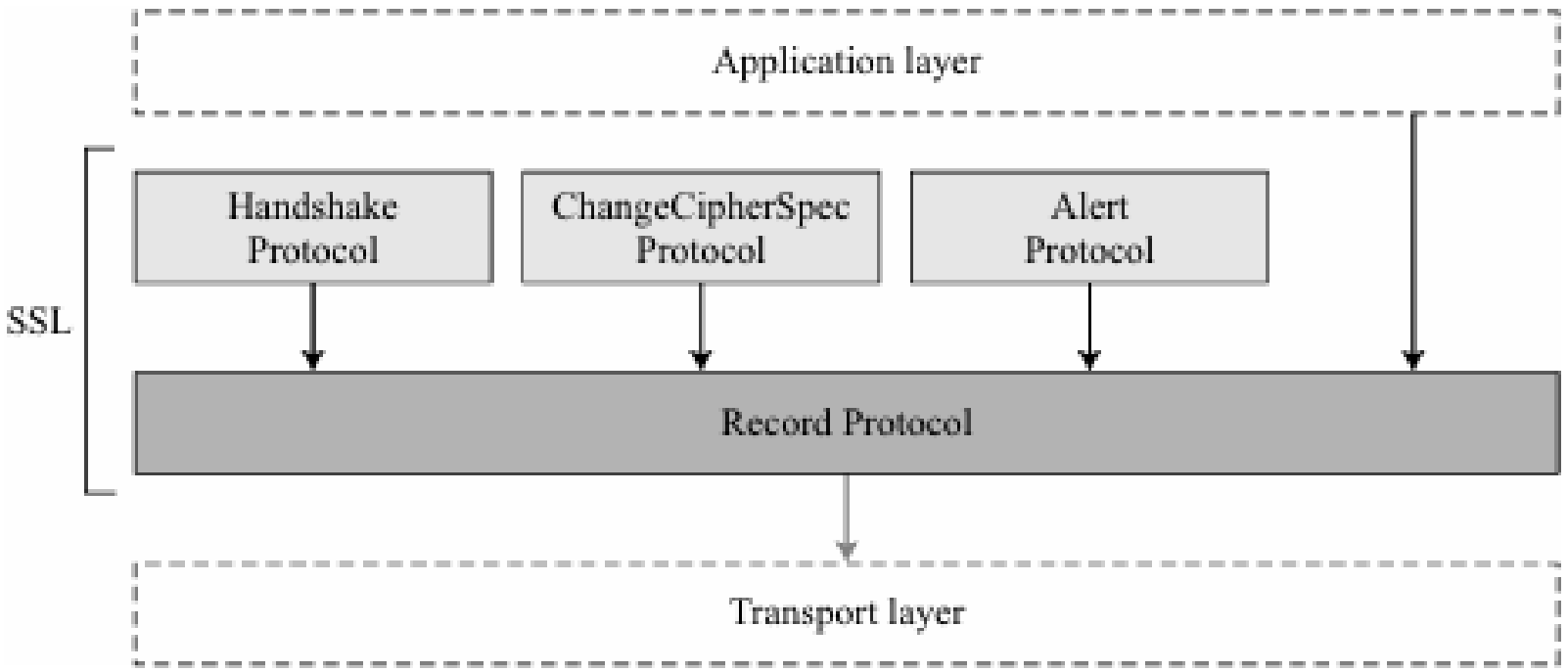
- A connection is defined by a connection state, a set of parameters established between two peers.
- SSL uses two attributes to distinguish cryptographic secrets: write and read.
- The term write specifies the key used for signing or encrypting outbound messages.
- The term read specifies the key used for verifying or decrypting inbound messages.
- The write key of the client is the same as the read key of the server; the read key of the client is the same as the write key of the server.

- The client and the server have six different cryptography secrets: three read secrets and three write secrets. The read secrets for the client are the same as the write secrets for the server and vice versa.

Connection state parameters

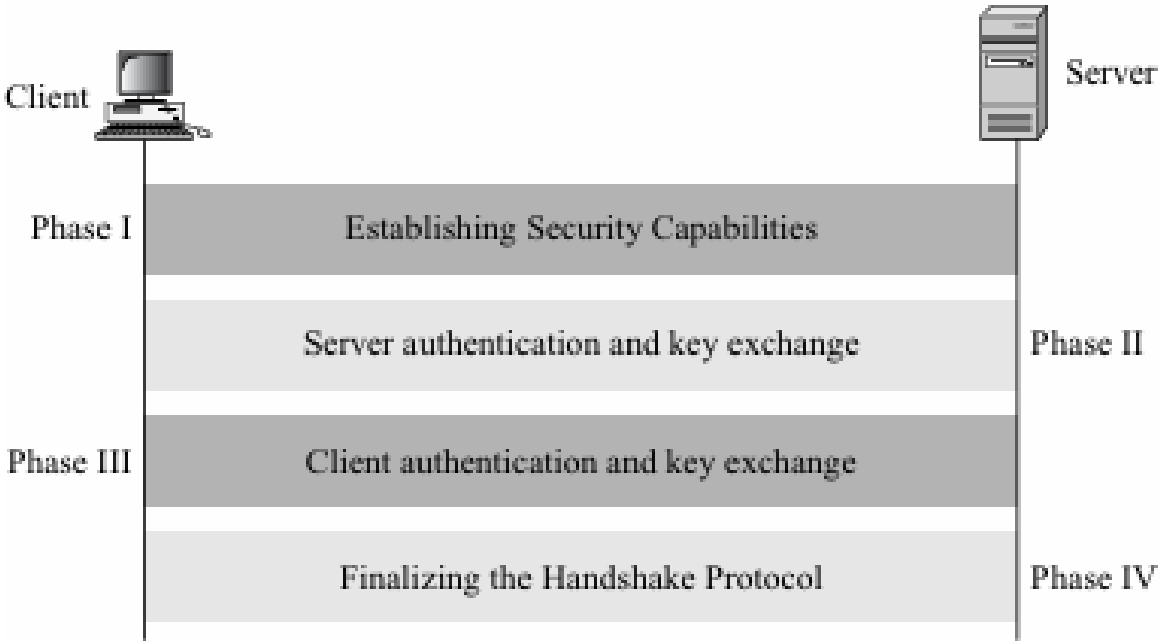
Parameter	*Description*
Server and client random numbers	A sequence of bytes chosen by the server and client for each connection.
Server write MAC secret	The outbound server MAC key for message integrity. The server uses it to sign; the client uses it to verify.
Client write MAC secret	The outbound client MAC key for message integrity. The client uses it to sign; the server uses it to verify.
Server write secret	The outbound server encryption key for message integrity.
Client write secret	The outbound client encryption key for message integrity.
Initialization vectors	The block ciphers in CBC mode use initialization vectors (IVs). One initialization vector is defined for each cipher key during the negotiation, which is used for the first block exchange. The final cipher text from a block is used as the IV for the next block.
Sequence numbers	Each party has a sequence number. The sequence number starts from 0 and increments. It must not exceed $2^{64} - 1$.

Four SSL Protocols



Handshake Protocol

- The Handshake Protocol uses messages to negotiate the cipher suite, to authenticate the server to the client and the client to the server if needed, and to exchange information for building the cryptographic secrets. The handshaking is done in four phases



Phase I: Establishing Security Capability

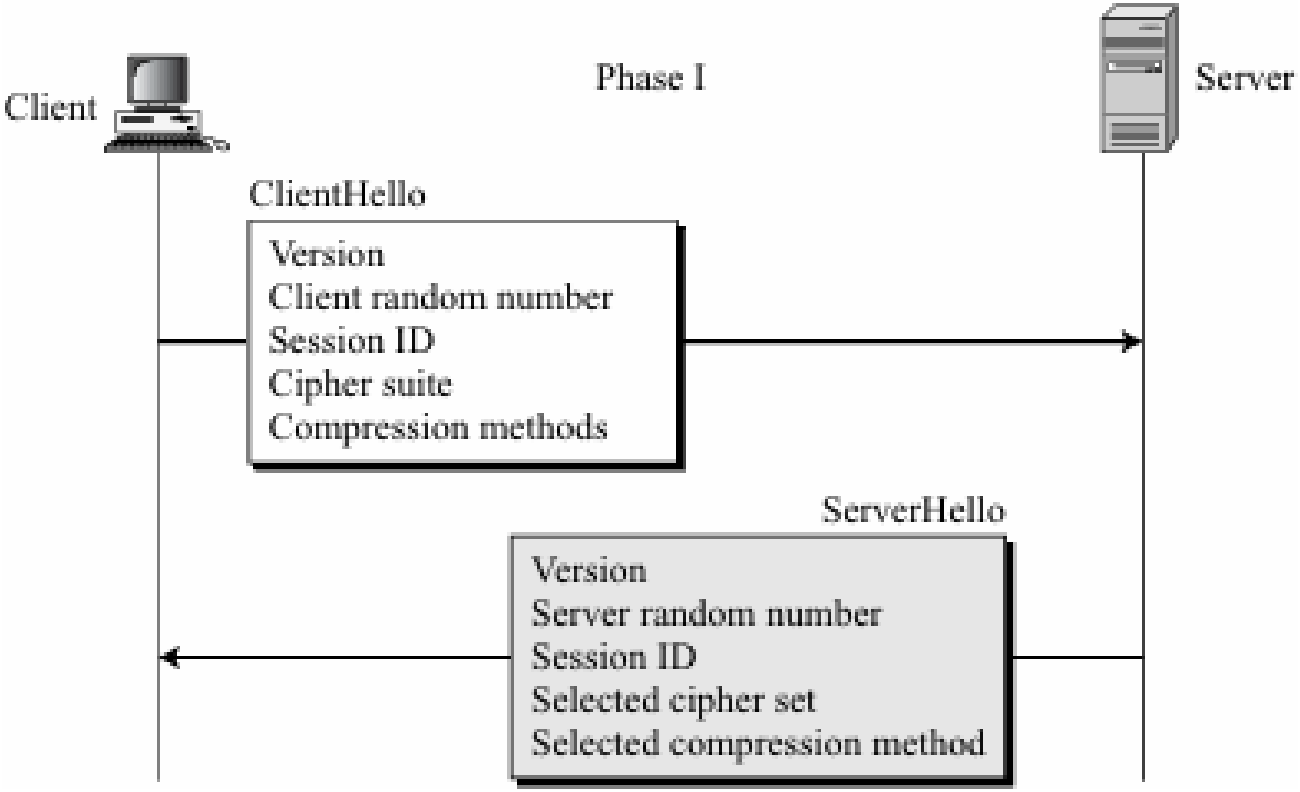
- In Phase I, the client and the server announce their security capabilities and choose those that are convenient for both.
- In this phase, a session ID is established and the cipher suite is chosen.
- The parties agree upon a particular compression method.
- Finally, two random numbers are selected, one by the client and one by the server, to be used for creating a master secret as we saw before.
- Two messages are exchanged in this phase: ClientHello and ServerHello messages

❑ ClientHello

The client sends the ClientHello message. It contains the following:

- The highest SSL version number the client can support.
- A 32-byte random number (from the client) that will be used for master secret generation.
- A session ID that defines the session.
- A cipher suite that defines the list of algorithms that the client can support.
- A list of compression methods that the client can support

Phase I of Handshake Protocol



❑ ServerHello

The server responds to the client with a ServerHello message. It contains the following:

- An SSL version number. This number is the lower of two version numbers: the highest supported by the client and the highest supported by the server.
- A 32-byte random number (from the server) that will be used for master secret generation.
- A session ID that defines the session.
- The selected cipher set from the client list.
- The selected compression method from the client list

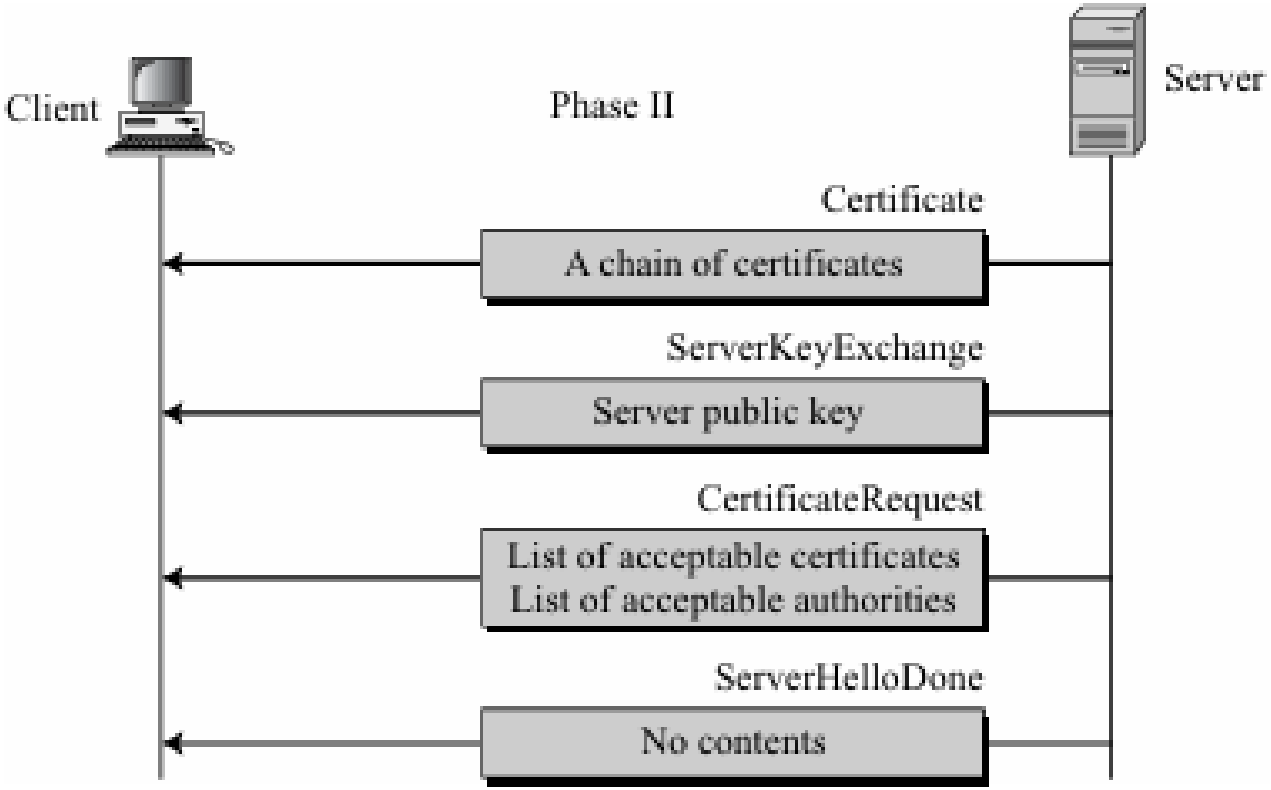
After Phase I, the client and server know the following:

- The version of SSL
- The algorithms for key exchange, message authentication, and encryption
- The compression method
- The two random numbers for key generation

Phase II: Server Key Exchange and Authentication

- In phase II, the server authenticates itself if needed.
- The sender may send its certificate, its public key, and may also request certificates from the client.
- At the end, the server announces that the serverHello process is done

Phase II of Handshake Protocol



- **Certificate** If it is required, the server sends a Certificate message to authenticate itself. The message includes a list of certificates of type X.509. The certificate is not needed if the key-exchange algorithm is anonymous Diffie-Hellman.

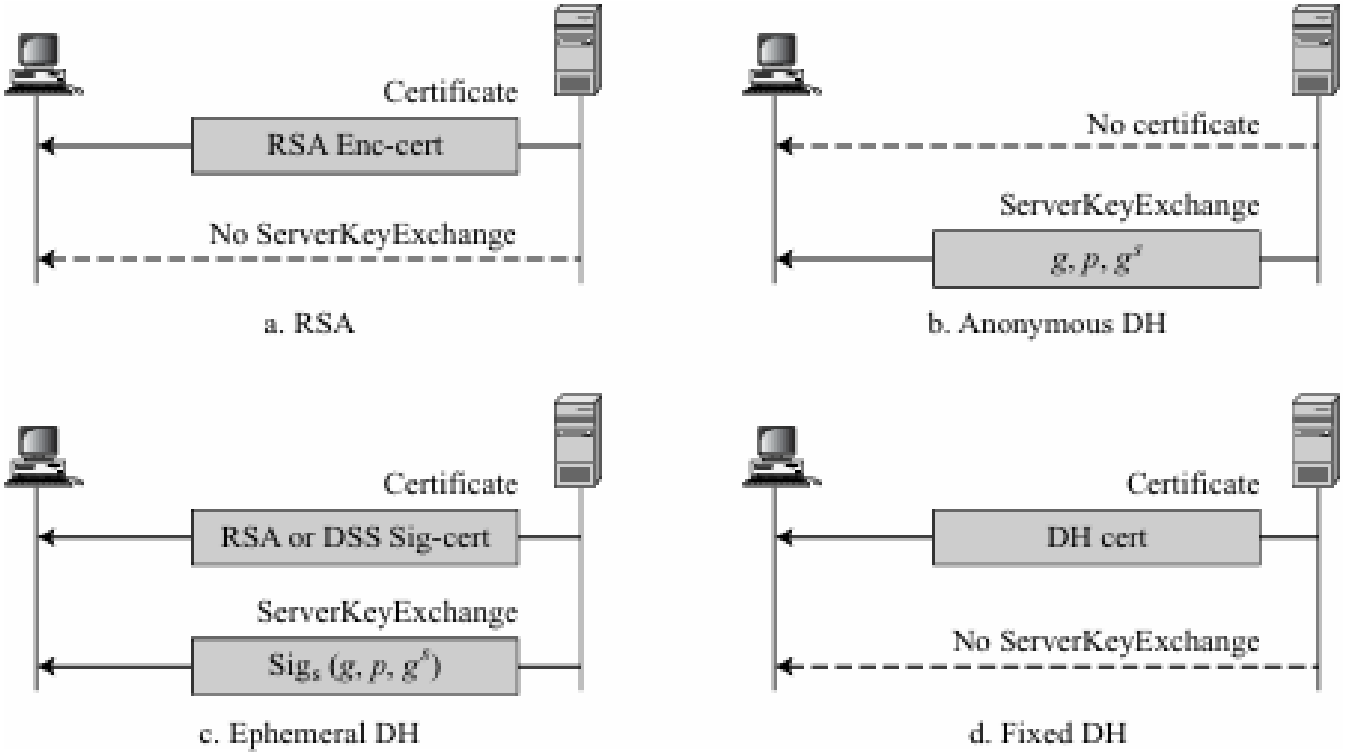
- **ServerKeyExchange** After the Certificate message, the server sends a ServerKey Exchange message that includes its contribution to the pre-master secret. This message is not required if the key-exchange method is RSA or fixed Diffie-Hellman.

- **CertificateRequest** The server may require the client to authenticate itself. In this case, the server sends a CertificateRequest message in Phase II that asks for certifica tion in Phase III from the client. The server cannot request a certificate from the client if it is using anonymous Diffie-Hellman.

- **ServerHelloDone** The last message in Phase II is the ServerHelloDone message, which is a signal to the client that Phase II is over and that the client needs to start Phase III.

- After Phase II, the server is authenticated to the client.
- The client knows the public key of the server if required.

Four cases in Phase II



- **RSA**.
  - In this method, the server sends its RSA encryption/decryption public-key certificate in the first message.
  - The second message, however, is empty because the pre-master secret is generated and sent by the client in the next phase. Note that the public-key certificate authenticates the server to the client.
  - When the server receives the pre-master secret, it decrypts it with its private key. The possession of the private key by the server is proof that the server is the entity that it claims to be in the public-key certificate sent in the first message.
- **Anonymous DH**.
  - In this method, there is no Certificate message.
  - An anonymous entity does not have a certificate.
  - In the ServerKeyExchange message, the server sends the Diffie-Hellman parameters and its half-key.

  Note that the server is not authenticated in this method.

❑ Ephemeral DH.

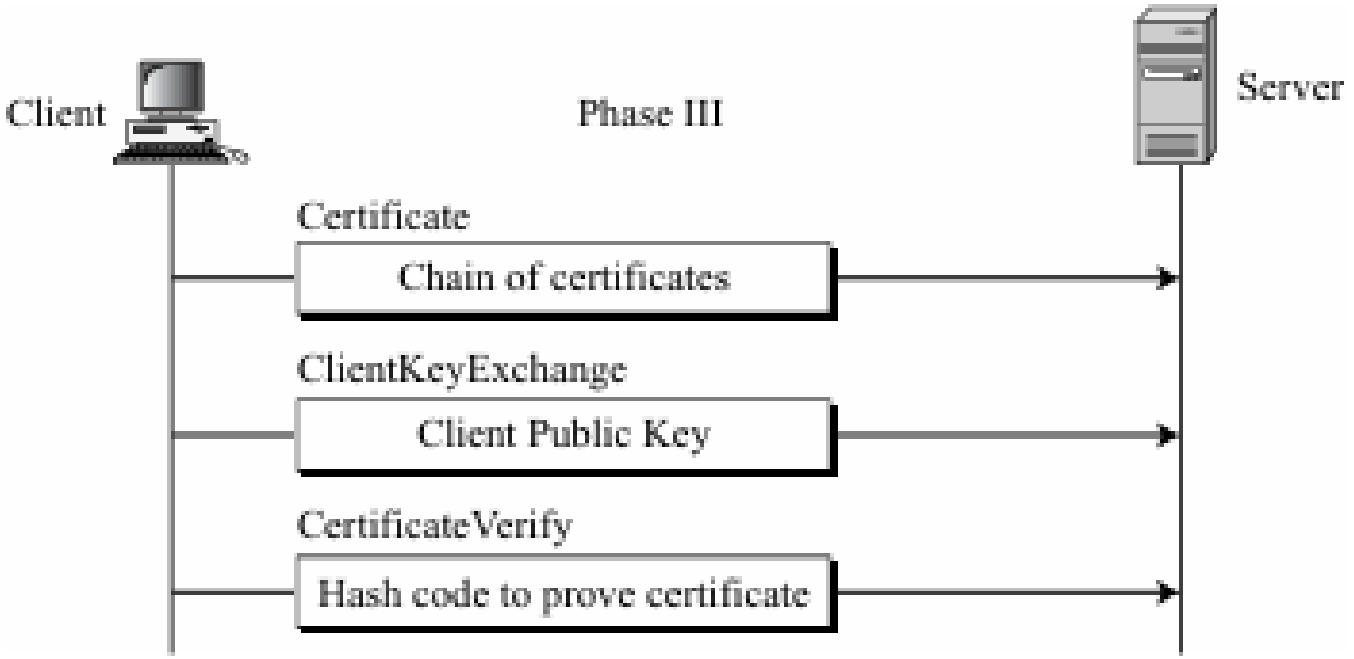
- In this method, the server sends either an RSA or a DSS digital signature certificate.
- The private key associated with the certificate allows the server to sign a message; the public key allows the recipient to verify the signature.
- In the second message, the server sends the Diffie-Hellman parameters and the half-key signed by its private key. Other text is also sent.
- The server is authenticated to the client in this method, not because it sends the certificate, but because it signs the parameters and keys with its private key.
- The possession of the private key is proof that the server is the entity that it claims to be in the certificate.
- If an impostor copies and sends the certificate to the client, pretending that it is the server claimed in the certificate, it cannot sign the second message because it does not have the private key.

❑ Fixed DH

- In this method, the server sends an RSA or DSS digital signature certificate that includes its registered DH half-key.
- The second message is empty.
- The certificate is signed by the CA's private key and can be verified by the client using the CA's public key. In other words, the CA is authenticated to the client and the CA claims that the half-key belongs to the server.

Phase III: Client Key Exchange and Authentication

Phase III is designed to authenticate the client. Up to three messages can be sent from the client to the server



❑ Certificate

- To certify itself to the server, the client sends a Certificate message. Note that the format is the same as the Certificate message sent by the server in Phase II, but the contents are different.
- It includes the chain of certificates that certify the client. This message is sent only if the server has requested a certificate in Phase II. If there is a request and the client has no certificate to send, it sends an Alert message (part of the Alert Protocol to be discussed later) with a warning that there is no certificate.
- The server may continue with the session or may decide to abort.

❑ ClientKeyExchange

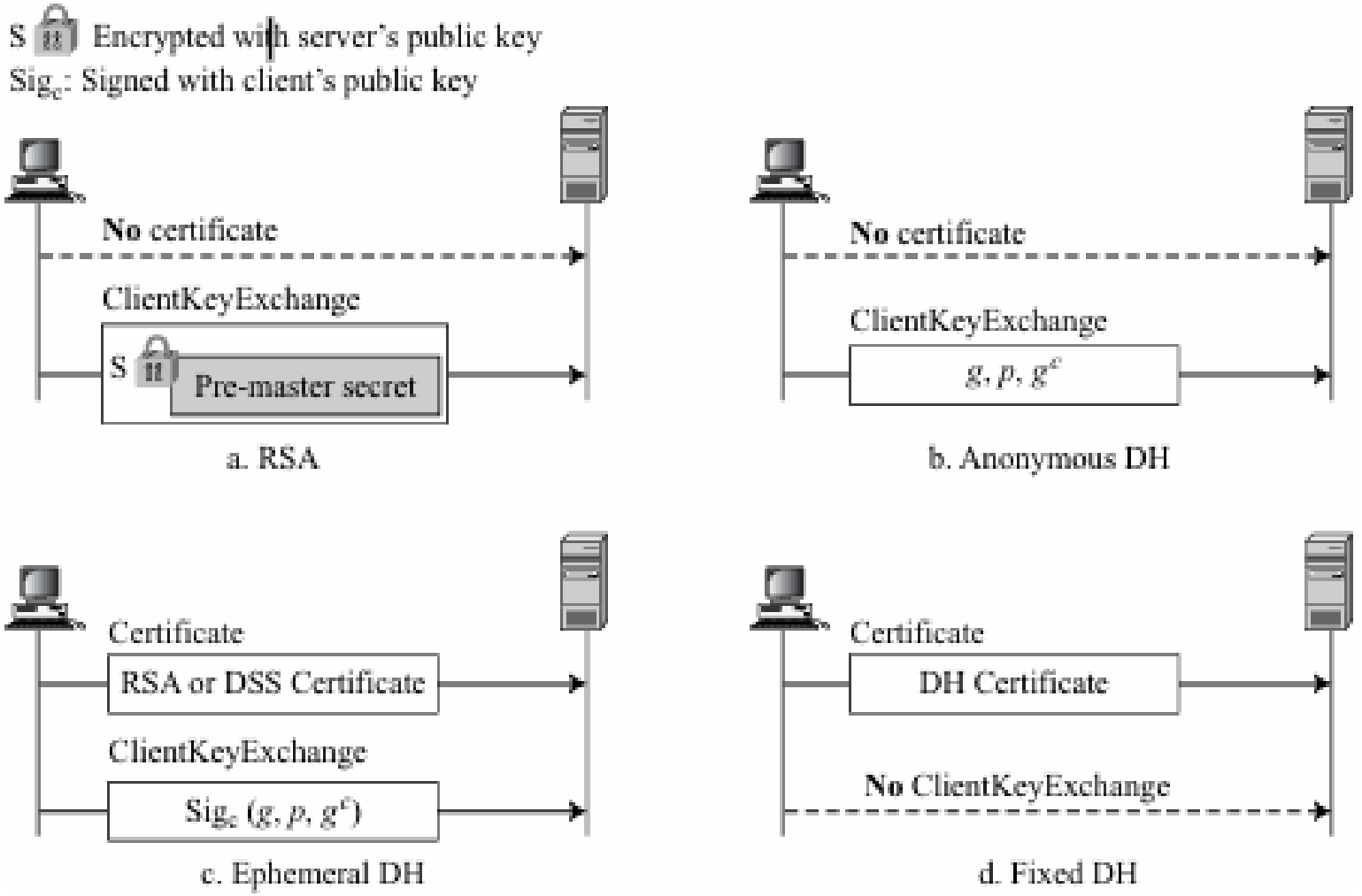
- After sending the Certificate message, the client sends a Client KeyExchange message, which includes its contribution to the pre-master secret.
- The contents of this message are based on the key-exchange algorithm used. If the method is RSA, the client creates the entire pre-master secret and encrypts it with the RSA public key of the server. If the method is anonymous or ephemeral Diffie-Hellman, the client sends its Diffie-Hellman half-key.
- If the method is Fortezza, the client sends the Fortezza parameters.
- The contents of this message are empty if the method is fixed Diffie-Hellman.

❑ CertificateVerify

- If the client has sent a certificate declaring that it owns the public key in the certificate, it needs to prove that it knows the corresponding private key.
- This is needed to thwart an impostor who sends the certificate and claims that it comes from the client.
- The proof of private-key possession is done by creating a message and signing it with the private key. The server can verify the message with the public key already sent to ensure that the certificate actually belongs to the client. Note that this is possible if the certificate has a signing capability; a pair of keys, public and private, is involved.
- The certificate for fixed Diffie-Hellman cannot be verified this way.

Note: After Phase III**,** the client is authenticated for the server and both the client and the server know the pre-master secret.

Four cases in Phase III



❑ RSA.

- In this case, there is o Certificate message unless the server has explicitly requested one in Phase II.
- The ClientKeyExchange method includes the pre-master key encrypted with the RSA public key received in Phase II.

❑ Anonymous DH.

- In this method, there is no Certificate message.
- The server does not have the right to ask for the certificate (in Phase II) because both the client and the server are anonymous.
- In the ClientKeyExchange message, the server sends the Diffie-Hellman parameters and its half-key. Note that the client is not authenticated to the server in this method.

❑ Ephemeral DH

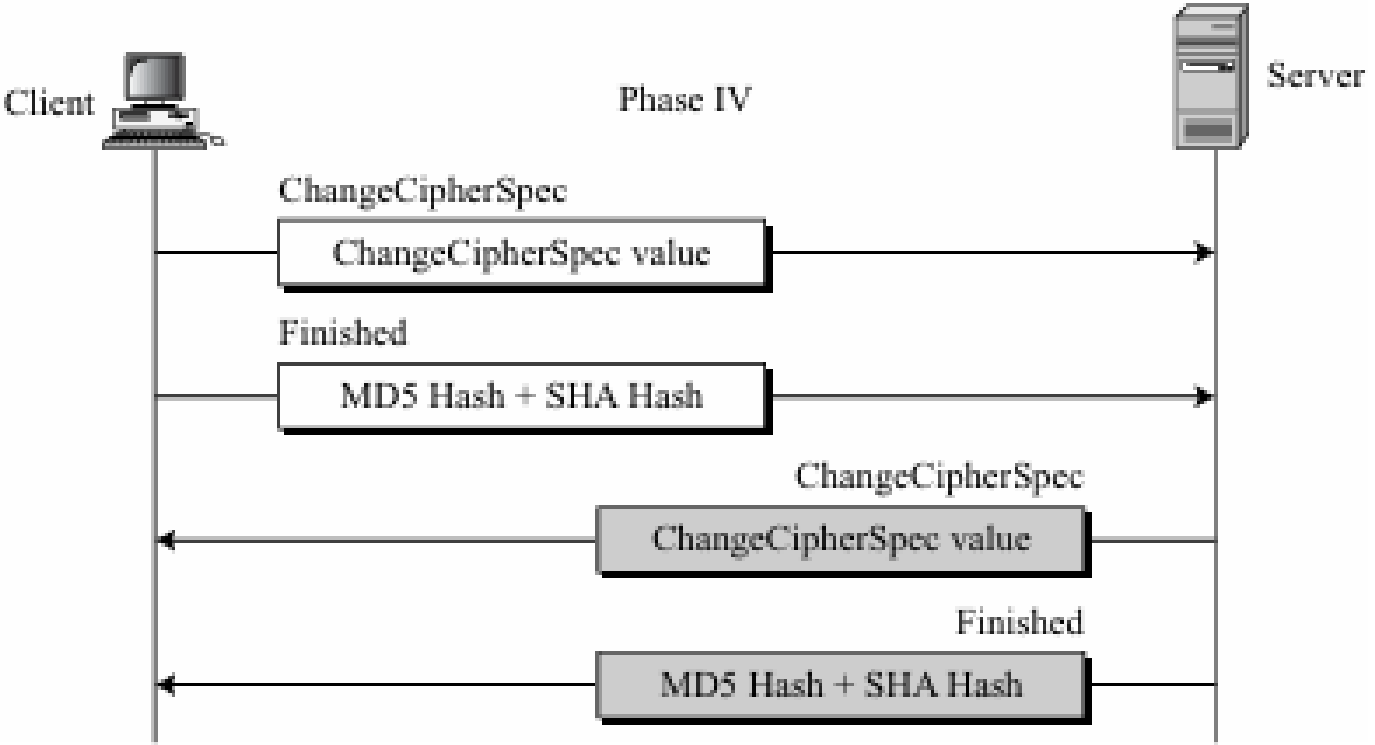
- In this method, the client usually has a certificate.
- The server needs to send its RSA or DSS certificate (based on the agreed-upon cipher set). In the ClientKeyExchange message, the client signs the DH parameters and its half key and sends them.
- The client is authenticated to the server by signing the second message. If the client does not have the certificate, and the server asks for it, the client sends an Alert message to warn the client. If this is acceptable to the server, the client sends the DH parameters and key in plaintext.
- The client is not authenticated to the server in this situation.

❑ Fixed DH.

- In this method, the client usually sends a DH certificate in the first message. Note that the second message is empty in this method.
- The client is authenticated to the server by sending the DH certificate.

Phase IV: Finalizing and Finishing

In Phase IV, the client and server send messages to change cipher specification and to finish the handshaking protocol. Four messages are exchanged in this phase



- **ChangeCipherSpec**

  The client sends a ChangeCipherSpec message to show that it has moved all of the cipher suite set and the parameters from the pending state to the active state.

- **Finished**

  The next message is also sent by the client. It is a Finished message that announces the end of the handshaking protocol by the client.

- **ChangeCipherSpec**

  The server sends a ChangeCipherSpec message to show that it has also moved all of the cipher suite set and parameters from the pending state to the active state.

- **Finished**

  Finally, the server sends a Finished message to show that handshaking is totally completed.

After Phase IV, the client and server are ready to exchange data**.**

ChangeCipherSpec Protocol

- The cipher suite and the generation of cryptographic secrets are formed gradually during the Handshake Protocol.

  The question now is: When can the two parties use these parameter secrets?

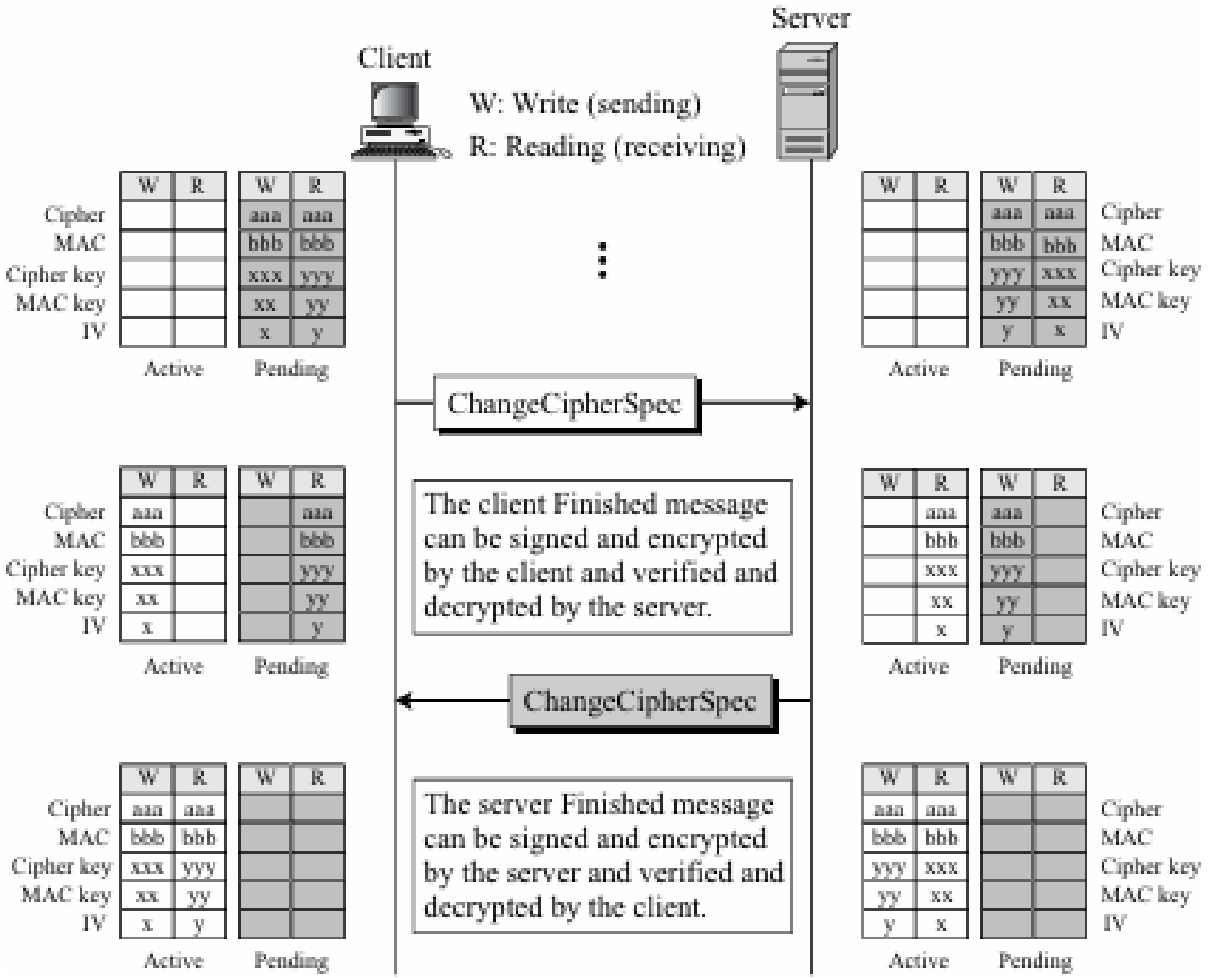
SSL mandates that the parties cannot use these parameters or secrets until they have sent or received a special message, the ChangeCipherSpec message, which is exchanged during the Handshake protocol and defined in the ChangeCipherSpec Protocol.

Note : Reason is that the issue is not just sending or receiving a message.

The sender and the receiver need two states, not one. One state, the pending state, keeps track of the parameters and secrets. The other state, the active state, holds parameters and secrets used by the Record Protocol to sign/verify or encrypt/decrypt messages. In addition, each state holds two sets of values: read (inbound) and write (outbound).

- The ChangeCipherSpec Protocol defines the process of moving values between the pending and active states.

Movement of parameters from pending state to active state



- First the client sends a ChangeCipherSpec message.
- After the client sends this message, it moves the write (outbound) parameters from pending to active.
- The client can now use these parameters to sign or encrypt outbound messages.
- After the receiver receives this message, it moves the read (inbound) parameters from the pending to the active state.
- Now the server can verify and decrypt messages.
- This means that the Finished message sent by the client can be signed and encrypted by the client and verified and decrypted by the server.
- The server sends the ChangeCipherSpec message after receiving the Finish message from the client.

- After sending this message it moves the write (outbound) parameters from pending to active.
- The server can now use these parameters to sign or encrypt outbound messages. After the client receives this message, it moves the read (inbound) parameters from the pending to the active state.
- Now the client can verify and decrypt messages.
- After the exchange completion, both parties can communicate in both directions using the read/write active parameters.

Alert Protocol

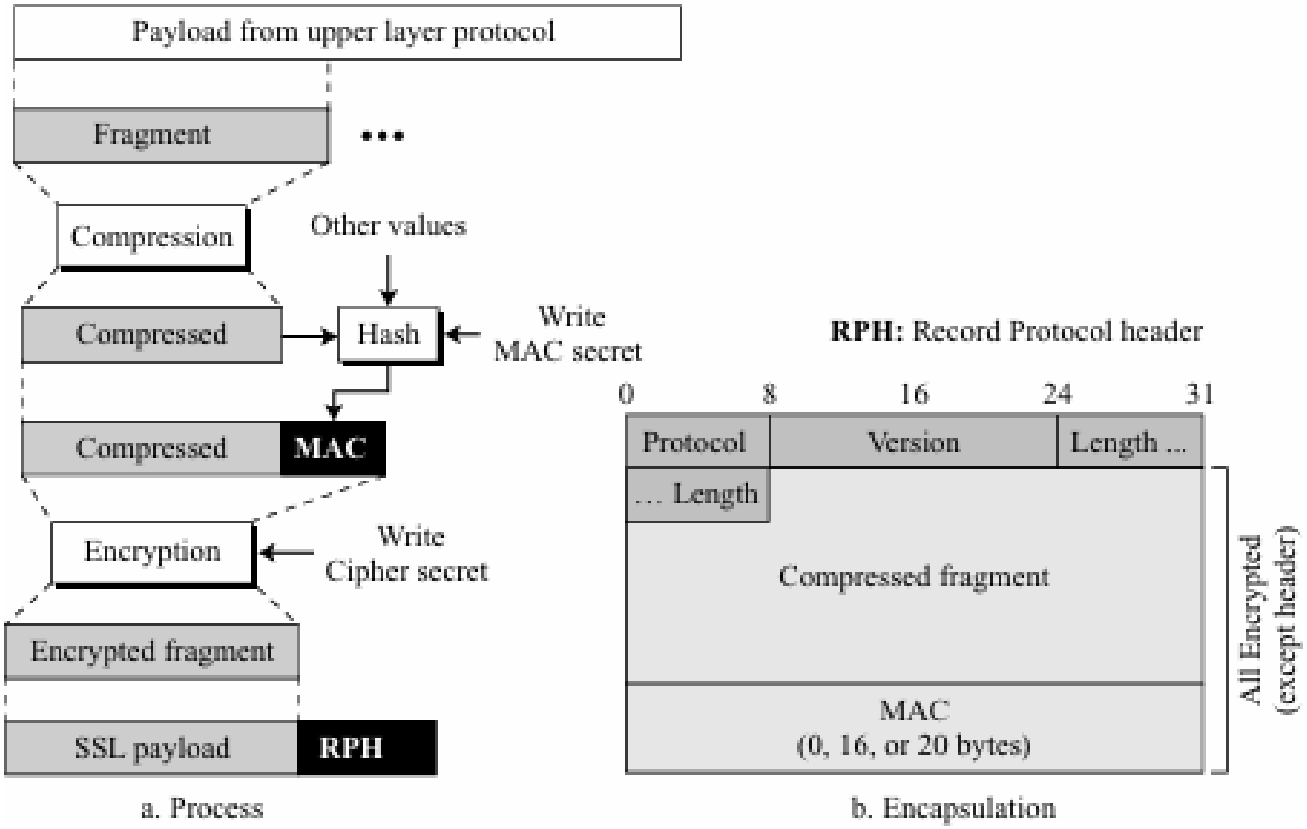
SSL uses the Alert Protocol for reporting errors and abnormal conditions. It has only one message type, the Alert message, that describes the problem and its level (warning or fatal).

Value	*Description*	*Meaning*
0	*CloseNotify*	Sender will not send any more messages.
10	*UnexpectedMessage*	An inappropriate message received.
20	*BadRecordMAC*	An incorrect MAC received.
30	*DecompressionFailure*	Unable to decompress appropriately.
40	*HandshakeFailure*	Sender unable to finalize the handshake.
41	*NoCertificate*	Client has no certificate to send.
42	*BadCertificate*	Received certificate corrupted.
43	*UnsupportedCertificate*	Type of received certificate is not supported.
44	*CertificateRevoked*	Signer has revoked the certificate.
45	*CertificateExpired*	Certificate expired.
46	*CertificateUnknown*	Certificate unknown.
47	*IllegalParameter*	An out-of-range or inconsistent field.

Record Protocol

- The Record Protocol carries messages from the upper layer (Handshake Protocol, ChangeCipherSpec Protocol, Alert Protocol, or application layer).
- The message is fragmented and optionally compressed; a MAC is added to the compressed message using the negotiated hash algorithm.
- The compressed fragment and the MAC are encrypted using the negotiated encryption algorithm.
- Finally, the SSL header is added to the encrypted message.

Processing done by the Record Protocol



Note-This process can only be done when the cryptographic parameters are in the active state. Messages sent before the movement from pending to active are neither signed nor encrypted

❑ Fragmentation/Combination

- At the sender, a message from the application layer is fragmented into blocks of 2^14 bytes, with the last block possibly less than this size.
- At the receiver, the fragments are combined together to make a replica of the original message.

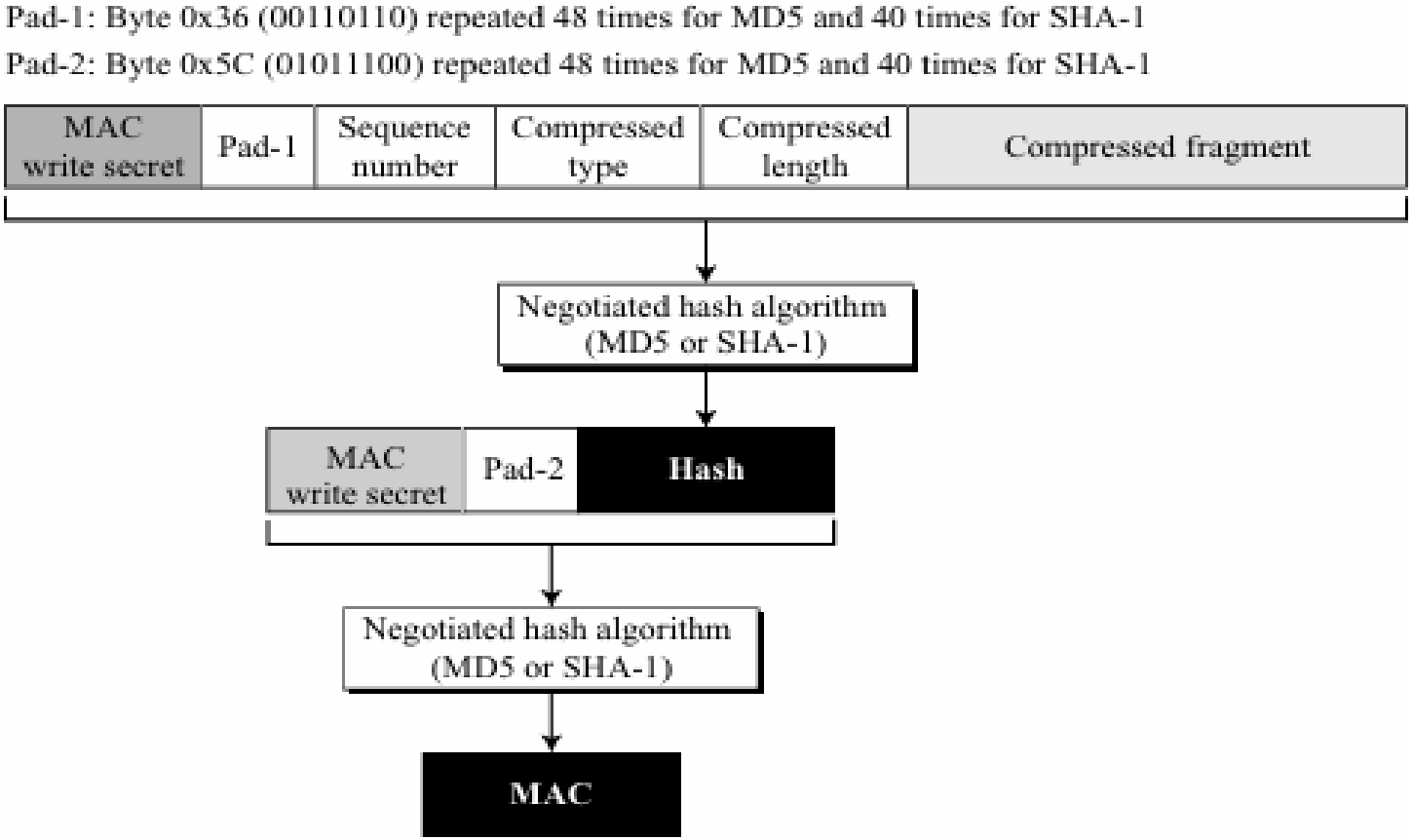
❑ Compression/Decompression

- At the sender, all application layer fragments are compressed by the compression method negotiated during the handshaking. The compression method needs to be loss less (the decompressed fragment must be an exact replica of the original fragment). The size of the fragment must not exceed 1024 bytes. Some compression methods work only on a predefined block size and if the size of the block is less than this, some pad ding is added. Therefore, the size of the compressed fragment may be greater than the size of the original fragment.

- At the receiver, the compressed fragment is decompressed to create a replica of the original. If the size of the decompressed fragment exceeds 2^14 , a fatal decompression Alert message is issued.

Note: Compression/Decompression is optional in SSL.

- **Signing/Verifying**:- At the sender, the authentication method defined during the handshake (NULL, MD5, or SHA-1) creates a signature (MAC)



- **The hash algorithm** is applied twice.

- First, a hash is created from the concatenations of the following values:

a. The MAC write secret (authentication key for the outbound message)
b. Pad-1, which is the byte 0x36 repeated 48 times for MD5 and 40 times for SHA-1
c. The sequence number for this message
d. The compressed type, which defines the upper-layer protocol that provided the compressed fragment
e. The compressed length, which is the length of the compressed fragment
f. The compressed fragment itself

- Second, **the final hash (MAC)** is created from the concatenation of the following values:

a. The MAC write secret
b. Pad-2, which is the byte 0x5C repeated 48 times for MD5 and 40 times for SHA-1
c. The hash created from the first step

At the receiver, the verifying is done by calculating a new hash and comparing it to the received hash.

- **Encryption/Decryption:-** At the sender, the compressed fragment and the hash are encrypted using the cipher write secret. At the receiver, the received message is decrypted using the cipher read secret. For block encryption, padding is added to make the size of the encryptable message a multiple of the block size.

- **Framing/Deframing:-** After the encryption, the Record Protocol header is added at the sender. The header is removed at the receiver before decryption.

SSL MESSAGE FORMATS

Record Protocol general header

- **Protocol**. This 1-byte field defines the source or destination of the encapsulated message. It is used for multiplexing and demultiplexing. The values are 20 (ChangeCipherSpec Protocol), 21 (Alert Protocol), 22 (Handshake Protocol), and 23 (data from the application layer).
- **Version.** This 2-byte field defines the version of the SSL; one byte is the major version and the other is the minor. The current version of SSL is 3.0 (major 3 and minor 0).
- **Length**. This 2-byte field defines the size of the message (without the header) in bytes

ChangeCipherSpec Protocol

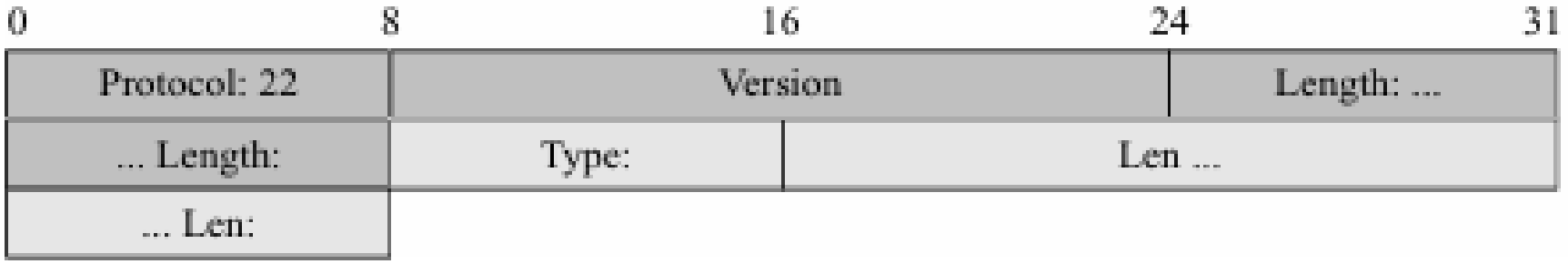
- The one-byte field in the message is called the CCS and its value is currently 1.

Alert Protocol

- **Level.** This one-byte field defines the level of the error. Two levels have been defined so far: warning and fatal.
- **Description**. The one-byte description defines the type of error**.**

Handshake Protocol

Generic header for Handshake Protocol



- **Type**. This one-byte field defines the type of message. So far ten types have been defined as listed in next slide
- **Length (Len)**. This three-byte field defines the length of the message (excluding the length of the type and length field).

Note: Why we need two length fields, one in the general Record header and one in the generic header for the Handshake messages ??

Record message may carry two Handshake messages at the same time if there is no need for another message in between.

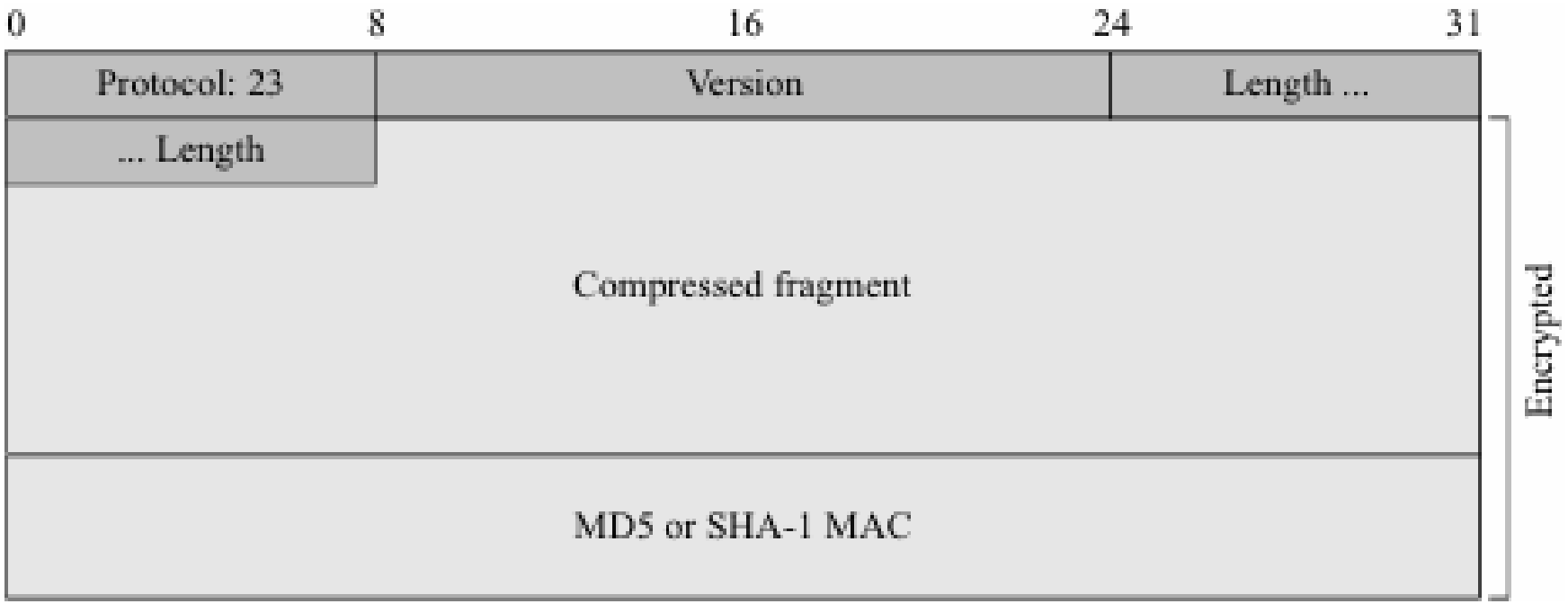
Types of Handshake messages

Type	*Message*
0	HelloRequest
1	ClientHello
2	ServerHello
11	Certificate
12	ServerKeyExchange
13	CertificateRequest
14	ServerHelloDone
15	CertificateVerify
16	ClientKeyExchange
20	Finished

TRANSPORT LAYER SECURITY

- The Transport Layer Security (TLS) protocol is the IETF standard version of the SSL protocol. The two are very similar, with slight differences. Instead of describing TLS in full, we highlight the differences between TLS and SSL protocols.

Record Protocol message for application data



Version Encrypted The first difference is the version number (major and minor). The current version of SSL is 3.0; the current version of TLS is 1.0. In other words, SSLv3.0 is compatible with TLSv1.0.

Cipher Suite Another minor difference between SSL and TLS is the lack of support for the Fortezza method. TLS does not support Fortezza for key exchange or for encryption/decryption.

Cipher suite	*Exchange*	*Encryption*	*Hash*
TLS_NULL_WITH_NULL_NULL	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	RSA	NULL	MD5
TLS_RSA_WITH_NULL_SHA	RSA	NULL	SHA-1
TLS_RSA_WITH_RC4_128_MD5	RSA	RC4	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RC4	SHA-1
TLS_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA	SHA-1
TLS_RSA_WITH_DES_CBC_SHA	RSA	DES	SHA-1
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES	SHA-1
TLS_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4	MD5
TLS_DH_anon_WITH_DES_CBC_SHA	DH_anon	DES	SHA-1
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES	SHA-1
TLS_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES	SHA-1
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES	SHA-1
TLS_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES	SHA-1
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES	SHA-1
TLS_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES	SHA-1
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES	SHA-1
TLS_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES	SHA-1
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES	SHA-1

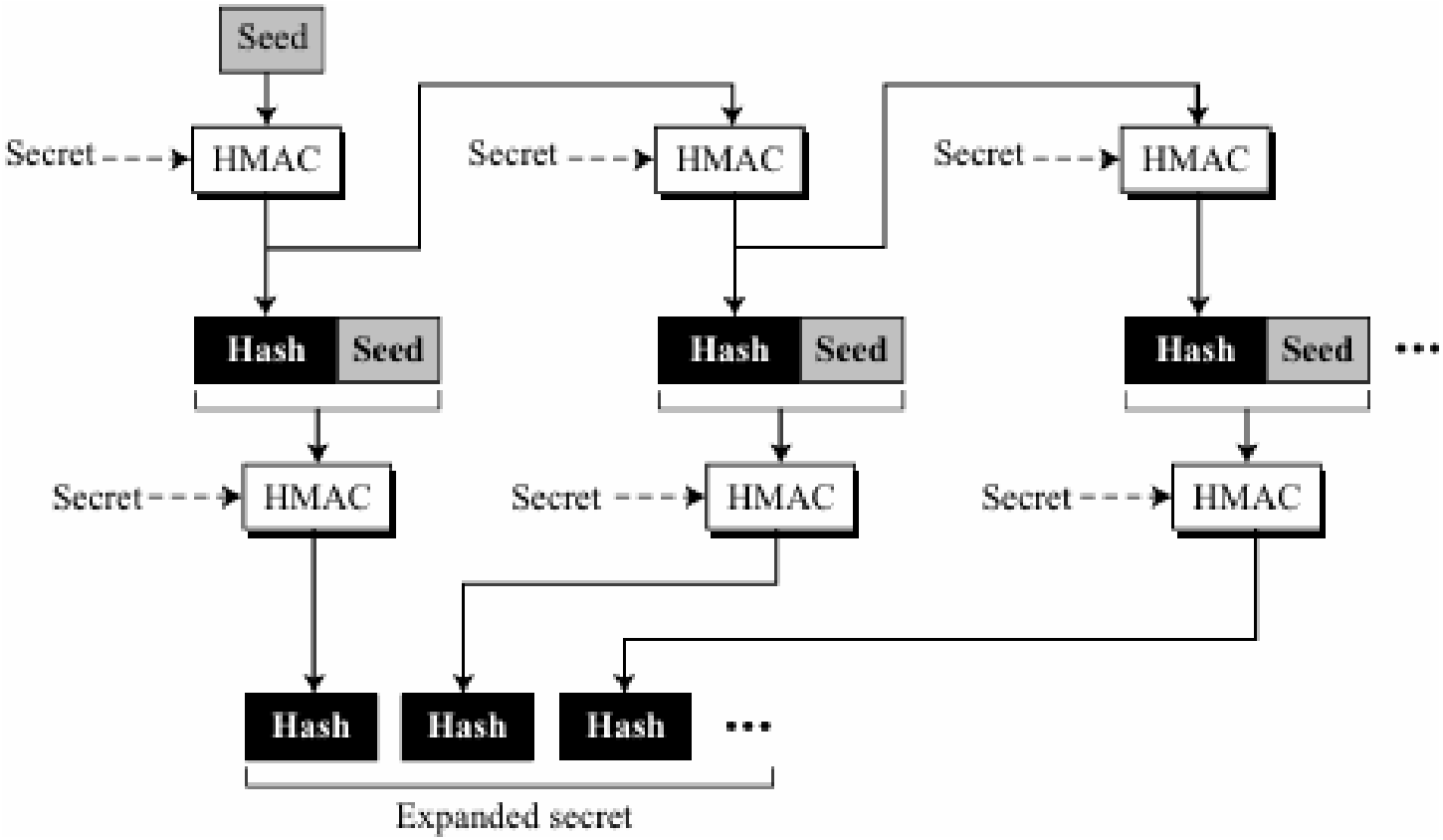
Generation of Cryptographic Secrets

- The generation of cryptographic secrets is more complex in TLS than in SSL. TLS first defines two functions: the data-expansion function and the pseudorandom function.

1.Data-Expansion Function

- The data-expansion function uses a predefined HMAC (either MD5 or SHA-1) to expand a secret into a longer one. This function can be considered a multiple section function, where each section creates one hash value. The extended secret is the concatenation of the hash values. Each section uses two HMACs, a secret and a seed. The data-expansion function is the chaining of as many sections as required. However, to make the next section dependent on the previous, the second seed is actually the out put of the first HMAC of the previous section

Data-expansion function

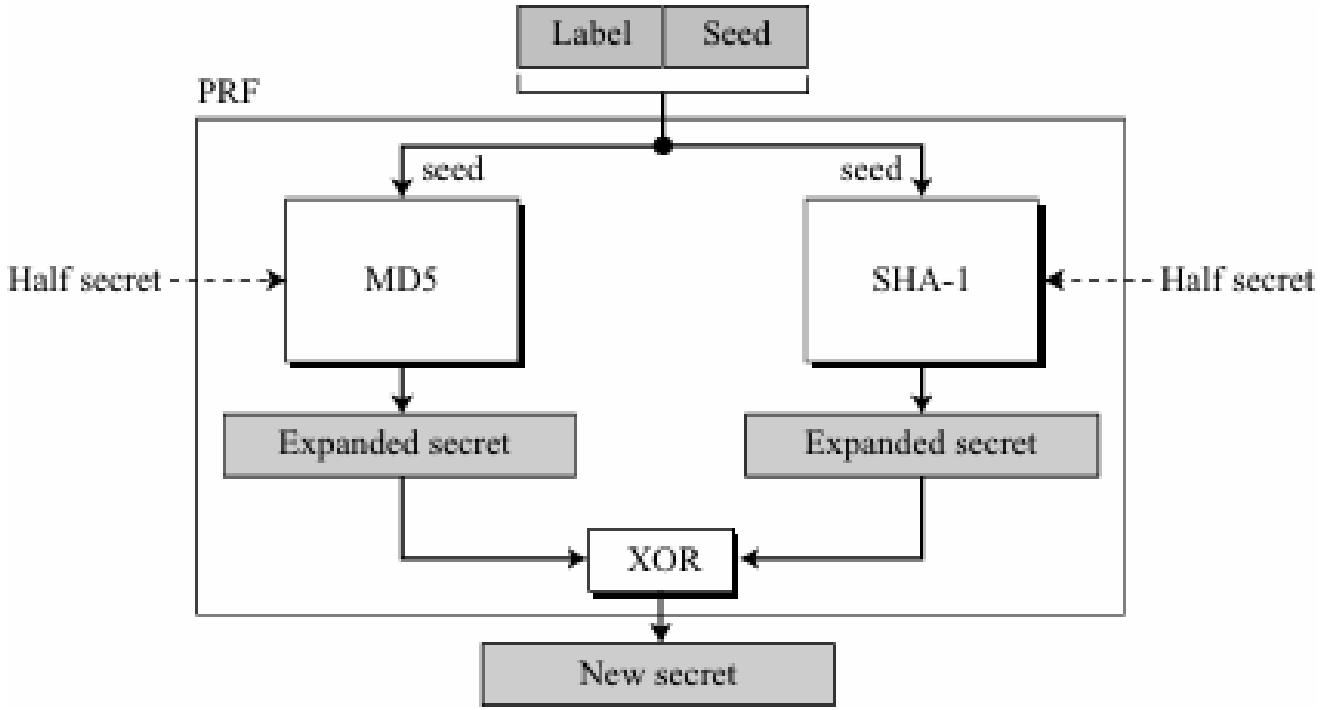


2.Pseudorandom Function (PRF)

- TLS defines a pseudorandom function (PRF) to be the combination of two data-expansion functions, one using MD5 and the other SHA-1. PRF takes three inputs, a secret, a label, and a seed. The label and seed are concatenated and serve as the seed for each data expansion function. The secret is divided into two halves; each half is used as the secret for each data-expansion function. The output of two data-expansion functions is exclusive ored together to create the final expanded secret.

- Note that because the hashes created from MD5 and SHA-1 are of different sizes, extra sections of MD5-based functions must be created to make the two outputs the same size.

PRF

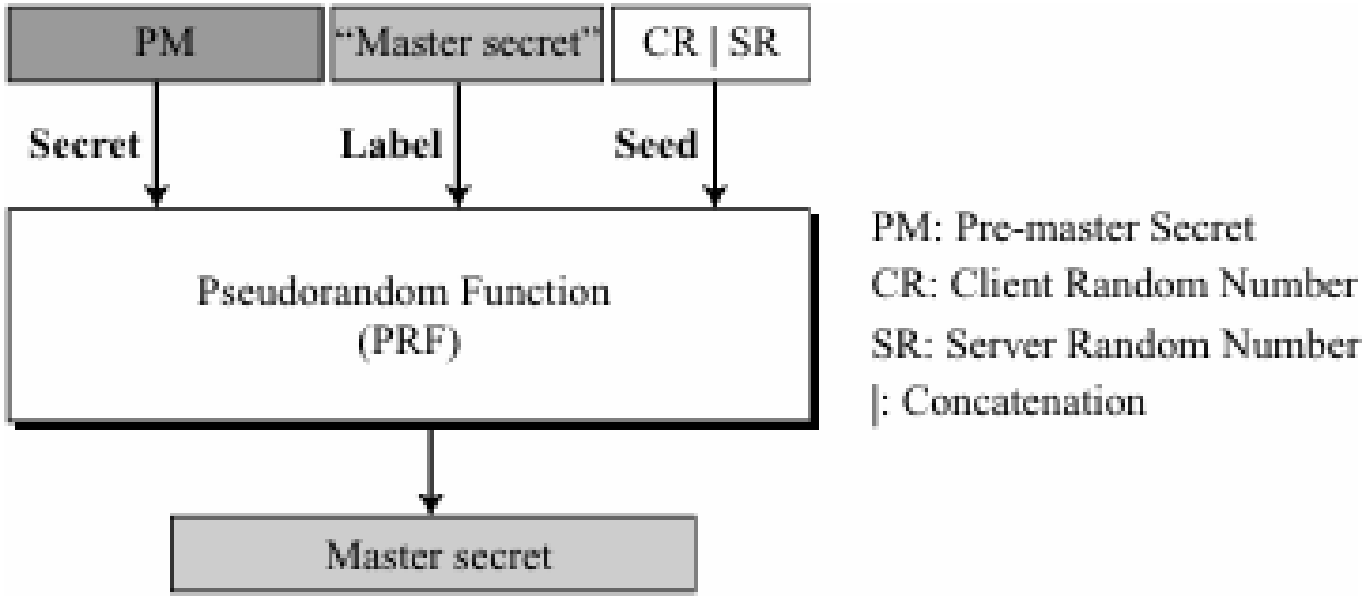


Pre-master Secret The generation of the pre-master secret in TLS is exactly the same as in SSL.

Master Secret

- TLS uses the PRF function to create the master secret from the pre-master secret. This is achieved by using the pre-master secret as the secret, the string "master secret" as the label, and concatenation of the client random number and server random number as the seed.

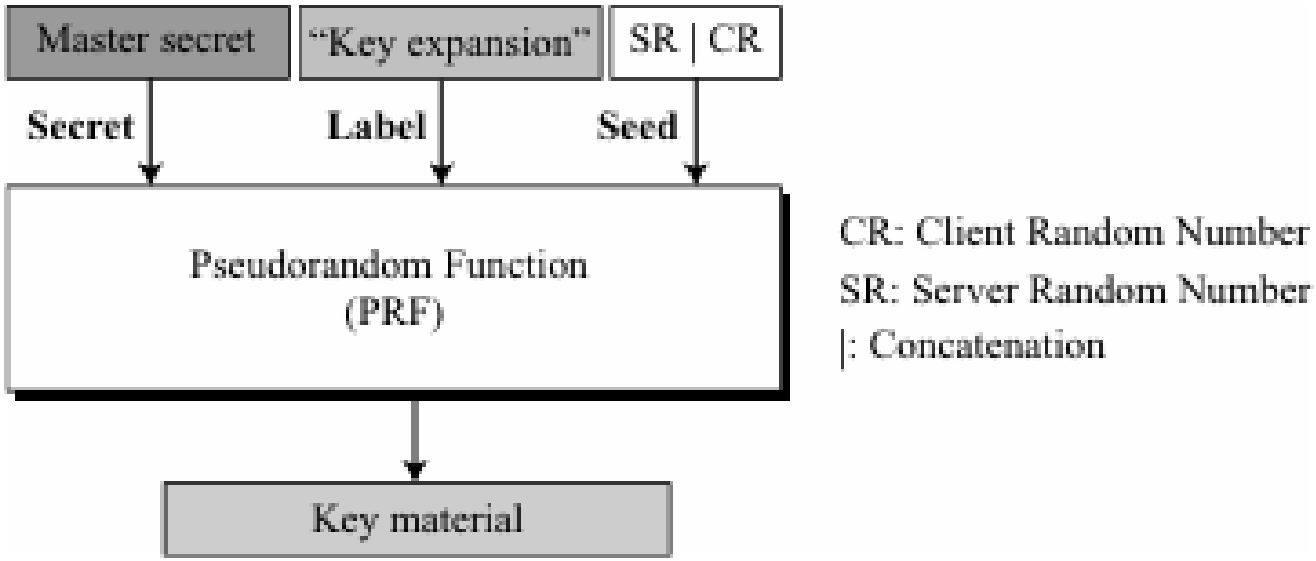
- Note that the label is actually the ASCII code of the string "master secret". In other words, the label defines the output (to be created), the master secret.



Key Material

TLS uses the PRF function to create the key material from the master secret. This time the secret is the master secret, the label is the string “key expansion”, and the seed is the concatenation of the server random number and the client random number,

Key material generation



Alert Protocol

TLS supports all of the alerts defined in SSL except for NoCertificate. TLS also adds some new ones to the list

Value	*Description*	*Meaning*
0	*CloseNotify*	Sender will not send any more messages.
10	*UnexpectedMessage*	An inappropriate message received.
20	*BadRecordMAC*	An incorrect MAC received.
21	*DecryptionFailed*	Decrypted message is invalid.
22	*RecordOverflow*	Message size is more than $2^{14} + 2048$.
30	*DecompressionFailure*	Unable to decompress appropriately.
40	*HandshakeFailure*	Sender unable to finalize the handshake.
42	*BadCertificate*	Received certificate corrupted.
43	*UnsupportedCertificate*	Type of received certificate is not supported.
44	*CertificateRevoked*	Signer has revoked the certificate.
45	*CertificateExpired*	Certificate has expired.
46	*CertificateUnknown*	Certificate unknown.
47	*IllegalParameter*	A field out of range or inconsistent with others.
48	*UnknownCA*	CA could not be identified.

Alerts defined for TLS (continued)

Value	*Description*	*Meaning*
49	*AccessDenied*	No desire to continue with negotiation.
50	*DecodeError*	Received message could not be decoded.
51	*DecryptError*	Decrypted ciphertext is invalid.
60	*ExportRestriction*	Problem with U.S. restriction compliance.
70	*ProtocolVersion*	The protocol version is not supported.
71	*InsufficientSecurity*	More secure cipher suite needed.
80	*InternalError*	Local error.
90	*UserCanceled*	The party wishes to cancel the negotiation.
100	*NoRenegotiation*	The server cannot renegotiate the handshake.

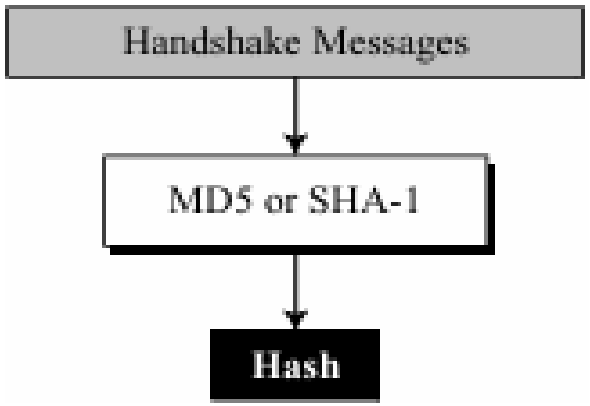
Handshake Protocol

- TLS has made some changes in the Handshake Protocol. Specifically, the details of the CertificateVerify message and the Finished message have been changed.

CertificateVerify Message

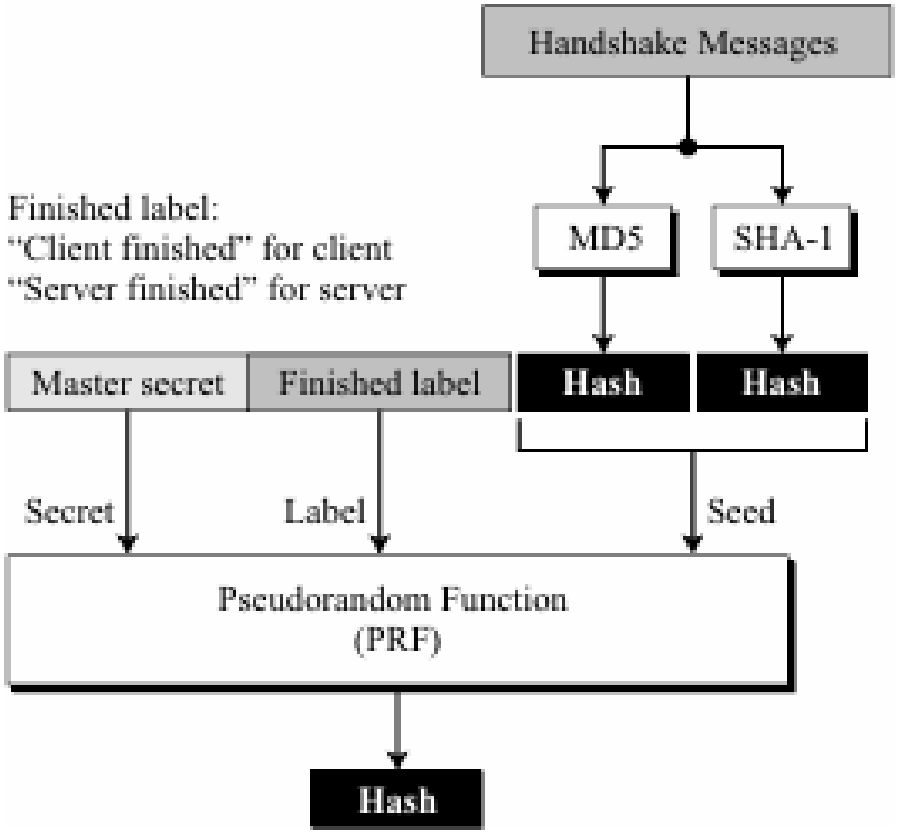
- In SSL, the hash used in the CertificateVerify message is the two-step hash of the hand shake messages plus a pad and the master secret.
- TLS has simplified the process. The hash in the TLS is only over the handshake messages,

Hash for CertificateVerify message in TLS



- **Finished Message** The calculation of the hash for the Finished message has also been changed. TLS uses the PRF to calculate two hashes used for the Finished message.

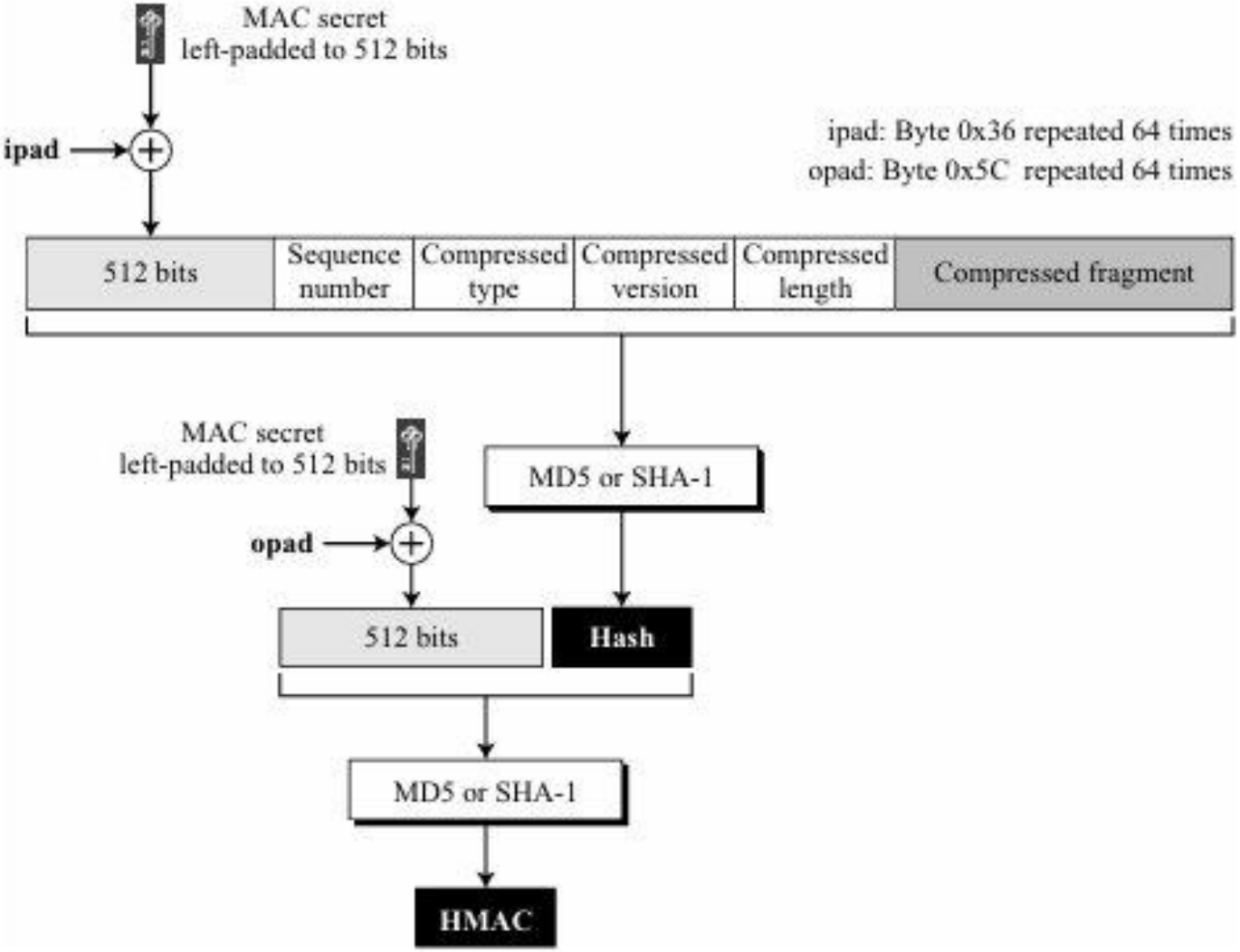
Hash for Finished message in TLS



Record Protocol

- The only change in the Record Protocol is the use of HMAC for signing the message. TLS uses the MAC, as defined in Chapter 11, to create the HMAC. TLS also adds the protocol version (called Compressed version) to the text to be signed.

HMAC for TLS


MAC secret
left-padded to 512 bits
ipad
ipad: Byte 0x36 repeated 64 times
opad: Byte 0x5C repeated 64 times
512 bits
Sequence number
Compressed type
Compressed version
Compressed length
Compressed fragment
MAC secret
left-padded to 512 bits
MD5 or SHA-1
opad
512 bits
Hash
MD5 or SHA-1
HMAC

Thank You