UNIT I NETWORK LAYER SECURITY &TRANSPORT LAYER SECURITY

IPSec Protocol – IP Authentication Header – IP ESP – Key Management Protocol for IPSec. Transport layer Security: SSL protocol, Cryptographic Computations – TLS Protocol.

Part-A

1.**Define IPsec Protocol.**

The IPsec protocol is a set of security extensions developed by the IETF and it provides Privacy and authentication services at the IP layer by using modern cryptography. To protect the contents of an IP datagram, the data is transformed using encryption algorithms.

There are two main transformation types that form the basics of IPsec,

1. The Authentication Header (AH).
2. The Encapsulating Security Payload (ESP).

- ✓ Both AH and ESP are two protocols that provide connectionless integrity, data origin authentication, confidentiality and an anti-replay service.

2**. write the basic components of IPsec architecture Protocol.**

The basic components of the IPsec security architecture are explained in terms of the following functionalities:

- Security Protocols for AH and ESP
- Security Associations for policy management and traffic processing
- Manual and automatic key management for the Internet Key Exchange (IKE), the Oakley key determination protocol and ISAKMP.
- Algorithms for authentication and encryption

3.Define IPsec Protocol Documents

The seven-group documents describing the set of IPsec protocols are explained in the following:

***Architecture*:**

The main architecture document covers the general concepts, security Requirements, definitions and mechanisms defining IPsec technology.

ESP:

This document covers the packet format and general issues related to the use of the ESP for packet encryption and optional authentication.

***AH*:**

This document covers the packet format and general issue related to the use of

AH for packet authentication.

Encryption algorithm:

This is a set of documents that describe how various encryptions algorithms are used for ESP.

***Authentication algorithm*:**

This is a set of documents that describe how various authentication algorithms are used for AH and for the authentication option of ESP.

***Key management*:**

This is a set of documents that describe key management schemes. These documents also provide certain values for the DOI. Currently the key management represents the Oakley, ISAKMP and Resolution protocols.

***DOI* :**

This document contains values needed for the other documents to relate each other. These include identifiers for approved encryption and authentication algorithms, as well as operational parameters such as key lifetime.

4. Define Security Associations (SAs)

An SA is a simplex connection between a sender and receiver that affords security services to the traffic carried on it. If both AH and ESP protection are applied to a traffic stream, then two SAs are required for two-way secure exchange.

An SA is uniquely identified by three parameters as follows:

- ***Security Parameters Index (SPI)***
- ***IP Destination Address***
- ***Security Protocol Identifier***

5. **Define Hashed Message Authentication Code (HMAC)**

- ✓ A mechanism that provides a data integrity check based on a secret key is usually called the Message Authentication Code (MAC).
- ✓ An HMAC mechanism can be used with any iterative hash functions in combination with a secret key.
- ✓ MACs are used between two parties (e.g. client and server) that share a secret key in order to validate information transmitted between them. An MAC mechanism based on a cryptographic hash function is called HMAC. MD5 and SHA-1 are examples of such hash functions. HMAC uses a secret key for computation and verification of the message authentication values.

6. Define IP Authentication Header.

- ✓ The IP AH is used to provide data integrity and authentication for IP packets.

- ✓ It also provides protection against replays. The AH provides authentication for the IP header, as well as for upper-level protocol (TCP, UDP) data.
- ✓ But some IP header fields may change in transit and the sender may not be able to predict the value of these fields when the packet arrives at the receiver.
- ✓ The AH can be used in conjunction with ESP or with the use of tunnel mode. Security services can be provided between a pair of hosts, between a pair of security gateway or between a security gateway and a host.

7. **Draw AH Format.**

INTERNET SECURITY

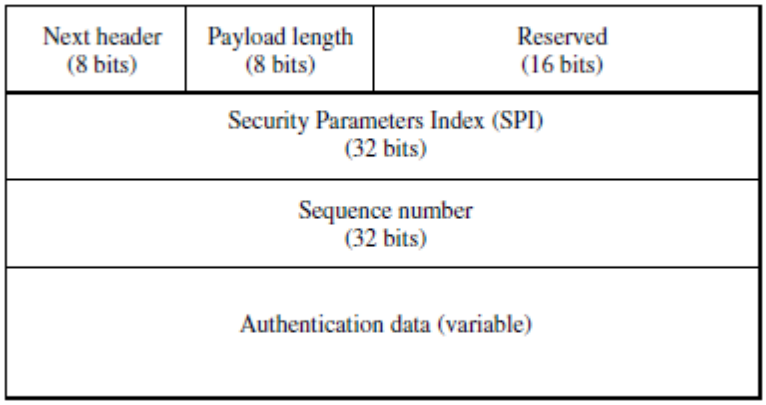


Figure 7.4 IPsec AH format.

8. **Define IP ESP.**
   - ✓ The ESP header is designed to provide security services in IPv4 and IPv6. ESP can be applied alone, in combination with the IP AH or through the use of tunnel mode.
   - ✓ Security services are provided between a pair of hosts, between a pair of security gateways or between a security gateway and a host.
   - ✓ The ESP header is inserted after the IP header and before the upper-layer protocol header (transport mode) or before an encapsulated IP header (tunnel mode).
   - ✓ ESP is used to provide confidentiality (encryption), data authentication, integrity and anti-replay service, and limited traffic flow confidentiality. Confidentiality could be selected independent of all other services.

9. **Define Packet Format**

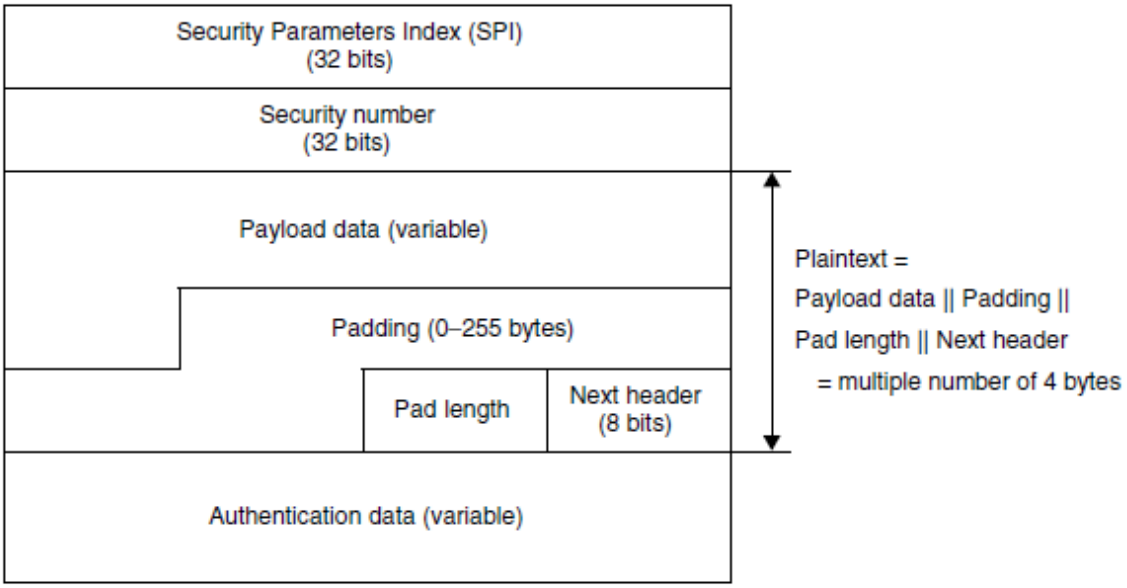


Figure 7.6 IPsec ESP format.

8. Define Key Management Protocol for IPSec.

- ✓ The key management mechanism of IPsec involves the determination and distribution of a secret key. Key establishment is at the heart of data protection that relies on cryptography.
- ✓ A secure key distribution for the Internet is an essential part of packet protection.
- ✓ Prior to establishing a secure session, the communicating parties need to negotiate the terms that are defined in the SA. An automated protocol is needed in order to establish the SAs for making the process feasible on the Internet. This automated process is the IKE.
- ✓ IKE combines ISAKMP with the Oakley key exchange. We begin our discussion with an overview of Oakley and then look at ISAKMP.

1. **OAKLEY Key Determination Protocol**
2. **ISAKMP**

9. **List out the different types of Payload Types for ISAKMP.**

1. Security Association Payload
2. Proposal Payload
3. Transform Payload
4. Key Exchange Payload
5. Identification Payload

6. Certificate Payload
7. Certificate Request Payload
8. Hash Payload
9. Signature Payload
10. Nonce Payload
11. Notification Payload
12. Delete Payload
13. Vendor ID Payload

10. Define SSL Protocol.

- SSL is a layered protocol. It is not a single protocol but rather two layers of protocols.
- At the lower level, the SSL Record Protocol is layered on top of some reliable transport protocol such as TCP.
- The SSL Record Protocol is also used to encapsulate various higher level protocols. A higher-level protocol can layer on top of the SSL protocol transparently.

11. Draw the SSL Protocol Overview stack.

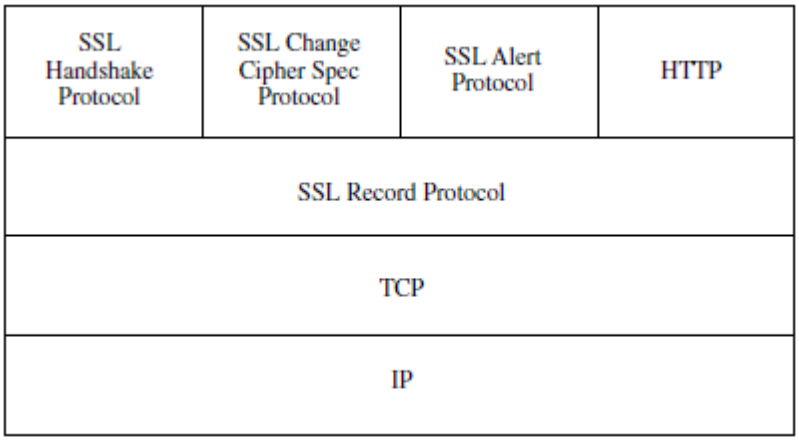


Figure 8.1 Two-layered SSL protocols.

12. Difference between SSL Session and SSL Connection.

SSL Session	***SSL Connection***
An SSL session is an association between a client and a server.	A connection is a transport (in the OSI layering model definition) that provides a suitable type of service.
They define a set of cryptographic security parameters, which can be shared among	For SSL, such connections are peer-to-peer relationships.

multiple connections.	
Sessions are created by the Handshake Protocol.	The connections are transient.
An SSL session coordinates the states of the client and server.	Every connection is associated with one session.

13.List out the SSL session elements.

1. Session identifier
2. Peer certificate
3. Compression method
4. Cipher spec:
5. Master secret
6. Is resumable

14. List out the SSL Connection elements.

1. Server and client random
2. Server write MAC secret
3. Client write MAC secret
4. Server write key
5. Client write key
6. Initialization vectors
7. Sequence numbers

15.Define SSL Record Protocol format.

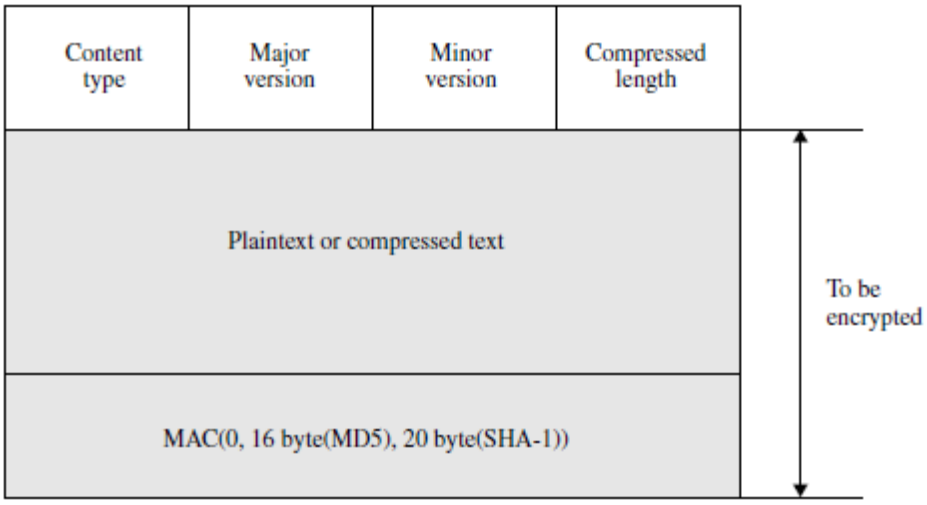


Figure 8.4 SSL Record Protocol format.

16. List out the phases of SSL Handshake Protocol.

- *Phase 1: Hello Messages for Logical Connection*
- Phase 2: Server Authentication and Key Exchange

- *Phase 3: Client Authentication and Key Exchange*
- *Phase 4: End of Secure Connection*

17.How to compute the master-secret for *Diffie–Hellman.*

master_secret = MD5(pre_master_secret||SHA('A'||
pre_master_secret||ClientHello.random||
ServerHello.random))||
MD5(pre_master_secret||SHA('BB'||
pre_master_secret||ClientHello.random||
ServerHello.random))||
MD5(pre_master_secret||SHA('CCC'||
pre_master_secret||ClientHello.random||
ServerHello.random))

18. Define HMAC-Algorithm and how to calculate the HMAC Algorithm.

A Keyed-hashing Message Authentication Code (HMAC) is a secure digest of some data protected by a secret. Forging the HMAC is infeasible without knowledge of the MAC secret. HMAC can be used with a variety of different hash algorithms, namely MD5 and SHA-1, denoting these as HMAC MD5(secret, data) and HMAC SHA-1(secret, data).

$$\mathbf{HMAC} = H[(K \oplus \mathbf{opad})||H[(K \oplus \mathbf{ipad})||M]]$$

where

ipad = 00110110*(*0x36*)* repeated 64 times (512 bits)
opad = 01011100*(*0x5c*)* repeated 64 times (512 bits)
H = one-way hash function for TLS (either MD5 or SHA-1)
M = message input to HMAC
K = padded secret key equal to the block length of the hash code (512 bits for MD5 and SHA-1)

19. List out the ISAKMP Payload Processing.

1. General Message Processing
2. ISAKMP Header Processing
3. Generic Payload Header Processing
4. Security Association Payload Processing
5. Proposal Paylaod Processing
6. Proposal Paylaod Processing
7. Transform Payload Processing
8. Key Exchange Payload Processing

9. Identification Payload Processing
10. Certificate Payload Processing
11. Certificate Request Payload Processing
12. Hash Payload Processing
13. Signature Payload Processing
14. Delete Payload Processing
15. Nonce Payload Processing
16. Notification Payload Processing

20. Define Cryptographic Computations.

- The key exchange, authentication, encryption and MAC algorithms are determined by the cipher suite selected by the server and revealed in the server hello message.
- The compression algorithm is negotiated in the hello messages, and the random values are exchanged in the hello messages.
- The creation of a shared master secret by means of the key exchange and the generation of cryptographic parameters from the master secrete.

Part-B

1. Explain in detail about the IPsec Protocol.

The IPsec protocol is a set of security extensions developed by the IETF and it provides Privacy and authentication services at the IP layer by using modern cryptography. To protect the contents of an IP datagram, the data is transformed using encryption algorithms.

There are two main transformation types that form the basics of IPsec,

3. The Authentication Header (AH).
4. The Encapsulating Security Payload (ESP).

- ✓ Both AH and ESP are two protocols that provide connectionless integrity, data origin authentication, confidentiality and an anti-replay service.
- ✓ These protocols may be applied alone or in combination to provide a desired set of security services for the IP layer.
- ✓ They are configured in a data structure called a Security Association (SA).

The basic components of the IPsec security architecture are explained in terms of the following functionalities:

- Security Protocols for AH and ESP
- Security Associations for policy management and traffic processing

- Manual and automatic key management for the Internet Key Exchange (IKE), the Oakley key determination protocol and ISAKMP.
- Algorithms for authentication and encryption

- ✓ An IPsec implementation operates in a host or a security gateway environment, affording protection to IP traffic.
- ✓ The protection offered is based on requirements defined by a Security Policy Database (SPD) established and maintained by a user or system administrator.

1.1 IPsec Protocol Documents

The seven-group documents describing the set of IPsec protocols are explained in the following:

***Architecture*:**

The main architecture document covers the general concepts, security Requirements, definitions and mechanisms defining IPsec technology.

ESP:

This document covers the packet format and general issues related to the use of the ESP for packet encryption and optional authentication. This protocol document also contains default values if appropriate, and dictates some of the values in the Domain of Interpretation (DOI).

***AH*:**

This document covers the packet format and general issue related to the use of AH for packet authentication. This document also contains default values such as the default padding contents, and dictates some of the values in the DOI document.

Encryption algorithm:

This is a set of documents that describe how various encryptions algorithms are used for ESP. Specifically:

– Specification of the key sizes and strengths for each algorithm.

– Any available estimates on performance of each algorithm.

– General information on how this encryption algorithm is to be used in ESP.

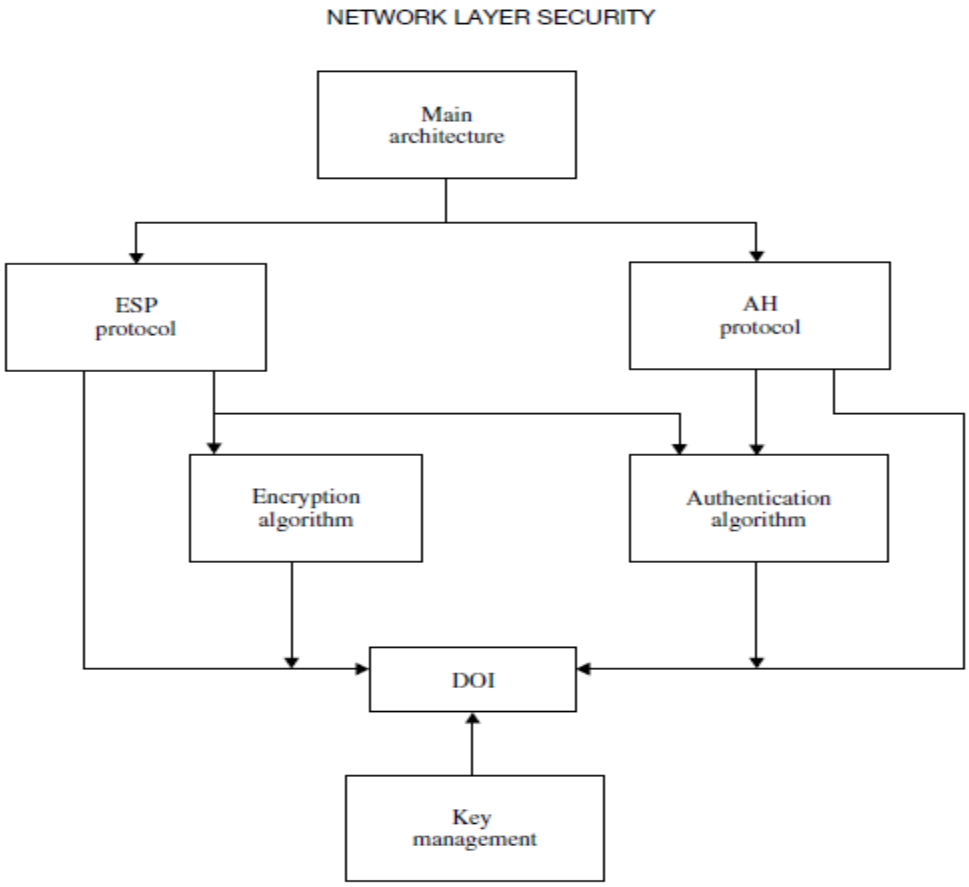


Figure 7.1 Document overview that defines IPsec.

When these encryption algorithms are used for ESP, the DOI document has to indicate certain values, such as an encryption algorithm identifier, so these documents provide input to the DOI.

***Authentication algorithm*:**

This is a set of documents that describe how various authentication algorithms are used for AH and for the authentication option of ESP.

Specifically:

– Specification of operating parameters such as number of rounds, and input or output block format.

– Implicit and explicit padding requirements of this algorithm.

– Identification of optional parameters/methods of operation.

– Defaults and mandatory ranges of the algorithm.

– Authentication data comparison criteria for the algorithm.

***Key management*:**

This is a set of documents that describe key management schemes. These documents also provide certain values for the DOI. Currently the key management represents the Oakley, ISAKMP and Resolution protocols.

***DOI* :**

This document contains values needed for the other documents to relate each other. These include identifiers for approved encryption and authentication algorithms, as well as operational parameters such as key lifetime.

1.2 Security Associations (SAs)

An SA is a simplex connection between a sender and receiver that affords security services to the traffic carried on it. If both AH and ESP protection are applied to a traffic stream, then two SAs are required for two-way secure exchange.

An SA is uniquely identified by three parameters as follows:

***Security Parameters Index (SPI)*:**

- ✓ This is assigned to each SA, and each SA is identified through an SPI.
- ✓ A receiver uses the SPI to identify the security association for a packet.
- ✓ Before a sender uses IPsec to communicate with a receiver, the sender must know the index value for a particular SA.
- ✓ The sender then places the value in the SPI field of each outgoing datagram. The SPI is carried in AH and ESP headers to enable the receiver to select the SA under which a received packet is processed.

IP Destination Address:

- ✓ Because, at present, unicast addresses are only allowed by IPsec SA management mechanisms, this is the address of the destination endpoint of the SA. The destination endpoint may be an end-user system or a network system such as a firewall or router.

***Security Protocol Identifier*:**

- ✓ This identifier indicates whether the association is an AH or ESP security association.

There are two nominal databases in a general model for processing IP traffic relative to SAs, namely, the Security Policy Database (SPD) and the Security Association Database (SAD).

Security policy database

- ✓ The SPD, which is an essential element of SA processing, specifies what services are to be offered to IP datagrams and in what fashion.

- ✓ The SPD is used to control the flow of all traffic (inbound and outbound) through an IPsec system, including security and key management traffic (i.e. ISAKMP). The SPD contains an ordered list of policy entries.
- ✓ Each policy entry is keyed by one or more selectors that define the set of all IP traffic encompassed by this entry.
- ✓ Each entry encompasses every indication mechanism for bypassing, discarding or IPsec processing.
- ✓ The entry for IPsec processing includes SA (or SA bundle) specification, limiting the IPsec protocols, modes and algorithms to be employed.

Security association database

- ✓ The SAD contains parameters that are associated with each security association. Each SA has an entry in the SAD.
- ✓ For outbound processing, entries are pointed to by entries in the SPD. For inbound processing, each entry in the SAD is indexed by a destination IP address, IPsec protocol type and SPI.

Transport mode SA

- ✓ There are two types of SAs to be defined: a transport mode SA and a tunnel mode SA.
- ✓ A transport mode provides protection primarily for upper-layer protocols, i.e. a TCP packet or UDP segment or an Internet Control Message Protocol (ICMP) packet, operating directly above the IP layer.
- ✓ A transport mode SA is a security association between two hosts.
- ✓ When a host runs AH or ESP over IPv4, the payload is the data that normally follows the IP header.
- ✓ For IPv6, the payload is the data that normally follows both the IP header and any IPv6 extension headers.
- ✓ In the case of AH, AH in transport mode authenticates the IP payload and the protection is also extended to selected portions of the IP header, selected portions of IPv6 extension headers and the selected options.
- ✓ In the case of ESP, ESP in transport mode primary encrypts and optionally authenticates the IP payload but not the IP header.
- ✓ A transport mode SA provides security services only for higher-layer protocols, not for the IP header or any extension headers proceeding the ESP header.

Tunnel mode SA

- ✓ Tunnel mode provides protection to the entire IP packet.

- ✓ A tunnel mode SA is essentially an SA applied to an IP tunnel.
- ✓ Whenever either end of an SA is a security gateway, the SA must be tunnel mode, as is an SA between a host and a security gateway.
- ✓ Note that a host must support both transport and tunnel modes, but a security gateway is required to support only tunnel mode.
- ✓ If a security gateway supports transport mode, it should be used as an acting host. But in this case, the security gateway is not as acting a gateway.
- ✓ When the entire inner (original) packet travels through a tunnel from one point of the IP network to another, routers along the path are unable to examine the inner IP header because the original inner packet is encapsulated.

1.3Hashed Message Authentication Code (HMAC)

- ✓ A mechanism that provides a data integrity check based on a secret key is usually called the Message Authentication Code (MAC).
- ✓ An HMAC mechanism can be used with any iterative hash functions in combination with a secret key.
- ✓ MACs are used between two parties (e.g. client and server) that share a secret key in order to validate information transmitted between them.
- ✓ An MAC mechanism based on a cryptographic hash function is called HMAC. MD5 and SHA-1 are examples of such hash functions. HMAC uses a secret key for computation and verification of the message authentication values.
- ✓ The MAC mechanism should allow for easy replacement of the embedded hash function in case faster or more secure hash functions are found or required. HMAC can be proven as secure provided that the underlying hash function has some reasonable cryptographic strengths.

HMAC Structure

- ✓ HMAC is a secret-key authentication algorithm which provides both data integrity and data origin authentication for packets sent between two parties.
- ✓ Its definition requires a cryptographic hash function H and a secret key K. H denotes a hash function where the message is hashed by iterating a basic compression function on data blocks.
- ✓ Let b denote the block length of 64 bytes or 512 bits for all hash functions such as MD5 and SHA-1. h denotes the length of hash values, i.e. h = 16 bytes or 128 bits for MD5 and 20 bytes or 160 bits for SHA-1.
- ✓ The secret key K can be of any length up to b = 512 bits. To compute HMAC over the message, the HMAC equation is expressed as follows:

HMAC = $H[(K \oplus \text{opad}) || H[(K \oplus \text{ipad}) || M]]$

where

ipad = 00110110*(*0x36*)* repeated 64 times (512 bits)

opad = 01011100*(*0x5c*)* repeated 64 times (512 bits)

ipad is inner padding opad is outer padding

The following explains the HMAC equation:

1.Append zeros to the end of *K* to create a *b*-byte string (i.e. if *K* = 160 bits in length and *b* = 512 bits, then *K* will be appended with 352 zero bits or 44 zero bytes 0x00).

2. XOR (bitwise exclusive-OR) *K* with ipad to produce the *b*-bit block computed in step 1.

3. Append *M* to the *b*-byte string resulting from step 2.

4. Apply *H* to the stream generated in step 3.

5. XOR (bitwise exclusive-OR) *K* with opad to produce the *b*-byte string computed in step 1.

6. Append the hash result *H* from step 4 to the *b*-byte string resulting from step5.

7. Apply *H* to the stream generated in step 6 and output the result.

2.Explain in detail about the IP Authentication Header.

- ✓ The IP AH is used to provide data integrity and authentication for IP packets.
- ✓ It also provides protection against replays. The AH provides authentication for the IP header, as well as for upper-level protocol (TCP, UDP) data.
- ✓ But some IP header fields may change in transit and the sender may not be able to predict the value of these fields when the packet arrives at the receiver.
- ✓ The AH can be used in conjunction with ESP or with the use of tunnel mode. Security services can be provided between a pair of hosts, between a pair of security gateway or between a security gateway and a host.
- ✓ The ESP provides a confidentiality service.

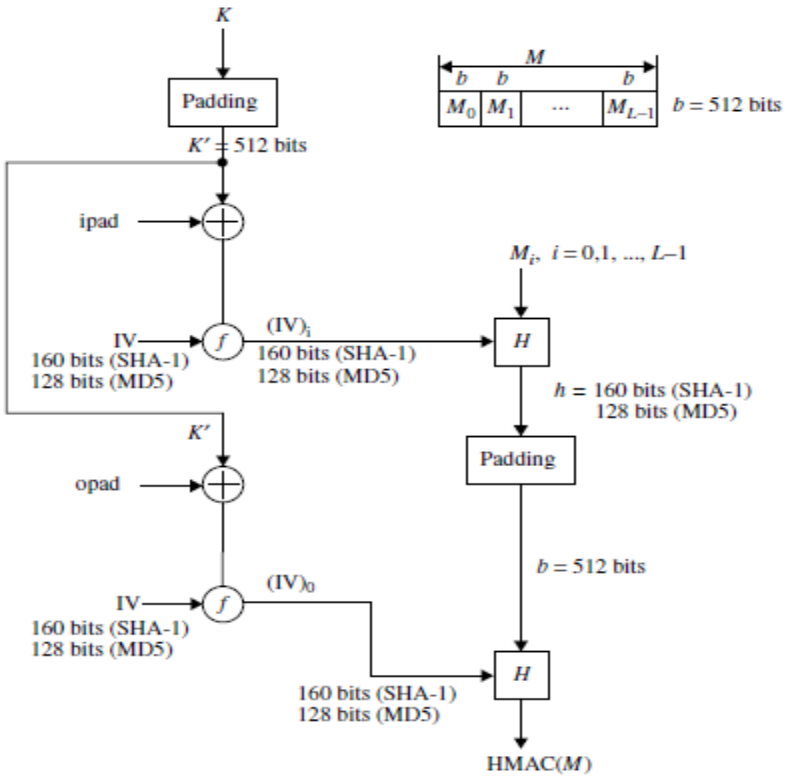


Figure 7.3 Alternative operation of HMAC computation using either MD5 or SHA-1 (message length computation based on M only).

2.1 AH Format

The IPsec AH format is shown in Figure 7.4. The following six fields comprise the AH format:

Next header (8 bits): This field identifies the type of the next payload after the AH. The value of this field is chosen from the set of IP numbers defined in the Internet Assigned Number Authority (IANA).

Next header (8 bits)	Payload length (8 bits)	Reserved (16 bits)
Security Parameters Index (SPI) (32 bits)		
Sequence number (32 bits)		
Authentication data (variable)		

Figure 7.4 IPsec AH format.

Payload length (8 bits):

- ✓ This field specifies the length of the AH in 32-bit words, minus 2.
- ✓ The default length of the authentication data field is 96 bits, or three 32-bit words.
- ✓ With a three-word fixed header, there are a total of six words in the header, and the payload length field has a value of 4.

***Reserved (16 bits)*:**

- ✓ This field is reserved for future use. It must be set to 'zero'.

***SPI (32 bits)*:**

- ✓ This field uniquely identifies the SA for this datagram, in combination with the destination IP address and security protocol (AH).
- ✓ The set of SPI values in the range 1–255 is reserved by the IANA for future use. The SPI value of zero (0) is reserved for local, implementation-specific use.

***Sequence number (32 bits)*:**

- ✓ This field contains the monotonically increasing counter value which provides an anti-replay function.
- ✓ Even if the sender always transmits this field, the receiver need not act on it, i.e. processing of the sequence number field is at the discretion of the receiver.
- ✓ The sender's counter and the receiver's counter are initialized to zero when an SA is established.
- ✓ The first packet sent using a given SA will have a sequence number of 1. The sender increments the sequence number for this SA and inserts the new value into the sequence number field.
- ✓ If anti-replay is enabled, the sender checks to ensure that the counter has not cycled before inserting the new value in the sequence number field.
- ✓ If the counter has cycled, the sender will set up a new SA and key.
- ✓ If the anti-replay is disabled, the sender does not need to monitor or reset the counter. However, the sender still increments the counter and when it reaches the maximum value, the counter rolls over to zero.

***Authentication data (variable)*:**

- ✓ This field is a variable-length field that contains the Integrity Check Value (ICV) or MAC for this packet. This field must be an integral Multiple of 32-bit words. It may include explicit padding. This padding is included to Ensure that the length of AH is an integral multiple of 32 bits (IPv4) or 64 bits (IPv6).

2.2 AH Location

- ✓ Either AH or ESP is employed in two ways: transport mode or tunnel mode.

- ✓ The transport mode is applicable only to host implementations and provides protection for upper-layer protocols.
- ✓ In the transport mode, AH is inserted after the IP header and before an upper layer protocol (TCP, UDP or ICMP), or before any other IPsec header that may have already been inserted.
- ✓ In the IPv4 context, AH is placed after the original IP header and before the upper-layer protocol TCP or UDP.
- ✓ Authentication covers the entire packet, excluding mutable fields in the IPv4 header that are set to zero for MAC computation.
- ✓ The positioning of AH transport mode for an IPv4 packet is illustrated in Figure 7.5(a).
- ✓ In the IPv6 context, AH should appear after hop-to-hop, routing and fragmentation extension headers.
- ✓ The destination options extension header(s) could appear either before or after AH, depending on the semantics desired.
- ✓ Authentication again covers the entire packet, excluding mutable fields that are set to zero for MAC computation.
- ✓ The positioning of AH transport mode for an IPv6 packet is illustrated in Figure 7.5(b).
- ✓ Tunnel mode AH can be employed in either hosts or security gateways.

3. Explain in detail about the IP ESP.

- ✓ The ESP header is designed to provide security services in IPv4 and IPv6. ESP can be applied alone, in combination with the IP AH or through the use of tunnel mode.
- ✓ The ESP header is inserted after the IP header and before the upper-layer protocol header (transport mode) or before an encapsulated IP header (tunnel mode).
- ✓ ESP is used to provide confidentiality (encryption), data authentication, integrity and anti-replay service, and limited traffic flow confidentiality. Confidentiality could be selected independent of all other services.

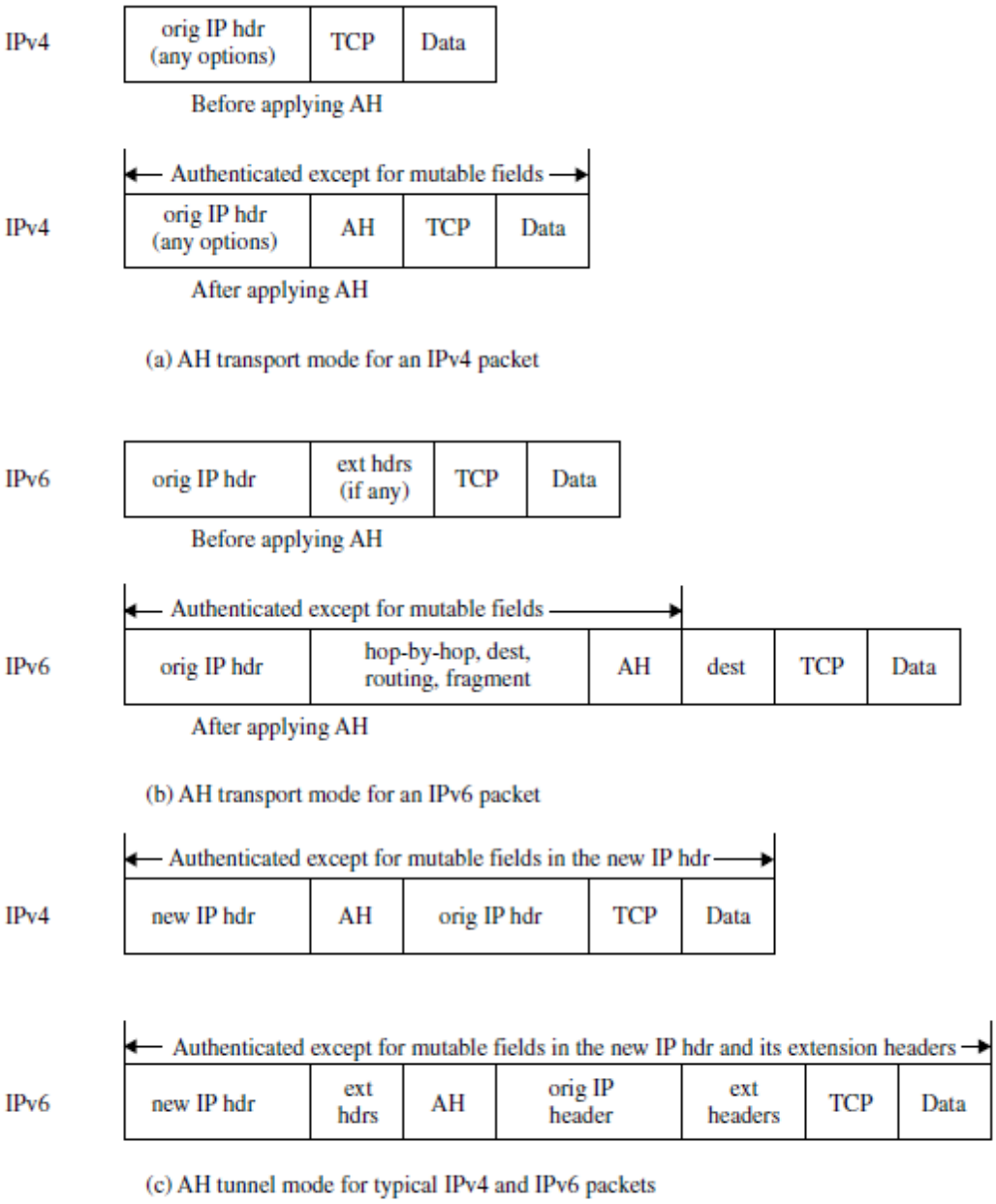


Figure 7.5 Transport mode and tunnel mode for AH authentication.

ESP Packet Format

- ✓ Figure 7.6 shows the format of an ESP packet and the fields in the header format are defined in the following.

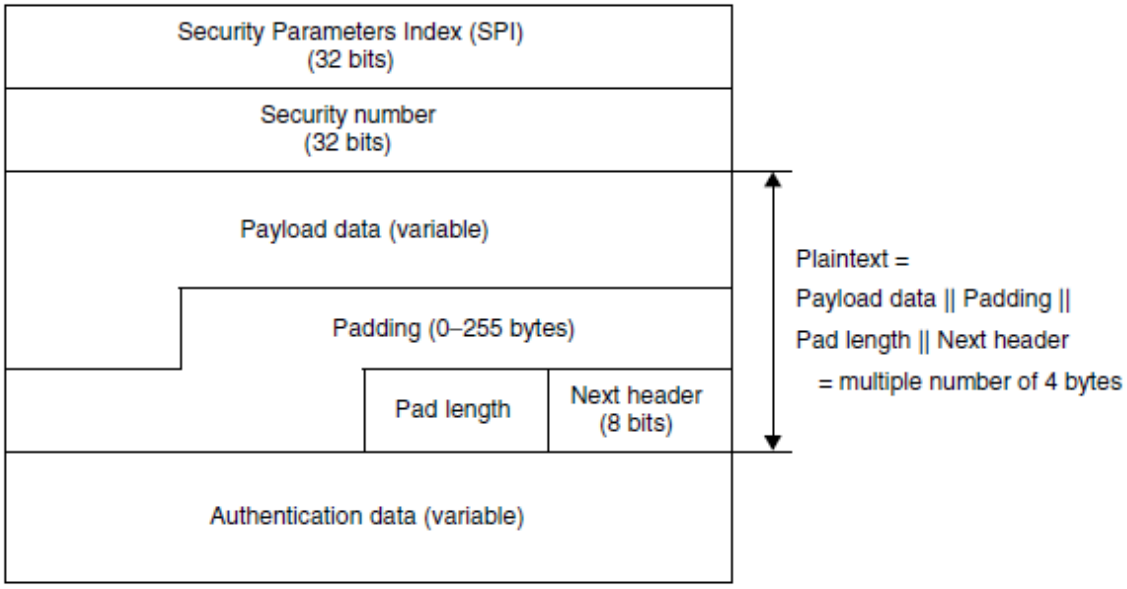


Figure 7.6 IPsec ESP format.

***SPI (32 bits)*:**

- ✓ The SPI is an arbitrary 32-bit value that uniquely identifies an SA for this datagram.
- ✓ The set of SPI values in the range 1–255 is reserved by the IANA for future use.

  The SPI field in the ESP packet format is mandatory and always present.

***Sequence number (32 bits)*:**

- ✓ This field contains a monotonically increasing counter value. This provides an anti-replay function.
- ✓ It is mandatory and is always present even if the receiver does not elect to enable the anti-replay service for a specific SA.
- ✓ If anti-replay is enabled, the transmitted sequence number must not be allowed to cycle.
- ✓ Thus, the sender's counter and the receiver's counter must be reset prior to the transmission of the 232nd packet on an SA.

***Payload data (variable)*:**

- ✓ This variable-length field contains data described by the next header field. The field is an integral number of bytes in length. If the algorithm requires an initialization vector (IV) to encrypt payload, then this data may be carried explicitly in the payload field.

- ✓ Any encryption algorithm that requires such IP data must indicate the length, structure and location of this data by specifying how the algorithm is used with ESP. For some IP-based modes of operation, the receiver treats the IP as the start of the ciphertext, feeding it into the algorithm directly.

***Padding*:** This field for encryption requires several factors:

- ✓ If an encryption algorithm requires the plaintext to be a multiple number of bytes, the padding field is used to fill the plaintext to the size required by the algorithm. The plaintext consists of the payload data, pad length and next header field, as well as the padding (see Figure 7.6)
- ✓ Padding is also required to ensure that the ciphertext terminates on a 32-bit boundary.
- ✓ Specifically, the pad length and next header fields must be right aligned within a 32-bit word to ensure that the authentication data field is aligned on a 32-bit boundary.

***Pad length*:** This field indicates the number of pad bytes immediately preceding it.The range of valid values is 0–255, where a value of 0 indicates that no padding bytes are present. This field is mandatory.

***Next header (8 bits)*:** This field identifies the type of data contained in the payload data field, i.e. an extension header in IPv6 or an upper-layer protocol identifier. The value of this field is chosen from the set of IP numbers defined by the IANA. The next header field is mandatory.

***Authentication data (variable)*:**

- ✓ This is a variable-length field containing an ICV computed over the ESP packet minus the authentication data.
- ✓ The length of this field is specified by the authentication function selected. The field is optional and is included only if the authentication service has been selected for the SA in question.
- ✓ The authentication algorithm must specify the length of the ICV and the comparison rules and processing steps for validation.

3.2 ESP Header Location

- ✓ ESP is also employed in the two transport or tunnel modes. The transport mode is applicable only to host implementations and provides protection for upper protocols, but not the IP header.

- ✓ In the transport mode, ESP is inserted after the IP header and before an upper-layer protocol (TCP, UDP or ICMP), or before any other IPsec headers that have already been inserted.
- ✓ In the IPv4 context, ESP is placed after the IP header, but before the upper-layer protocol. The ESP trailer encompasses any padding, plus the pad length, and next header fields.

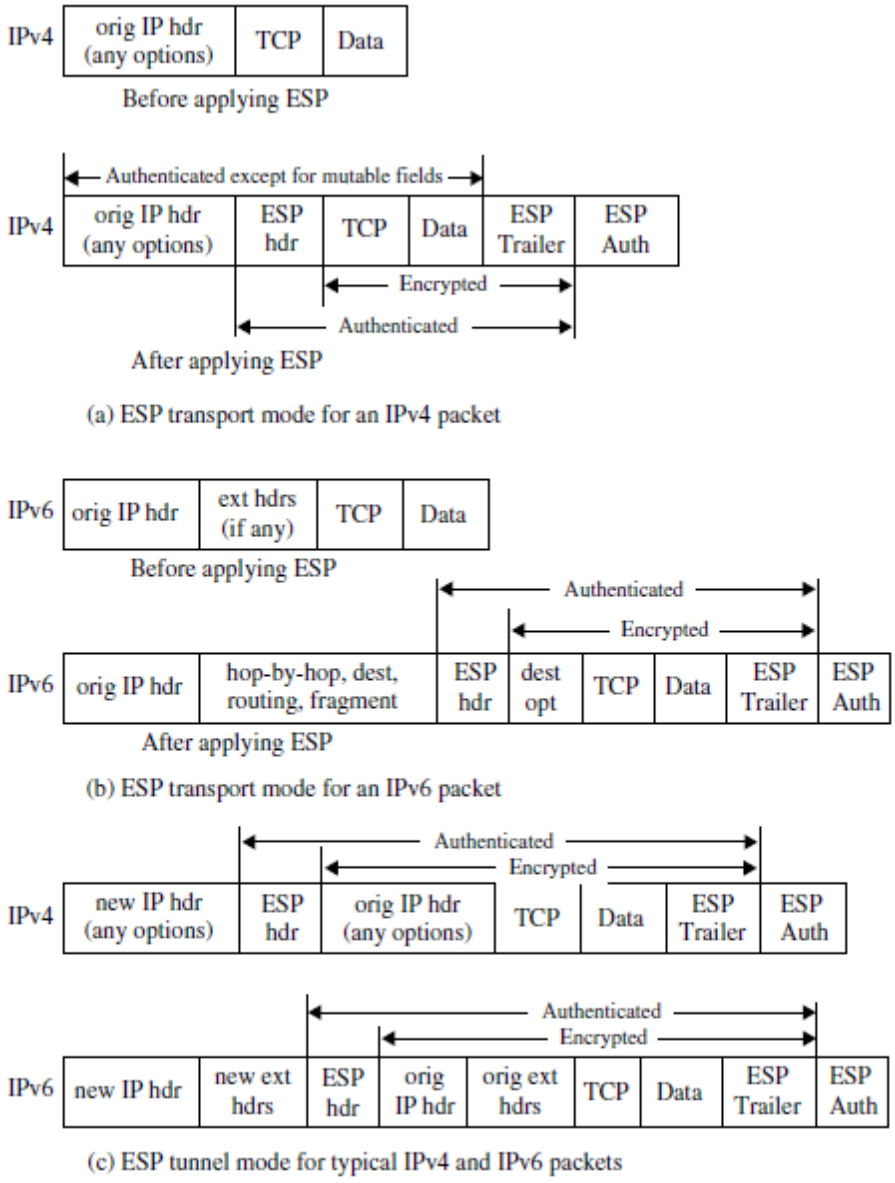


Figure 7.7 Transport mode and tunnel mode for ESP authentication.

In the IPv6 context, the ESP appears after hop-by-hop, routing and fragmentation extension headers. The destination options extension header(s) could appear either before or after the ESP header depending on the semantics desired. However, since ESP protects only fields after the ESP header, it is generally desirable to place the

destination options header(s) after the ESP header. Figure 7.7(b) illustrates ESP transport mode positioning for a typical IPv6 packet.

3.3 Encryption and Authentication Algorithms

- ✓ ESP is applied to an outbound packet associated with an SA that calls for ESP processing.
- ✓ The encryption algorithm employed is specified by the SA, as is the authentication algorithm.

3.3.1 Encryption

- ✓ ESP is designed for use with symmetric algorithms like a triple DES in CBC mode. However, a number of other algorithms have been assigned identifiers in the DOI document.

✓ These algorithms for encryption are: RC5, IDEA, CAST and Blowfish.

✓ For encryption to be applied, the sender encapsulates the ESP payload field, adds any necessary padding, and encrypts the result (i.e. payload data, padding, pad length and next header).

✓ The sender encrypts the fields (payload data, padding, pad length and next header) using the key, encryption algorithm, algorithm mode indicated by the SA and an IV (cryptographic synchronization data).

✓ If the algorithm to be encrypted requires an IV, then this data is carried explicitly in the payload field.

✓ The payload data field is an integral number of bytes in length. Since ESP provides padding for the plaintext, encryption algorithms employed by ESP exhibit either block or stream mode characteristics.

✓ The encryption is performed before the authentication and does not encompass the authentication data field.

- ✓ The order of this processing facilitates rapid detection and rejection of replayed or bogus packets by the receiver, prior to decrypting the packet. Therefore, it will reduce the impact of service attacks.
- ✓ At the receiver, parallel processing of packets is possible because decryption can take place in parallel with authentication.
- ✓ For successive blocks, the previous ciphertext block is XORed with the current plaintext before it is encrypted. Triple DES, known as DES–EDE3, processes each block three times, each time with a different key.
- ✓ Therefore, the triple DES algorithm has 48 rounds. In DES–EDE3-CBC, an IV is XORed with the first 64-bit plaintext block (P1). Some cipher

algorithms allow for a variable-sized key (RC5), while others only allow a specific key size (DES, IDEA).

3.3.2 Decryption

- ✓ The receiver decrypts the ESP payload data, padding, pad length and next header using the key, encryption algorithm, algorithm mode and IV data.
- ✓ If explicit IV data is indicated, it is taken from the payload field and input to the decryption algorithm.
- ✓ If implicit IV data is indicated, a local version of the IV is constructed and input to the decryption algorithm.

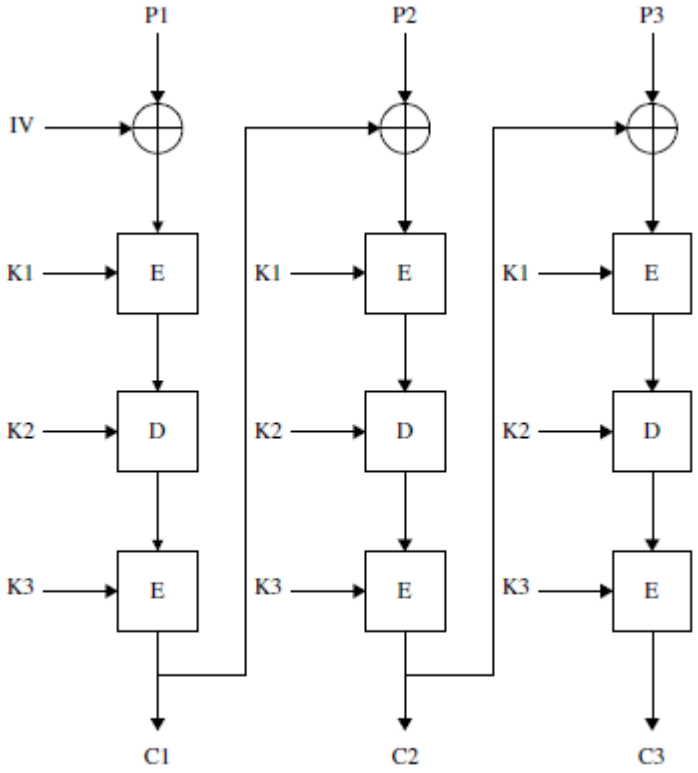


Figure 7.8 DES–EDE3–CBC algorithm.

The exact steps for reconstructing the original datagram depend on the mode (transport or tunnel) and are described in the Security Architecture document. The receiver processes any padding as given in the encryption algorithm specification. For transport mode, the receiver reconstructs the original IP datagram from the original IP header plus the original upper-layer protocol information in the ESP payload field. For tunnel mode, the receiver reconstructs the tunnel IP header plus the entire IP datagram in the ESP payload field.

3.3.3 Authentication

- ✓ The authentication algorithm employed for the ICV computation is specified by the SA.
- ✓ For communication between two points, suitable authentication algorithms include Keyed Message Authentication Codes (MACs) based on symmetric encryption algorithms (i.e. DES) or on one-way hash function (i.e. MD5 or SHA-1).

✓ For multicast communication, one-way hash algorithms combined with asymmetric signature algorithms are appropriate.

✓ If authentication is selected for the SA, the sender computes the ICV over the ESP packet minus the authentication data.

3.3.4 ICV

- ✓ Once the SA selects the authentication algorithm, the sender computes the ICV over the ESP packet minus the authentication data.
- ✓ The ICV is an MAC or a truncated value of a code produced by an MAC algorithm.
- ✓ As with AH, ESP supports the use of an MAC with a default length of 96 bits. The current specification for use of the HMAC computation must support:

HMAC–MD5–96

HMAC–SHA-1–96

4.Explain in detail about the Key Management Protocol for IPSec.

- ✓ The key management mechanism of IPsec involves the determination and distribution of a secret key. Key establishment is at the heart of data protection that relies on cryptography.
- ✓ A secure key distribution for the Internet is an essential part of packet protection.
- ✓ IKE combines ISAKMP with the Oakley key exchange. We begin our discussion with an overview of Oakley and then look at ISAKMP.

4.1 OAKLEY Key Determination Protocol

- ✓ The Diffie–Hellman key exchange algorithm provides a mechanism that allows two users to agree on a shared secret key without requiring encryption.
- ✓ This shared key is immediately available for use in encrypting subsequent data transmission.

- ✓ Oakley is not only a refinement of the Diffie–Hellman key exchange algorithm, but a method to establish an authentication key exchange.
- ✓ The Oakley protocol is truly used to establish a shared key with an assigned identifier and associated authenticated identities for the two parties.
- ✓ Oakley can be used directly over the IP protocol or over UDP protocol using a well-known port number assignment available.

4.2 ISAKMP

- ✓ ISAKMP defines a framework for SA management and cryptographic key establishment for the Internet.
- ✓ This framework consists of defined exchange, payloads and processing guidelines that occur within a given DOI.
- ✓ ISAKMP defines procedures and packet formats to establish, negotiate, modify and delete SAs.
- ✓ It also defines payloads for exchanging key generation and authentication data.
- ✓ These payload formats provide a consistent framework for transferring key and authentication data which is independent of the key generation technique, encryption algorithm and authentication mechanism.

ISAKMP is intended to support the negotiation of SAs for security protocols at all layers of the network stack. By centralizing the management of the SAs, ISAKMP reduces the amount of duplicated functionality within each security protocol.

(I) ISAKMP Payloads

ISAKMP payloads provide modular building blocks for constructing ISAKMP messages.The presence and ordering of payloads in ISAKMP are defined by and dependent upon the Exchange Type Field located in the ISAKMP Header.

ISAKMP Header

The ISAKMP header fields are fined as shown in Figure 7.9.

Initiator Cookie (64 bits) - This field is the cookie of entity that initiated SA establishment, SA notification, or SA deletion.

Responder Cookie (64 bits) - This field is the cookie of entity that is corresponded to an SA establishment request, SA notification, or SA deletion.

Next Payload (8 bits) - This field indicates the type of the first payload in the message.

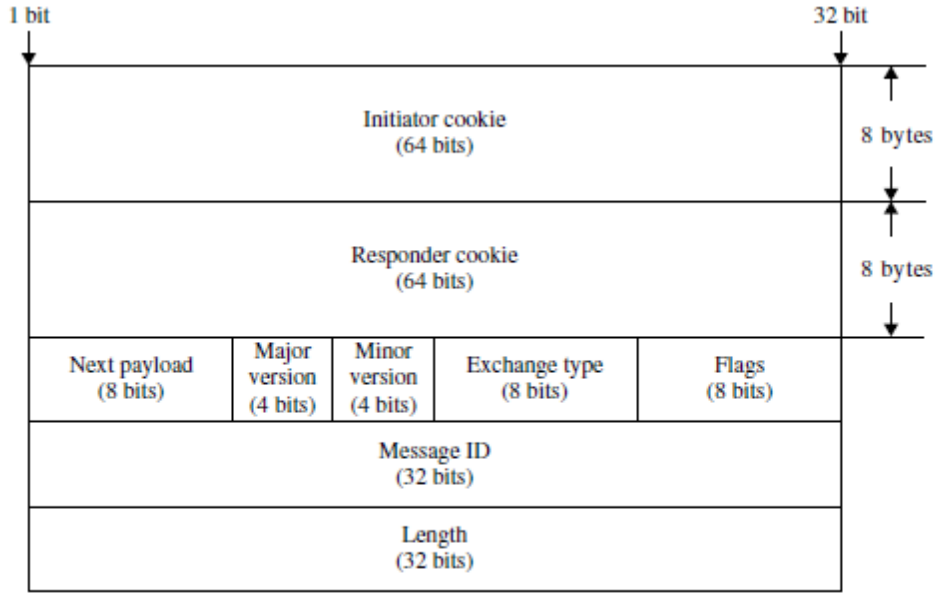


Figure 7.9 ISAKMP header format.

Major Version (4 bits) - This field indicates the Major version of the ISAKMP protocol in use. Set the Major version to 1 according to ISAKMP Internet-Draft.

Minor Version (4 bits) -- This field indicates the Minor version of ISAKMP protocol in use. Set the Minor version to 0 according to implementations based on the ISAKMP Internet-Draft.

Exchange Type (8 bits) - This field indicates the type of exchange being used. This dictates the message and payload orderings in the ISAKMP exchanges.

Flags (8 bits) - This field indicates specific options that are set for the ISAKMP exchange. The Flags are specified in the Flags field beginning with the least significant bit: the encryption bit is bit 0 of the Flags field, the commit bit is bit 1, and authentication only bit is bit 2 of the Flags field. The remaining bits of the Flags field must be set to 0 prior to transmission.

Message ID (32 bits) - Message ID is used to identify protocol state during Phase 2 negotiations. This value is randomly generated by the initiator of the phase 2 negotiation. During Phase 1 negotiation, this value must be set to 0.

Length (32 bits) - Length of total message (header || payload) is 32 bits. Encryption can expand the size of an ISAKMP message.

Generic Payload Header

- ✓ Each ISAKMP payload begins with a generic header which provides a payload chaining capability and clearly defines the boundaries of a payload.
- ✓ The generic payload header fields in 32 bits are defined as follows:

Next Payload (8 bits) - This field is identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0. This field provides the chaining capability.

Reserved (8 bits) - This field is not used and set to 0.

Payload Length (16 bits) - This field indicates the length in bytes of the current payload, including the generic payload header.

(II) Payload Types for ISAKMP

ISAKMP defines several types of payloads that are used to transfer information such as SA data or key exchange data in DOI-defined formats.

Security Association Payload

- ✓ The Security Association Payload is used to negotiate security attributes and to identify the Domain of Interpretation (DOI, 32 bits) under which negotiation is taking place.
- ✓ A DOI value of 0 during a Phase 1 exchange specifies a Generic ISAKMP which can be used for any protocol during the Phase 2 exchange. A DOI value of 1 is assigned to the IPsec DOI.

The Security Association Payloads are defined as follows:

- ✓ The Next Payload field (8 bits) is the identifier for the payload type of the next payload in the message. This field has a value of 0 if this is the last payload in the message.
- ✓ The Reserved field (8 bits) is unused, set to 0.
- ✓ The Payload Length field (16 bits) indicates the length in octets of the entire Security Association payload, including the SA payload, all Proposal payloads, and all Transform payloads associated with the proposed SA

Proposal Payload

The Proposal Payload is used to build ISAKMP message for the negotiation and establishment of SAs. The Proposal Payload field contains information used during SA negotiation for securing the communications channel. The payload type for the Proposal Payload is two(2).

The Proposal Payload fields are defined as follows:

- ✓ The Next Payload field (8 bits) is the identifier for the payload type of the next payload in the message.
- ✓ This field must only contain the value 2 or 0. This field will be 2 for additional Proposal Payloads in the message and 0 when the current Proposal Payload is the last within the SA proposal.

- ✓ The Reserved field (8 bits) is set to 0 and is reserved it for the future use.
- ✓ The Payload Length field (16 bits) is the length in octets of the entire Proposal payload, including generic payload header, the Proposal Payload, and all Transform payloads associated with this proposal.

- ✓ The Proposal # field (8 bits) identifies the proposal number for the current payload.
- ✓ The Protocol-id field (8 bits) specifies the protocol identifier for the current negotiation.

Examples might include IPsec ESP, IPsec AH, OSPF, TLS, etc.

Transform Payload

- ✓ The Transform Payload contains information used during Security Association negotiation.
- ✓ The Transform Payload consists of a specific security mechanism to be used to secure the communications channel.
- ✓ The Transform Payload also contains the security association attributes associated with the specific transform.
- ✓ These SA attributes are DOI-specific. The Transform Payload allows the initiating entity to present several possible supported transforms for that proposed protocol.

The Transform Payload field s are defined as follows:

- ✓ The Next Payload field (8 bits) is the identifier for the payload type of the next payload in the message.
- ✓ This field must only contain the value 3 or 0. This field is 3 when there are additional Transform payloads in the proposal. This field is 0 when the current Transform Payload is the last within the proposal.
- ✓ The Reserved field (8 bits) is for unused, set to 0.
- ✓ The Transform # field (8 bits) identifies the Transform number for the current payload.
- ✓ If there is more than one transform within the Proposal Payload, then each Transform Payload has a unique Transform number.
- ✓ The Transform-id field (8 bits) specifies the Transform identifier for the protocol within the current proposal.
- ✓ The Reserved 2 field (16 bits) is for unused, set to 0.

Key Exchange Payload

The Key Exchange Payload supports a variety of key exchange techniques. Example

key exchanges are Oakley, Diffie-Hellman, the enhanced D-H key exchange, and the RSA-based key exchange used by PGP.

The Key Exchange Payload fields are defined as follows:

- ✓ The Next Payload field (8 bits) is the identifier for the payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0.
- ✓ The Reserved field (8 bits) is unused for the future use, set to 0.
- ✓ The Payload Length field (16 bits) is the length in octets of the current payload, including the generic payload header.
- ✓ The Key Exchange Data field (variable length) is the data required to generate a session key. The interpretation of this data is specified by the DOI and the associated Key Exchange algorithm. This field may also contain pre-placed key indicators.

Identification Payload

The Identification Payload contains DOI-specific data used to exchange identification information. This information is used for determining the identities of communication partners and may be used for determining authenticity of information.

The Identification Payload fields are described as follows:

- ✓ The Next Payload field (8 bits) is the identifier for the payload type of the Next Payload in the message. If the current payload is the last in the message, then this field will be 0.
- ✓ The Reserved field (8 bits) is not used, but set to 0.
- ✓ The Payload Length field (16 bits) is the length in octets of the current payload, including the generic payload header.
- ✓ The ID type field (8 bits) specifies the type of identification being used. This field is DOI-dependent.
- ✓ The DOI specific ID Data field (24 bits) contains DOI specific identification data. If unused, then this field must be set to 0.

Certificate Payload

The Certificate Payload provides a mean to transport certificates via ISAKMP and can appear in any ISAKMP message. Certificate payloads should be included in an exchange whenever an appropriate directory service is not available to distribute certificates. The Certificate payload must be accepted at any point during an exchange.

The Certificate Payload fields are defined as follows:

- ✓ The Next Payload field (8 bits) is the identifier for the Payload type of the next payload in the message. If the current payload is the last in the message, then this field will be 0.
- ✓ The Reserved field (8 bits) is unused, set to 0.
- ✓ The Payload Length field (16 bits) is the length in octets of the current payload, including the generic payload header.
- ✓ The Certificate Encoding field (8 bits) indicates the type of certificate or certificate-related information contained in the Certificate Data field.

Certificate Type	**Value**
NONE	0
PKCS #7 wrapped X.509 certificate	1
PGP Certificate	2
DNS Signed Key	3
X.509 Certificate-Signature	4
X.509 Certificate-Key Exchange	5
Kerberos Tokens	6
Certificate Revocation List (CRL)	7
Authority Revocation List (ARL)	8
SPKI Certificate	9
X.509 Certificate-Attribute	10
Reserved	11–255

Certificate Request Payload

- ✓ The Certificate Request Payload provides a mean to request certificate via ISAKMP and can appear in any message. Certificate Request Payloads should be included in an exchange whenever an appropriate directory service is not available to distribute certificates.
- ✓ The Certificate Request Payload must be accepted at any point during the exchange.
- ✓ The responder to the Certificate Request payload must send its certificate, if certificates are based on the values contained in the payload. If multiple certificates are required, then multiple Certificate Request Payloads should be transmitted.

Hash Payload

- ✓ The Hash Payload contains data generated by the hash function over some part of the message and/or ISAKMP state. This payload possibly be used to verify

the integrity of the data in an ISAKMP message or for authentication of the negotiating entities.

Signature Payload

- ✓ The Signature Payload contains data generated by the digital signature function, over some part of the message and/or ISAKMP state. This payload is used to verify the integrity of the data in the ISAKMP message, and may be of use for non-repudiation services.

Nonce Payload

- ✓ The Nonce Payload contains random data used to guarantee liveness during an exchange and protect against replay attacks. If nonce are used by a particular key exchange, the use of the Nonce Payload wil be dictated by the key exchange. The nonces may be transmitted as part of the key exchange data, or as a separate payload. However, this is defined by the key exchange, not by ISAKMP.

Notification Payload

- ✓ The Notification Payload can contain both ISAKMP and DOI-specific data and is used to transmit information data, such as error conditions to an ISAKMP peer. It is possible to send multiple Notification Payloads in a single ISAKMP message. Notification which occurs during a Phase 1 negotiation is identified by the Initiator and Responder cookie pair in the ISAKMP Header.
- ✓ Notification which occurs during a Phase 2 negotiation is identified by the Initiator and Responder cookie pair in the ISAKMP header and the Message ID and SPI associated with the current negotiation.

Delete Payload

- ✓ The Delete Payload contains a protocol-specific security association identifier that the sender has removed from its SA database. Therefore, the sender is no longer valid. It is possible to send multiple SPIs in a Delete Payload. But each SPI must be for the same protocol.

Vendor ID Payload

- ✓ The Vendor ID Payload contains a vendor defined constant. The constant is used by vendors to identify and recognize remote instances of their implementations. This mechanism allows a vendor to experiment with new features while maintaining backwards compatibility.

(III) ISAKMP Exchanges

- ✓ ISAKMP supplies the basic syntax of a message exchange. ISAKMP allows the creation of exchanges for SA establishment and key exchange. There are currently five default Exchange Types defined for ISAKMP. Exchanges define the content and ordering of ISAKMP messages during communications between peers. Most exchanges includes all the basic payload types: SA (Security Association Payload), KE (Key Exchange Payload), ID (Identity Payload), SIG (Signature Payload), etc. The primary difference between exchange types is the ordering of messages and the payload ordering within each message.

Base Exchange

- ✓ The Base Exchange is designed to allow the Key Exchange and Authentication-related information to be transmitted together. Combining the Key Exchange and Authentication related information into one message reduces the number of round-trips at the expense of not providing identity protection.

Identity Protection Exchange

- ✓ The Identity Protection Exchange is designed to separate the Key Exchange information from the Identity and Authentication-related information.

Authentication Only Exchange

- ✓ The Authentication Only Exchange is designed to allow only Authentication-related information to be transmitted.
- ✓ The benefit of this exchange is the ability to perform only authentication without the computational expense of computing keys.
- ✓ Using this exchange during negotiation, none of the transmitted information will be encrypted. But the authentication only exchange will be encrypted by the ISAKMP SA, negotiated in the first phase.

Aggressive Exchange

- ✓ The Aggressive Exchange is designed to allow the Security Association, Key Exchange and Authentication-related payloads to be transmitted together. Combining these SA, KE, and Auth information into one message reduces the number of round-trips at the expense of not providing identity protection. Identity protection is not provided because identities are exchanged before a common shared secret has been established.

Informational Exchange

- ✓ The Information Exchange is designed as a one-way transmittal of information that can be used for security association management.

- ✓ If the Informational Exchange occurs prior to the exchange of keying material during an ISAKMP Phase 1 negotiation, there will be no protection provided for the Information Exchange.

(IV) ISAKMP Payload Processing

- ✓ The ISAKMP payloads are used in the exchanges described in Part III above and can be used in exchanges defined for a specific DOI.

General Message Processing

- ✓ Every ISAKMP message has basic processing applied to insure protocol reliability and to minimize threats such as denial of services and replay attacks. All processing should include packet length checks to insure the packet received is at least as long as the length given in the ISAKMP Header.

ISAKMP Header Processing

- ✓ When an ISAKMP message is created at the transmitting entity, the initiator (transmitter) must create the respective cookie, determine the relevant security characteristics of the session, construct an ISAKMP Header with fields, and transmit the message to the destination host (responder).
- ✓ When an ISAKMP is received at the receiving entity, the responder (receiver) must verify the Initiator and Responder cookies, check the Next Payload field to confirm it is valid, check the Major and Minor Version fields to confirm they are correct, check the Exchange Type field to confirm it is valid, check the Flags field to ensure it contains correct values, and check the Message ID field to ensure it contains correct values.

Generic Payload Header Processing

- ✓ When any of the ISAKMP Payloads are created, a Generic Payload Header is placed at the beginning of these payloads.
- ✓ When creating the Generic Payload Header, the transmitting entity (initiator) must place the value of the Next Payload in the Next Payload field, place the value zero(0) in the Reserved field, place the length (in octets) of the payload in the Payload Length field, and construct the payloads.

Security Association Payload Processing

- ✓ When a Security Association Payload is created, the transmitting entity (initiator) must determine the Domain of Interpretation (DOI) for which this negotiation is being preformed, determine the situation within the determined DOI for which this negotiation is being formed, determine the proposal(s) and

transform(s) within the situation, construct a Security Association payload, and transmit the message to the receiving entity (responder).

Proposal Paylaod Processing

- ✓ When a Proposal Payload is created, the transmitting entity (initiator) must determine the Protocol for this proposal, determine the number of proposals to be offered for this proposal and the number of transform for each proposal, generate a unique pseudo-random SPI, and construct a Proposal payload.

Transform Payload Processing

- ✓ When a Transform payload is received, the receiving entity (responder) must do as follows: Determine if the Transform is supported. If the Transform-ID field contains an unknown or unsupported value, then that Transform payload must be ignored. Ensure Transforms are presented according to the details given in the Transform Payload and Security Association Establishment. Finally, process the subsequent Transform and Proposal payloads as defined by the Next Payload field.

Key Exchange Payload Processing

- ✓ When creating a Key Exchange payload, the transmitting entity (initiator) must determine the Key Exchange to be used as defined by the DOI, determine the usage of Key Exchange Data field as defined by the DOI, and construct a Key Exchange payload. Finally, transmit the message to the receiving entity (responder).

Identification Payload Processing

- ✓ When an Identification Payload is created, the transmitting entity (initiator) must determine the Identification information to be used as defined by the DOI, determine the usage of the Identification Data field as defined by the DOI, construct an Identification payload, and finally transmit the message to the receiving entity.

Certificate Payload Processing

- ✓ When a Certificate Payload is created, the transmitting entity (initiator) must determine the Certificate Encoding which is specified by the DOI, ensure the existence of a certificate formatted as defined by the Certificate Encoding, construct a Certificate payload, and then transmit the message to the receiving entity (responder).

- ✓ When a Certificate payload is received, the receiving entity (responder) must determine if the Certificate Encoding is supported. If the Certificate Encoding is not supported, the payload is discarded.

Certificate Request Payload Processing

- When creating a Certificate Request Payload, the transmitting entity (initiator) must determine the type of Certificate Encoding to be requested, determine the name of an acceptable Certificate Authority, construct a Certificate Request payload, and then transmit the message to the receiving entity (responder).
- When a Certificate Request payload is received, the receiving entity (responder) must determine if the Certificate Encoding is supported. If the Certificate Encoding is invalid, the payload is discarded.

Hash Payload Processing

- When creating a Hash Payload, the transmitting entity (initiator) must determine the Hash function to be used as defined by the SA negotiation, determine the usage of the Hash Data field as defined by the DOI, construct a Hash payload, and then transmit the message to the receiving entity (responder).
- When a Hash Payload is received, the receiving entity (responder) must determine if the Hash is supported. If the Hash determination fails, the message is discarded.

Signature Payload Processing

- When a Signature Payload is created, the transmitting entity(initiator) must determine the Signature function to be used as defined by the SA negotiation, determine the usage of the Signature Data filed as defined by the DOI, construct a Signature payload, and finally transmit the message to the receiving entity (responder).
- When a Signature payload is received, the receiving entity must determine if the

Signature is supported. If the Signature determination fails, the message is discarded.

Nonce Payload Processing

- When creating a Nonce Payload, the transmitting entity (initiator) must create unique random values to be used as a nonce, construct a Nonce payload, and transmit the message to the receiving entity.
- When a Nonce Payload is received, the receiving entity (responder) must do as follows:

- ✓ There are no specific procedures for handling Nonce payloads. The procedures are defined by the exchange types and possibly the DOI and Key Exchange descriptions.

Notification Payload Processing

- ✓ When a Notification Payload is created, the transmitting entity (initiator) must determine the DOI for this Notification, determine the Protocol-ID for this Notification, determine the SPI size based on the Protocol-ID field, determine the Notify Message Type based on the error or status message desired, determine the SPI which is associated with this notification, determine if additional Notification Data is to be included, construct a Notification Payload, and finally transmit the messages to the receiving entity.

Delete Payload Processing

- ✓ When a Delete Payload is created, the transmitting entity (initiator) must determine the DOI for this Deletion, determine the Protocol-ID for this Deletion, determine the SPI size based on the Protocol-id field, determine the # of SPIs to be deleted for this protocol, determine the SPI(s) which is (are) associated with this deletion, construct a Delete payload, and then transmit the message to the receiving entity.

5. Explain in detail about the SSL protocol.

- ✓ SSL is a layered protocol. It is not a single protocol but rather two layers of protocols. At the lower level, the SSL Record Protocol is layered on top of some reliable transport protocol such as TCP.
- ✓ The SSL Record Protocol is also used to encapsulate various higher level protocols. A higher-level protocol can layer on top of the SSL protocol transparently.Figure 8.1 illustrates the overview of the SSL protocol stack.

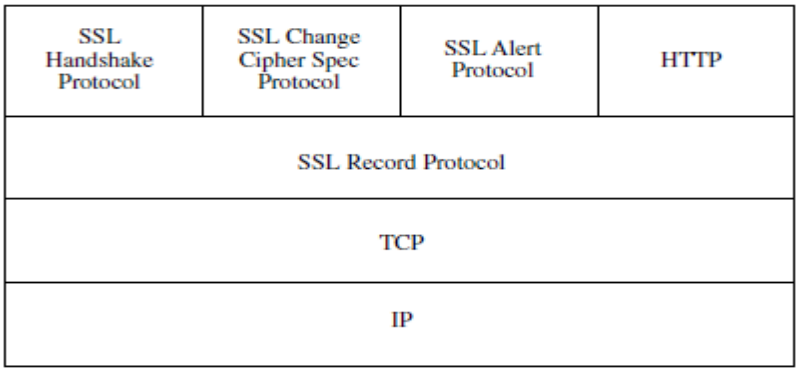


Figure 8.1 Two-layered SSL protocols.

Session and Connection States

There are two defined specifications: SSL session and SSL connection.

SSL session

- ✓ An SSL session is an association between a client and a server. Sessions are created by the Handshake Protocol.
- ✓ They define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.
- ✓ An SSL session coordinates the states of the client and server.

The session state is defined by the following elements:

Session identifier: This is a value generated by a server that identifies an active or reusable session state.

Peer certificate: This is an X.509 v3 certificate of the peer. This element of the state may be null.

Compression method: This is the algorithm used to compress data prior to encryption.

Cipher spec: This specifies the bulk data encryption algorithm (such as null, DES, etc.) and a hash algorithm (such as MD5 or SHA-1) used for MAC computation. It also defines cryptographic attributes such as the hash size.

***Master secret* :** This is a 48-byte secret shared between the client and server. It represents secure secret data used for generating encryption keys, MAC secrets and IVs.

***Is resumable*:** This designates a flag indicating whether the session can be used to initiate new connections.

SSL connection

- ✓ A connection is a transport (in the OSI layering model definition) that provides a suitable type of service. For SSL, such connections are peer-to-peer relationships. The connections are transient. Every connection is associated with one session.

The connection state is defined by the following elements:

***Server and client random*:** These are byte sequences that are chosen by the server and client for each connection.

***Server write MAC secret* :** This indicates the secret key used in MAC operations on data sent by the server.

***Client write MAC secret* :** This represents the secret key used in MAC operations on data sent by the client.

***Server write key*:** This is the conventional cipher key for data encrypted by the server and decrypted by the client.

***Client write key*:**

This is the conventional cipher key for data encrypted by the client and decrypted by the server.

Initialisation vectors: When a block cipher in CBC mode is used, an IV is maintained for each key. This field is first initialised by the SSL Handshake Protocol.

Sequence numbers: Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero. Sequence numbers may not exceed 264 − 1.

SSL Record Protocol

- ✓ The SSL Record Protocol provides basic security services to various higher-layer protocols.
- ✓ Three upper-layer protocols are defined as part of SSL: the Handshake Protocol, the Change Cipher Spec Protocol and the Alert Protocol.
- ✓ Two layers of SSL protocols are shown in Figure 8.1. The SSL Record Layer receives data from higher layers in blocks of arbitrary size.
- ✓ The SSL Record Protocol takes an application message to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies an MAC, encrypts, adds a header, and transmits the result in a TCP segment.

***Fragmentation*:** A higher-layer message is fragmented into blocks (SSLPlaintext records) of 214 bytes or less.

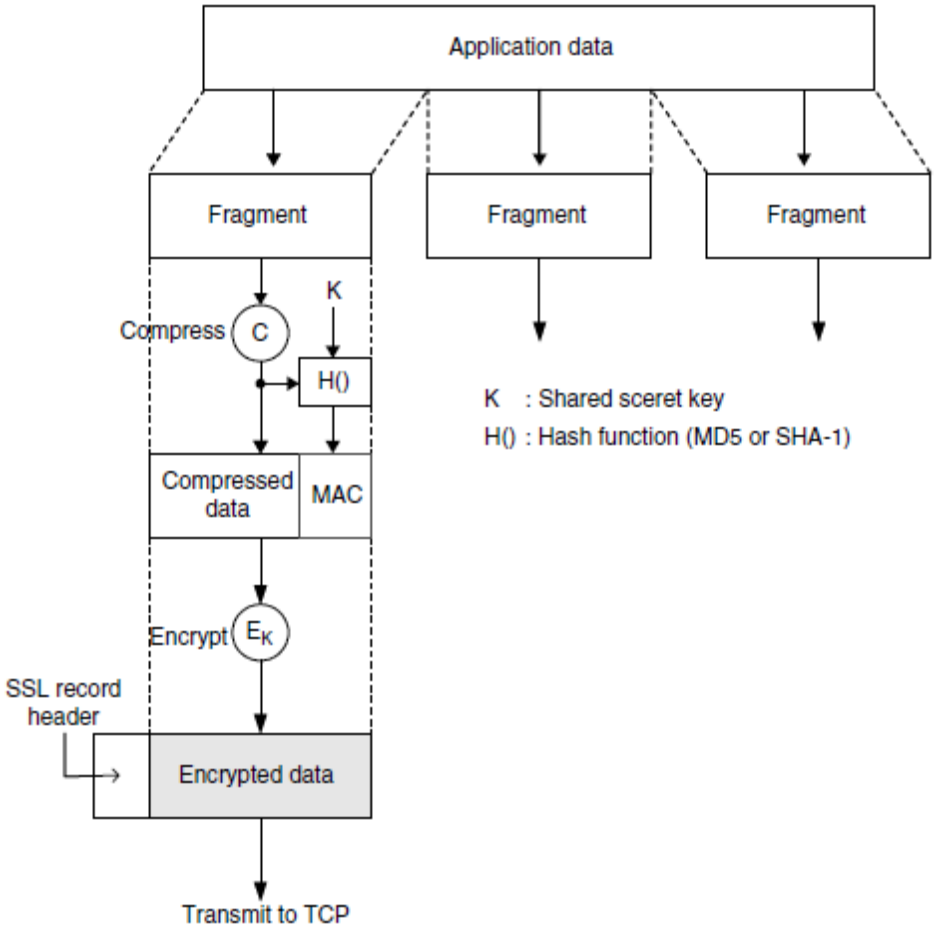


Figure 8.2 The overall operation of the SSL Record Protocol.

Compression and decompression:

- ✓ All records are compressed using the compression algorithm defined in the current session state. The compression algorithm translates an SSL Plaintext structure into an SSL Compressed structure.
- ✓ Compression must be lossless and may not increase the current length by more than 1024 bytes.
- ✓ If the decompression function encounters an SSL Compressed. fragment that would decompress to a length in excess of 214 = 16 348 bytes, it should issue a fatal decompression failure alert.

MAC: The MAC is computed before encryption. The computation of an MAC over the compressed data is illustrated in Figure 8.3. Using a shared secret key, the calculation

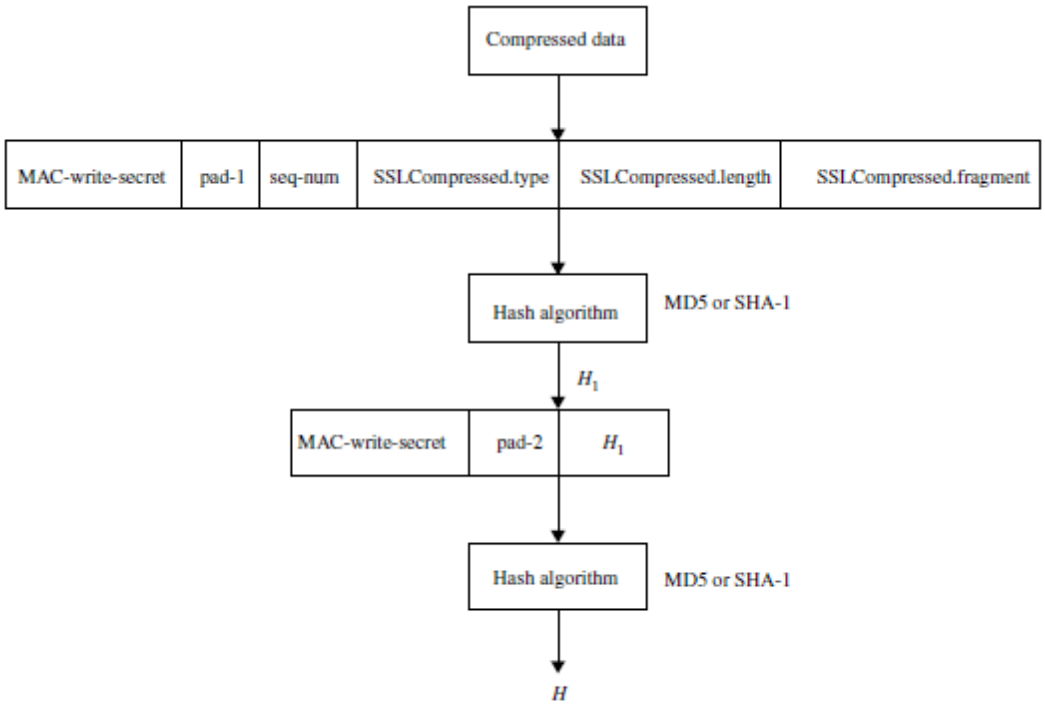


Figure 8.3 Computation of MAC over the compressed data.

is defined as follows:

$H1$ = hash(MAC-write-secret || pad-1 || seq-num || SSLCompressed.type || SSLCompressed.length || SSLCompressed.fragment)

H = hash(MAC-write-secret || pad-2 ||$H1$)

where

MAC-write-secret:	Shared secret key
Hash (H_1 and H):	Cryptographic hash algorithm; either MD5 or SHA-1
Pad-1:	The byte 0x36 (0011 0110) repeated 48 times (384 bits) for MD5 and 40 times (320 bits) for SHA-1
Pad-2:	The byte 0x5C (0101 1100) repeated 48 times for MD5 and 40 times for SHA-1
Seq-num:	The sequence number for this message
SSLCompressed.type:	The higher-level protocol used to process this fragment
SSLCompressed.length:	The length of the compressed fragment

SSLCompressed.fragment:	The compressed fragment (the plaintext fragment if not compressed)
‖:	concatenation symbol

The compressed message plus the MAC are encrypted using symmetric encryption. The block ciphers being used as encryption algorithms are:

DES(56), Triple DES(168), IDEA(128),

RC5(variable) and Fortezza(80)

where the number inside the brackets indicates the key size. Fortezza is a PCMCIA card that provides both encryption and digital signing.

Append SSL record header: The final processing of the SSL Record Protocol is to append an SSL record header. The composed fields consist of:

***Content type (8 bits)*:** This field is the higher-layer protocol used to process the enclosed fragment.

***Major version (8 bits)*:** This field indicates the major version of SSL in use. For SSLv3, the value is 3.

***Minor version (8 bits)*:** This field indicates the minor version of SSL in use. For SSLv3, the value is 0.

***Compressed length (16 bits)*:** This field indicates the length in bytes of the plaintext fragment or compressed fragment if compression is required. The maximum value is 214 + 2048.

SSL Change Cipher Spec Protocol

- ✓ The Change Cipher Spec Protocol is the simplest of the three SSL-specific protocols.
- ✓ This protocol consists of a single message, which is compressed and encrypted under the current CipherSpec. The message consists of a single byte of value 1.

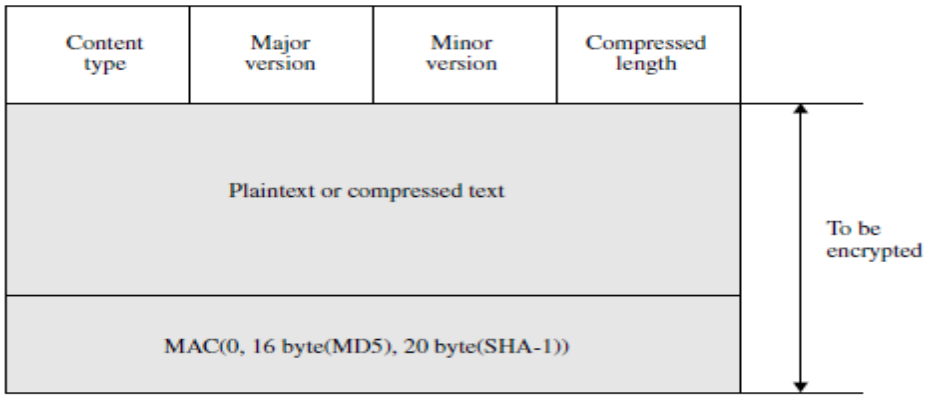


Figure 8.4 SSL Record Protocol format.

SSL Alert Protocol

- ✓ One of the content types supported by the SSL Record Layer is the alert type. Alert messages convey the severity of the message and a description of the alert. Alert messages consist of 2 bytes.
- ✓ The first byte takes the value warning or fatal to convey the seriousness of the message. If the level is fatal, SSL immediately terminates the connection. In this case, other connections on the same session may continue, but the session identifiers must be invalidated, preventing the failed session from being used to establish new connections.

The second byte contains a code that indicates the specific alert. As with other applications that use SSL, alert messages are compressed and encrypted, as specified by the current connection state.

A specification of SSL-related alerts that are always fatal is listed in the following:

Unexpected-message: An inappropriate message was received. This alert is always fatal.

Bad-record-mac: This alert is returned if a record is received with an incorrect MAC. This message is always fatal.

Decompression-failure: The decompression function received improper input (i.e. data that would expand to a length that is greater than the maximum allowable length). This message is always fatal.

No-certificate: This alert message may be sent in response to a certificate request if no appropriate certificate is available.

***Bad-certificate*:** A received certificate was corrupt, i.e. contained a signature that did not verify correctly.

Unsupported certificate: The type of the received certificate is not supported.

***Certificate-revoked*:** A certificate has been revoked by its signer.

***Certificate-expired*:** A certificate has expired or is not currently valid.

***Certificate-unknown*:** This means some other unspecified issue arose in processing the certificate, rendering it unacceptable.

***Illegal-parameter*:** A field in the handshake was out of range or inconsistent with other fields. This is always fatal.

***close-notify*:** This message notifies the recipient that the sender will not send any more messages on this connection. The session becomes unreasonable if any connection is terminated without proper close-notify messages with level equal to

warning. Each party is required to send a close-notify alert before closing the write side of the connection. Either party may initiate a close-notify alert. Any data received after a closure alert is ignored.

SSL Handshake Protocol

- ✓ The SSL Handshake Protocol being operated on top of the SSL Record Layer is the most important part of SSL. This protocol provides three services for SSL connections between the server and client.
- ✓ The Handshake Protocol allows the client/server to agree on a protocol version, to authenticate each other by forming an MAC, and to negotiate an encryption algorithm and cryptographic keys for protecting data sent in an SSL record before the application protocol transmits or receives its first byte of data.

The Handshake Protocol consists of a series of messages exchanged by the client and server. Figure 8.5 shows the exchange of handshake messages needed to establish a logical connection between client and server. The contents and significance of each message are presented in detail in the following sections.

Phase 1: Hello Messages for Logical Connection

- ✓ The client sends a client hello message to which the server must respond with a server hello message, or else a fatal error will occur and the connection will fail. The client hello and server hello are used to establish security enhancement capabilities between client and server. The client hello and server hello establish the following attributes: protocol version, random values (ClientHello.random and ServerHello.random), session ID, cipher suite and compression method.

Hello messages

- ✓ The hello phase messages are used to exchange security enhancement capabilities between client and server.

***Hello request*:**

- ✓ This message is sent by the server at any time, but may be ignored by the client if the Handshake Protocol is already underway. A client who receives a hello request while in a handshake negotiation state should simply ignore the message.

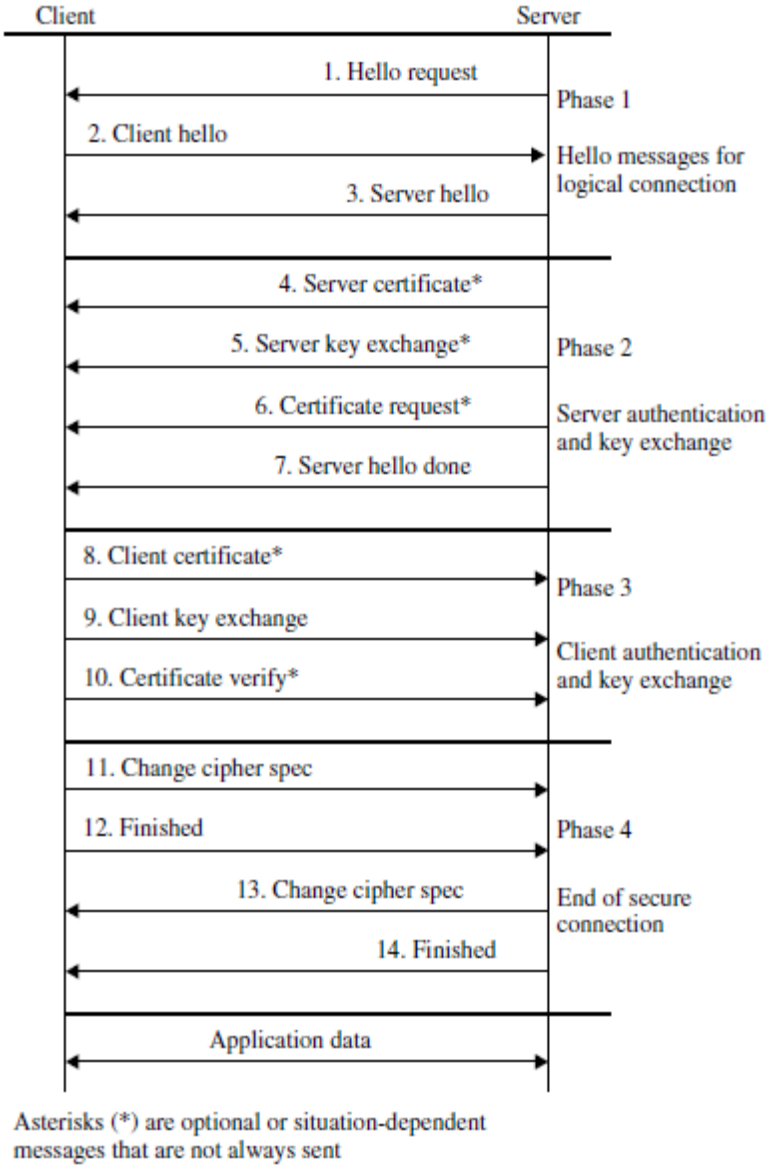


Figure 8.5 SSL Handshake Protocol.

***Client hello*:** The exchange is initiated by the client. A client sends a client hello message using the session ID of the session to be resumed. The server then checks its session cache for a match. If a match is found, the server will send a server hello message with the same session ID value. The client sends a client hello message with the following parameters:

***Client version*:** This is the version of the SSL protocol in which the client wishes to communicate during this session. This should be the most recent (highest-valued) version supported by the client. The value of this version will be 3.0.

***Random*:** This is a client-generated random structure with 28 bytes generated by a secure random number generator.

***Session ID*:** This is the identity of a session when the client wishes to use this connection. A nonzero value indicates that the client wishes to update the parameters of an existing connection or create a new connection in this session.

Cipher suites: This is a list of the cryptographic options supported by the client, with the client's first preference first. The single cipher suite is an element of a list selected by the server from the list in ClientHello.cipher suites.

***Compression method*:** This is a list of the compression methods supported by the client, sorted by client preference. If the session ID field is not empty, it must include the compression method from that session.

Server hello: The server will send the server hello message in response to a client hello message when it has found an acceptable set of algorithms. If it is unable to find such a match, it will respond with a handshake failure alert.

The structure of this message consists of: server version, random, session ID, cipher suite and compression method.

Server version: This field will contain the lower-valued version suggested by the client in the client hello and the highest-valued version supported by the server. The value of this version is 3.0.

Random: This structure is generated by the server and must be different from ClientHello.random.

Session ID: This field represents the identity of the session corresponding to this connection. If the ClientHello. session id is non-empty, the server will look in its session cache for a match

Cipher suite: This is the single cipher suite selected by the server from the list in ClientHello.cipher suites. For a resumed session, this field is the value from the state of the session being resumed.

Compression method: This is the single compression algorithm selected by the server from the list in ClientHello.compression methods. For a resumed sessions, this field is the value from the resumed session state.

Phase 2: Server Authentication and Key Exchange

The server is authenticated, it may request a certificate from the client, if that is appropriate to the cipher suite selected. Then the server will send the server hello done message, indicating that the hello message phase of the handshake is complete. The

server will then wait for a client response. If the server has sent a certificate request message, the client must send the certificate message.

***Server certificate*:**

- ✓ If the server is to be authenticated, it must send a certificate immediately following the server hello message.
- ✓ The certificate type must be appropriate for the selected cipher suite's key exchange algorithm, and is generally an X.509 v3 certificate.
- ✓ It must contain a key which matches the key exchange method. The signing algorithm for the certificate must be the same as the algorithm for the certificate key.

***Server key exchange message*:** The server key exchange message is sent by the server only when it is required. This message is not used if the server certificate contains Diffie–Hellman parameters, or RSA key exchange is to be used for a signatureonly RSA.

– params: the server's key exchange parameters.

– signed-params: for non-anonymous key exchange, a hash of the corresponding params value, with the signature appropriate to that hash applied.

As usual, a signature is created by taking the hash of the message and encrypting it with the sender's public key. Hence, the hash is defined as:

md5-hash : MD5(ClientHello.random||ServerHello.random||serverParams)

sha-hash : SHA(ClientHello.random||ServerHello.random||serverParams)

enum {anonymous, rsa, dsa} SignatureAlgorithm;

For a DSS signature, the hash is performed using the SHA-1 algorithm. In the case of an RSA signature, both an MD5 and an SHA-1 hash are calculated, and the concatenation of the two hashes is encrypted with the server's public key.

***Certificate request message*:** A non-anonymous server can optionally request a certificate from the client, if appropriate for the selected cipher suite. This message includes two parameters, certificate type and certificate authorities. Its structure is as follows:

enum{

rsa_sign(1), des_sign(2), rsa_fixed_dh(3),

dss_fixed_dh(4),

rsa_ephemeral_dh(5), dss_ephemeral_dh(6),

fortezza_dms(20), (255)

```
} ClientCertificateType;
opaque DistinguishedName<1..216-1>;
struct {
ClientCertificateType certificate_types<1..28-1>;
DistinguishedName certificate_authorities<3..216-1>
} CertificateRequest;
```

certificate types: This field is a list of the types of certificates requested, sorted in order of the server's preference.

certificate authorities: This is a list of the distinguished names of acceptable certificate authorities. These distinguished names may specify a desired distinguished name for a root CA or for a subordinate CA; thus, this message can be used to describe both known roots and a desired authorization space.

Phase 3: Client Authentication and Key Exchange

***Client certificate message*:** This is the first message the client can send after receiving a server hello done message. This message is sent only when the server requests a certificate. If no suitable certificate is available, the client should send a certificate message containing no certificates.

***Client key exchange message*:** This message is always sent by the client. It will immediately follow the client certificate message, if it is sent. Otherwise it will be the first message sent by the client after it receives the server hello done message.

***Certificate verify message*:** This message is used to provide explicit verification of a client certificate. The message is only sent following any client certificate that has signing capability (i.e. all certificates except those containing fixed Diffie–Hellman parameters).

Phase 4: End of Secure Connection

Change cipher spec messages: The client sends a change cipher spec message and copies the pending CipherSpec in the current CipherSpec.

This message is immediately sent after the certificate verify message that is used to provide explicit verification of a client certificate.

It is essential that a change cipher spec message is received between the other handshake messages and the finished message. It is a fatal error if a change cipher spec message is not preceded by a finished message at the appropriate point in the handshake.

***Finished message*:** This is always sent immediately after a change cipher spec message to verify that the key exchange and authentication processes were successful. The content of the finished message is the concatenation of two hash values:

MD5(master_secret||pad2||MD5(handshake_messages||Sender||
master_secret||pad1))

SHA(master_secret||pad2||SHA(handshake_messages||Sender||
master_secret||pad1))

where 'Sender' is a code that identifies that the sender is the client and 'handshake messages' is code that identifies the data from all handshake messages up to but not including this message.

6. Explain in detail about the Cryptographic Computations

The key exchange, authentication, encryption and MAC algorithms are determined by the cipher suite selected by the server and revealed in the server hello message. The compression algorithm is negotiated in the hello messages, and the random values are exchanged in the hello messages.

Computing the Master Secret

- ✓ For all key exchange methods, the same algorithm is used to convert the premaster secret into the master secret.
- ✓ In order to create the master secret, a premaster secret is first exchanged between two parties and then the master secret is calculated from it.
- ✓ The master secret is always exactly 48 bytes (384 bits) shared between the client and server. But the length of the premaster secret is not fixed and will vary depending on the key exchange method.

There are two ways for the exchange of the premaster secret:

***RSA*:** When RSA is used for server authentication and key exchange, a 48-byte premaster secret is generated by the client, encrypted with the server's public key and sent to the server. The server decrypts the ciphertext (of the premaster secret) using its private key to recover the premaseter secret. Both parties then convert the premaster secret into the master secret as specified below.

Diffie–Hellman: A conventional Diffie–Hellman computation is performed. Both client and server generate a Diffie-Hellman common key. This negotiated key is used as the premaster secret and is converted into the master secret, as specified below. The client and server then compute the master secret as follows:

master_secret = MD5(pre_master_secret||SHA('A'||
pre_master_secret||ClientHello.random||
ServerHello.random))||
MD5(pre_master_secret||SHA('BB'||
pre_master_secret||ClientHello.random||
ServerHello.random))||
MD5(pre_master_secret||SHA('CCC'||
pre_master_secret||ClientHello.random||
ServerHello.random))

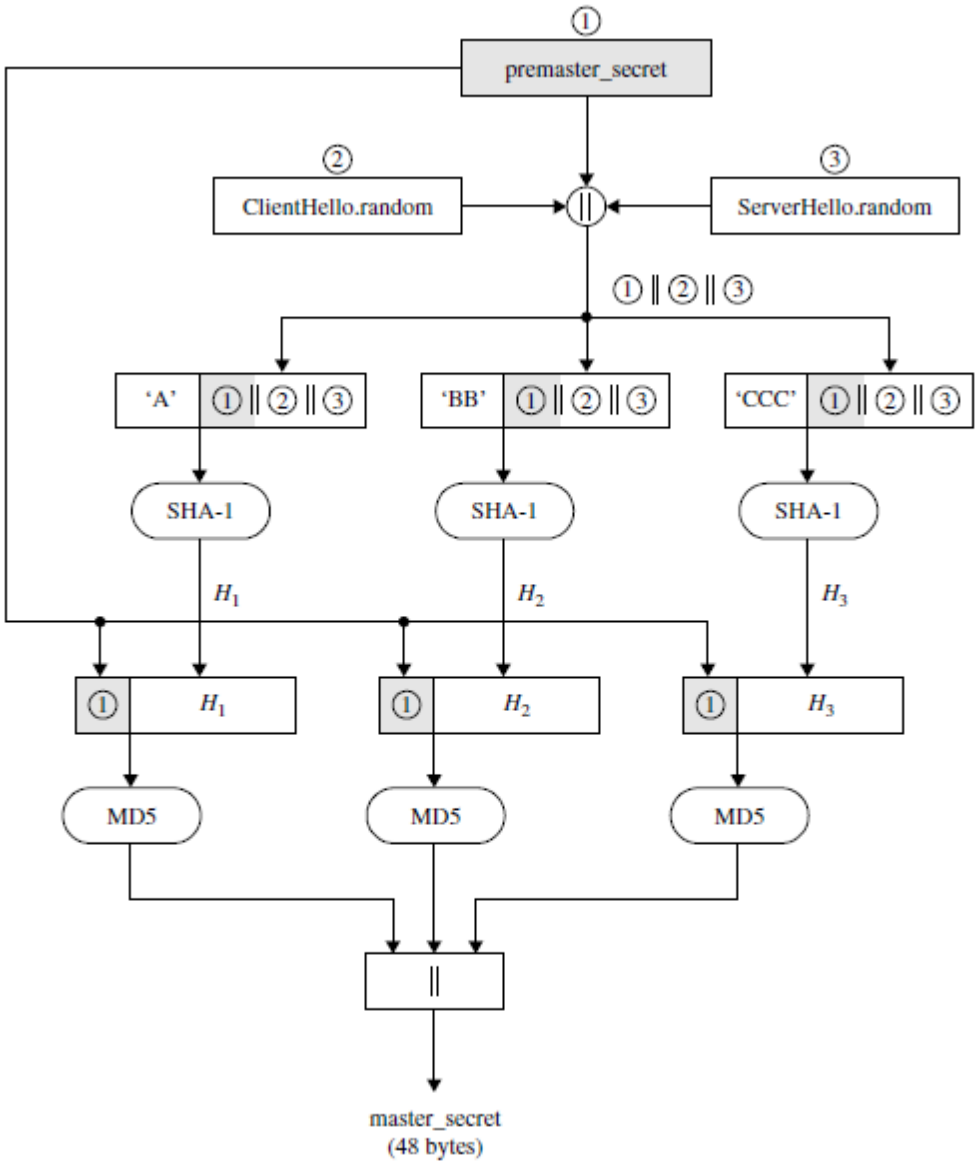


Figure 8.6 Computation of the master secret.

Converting the Master Secret into Cryptographic Parameters

CipherSpec specifies the bulk data encryption algorithm and a hash algorithm used for MAC computation, and defines cryptographic attributes such as the hash size.

To generate the key material, the following is computed:

key_block = MD5(master_secret||SHA('A'||master_secret||
ServerHello.random||ClientHello.random))||
MD5(master_secret||SHA('BB'||master_secret||
ServerHello.random||ClientHello.random))||
MD5(master_secret||SHA('CCC'||master_secret||
ServerHello.random||ClientHello.random))||...

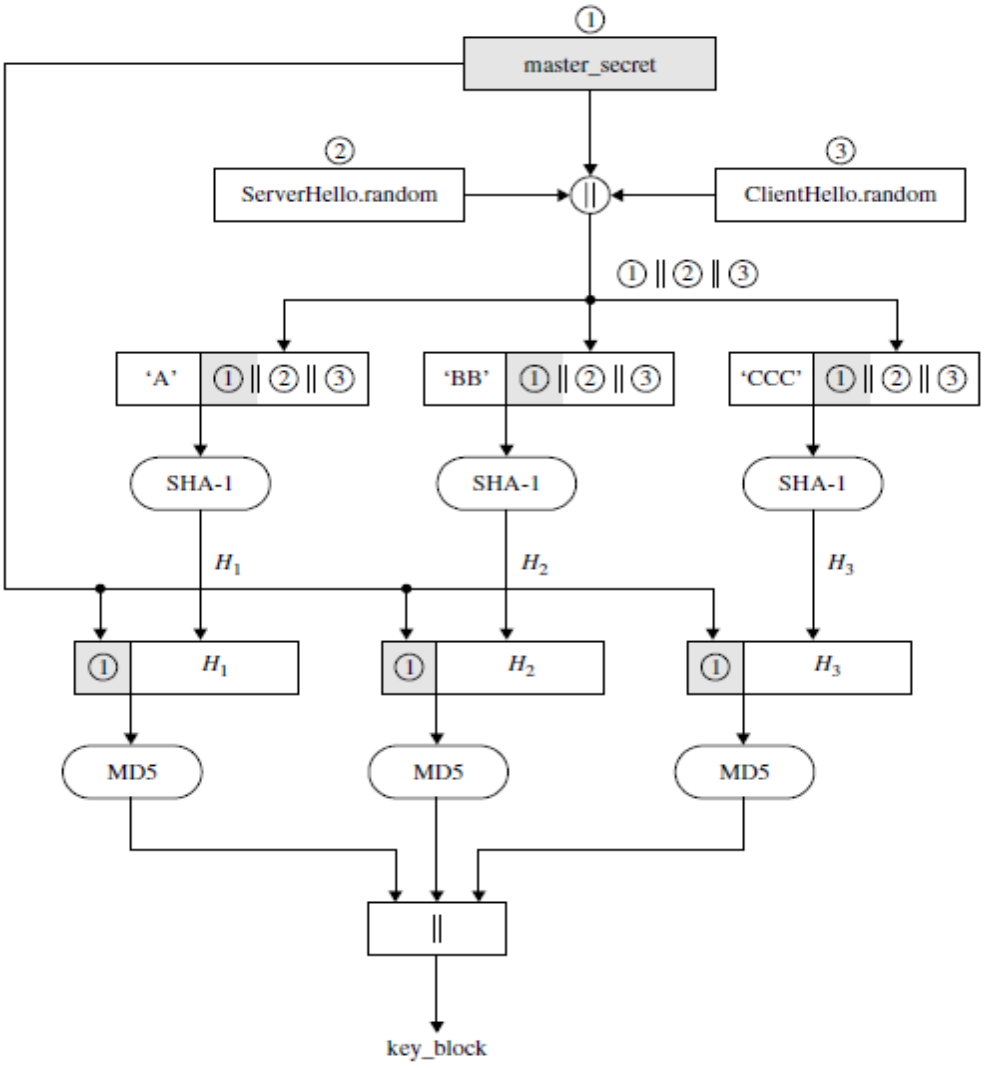


Figure 8.7 Generation of key block.

7. Explain in detail about the TLS Protocol.

TLS Protocol

- ✓ The TLSv1 protocol itself is based on the SSLv3 protocol specification as published by Netscape. Many of the algorithm-dependent data structures and rules are very close so that the differences between TLSv1 and SSLv3 are not dramatic. The current work on TLS is aimed at producing an initial version as an Internet standard. It is recommended that readers examine the comparative studies between the TLSv1 of RFC 2246 and SSLv3 of Netscape. In this section, we will not repeat every detailed step of identical protocol contents, but only highlight the differences.

HMAC Algorithm

- ✓ A Keyed-hashing Message Authentication Code (HMAC) is a secure digest of some data protected by a secret. Forging the HMAC is infeasible without knowledge of the MAC secret. HMAC can be used with a variety of different hash algorithms, namely MD5 and SHA-1, denoting these as HMAC MD5(secret, data) and HMAC SHA-1(secret, data).

There are two differences between the SSLv3 and TLSMAC schemes. TLS makes use of the HMAC algorithm defined in RFC 2104. HMAC was fully discussed in Chapters 4 and 7 and defined as:

$$\mathbf{HMAC} = H[(K \oplus \mathbf{opad}) || H[(K \oplus \mathbf{ipad}) || M]]$$

where

ipad = 00110110*(*0x36*)* repeated 64 times (512 bits)

opad = 01011100*(*0x5c*)* repeated 64 times (512 bits)

H = one-way hash function for TLS (either MD5 or SHA-1)

M = message input to HMAC

K = padded secret key equal to the block length of the hash code (512 bits for MD5 and SHA-1)

The following explains the HMAC equation:

1. Append zeros to the end of *K* to create a *b*-byte string (i.e. if *K* = 160 bits in length and *b* = 512 bits, then *K* will be appended with 352 zero bits or 44 zero bytes 0x00).
2. XOR (bitwise exclusive-OR) *K* with ipad to produce the *b*-bit block computed in step 1.
3. Append *M* to the *b*-byte string resulting from step 2.
4. Apply *H* to the stream generated in step 3.
5. XOR (bitwise exclusive-OR) *K* with opad to produce the *b*-byte string computed in

step 1.

6. Append the hash result *H* from step 4 to the *b*-byte string resulting from step 5.

7. Apply *H* to the stream generated in step 6 and output the result.

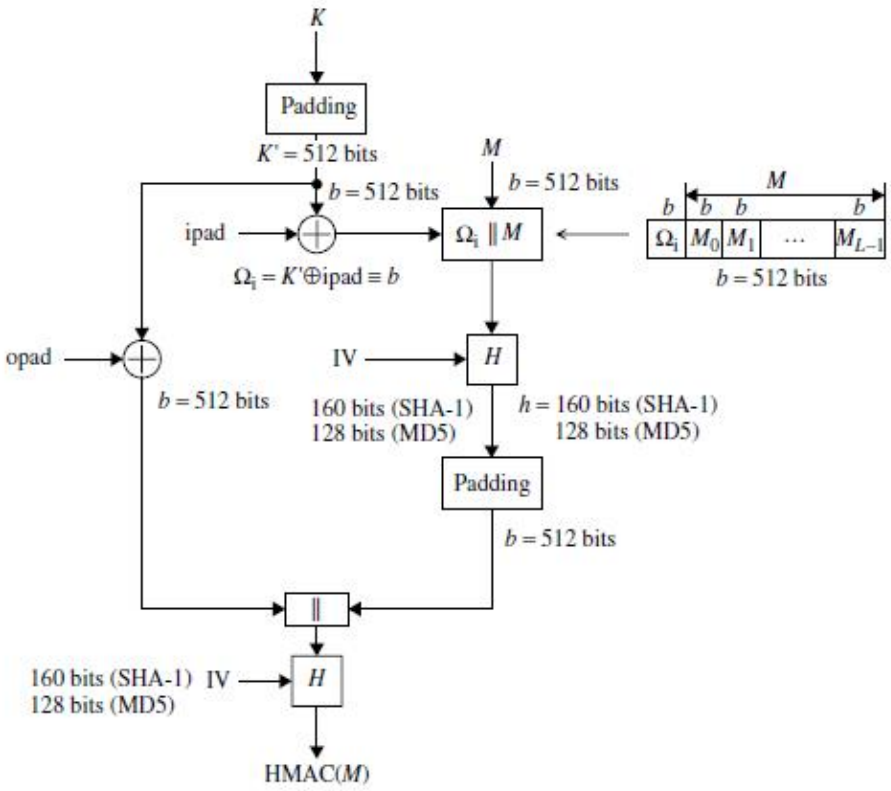


Figure 8.8 Overall operation of HMAC computation using either MD5 or SHA-1 (message length computation based on $\Omega_i \| M$).

Example 8.1 HMAC–SHA-1 computation using RFC method:

Data : 0x 7104f218 a3192e65 1cf7025d 8011bf79 4a19

Key : 0x 31fa7062 c45113e3 2679fd13 53b71264

–	A	B	C	D	E
IV	67452301	efcdab89	98badcfe	10325476	c3d2e1f0
$H[(K \oplus \text{ipad}) \| M]$	8efeef30	f64b360f	77fd8236	273f0784	613bbd4b
$H[(K \oplus \text{opad}) \| H[(K \oplus \text{ipad}) \| M]]$	31db10b8	ed346850	d0f0b7dd	50fd71f4	2dacd24c

HMAC–SHA-1 = 0x 31 db10b8 ed346850 d0f0b7dd 50fd71f4 2dacd24c

The alternative operation for computation of either HMAC–MD5 or HMAC–SHA-1 is described in the following:

1. Append zeros to *K* to create a *b*-bit string *K*, where *b* = 512 bits.

2. XOR *K* (padding with zero) with ipad to produce the *b*-bit block.

3. Apply the compression function f(IV, $K\_\oplus$ ipad) to produce (IV)i = 128 bits.
4. Compute the hash code h with (IV)i and Mi .
5. Raise the hash value computed from step 4 to a b-bit string.
6. XOR $K\_$ (padded with zeros) with opad to produce the b-bit block.
7. Apply the compression function f(IV, $K\_\oplus$ opad) to produce (IV)o = 128 bits.
8. Compute the HMAC with (IV)o and the raised hash value resulting from step 5.

Pseudo-random Function

- ✓ TLS utilizes a pseudo-random function (PRF) to expand secrets into blocks of data for the purposes of key generation or validation. The PRF takes relatively small values such as a secret, a seed and an identifying label as input and generates an output of arbitrary longer blocks of data.

P_hash(secret, seed) = HMAC_hash (secret, A(1)||seed) ||
HMAC_hash (secret, A(2)||seed) ||
HMAC_hash (secret, A(3)||seed) ||. . .

where A() is defined as:

A(0) = seed

A(i) = HMAC hash(secret, A(i-1)) and || indicates concatenation.

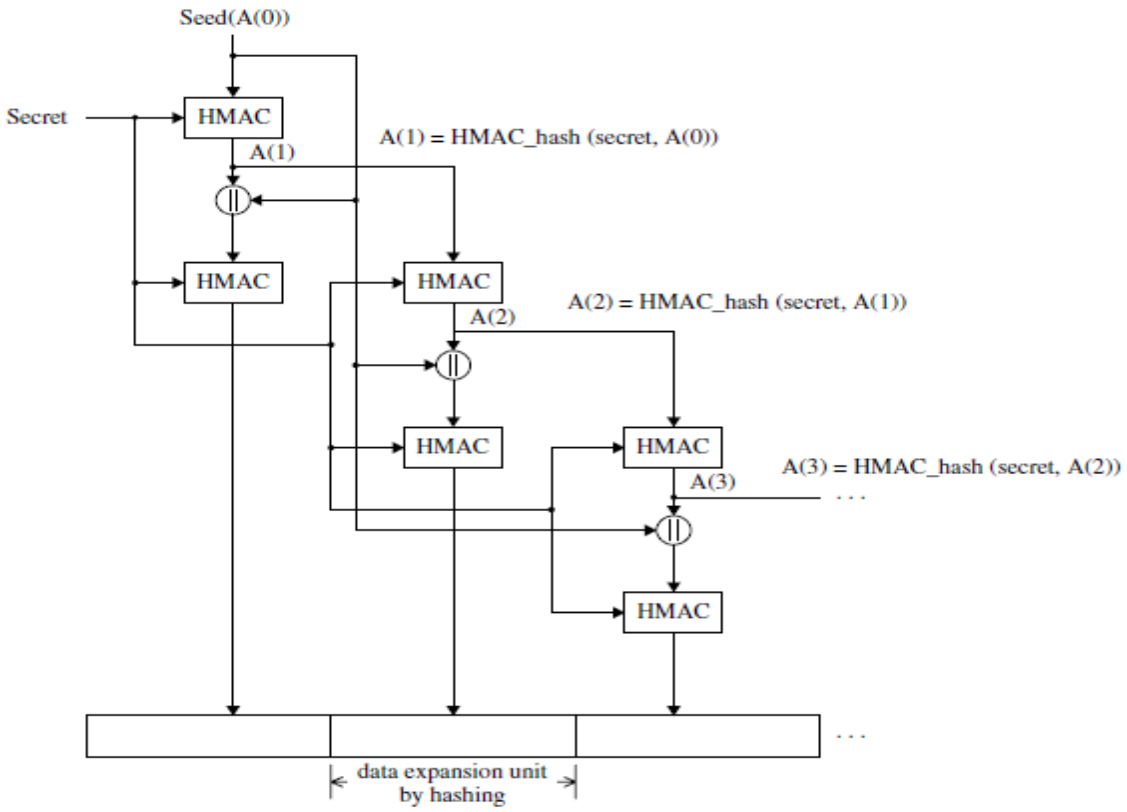


Figure 8.10 TLS data expansion mechanism using P_hash(secret,seed).

Thus, if the original secret is an *odd* number of bytes long, the last bytes of S1 will be the same as the first byte of S2:

L S = length in bytes of secret

L S1 = L *S*2 = ceil*(L S/*2*)*

The PRF is then defined as the result of mixing the two pseudo-random streams by XORing them together. The PRF is defined as:

PRF(secret, label, seed) = P MD5(S1, label||seed) ⊕ P SHA − 1(S2, label||seed)

The label is an ASCII string. Figure 8.11 illustrates the PRF generation scheme to expand secrets into blocks of data.

Example 8.3 Refer to Figure 8.11. Suppose the following parameters are given:

seed = 0x 80 af 12 5c 7e 36 f3 21

label = rocky mountains = 0x 72 6f 63 6b 79 20 6d 6f 75 6e 74 61 69 6e 73

secret = 0x 35 79 af 12 c4

Then

label||seed = 0x 72 6f 63 6b 79 20 6d 6f 75 6e 74 61 69 6e 73 80 af 12 5c 7e 36 f3 21

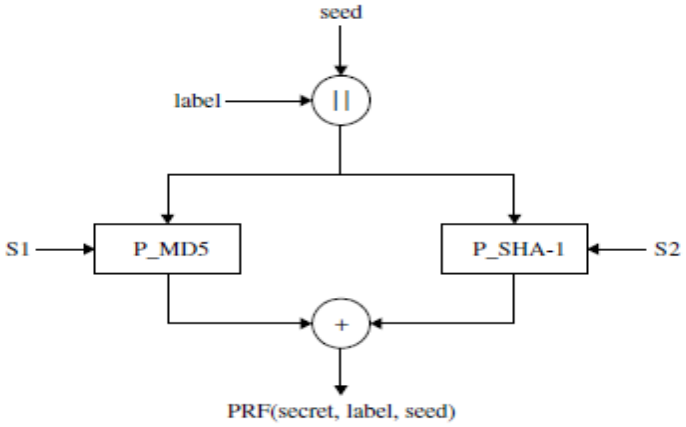
= A*(0)*

S1 = 0x 35 79 af for P MD5, S2 = 0x af 12 c4 for P SHA − 1

Data expansion by P MD5:

A(1) = HMAC MD5(S1, A(0))

= d0 de 36 53 79 78 04 a0 21 b8 6f f8 29 60 d5 f7



S1: First half of the secret
S2: Second half of the secret

P_MD5: Data expansion function to expand a secret S1 and (seed||secret) using MD5

P_SHA-1: Data expansion function to expand a secret S2 and (seed||secret) using SHA-1

Figure 8.11 A pseudo-random function (PRF) generation scheme.

HMAC MD5(S1, A(1)||A(0))

= 32 fd b3 70 eb 36 11 70 a4 3b 50 a9 fb ea 2a ec

A(2) = HMAC MD5(S1, A(1))

= 8c ce 5b 50 02 af 75 91 e7 20 cd 86 d9 3e 67 9d

HMAC MD5(S1, A(2)||A(0))

= 1f a8 4c af 5d e1 20 01 ea b0 38 6a a5 76 f9 8e

A(3) = HMAC MD5(S1, A(2))

= 45 48 5d 00 4e 64 07 45 eb 2c 18 60 7c e6 fa 1f

HMAC MD5(S1, A(3)|| A(0))

= f0 23 29 d9 5e 89 4b 70 cc 45 f8 aa 1f 58 8e 55

A(4) = HMAC MD5(S1, A(3))

= 87 39 c6 d3 7a b f8 e3 29 79 3a ae 63 24 6a ff

HMAC MD5(S1, A(4)|| A(0))

= 2e 0c 27 26 d0 b4 78 85 09 a2 69 1c 1b 1b d7 8d

A(5) = HMAC MD5(S1, A(4))

= 3a 2c aa d8 b3 ec 2e 5d 40 1c 39 bd 3e 48 1a d9

HMAC MD5(S1, A(5)|| A(0))

= 92 f2 63 5d 88 3a dd bf 8d ec e1 cf 0c 5c 8f 4c

where S1 = 0x 35 79 af = first half of the secret, and

A(0) = label||seed

Thus, P MD5 equals:

32 fd b3 70 eb 36 11 70 a4 3b 50 a9 fb ea 2a ec

1f a8 4c af 5d e1 20 01 ea b0 38 6a a5 76 f9 8e

f0 23 29 d9 5e 89 4b 70 cc 45 f8 aa 1f 58 8e 55

2e 0c 27 26 d0 b4 78 85 09 a2 69 1c 1b 1b d7 8d

92 f2 63 5d 88 3a dd bf 8d ec e1 cf 0c 5c 8f 4c (80 bytes)

Data expansion by P SHA-1 :

A(1) = HMAC SHA1(S2, A(0))

= aa ea 46 1b a6 ad 43 34 51 f8 c6 ef 70 dd f4 60 ca b9 40 2f

HMAC SHA1(S2, A(1)|| A(0))

= d0 8a d5 07 e0 b8 30 78 70 d9 c8 bb dd ba f5 a3 d0 77 49 e8

A(2) = HMAC SHA1(S2, A(1))

= 33 fd 23 41 01 ce 06 f8 c0 2b b3 e6 54 21 1c f4 6c 88 ab da

HMAC SHA1(S2, A(2)|| A(0))

= 64 b5 cc 3f 79 31 5b 5d e6 e4 4f eb 98 a8 bf 3f 97 13 38 e1

A(3) = HMAC SHA1(S2, A(2))

= 86 1f a3 a5 37 58 41 71 f1 9f a5 f3 48 2e 5d 84 7c a8 b6 52

HMAC SHA1(S2, A(3)|| A(0))

= 03 26 11 02 ce 69 74 4a 21 f4 76 55 13 af 77 80 2d fb 2f 36

A(4) = HMAC SHA1(S2, A(3))

= 9c 4d 01 3a 8c 48 54 42 68 07 4d f1 f0 a9 78 c3 6f ab d8 b4

HMAC SHA1(S2, A(4)|| A(0))

= 48 56 04 b5 b4 5f 9b d8 c7 2f 28 f6 9e 1d 8a c4 72 9a b9 32

where S2 = 0x af 12 c4 = second half of the secret, and

A(0) = label||seed

Thus, P SHA1 equals:

d0 8a d5 07 e0 b8 30 78 70 d9 c8 bb dd ba f5 a3

d0 77 49 e8 64 b5 cc 3f 79 31 5b 5d e6 e4 4f eb

98 a8 bf 3f 97 13 38 e1 03 26 11 02 ce 69 74 4a

21 f4 76 55 13 af 77 80 2d fb 2f 36 48 56 04 b5

b4 5f 9b d8 c7 2f 28 f6 9e 1d 8a c4 72 9a b9 32 (80 bytes)

Finally, P MD5 $\oplus$ P SHA − 1 equals:

e2 77 66 77 0b 8e 21 08 d4 e2 98 12 26 50 df 4f

cf df 05 47 39 54 ec 3e 93 81 63 37 43 92 b6 65

68 8b 96 e6 c9 9a 73 91 cf 63 e9 a8 d1 31 fa 1f

0f f8 51 73 c3 1b 0f 05 24 59 46 2a 53 4d d3 38

26 ad f8 85 4f 15 f5 49 13 f1 6b 0b 7e c6 36 7e (80 bytes)

Error Alerts

- ✓ The Alert Protocol is classified into the closure alert and the error alert. One of the content types supported by the TLS Record Layer is the alert type. Alert messages convey the severity of the message and a description of the alert. Alert messages with a fatal level result in the immediate termination of the connection.
- ✓ The client and the server must share knowledge that the connection is ending in order to avoid a truncation attack. Either party may initiate a close by sending a close notify alert. This message notifies the recipient that the sender will not send any more messages on this connection.
- ✓ Error handling in the TLS Handshake Protocol is very simple. When an error is detected, the detecting party sends a message to the other party. Upon transmission or receipt of a fatal alert message, both parties immediately close the connection.

TLS supports all of the error alerts defined in SSLv3 with the exception of additional alert codes defined in TLS. The additional error alerts are described in the following:

Decryption failed: A TLS ciphertext is decrypted in an invalid way: either it was not an even multiple of the block length or its padding values, when checked, were incorrect. This message is always fatal.

Record overflow: A TLS record was received with a ciphertext whose length exceeds 214 + 2048 bytes, or the ciphertext decrypted to a TLS compressed record with more than 214 + 1024 bytes. This message is always fatal.

unknown ca: A valid certificate chain or partial chain was received, but the certificate was not accepted because the CA certificate could not be located or could not be matched with a known, trusted CA. This message is always fatal.

access denied: A valid certificate was received, but when access control was applied, the sender decided not to proceed with the negotiation. This message is always fatal.

Decode error: A message could not be decoded because a field was out of its specified range or the length of the message was incorrect. This message was incorrect. It is always fatal.

decrypt error: A handshake cryptographic operation failed, including being unable to verify a signature, decrypt a key exchange or validate a finished message.

export restriction: A negotiation not in compliance with export restrictions was detected; for example, attempting to transfer a 1024-bit ephemeral RSA key for the RSA EXPORT handshake method. This message is always fatal.

protocol version: The protocol version the client has attempted to negotiate is recognized but not supported due to the fact that old protocol versions might be avoided for security reasons. This message is always fatal.

insufficient security: Returned instead of hanshake failure when a negotiation has failed specifically because the server requires ciphers more secure than those supported by the client. This message is always fatal.

internal error: An internal error unrelated to the peer or the correctness of the protocol, such as a memory allocation failure, makes it impossible to continue. This message is always fatal.

user canceled: This handshake is being cancelled for some reason unrelated to a protocol failure. If the user cancels an operation after the handshake is complete, just closing the connection by sending a close notify is more appropriate. This alert should be followed by a close notify. This message is generally a warning.

no renegotiation: This is sent by the client in response to a hello request or by the

server in response to a client hello after initial handshaking. Either of these messages would normally lead to renegotiation, but this alert indicates that the sender is not able to renegotiate. This message is always a warning.

Certificate Verify Message

Recall that the hash computations for SSLv3 are included with the master secret, the handshake message and pads. In the TLS certificate verify message, the MD5 and SHA-1 hashes are calculated only over handshake messages as shown below:

CertificateVerify.signature.md5_hash

MD5(handshake_message)

CertificateVerify.signature.sha_hash

SHA(handshake_message)

Here handshake messages refer to all handshake messages sent or received starting at client hello up to, but not including, this message, including the type and length fields of the handshake messages.

Finished Message

A finished message is always sent immediately after a change cipher spec message to verify that the key exchange and authentication processes were successful. It is essential that a change cipher spec message be received between the other handshake messages and the finished message. As with the finished message in SSLv3, the finished message in TLS is a hash based on the shared master secret, the previous handshake messages, and a label that identifies client and server. However, the TLS computation for verify data is somewhat different from that of the SSL calculation as shown below:

PRF(master_secret, finished_label, MD5(handshake_message)||
SHA-1(handshake_message))

where

- ✓ The finished label indicates either the string 'client finished' sent by the client or the string 'server finished' sent by the server, respectively.
- ✓ The handshake message includes all handshake messages starting at client hello up to, but not including, this finished message. This is only visible at the handshake layer and does not include record layer headers.

Cryptographic Computations (for TLS)

- ✓ In order to begin connection protection, the TLS Record Protocol requires specification of a suite of algorithms, a master secret, and the client and server random values.
- ✓ All that remains is to compute the master secret and the key block. The premaster secret for TLS is calculated in the same way as in SSLv3. The presmater_ secret should be deleted from memory once the master secret has been computed. As in SSLv3, the master secret in TLS in calculated as a hash function of the premaster secret and two hello random numbers. The TLS master secret computation is different from that of SSLv3 and is defined as follows:

master_secret = PRF(premaster_secret, "master secret",
ClientHello.random||ServerHello.random)

The master secret is always exactly 48 bytes (384 bits) in length. The length of the premaster secret will vary depending on key exchange method:

***RSA*:** When RSA is used for server authentication and key exchange, a 48-byte premaster secret is generated by the client, encrypted with the server's public key, and sent to the server. The server uses its private key to decrypt the premaster secret. Both parties then convert the premaster secret into the master secret, as specified above.

***Diffie–Hellman*:** A conventional Diffie–Hellman computation is performed. The negotiated key Z is used as the premaster secret, and is converted into the master secret, as specified above.

The computation of the key block parameters (MAC secret keys, session encryption keys and IVs) is defined as follows:

key_block = PRF(master_secret, "key expansion",
SecurityParameters.server_random||SecurityParameters.client_random)

until enough output has been generated. As with SSLv3, key block is a function of the master secret and the client and server random numbers, but for TLS the actual algorithm is different.

On leaving this chapter, it is recommended that readers search for and find any other small differences between SSL and TLS.

Important Questions

Part-A

1.Define IPsec Protocol.

2. write the basic components of IPsec architecture Protocol.

3.Define IPsec Protocol Documents

4. Define Security Associations (SAs)

5. Define Hashed Message Authentication Code (HMAC)

6. Define IP Authentication Header.

7. Draw AH Format.

8. Define IP ESP.

9. Define Packet Format

8. Define Key Management Protocol for IPSec.

9. List out the different types of Payload Types for ISAKMP.

10.Define SSL Protocol.

11.Draw the SSL Protocol Overview stack.

12.Difference between SSL Session and SSL Connection.

13.List out the SSL session elements.

14. List out the SSL Connection elements.

15.Define SSL Record Protocol format.

16. List out the phases of SSL Handshake Protocol.

17.How to compute the master-secret for *Diffie–Hellman.*

18. Define HMAC-Algorithm and how to calculate the HMAC Algorithm.

19. List out the ISAKMP Payload Processing.

20. Define Cryptographic Computations.

Part-B

1. Explain in detail about the IPSec Protocol.
2. Write short notes on the following protocol:
   - ✓ IP Authentication Header
   - ✓ IP ESP
3. Explain in detail about the Key Management Protocol for IPSec.
4. Explain in detail about the SSL protocol in Transport layer Security.
5. Explain in detail about the Cryptographic Computations in Transport layer Security.
6. Explain in detail about the TLS Protocol in Transport layer Security.