

UNIT -V:

Network Security-I

Security at application layer: PGP and S/MIME, Security at the Transport Layer: SSL and TLS

UNIT- VI: Network Security-II

Security at the Network Layer: IPSec, System Security

Ch-16- Security at the Application Layer: PGP and S/MIME

1. E-Mail System

Let us first discuss the electronic mail system in general.

1.1 E-mail Architecture

Figure 16.1 shows the most common scenario in a one-way e-mail exchange.

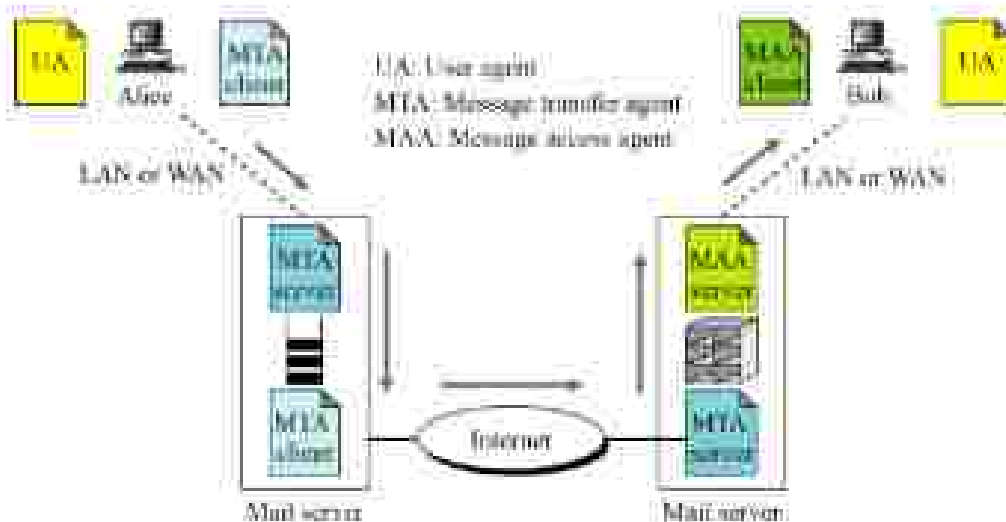


Figure 16.1 E-mail architecture

- When Alice needs to send a message to Bob, she invokes a *user agent (UA)* program to prepare the message. She then uses another program, a *message transfer agent (MTA)*, to send a message to the mail server at her site.
- Note that MTA is a client/server program with the client installed at Alice's computer and the server installed at the mail server.
- The message received at the mail server at Alice's site is queued with all other messages; each goes to its corresponding destination. In Alice's case, her message goes to the mail server at Bob's site. A client/server MTA is responsible for the e-mail transfer between the two servers.
- When the message arrives at the destination mail server, it is stored in Bob's mailbox, a special file that holds the message, until it is retrieved by Bob.
- When Bob needs to retrieve his message, including the one sent by Alice, he invokes another program, which we call a *message access agent (MAA)*.
- The MAA is also designed as a client/server program with the client installed at Bob's computer and the server installed at the mail server.
- There are several important points about the architecture of the e-mail system.

- The sending of an e-mail from Alice to Bob is a store-retrieve activity. Alice can send an e-mail today; Bob being busy, may check his e-mail three days later. During this time, the e-mail is stored in Bob's mailbox until it is retrieved.
- The main communication between Alice and Bob is through two application programs; the MTA client at Alice's computer and the MAA client at Bob's computer.
- The MTA client program is a push program; the client pushes the message when Alice needs to send it. The MAA client program is a pull program; the client pulls the messages when Bob is ready to retrieve his e-mail.
- Alice and Bob cannot directly communicate using MTA client at the sender site and an MTA server at the receiver site. This requires that the MTA server be running all the time, because Bob does not know when a message will arrive.

1.2 E-Mail Security

- Sending an e-mail is a one-time activity.
- Alice sends a message to Bob; sometime later, Bob reads the message and may or may not send a reply.

Cryptographic Algorithms

In e-mail security, the sender of the message needs to include the name or identifiers of the algorithms used in the message.

For example, Alice can choose triple DES for encryption/decryption and MD5 for hashing. When Alice sends a message to Bob, she includes the corresponding identifier for triple DES and MD5 in her message. Bob receives the message and extracts the identifiers first. He then knows which algorithm to use for decryption and which one for hashing.

Cryptographic secrets (keys)

In e-mail security, the encryption/decryption is done using a symmetric-key algorithm, but the secret key to decrypt the message is encrypted with the public key of the receiver and is sent with the message.

Certificates

- One more issue needs to be considered before we discuss any e-mail security protocol in particular. It is obvious that some public-key algorithms must be used for e-mail security.
- For example, we need to encrypt the secret key or sign the message. To encrypt the secret key, Alice needs Bob's public key; to verify a signed message, Bob needs Alice's public key.
- So, for sending a small authenticated and confidential message, two public keys are needed. *How can Alice be assured of Bob's public key, and how can Bob be assured of Alice's public key?* Each e-mail security protocol has a different method of certifying keys.

2. Pretty Good Privacy (PGP)

- PGP was invented by Phil Zimmermann to provide e-mail with *privacy*, *integrity*, and *authentication*.
- PGP can be used to create a secure e-mail message or to store a file securely for future retrieval.

2.1 Scenarios

Let us first discuss the general idea of PGP, moving from a simple scenario to a complex one.

Plaintext

- The simplest scenario is to send the e-mail message in plaintext as shown in [Figure 16.2](#).
- There is no message integrity or confidentiality in this scenario.
- Alice, the sender, composes a message and sends to Bob, the receiver.

The message is stored in Bob's mailbox until it is retrieved by him.

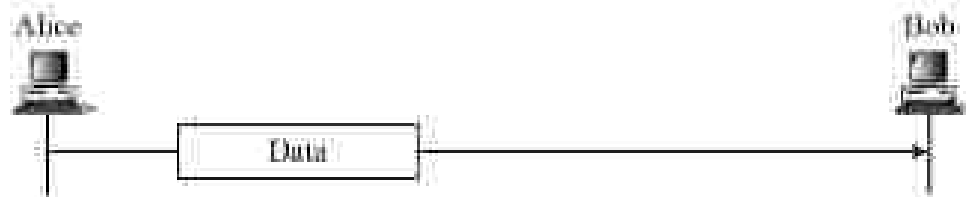


Figure 16.2 A plaintext message

Message Integrity and Authentication

- Probably the next improvement is to let Alice sign the message. Alice creates a digest of the message and signs it with her private key.
- When Bob receives the message, he verifies the message by using Alice's public key.



Figure 16.3 An authenticated message

Compression

- A further improvement is to compress the message and digest to make the packet more compact.
- This improvement has no security benefit, but it eases the traffic. [Figure 16.4](#) shows the new scenario.



Figure 16.4 A compressed message

Confidentiality with One-Time Session Key

- Confidentiality in an e-mail system can be achieved using conventional encryption with a one-time session-key.

- Alice can create a session key, use session key to encrypt the message and the digest, and send the key itself with the message.
- However, to protect the session key, Alice encrypts it with Bob's public key. **Figure 16.5** shows the situation.
- When Bob receives the packet, he first decrypts the key, using his private key to remove the key.
- He then uses the session key to decrypt the rest of the message.
- After decomposing the rest of the message, Bob creates a digest of the message and checks to see if it is equal to the digest sent by Alice.
- If it is, then the message is authentic.

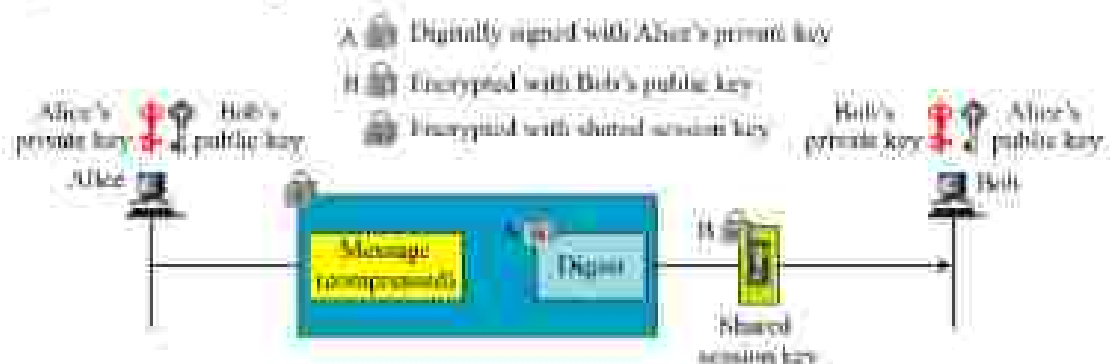


Figure 16.5 A confidential message

Code conversion

Another service provided by PGP is code conversion. *Most e-mail systems allow the messages to consist of only ASCII characters. To translate other characters not in the ASCII set, PGP uses Radix-64 conversion.* Each character to be sent (after encryption) is converted to Radix-64 code, which is discussed later in the chapter.

Segmentation

PGP allows segmentation of the message after it has been converted to Radix-64 to make each transmitted until the uniform size as followed by the underlying e-mail protocol.

2.2 Key Rings

- In all previous scenarios, we assumed that Alice needs to send a message only to Bob. That is not always the case.
- Alice may need to send messages to many people; she needs *key rings*.
- Alice may wish to use a different key pair for each group.
- Alice, for example, has several pairs of private/public keys belonging to her and public keys belonging to other people.

Note that everyone can have more than one public key, two cases may arise.

1. Alice needs to send a message to another person in the community.
 - a. She uses her private key to sign the digest.
 - b. She uses the receiver's public key to encrypt a newly created session key.
 - c. She encrypts the message and signed digest with the session key created.

2. Alice receives a message from another person in the community.
 - a. She uses her private key to decrypt the session key.
 - b. She uses the session key to decrypt the message and digest.
 - c. She uses her public key to verify the digest.

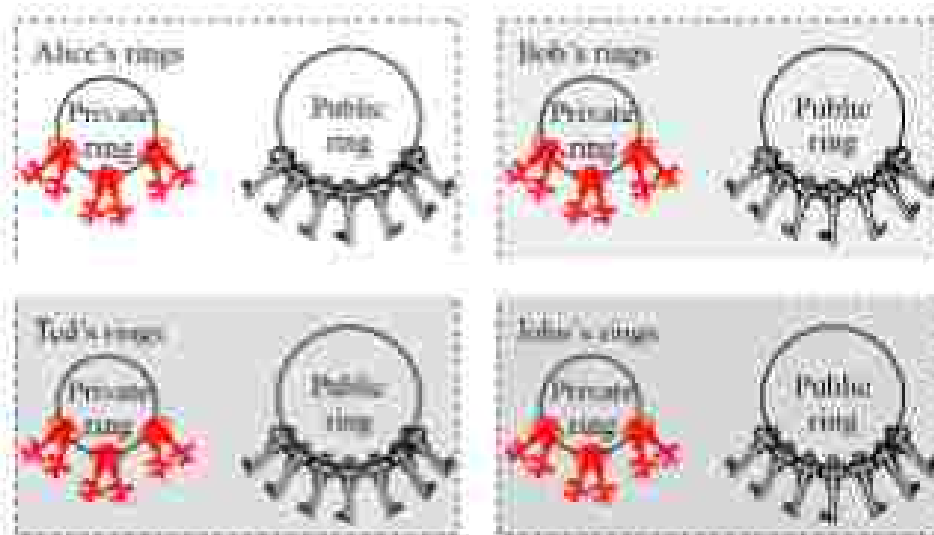


Figure 16.6 Key rings in PGP

PGP Algorithms

The following algorithms are used in PGP.

Public-Key Algorithm The public-key algorithms that are used for signing the digests or encrypting the messages are listed in below [table 16.1](#).

ID	Description
1	RSA (encryption or signing)
2	RSA (for encryption only)
3	RSA (for signing only)
16	EKEM (for encryption only)
17	DSS
18	Reserved for elliptic curve
19	Reserved for ECDSA
20	EKEM (for encryption or signing)
21	Reserved for Diffie-Hellman
100-255	Private algorithms

Table 16.1 Public-key algorithms

Symmetric-Key Algorithms

The Symmetric-key algorithms that are used for conventional encrypting are shown in [table 16.2](#).

ID	Description
0	No Encryption
1	IDEA
2	Triple DES
3	CAST-128
4	Blowfish
5	Serpent
6	Reserved for DES-SK
7	Reserved for AES-128
8	Reserved for AES-192
9	Reserved for AES-256
100-110	Private algorithms

Table 16.2 Symmetric-key algorithms

Hash Algorithms

The hash algorithms that are used for creating hashes in PGP shown in [table 16.3](#).

ID	Description
1	MD5
2	SHA-1
3	RIPE-MD160
4	Reserved for double with SHA
5	MD2
6	TIGER192
7	Reserved for HAVAL
100-110	Private algorithms

Table 16.3 Hash algorithms

Compression Algorithms

The compression algorithms that are used for compressing text are shown in [table 16.4](#).

Table 16.4 Compression methods

ID	Description
0	Uncompressed
1	ZIP
2	ZLIB
100-110	Private methods

Table 16.4 Compression algorithms

2.3 PGP Certificates

PGP, like other protocols we have seen so far, uses certificates to authenticate public keys. However, the process is totally different.

X.509 Certificates

- Protocol that use X.509 certificates depend on the hierarchical structure of the trust.

- There is a predefined chain of trust from the root to any certificate. Every user fully trusts the authority of the CA at the root level.
- The root issues certificates for the CAs at the second level, a second level CA issues a certificate for the third level, and so on.
- Every party that needs to be trusted presents a certificate from some CA in the tree.
- If Alice does not trust the certificate issuer for Bob, she can appeal to a higher level authority up to the root.
- In other words, there is one single path from a fully trusted CA to a certificate.

In X.509, there is a single path from the fully trusted authority to any certificate.

PGP Certificate

- In PGP, there is no need for CAs; anyone in the ring can sign a certificate for anyone else in the ring.
- Bob can sign a certificate for Ted, John, Anne, and so on. There is no hierarchy of trust in PGP; there is no tree.
- The lack of hierarchical structure may result in the fact that Ted may have one certificate from Bob and another certificate for Ted, there are two paths: one starts from Bob and one starts from Liz.
- There can be multiple paths in the line of trust from a fully or partially trusted authority to a certificate.
- In PGP, the issuer of a certificate is usually called an *introducer*.

Trusts and Legitimacy

- The entire operation of PGP is based on introducer trust, the certificate trust, and the legitimacy of the public keys.

Introducer Trust Levels

- With the lack of central authority.
- It is obvious that the ring cannot be very large if every user in the PGP ring of users has to fully trust everyone else.
- To solve this problem, PGP allows different levels of trust.
- The number of levels is mostly implementation dependent, but for simplicity, let us assign three levels of trust to any introducer, none, partial and full.
- The introducer trust level specifies trust levels issued by the introducer for other people in the ring.
- For example, Aline may fully trust Bob, partially trust Anne, and not trust John at all.
- There is no mechanism in PGP to determine how to make a decision about the trustworthiness of the introducer; it is up to the user to make this decision.

Certificate Trust Levels

- When Alice receives a certificate from an introducer, she stores the certificate under the name of the subject.
- She assigns a level of trust to this certificate.
- The certificate trust level is normally the same as the introducer trust level that issued the certificate.

- Assume that Alice fully trusts Bob, partially trusts Anne and Janette and has no trust in John. The following scenario can happens.
 1. Bob issues two certificates, one for Linda (with public key K1) and one for Lesley (with public key K2). Alice stores the public key and certificate for Linda under Linda's name and assigns a full level of trust to this certificate. Alice stores the certificate and public key for Lesley under Lesley's name and assigns a full level of trust to this certificate.
 2. Anne issues a certificate for John.
 3. Janette issues two certificates, one for John (with public key K3) and one for Lee (with public key K4).
 4. John issues a certificate for Liz.

Key Legitimacy

PGP defines a very clear procedure for determining key legitimacy. The level of the key legitimacy for a user is the weighted trust levels of that user. For example, suppose we assign the following weights to certificate trust levels:

1. A weight of 0 to a non-trusted certificate.
2. A weight of 1/2 to a certificate with partial trust.
3. A weight of 1 to a certificate with full trust.

Starting the Ring


In PGP, the key legitimacy of a trusted or partially trusted entity can be also determined by other methods:

1. Alice can physically obtain Bob's public key. For example, Alice and Bob can meet personally and exchange a public key written on a piece of paper or to a disk.
2. If Bob's voice is recognizable to Alice, Alice can call him and obtain his public key on the phone.
3. A better solution proposed by PGP is Bob to send his public key to Alice by e-mail. Both Alice and Bob make a 16-byte MD5 (or 20-Byte SHA-1) digest from the key. The digest is normally displayed as eight groups of 4- digits in hexadecimal and is called fingerprint. Alice can then call Bob and verify the fingerprint on the phone.
4. In PGP, nothing prevents Alice from getting Bob's public key from a CA in a separate procedure. She can then insert the public key in the public key ring.

Key Ring Tables

Each user such as Alice, keeps track of two key rings: one private key ring and one public key ring. PGP defines a structure for each of these key rings in the form of table.

Private Key Ring Table



User ID	Key ID	Public Key	Encrypted private key	Trust level
⋮	⋮	⋮	⋮	⋮

Figure 16.7 Format of private key ring table

- **User ID:** As in the private key ring table, the user ID is usually the e-mail address of the entity.
- **Key ID:** As in the private key ring table, the key ID is the first (least significant) 64 bits of the public key.
- **Public Key:** This is the public key of the entity.
- **Producer Trust:** This column defines the producer level of trust. In most implementations, it can only be of one of three values: none, partial, or full.
- **Certificate(s):** This column holds the certificate or certificates signed by other entities for this entity. A user ID may have more than one certificate.
- **Certificate Trust(s):** This column represents the certificate trust or trusts.
- **Key Legitimacy:** This value is calculated by PGP based on the value of the certificate trust and the predefined weight for each certificate trust.
- **Timestamp:** This column holds the date and time of the column creation.



User ID	Key ID	Public key	Producer trust	Certificate(s)	Certificate trust(s)	Key Legitimacy	Timestamp
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Public ring

2.4 Trust Model in PGP

As Zimmermann has proposed, we can create a trust model for any user in a ring with the user as the center of activity. Such a model can look like the one shown in [Figure 16.9](#). The diagram may change with any changes in the public key ring table.

The figure shows that there are three entities in Alice's ring with full trust (Alice, herself, Bob, and Ted). The figure also shows three entities with partial trust (Anne, Mark, and Bruce). There are also six entities with no trust. Nine entities have a legitimate key.

Alice can encrypt a message to any one of these entities or verify a signature received from one of these entities. There are also three entities that do not have any legitimate keys with Alice.

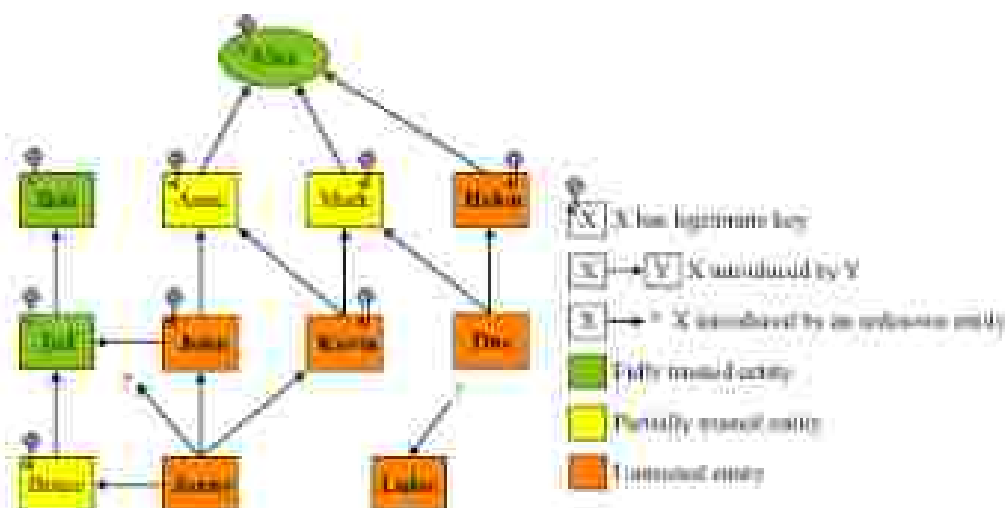


Figure 16.9 Trust model

Web of Trust

- PGP can eventually make a web of trust between a group of people. If each entity introduces more entities to other entities, the public key ring for each entity gets larger and larger and entities in the ring can send secure e-mail to each other.

2.5 Key Revocation

- It may become necessary for an entity to revoke his or her public key from the ring.
- This may happen if the owner of the key feels that the key is compromised or just too old to be safe.
- To revoke a key, the owner can send a revocation certificate signed by herself.
- The revocation certificate must be signed by the old key and disseminated to all the people in the ring that use that public key.

2.6 Extracting Information from Rings

- The sender and receiver each have two key rings, one private and one public.

Sender site

- Assume that Alice is sending an e-mail to Bob. Alice needs five pieces of information: the key ID of the public key she is using, her private key, the session key, Bob's public key ID, and Bob's public key.
- To obtain these five pieces of information, Alice needs to feed four pieces of information to PGP: her user ID, her passphrase, a sequence of key strokes with possible pauses, and Bob's user ID. (See [figure 16.10](#))
- Alice's public-key ID and her private key are stored in the private key ring table.
- Alice selects the user ID that she wants to use as an index to this ring.
- PGP extracts the key ID and the encrypted private key.
- PGP uses the predefined decryption algorithm and her hashed passphrase to decrypt this private key.

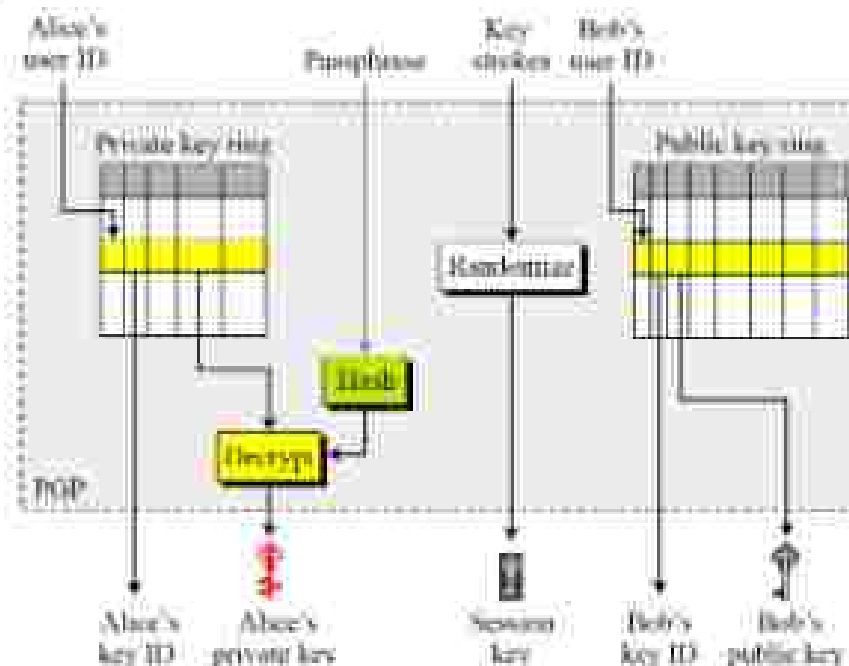


Figure 16.10 Extracting information at the sender site

- Alice also needs a secret session key. The session key in PGP is a random number with a size defined in the encryption / decryption algorithm.
- PGP uses a random number generator to create a random session key; the seed is a set of arbitrary key strokes typed by Alice on her keyboard.
- Each key stroke is converted to 8 bits and each pause between the keystrokes is converted to 32-bits.
- The combination goes through a complex random number generator to create a very reliable random number as the session key.
- Note that the session key in PGP is one-time random key and used only once.
- Alice also needs Bob's key ID and Bob's public key.
- These two pieces of information are extracted from the public key ring table using Bob's user ID.

Receiver Site

- At the receiver site, Bob needs three pieces of information: Bob's private key, the session key, and Alice's public key.
- Bob uses the key ID of this public key sent by Alice to find his corresponding private key needed to decrypt the session key.
- This pieces of information can be extracted from Bob's private key ring table.
- The private key, is encrypted when stored.
- Bob needs to use his passphrase and the hash function to decrypt it.
- The encrypted session key is sent with the message. Bob uses his decrypted private key to decrypt the session key.
- Bob uses Alice's key ID sent with the message to extract Alice's public key, which is stored in Bob's public key ring table.

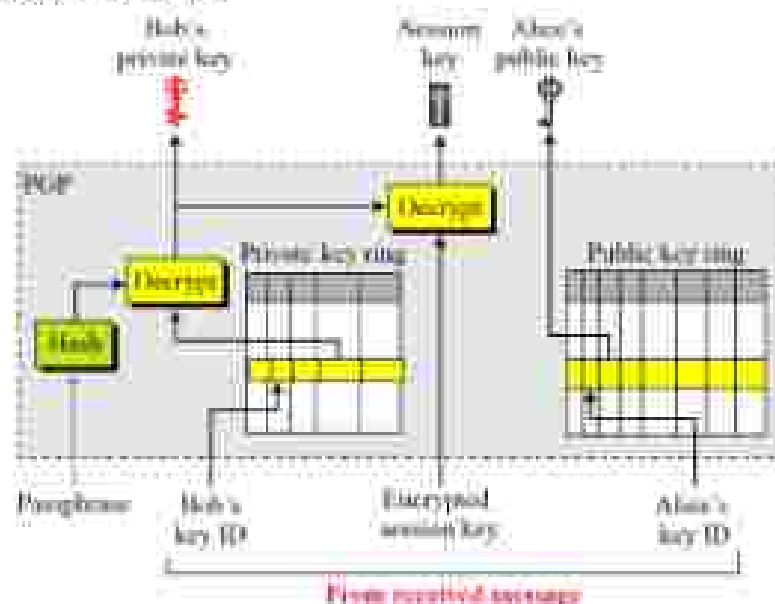


Figure 16-11 Extracting information at the receiver site

2.7 PGP Packets

- A message in PGP consists of one or more packets.

- During the evolution of PGP, the format and the number of packet types have changed.
- Like other protocols we have PGP has a generic header that applies to every packet.
- The generic header, in the most recent version, has only two fields as shown in [figure 16.12](#).

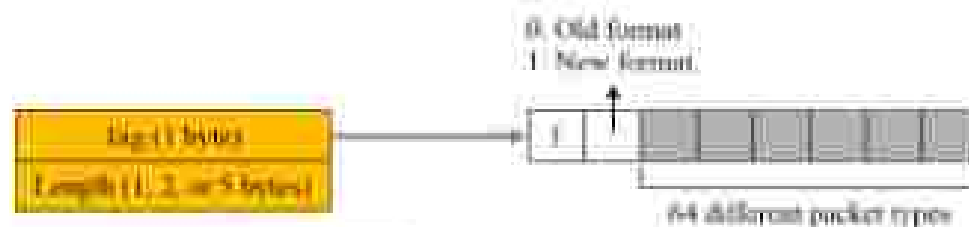


Figure 16.12 Format of packet header

- **Tag.** The recent format for this field defines a tag as an 8-bit flag; the first bit (most significant) is always 1. The second bit is 1 if we are using the latest version. The remaining six bits can define up to 64 different packet types, as shown in [table 16.12](#).
- **Length.** The length field defines the length of the entire packet in bytes. The size of this field is variable; it can be 1, 2, or 5 bytes. The receiver can determine the number of bytes of the length field by looking at the value of the byte immediately following the tag field.
 - a. If the value of the byte after the tag field is less than 192, the length field is only one byte. The length of the body (packet minus header) is calculated as:

$$\text{Body length} = \text{first byte}$$
 - b. If the value of the byte after the tag field is between 192 and 223 (inclusive), the length field is two bytes. The length of the body can be calculated as:

$$\text{Body length} = (\text{first byte} - 192) \ll 8 + \text{second byte} - 192$$
 - c. Parity body length = $1 \ll (\text{first byte} \& 0x1F)$
 - d. Body length = $\text{second byte} \ll 24 \mid \text{third byte} \ll 16 \mid \text{fourth byte} \ll 8 \mid \text{fifth byte}$

Tag	Packet type
1	Secret-key packet encrypted using a public key
2	Signature packet
3	Private-key packet
4	Public-key packet
5	Compressed data packet
6	Data packet encrypted with a secret key
11	Literal data packet
13	User ID packet

Table 16.12 Some commonly used packet types

Literal Data Packet

- The literal data packet is the packet that carries or holds the actual data that is being transmitted or stored.
- This packet is the most elementary type of message; that is, it cannot carry any other packet.
- The format of this packet is shown in [figure 16.13](#).

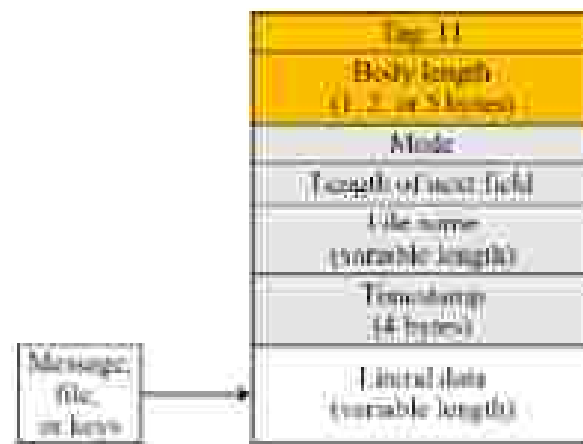


Figure 16.13 Literal data packet

- **Mode:** This one-byte field defines how data is written to the packet. The value of this field can be "b" for binary, "t" for text, or any other locally defined value.
- **Length of next field:** This one-byte field defines the length of the next field.
- **File name:** This variable-length field defines the name of the file or message as an ASCII string.
- **Timestamp:** This four-byte field defines the time of creation or last modification of the message. The value can be 0, which means that the user chooses not to specify a time.
- **Literal data:** This packet carries compressed data packets.

Compressed Data Packet

This packet carries compressed data packets. Figure 16.14 shows the format of a compressed data packet.

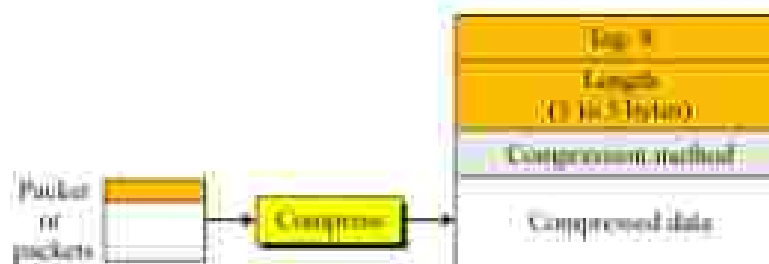


Figure 16.14 Compressed data packet

- **Compression method:** This one-byte field defines the compression method used to compress the data (next field). The values defined for this field so far are 1 (ZIP) and 2 (ZLIP). Also, an implementation can use other experimental compression methods.
- **Compressed data:** This variable-length field the data after compression.

Data Packet Encrypted with Secret Key

This packet carries data from one packet or a combination of packets that have been encrypted using a conventional symmetric-key algorithm. Figure 16.15 shows the format of the encrypted data packet.

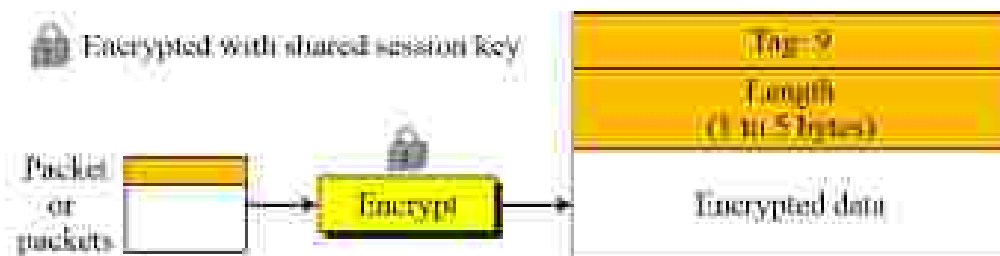


Figure 16.15 Encrypted data packet

Signature packet

A signature packet protects the integrity of the data. Figure 16.16 shows the format of the signature packet.

- **Version** This one-byte field defines the PGP version that is being used.
- **Length** This field was originally designed to show the length of the next two fields, but because the size of these fields is now fixed, the value of this field is 5.
- **Signature type** This one-byte field defines the purpose of the signature, the document is signed. Table 16.13 shows some signature types.
- **Timestamp** This four-byte field defines the time the signature was calculated.
- **Key ID** This eight-byte field defines the public-key ID of the signer. It indicates to the verifier which signer public key should be used to decrypt the digest.
- **Public-key algorithm** This one-byte field gives the code for the public-key algorithm used to encrypt the digest. The verifier uses the same algorithm to decrypt the digest.
- **Hash algorithm** This one-byte field gives the code for the hash algorithm used to create the digest.
- **First two bytes of message digest** These two bytes are used as a kind of checksum.
- **Signature** This variable length field is the signature. It is the encrypted digest signed by the sender.

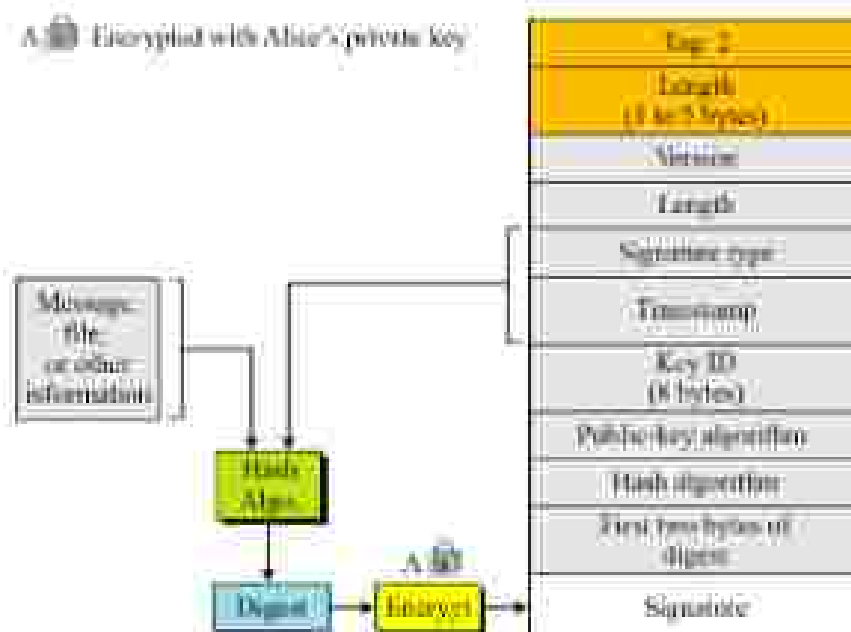


Figure 16.16 Signature packet

Value	Signature
0x00	Signature of a binary document (message or file)
0x01	Signature of a text document (message or file)
0x10	Generic certificate of a user ID and public-key packet. The signer does not make any particular assertion about the owner of the key.
0x11	Personal certificate of a user ID and public-key packet. No verification is done on the owner of the key.
0x12	Casual certificate of a User ID and public-key packet. Some casual verification done on the owner of the key.
0x13	Positive certificate of a user ID and public-key packet. Substantial verification done.
0x19	Certificate revocation signature. This removes an earlier certificate (0x10 through 0x13).

Table 16.13 Some signature values

Session key packet encrypted with public key

This packet is used to send the session key encrypted with the receiver public key. The format of the packet is shown in [figure 16.17](#).

- **Version** This one-byte field defines the PGP version being used.
- **Key ID** This eight-byte field defines the public-key ID of the sender. It indicates to the receiver which sender public key should be used to decrypt the session key.
- **Public key algorithm** This one-byte field gives the code for the public-key algorithm used to encrypt the session key. The receiver uses the same algorithm to encrypt the session key.

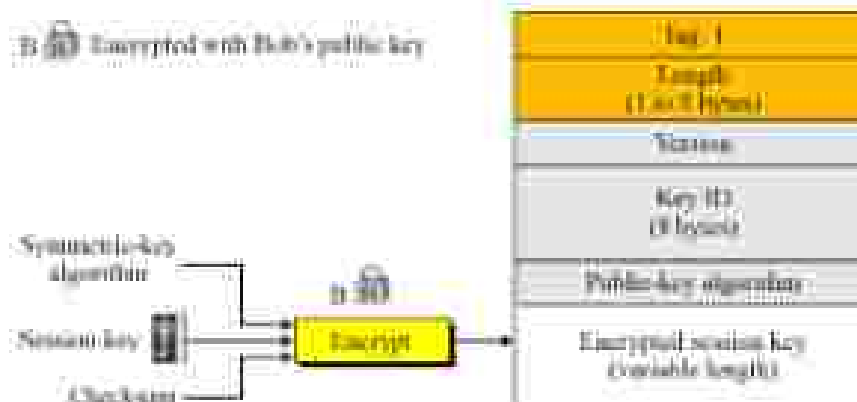


Figure 16.17 Session-key packet

- **Encrypted session** This variable-length field is the encrypted value of the session key created by the sender and sent to the receiver. The encryption is done on the following:
 - a. One-octet symmetric encryption algorithm.
 - b. The session key.
 - c. A two-octet checksum equal to the sum of the preceding session-key octets.

Public key packet

This packet contains the public key of the sender. The format of the packet is shown in [figure 16.18](#).



Figure 16.18 Public key packet

- **Version.** This one-byte field defines the PGP version of the PGP being used.
- **Timestamp.** This four-byte field defines the time when the key was created.
- **Validity.** This two-byte field shows the number of days the key is valid. If the value is 0, it means the key does not expire.
- **Public-key algorithm.** This one-byte field gives the code for the public-key algorithm.
- **Public key.** This variable-length field holds the public key itself. Its contents depend on the public-key algorithm used.

User ID Packet

This packet identifies a user and can normally associate the user ID contents with a public key of the sender. Figure 16.19 shows the format of the user ID packet.



Figure 16.19 User ID packet

- **User ID** This variable-length string defines the user ID of the sender. It is normally the name of the user followed by an e-mail address.

2.8 PGP Messages

A message in PGP is a combination of sequenced and/or nested packets. Even though not all possible combinations cannot make a message.

Encrypted message

An encrypted message can be a sequence of two packets, a session key packet and a symmetrically encrypted packet. Figure 16.20 shows this combination.

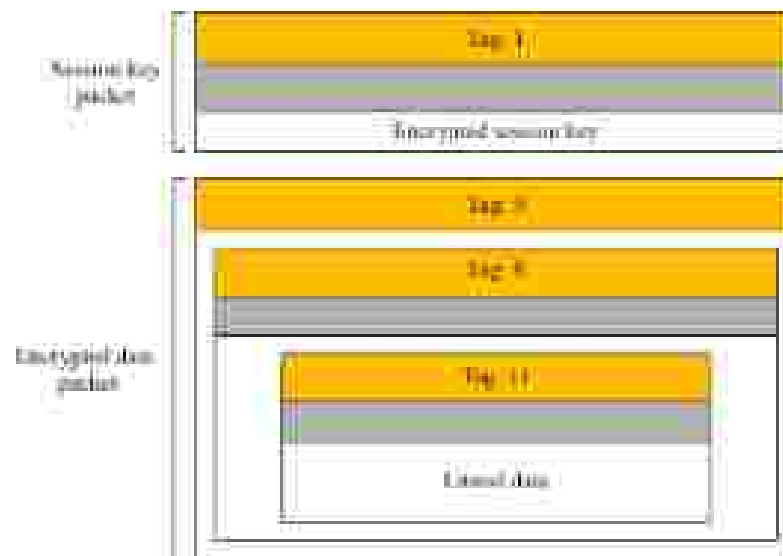


Figure 16.20 Encrypted message

Signed message

A signed message can be a combination of a signature packet and a literal packet as shown in Figure 16.21.



Figure 16.21 Signed message

Certificate Message

Although a certificate can take many forms, one simplest example is the combination of a user ID packet and a public-key packet as shown in Figure 16.22. The signature is then calculated on the concatenation of the key and the user ID.



Figure 16.22 Certificate message

2.9. Applications of PGP

PGP has been extensively used for personal e-mails.

3. S/MIME

Another security service designed for electronic mail is *Secure/Multipurpose Internet Mail Extension*, (**S/MIME**). It is an extension of MIME.

3.1. MIME

- Electronic mail has a simple structure. Its simplicity, however, comes with a price.
- It can send messages only in NVT 7-bit ASCII format.
- MIME is a supplementary protocol that allows non-ASCII data to be sent through e-mail.
- MIME transforms non-ASCII data at the sender site to NVT ASCII data and delivers it to the client MTA to be sent through the internet.
- The message at the receiving side is transformed back to the original data.
- We can think of MIME as a set of software functions that transform non-ASCII data to ASCII data and vice-versa as shown in figure 16.23.

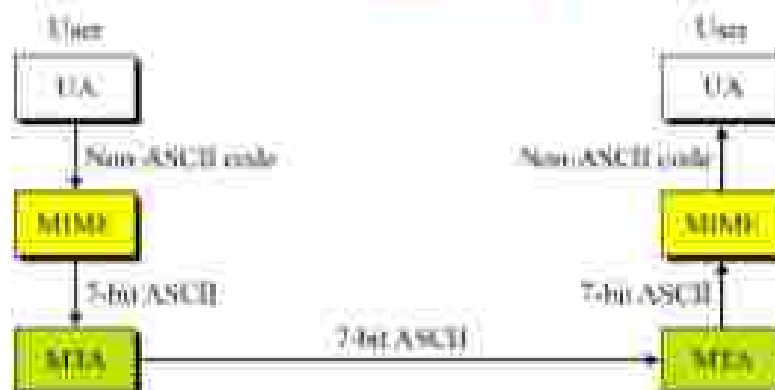


Figure 16.23 MIME

MIME defines five headers that can be added to the original e-mail header section to define the transformation parameters.

1. MIME-Version
2. Content-Type
3. Content-Transfer Encoding
4. Content-Id
5. Content-Description

Figure 16.24 shows the MIME header.



Figure 16.24 MIME header

MIME-Version

This header defines the version of MIME used. The current version is 1.1.

MIME-Version: 1.1

Content-Type

This header defines the type of data used in the body of the message. The content type and the content subtype are separated by a slash. Depending on the subtype, the header may contain other parameters.

Content-Type: <type/ subtype: parameters>

MIME allows seven different types of data. These are listed in table 16.14 and describe in more detail below.

- **Text.** The original message is 7-bit ASCII format and no transformation by MIME is needed. There are two subtypes currently used, plain and HTML.
- **Multipart.** The body contains multiple, independent parts. The multipart header needs to define the boundary between each part. A parameter is used for this purpose.

Type	Subtype	Description
Multipart	Plain	Unformatted.
	HTML	HTML format.
	Mixed	Body contains ordered parts of different data types.
	Parallel	Same as above, but no order.
	Digest	Similar to Mixed, but the default is message/RFC822.
Message	Alternative	Parts are different expressions of the same message.
	RFC822	Body is an encapsulated message.
	Partial	Body is a fragment of a bigger message.
Image	External-Body	Body is a reference to another message.
Image	JPEG	Image in JPEG format.
	GIF	Image in GIF format.
Video	MPEG	Video in MPEG format.
Audio	Basic	Single channel encoding of voice at 8 KHz.
Application	PostScript	Adobe PostScript.
	Octet-stream	General binary data (eight-bit bytes).

Table 16.14 Data types and subtypes in MIME

Four subtypes are defined for this type: Mixed, parallel, digest, Alternative.

- In the mixed type, the parts must be presented to the recipient in the exact order.
- Each part has a different type and is defined at the boundary.
- The parallel subtype is similar to the mixed subtype, except that the order of the parts is unimportant.
- The digest subtype is also similar to the mixed subtype except that the default type/subtype is message/RFC822.
- In the alternative subtype, the same message is repeated using different formats.

The following is example of a multipart message using a mixed subtype.

```
Content-Type: multipart/mixed; boundary = xxxxx
```

```
--xxxxx
```

```
Content-Type: text/plain;
```

```
.....
```

```
--xxxxx
```

```
.....
```

```
--xxxxx--
```

- **Message.** In the message type, the body is itself an entire mail message, a part of a mail message, or a pointer to a message. These subtypes are currently used: RFC822, partial, and external body. The subtype RFC822 is used if the body is encapsulating another message.

```
Content-Type: message/partial;
```

```
Id="123@gmail.com";
```

```
Number=1
```

```
Total=3
```

```
-----
```

```
-----
```

The subtype `external-body` indicates that the body does not contain the actual message but is only a reference to the original message. The parameters following the subtype define how to access the subtype define how to access the original message. The following is an example.

```
Content-Type: message/external-body,
```

```
Name="report.txt",
```

```
Site="finda.edu",
```

```
Access-type="ftp";
```

```
-----
```

```
-----
```

- **Image** The original message is a stationary image, indicating that there is no animation. The two currently used subtypes are Joint Photographic Experts Group (JPEG), which uses image compression, and Graphics Interchange Format (GIF).
- **Video** The original message is time-varying image (animation). The only subtype is Moving Picture Experts Group (MPEG). If the animated image contains sounds, it must be sent separately using the audio content type.
- **Audio** The original message is sound. The only subtype is basic, which uses 8KHZ standard audio data.
- **Application** The original message is a type of data not previously defined. There are two subtypes used currently. Postscript and octet-stream. Postscript is used when the data are in Adobe postscript format. Octet-stream is used when the data must be interpreted as a sequence of 8-bit bytes (binary file).

Content-Transfer- Encoding

This header defines the method used to encode the message into 0's and 1's for transport:

```
Content-Transfer-Encoding: < type>
```

The five types of encoding methods are listed in [table:16.15](#).

Type	Description
7bit	Non-ASCII characters and short lines.
8bit	Non-ASCII characters and short lines.
Binary	Non-ASCII characters with undivided-length lines.
Radix-64	6-bit blocks of data are encoded into 8-bit ASCII characters using Radix-64 conversion.
Quoted-printable	Non-ASCII characters are encoded as an equal sign followed by an ASCII code.

Table 16.15 Content-transfer-Encoding

Radix-64 conversion

Each 6-bit section is interpreted as one character according to [Table 16.16](#).

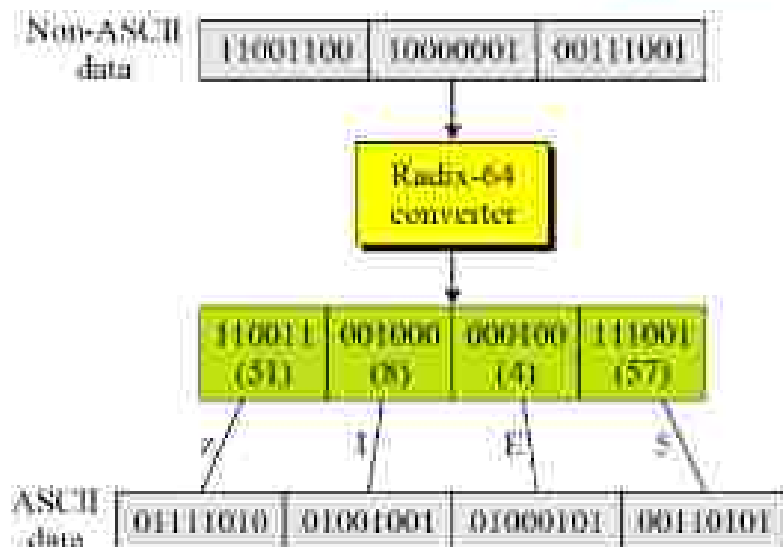


Figure 16.25 Radix-64 conversion.

Table 16.16 Radix-64 encoding table

Value	Code	Value	Code	Value	Code	Value	Code	Value	Code	Value	Code
0	A	11	L	22	W	33	k	44	x	55	9
1	B	12	M	23	X	34	l	45	y	56	0
2	C	13	N	24	Y	35	j	46	z	57	1
3	D	14	O	25	Z	36	k	47	a	58	2
4	E	15	P	26	a	37	l	48	b	59	3
5	F	16	Q	27	b	38	m	49	c	60	4
6	G	17	R	28	c	39	n	50	d	61	5
7	H	18	S	29	d	40	e	51	e	62	6
8	I	19	T	30	e	41	f	52	f	63	7
9	J	20	U	31	f	42	g	53	g		
10	K	21	V	32	g	43	h	54	h		

Table 16.16 Radix-64 encoding table

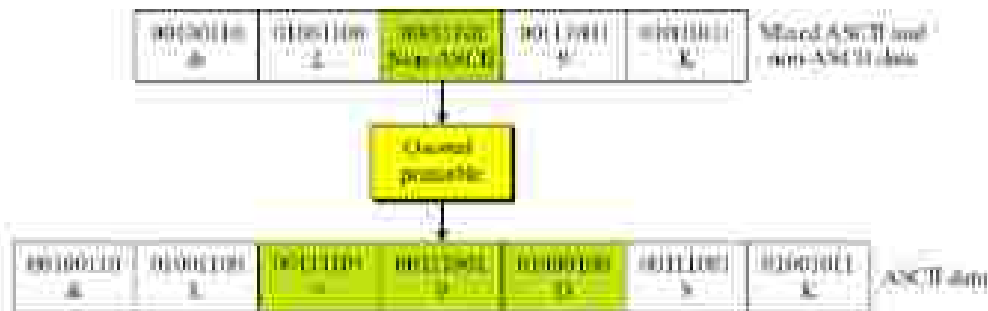


Figure 16.26 Quoted-printable

Content-Id

This header uniquely identifies the whole message in a multiple message environment.

Content-Id: id= <Content-Id>

Content-Description

This header defines whether the body is image, audio or video.

Content-Description: <description>

3.2 S/MIME

S/MIME adds some new content types to include security services to the MIME. All of these new types include the parameter "application/pkcs7-mime," in which "pkcs" defines "Public Key Cryptography Specification."

Cryptographic Message Syntax (CMS)

To define how security services, such as confidentiality or integrity, can be added to MIME content types, S/MIME has defined Cryptographic Message Syntax (CMS). The syntax in each case defines the exact encoding scheme for each content type. For details, the reader is referred to RFC 3369 and 3370.

Signed data content-type

This type provides only integrity of data. It contains any type and zero or more signature values. The encoded result is an object called signedData. Figure 16.27 shows the process of creating an object of this type. The following are the steps in the process:

1. For each signer, a message digest is created from the content using the specific hash algorithm chosen by that signer.
2. Each message digest is signed with the private key of the signer.
3. The content, signature values, certificates, and algorithms are then collected to create the signedData object.

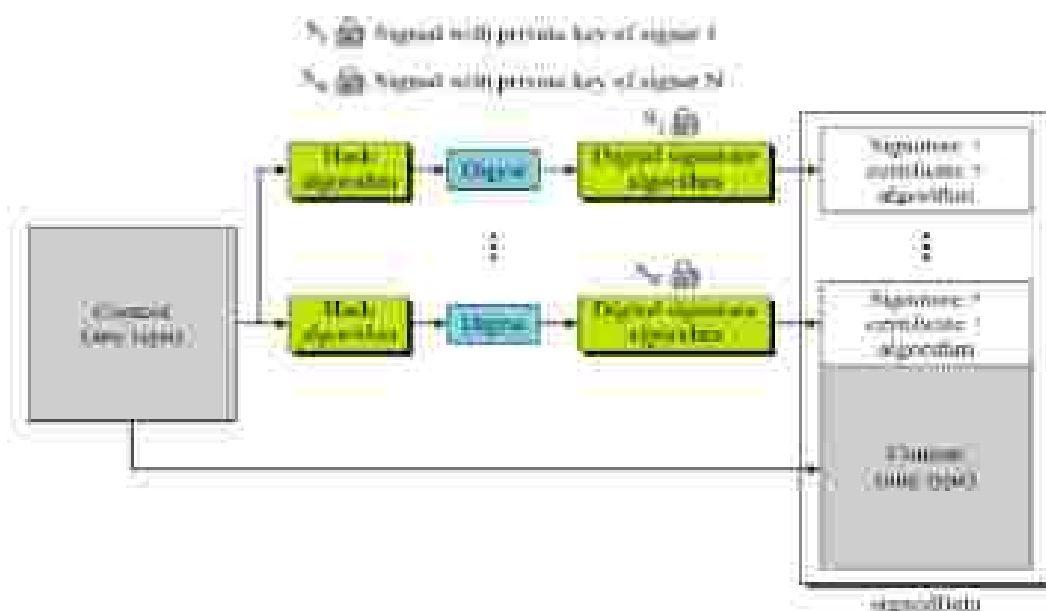


Figure 16.27 Signed-data content type

Enveloped data content-type

This type is used to provide privacy for the message. It contains any type and zero or more encrypted keys and certificates. The encoded result is an object called `envelopedData`. Figure 16.28 shows the process of creating an object of this type.

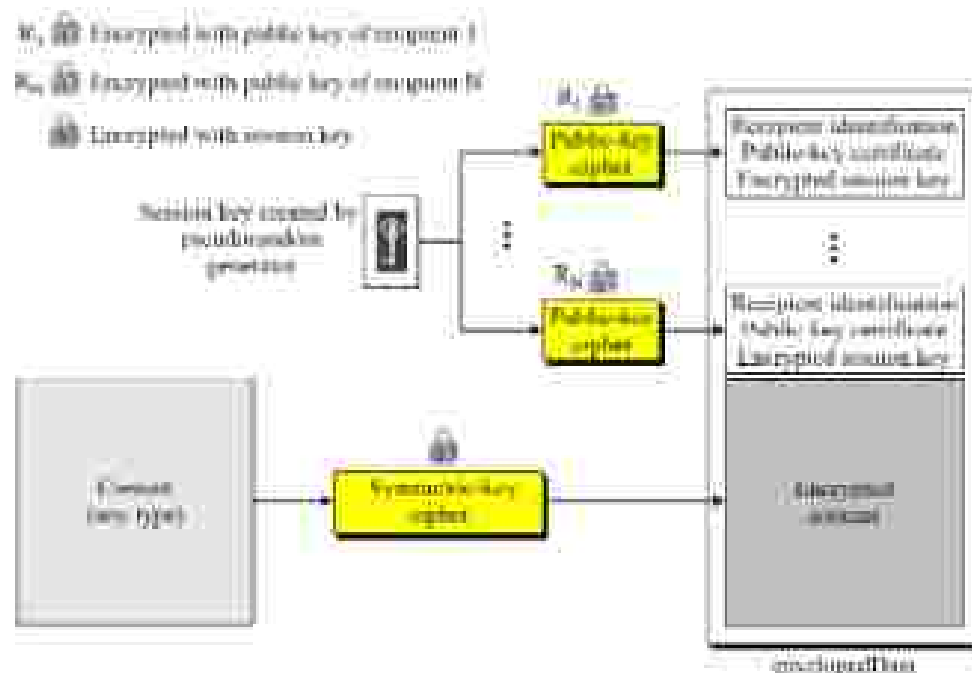


Figure 16.28 Enveloped-data content type

1. A pseudorandom session key is created for the symmetric-key algorithms to be used.
2. For each recipient, a copy of the session key is encrypted with the public key of each recipient.
3. The content is encrypted using the defined algorithm and created session key.
4. The encrypted contents, encrypted session keys, algorithm used, and certificates are encoded using Radix-64.

Digest data content-type

This type is used to provide integrity for the message. The result is normally used as the content for the enveloped-data content type. The encoded result is an object called digestedData. Figure 16.29 shows the process of creating an object of this type.

1. A message digest is calculated from the content.
2. The message digest, the algorithm, and the content are added together to create the digestedData object.

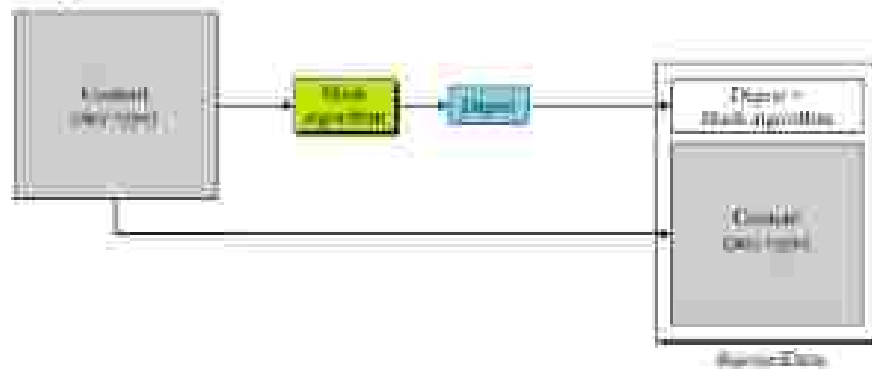


Figure 16.29 Digest-data content type

Encrypted-Data content Type

This type is used to create an encrypted version of any content type. Although this looks like the enveloped-data content type, the encrypted-data content type has no recipient. It can be used to store the encrypted data instead of transmitting it. The process is very simple, the user employs any key and any algorithm to encrypt the content. The encrypted content is stored without including the key or the algorithm. The object created is called encryptedData.

Authenticated data content type

H_1 Encrypted with public key of recipient 1

H_n Encrypted with public key of recipient N

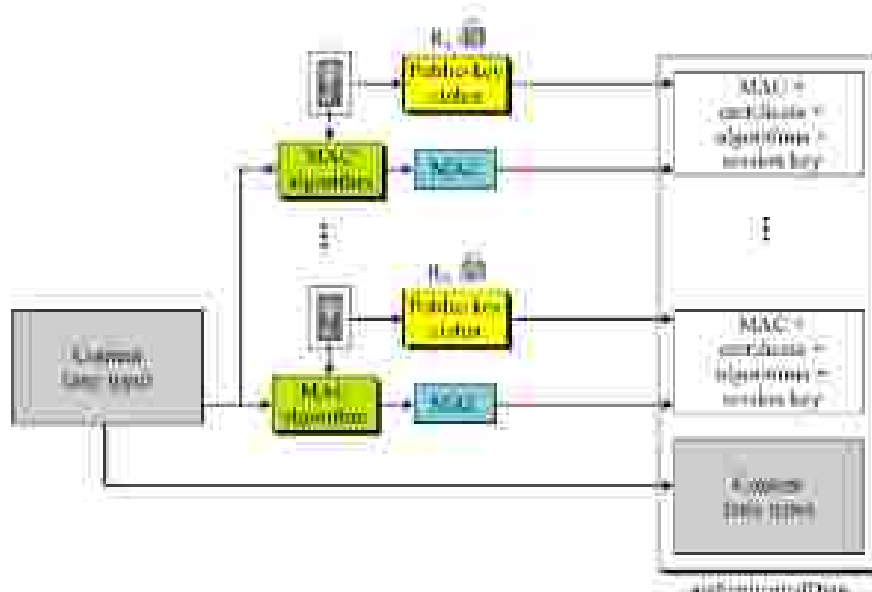


Figure 16.30 Authenticated-data content type

This type is used to provide authentication of the data. The object is called `authenticatedData`. Figure 16.30 shows the process.

1. Using a pseudorandom generator, a MAC key is generated for each recipient.
2. The MAC key is encrypted with the public key of the recipient.
3. A MAC is created for the content.
4. The content, MAC, algorithms, and other information are collected together to form the `authenticatedData` object.

Cryptographic Algorithms

S/MIME defines several cryptographic algorithms as shown in table 16.17. The term “must” means an absolute requirement; the term “should” means recommendation.

Algorithm	Sender must support	Receiver must support	Sender should support	Receiver should support
Content-encryption algorithm	Triple DES	Triple DES		1. AES 2. RC2/40
Session-key encryption algorithm	RSA	RSA	Diffie-Hellman	Diffie-Hellman
Hash algorithm	SHA-1	SHA-1		MD5
Digest-encryption algorithm	DSS	DSS	RSA	RSA
Message-authentication algorithm		HMAC with SHA-1		

Table 16.17 Cryptographic algorithm for S/MIME

The following shows an example of an enveloped-data in which a mail message is encrypted using triple DES:

```
Content-Type: application/pkcs7-mime; mime-type=enveloped-data
Content-Transfer-Encoding: Radix-64
Content-Description: attachment
name="report.txt":
cb32u67i4bbjHLE21o87eryb0287bmwkkgfDoY8hc659GbhGdH6543mhJdxaH123YjBamN
ybunLzjhgdyl6Cez3KjK34XvD678E46ce00xy76jHoyTMDcbamkpaTJdayu678543mOn3h
G34auT2P24S4Huo87e2ry60H2MjN6KuyrvgfDoy8970k923jHk130HXnd26gh78EwlyT23y
```

Applications of S/MIME

It is predicted that S/MIME will become the industry choice to provide security for commercial e-mail.

1. Security service at transport layer

- Transport layer security provides end-to-end security services for applications that use a reliable transport layer protocol such as TCP.
- Two protocols are dominant today for providing security at the transport layer: the Secure Socket Layer (SSL) and the Transport Layer Security (TLS) protocol.
- We first discuss SSL, then TLS, and then compare and contrast the two. [Figure 17.1](#) shows the position of SSL and TLS in the Internet model.

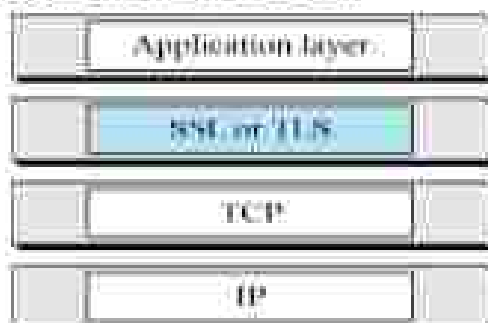


Figure 17.1 Location of SSL and TLS in the Internet model

2. SSL Architecture

SSL is designed to provide security and compression services to data generated from the application layer.

2.1 Services

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

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

2.2 Key Exchange Algorithms

The client and the server each need six cryptographic secrets (four keys and two initialization vectors). SSL defines six key-exchange methods to establish this pre-master secret: NULL, RSA, anonymous Diffie-Hellman, ephemeral Diffie-Hellman, fixed Diffie-Hellman, and Fortezza, as shown in [Figure 17.2](#).



Figure 17.2 Key-exchange methods

NULL

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

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

RSA

In this method, the pre-master secret is a 48-byte random number created by the client encrypted with the server's RSA public key, and sent to the server. The server needs to send its RSA encryption/decryption certificate. Figure 17.3 shows the idea.



Figure 17.3 RSA key exchange: server public key

Anonymous Diffie-Hellman

- This is the simplest and most secure method.
- The pre-master secret is established between the client and the server using the Diffie-Hellman (DH) protocol.
- The Diffie-Hellman half-keys are sent in plaintext, it is called anonymous Diffie-Hellman because neither party is known to the other. Figure 17.4 shows the idea.



Figure 17.4 Anonymous Diffie-Hellman key exchange

Ephemeral Diffie-Hellman

- To identify the man-in-the middle attack, the ephemeral Diffie-Hellman key exchange can be used.

- Each party sends a Diffie-Hellman key signed by its private key.
- The receiving party needs to verify the signature using the public key of the sender. Figure 17.5 shows the idea.
- The public key for verification is exchanged using either RSA or DSS digital signature certificates.

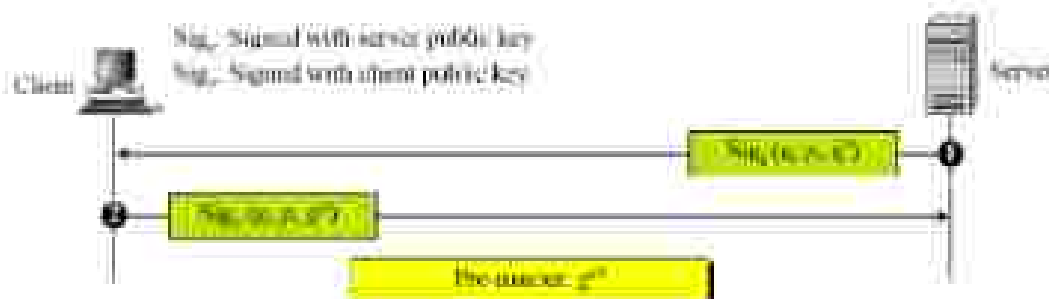


Figure 17.5 Ephemeral Diffie-Hellman key exchange

Fixed Diffie-Hellman

- Another solution is the fixed Diffie-Hellman method.
- All entities in a group can prepare fixed Diffie-Hellman parameters (g and p).
- Then each entity can create a fixed Diffie-Hellman half-key (g^a).
- For additional security, each individual half-key is inserted into a certificate verified by a certification authority (CA).
- The two parties do not directly exchange the half-keys; the CA sends the half-keys in an RSA or DSS special certificate.
- When the client needs to calculate the pre-master, it uses its own fixed half-key and the server half-key received in a certificate.
- The server does the same, but in the reverse order.

Fortezza

- It is a registered trademark of the U.S. National Security Agency (NSA).
- It is a family of security protocols developed for the Defense Department.

2.3 Encryption/Decryption Algorithm

There are several choices for encryption/decryption algorithm. We can divide the algorithms into 6 groups as shown in Figure 17.6.

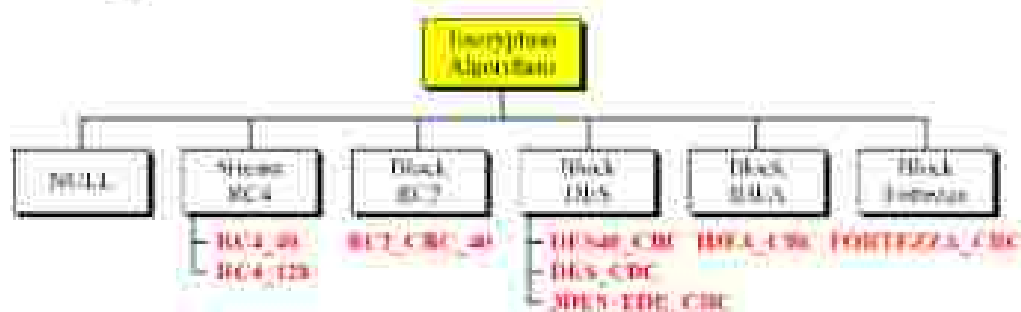


Figure 17.6 Encryption/decryption algorithms

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

Stream RC Two RC algorithms are defined in stream mode.

Block RC One RC algorithm is defined in block mode.

DES All DES algorithms are defined in block mode.

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

Fortezza The one Fortezza algorithm defined in block mode is FORTEZZA_CBC.

2.4 Hash Algorithms

SSL uses hash algorithms to provide message integrity. Three hash functions are defined as shown in figure 17.7.

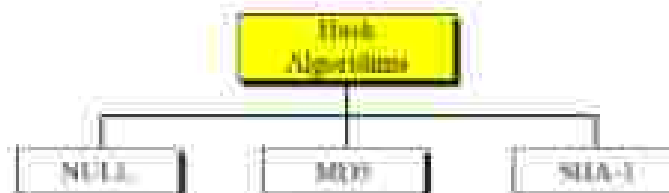


Figure 17.7 Hash algorithms for message integrity.

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

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

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

2.5 Cipher Suite

The combination of key exchange, hash, and encryption algorithms defines a cipher suite for each SSL session. Table 17.1 shows the suites used in the United States.

SSL_DHE_RSA_WITH_DES_CBC_SHA

Cipher Suite	Key Exchange	Encryption	MAC
SSL_NULL_WITH_NULL_NULL	NULL	NULL	NULL
SSL_RSA_WITH_NULL_MD5	RSA	NULL	MD5
SSL_RSA_WITH_NULL_SHA	RSA	NULL	SHA-1
SSL_RSA_WITH_RC4_128_MD5	RSA	RC4	MD5
SSL_RSA_WITH_RC4_128_SHA	RSA	RC4	SHA-1
SSL_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA	SHA-1
SSL_RSA_WITH_DES_CBC_SHA	RSA	DES	SHA-1
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES	SHA-1
SSL_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4	MD5
SSL_DH_anon_WITH_DES_CBC_SHA	DH_anon	DES	SHA-1
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES	SHA-1
SSL_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES	SHA-1
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES	SHA-1
SSL_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES	SHA-1
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES	SHA-1
SSL_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES	SHA-1
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES	SHA-1
SSL_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES	SHA-1
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES	SHA-1
SSL_FORTezza_DSS_WITH_NULL_SHA	Fortezza	NULL	SHA-1
SSL_FORTezza_DSS_WITH_FORTezza_CBC_SHA	Fortezza	Fortezza	SHA-1
SSL_FORTezza_DSS_WITH_RC4_128_SHA	Fortezza	RC4	SHA-1

Table 17.1 SSL cipher suite list

2.6 Compression Algorithms

Compression is optional in SSLv3. No specific compression algorithm is defined for SSLv3. Therefore, the default compression method is NULL.

2.7 Cryptographic Parameter Generation

- To achieve message integrity and confidentiality, SSL needs six cryptographic secrets, four keys and two IVs.
- The client needs one key for message authentication (HMAC), one key for encryption, and one IV for block encryption.
- The server needs the same SSL requires that the keys for one direction be different from those for the other directions.
- If there is an attack in one direction, the other direction is not affected.
- The parameters are generated using the following procedure:
 1. The client and server exchange two random numbers; one is created by the client and the other by the server.
 2. The client and server exchange one pre-master secret using one of the key-exchange algorithms we discussed before.
 3. A 48-byte master secret is created from the pre-master secret by applying two hash functions (SHA-1 and MD5) as shown in [figure 17.8](#).
 4. The master secret is used to create variable-length key material by applying some set of hash functions and padding with different constants as shown in [figure 17.9](#). The process is repeated until key material of adequate size is created.
 5. Six different keys are extracted from the key material, as shown in [figure 17.10](#).

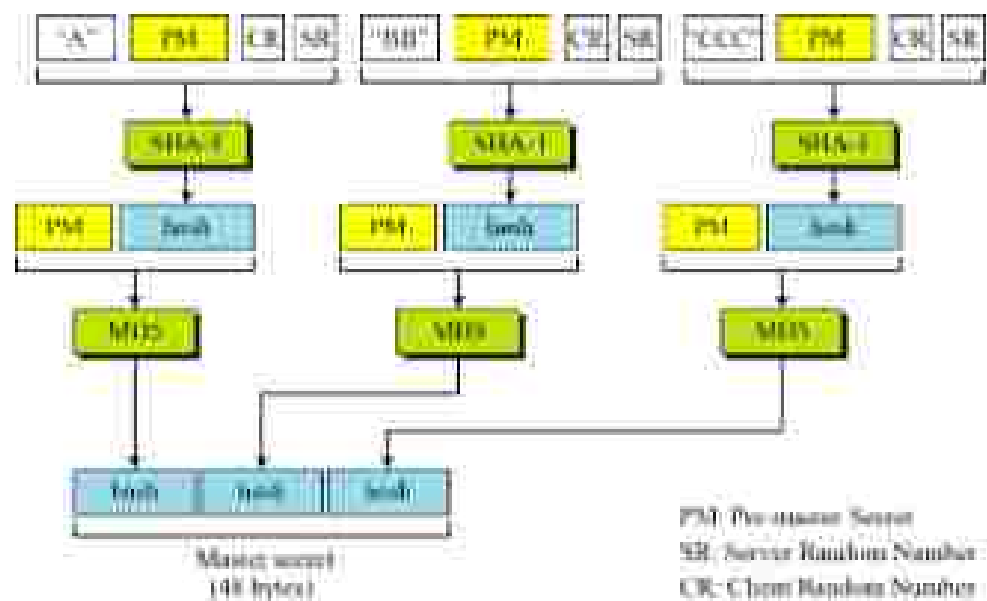


Figure 17.8 Calculation of master secret from pre-master secret

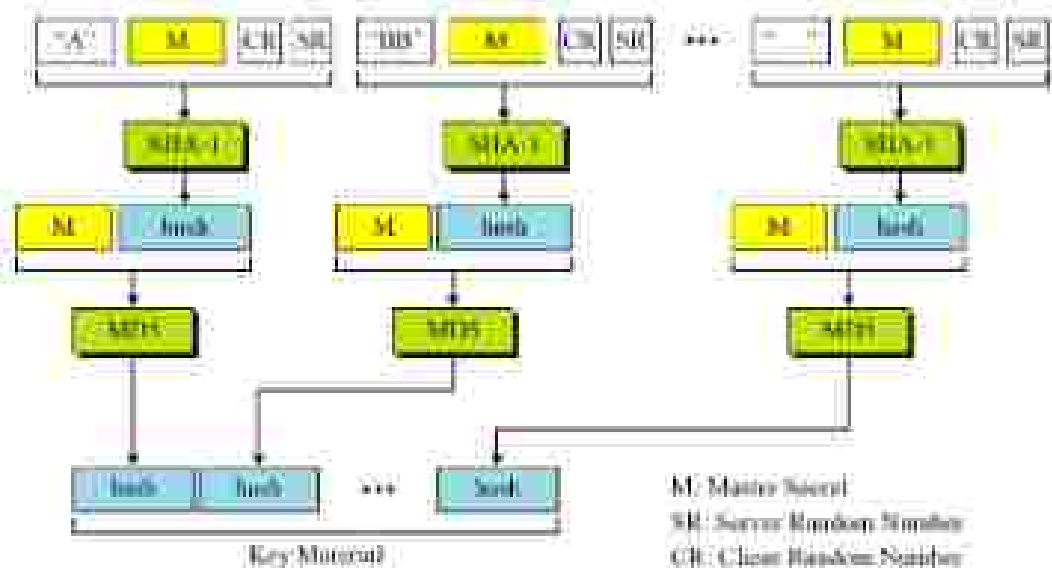


Figure 17.9 Calculation of key material from master secret

Auth. Key: Authentication Key

Enc. Key: Encryption Key

IV: Initialization Vector

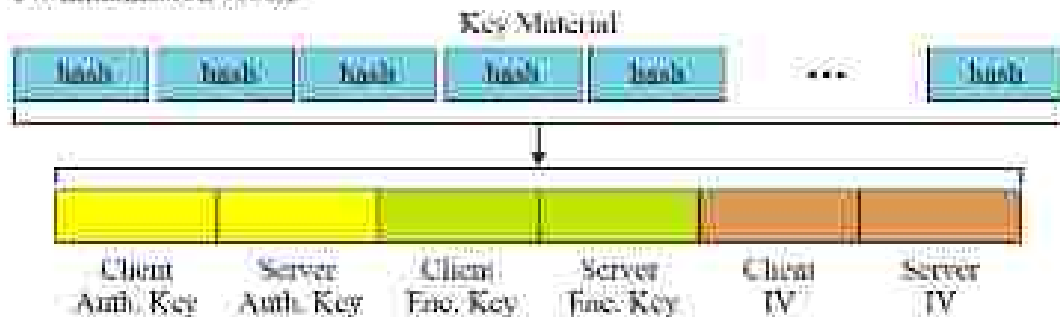


Figure 17.10 Extractions of cryptographic Secrets from key material

2.8 Sessions and Connections

- SSL differentiates a connection from a session.
- Let us elaborate on these two items here. A session is an association between a client and a server.
- After a session is established, the two parties have common information such as the session identifier, the certificate authenticating each of them, the compression method, the cipher suite, and a master secret that is used to create keys for message authentication encryption.
- For two entities to exchange data the establishment of a session is necessary, but not sufficient, they need to create a connection between themselves.
- A connection between two parties can be terminated and reestablished within the same session.
- To create a new session, the two parties need to go through a negotiate process.
- The separation of a session from a connection prevents the high cost of creating a master secret. Figure 17.11 shows the idea of a session and connections inside that session.

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

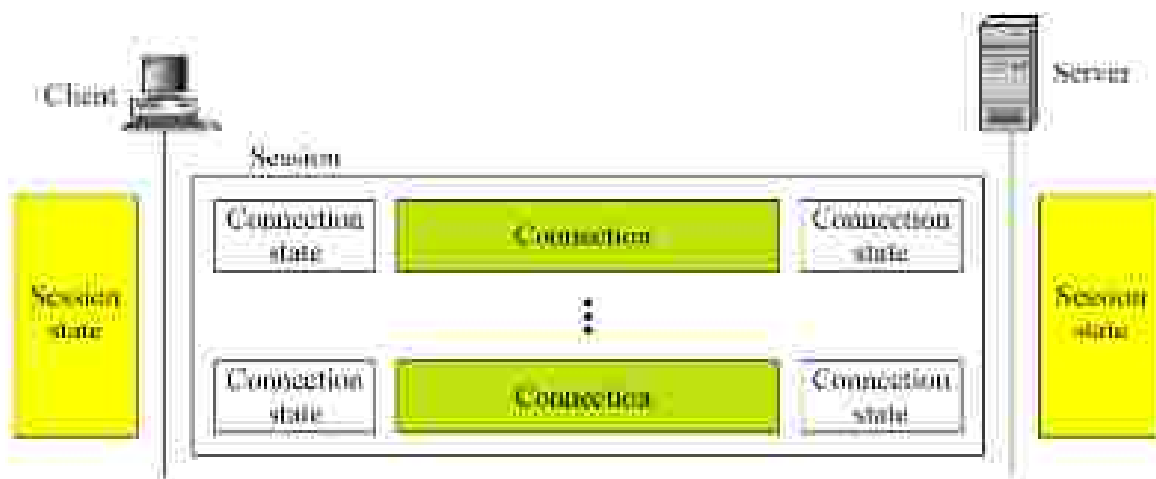


Figure 17.11 A session and connections

Session state

A session is defined by session state, a set of parameters established between the server and the client. Table 17.2 shows the list of parameters for a session state.

Parameter	Description
Session ID	A server-chosen 96-bit number defining a session.
Peer Certificate	A certificate of type X.509.v3. This parameter may be empty (null).
Compression Method	The compression method.
Cipher Suite	The agreed-upon cipher suite.
Master Secret	The 48-byte secret.
Is resumable	A yes/no flag that allows new connections to an old session.

Table 17.2 shows the list of parameters for a session state.

Connection state

- A connection is defined by a connection state, a set of parameters established between two peers. Table 17.3 shows the list of parameters for a session state.
- SSL uses two attributes to distinguish cryptographic secrets: write and read.
- The term write specifies the key used for signing or encrypting outbound messages.
- The term read specifies the key used for verifying or decrypting inbound messages.

The client and the server have six different cryptography secrets: three read secrets and three write secrets.

The read secrets for the client are the same as the write secrets for the server and vice versa.

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

Figure 17.3 connection state parameters.

3. Four protocols

We have discussed the idea of SSL without showing how SSL accomplishes its tasks. SSL defines four protocols in two layers, as shown in Figure 17.12.

1. Handshake Protocol
2. ChangeCipherSpec Protocol
3. Alert Protocol
4. Record Protocol

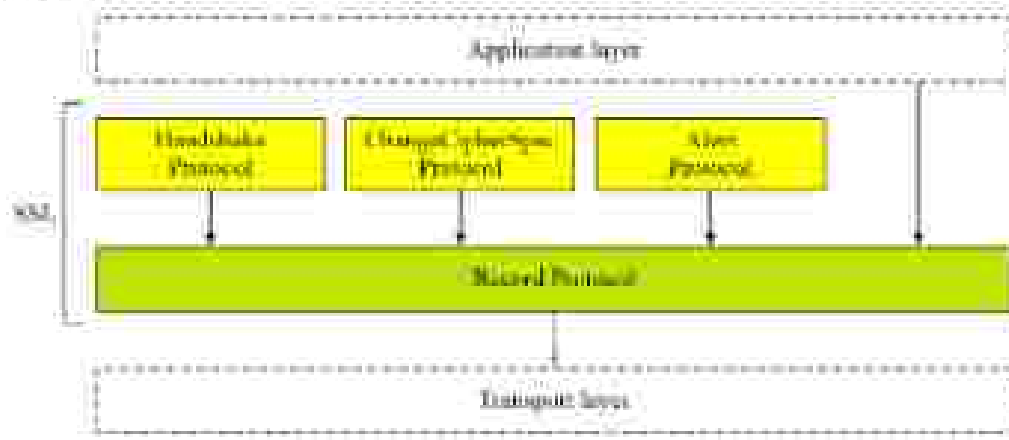


Figure 17.12 Four SSL protocols

3.1. Handshake protocol

- The handshake protocol uses messages to negotiate the cipher suite, to authenticate the server to the client and the client to the server if needed, and to exchange information for building the cryptographic secrets.
- The handshaking is done in four phases as shown in figure 17.13.



Figure 17.13 Handshake protocol

Phase I: Establishing Security Capability

- In phase I, the client and the server announce their security capabilities and choose those that are convenient for both.
- In this phase, a session ID is established and the cipher suite is chosen.
- The parties agree upon a particular compression method.

- Finally two random numbers are selected, one by the client and one by the server to be used for creating a master secret as we saw. **Figure 17.14** gives additional details about Phase I.

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

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

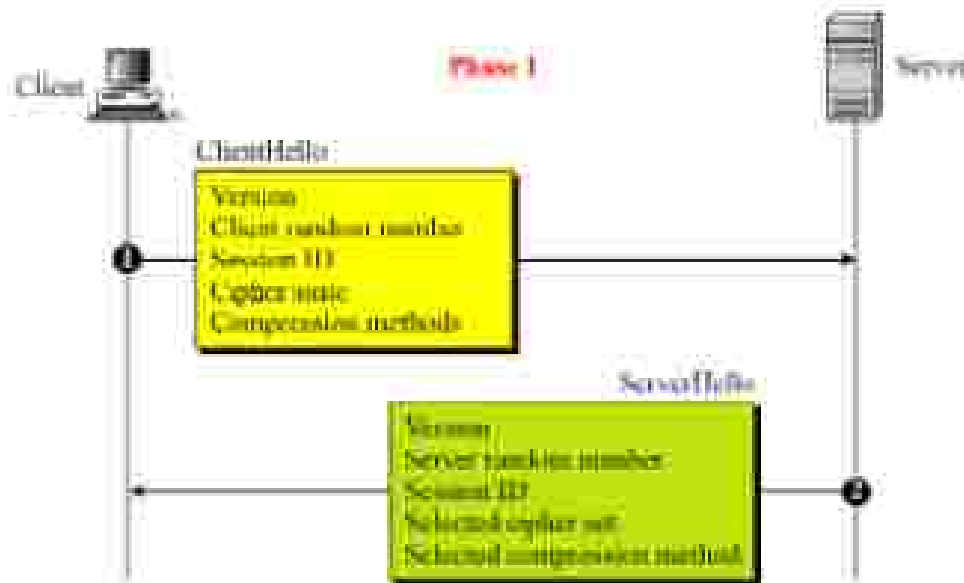


Figure 17.14 Phase I of Handshake Protocol

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

- An SSL version number.
- A 32-byte random number that will be used for master secret generation.
- A session ID that defines the session.
- The selected cipher suite from the client list.
- The selected compression methods from the client list.

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

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

Phase II. Server Key Exchange and Authentication

- In phase II, the server authenticates itself if needed.

- The sender may send its certificate, its public key, and may also request certificates from the client.
- At the end, the server announces that the ServerHello process is done. Figure 17.15 gives additional details about Phase II.



Figure 17.15 Phase II of Handshake Protocol

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

ServerKeyExchange After the certificate message, the server sends a ServerKeyExchange message that includes its contribution to the pre-master secret. The message is not required if the key exchange is RSA or fixed Diffie-Hellman.

CertificateRequest The server may require the client to authenticate itself.

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

After Phase II,

- The server is authenticated to the client.
- The client knows the public key of the server if required.

Fig. four cases in phase II

1. RSA
2. Anonymous DH
3. Ephemeral DH
4. Fixed DH

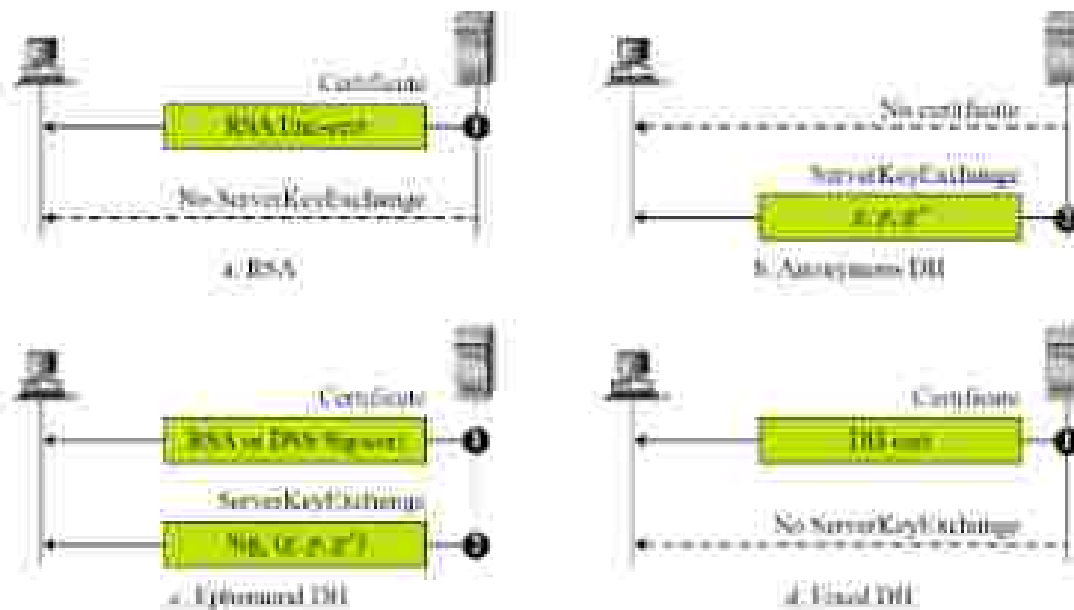


Figure 17.16 Four cases in phase II

Phase III: Client Key Exchange and Authentication

Phase III is designed to authenticate the client. Up to three messages can be sent from the client to the server, as shown in figure 17.17.

Phase III of Handshake Protocol:

Certificate

ClientKeyExchange

CertificateVerify

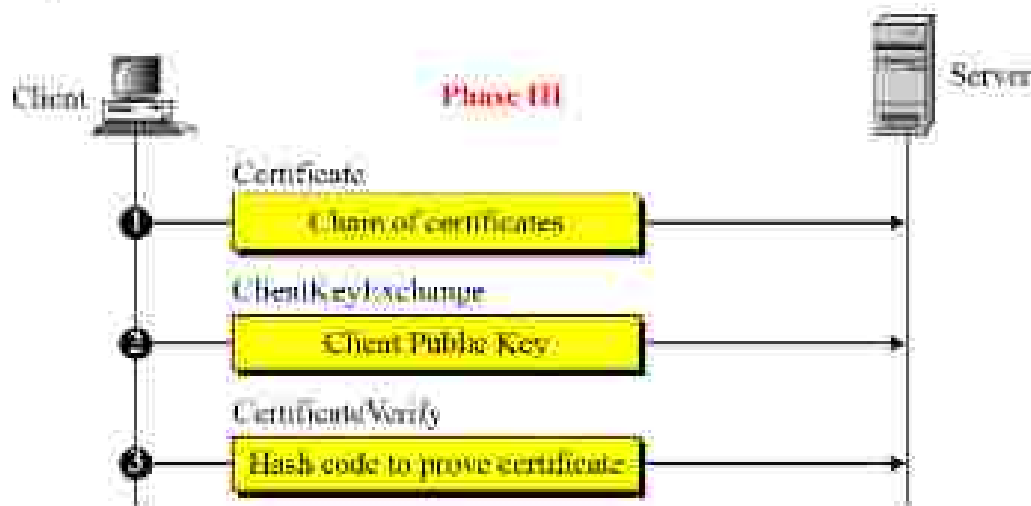


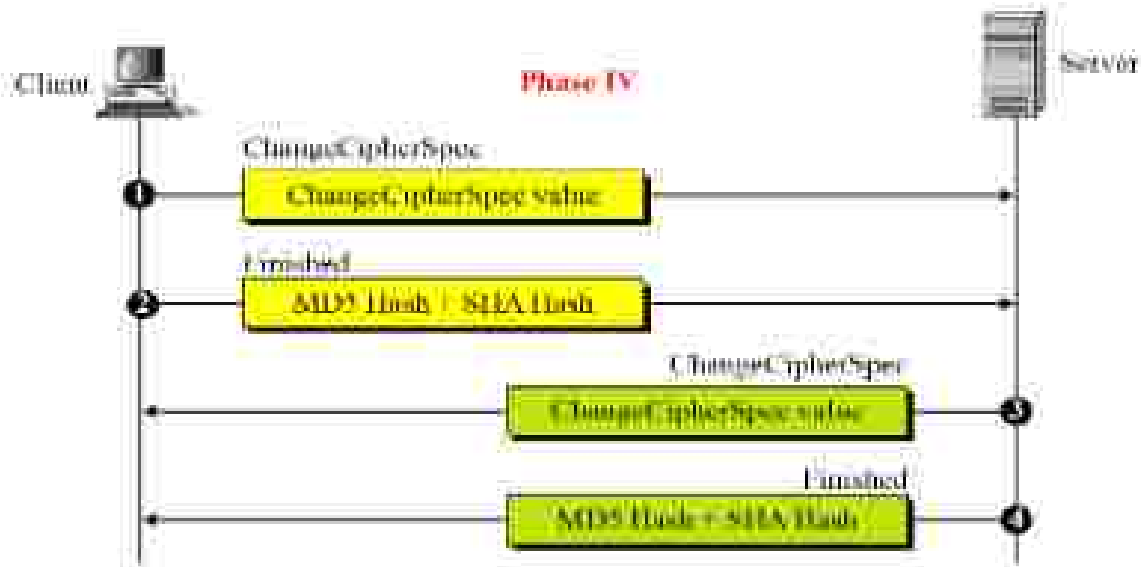
Figure 17.17 Phase III of Handshake Protocol

After Phase III,

- The client is authenticated for the server.
- Both the client and the server know the pre-master secret.

Phase IV: Finalizing and Finishing

In phase IV, the client and server send message to change cipher specification and to finish the handshaking protocol. Four message are exchanged in this phase, as shown in [Figure 17.19](#).



[Figure 17.19](#) Phase IV of handshake protocol

ChangeCipherSpec The client sends a **ChangeCipherSpec** message to show that it has moved all of the cipher suite set and the parameters from the pending state to the active state. This message is actually part of the **ChangeCipherSpec Protocol** that will discuss later.

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

ChangeCipherSpec the server sends a **ChangeCipherSpec** message to show that it has also moved all of the cipher suite set and parameters from the pending state to the active state. This message is part of the **ChangeCipherSpec Protocol**, which will be discussed later.

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

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

3.2 ChangeCipherSpec protocol

- The **changecipherspec** protocol defines the process of moving values between the pending and active states.
- First, the client sends a **ChangeCipherSpec** message.
- After, the client sends the message, it moves the write parameters from pending to active.
- The client can now use these parameters to sign or encrypt outbound messages.
- After the receiver receives this message, it moves the read (inbound) parameters from the pending to the active state.
- Now the server can verify and decrypt the messages.

3.3 Alert protocol

- SSL uses the Alert Protocol for reporting errors and abnormal conditions. It has only one message type, the alert message that describes the problem and its level. **Table 17.4** shows the types of Alert messages defined for SSL.

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

Table 17.4 Alerts defined for SSL

3.4 Record Protocol

- The record protocol carries messages from the upper layers.
- The message is fragmented and optionally compressed; a MAC is added to the compressed message using the negotiated hash algorithm.
- The compressed fragment and the MAC using the negotiated encryption algorithm.
- Finally the SSL header encrypted message. **Figure 17.21** shows this process at the sender.

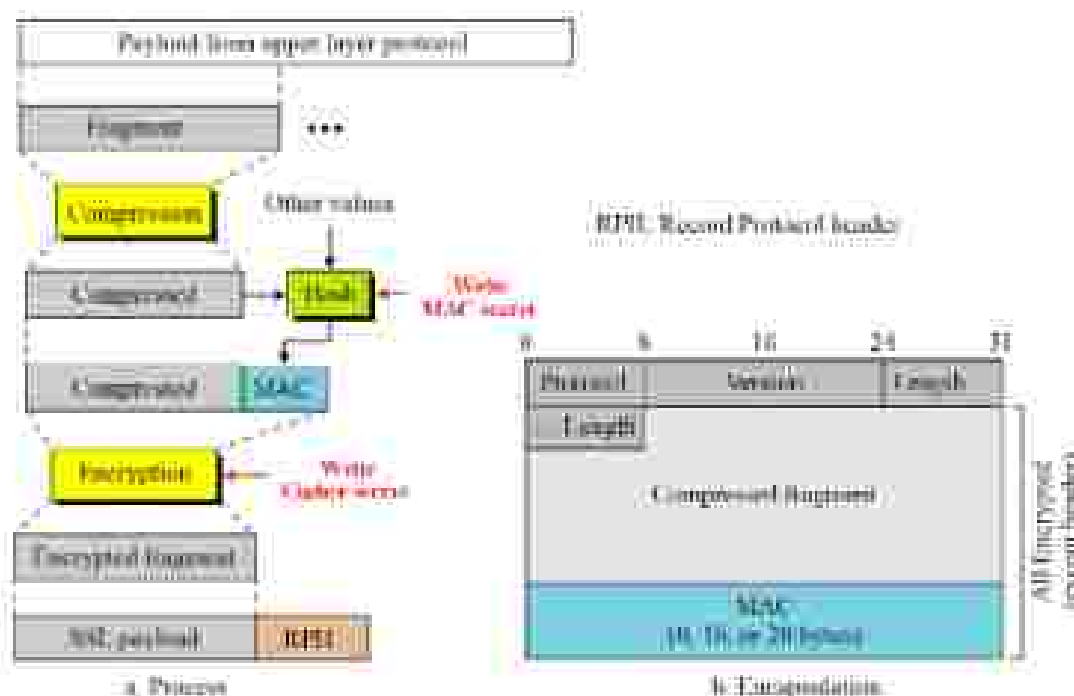


Figure 17.21 Processing done by the Record Protocol

Fragmentation/Combination

- At the sender, a message from the application layer is fragmented bytes, with the last block possibly less than the size.
- At the receiver combined together to make a replica of the original message.

Compression/Decompression

- At the sender site, application layer fragments are compressed by the compression method negotiated during the handshaking.
- The compression method needs to be lossless.
- The size of the fragment must not exceed 1024 bytes.
- At the receiver, the compressed fragment is decompressed to create a replica of the original.
- If the decomposed fragment exceeds 2^{14} , a fatal decompression Alert message is issued.
- Note that compression/Decompression is optional in SSL.

Signing / Verifying

- At the sender, the authentication method defined during the handshake creates a signature (MAC) as shown in [figure 17.22](#).

Pad-1: Byte 0x36 (00110110) repeated 48 times for MD5 and 40 times for SHA-1

Pad-2: Byte 0x5C (01011100) repeated 48 times for MD5 and 40 times for SHA-1

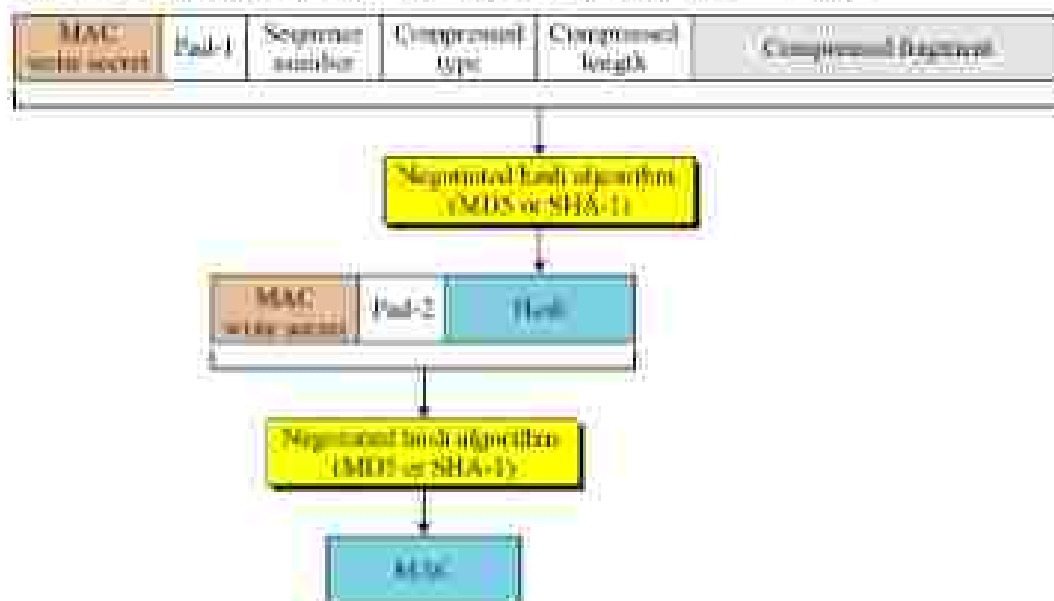


Figure 17.22 Calculation of MAC

The hash algorithm is applied twice. First, a hash is created from the concatenations of the following values:

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

- e. The compressed length, which is the length of the compressed format.
- f. The compressed fragment itself.

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

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

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

Encryption/Decryption

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

Framing/Deframing

After the encryption, the Record Protocol header is added at the sender. The header is removed at the receiver before decryption.

4. SSL Message formats

As we have discussed, messages from three protocols and data from the application layer are encapsulated in the Record Protocol messages. The record protocol has a general header that is added to each message coming from the sources, as shown in [figure 17.23](#).



Figure 17.23 Record Protocol general header

- **Protocol** This 1-byte field defines the source or destination of the message. It is used for multiplexing and demultiplexing.
- **Version** This 2-byte field defines the version of the SSL: one byte is the major version and the other is the minor. The current versions of SSL is 3.0.
- **Length** This 2-byte field defines the size of the message in bytes.

4.1 ChangeCipherSpec Protocol

- The ChangeCipherSpec protocol has one message, the ChangeCipherSpec message.
- The message is only one byte, encapsulated in the Record Protocol message with protocol value 20, as shown in [figure 17.24](#).

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



Figure 17.24 ChangeCipherSpec message

4.2 Alert Protocol

- The alert protocol has one message that reports errors in the process.
- The below figure shows the encapsulation of this single message in the record protocol with protocol value 21.

The two fields of the Alert protocol are listed below.

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



Figure 17.25 Alert protocol

4.3 Handshake Protocol

Several messages have been defined for the Handshake protocol.

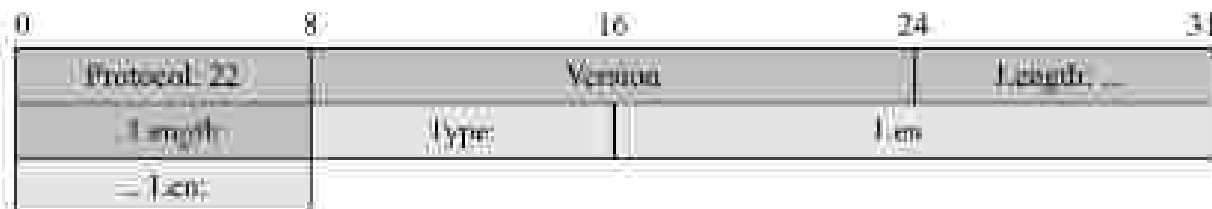


Figure 17.26 Generic header for Handshake Protocol

Type This one-byte field defines the type of message. So far ten types have been defined as listed in [table 17.5](#).

Type	Message
0	HelloRequest
1	ChangeCipher
2	ServerHello
11	ClientHello
12	ServerKeyExchange
13	ClientKeyExchange
14	ServerHelloDone
15	ClientHelloVerify
16	ClientKeyExchange
20	Finished

Table 17.5 Types of Handshake messages

Length (Len) This three-byte field defines the length of the message (excluding the length of the type and length field).

HelloRequest Message

The HelloRequest message, which is rarely used, is a request from the server to the client to restart a session. This may be needed if the server feels that something is wrong with the session and a fresh session is needed. Figure 17.27 shows the format of this message. It is four bytes with a type value of 0. The message has no body, so the value of the length field is also 0.

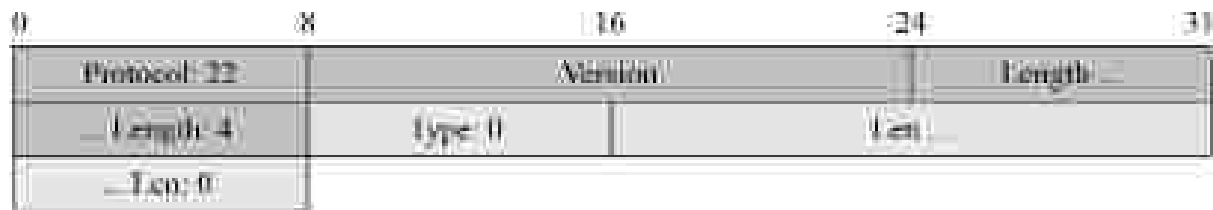


Figure 17.27 HelloRequest message

ClientHello Message

The ClientHello message is the first message exchanged during handshaking. Figure 17.28 shows the format of the message.

The type and length fields are as discussed previously. The following is a brief description of the other fields.

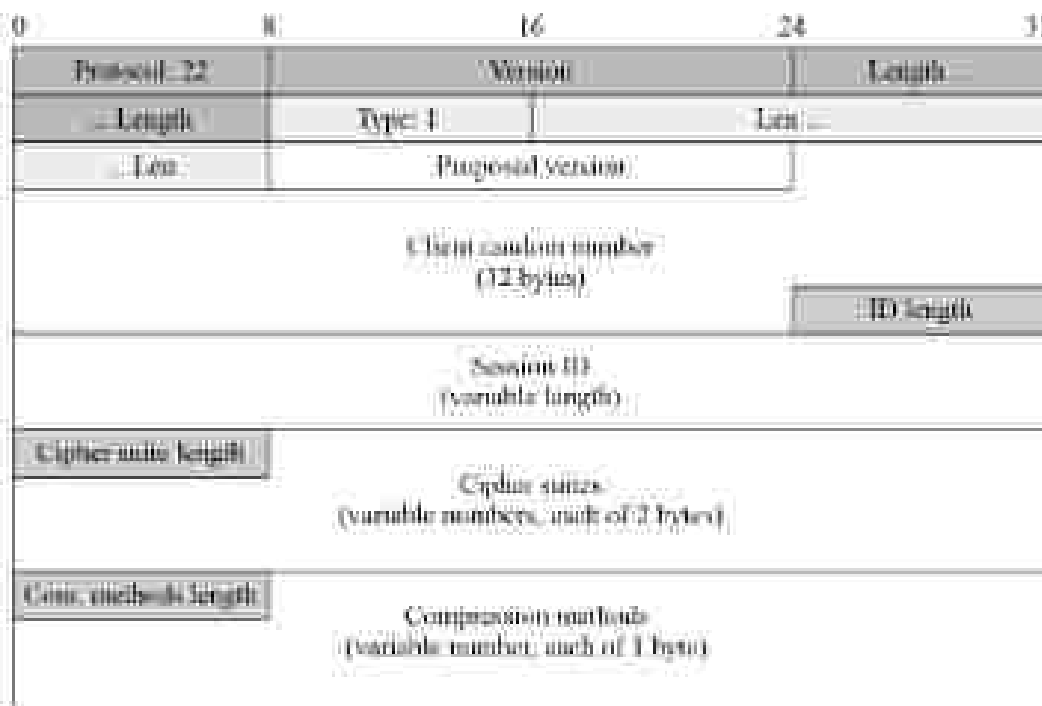


Figure 17.28 ClientHello message

Version This 2-byte field shows the version of the SSL used. The version 3.0 for SSL and 3.1 for TLS. Note that the version value, for example, 3.0, is stored in two bytes: 3 in the first byte and 0 in the second.

Client Random Number this 32-byte field is used by the client to send the client random number, which creates security parameters.

Session ID Length This 1-byte field defines the length of the session ID (next field). If there is no session ID, the value of this field is 0.

Session ID The value of this variable-length field is 0 when the client starts a new session. The session ID is initiated by the server. However, if a client wants to resume a previously stopped session, it can include the previously-defined session ID in this field. The protocol defines a maximum of 32 bytes for the session ID.

Cipher Suite Length This 2-byte field defines the length of the client-proposed cipher suite list (next field).

Compressed Method List This variable-length field gives the list of compression methods that the client supports. The field lists the methods from the most preferred to the least preferred. Each method is encoded as a one-byte number. So far, the only method is the NULL method (no compression). In this case, the value of the compression method length is 1 and the compression method list has only one element with the value of 0.

ServerHello Message

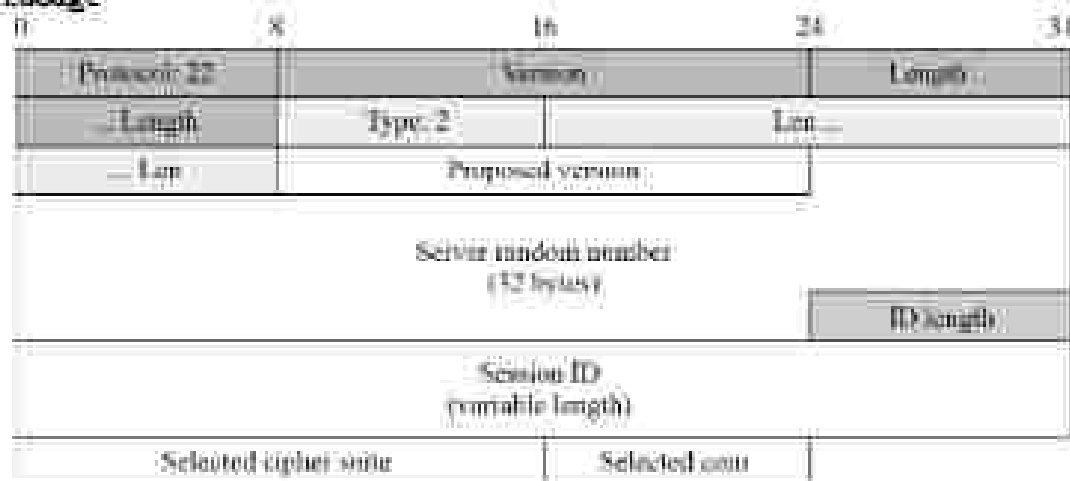


Figure 17.29 ServerHello message

The ServerHello message is the server response to the ClientHello message. The format is similar to the ClientHello message, but with fewer fields. **Figure 17.29** shows the format of the message.

The version field is the same. The server random number field defines a value selected by the server. The session ID length and the session ID field are the same as those in the ClientHello message. However, the session ID is usually blank (and the length is usually set to 0) unless the server is resuming an old session. In other words, if the server allows a session to resume, it inserts a value in the session ID field to be used by the client if the client wishes to reopen an old session.

The selected cipher suit field defines the single cipher suite selected by the server from the list sent by the client. The compression method field defines the method selected by the server from the list sent by the client.

Certificate Message

The certificate message can be sent by the client or the server to list the chain of public-key certificates. **Figure 17.30** shows the format.

The value of the type field is 11. The body of the message includes the following fields:

Certificate Chain Length This variable-length field lists the chain. This field is redundant because its value is always 3 less than the value of the length field.

Certificate Chain This variable-length field lists the chain of public-key certificates that the client or the server carries. For each certificate, there are two sub-fields:

- A three-byte length field
- The variable-size certificate itself

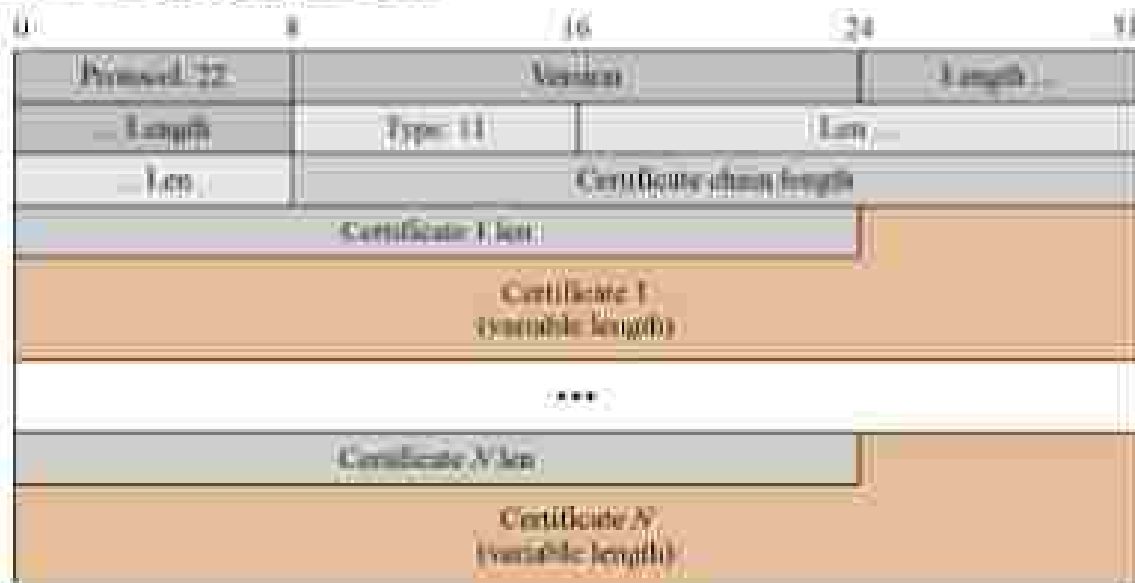


Figure 17.30 Certificate message

ServerKeyExchange Message

- The ServerKeyExchange Message is sent from the server to the client. Figure 17.31 shows the general format.
- The message contains the keys generated by the server.
- The format of the message is dependent on the cipher suite selected in the previous message.
- The client that receives the message needs to interpret the message according to the previous information.
- If the server has sent a certificate message, then the message also contains a signed parameter.



Figure 17.31 ServerKeyExchange message

CertificateRequest Message

- The certificateRequest message is sent from the server to the client.

- The message asks the client to authenticate itself to the server using one of the acceptable certificates and one of the certificate authorities named in the message. **Figure 17.32** shows the format.

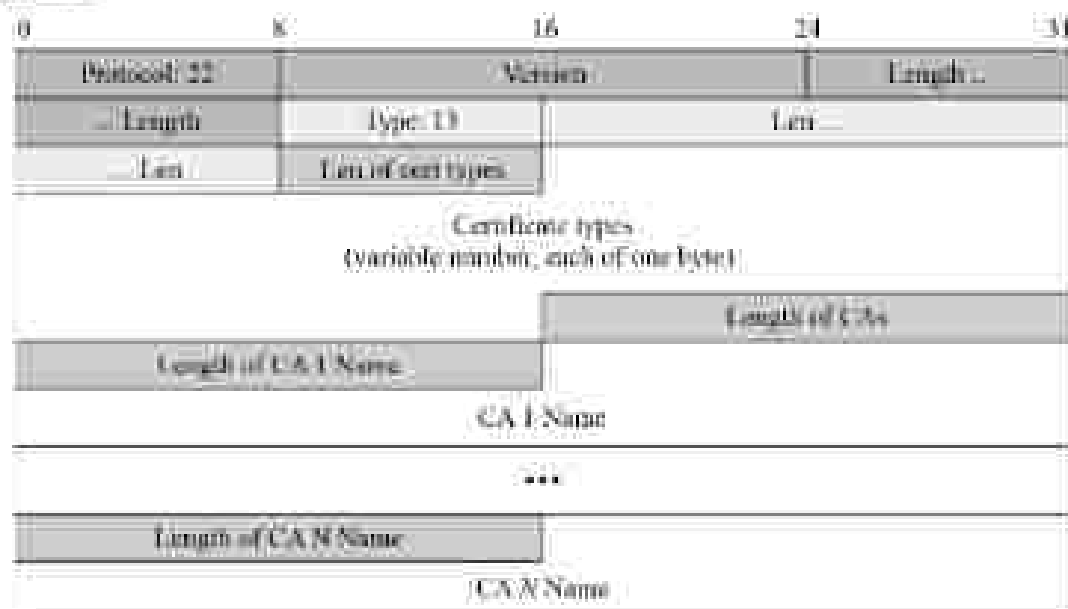


Figure 17.32 CertificateRequest message

The value of the type field is 13. The body of the message includes the following fields:

Len of Cert Types This one-byte field shows the length of the certificate types.

Certificate Types This variable-length field gives the list of the public-key certificates types that the server accepts. Each type is one byte.

Length of CAs This two-byte field gives the length of the certificate authorities.

Length of CA x Name This two-byte field defines the length of the xth certificate authority name. The value of x can be between 1 to N.

CA x Name This variable-length field defines the name of the xth certificate authority. The value of x can be between 1 to N.

ServerHelloDone Message

The ServerHelloDone message is the last message sent in the second phase of handshaking. The message signals that phase II does not carry any extra information. **Figure 17.33** shows the format.



Figure 17.33 ServerHelloDone Message

CertificateVerify Message

The CertificateVerify message is the last message of Phase III. In this message, the client proves that it actually owns the private key related to its public-key certificate to do so, the client creates a hash of all handshake messages sent before this message, and signs them using the MD5 or SHA-1 algorithm based on the certificate type of the client. **Figure 17.34** shows the format.

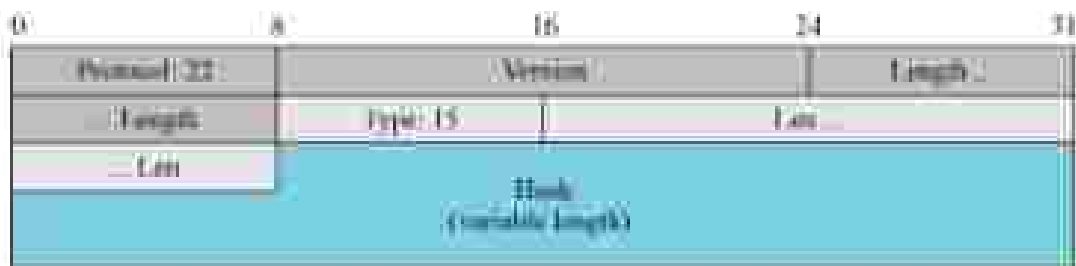


Figure 17.34 CertificateVerify message

If the client private key is related to a DSS certificate, then the hash is based only on the SHA-1 algorithm and the length of the hash is 20 bytes. If the client private key is related to an RSA certificate, then there are two hashes, one based on MD5 and the other based on SHA-1. The total length is $16 + 20 = 36$ bytes. Figure 17.35 shows the hash calculation.

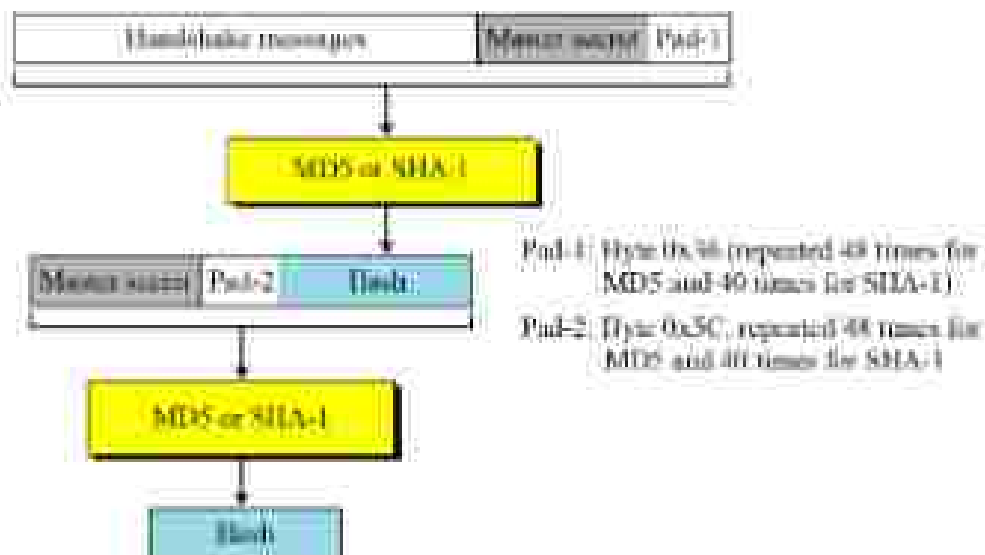


Figure 17.35 Hash calculation for CertificateVerify message

ClientKeyExchange Message

The ClientKeyExchange is the second message sent during the third phase of handshaking. In this message the client provides the keys. The format of the message depends on the specific key exchange algorithms selected by two parties. Figure 17.36 shows the general idea.



Figure 17.36 ClientKeyExchange message

Finished Message

The finished message shows that the negotiation is over. It contains all of the messages exchanged during handshaking, followed by the sender role, the master secret, and the padding. The exact format depends on the type of cipher suite used. The general format is shown in Figure 17.37.

Figure 17.37 shows that there is a concatenation of two hashes in the message. Figure 17.38 shows how each is calculated.

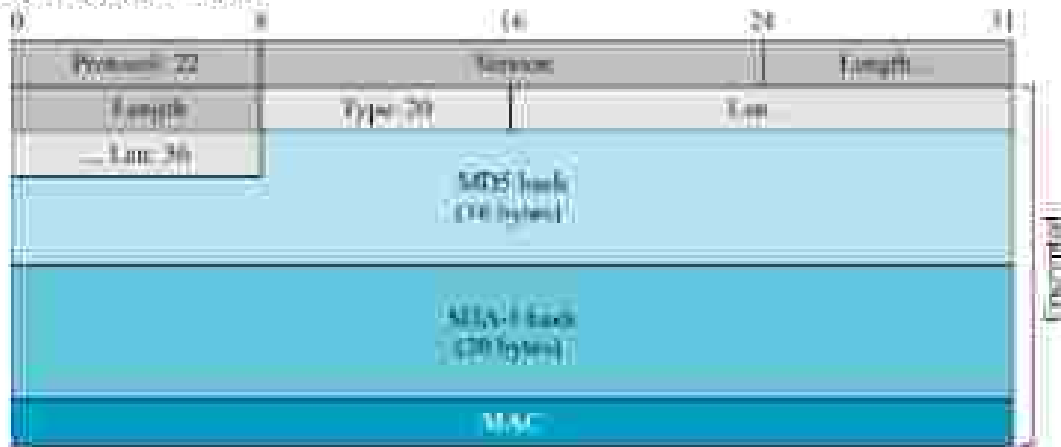


Figure 17.37 Finished message

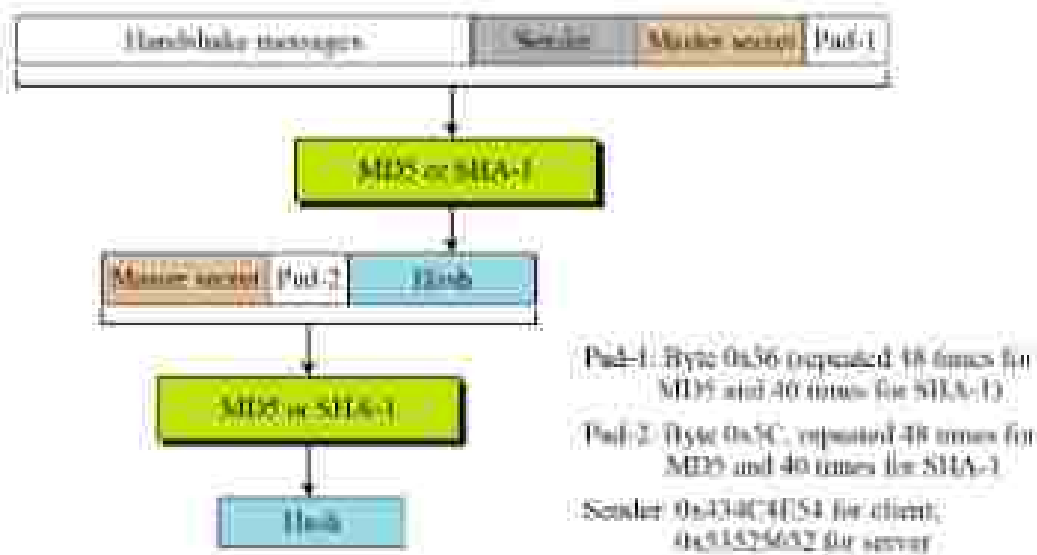


Figure 17.38 Hash calculation for finished message

4.4 Application Data

The record protocol adds a signature (MAC) at the end of the fragment coming from the application layer and then encrypts the fragment and the MAC. After adding the general header with protocol value 23, the record message is transmitted. Note that the general header is not encrypted. Figure 17.39 shows the format.



Figure 17.39 Record Protocol message for application data

5. Transport Layer Security

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

5.1 Version

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

5.2 Cipher Suite

Another minor difference between SSL and TLS is the lack of support for the Fortezza method. TLS does not support Fortezza for key exchange or for encryption/decryption. [Table 17.6](#) shows the cipher suite list for TLS (without export entries).

Cipher suite	Key Exchange	Encryption	Hash
TLS_NULL_WITH_NULL_NULL	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	RSA	NULL	MD5
TLS_RSA_WITH_NULL_SHA	RSA	NULL	SHA-1
TLS_RSA_WITH_RC4_128_MD5	RSA	RC4	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RC4	SHA-1
TLS_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA	SHA-1
TLS_RSA_WITH_DES_CBC_SHA	RSA	DES	SHA-1
TLS_RSA_WITH_AES_128_CBC_SHA	RSA	TDES	SHA-1
TLS_DH_1024_WITH_RC4_128_MD5	DH_1024	RC4	MD5
TLS_DH_1024_WITH_DES_CBC_SHA	DH_1024	DES	SHA-1
TLS_DH_1024_WITH_3DES_EDE_CBC_SHA	DH_1024	3DES	SHA-1
TLS_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES	SHA-1
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES	SHA-1
TLS_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES	SHA-1
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES	SHA-1
TLS_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES	SHA-1
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES	SHA-1
TLS_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES	SHA-1
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES	SHA-1

Table 17.6 Cipher Suite for TLS

5.3 Generation of Cryptographic Secrets

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

Data-Expansion Function

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

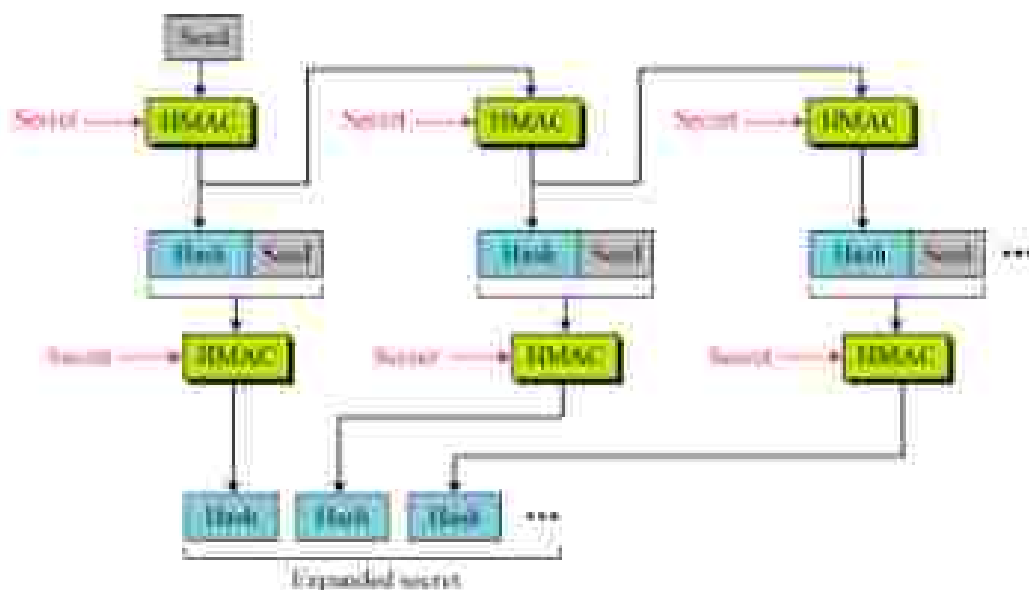


Figure 17.40 Data-expansion function

5.4 Pseudorandom function

- TLS defines a pseudorandom function (PRF) to be the combination of two data-expansion function, one using MD5 and the other SHA-1.
- PRF takes three label and a seed.
- The label and seed are concatenated and serve as the seed for each data expansion function.
- The seed is divided into two halves: each half is used as the secret for each data expansion function.
- The output of two data-expansion function is exclusive-ored together to create the final explained secret. Figure 17.40 shows the idea of PRF.

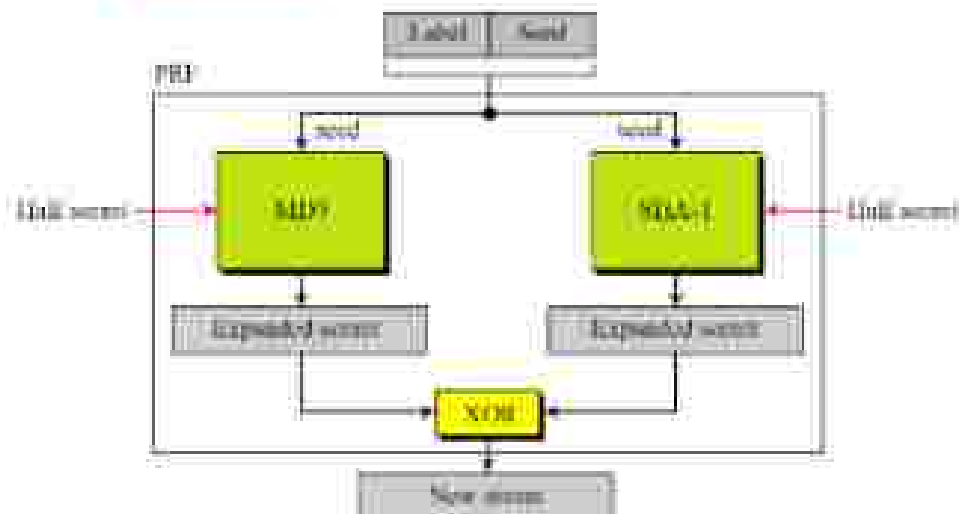


Figure 17.41 PRF

Pre-master Secret

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

Master Secret

- TLS uses the PRF function to create the master secret from the pre-master secret.

- This is achieved by using the pre-master secret as the secret, the string "master secret" as the seed.
- Note that the label is actually the ASCII code of the string "master secret".
- In other words, the label defines the output we want to create, the master secret. [Figure 17.42](#) shows the idea.

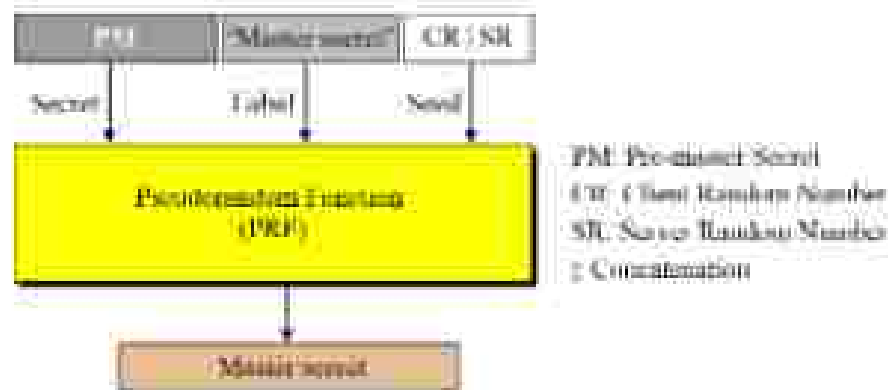


Figure 17.42 Master secret generation

Key material

- TLS uses the PRF to create the key material from the master secret.
- This time the secret is the master secret, the label is the string "key expansion" and the seed is the concatenation of the server random number and the client random number as shown in [Figure 17.43](#).

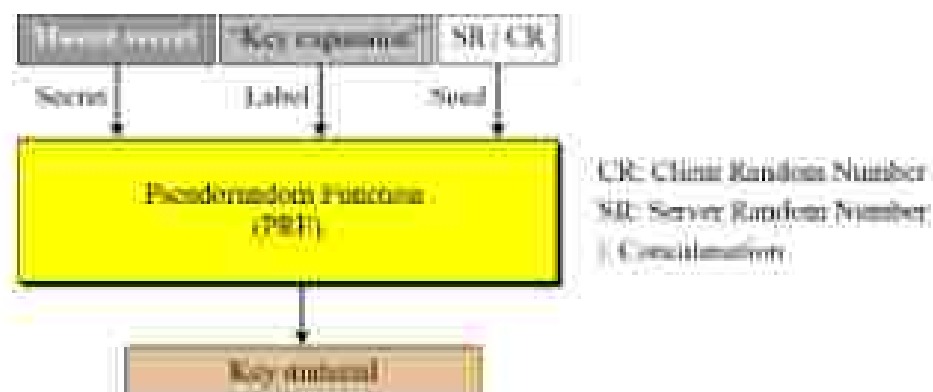


Figure 17.43 Key material generation

5.5 Alert Protocol

- TLS supports all of the alerts defined in SSL except for NoCertificate.
- TLS also adds some new ones to the list. [Table 17.7](#) shows the full list of alerts supported by TLS.

Alert	Description	Meaning
0	<i>CloseNotify</i>	Sender will not send any more messages.
10	<i>UnexpectedMessage</i>	An inappropriate message received.
20	<i>BadRecordMAC</i>	An incorrect MAC received.
21	<i>DecryptionFailed</i>	Decrypted message is invalid.
22	<i>RecordOverflow</i>	Message size is more than $2^{16} + 2048$.
30	<i>DecompressionFailure</i>	Unable to decompress appropriately.
40	<i>HandshakeFailure</i>	Sender unable to finalize the handshake.
42	<i>BadCertificate</i>	Received certificate corrupted.
43	<i>UnsupportedCertificate</i>	Type of received certificate is not supported.
44	<i>CertificateRevoked</i>	Signer has revoked the certificate.
45	<i>CertificateExpired</i>	Certificate has expired.
46	<i>CertificateUnknown</i>	Certificate unknown.
47	<i>IllegalParameter</i>	A field out of range or inconsistent with others.
48	<i>UnknownCA</i>	CA could not be identified.

Table 17.7 Alerts defined for TLS

5.6 Handshake Protocol

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

CertificateVerify Message

- In SSL, the hash used in the *CertificateVerify* message is the hash of the handshake messages plus a pad and the master secret.
- TLS is simplified the process.
- The hash in the TLS is only over the handshake messages, as shown in [figure 17.44](#).

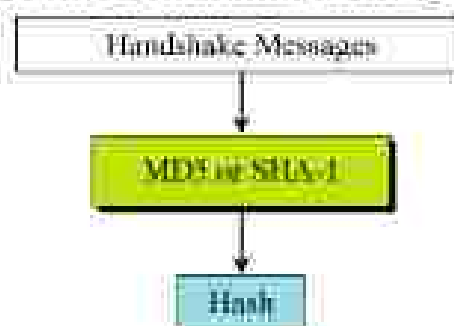


Figure 17.44 Hash for *CertificateVerify* message in TLS

Finished Message

- The calculation of the hash for the *Finished* message has also been changed.
- TLS uses the PRF to calculate two hashes used for the *Finished* message as shown in [figure 17.45](#).

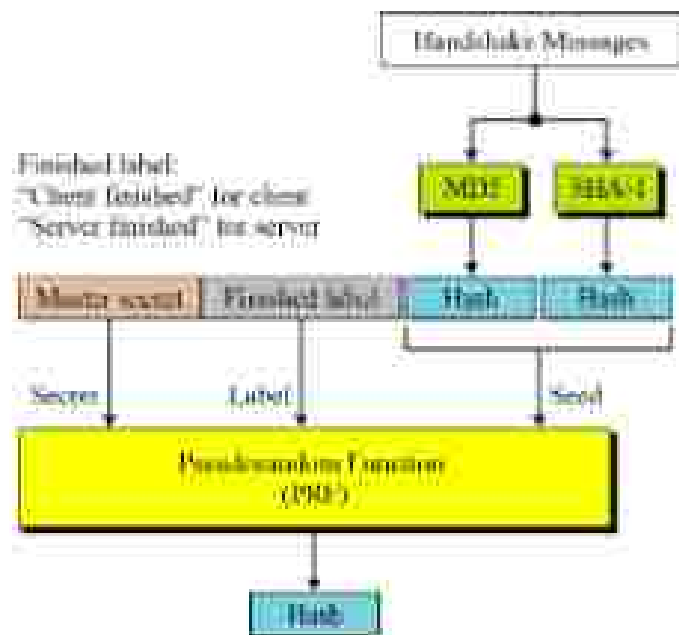


Figure 17.45 Hash for Finished message in TLS

5.7 Record Protocol

- The only change in the record protocol is the use of HMAC for signing the message.
- TLS uses the MAC, as defined as to create the HMAC.
- TLS also adds the protocol version (called compression version) to the text to be signed. **Figure 17.46** shows how the HMAC is formed.

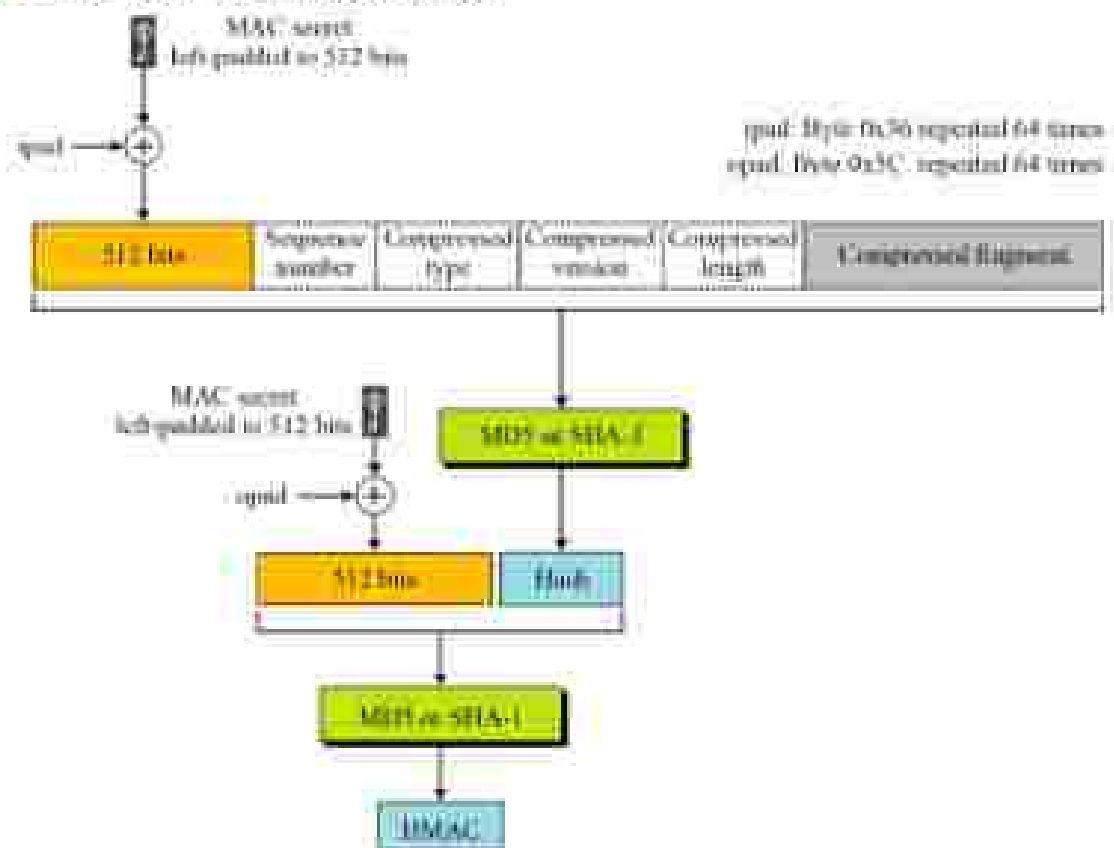


Figure 17.46 HMAC for TLS

Ch-18 – Security at the Network Layer: IPSec

1. IP SECURITY (IPSec)

IP Security (IPSec) is a collection of protocols designed by the Internet Engineering Task Force (IETF) to provide security for a packet at the network level. The network layer in the Internet is often referred to as the Internet Protocol or IP layer. IPSec helps create authenticated and confidential packets for the IP layer as shown in [figure 18.1](#).

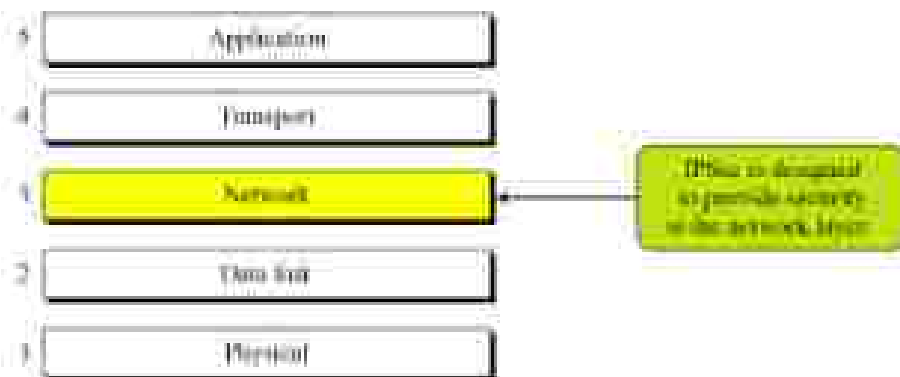


Figure 18.1 TCP/IP Protocol Suite and IPSec

2. Modes of IPSec

IPSec operates in one of two different modes: transport mode or tunnel mode.

2.1 Transport Mode

In transport mode, IPSec protects what is delivered from the transport layer to the network layer. In other words, transport mode protects the network layer payload, the payload to be encapsulated in the network layer, as shown in [figure 18.2](#).

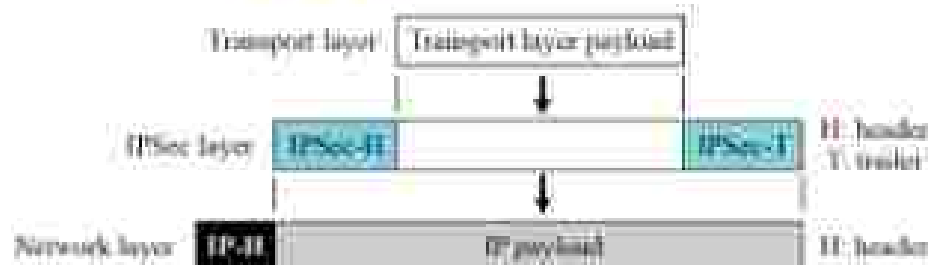


Figure 18.2 IPSec in transport mode

Note that transport mode does not protect the IP header. In other words, transport mode does not protect the whole IP packet, it protects only the packet from the transport layer (the IP layer payload). In this mode, the IPSec header and trailer are added to the information coming from the transport layer. The IP header is added later.

IPSec in transport mode does not protect the IP header; it only protects the information coming from the transport layer.

Transport mode is normally used when we need host-to-host (end-to-end) protection of data. The sending host uses IPSec to authenticate and/or encrypt the payload delivered from the transport layer. The receiving host uses IPSec to check the authentication and/or decrypt the IP packet it to the transport layer. [Figure 18.3](#) shows this concept.



Figure 18.3 Transport mode in action

2.2 Tunnel mode

In tunnel mode, IPSec protects the entire IP packet. It takes an IP packet, including the header, applies IPSec security methods to the entire packet, and then adds a new IP header, as shown in [figure 18.4](#).

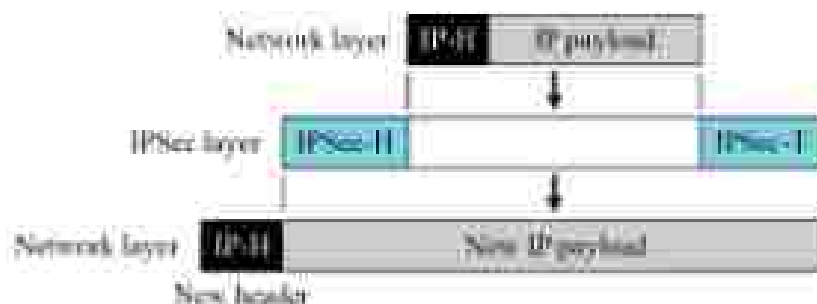


Figure 18.4 IPSec in tunnel mode

The new IP header, as we will see shortly, has different information than the original IP header. Tunnel mode is normally used between two routers, between a host and a router, or between a router and a host, as shown in [Figure 18.5](#). In other words, tunnel mode is used when either the sender or the receiver is not a host. Packet goes through an imaginary tunnel.



Figure 18.5 Tunnel mode in action

2.3 Comparison

In transport mode, the IPSec layer comes between the transport layer and the network layer. In tunnel mode, the flow is from the network layer to the IPSec layer and then back to the network layer again. [Figure 18.6](#) compares the two modes.

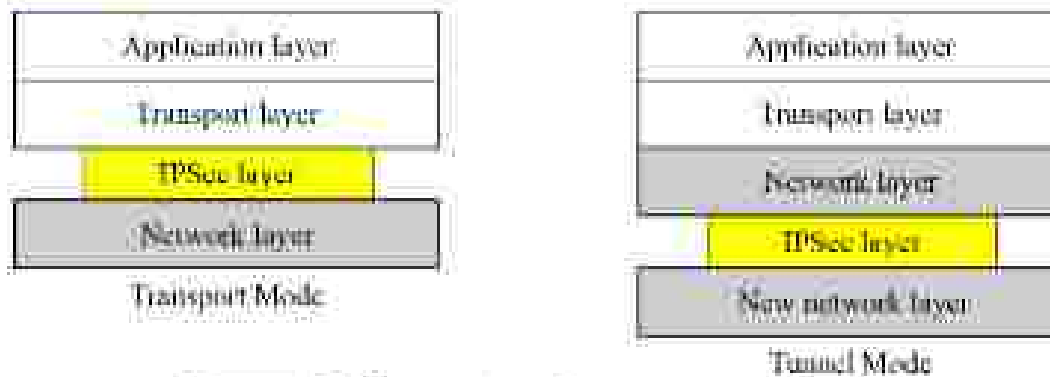


Figure 18.6 Transport mode versus tunnel mode

3. Two security protocols

IPSec defines two protocols—the Authentication Header (AH) protocol is designed to authenticate the source host and to ensure the integrity of the payload carried in the IP packet. The protocol uses a hash function and a symmetric key to create a message digest; the digest is inserted in the authentication header. The AH is then placed in the appropriate location, based on the mode (transport or tunnel). [Figure 18.7](#) shows the fields and the position of the authentication header in transport mode.

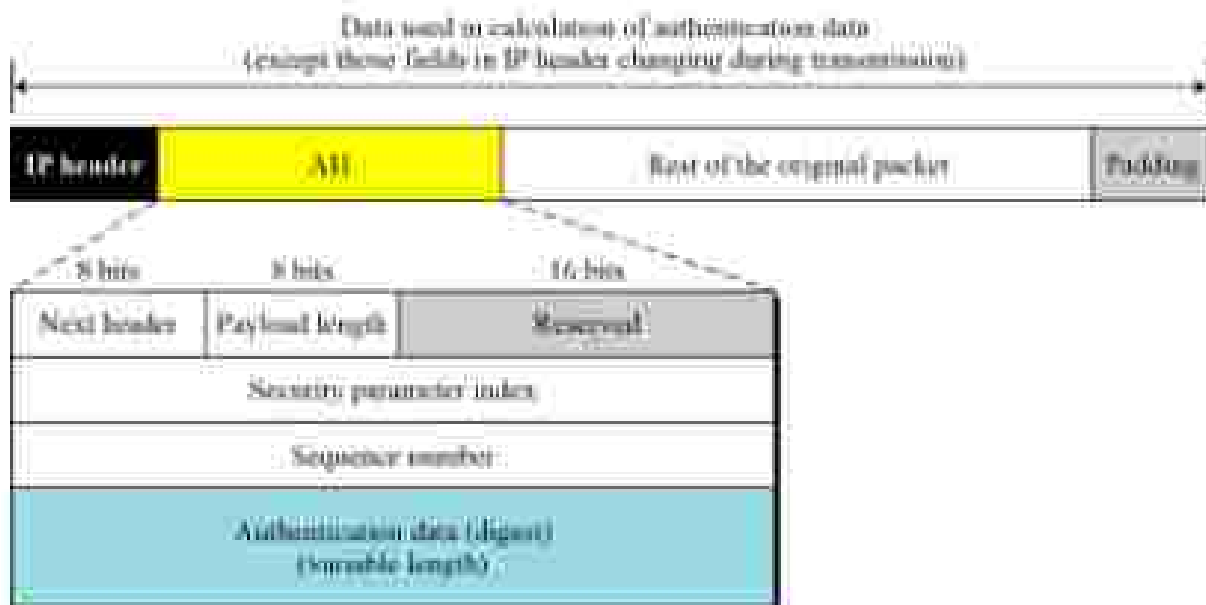


Figure 18.7 Authentication Header (AH) protocol

When an IP datagram carries an authentication header, the original value in the protocol field of the IP header is replaced by the value 51. A field inside the authentication header (the next header field) holds the original value of the protocol field (the type of payload being carried by the IP datagram). The addition of an authentication header follows these steps:

1. An authentication header is added to the payload with the authentication data field set to 0.
2. Padding may be added to make the total length even for a particular hashing algorithm.

3. Hashing is based on the total packet. However, only those fields of the IP header that do not change during transmission are included in the calculation of the message digest (authentication data).
4. The authentication data are inserted in the authentication header.
5. The IP header is added after changing the value of the protocol field to 51.

A brief description of each field follows:

Next Header The 8-bit next header field defines the type of payload carried by the IP datagram (such as TCP, UDP, ICMP, or OSPF). It has the same function as the protocol field in the IP header before encapsulation. In other words, the process copies the value of the protocol field in the IP datagram to this field. The value of the protocol field in the new IP datagram is now set to 51 to show that the packet carries an authentication header.

Payload Length The name of this 8-bit field is misleading. It does not define the length of the payload; it defines the length of the authentication header in 4-byte multiples, but it does not include the first 8 bytes.

Security Parameter Index The 32-bit security parameter index (SPI) field plays the role of a virtual circuit identifier and is the same for all packets sent during a connection called a Security Association (discussed later).

Sequence Number A 32-bit sequence number provides ordering information for a sequence of datagrams. The sequence numbers prevent a playback. Note that the sequence number is not repeated even if a packet is retransmitted. A sequence number does not wrap around after it reaches 2^{32} ; a new connection must be established.

Authentication Data Finally, the authentication data field is the result of applying a hash function to the entire IP datagram except for the fields that are changed during transit (e.g., time-to-live).

The AH protocol provides source authentication and data integrity, but not privacy.

3.2 Encapsulating Security Payload (ESP)

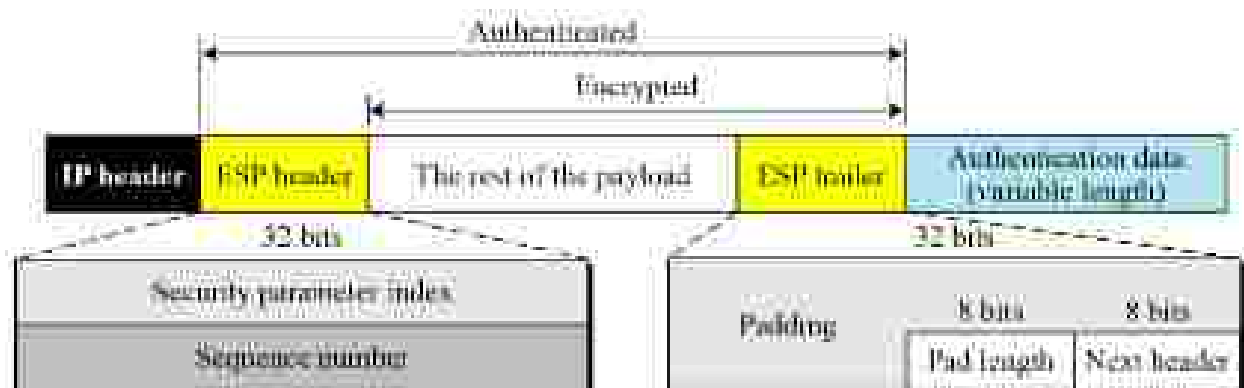


Figure 18-8 ESP

The AH protocol does not provide privacy, only source authentication and data integrity. IPSec later defined an alternative protocol, Encapsulating Security Payload (ESP) that provides source authentication, integrity, and privacy. ESP adds a header and trailer. Note that ESP's authentication

data are added at the end of the packet, which makes its calculation easier. [Figure 18.8](#) shows the location of the ESP header and trailer.

When an IP datagram carries an ESP header and trailer, the value of the protocol field in the IP header is 50. A field inside the ESP trailer (the next-header field) holds the original value of the protocol field (the type of payload being carried by the IP datagram, such as TCP or UDP). The ESP procedure follows these steps:

1. An ESP trailer is added to the payload.
2. The payload and the trailer are encrypted.
3. The ESP header is added.
4. The ESP header, payload, and ESP trailer are used to create the authentication data.
5. The authentication data are added to the end of the ESP trailer.
6. The IP header is added after changing the protocol value to 50.

The fields for the header and trailer are as follows:

- **Security Parameter Index** The 32-bit security parameter index field is similar to that defined for the AH protocol.
- **Sequence Number** The 32-bit sequence number field is similar to that defined for the AH protocol.
- **Pad Length** The 8-bit pad-length field defines the number of padding bytes. The value is between 0 and 255; the maximum value is rare.
- **Padding** The variable-length field (0 to 255 bytes) of 0s serves as padding.
- **Next Header** The 8-bit next-header field is similar to that defined in the AH protocol. It serves the same purpose as the protocol field in the IP header before encapsulation.
- **Authentication Data** Finally, the authentication data field is the result of applying an authentication scheme to parts of the datagram. Note the difference between the authentication data in AH and ESP. In AH, part of the IP header is included in the calculation of the authentication data; in ESP, it is not.

ESP provides source authentication, data integrity, and privacy.

3.3. IPv4 and IPv6

IPSec supports both IPv4 and IPv6, however, AH and ESP are part of the extension header.

3.4 AH versus ESP

The ESP protocol was designed after the AH protocol was already in use. ESP does whatever AH does with additional functionality (privacy). The question is, why do we need AH? The answer is that we don't. However, the implementation of AH is already included in some commercial products, which means that AH will remain part of the Internet until these products are phased out.

3.5 Services provided by IPSec

The two protocols, AH and ESP, can provide several security services for a packets at the network layer. [Table 18.1](#) shows the list of services available for each protocol.

Service	AH	ESP
Access control	yes	yes
Message authentication (message integrity)	yes	yes
Entity authentication (data source authentication)	yes	yes
Confidentiality	no	yes
Replay attack protection	yes	yes

Figure 18.1 IPSec services

Access Control IPSec provides access control indirectly using a Security Association Database (SAD), as we will see in the next section, when a packet arrives at a destination, and there is no Security Association already established for this packet, the packet is discarded.

Message Integrity Message integrity is preserved in both AH and ESP. A digest of data is created and sent by the sender to be checked by the receiver.

Confidentiality The encryption of the message in ESP provides confidentiality. AH, however, does not provide confidentiality. If confidentiality is needed, one should use ESP instead of AH.

Replay Attack Protection In both protocols, the replay attack is prevented by using sequence numbers and a sliding receiver window. Each IPSec header contains a unique sequence number when the security Association is established. The number starts from 0 and increases until the value reaches $2^{32} - 1$ (the size of the sequence number field is 32-bits). Association is deleted and a new one is established. To prevent processing duplicate packets, IPSec mandates the use of a fixed-size window at the receiver. The size of the window is determined by the receiver with a default value of 64. Figure 18.9 shows a replay window. The window is of a fixed size, W . The shaded packets signify received packets that have been checked and authenticated.



Figure 18.9 Replay window

When a packet arrives at the receiver, one of three things can happen, depending on the value of the sequence number.

1. The sequence number of the packet is less than N . This puts the packet to the left of the window. In this case, the packet is discarded. It is either a duplicate of its arrival time has expired.
2. The sequence number of the packet is between N and $(N + W - 1)$, inclusive. This puts the packet inside the window. In this case, if the packet is new (not marked) and it passes the authentication test, the sequence number is marked and the packet is accepted. Otherwise, it is discarded.

3. The sequence number of the packet is greater than $(N + W - 1)$. This puts the packet to the right of the window. In this case, if the packet is authenticated, the corresponding sequence number is marked and the window slides to the right to cover the newly marked sequence number. Otherwise, the packet is discarded. Note that it may happen that a packet arrives with a sequence number much larger than $(N + W)$ (very far from the right edge of the window). These packets, when they arrive, will never be accepted; their time has expired. For example, in Figure 18.9, if a packet arrives with sequence number $(N + W + 3)$, the window slides and the left edge will be at the beginning of $(N + 3)$. This means the sequence number $(N + 2)$ is now out of the window. If a packet arrives with this sequence number, it will be discarded.

4. Security Association

Security Association is a very important aspect of IPSec. IPSec requires a logical relationship, called a **Security Association (SA)**, between two hosts. This section first discusses the idea and then shows how it is used in IPSec.

4.1 Idea of Security Association

A Security Association is a contract between two parties; it creates a secure channel between them. Let us assume that Alice needs to unidirectionally communicate with Bob. If Alice and Bob are interested only in the confidentiality aspect of security, they can get a shared secret key between themselves. We can say that there are two security Associations (SAs) between Alice and Bob, one outbound S and one inbound SA. Each of them stores the value of the key in a variable and the name of the encryption/decryption algorithm in another. Alice uses the algorithm and the key to encrypt a message to Bob; Bob uses the algorithm and the key when he needs to decrypt the message received from Alice. Figure 18.10 shows a simple SA.

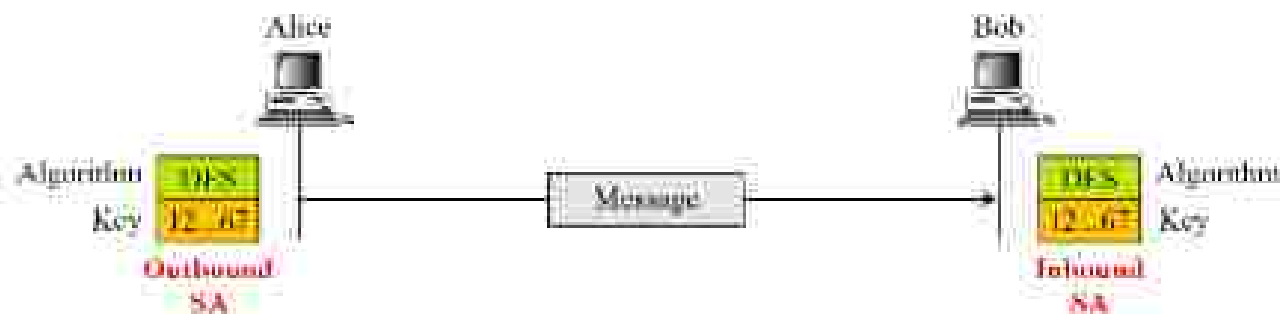


Figure 18.10 Simple SA

The security Association can be more involved if the two parties need message integrity and authentication. Each association needs other data such as the algorithm for message integrity, the key, and other parameters. It can be much more complex if the parties need to use specific algorithms and specific parameters for different protocols, such as IPSec AH or IPSec ESP.

4.2 Security Association Database (SAD)

A Security Association can be very complex. This is particularly true if Alice wants to send messages to many people and Bob needs to receive messages from many people. In addition, each site needs to have both inbound and outbound SAs to allow bidirectional communication. In other words we need a set of SAs that can be collected into a database. The database is called the Security Association Database (SAD). The database can be thought of as a two-dimensional table with each row defining a single SA. Normally, there are two SADs, one inbound and one outbound. Figure 18.11 shows the concept of outbound and inbound SADs for one entity.

When a host needs to send a packet that must carry an IPSec header, the host needs to find the corresponding entry in the outbound SAD to find the information for applying security to the packet. Similarly, when a host receives a packet that carries an IPSec header, the host needs to find

the corresponding entry in the inbound SAD to find the information for checking the security of the packet. This searching must be specific in the sense that the receiving host needs to be sure that correct information is used for processing the packet. Each entry in an inbound sad is selected using a triple index: security parameter index, destination address, and protocol.

Index	SN	OF	AEW	AH/ESP	LT	Mode	MTU
<SPI, DA, P>							
<SPI, DA, P>							
<SPI, DA, P>							
<SPI, DA, P>							

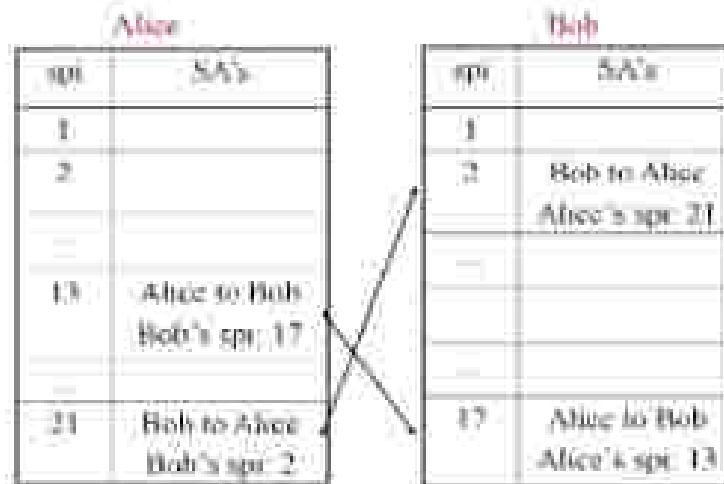
Security Association Database

Legend:

SPI: Security Parameter Index	SN: Sequence Number
DA: Destination Address	OF: Overflow Flag
AH/ESP: Information for either one	AEW: Anti-Replay Window
P: Protocol	LT: Lifetime
Mode: IPSec Mode Flag	MTU: Path MTU (Maximum Transfer Unit)

Figure 18.11 SAD

Example:



- **Security Parameter Index** The security parameter index (SPI) is a 32-bit number that defines the SA at the destination. As we will see later, the SPI is determined during the SA negotiation. The same SPI is included in all IPSec packets belonging to the same inbound SA.
- **Destination Address** The second index is the destination address of the host. We need to remember that a host in the Internet normally has one unicast destination address, but it may have several multicast addresses. IPSec requires that the SAs be unique for each destination address.
- **Protocol** IPSec has two different security protocols: AH and ESP. To separate the parameters and information used for each protocol, IPSec requires that a destination define a different SA for each protocol.

The entries for each row are called the SA parameters. Typical parameters are shown in [Table 16.2](#).

Sequence Number Counter	This is a 32-bit value that is used to generate sequence numbers for the AH or ESP header.
Sequence Number Overflow	This is a flag that defines a station's options in the event of a sequence number overflow.
Anti-Replay Window	This detects an inbound replayed AH or ESP packet.
AH Information	This section contains information for the AH protocol: 1. Authentication algorithm 2. Keys 3. Key lifetime 4. Other related parameters
ESP Information	This section contains information for the ESP protocol: 1. Encryption algorithm 2. Authentication algorithm 3. Keys 4. Key lifetime 5. Initiator vectors 6. Other related parameters
SA Lifetime	This defines the lifetime for the SA.
IPSec Mode	This defines the mode, transport or tunnel.
Path MTU	This defines the path MTU (fragmentation).

Table 18.2 Typical SA Parameters

5. Security Policy

Another import aspect of IPSec is the Security Policy (SP), which defines the type of security applied to a packet when it is to be sent or when it has arrived. Before using the SAD, discussed in the previous section, a host must determine the predefined policy for the packet.

5.1 Security policy Database

Index	Policy
<SA, DA, Name, P, SPort, DPort>	
<SA, DA, Name, P, SPort, DPort>	
<SA, DA, Name, P, SPort, DPort>	
<SA, DA, Name, P, SPort, DPort>	

Legend:

SA: Source Address SPort: Source Port
DA: Destination Address DPort: Destination Port
P: Protocol

Figure 18.12 SPD

Each host that is using the IPSec protocol needs to keep a Security Policy Database (SPD). Again, there is a need for an inbound SPD and an outbound SPD. Each entry in the SPD can be accessed using a sextuple index: source address, destination address, name, protocol, source port, and destination port, as shown in [figure 18.12](#).

Source and destination addresses can be unicast, multicast or wildcard addresses. The name usually defines a DNS entry. The protocol is either AH or ESP. The source and destination ports are the port addresses for the process running at the source and destination hosts.

Outbound SPD

When a packet is to be sent out, the outbound SPD is consulted. [Figure 18.13](#) shows the processing of a packet by a sender.

The input to the outbound SPD is the sextuple index, the output is one of the three following cases:

- **Drop** This means that the packet defined by the index cannot be sent; it is dropped.
- **Bypass** This means that there is no policy for the packet with this policy index; the packet is sent, bypassing the security header application.
- **Apply** In this case, the security header is applied. Two situations may occur.
 - a. If an outbound SA is already established, the triple SA index is returned that selects the corresponding SA from the outbound SAD. The AH or ESP header is formed; encryption, authentication, or both are applied based on the SA selected. The packet is transmitted.
 - b. If an outbound SA is not established yet, the Internet Key Exchange (IKE) protocol (see the next section) is called to create an outbound and inbound SA for this traffic. The outbound SA is added to the outbound SAD by the source; the inbound SA is added to the inbound SAD by the destination.

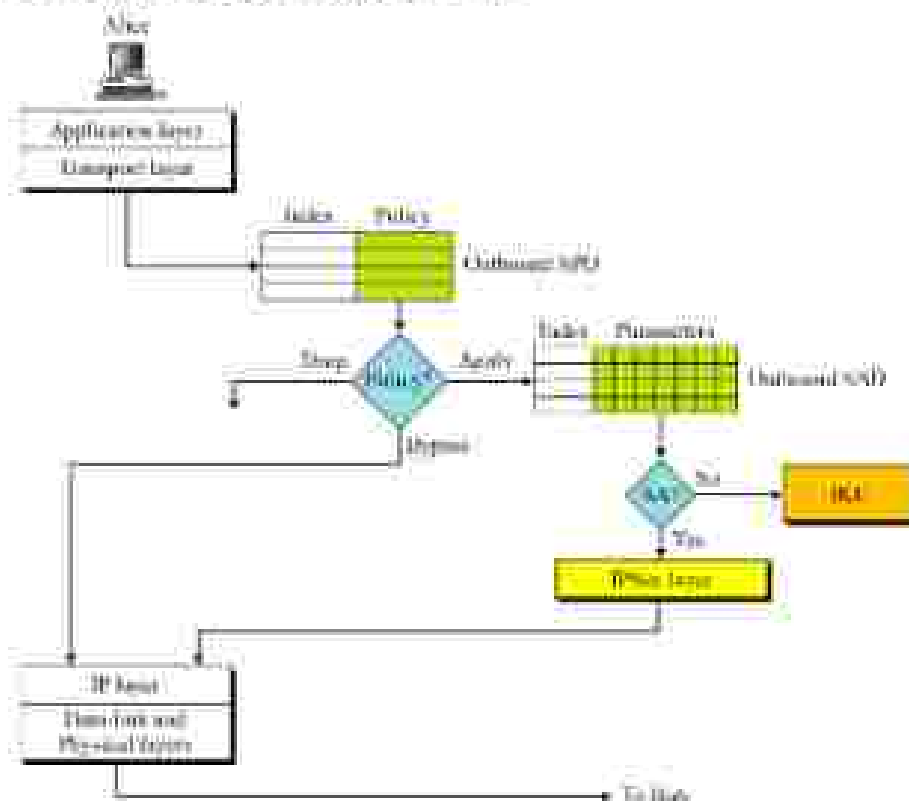


Figure 18.13 outbound processing

Inbound SPD

When a packet arrives, the inbound SPD is consulted. Each entry in the inbound SPD is also accessed using the same sextuple index. Figure 18.14 shows the processing of a packet by a receiver.

The input to the inbound SPD is the sextuple index; the output is one of the three following cases:

- **Discard** This means that the packet defined by that policy must be dropped.
- **Bypass** This means that there is no policy for a packet with this policy index; the packet is processed, ignoring the information from AH or ESP header. The packet is delivered to the transport layer.
- **Apply** In this case, the security header must be processed. Two situations may occur:
 - a. If an inbound SA is already established, the triple SA index is returned that selects the corresponding inbound SA from the inbound SAD. Decryption, authentication, or both are applied. If the packet passes the security criteria, the AH or ESP header is discarded and the packet is delivered to the transport layer.
 - b. If an SA is not yet established, the packet must be discarded.

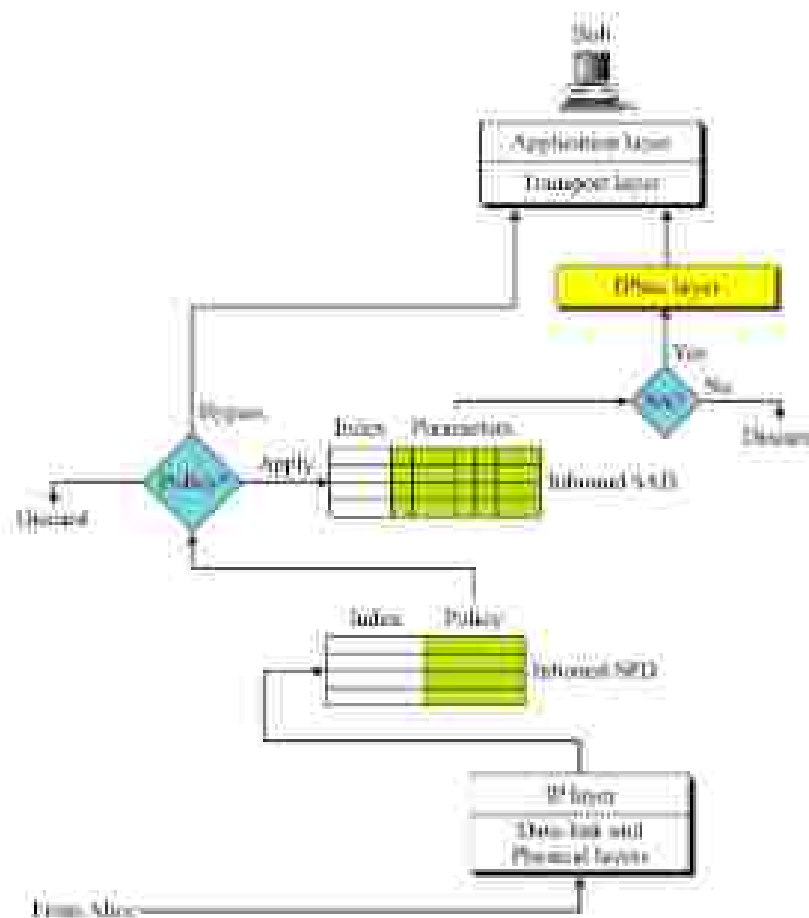


Figure 18.14 inbound processing

6. Internet Key Exchange (IKE)

The Internet Key Exchange (IKE) is a protocol designed to create both inbound and outbound Security Associations. As we discussed in the previous section, when a peer needs to send an IP packet, it consults the Security Policy Database (SPDB) to see if there is an SA for that type of traffic. If there is no SA, IKE is called to establish one.

IKE creates SAs for IPSec.

IKE is a complex protocol based on three other protocols: *Oakley*, SKEME, and ISAKMP, as shown in [figure 18.15](#).



Figure 18.15 IKE Components

The *Oakley* protocol is a key creation protocol based on the Diffie-Hellman key-exchange method, but with some improvements as we shall see shortly.

SKEME, is another protocol for key exchange. It uses public-key encryption for entity authentication in a key-exchange protocol. We will see shortly that one of the methods used by IKE is based on SKEME.

The ISAKMP protocol defines several packets, protocols, and parameters that allow the IKE exchanges to take place in standardized, formatted message to create SAs. We will discuss ISAKMP in the next section as the carrier protocol that implements IKE.

6.1 Improved Diffie-Hellman Key Exchange

The key-exchange idea in IKE is based on the Diffie-Hellman protocol. This protocol provides a session key between two peers without the need for the existence of any previous secret. We have discussed Diffie-Hellman in Chapter 15; the concept is summarized in [Figure 18.16](#).

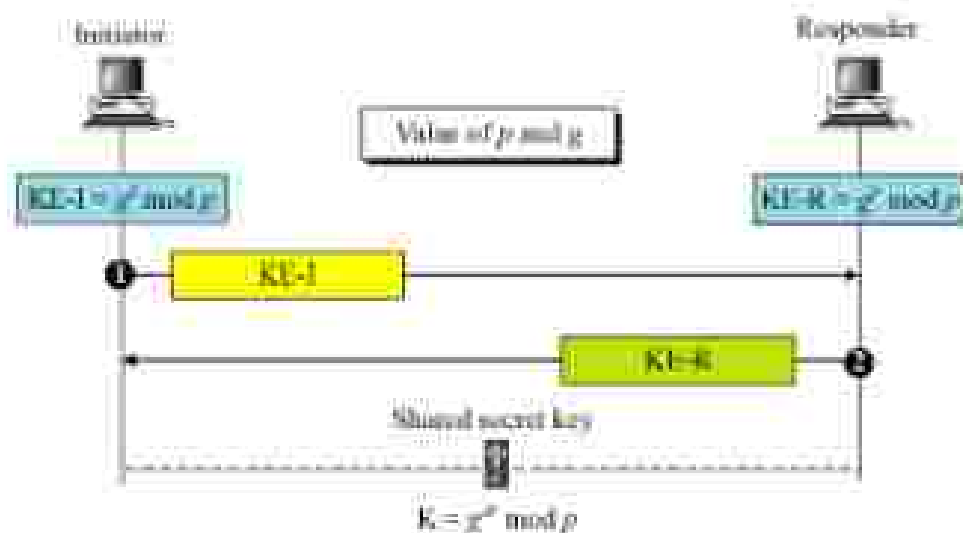


Figure 18.16 Diffie-Hellman key exchange

- In the original Diffie-Hellman key exchange, two parties create a symmetric session key to exchange data without having to remember or store the key for future use.

- Before establishing a symmetric key, the two parties need to choose two numbers p and g . The first number, p , is a large prime on the order of 300 decimal digits (1024 bits). The second number, g , is a generator in the group $\langle \mathbb{Z}_p^*, x \rangle$.
- Alice chooses a large random number i and calculates $KE-I = g^i \bmod p$. She sends $KE-I$ to Bob.
- Bob chooses another large random number r and calculates $KE-R = g^r \bmod p$. He sends $KE-R$ to Alice.
- We refer to $KE-I$ and $KE-R$ as Diffie-Hellman half-keys because each is a *half-key* generated by a peer.
- They need to be combined together to create the full key, which is $K = g^{ir} \bmod p$. K is the symmetric key for the session.

The Diffie-Hellman protocol has some weaknesses that need to be eliminated before it is suitable as an Internet key exchange.

Clogging Attack The first issue with the Diffie-Hellman protocol is the clogging attack or denial-of-service attack. A malicious intruder can send many half-key ($g^i \bmod p$) messages to Bob, pretending that they are from different sources. Bob then need to calculate different responses ($g^r \bmod p$) and at the same time calculate the full-key ($g^{ir} \bmod p$). This keeps Bob so busy that he may stop responding to any other messages. He denies services to clients. This can happen because the Diffie-Hellman protocol is computationally intensive.

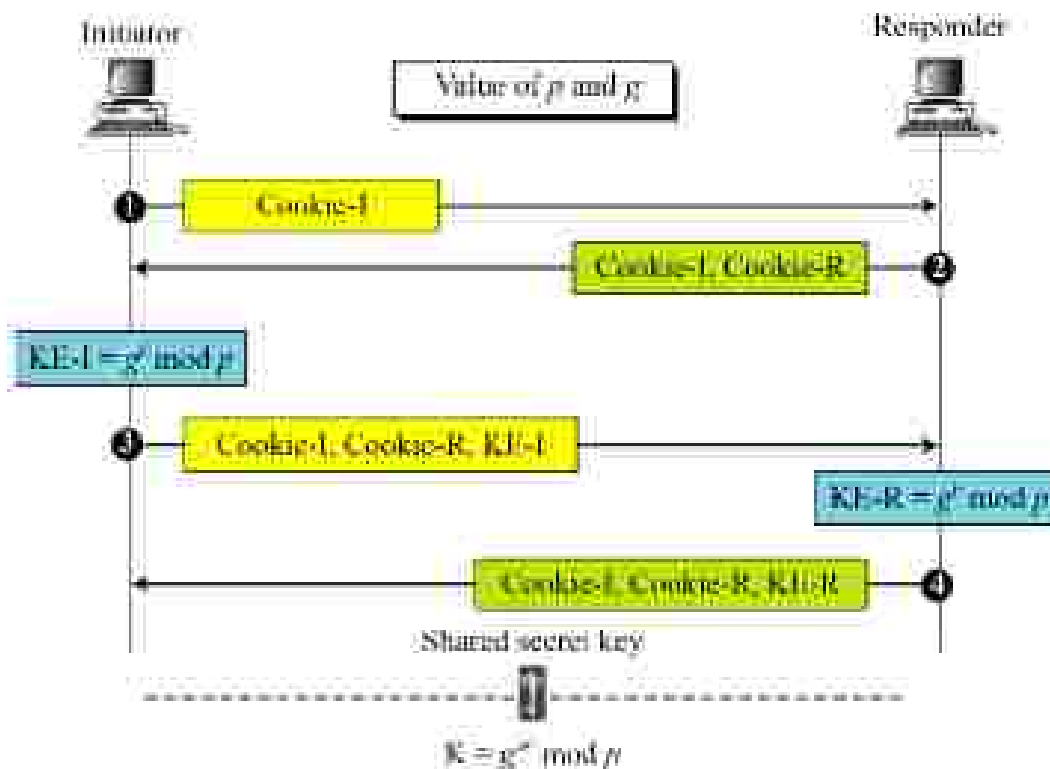


Figure 18.17 Diffie-Hellman with cookies.

To prevent this clogging attack, we can add two extra messages to the protocol to force the two parties to send cookies. Figure 18.17 shows the refinement that can prevent a clogging attack. The cookie is the result of hashing a unique identifier of the peer (such as IP address, port number, and protocol), a secret random number known to the party that generates the cookie, and a timestamp.

The initiator sends its own cookie; the responder its own. Both cookies are repeated, unchanged, in every following message. The calculations of half-keys and the session key are postponed until the cookies are returned. If any of the peers is a hacker attempting a clogging attack, the cookies are not returned; the corresponding party does not spend the time and effort to calculate the half-key or the session key. For example, if the initiator is a hacker using a bogus IP address, the initiator does not receive the second message and cannot send the third message. The process is aborted.

To protect against a clogging attack, IKE uses cookies.

Replay Attack Like other protocols we have seen so far, Diffie-Hellman is vulnerable to a replay attack: the information from one session can be replayed in a future session by a malicious intruder.

To prevent this, we can add nonces to the third and fourth messages to preserve the freshness of the message.

To protect against a replay attack, IKE uses nonces.

Man-in-The-Middle Attack The third, and the most dangerous, attack on the Diffie-Hellman protocol is the man-in-the-middle attack, previously discussed in Chapter 15. Eve can come in the middle and create one key between Alice and herself and another key between Bob and herself. Thwarting this attack is not as simple as the other two. We need to authenticate each party. Alice and Bob need to be sure that the integrity of the messages is preserved and that both are authenticated to each other.

Authentication of the messages exchanged (message integrity) and the authentication of the parties involved (entity authentication) that each party proves his/her claimed identity. To do this, each must prove that it possesses a secret.

To protect against man-in-the-middle attack, IKE requires that each party shows that it possesses a secret.

In IKE, the secret can be one of the following:

- a. A preshared secret key.
- b. A preknown encryption/decryption public-key pair. An entity must show that a message encrypted with the announced public key can be decrypted with the corresponding private key.
- c. A preknown digital signature public-key pair. An entity must show that it can sign a message with its private key which can be verified with its announced public key.

6.2 IKE Phases

IKE creates SAs for a message-exchange protocol such as IPSec. IKE, however, needs to exchange confidential and authenticated messages. What protocol provides SAs for IKE itself? The reader may realize that this requires a never-ending chain of SAs: IKE must create SAs for IPSec, protocol X must create SAs for IKE, protocol Y needs to create SAs for protocol X, and so on. To solve this dilemma and, at the same time, make IKE independent of the IPSec protocol, the designers of IKE divided IKE into two phases. In phase I, IKE creates SAs for phase II. In phase II, IKE creates SAs for IPSec or some other protocol. Phase I is generic; phase II is specific for the protocol.

IKE is divided into two phases: phase I and phase II. Phase I creates SAs for phase II; phase II creates SAs for a data exchange protocol such as IPSec.

Still, the question remains: How is phase I protected? In the next sections we show how phase I uses an SA that is formed in a gradual manner. Earlier messages are exchanged in plaintext; later messages are authenticated and encrypted with the keys created from the earlier messages.

6.3 Phases and Modes

To allow for a variety of exchange methods, IKE has defined modes for the phases. So far, there are two modes for phase I: the *main mode* and the *aggressive mode*. The only mode for phase II is the *quick mode*. Figure 18.18 shows the relationship between phases and modes.

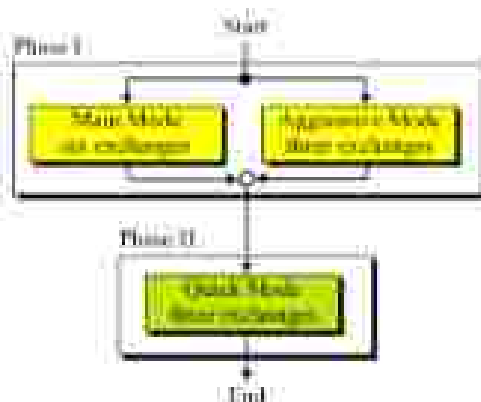


Figure 18.18 IKE Phases

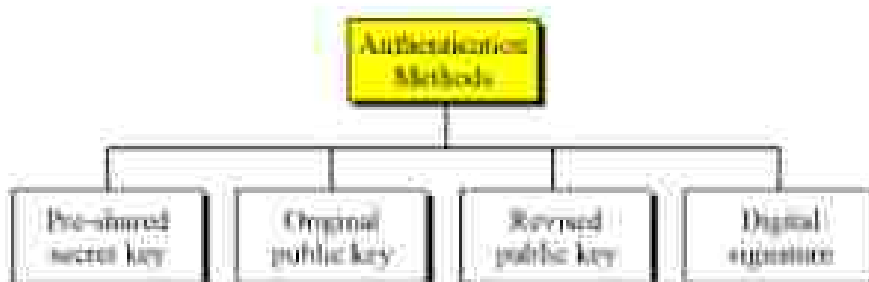


Figure 18.19 Main-mode or aggressive-mode methods

Based on the nature of the pre-secret between the two parties, the phase I modes can use one of four different authentication methods: the preshared secret key method, the original public-key method, the revised public-key method, or the digital signature method, as shown in Figure 18.19.

6.4 Phase I: Main Mode

In the main mode, the initiator and the responder exchange six messages. In the first two messages, they exchange cookies (to protect against a clogging attack) and negotiate the SA parameters. The initiator sends a series of proposals; the responder selects one of them. When the first two messages are exchanged, the initiator and the responder know the SA parameters and are confident that the other party exists (no clogging attack occurs).

In the third and fourth messages, the initiator and responder usually exchange their half-keys (g and g^r of the Diffie-Hellman method) and their nonces (for replay protection). In some methods other information is exchanged; that will be discussed later. Note that the half-keys and nonces are not sent with the first two messages because the two parties must first ensure that a clogging attack is not possible.

After exchanging the third and fourth messages, each party can calculate the common secret between them in addition to its individual hash digest.

The common secret SKEYID (secret key ID) is dependent on the calculation method as shown below. In the equations, prf (pseudorandom function) is a keyed hash function defined during the negotiation phase.

$SKEYID = prf(\text{preshared-key}, N-I || N-R)$

(preshared-key method)

$SKEYID = prf(N-I || N-R, g^H)$

(public-key method)

$SKEYID = prf(\text{hash}(N-I || N-R), \text{Cookie-I} || \text{Cookie-R})$

(digital signature)

Other common secrets are calculated as follows:

$SKEYID_d = prf(SKEYID, g^H || \text{Cookie-I} || \text{Cookie-R} || 0)$

$SKEYID_a = prf(SKEYID, SKEYID_d || g^H || \text{Cookie-I} || \text{Cookie-R} || 1)$

$SKEYID_e = prf(SKEYID, SKEYID_a || g^H || \text{Cookie-I} || \text{Cookie-R} || 2)$

Preshared Secret-Key Method: In the preshared secret-key method, a symmetric key is used for authentication of the peers to each other. Figure 18.20 shows shared-key authentication in the main mode.

(The common secret $SKEYID$ (secret key ID), $SKEYID_a$ is the authentication key, $SKEYID_e$ is used for the encryption key, $SKEYID_d$ (derived key) is a key to create other keys.)

$KE-I$ ($KE-R$): Initiator's (responder's) half-key	HDR : General header including cookies
$N-I$ ($N-R$): Initiator's (responder's) nonce	 Encrypted with $SKEYID_e$
$ID-I$ ($ID-R$): Initiator's (responder's) ID	
$HASH-I$ ($HASH-R$): Initiator's (responder's) hash	

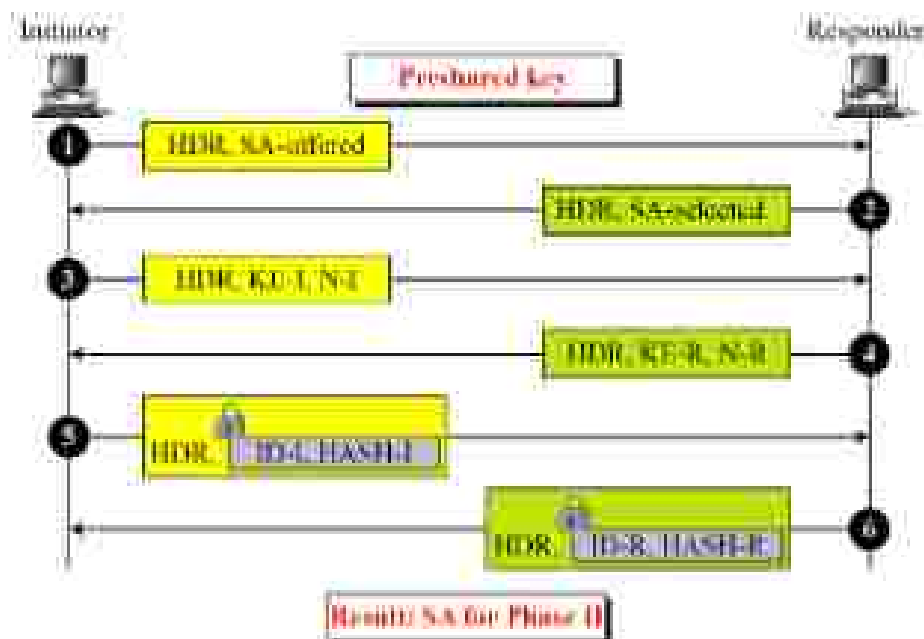


Figure 18.20 Main mode, preshared secret-key method

In the first two messages, the initiator and responder exchange cookies (inside the general header) and SA parameters. In the next two messages, they exchange the half-keys and the nonces (see Chapter 15). Now the two parties can create $SKEYID$ and the two keyed hashes ($HASH-I$ and $HASH-R$). In the fifth and sixth messages, the two parties exchange the created hashes and their IDs. To protect the IDs and hashes, the last two messages are encrypted with $SKEYID_e$.

Note that the pre-shared key is the secret between Alice (initiator) and Bob (responder). Eve (intruder) does not have access to this key. Eve cannot create SKEYID and therefore cannot create either HASH_I or HASH_R. Note that the IDs need to be exchanged in messages 5 and 6 to allow the calculation of the hash.

Original Public-Key Method In the original public-key method, the initiator and the responder prove their identities by showing that they possess a private key related to their announced public key. Figure 18.21 shows the exchange of messages using the original public-key method.

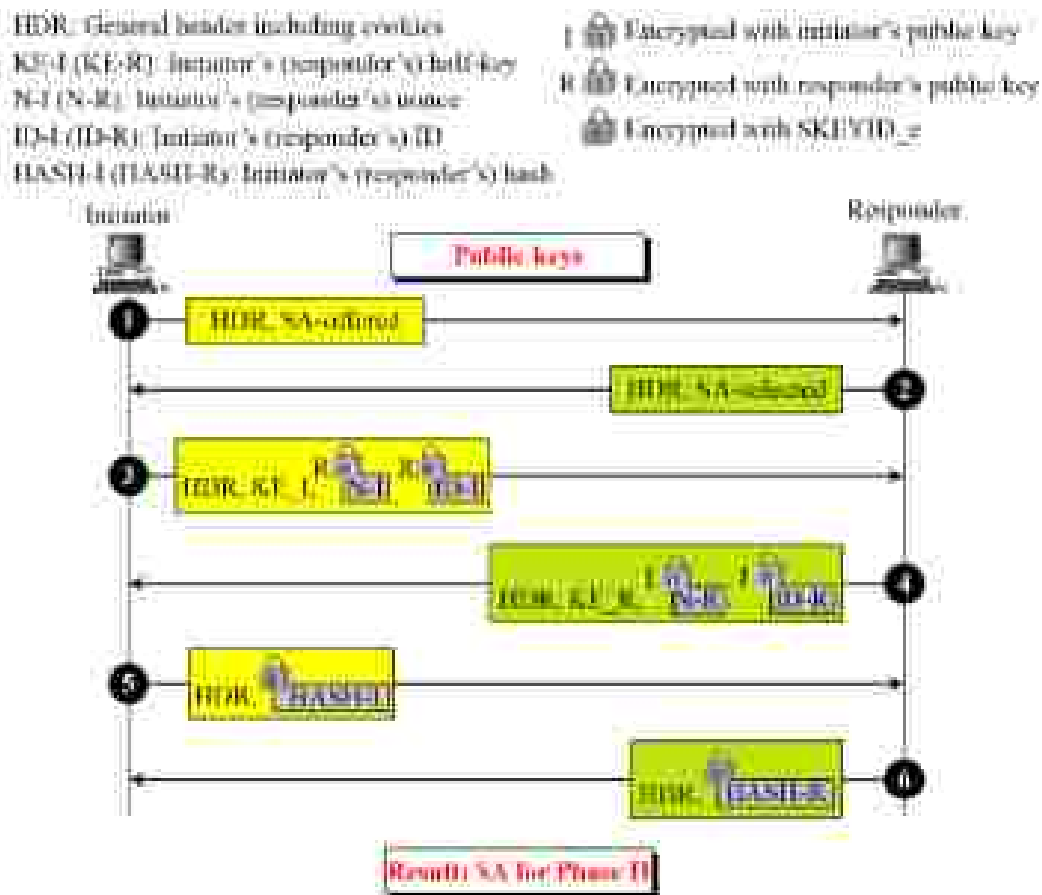


Figure 18.21 Main mode, original public-key method

The first two messages are the same as in the previous method. In the third message, the initiator sends its half-key, the nonce, and the ID. In the fourth message, the responder does likewise. However, the nonces and IDs are encrypted by the public key of the receiver and decrypted by the private key of the receiver. As can be seen from figure 18.21, the nonces and IDs are encrypted separately, because, as we will see later, they are encoded separately from separate payloads.

One difference between this method and the previous one is that the IDs are exchanged with the third and fourth messages instead of the fifth and sixth messages. The fifth and sixth messages just carry the HASHs.

Revised Public-Key Method The original public-key method has some drawbacks. First, two instances of public-key encryption/decryption place a heavy load on the initiator and responder. Second, the initiator cannot send its certificate encrypted by the public key of the responder, since anyone could do this with a false certificate. The method was revised so that the public key is used only to create a temporary secret key, as shown in figure 18.22.

Note that two temporary secret keys are created from a hash of nonces and cookies. The initiator uses the public key of the responder to send its nonce. The responder decrypts the nonce and calculates the initiator's temporary secret key. After that the half-key, the ID, and the optional certificate can be decrypted.

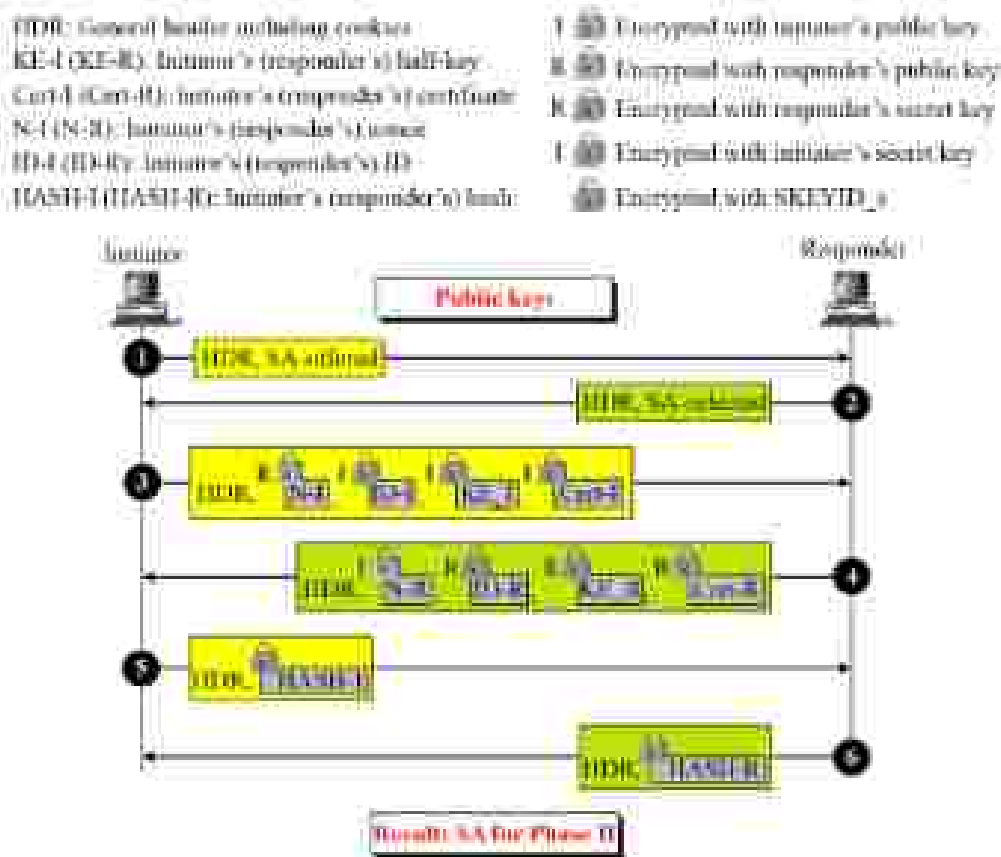


Figure 18.22 Main mode, revised public-key method

Digital Signature Method In this method, each party shows that it possesses the certified private key related to a digital signature. Figure 18.23 shows the exchanges in this method. It is similar to the preshared_key method except for the SKEYID calculation.

Note that in this method the sending of the certificates is optional. The certificate can be sent here because it can be encrypted with SKEYID_s, which does not depend on the signature key. In message 5, the initiator signs all the information exchanged in messages 1 to 4 with its signature key. The responder verifies the signature using the public key of the initiator, which authenticates the initiator. Likewise, in message 6, the responder signs all the information exchanged with its signature key. The initiator verifies the signature.

HDR: General header including cookies
 Sig-I: Initiator's signature on messages 1-4
 Sig-R: Initiator's signature on messages 1-5
 Cert-I (Cert-R): Initiator's (responder's) certificate
 N-I (N-R): Initiator's (responder's) nonce
 KI-I (KI-R): Initiator's (responder's) half key
 ID-I (ID-R): Initiator's (responder's) ID
 Encrypted with SKEYID₀

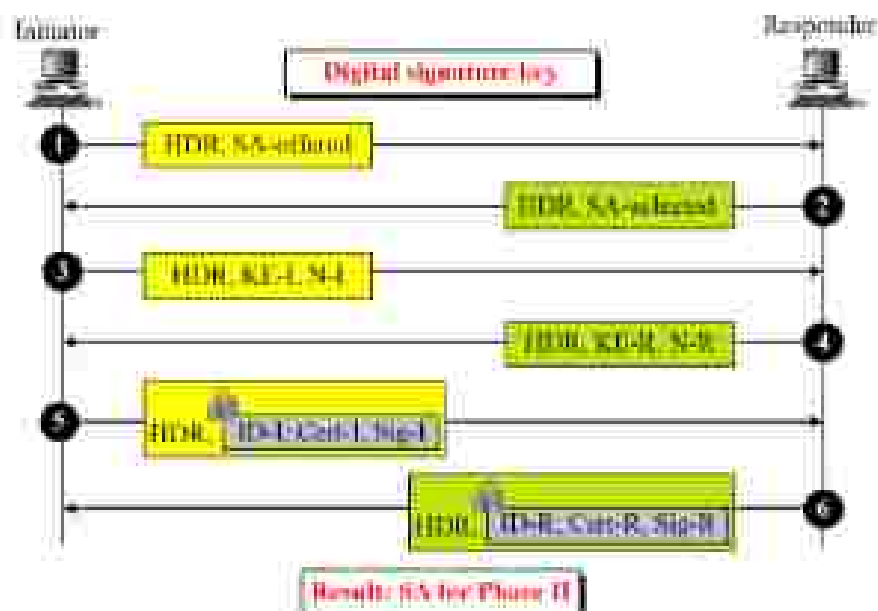


Figure 18.23 Main mode, digital signature method

6.5 Phase 1: Aggressive Mode

Each aggressive mode is a compressed version of the corresponding main mode. Instead of six messages, only three are exchanged. Messages 1 and 3 are combined to make the first message. Messages 2, 4, and 6 are combined to make the second message. Message 5 is sent as the third message. The idea is the same.

Preshared-Key Method Figure 18.24 shows the preshared-key method in the aggressive mode. Note that after receiving the first message, the responder can calculate SKEYID and consequently, HASH-R. But the initiator cannot calculate SKEYID until it receives the second message. HASH-I in the third message can be encrypted.

KI-I (KI-R): Initiator's (responder's) half key
 N-I (N-R): Initiator's (responder's) nonce
 HASH-I (HASH-R): Initiator's (responder's) hash
 HDR: General header including cookies
 Encrypted with SKEYID₀
 ID-I (ID-R): Initiator's (responder's) ID



Figure 18.24 Aggressive mode, preshared-key method

Original Public-Key Method Figure 18.25 shows the exchange of messages using the original public-key method in the aggressive mode. Note that the responder can calculate the SKEYID and HASH-R after receiving the first message, but the initiator must wait until it receives the second message.

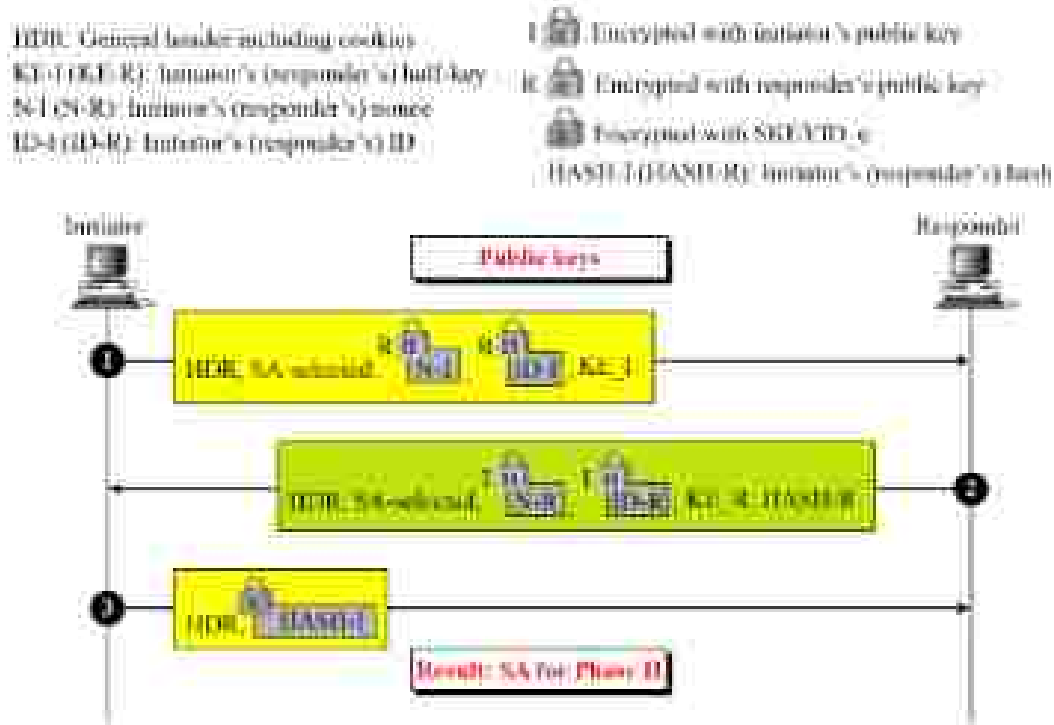


Figure 18.25 Aggressive mode, original public-key method

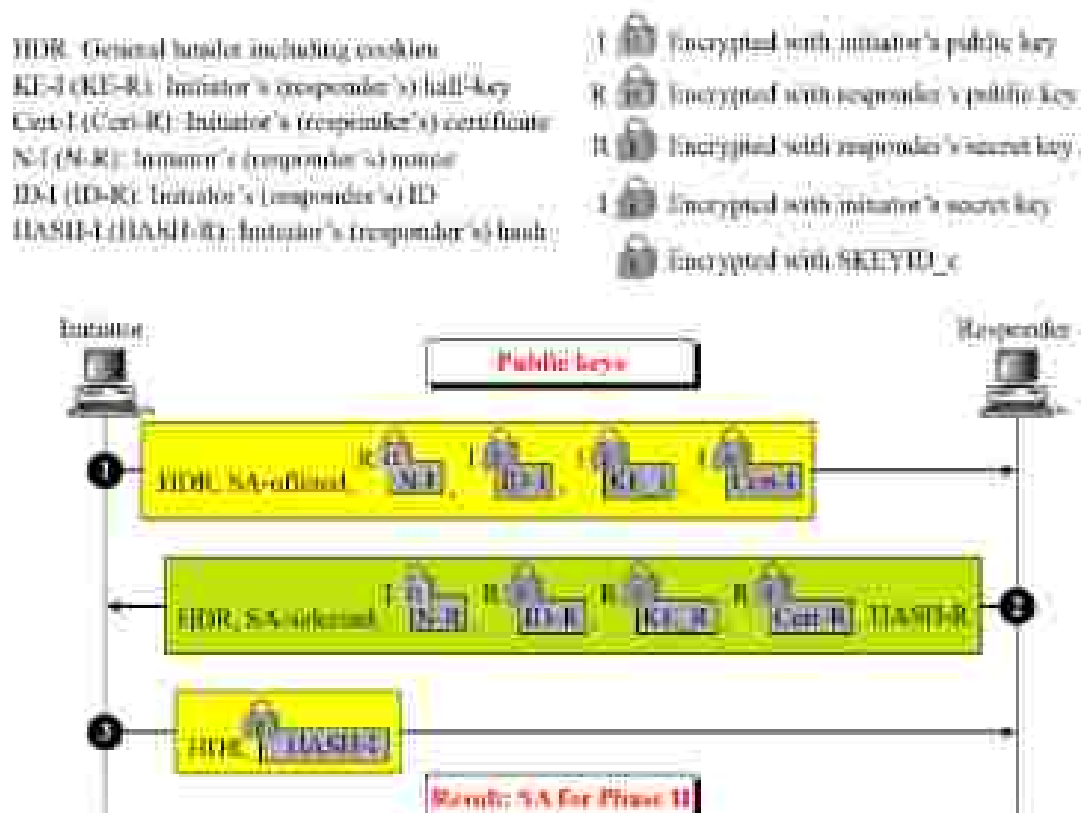


Figure 18.26 Aggressive mode, revised public-key method

Revised Public-Key Method Figure 18.26 shows the revised public-key method in the aggressive mode. The idea is the same as for the main mode, except that some messages are combined.

Digital Signature Method Figure 18.27 shows the digital signature method in the aggressive mode. The idea is the same as for the main mode, except that some messages are combined.

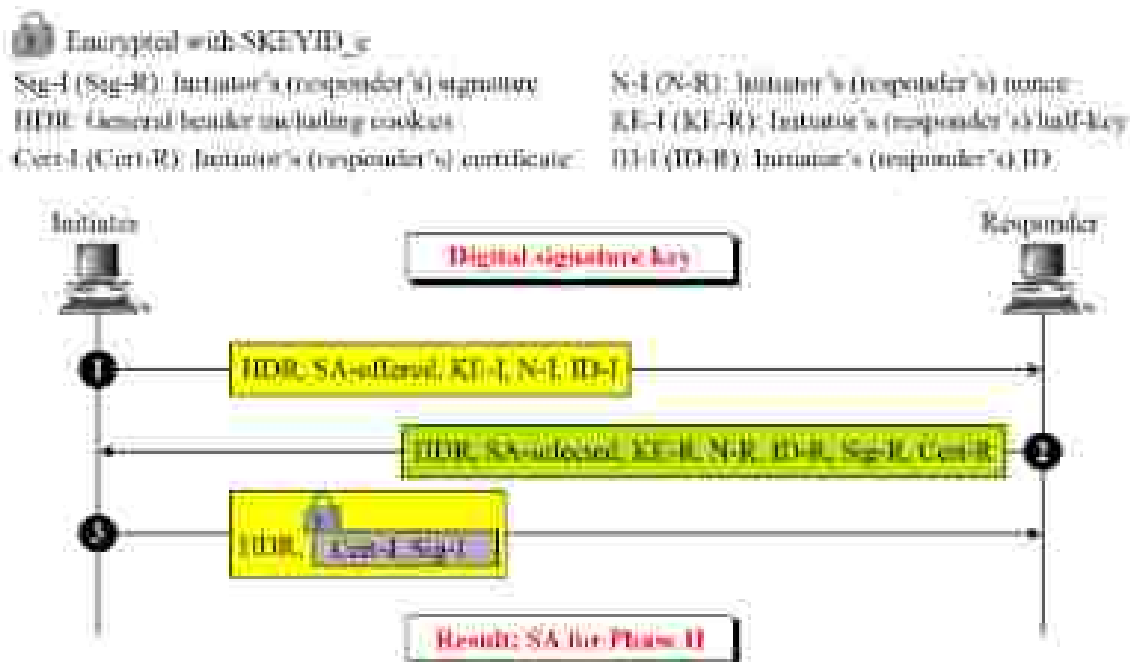


Figure 18.27 Aggressive mode, digital signature method

6.6 Phase II: Quick Mode

After SAS have been created in either the main mode or the aggressive mode, phase II can be started. There is only one mode defined for phase II so far, the quick mode. This mode is under the supervision of the IKE SAS created by phase I. However, each quick-mode method can follow any main or aggressive mode.

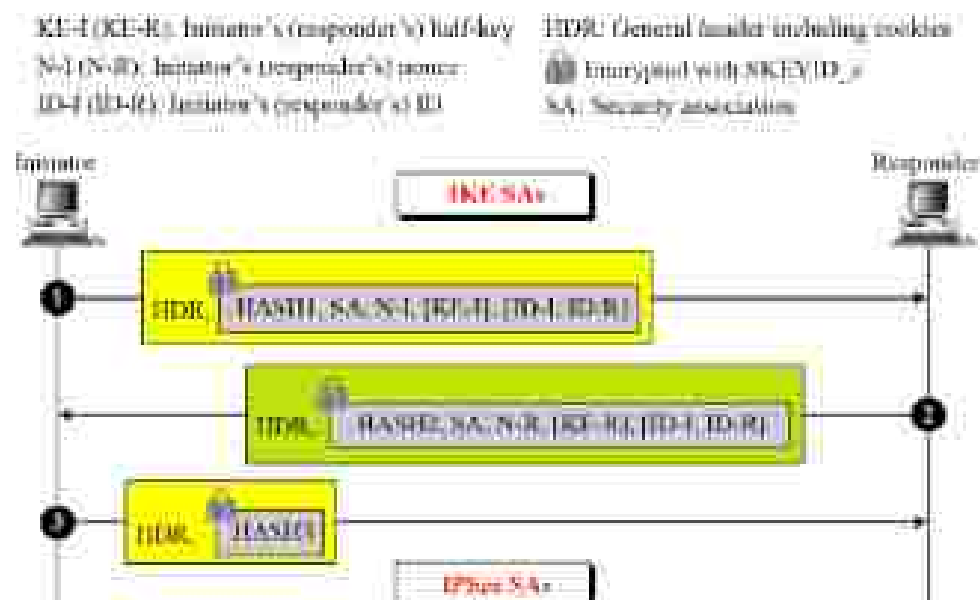


Figure 18.28 Quick Mode

The quick mode uses IKE SAS to create IPSec SAS (or SAS for any other protocol). [Figure 18.28](#) shows the messages exchanged during the quick mode.

In phase II, either party can be the initiator. That is, the initiator of phase II can be the initiator of phase I or the responder of phase I.

The initiator sends the first message, which includes the keyed-HMAC HASH1 (explained later), the entire SA created in phase I, a new nonce (N-I), an optional new Diffie-Hellman half-key (KE-I), and the optional IDs of both parties. The second message is similar, but carries the keyed-HMAC HASH2, the responder nonce (N-R), and, if present, the Diffie-Hellman half-key created by the responder. The third message contains only the keyed-HMAC HASH3.

The key material created after phase II is unidirectional; there is one key for each direction.

6.7 SA Algorithms

Before leaving this section, let us give the algorithms that are negotiated during the first two IKE exchanges.

Diffie-Hellman Groups The first negotiation involves the Diffie-Hellman group used for exchanging half-keys. Five groups have been defined, as shown in [Table 18.3](#).

Value	Description
1	Modular exponentiation group with a 768-bit modulus
2	Modular exponentiation group with a 1024-bit modulus
3	Elliptic curve group with a 155-bit field size
4	Elliptic curve group with a 185-bit field size
5	Modular exponentiation group with a 1680-bit modulus

Table 18.3 Diffie-Hellman groups

Hash Algorithms The hash algorithms that are used for authentication are shown in [Table 18.4](#).

Value	Description
1	MD5
2	SHA
3	Tiger
4	SHA2-256
5	SHA2-384
6	SHA2-512

Table 18.4 Hash Algorithms

Encryption Algorithms The encryption algorithms that are used for confidentiality are shown in [Table 18.5](#). All of these are normally used in CBC mode.

Index	Description
1	DES
2	IDEA
3	Blowfish
4	RC4
5	IDEA
6	CAST
7	AES

Table 18.5 Encryption algorithms

ISAKMP

The ISAKMP protocol is designed to carry messages for the IKE exchange.

6.8 General Header

The format of the general header is shown in [figure 18.29](#).



Figure 18.29 ISAKMP general header

Initiator Cookie This 32-bit field defines the cookie of the entity that initiates the SA establishment, SA notification, or SA deletion.

Responder Cookie This 32-bit field defines the cookie of the responding entity. The value of this field is 0 when the initiator sends the first message.

Next Payload This 8-bit field defines the type of payload that immediately follows the header. We discuss the different types of payload in the next section.

Major Version This 4-bit version defines the major version of the protocol. Currently, the value of this field is 1.

Minor Version This 4-bit version defines the minor version of the protocol. Currently, the value of this field is 0.

Exchange Type This 8-bit field defines the type of exchange that is being carried by the ISAKMP packets. We have discussed the different exchange types in the previous section.

Flags This is an 8-bit field in which each bit defines an option for the exchange. So far only the three least significant bits are defined. The encryption bit, when set to 1, specifies that the rest of the payload will be encrypted using the encryption key and the algorithm defined by SA. The commitment bit, when set to 1, specifies that encryption material is not received before the establishment of the SA. The authentication bit, when set to 1, specifies that the rest of the payload, though not encrypted, is authenticated for integrity.

Message ID This 32-bit field is the unique message identity that defines the protocol state. This field is used only during the second phase of negotiation and is set to 0 during the first phase.

Message Length Because different payloads can be added to each packet, the length of a message can be different for each packet. This 32-bit field defines the length of the total message, including the header and all payloads.

6.9 Payloads

The payloads are actually designed to carry messages. Table 18.6 shows the types of payloads.

Type	Name	Brief Description
0	None	Used to show the end of the payloads
1	SA	Used for starting the negotiation
2	Proposal	Contains information used during SA negotiation
3	Transform	Defines a security transform to create a secure channel
4	Key Exchange	Carries data used for generating keys
5	Identification	Carries the identification of communicating party
6	Certificate	Carries a public-key certificate
7	Certificate Request	Used to request a certificate from the other party
8	Hash	Carries data generated by a hash function
9	Signature	Carries data generated by a signature function
10	Noise	Carries randomly generated data as a noise
11	Notification	Carries error message or status associated with an SA
12	Delete	Carries one more SA that the sender has deleted
13	Vendor	Defines vendor-specific extension

Table 18.6 Payloads

Each payload has a generic header and some specific fields. The format of the generic header is shown in Figure 18.30.



Figure 18.30 Generic payload header

Next Payload This 8-bit field identifies the type of the next payload. When there is no next payload, the value of this field is 0. Note that there is no type field for the current payload. The type of the current payload is determined by the previous payload or the general header (if the payload is the first one).

Payload Length This 16-bit field defines the length of the total payload (including the generic header) in bytes.

SA Payload The SA payload is used to negotiate security parameters. However, these parameters are not included in the SA payload; they are included in two other payloads (proposal and transform) that we will discuss later. An SA payload is followed by one or more proposal payloads, and each proposal payload is followed by one or more transform payloads. The SA payload just defines the domain of interpretation field and the situation field. Figure 18.31 shows the format of the SA payload.

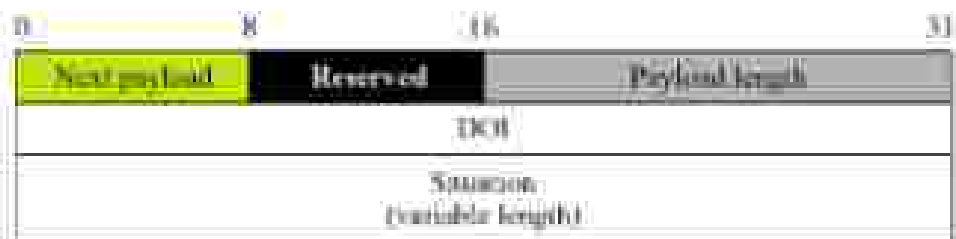


Figure 18.31 SA payload

The fields in the generic header have been discussed. The descriptions of the other fields follow:

Proposal # The initiator defines a number for the proposal so that the responder can refer to it. Note that an SA payload can include several proposal payloads. If all of the proposals belong to the same set of protocols, the proposal number must be the same for each protocol in the set. Otherwise, the proposals must have different numbers.

Protocol ID This 8-bit field defines the protocol for the negotiation. For example, IKE phase 1 = 0, ESP = 1, AH = 2, etc.

SPI Size This 8-bit field defines the size of the SPI in bytes.

Number of Transforms This 8-bit field defines the number of transform payloads that will follow this proposal payload.

SPI This variable-length field is the actual SPI. Note that if the SPI does not fill the 32-bit space, no padding is added.

Transform Payload The transform payload actually carries attributes of the SA negotiation.

Figure 18.33 shows the format of the transform payload.

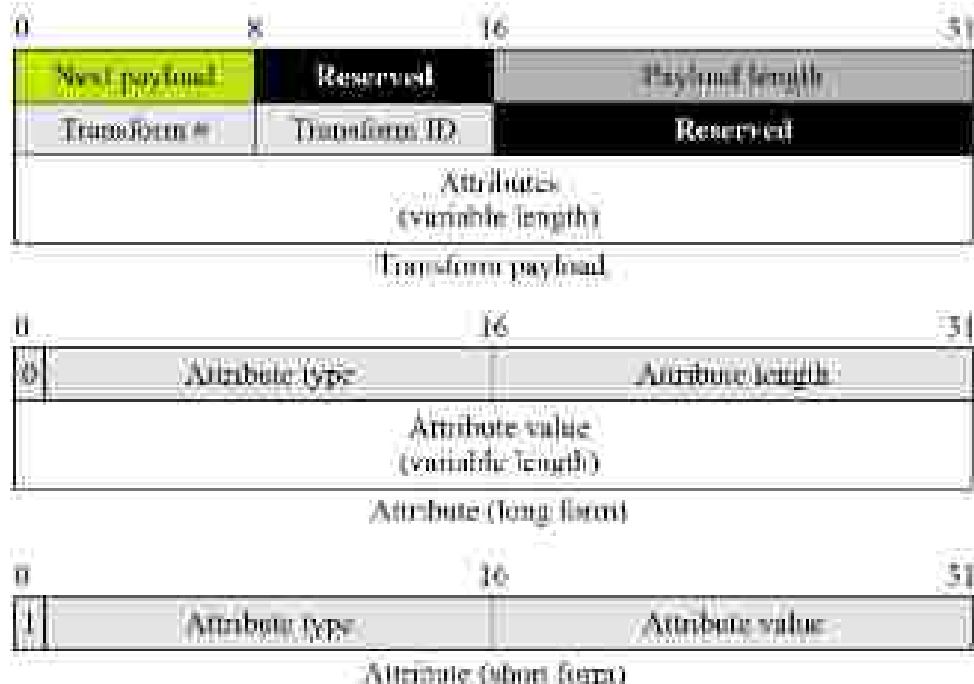


Figure 18.33 Transform payload

The fields in the generic header have been discussed. The descriptions of the other fields follow:

Transform # This 8-bit field defines the transform number. If there is more than one transform payload in a proposal payload, then each must have its own number.

Transform ID This 8-bit field defines the identity of the payload.

Attributes Each transform payload can carry several attributes. Each attribute itself can have three or two subfields (see Figure 18.33). The attribute type subfield defines the type of attribute as defined in the DOI. The attribute length subfield, if present, defines the length of the attribute value. The attribute value field is two bytes in the short form or of variable-length in the long form.

Key-Exchange Payload The key exchange payload is used in those exchanges that need to send Preliminary keys that are used for creating session keys. For example, it can be used to send a Diffie-Hellman half-key. Figure 18.34 shows the format of the key-exchange payload.



Figure 18.34 Key-exchange payload

The fields in the generic header have been discussed. The description of the KE file

KE This variable-length field carries the data needed for creating the session key.

Identification Payload The identification payload allows entities to send their identifications to each other. Figure 18.35 shows the format of the identification payload.

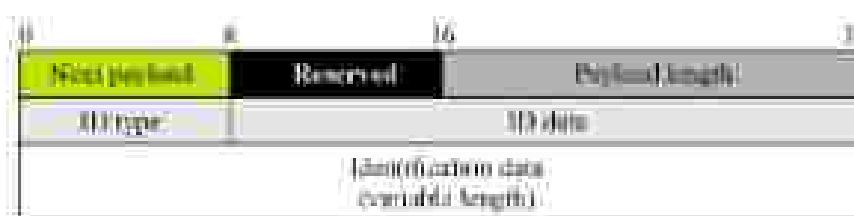


Figure 18.35 Identification payload

The fields in the generic header have been discussed. The descriptions of the other fields follow:

ID Type This 8-bit field is DOI specific and defines the type of ID being used.

ID Data This 24-bit field is usually set to 0.

Identification Data The actual identity of each entity is carried in this variable-length field.

Certification Payload Anytime during the exchange, an entity can send its certification (for public-encryption/decryption keys or signature keys). Although the inclusion of the certification payload in an exchange is normally optional, it needs to be included if there is no secure directory available to distribute the certificates. Figure 18.36 shows the format of the certification payload.

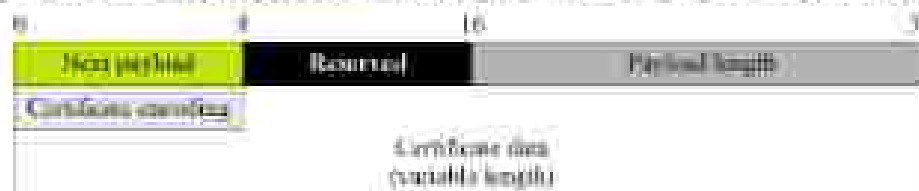


Figure 18.36 Certification payload

The fields in the generic header have been discussed. The descriptions of the other fields follow:

Certificate Encoding This 8-bit field defines the encoding (type) of the certificate. Table 18.7 shows the types defined so far.

Value	Type
0	None
1	Wrapped X.509 Certificate
2	PGP Certificate
3	DNS Signed Key
4	X.509 Certificate—Signature
5	X.509 Certificate—Key Exchange
6	Kerberos Tokens
7	Certification Revocation List
8	Authority Revocation List
9	SPKI Certificate
10	X.509 Certificate—Attribute

Table 18.7 Certification types

Certificate Data This variable-length field carries the actual value of the certificate. Note that the previous field implicitly defines the size of this field.

Certificate Request Payload Each entity can explicitly request a certificate from the other entity using the certificate request payload. Figure 18.37 shows the format of this payload.



Figure 18.37 Certification request payload

The fields in the generic header have been discussed. The descriptions of the other fields follow:

- **Certificate Type** This 8-bit field defines the type of certificate as previously defined in the certificate payload.
- **Certificate Authority** This is a variable-length field that defines the authority for the type of certificate issued.

Hash Payload The hash payload contains data generated by the hash function as described in the IKE exchanges. The hash data guarantee the integrity of the message or part of the ISAKMP states. Figure 18.38 shows the format of the hash payload.



Figure 18.38 Hash payload

The fields in the generic header have been discussed. The description of the last field follows:

Hash data This of the variable-length field carries the hash data generated by applying the hash function to the message or part of the ISAKMP states.

Signature Payload The signature payload contains data generated by applying the digital signature procedure over some part of the message or ISAKMP state. Figure 18.39 shows the format of the Signature payload.



Figure 18.39 Signature payload

The fields in the generic header have been discussed. The description of the last field follows:

Signature This variable-length field carries the digest resulting from applying the signature over part of the message or ISAKMP state.

Nonce Payload The nonce payload contains random data used as a nonce to assure liveness of the message and to prevent a replay attack. Figure 18.40 shows the format of the nonce payload.



Figure 18.40 Nonce payload

The fields in the generic header have been discussed. The description of the last field follows:

Nonce This is a variable-length field carrying the value of the nonce.

Notification Payload During the negotiation process, sometimes a party needs to inform the other party of the status or errors. The notification payload is designed for these two purposes. Figure 18.41 shows the format of the notification payload.

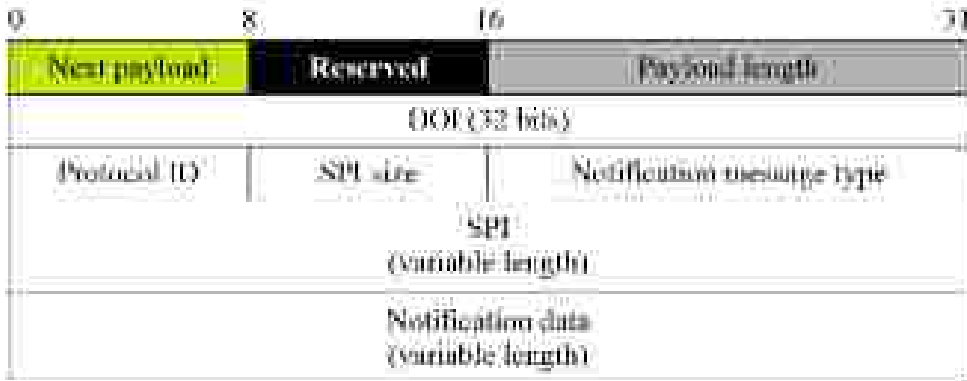


Figure 18.41 Notification payload

The fields in the generic header have been discussed. The descriptions of the other fields follow:

DOI This 32-bit field is the same as that defined for the Security Association payload.

Protocol ID This 8-bit field is the same as that defined for the proposal payload.

SPI Size This 8-bit field is the same as that defined for the proposal payload.

Notification Message Type This 16-bit field specifies the status or the type of error that is to be reported.

SPI This variable-length field is the same as that defined for the proposal payload.

Notification Data This variable-length field can carry extra textual information about the status or errors.

Table 18.9 is a list of status notifications. Values from 16385 to 24575 and 40960 to 65535 are reserved for future use. Values from 32768 to 40959 are for private use.

Value	Description
16384	CONNECTED
24576-32767	DOI-specific codes

Table 18.9 Status notification values

Delete Payload The delete payload is used by an entity that has deleted one or more SAS and needs to inform the peer that these SAS are no longer supported. Figure 18.42 shows the format of the delete payload.



Figure 18.42 Delete payload

The fields in the generic header have been discussed. The descriptions of the other fields follow:

DOI This 32-bit field is the same as that defined for the Security Association payload.

Protocol ID This 8-bit field is the same as that defined for the proposal payload.

SPI Size This 8-bit field is the same as that defined for the proposal payload.

Number of SPIs This 16-bit field defines the number of SPIs. One delete payload can report the deletion of several SAS.

SPIs This variable-length field defines the SPIs of the deleted SAS.

Vendor Payload ISAKMP allows the exchange of information particular to a specific vendor. Figure 18.43 shows the format of the vendor payload.



Figure 18.43 Vendor payload

The fields in the generic header have been discussed. The description of the last field follows:

Vendor ID This variable-length field defines the constant used by the vendor.

BUFFER OVERFLOW AND MALICIOUS SOFTWARE

- Buffer overflow is a commonly known mistake that exist in some C implementations.
- These classes of bugs are extremely dangerous, as they write past the end of a buffer or array and hence corrupt the process stack.
- Often, they change the return address of a process after a function call to a secret memory location where a malicious code is planted.
- This calls for the understanding of these kinds of errors to develop secure coding principles to lessen these security threats in systems.
- A buffer, often visualised as an array is a contiguous space of related variables of the same data type. In C or C++ there are no automatic checks on the buffer, which means a user may write past a buffer.
- This phenomenon is known as buffer overflow. Arrays, like all variables in C, can be either static or dynamic.
- Static variables are allocated at load time on data segment. Dynamic variables are allocated at run time on the stack. We shall discuss on the overflow of dynamic buffers, commonly referred as stack-based buffer overflows.

For example, consider the following C code snippet as a simple example of buffer overflow,

```
int main() {  
    int buffer[10];  
    buffer[20] = 5;  
}
```

- Most compilers will compile the above program without errors.
- This seemingly simple mistake can be used by the authors of malicious software to trigger their codes and thus compromise security.
- In order to understand the principle behind these attacks let us look at the organization of the process in memory.
- A process, which is a program in execution, is divided into three regions: text, data and stack, as shown in Fig.
- The text region is decided by the program and is made of binary instructions and read-only data. Any effort to write in this area leads to a segmentation fault.
- The initialized or uninitialised data region stores static variables.
- If the available memory is rendered unavailable, the process is blocked and rescheduled with a larger memory space between the text and stack segments.



- A stack is a contiguous block of memory containing data, which grows from higher memory address to lower ones. A stack of object has the property of LIFO (Last In First Out), which means that the last object placed on the stack is the first object to be removed.
- Two of the most important stack operations are the PUSH and the POP. PUSH adds an element at the top of the stack, while POP removes the last element from the stack.
- This removes the last element from the stack. High level languages use stacks to implement procedure or function calls.
- After the call of a function, the program flow is changed towards the function being called, but after the function finishes its task, the control returns to the instruction immediately after the call instruction.
- The parameters of the function call, the return address and the local variables used in the function are dynamically located in the stack.
- A register called the stack pointer (sp) points to the top of the stack. The bottom of the stack is a fixed address, while the sp is updated by the kernel at run time to accommodate for extra space required by the local variables of the function.
- In general architectures like Intel, SPARC and MIPS processors, the stack grows down, that is from higher memory address to lower memory addresses.
- The stack pointer is thus decremented to allocate the extra space. The reduction of the sp to allocate the extra space depends on the gcc compiler, which converts the code to assembly instructions.

Malicious programs:

- Malicious programs, are those programs which try to subvert the expected operation of secured and benign codes, buffer overflow attacks give the adversary the power to change the return address.
- Typically, the changed return address points back to the buffer itself, where a malicious exploit is written.
- The exploit could be a program to run a shell in the victim computer.
- The ability to open up a shell in a target computer, gives in turn the ability to the attacker to run various types of harmful programs.
- The two most commonly known categories of malicious programs are worms and viruses. Although often worms are also referred to as viruses, it would be appropriate to mention at this point the difference between these two types.

Worms:

- These are programs that can run independently. It can propagate a full working version of itself to other machines.
- The term tries to draw an analogy of these programs with that of parasites which live inside a host and uses its resources for its existency.

- The concept of a worm program that spreads itself among machines was first mentioned in the classic science fiction, named 'The Shockwave Rider' by John Brunner in 1975. Thereafter researchers at Xerox PARC reported their experiments on the worm programs in an article in the Communications of the ACM.
- Interestingly, the initial worm programs were designed to do "good" work in a distributive way, and not break systems.
- However as of now, worms are used only as a special category of harmful programs.

Viruses:

- These programs on the other hand, cannot run independently. It requires the host program to run and activate them.
- These are analogous to biological viruses, which are not alive themselves, but invade and corrupt host cells.
- The first time when the word VIRUS was used to indicate a program that infects a computer was by David Gerrold in a science fiction named "When Harlie Was One".
- The first formal usage of the term was made by Fred Cohen around defined computer virus- to be a program that can infect other programs by modifying them to include a copy of itself.

Logic Bombs:

- A logic bomb is a malicious program which has typically two parts: payload and the trigger.
- The payload typically is a malicious piece of code, and the trigger is usually a Boolean logic that triggers the malicious code when the condition is satisfied. The trigger is usually developed using local conditions like date.
- Logic bombs are inserted stealthily into a big program. They often have an objective of causing financial harm.
- Thus, the logic bomb creates violation of security when some external events occur.
- Historically, there are several examples of such codes. Recently, a person fired from his job, implanted a logic bomb in the system such that it will be fired on a particular date after he left the company.
- The logic bomb would delete all the files in the system.
- The man was ultimately detained in jail for several years for this misdoing. Such examples of logic bombs which fires on a particular time or date, say April 1, 2010 is known as time bombs.
- Once, a program was posted in the USENET news network to make system administration easy. In the program which was to be compiled and run with root privilege, dangerously there were lines to change to the root directory and remove all files.

Trojans:

- Trojans are malicious programs that perform some harmless activities in addition to some malicious activities.
- A Trojan horse is a program with some known or documented effect and some undocumented or unexpected effects.
- A Classic example is password grabbing programs. A fake login prompt asks the user to enter the password.

- The program then obtains the password and displays an error message showing incorrect password.
- Then the actual login prompt is displayed, thus making the user believe that he typed in the password incorrectly. NOW he enters the system with correct password, but meanwhile he has given away his password.
- A propagating Trojan horse or a replicating Trojan horse is an example of Trojan horse which can create a copy of itself. One of the earliest examples, called the animal, was an example of a Trojan which would create an extra copy of itself.

Spyware:

- Spyware is software that is used to collect information from a computer and transmit it to another computer.
- The information which the spyware exports to another system could be of the same type as done by viruses, but the essential difference is that spywares do not replicate.
- They are also different from Trojans, in the fact that the spywares do not deceive the user but does its malice passively.
- Spywares Often get downloaded when we are viewing some webpage, the phenomenon being called drive by download.

Examples of information gathered include the following:

- **Passwords** A key logger is often used to retrieve the information typed in as password by the user during login. A use of a virtual key board can often be a way to subvert actions of these key loggers which uses the key strokes to ascertain the content of the typed message.
- **Credit Card Numbers and Bank Secrets** These are special subjects of spywares for their financial benefits.

Adware:

- Adware have much similarity with spywares.
- They are also not self-replicating like the spywares.
- Their objective is more from the marketing perspective.
- Examples of them could include the advertisements popping up unintentionally depending on the content/ or website we are viewing. Their purpose is to increase the sale of some products. They can also transmit information which may be useful from the marketing point of view.

Intrusion Detection Systems (IDS)

- Intrusion Detection is the process of monitoring the events occurring in a computer system or network.
- Signs of violations of computer security policies, acceptable use policies, or standard security practices are analyzed.
- Intrusion Prevention is the process of detecting the signs of intrusion and attempting to stop the intrusive efforts.

- Collectively the system is known as Intrusion Detection and Prevention System (IDPS). IDPSs have become a necessary addition to the security infrastructure of nearly every organization. Intruders can be broadly divided into three different types:

Masquerader They are typically outsiders from the trusted users and are not authorized to use the computer systems. These intruders penetrate the system protection by way of legitimate user accounts.

Misfeasor They are typically insiders and legitimate users who accesses resources that they are not authorized to use. Or, they may be authorized but misuses her privileges.

Clandestine Users They can be both insiders and outsiders. These type of intruders gain supervisory access to the system.

Types of IDS Technologies

An Intrusion Detection System (IDS) is software that automates the intrusion detection process, while the Intrusion Prevention System (IPS) is software with all the properties of IDS, with the additional feature that it stops the intrusions.

The types of IDS technologies are differentiated primarily by the types of events that they monitor and the ways by which the features are achieved. Broadly there are four types of IDS technologies:

- **Network-based:** These monitor the network traffic for a segment of the network. It also analyzes the network and application protocol activity to identify suspicious activity.
- **Wireless:** These IDS monitor wireless network traffic. Its analysis is to identify suspicious activities involving the wireless protocols.
- **Network Behavior Analysis:** The network traffic is again analyzed to identify threats that create unusual traffic flows, Distributed Denial of Service (DDoS) attacks, and malwares and policy violations.
- **Host-based:** These IDS monitors the host and the events that occur within the host.

Usage of IDS

Apart from identifying suspicious incidents, there are some other usages of IDSs also:

- **Identifying Security Policy Problems** A IDS can provide amount of quality control for security policy implementations. This can include duplicating firewalls and also raising alerts when it sees that the network traffic is not blocked by configuration errors.
- **Documenting the Existing Threat to an Organization** They maintain logs about the threats that they detect.
- **Deterring Individuals from Violating Security Policies** The fact that the users are monitored by IDS makes them less likely to commit violations.
- **Preventive Actions of the IDS** The IDS uses several response techniques to prevent the attempts to perform security attacks on the systems. Some of the ways of how this can be achieved are by terminating the network connection or user sessions that [are being used in the attack, block access to the target from the offending accounts or IPS. The preventive step could be a drastic measure like blocking all accesses to the

Other important functions of the IPS involve the following:

- **The IDS Change the Security Environment** The IDS changes security environment to stop an attack. This could include reconfiguration of the network firewalls, application of patches onto the host computers which are suspected to have vulnerabilities.
- **The IDS can Change the Content of the Attack** The IDS often acts like a proxy which does normalization. That means they unpack the payloads of the request, remove the headers. This step nullifies certain attacks. They often remove malicious attachments from incoming files and pass the cleaned email to the recipient.

False Positives and Negatives

The IPS technology adopts statistical methods to comprehend the threats to the system. Thus, this has an accompanying attribute of false positives and negatives. They arise because of the fact that the IDPS cannot provide complete and accurate detection. The false alarms are defined as follows:

- **False Positive** When the IDPS incorrectly identifies a benign (harmless) activity as malicious, a false positive is said to have occurred.
- **False Negative** When the IDPS fails to identify a malicious activity, a false negative is said to have occurred.

It is not possible to eliminate both false positives and negatives, as reducing one of the errors has a consequence of increasing the other. It is intuitive that organizations prefer to reduce the false negatives at the cost of an increased false positive. Thus, more malicious events are detected. However, extra work needs to be done to be sure (or surer) that they are really malicious. Altering the configurations of an IDPS to improve the detection accuracies is technically known as tuning of the IDPS.

The malicious activities adopt several evasive techniques to bypass the IDPS. For example, the attack can encompass changing the encoding of characters, hoping that while the target will be able to understand the encoding, the IDPS would not. Thus, the IDPS needs to take measures to compensate these evasive techniques. The thumb rule is that if the IDPS is capable to see the activity the same way as the target, then most of the evasive techniques will fail.

Intrusion Detection Techniques

The classes of detection methodologies are: signature-based, anomaly-based and stateful protocol analysis.

Most of the IDPS uses a combination of these techniques to reduce the error of its detections.

Signature Based Detection

- A signature is a pattern that corresponds to a known threat.
- Signature based detection is the process of comparing the signature, which signifies a known threat against the events that are observed.

Examples of signatures could be a telnet attempt with root as the username. Or, an email with a subject name as "Free xyz" or an attachment, are other examples of signatures of malicious events.

Signature based detection schemes are simple methods which use string matching as the underlying technique. The current packet or log entry is matched to a list of signatures. Although simple and effective.

1. They are ineffective against unknown threats. Simply modifying the subject name to "Free xyzw"
2. They cannot pair a request with the corresponding response, like knowing that a request to a web server for a particular page generated a response status code of 403.
3. They cannot detect attacks that comprise multiple events if none of the events alone contains an indication of an attack. This limitation is because of the inherent incapability of this method to remember previous requests when processing the current one.

Anomaly Based Detection

- Anomaly based detection is the process of comparing definitions of activities which are supposed to be normal against observed events to identify deviations.
- An IDPS which uses anomaly-based detection techniques has profiles that represent the normal behaviors of users, hosts, network connections, or applications.

For example, a normal profile could include the fact that web activity is the most commonly done activity during day hours.

- The major benefit of anomaly detection is that they can be very effective in detecting previously unknown threats.
- For example, the power consumption of a computer may increase drastically compared to normal characteristic due to an infection from a malware.
- An initial profile is generated over a period of time, this period being called a training period. Profiles can be either static or dynamic. Static profiles are not changed for a long period of time, unless the IDPS is specifically directed to obtain a new profile.
- A dynamic profile on the other hand, constantly gets updated with additional events. Because of the inherent dynamic behavior of networks and systems, static profiles are not suitable as they get out-dated soon.
- Dynamic profiles do not suffer from this deficiency. But they suffer from the fact that the attackers can adopt evasive techniques to fool such an IDPS. For example, the attacker can slowly increase its activity.
- The IDPS may think that the rate of change is quite less and the small increase in activity may get included in the present profile.
- The malicious programs then further increase its activity and thus incrementally evade the IDPS.
- Another problem with IDPS technology implemented with anomaly detection methods have the problem of false positives.
- They often treat benign activities as malicious events.

For example, a system administrator's may include backup of large files. Such large file transfers may not be specified during the training period and thus this perfectly benign activity may raise an alarm.

It is often quite difficult to decide whether a raised alarm is false, due to the complexity and the number of events that have resulted in the alarm.

Stateful Protocol Analysis:

- This is the process of comparing predetermined profiles of generally accepted definitions of benign protocol activity for each protocol state against observed events to identify deviations.
- Unlike anomaly-based detection, which uses host or network specific profiles, stateful protocol analysis relies on vendor-developed universal profiles that specify how protocols should work.
- The "stateful" in this protocol analysis means that the IDPS is capable of checking networks, applications and application protocols that have the notion of state in them.

For example, the FTP (File Transfer Protocol) session can be visualized to consist of two states: unauthenticated and authenticated.

While the authenticated state can consist of several operations, there are few "benign" operations in the unauthenticated state, viz. providing user names and passwords and seeing help manuals.

Thus, the IDPS considers operations benign or malicious depending on the present state of the protocol.

Stateful protocol analysis can identify unexpected sequences of commands. Such suspicious sequences could be repeated issues of same commands or issuing of a command without first issuing a command upon which it is dependent on.

Because of the complexity involved in tracking the transitions of states for several sessions. Another serious problem is that these methods do not capture attacks that do not violate the characteristics of generally accepted protocol behavior. These may be several benign requests to create a denial of service attack.

Firewalls: Definition, construction, and working principles

The primary drawback of these analysis methods is that they are extremely resource intensive. This is because of the complexity involved in tracking the transitions of states for several sessions. Another serious problem is that these methods do not capture attacks that do not violate the characteristics of generally accepted protocol behavior. These may be several benign requests to create a denial of service attack.

LOS

FIREWALLS: DEFINITIONS, CONSTRUCTION, AND WORKING PRINCIPLES

A firewall is a single point of defense between two networks. It is an extremely important field of study with the tremendous growth of industry in the recent days. Access to Internet is a great source of information and fast transaction, which is necessary for the steep competition faced in the modern industrial era. However, internet access comes with a cautionary note. Not only does it bring information from the outside world to the insiders of a company, but it has the eerie fact of opening the inside information to the external world. Thus understanding and design of proper firewalls is extremely important.

A firewall can be simply a router that is used to filter the packets or a complex multi-computer, multi-router solution that performs filtering of packets along with application level proxy services. A firewall is essentially a router or a group of routers and computers to enforce access control between two networks. A firewall can be thought of as a pair of mechanisms: allow, which permits traffic and deny, which blocks traffic. There are some firewalls which emphasize on blocking traffic, while others emphasize on permitting traffic.

A standard corporate network topology has a hierarchy, often referred to as the security perimeters. Broadly the external perimeter of network and the internal network perimeter are separated by a DMZ (Demilitarized Zone). When information moves from the Internet to the internal world, integrity of data is a greater concern than the confidentiality of data. Suitable guards are enforced between the Internet and the DMZ and between the DMZ and the internal network to ensure that messages which can cause servers to function incorrectly or crash are not accepted. However when information travels from the internal network to the external world, both integrity and confidentiality are concerns. Guards placed have to check further that no confidential information is leaked and the data is not altered (or spoofed) while traveling from the internal computers to the Internet. If such changes are found in the packets being transmitted either while traveling into or out of the internal network, it is assumed that the network is attacked. Thus the job of the guards is reduced to allow or deny access to the external network of internal systems selectively. These guards are technically known as **firewalls** and are hence essential for the integrity and confidentiality of the information present in the internal network.

The first generation of firewall architectures were essentially packet filters and appeared first around 1985 and came out of Cisco's IOS software division. However the first paper describing the principles of packet filters was published in 1988 by Jeff Mogul from Digital Equipment Corporation. Since then there has been evolutions of the firewall industry. After packet filters, circuit level firewalls, application layer firewalls, dynamic firewalls etc were developed. Most firewalls technologies provide different capabilities of auditing communication events. The firewalls record the causes which lead to the firing of auditing events. With the evolution of the firewalls, they inspect additional network packet information, use more sophisticated inspection algorithms, maintain more state information, and inspect the network packets at more network layers. In the next sections we present an overview on the firewall architectures.

5-1 Packet Filters

A *packet filter* is one of the foremost firewall technologies that analyze network traffic at the transport protocol layer. Each IP network packet is examined to see if it matches one of a set of rules which defines the nature

of allowable data flow. These rules specify the allowable data flow and also the direction of data flow, i.e. internal to external network or vice-versa.

Following factors allow or deny the data flow through the packet filters:

1. The physical network interface (network adaptor) that the packet arrives on.
2. The address the data is coming from.
3. The address the data is going to.
4. The type of transport layer, TCP, UDP.
5. The transport layer source port.
6. The transport layer destination port.

Packet filters generally do not understand the application layer protocols used in the communication packets. The rules are instead kept in the TCP/IP kernel and applied to any packet. The actions taken may be either deny or permit the packet. For a network packet to be routed to its destination, it has to match with a permit list rule maintained in the kernel. If a packet matches with a deny rule, then it is dropped. However, if a packet does not match with either an allow rule or a deny rule, then also it is dropped. There are however some packet filters which allow packets, if it does not match with a deny rule.

Commands in packet filters check the source and destination port numbers on the TCP and UDP transport layer protocols. The combinations of ports and protocols which are to be allowed are mentioned in the allow list.

Because packet filters work in the network layer, they are unable to process protocols with state information. The packet filters do not also inspect the application layer data in the packets. This makes these filters the least secured. However they are the fastest firewall technologies and often a part of the IP routers. The packet filters often do network address translation so that the topology of the network and the addressing scheme of the network is hidden to untrusted or external network.

The advantages of the packet firewalls are summarized below:

❑ Advantages

1. Packet filters are faster than other techniques.
2. Less complicated, in the sense that a single rule controls deny or allow of packets.
3. They do not require client computers to be configured specially.
4. They shield the internal IP address from the external world.

❑ Disadvantages

1. Packet filters do not understand application layer protocols and hence cannot restrict access to FTP services, such as PUT and GET commands.
2. They are stateless, and hence not suitable for application layer protocols.
3. Packet filters have almost no audit event generation and alerting mechanisms.

Next we present an example of a packet filter. This example is written specifically for *ipfwadm*, which is an example of a cheap packet filtering tools. It is a kernel based tool on Linux. The principles (and even much of the syntax) can be applied for other kernel interfaces for packet filtering on open source Unix systems.

There are four basic categories covered by the *ipfwadm* rules:

- A
Packet Accounting
- I
Input firewall
- O

Output firewall

-F

Forwarding firewall

ipfwadm also has masquerading (-M) capabilities. For more information on the switches and options, see the *ipfwadm* manual page.

Imagine that the organization uses a private network 192.168.1.0. The Internet Service Provider has assigned the address 201.123.102.32 as the gateway and 201.123.102.33 as the mail server. The policies of the organization are as follows:

- To allow all outgoing TCP connections
- To allow incoming SMTP and DNS to external mail server
- To block all other traffic

The following block of commands can be placed in a system boot file (perhaps *rc.local* on Unix systems).

```
ipfwadm -F -f
ipfwadm -F -p deny
ipfwadm -F -i m -b -P tcp -S 0.0.0.0/0 1024:65535 -D 201.123.102.33 25
ipfwadm -F -i m -b -P tcp -S 0.0.0.0/0 1024:65535 -D 201.123.102.33 53
ipfwadm -F -i m -b -P udp -S 0.0.0.0/0 1024:65535 -D 201.123.102.33 53
ipfwadm -F -a m -S 192.168.1.0/24 -D 0.0.0.0/0 -W eth0
/sbin/route add -host 201.123.102.33 gw 192.168.1.2
```

The lines can be explained as follows: the first line flushes (-f) all forwarding (-F) rules. The second line sets the default policy (-p) to deny, while the line three to five are input rules (-i) in the following format:

ipfwadm -F (forward) *-i* (input) *m* (masq.) *-b* (bi-directional) *-P* (protocol)[protocol] *-S* (source)[subnet/mask] [originating ports] *-D* (destination)[subnet/mask] [port]

The sixth line appends (-a) a rule that permits all internal IP addresses output to all external addresses on all protocols, all ports. The seventh line adds a route so that the traffic going to 201.123.102.33 will be directed to the internal address 192.168.1.2.

In the next section we present the working principle of circuit level firewalls, which are second generation firewalls.

5.2 Circuit-Level Firewalls

Circuit-level firewalls are similar in operation to packet-filtering firewalls, but they operate at the transport and session layers of the OSI model. The biggest difference between a packet-filtering firewall and a circuit-level firewall is that a circuit-level firewall validates TCP and UDP sessions before opening a connection, or circuit, through the firewall. When the session is established, the firewall maintains a table of valid connections and lets data pass through when session information matches an entry in the table. The table entry is removed, and the circuit is closed when the session is terminated.

To validate a session, these kinds of firewall thus examine each connection set up to ensure that it follows a legitimate handshake for the transport layer protocol. TCP is a widely used protocol in the transport layer which uses handshake. The firewall maintains a virtual circuit table, which stores the connection details, namely the session state and the sequencing information, of the successful connections. When a connection is set up, the circuit level firewall typically stores the following:

1. A unique session identifier for the connection.
2. The state of the connection, namely handshake, established, or closing.
3. The sequencing information.

4. The source IP address, from where the data has arrived.
5. The destination IP address, where the data is to be delivered.
6. The physical network interface through which the data arrives.
7. The physical network interface, through which the packet goes out.

The circuit level firewall checks the header information contained in the network packet to see whether it has the necessary permissions to be transmitted. These firewalls have a limited understanding of the protocols used in network packets. They can only detect one transport layer protocol, TCP. Like packet filters, the rule sets are kept in the TCP kernel.

These firewalls perform a minimal security check compared to the application layer firewalls. Only those network connections that are associated with existing ones are allowed. But once a connection is allowed, all packets associated with the connection are allowed without further security checks. The method is hence fast and performs a limited amount of state checking, but is less secured. However, they perform limited checks to detect whether the packet data has been modified or spoofed. They check that the data contained in the transport protocol header complies with the definition of the corresponding protocol. Like packet filters, these firewalls also perform network address translation to hide the internal addresses from the external world.

We summarize the advantages and disadvantages of the circuit level firewalls:

❑ Advantages

1. They are faster than application layer firewalls.
2. They are more secured than packet filter firewalls.
3. They maintain limited state information of the protocols.
4. They protect against spoofing of packets.
5. They shield internal IP addresses from external networks by network address translation.

❑ Disadvantages

1. They cannot restrict access to protocol subsets other than TCP.
2. They have limited audit event generation capabilities.
3. They cannot perform security checks on higher level protocols.

5.3 Application-Layer Firewalls

An **application-layer firewall** is a third generation firewall technology that evaluates network packets for valid data at the application layer before allowing a connection. It examines the data in all network packets at the application layer and maintains a complete list of connection states and sequencing information. Further, other security items that appear only in the application layer protocols, like user passwords and service requests are validated.

Application layer firewalls use special purpose programs, called proxy services to manage data transfer through a firewall for a specific service such as *ftp* or *http*. Proxy services are thus dedicated to a particular protocol and provide increased security checks, access controls and generate appropriate audit records.

Proxy services do not allow direct connection between the real service and the user. They sit transparently between the user and the real server and handles and inspects every communication between them. A proxy service has two components that are typically implemented as a single executable: *proxy server* and *proxy client*.

When a real client wants to communicate to an external service in the internet, like *ftp* or *telnet*, the request is directed to the proxy server, because the user's default gateway is set to the proxy server. The proxy server then evaluates the request and decides to deny or allow it, depending on a set of rules that are managed for the network service. Proxy servers are aware of the protocols, and thus allow only complying packets with

the protocol definitions. They also perform auditing, user authentication and caching, services which were not performed by the packet filters or the circuit level firewalls.

On the other hand, once the packet from the real client is allowed by the proxy server the packet is forwarded to a proxy client who contacts the actual server providing the service. The proxy client subsequently relays back the information sent by the actual server to the proxy server, who decides whether to send the information to the actual client. Thus the proxy service is transparent to a user, who believes that he is communicating directly with the service in the Internet. However, the proxy services are implemented on the top of the firewall host's network stack and operate only in the application layer of the operating system. Hence each packet must pass through the low-level protocols in the kernel before being passed to the top of the stack to the application layer for a thorough analysis by the proxy services. Then the packet must travel back down the stack and then be distributed by the low level protocols in the kernel. Hence the application layer protocols are very slow.

The advantages and disadvantages of the application layer protocols are summarized below:

□ **Advantages**

1. They enforce and understand high level protocols, like HTTP and FTP.
2. They maintain information about the communication passing through the firewall server: partial communication derived state information, full application derived state information, partial session information.
3. They can be used to deny access to certain network services, while allowing others.
4. They are capable of processing and manipulating packet data.
5. They do not allow direct communication between external servers and internal systems, thus shielding internal IP addresses from the outside network.
6. They are transparent between the user and the external network.
7. They provide features like HTTP object caching, URL filtering, and user authentication.
8. They are good at generating auditing records, allowing administrators to monitor threats to the firewall.

□ **Disadvantages**

1. They require replacing the native network stack on the firewall server.
2. They do not allow network servers to run on the firewall servers, as the proxy servers use the same port to listen.
3. They are slow and thus lead to degradation in performance.
4. They are not scalable, as each new network service adds onto the number of proxy services required.
5. Proxy services require modifications to client procedures.
6. They rely on operating system support and thus are vulnerable to bugs in the system. Thus bugs in NDIS, TCP/IP, WinSock, Win32 or the standard C library can cause security concerns in the security provided by the application layer firewalls.

Dynamic packet filter firewalls are a fourth generation firewalls that allow modifications of the security rules on the fly. This technology is most suitable for providing limited support for the UDP transport protocol. This firewall associates all UDP packets that cross from the internal network to the external network or vice-versa, with a virtual connection. If a response packet is generated and sent back to the original requester, then a virtual connection is established and the packet is allowed to pass the firewall server. The information corresponding to a virtual connection is remembered for a small unit of time. If no response packet is received within this time frame then the virtual connection is invalidated. The response packet that is allowed back

must contain a destination address that matches the original source address, a transport layer destination port that matches the original source port, and the same transport layer protocol type. This feature is useful for allowing application layer protocols like Domain Name System (DNS). An internal DNS server must request other DNS servers on the Internet to obtain address information for unknown hosts. These connections may be made by TCP or UDP virtual connections.

Thus to summarize application layer firewalls are the most secured among the firewall technologies. They are more secure than the dynamic firewalls, which are more secure than the circuit level firewalls, which are in turn more secure than the packet level filters. However performance wise, the application layer filters are slowest. A point to be noted is that the circuit level firewalls are often faster than the packet level filters, as they do not perform extensive security checks, other than whether a network packet is associated with a valid connection in contrast to packet filters which have a large set of allow and deny rules.

An IDS (Intrusion Detection System) may only detect and warn about security violations. A firewall, on the other hand, may not notify a security violation but may simply block the attack or action violating the security policy of the firewall.

In practice, it is good to have both an IDS and a firewall, because the IDS warns us and a firewall blocks attacks on the systems security. Some firewall and IDS are combined into one security program, like Norton Internet Security.