CHAPTER 18

Security at the Network Layer: IPSec

Objectives

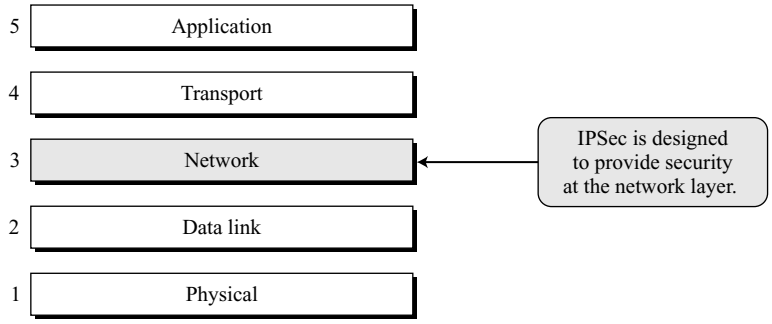
This chapter has several objectives:

- ❏ To define the architecture of IPSec
- ❏ To discuss the application of IPSec in transport and tunnel modes
- ❏ To discuss how IPSec can be used to provide only authentication
- ❏ To discuss how IPSec can be used to provide both confidentiality and authentication
- ❏ To define Security Association and explain how it is implemented for IPSec
- ❏ To define Internet Key Exchange and explain how it is used by IPSec

The two previous chapters have discussed the security at the application layer and transport layer. However, security at the above two layers may not be enough in some cases. First, not all client/server programs are protected at the application layer; for example, PGP and S/MIME protect only electronic mail. Second, not all client/server programs at the application layer use the service of TCP to be protected by SSL or TLS; some programs use the service of UDP. Third, many applications, such as routing protocols, directly use the service of IP; they need security services at the IP layer.

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*



IPSec can be useful in several areas. First, it can enhance the security of those client/server programs, such as electronic mail, that use their own security protocols. Second, it can enhance the security of those client/server programs, such as HTTP, that use the security services provided at the transport layer. It can provide security for those client/server programs that do not use the security services provided at the transport layer. It can provide security for node-to-node communication programs such as routing protocols.

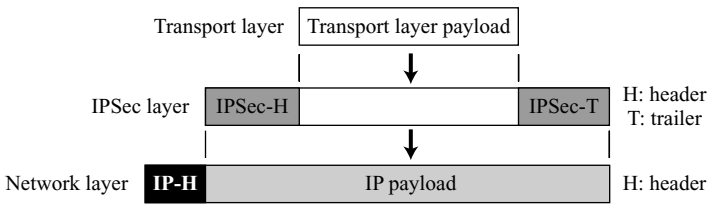
18.1 TWO MODES

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

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*

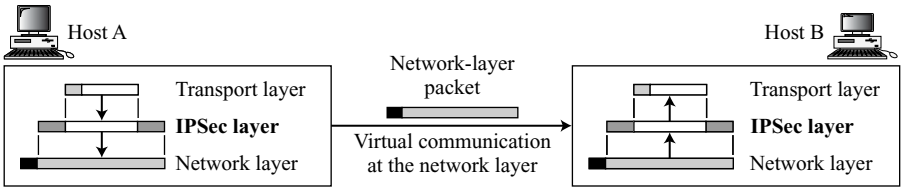


Note that transport mode does not protect the IP header. In other words, transport mode does not protect the whole IP packet; it protects only the packet from the transport layer (the IP layer payload). In this mode, the IPSec header (and trailer) are added to the information coming from the transport 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 and deliver it to the transport layer. Figure 18.3 shows this concept.

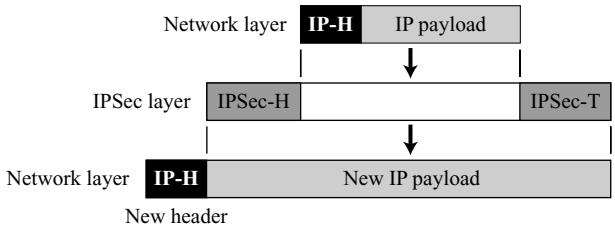
Figure 18.3 *Transport mode in action*



Tunnel Mode

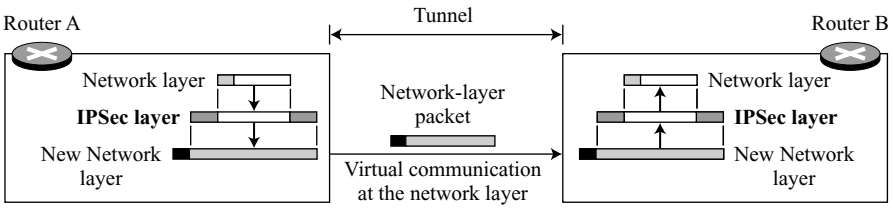
In **tunnel mode,** IPSec protects the entire IP packet. It takes an IP packet, including the header, applies IPSec security methods to the entire packet, and then adds a new IP header, as shown in Figure 18.4.

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. The entire original

Figure 18.5 *Tunnel mode in action*



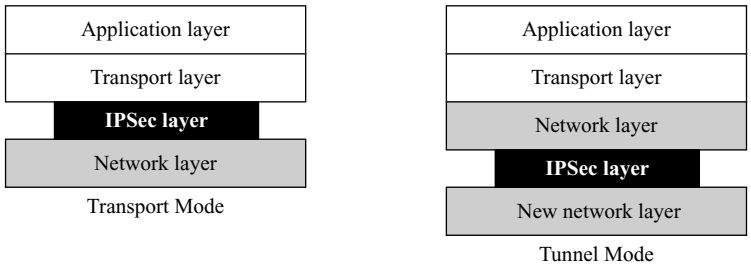
packet is protected from intrusion between the sender and the receiver, as if the whole packet goes through an imaginary tunnel.

IPSec in tunnel mode protects the original IP header.

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*



18.2 TWO SECURITY PROTOCOLS

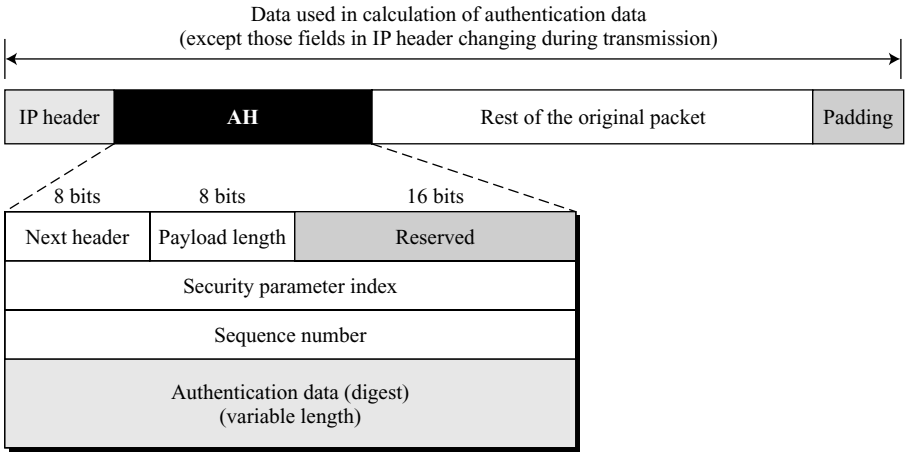
IPSec defines two protocols—the Authentication Header (AH) Protocol and the Encapsulating Security Payload (ESP) Protocol—to provide authentication and/or encryption for packets at the IP level.

Authentication Header (AH)

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.

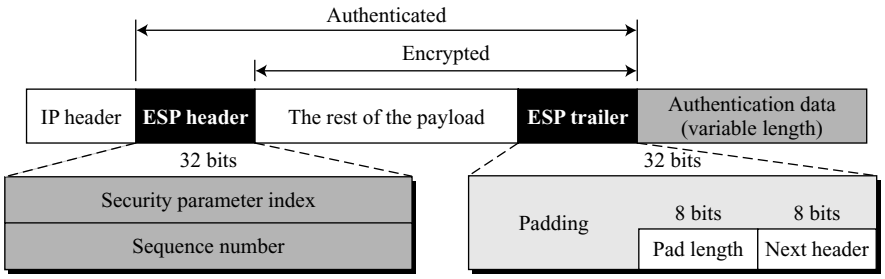
- **Payload length.** The name of this 8-bit field is misleading. It does not define the length of the payload; it defines the length of the authentication header in 4-byte multiples, but it does not include the first 8 bytes.
- **Security parameter index.** The 32-bit security parameter index (SPI) 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.

Encapsulating Security Payload (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.

Figure 18.8 *ESP*



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.
- **Padding.** This variable-length field (0 to 255 bytes) of 0s serves as padding.
- **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.
- **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.

IPv4 and IPv6

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

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.

Services Provided by IPSec

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

Table 18.1 *IPSec services*

Services	*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

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.

Entity Authentication

The Security Association and the keyed-hash digest of the data sent by the sender authenticate the sender of the data in both AH and ESP.

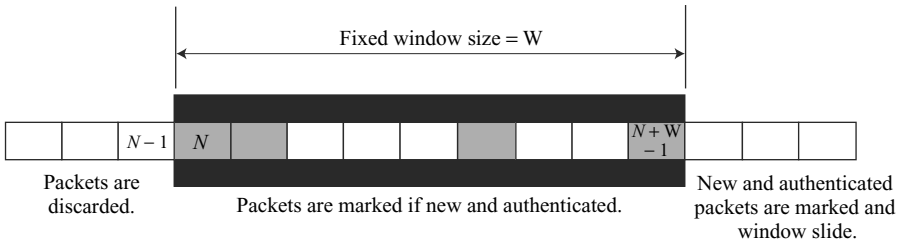
Confidentiality

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). When the sequence number reaches the maximum, it is reset to 0 and, at the same time, the old Security Association (see the next section) 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 or 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). In this case, the sliding of the window may cause many unmarked numbers to fall to the left 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.

18.3 SECURITY ASSOCIATION

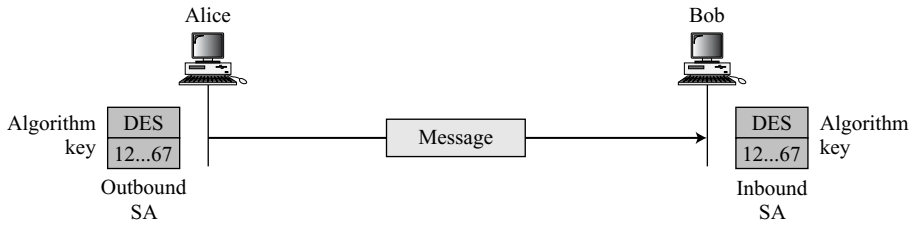
Security Association is a very important aspect of IPSec. IPSec requires a logical relationship, called a **Security Association (SA),** between two hosts. This section first discusses the idea and then shows how it is used in IPSec.

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 SA 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.

The Security Associations 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.

Figure 18.10 *Simple SA*



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. This 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.

Figure 18.11 *SAD*

< 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	ARW: Anti-Replay Window
P: Protocol	LT: Lifetime
Mode: IPSec Mode Flag	MTU: Path MTU (Maximum Transfer Unit)

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.

- **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 18.2.

Table 18.2 *Typical SA Parameters*

Parameters	*Descriptions*
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).

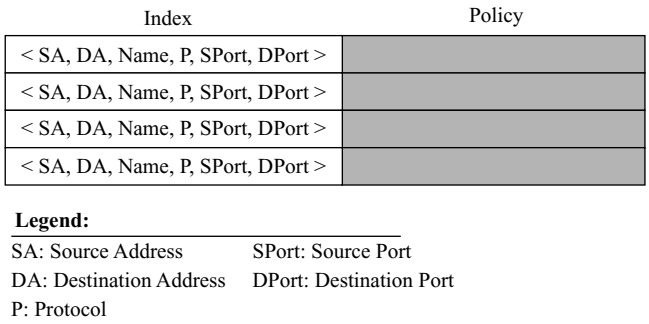
18.4 SECURITY POLICY

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.

Security Policy Database

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.

Figure 18.12 *SPD*



Source and destination addresses can be unicast, multicast, or wildcard addresses. The name usually defines a DNS entity. 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.

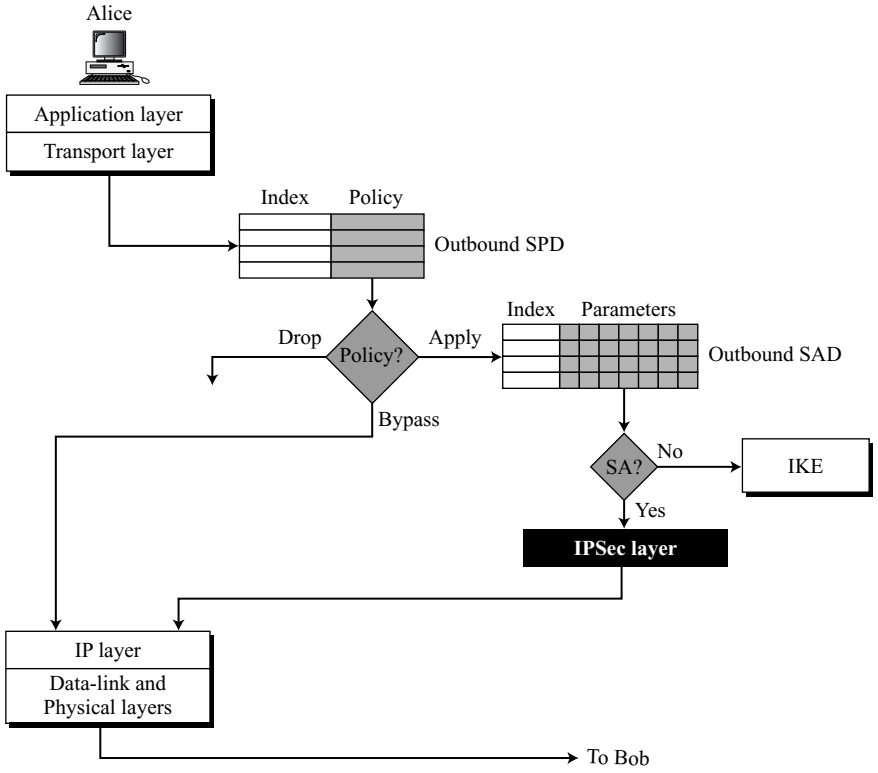
Outbound SPD

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:

1. **Drop.** This means that the packet defined by the index cannot be sent; it is dropped.
2. **Bypass.** This means that there is no policy for the packet with this policy index; the packet is sent, bypassing the security header application.

Figure 18.13 *Outbound processing*

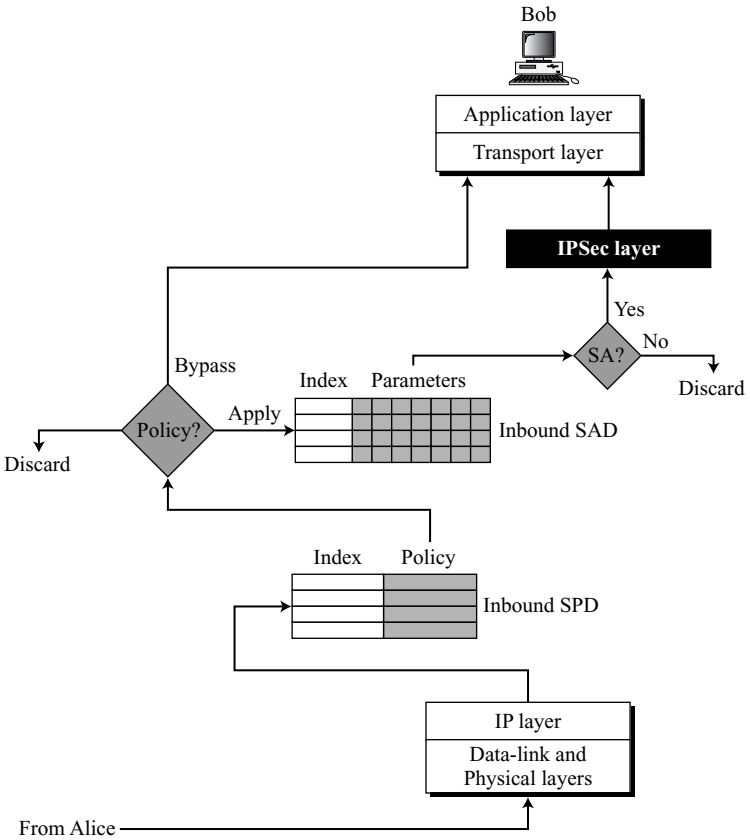


3. **Apply.** In this case, the security header is applied. Two situations may occur.
   a. If an outbound SA is already established, the triple SA index is returned that selects the corresponding SA from the outbound SAD. The AH or ESP header is formed; encryption, authentication, or both are applied 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.

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.

Figure 18.14 *Inbound processing*



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

1. **Discard.** This means that the packet defined by that policy must be dropped.
2. **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.
3. **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.

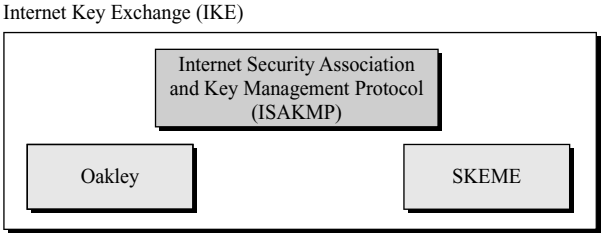
18.5 INTERNET KEY EXCHANGE (IKE)

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 was developed by Hilarie Orman. It is a key creation protocol based on the Diffie-Hellman key-exchange method, but with some improvements as we shall see shortly. Oakley is a free-formatted protocol in the sense that it does not define the format of the message to be exchanged. We do not discuss the Oakley protocol directly in this chapter, but we show how IKE uses its ideas.

SKEME, designed by Hugo Krawcyzk, 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 **Internet Security Association and Key Management Protocol (ISAKMP)** is a protocol designed by the National Security Agency (NSA) that actually implements the exchanges defined in IKE. It defines several packets, protocols, and parameters that allow the IKE exchanges to take place in standardized, formatted messages to create SAs. We will discuss ISAKMP in the next section as the carrier protocol that implements IKE.

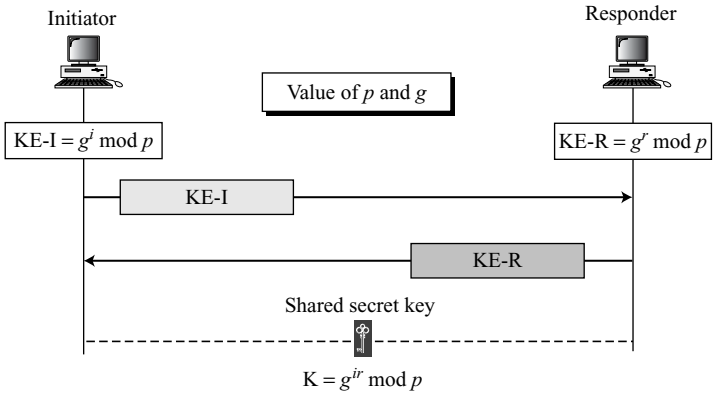
In this section, we discuss IKE itself; the mechanism for creating SAs for IPSec.

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 $<\mathbf{Z}_p^*, \times>$. 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.

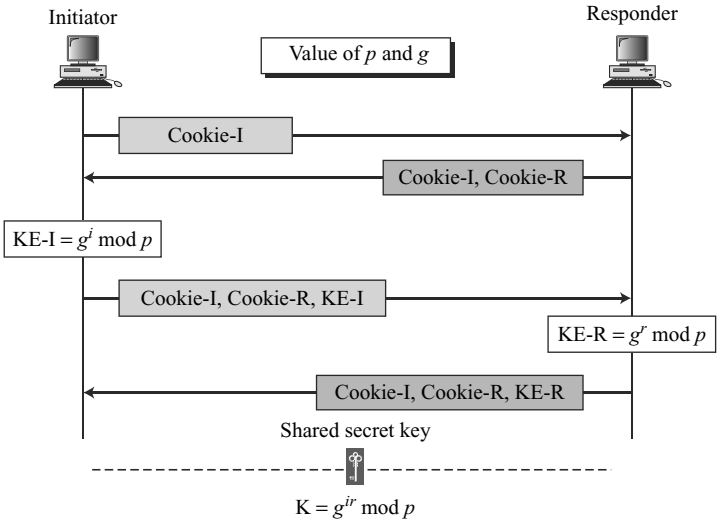
The Diffie-Hellman protocol has some weaknesses that need to be eliminated before it is suitable as an Internet key exchange.

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 ($g^x \bmod q$) messages to Bob, pretending that they are from different sources. Bob then needs to calculate different responses ($g^y \bmod q$) and at the same time calculate the full-key ($g^{xy} \bmod q$). 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.

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.

Figure 18.17 *Diffie-Hellman with cookies*



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) require 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.

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.

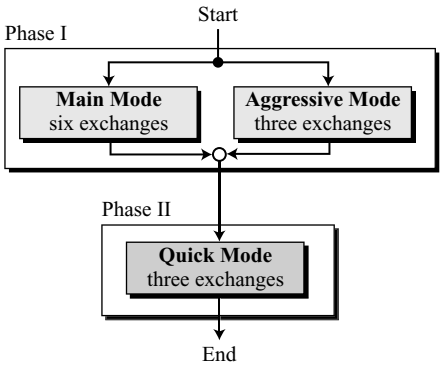
IKE is divided into two phases: phase I and phase II. Phase I creates SAs for phase II; phase II creates SAs for a data exchange protocol such as IPSec.

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.

Phases and Modes

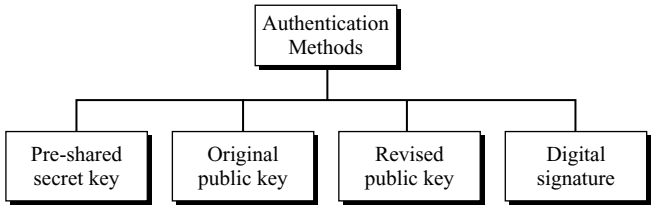
To allow for a variety of exchange methods, IKE has defined modes for the phases. So far, there are two modes for phase I: the *main mode* and the *aggressive mode*. The only mode for phase II is the *quick mode*. Figure 18.18 shows the relationship between phases and modes.

Figure 18.18 *IKE Phases*



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.

Figure 18.19 *Main-mode or aggressive-mode methods*



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^i 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.

$$\text{SKEYID} = \textbf{\textit{prf}}(\text{preshared-key}, \text{N-I} \mid \text{N-R}) \qquad \text{(preshared-key method)}$$

$$\text{SKEYID} = \textbf{\textit{prf}}(\text{N-I} \mid \text{N-R}, g^{ir}) \qquad \text{(public-key method)}$$

$$\text{SKEYID} = \textbf{\textit{prf}}(\text{hash (N-I} \mid \text{N-R), Cookie-I} \mid \text{Cookie-R)} \qquad \text{(digital signature)}$$

Other common secrets are calculated as follows:

$$\text{SKEYID\_d} = \textbf{\textit{prf}}(\text{SKEYID}, g^{ir} \mid \text{Cookie-I} \mid \text{Cookie-R} \mid 0)$$

$$\text{SKEYID\_a} = \textbf{\textit{prf}}(\text{SKEYID}, \text{SKEYID\_d} \mid g^{ir} \mid \text{Cookie-I} \mid \text{Cookie-R} \mid 1)$$

$$\text{SKEYID\_e} = \textbf{\textit{prf}}(\text{SKEYID}, \text{SKEYID\_a} \mid g^{ir} \mid \text{Cookie-I} \mid \text{Cookie-R} \mid 2)$$

SKEYID_d (derived key) is a key to create other keys. SKEYID_a is the authentication key and SKEYID_e is used for the encryption key; both are used during the negotiation phase. The first parameter (SKEYID) is calculated for each key-exchange method separately. The second parameter is a concatenation of various data. Note that the key for prf is always SKEYID.

The two parties also calculate two hash digests, HASH-I and HASH-R, which are used in three of the four methods in the main mode. The calculation is shown below:

$$\text{HASH-I} = \textbf{\textit{prf}}(\text{SKEYID}, \text{KE-I} \mid \text{KE-R} \mid \text{Cookie-I} \mid \text{Cookie-R} \mid \text{SA-I} \mid \text{ID-I})$$

$$\text{HASH-R} = \textbf{\textit{prf}}(\text{SKEYID}, \text{KE-I} \mid \text{KE-R} \mid \text{Cookie-I} \mid \text{Cookie-R} \mid \text{SA-I} \mid \text{ID-R})$$

Note that the first digest uses ID-I, while the second uses ID-R. Both use SA-I, the entire SA data sent by the initiator. None of them include the proposal selected by the responder. The idea is to protect the proposal sent by the initiator by preventing an intruder from making changes. For example, an intruder might try to send a list of proposals more vulnerable to attack. Similarly, if the SA is not included, an intruder might change the selected proposal to one more favorable to himself. Note also a party does not need to know the ID of the other party in the calculation of the HASHs.

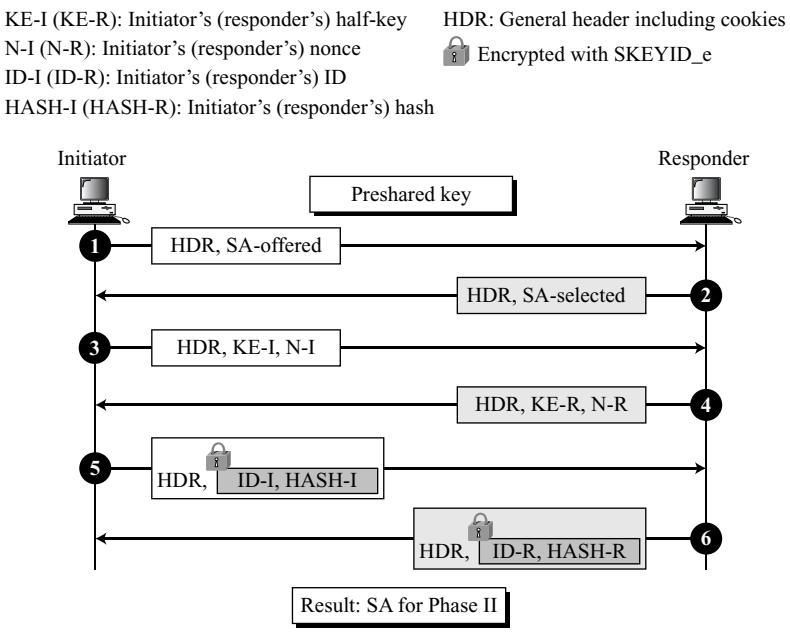
After calculating the keys and hashes, each party sends the hash to the other party to authenticate itself. The initiator sends HASH-I to the responder as proof that she is Alice. Only Alice knows the authentication secret and only she can calculate HASH-I. If the HASH-I then calculated by Bob matches the HASH-I sent by Alice, she is authenticated. In the same way, Bob can authenticate himself to Alice by sending HASH-R.

Note that there is a subtle point here. When Bob calculates HASH-I, he needs Alice's ID and vice versa. In some methods, the ID is sent by previous messages; in others it is sent with the hash, with both the hash and the ID encrypted by SKEYID_e.

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.

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.

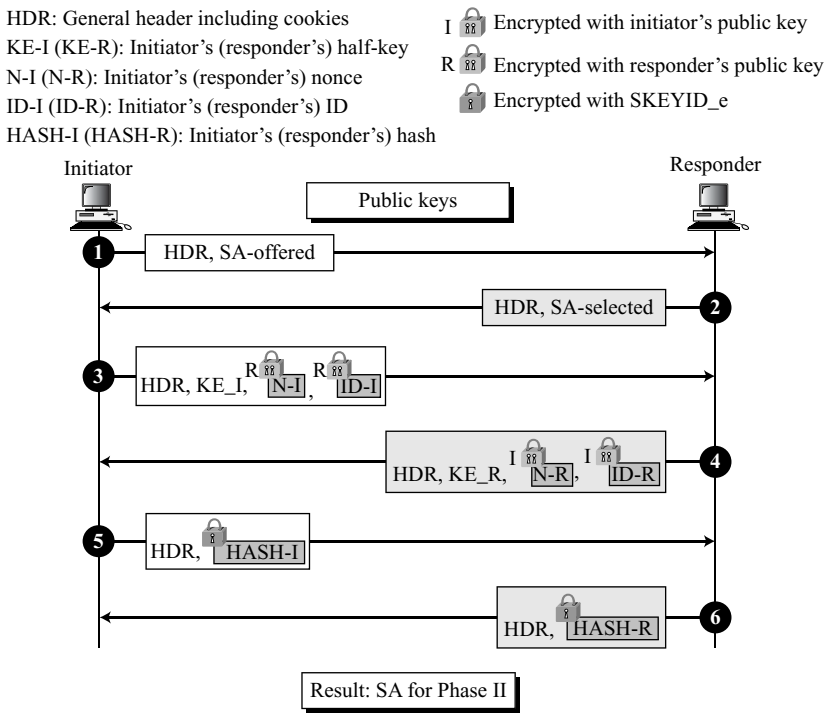
There is one problem with this method. Bob cannot decrypt the message unless he knows the preshared key, which means he must know who Alice is (know her ID). But Alice's ID is encrypted in message 5. The designer of this method has argued that the

ID in this case must be the IP address of each party. This is not an issue if Alice is on a stationary host (the IP address is fixed). However, if Alice is moving from one network to another, this is a problem.

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.

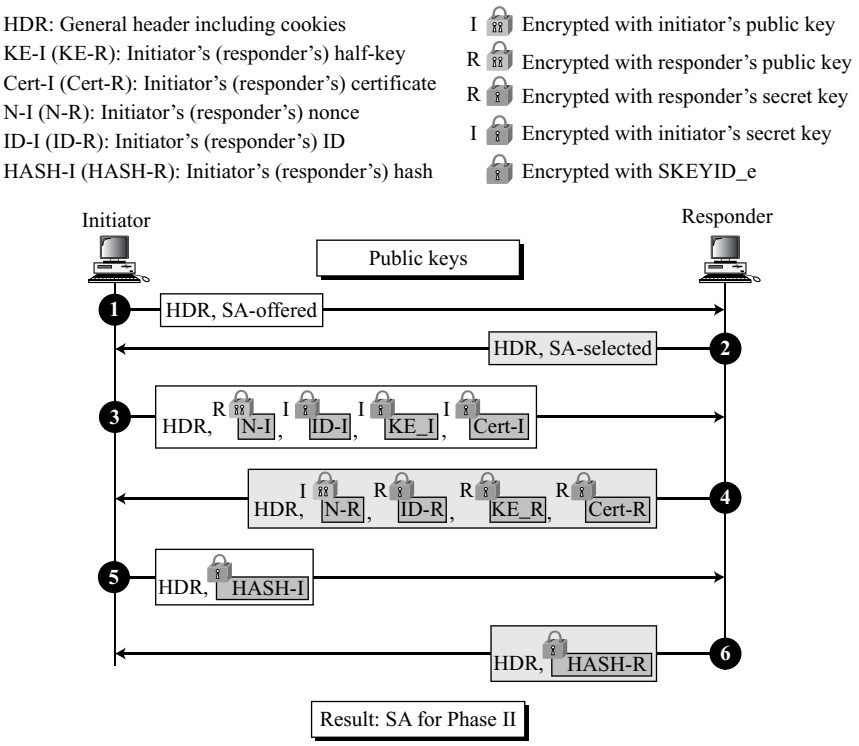
One difference between this method and the previous one is that the IDs are exchanged with the third and fourth messages instead of the fifth and sixth messages. The fifth and sixth messages just carry the HASHs.

The calculation of SKEYID in this method is based on a hash of the nonces and the symmetric key. The hash of the nonces is used as the key for the keyed-HMAC function. Note that here we use a double hash. Although SKEYID, and consequently, the hashes are not directly dependent on the secret that each party possesses, they are related indirectly. SKEYID depends on the nonces and the nonces can only be decrypted by the private key (secret) of the receiver. So if the calculated hashes match those received, it is proof that each party is who it claims to be.

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.

Figure 18.22 *Main mode, revised public-key method*



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

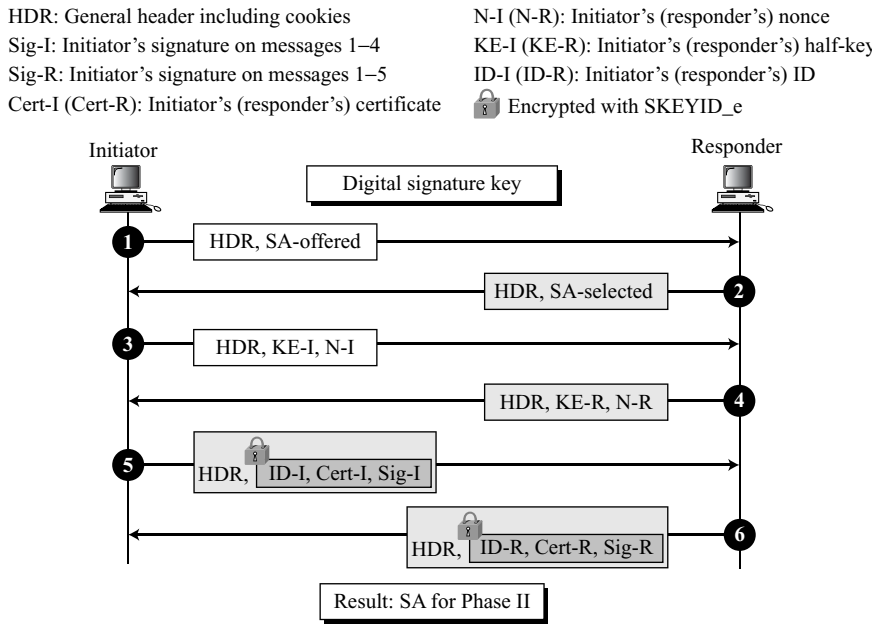
decrypts the nonce and calculates the initiator's temporary secret key. After that the half-key, the ID, and the optional certificate can be decrypted. The two temporary secret keys, K-I and K-R, are calculated as

$$\text{K-I} = \textit{prf}(\text{N-I, Cookie-I}) \qquad \text{K-R} = \textit{prf}(\text{N-R, Cookie-R})$$

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.

Figure 18.23 *Main mode, digital signature method*



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_e, 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.

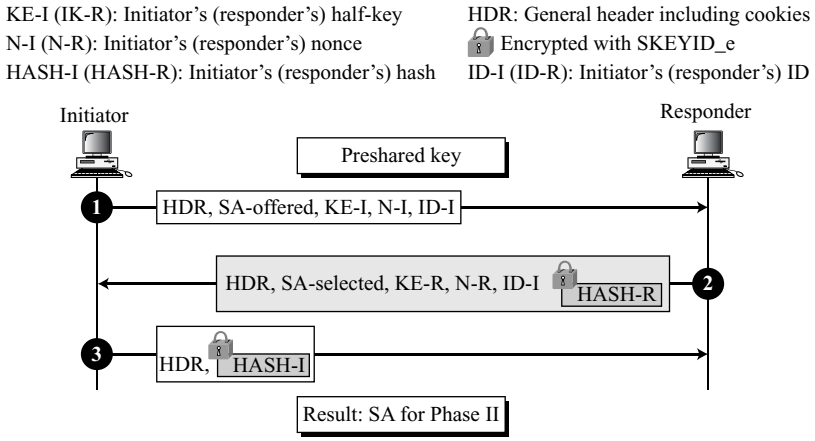
Phase I: Aggressive Mode

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.

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.

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 18.25 *Aggressive mode, original public-key method*

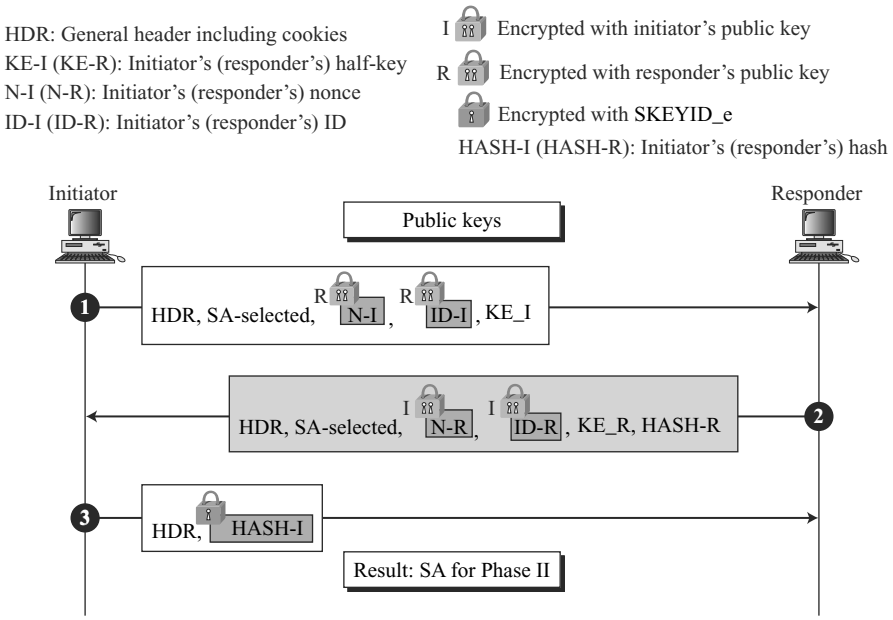


Figure 18.26 *Aggressive mode, revised public-key method*

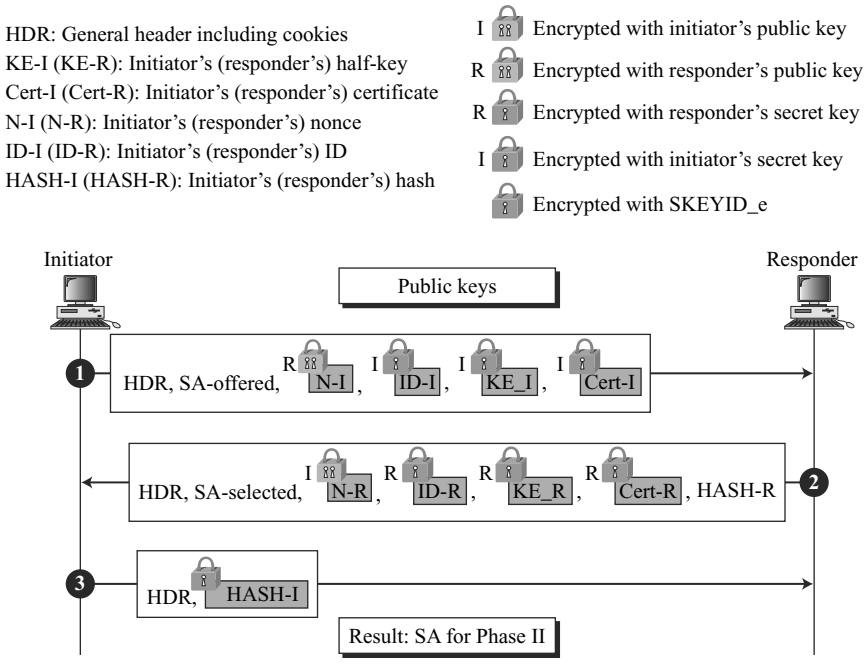
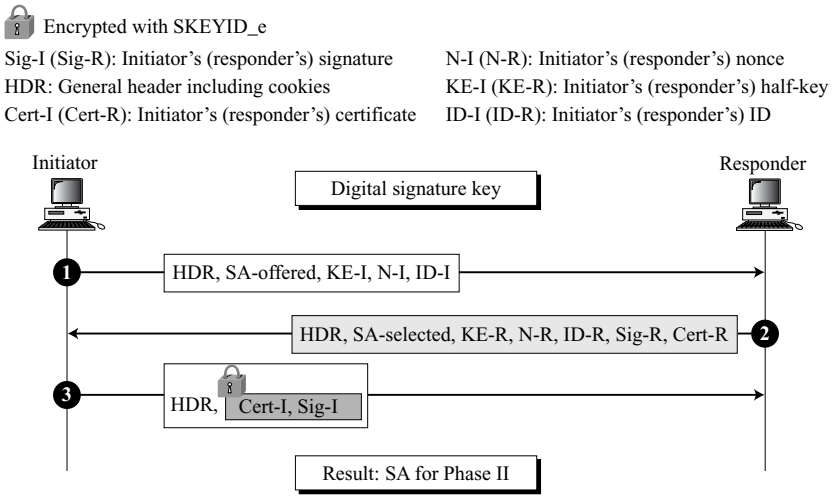


Figure 18.27 *Aggressive mode, digital signature method*

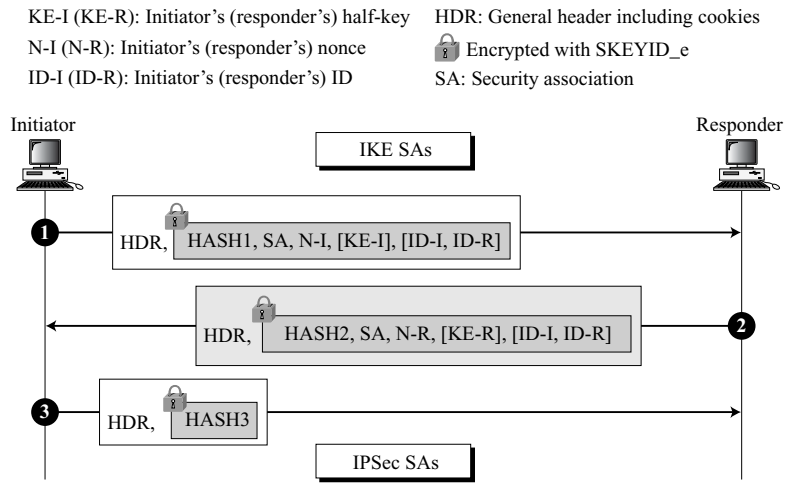


Phase II: Quick Mode

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.

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.

Figure 18.28 *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 messages are authenticated using three keyed-HMACs: HASH1, HASH2, and HASH3. These are calculated as follows:

$$\text{HASH1} = \textbf{\textit{prf}}(\text{SKEYID\_d}, \text{MsgID} \mid \text{SA} \mid \text{N-I})$$

$$\text{HASH2} = \textbf{\textit{prf}}(\text{SKEYID\_d}, \text{MsgID} \mid \text{SA} \mid \text{N-R})$$

$$\text{HASH3} = \textbf{\textit{prf}}(\text{SKEYID\_d}, 0 \mid \text{MsgID} \mid \text{SA} \mid \text{N-I} \mid \text{N-R})$$

Each HMAC includes the message ID (MsgID) used in the header of ISAKMP headers. This allows multiplexing in phase II. The inclusion of MsgID prevents simultaneous creations of phase II from bumping into each other.

All three messages are encrypted for confidentiality using the SKEYID_e created during phase I.

Perfect Forward Security (PFS)

After establishing an IKE SA and calculating SKEYID_d in phase I, all keys for the quick mode are derived from SKEYID_d. Since multiple phase IIs can be derived from a single phase I, phase II security is at risk if the intruder has access to SKEYID_d. To prevent this from happening, IKE allows **Perfect Forward Security (PFS)** as an option. In this option, an additional Diffie-Hellman half-key is exchanged and the resulting shared key (g^{ir}) is used in the calculation of key material (see the next section) for IPSec. PFS is effective if the Diffie-Hellman key is immediately deleted after the calculation of the key material for each quick mode.

Key Materials

After the exchanges in phase II, an SA for IPSec is created including the key material, K, that can be used in IPSec. The value is derived as:

$$\text{K} = \textbf{\textit{prf}}(\text{SKEYID\_d}, \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R}) \quad \text{(without PFS)}$$

$$\text{K} = \textbf{\textit{prf}}(\text{SKEYID\_d}, g^{ir} \mid \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R}) \quad \text{(with PFS)}$$

If the length of K is too short for the particular cipher selected, a sequence of keys is created, each key is derived from the previous one, and the keys are concatenated to

make a longer key. We show the case without PFS; we need to add g^{ir} for the case with PFS.

The key material created is unidirectional; each party creates different key material because the SPI used in each direction is different.

$$K_1 = \textbf{\textit{prf}}(\text{SKEYID\_d}, \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R})$$
$$K_2 = \textbf{\textit{prf}}(\text{SKEYID\_d}, K_1 \mid \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R})$$
$$K_3 = \textbf{\textit{prf}}(\text{SKEYID\_d}, K_2 \mid \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R})$$
$$\ldots$$
$$K = K_1 \mid K_2 \mid K_3 \mid \ldots$$

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

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.

Table 18.3 *Diffie-Hellman groups*

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

Hash Algorithms

The hash algorithms that are used for authentication are shown in Table 18.4.

Table 18.4 *Hash algorithms*

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

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.

Table 18.5 *Encryption algorithms*

Value	*Description*
1	DES
2	IDEA
3	Blowfish
4	RC5
5	3DES
6	CAST
7	AES

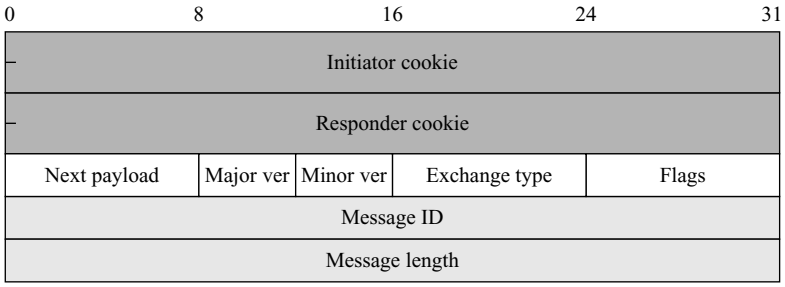
18.6 ISAKMP

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

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.

Payloads

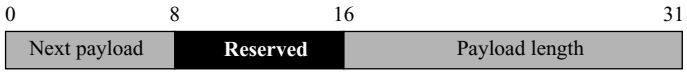
The payloads are actually designed to carry messages. Table 18.6 shows the types of payloads.

Table 18.6 *Payloads*

Types	*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 communication peers
6	Certification	Carries a public-key certificate
7	Certification 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	Nonce	Carries randomly generated data as a nonce
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-specification extensions

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*

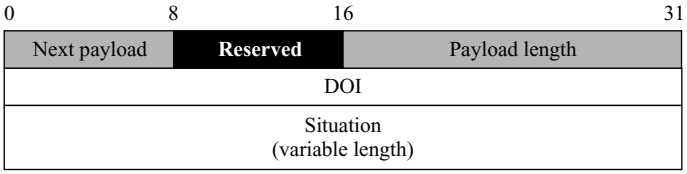


- **Next payload.** This 8-bit field identifies the type of the next payload. When there is no next payload, the value of this field is 0. Note that there is no type field for the current payload. The type of the current payload is determined by the previous payload or the general header (if the payload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



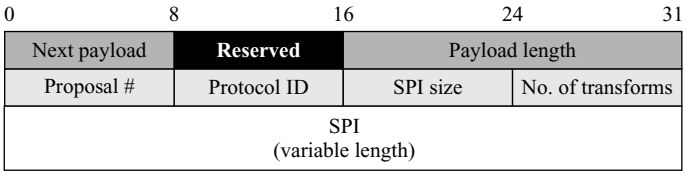
The fields in the generic header have been discussed. The descriptions of the other fields follow:

- **Domain of interpretation (DOI).** This is a 32-bit field. For phase I, a value of 0 for this field defines a generic SA; a value of 1 defines IPSec.
- **Situation.** This is a variable-length field that defines the situation under which the negotiation takes place.

Proposal Payload

The *proposal payload* initiates the mechanism of negotiation. Although by itself it does not propose any parameters, it does define the protocol identification and the SPI. The parameters for negotiation are sent in the transform payload that follows. Each proposal payload is followed by one or more transform payloads that give alternative sets of parameters. Figure 18.32 shows the format of the proposal payload.

Figure 18.32 *Proposal 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 phase1 = 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.

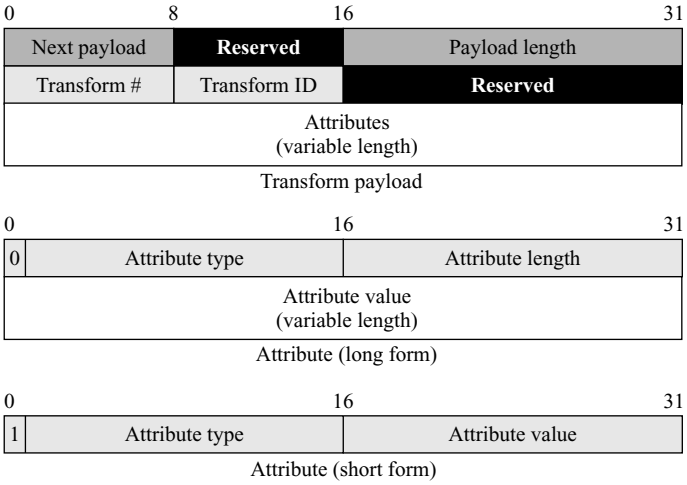
Transform Payload

The *transform payload* actually carries attributes of the SA negotiation. Figure 18.33 shows the format of the transform payload.

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.

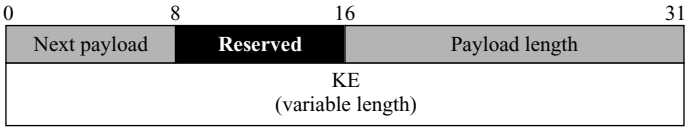
Figure 18.33 *Transform payload*



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*



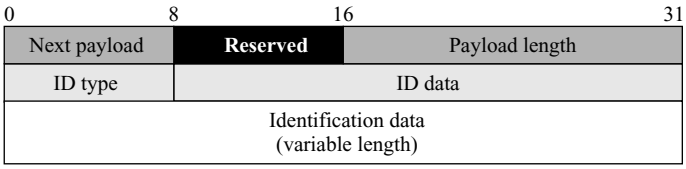
The fields in the generic header have been discussed. The description of the KE field follows:

- **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*



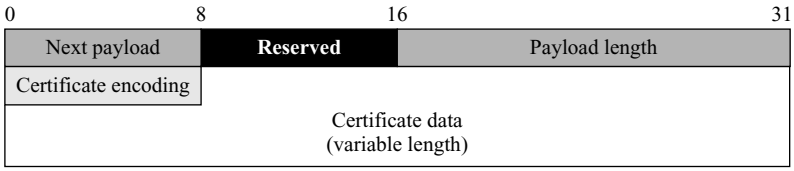
The fields in the generic header have been discussed. The descriptions of the other fields follow:

- **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.
- **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.

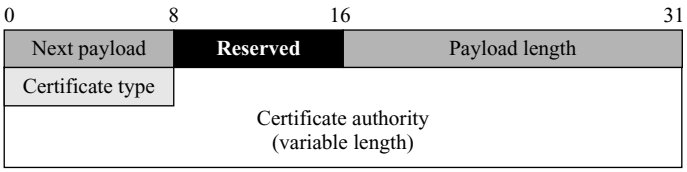
Table 18.7 *Certification types*

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

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*



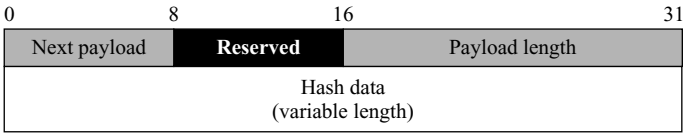
The fields in the generic header have been discussed. The descriptions of the other fields follow:

- **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*



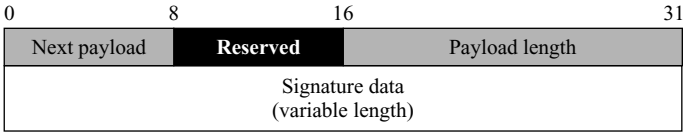
The fields in the generic header have been discussed. The description of the last field follows:

- **Hash data.** This 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*



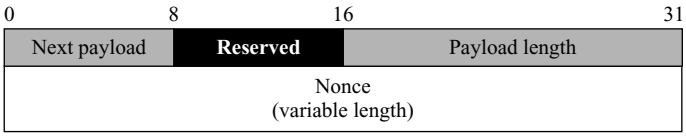
The fields in the generic header have been discussed. The description of the last field follows:

- **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 liveliness of the message and to prevent a replay attack. Figure 18.40 shows the format of the nonce payload.

Figure 18.40 *Nonce payload*



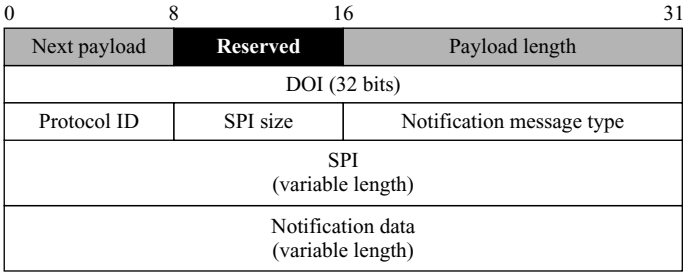
The fields in the generic header have been discussed. The description of the last field follows:

- **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. Table 18.8 gives a brief description of these types.
- **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. The types of errors are listed in Table 18.8. The values 31 to 8191 are for future use and the values 8192 to 16383 are for private use.

Table 18.8 *Notification types*

Value	*Description*	*Value*	*Description*
1	INVALID-PAYLOAD-TYPE	8	INVALID-FLAGS
2	DOI-NOT-SUPPORTED	9	INVALID-MESSAGE-ID
3	SITUATION-NOT-SUPPORTED	10	INVALID-PROTOCOL-ID
4	INVALID-COOKIE	11	INVALID-SPI
5	INVALID-MAJOR-VERSION	12	INVALID-TRANSFORM-ID
6	INVALID-MINOR-VERSION	13	ATTRIBUTE-NOT-SUPPORTED
7	INVALID-EXCHANGE-TYPE	14	NO-PROPOSAL-CHOSEN

Table 18.8 *Notification types (continued)*

Value	*Description*	*Value*	*Description*
15	BAD-PROPOSAL-SYNTAX	23	INVALID-HASH-INFORMATION
16	PAYLOAD-MALFORMED	24	AUTHENTICATION-FAILED
17	INVALID-KEY-INFORMATION	25	INVALID-SIGNATURE
18	INVALID-ID-INFORMATION	26	ADDRESS-NOTIFICATION
19	INVALID-CERT-ENCODING	27	NOTIFY-SA-LIFETIME
20	INVALID-CERTIFICATE	28	CERTIFICATE-UNAVAILABLE
21	CERT-TYPE-UNSUPPORTED	29	UNSUPPORTED EXCHANGE-TYPE
22	INVALID-CERT-AUTHORITY	30	UNEQUAL-PAYLOAD-LENGTHS

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.

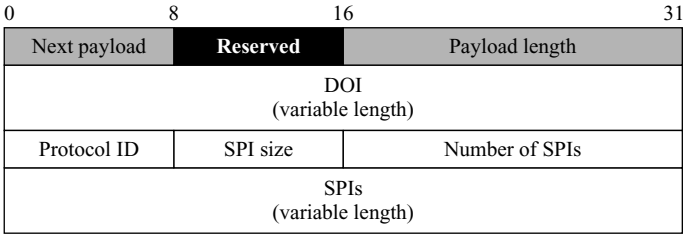
Table 18.9 *Status notification values*

Value	*Description*
16384	CONNECTED
24576-32767	DOI-specific codes

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*



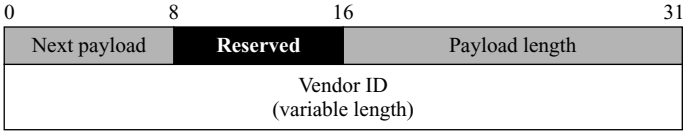
The fields in the generic header have been discussed. The descriptions of the other fields follow:

- **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.

18.7 RECOMMENDED READING

The following books and websites give more details about subjects discussed in this chapter. The items enclosed in brackets refer to the reference list at the end of the book.

Books

[DH03], [Fra01], [KPS02], [Res01], [Sta06], and [Rhe03] discuss IPSec thoroughly.

WebSites

The following websites give more information about topics discussed in this chapter.

http://www.ietf.org/rfc/rfc2401.txt
http://www.unixwiz.net/techtips/iguide-ipsec.html
http://rfc.net/rfc2411.html

18.8 KEY TERMS

aggressive mode
Authentication Header (AH) Protocol
clogging attack
cookie
Encapsulating Security Payload (ESP)
Internet Key Exchange (IKE)
Internet Security Association and Key Management Protocol (ISAKMP)
IP Security (IPSec)
main mode
Oakley
Perfect Forward Security (PFS)

replay attack
Security Association Database (SAD)
Security Association (SA)
Security Policy (SP)
Security Policy Database (SPD)
SKEME
transport mode
tunnel mode

18.9 SUMMARY

- IP Security (IPSec) is a collection of protocols designed by the IETF (Internet Engineering Task Force) to provide security for a packet at the network level.
- IPSec operates in transport or tunnel mode. In transport mode, IPSec protects information delivered from the transport layer to the network layer, but does not protect the IP header. In tunnel mode, IPSec protects the whole IP packet, including the original IP header.
- IPSec defines two protocols: Authentication Header (AH) Protocol and Encapsulating Security Payload (ESP) Protocol to provide authentication and encryption or both for packets at the IP level. The Authentication Header (AH) Protocol authenticates the source host and ensures the integrity of the payload carried by the IP packet. Encapsulating Security Payload (ESP) provides source authentication, integrity, and privacy. ESP adds a header and trailer.
- IPSec indirectly provides access control using a Security Association Database (SAD).
- In IPSec, Security Policy (SP) defines what type of security must be applied to a packet at the sender or at the receiver. IPSec uses a set of SPs called Security Policy Database (SPD).
- The Internet Key Exchange (IKE) is the protocol designed to create Security Associations, both inbound and outbound. IKE creates SAs for IPSec. IKE is a complex protocol based on three other protocols: Oakley, SKEME, and ISAKMP.
- IKE is designed in 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.
- The ISAKMP protocol is designed to carry the message for IKE exchange.

18.10 PRACTICE SET

Review Questions

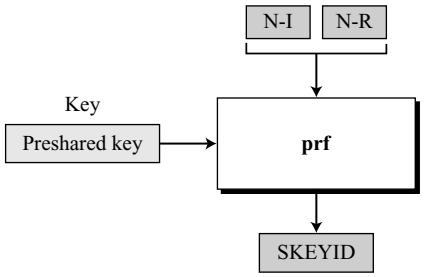
1. Distinguish between two modes of IPSec.
2. Define AH and the security services it provides.
3. Define ESP and the security services it provides.
4. Define Security Association (SA) and explain its purpose.
5. Define SAD and explain its relation to Security Association.

6. Define Security Policy and explain its purpose with relation to IPSec.
7. Define IKE and explain why it is needed in IPSec.
8. List phases of IKE and the goal of each phase.
9. Define ISAKMP and its relation to IKE.
10. List ISAKMP payload types and the purpose of each type.

Exercises

11. A host receives an authenticated packet with the sequence number 181. The replay window spans from 200 to 263. What will the host do with the packet? What is the window span after this event?
12. A host receives an authenticated packet with the sequence number 208. The replay window spans from 200 to 263. What will the host do with the packet? What is the window span after this event?
13. A host receives an authenticated packet with the sequence number 331. The replay window spans from 200 to 263. What will the host do with the packet? What is the window span after this event?
14. The diagram for calculation of SKEYID for the preshared-key method is shown in Figure 18.44. Note that the key to the prf function in this case is a preshared key.

Figure 18.44 *Exercise 14*



   a. Draw a similar diagram of SKEYID for the public-key method.
   b. Draw a similar diagram of SKEYID for the digital signature method.
15. Draw a diagram similar to Figure 18.44 for the following; the key in each case is SKEYID.
   a. SKEYID_a
   b. SKEYID_d
   c. SKEYID_e
16. Draw a diagram similar to Figure 18.44 for the following, the key in each case is SKEYID.
   a. HASH-I
   b. HASH-R

17. Draw a diagram similar to Figure 18.44 for the following; the key in each case is SKEYID_d:
    a. HASH1
    b. HASH2
    c. HASH3
18. Draw a diagram similar to Figure 18.44 for the following; the key in each case is SKEYID_d:
    a. K for the case without PFS
    b. K for the case with PFS
19. Repeat Exercise 18 for the case in which the length of K is too short.
20. Draw a diagram and show actual ISAKMP packets that are exchanged between an initiator and a responder using the *preshared-key* method in the *main* mode (see Figure 18.20). Use at least two proposal packets with at least two transform packets for each proposal.
21. Repeat Exercise 20 using the *original public-key* method in the *main* mode (see Figure 18.21).
22. Repeat Exercise 20 using the *revised public-key* method in the *main* mode (see Figure 18.22).
23. Repeat Exercise 20 using the *digital signature* method in the *main* mode (see Figure 18.23).
24. Repeat Exercise 20 in the *aggressive* mode (see Figure 18.24).
25. Repeat Exercise 21 in the *aggressive* mode (see Figure 18.25).
26. Repeat Exercise 22 in the *aggressive* mode (see Figure 18.26).
27. Repeat Exercise 23 in the *aggressive* mode (see Figure 18.27).
28. Draw a diagram and show the actual ISAKMP packets that are exchanged between an initiator and a responder in the *quick* mode (see Figure 18.28).
29. Compare the preshared-key methods in the main mode and aggressive modes. How much compromise is made in the aggressive mode with respect to security? What is the gain with respect to efficiency?
30. Compare the original public-key methods in the main and aggressive modes. How much compromise is made in the aggressive mode with respect to security? What is the gain with respect to efficiency?
31. Compare the revised public-key methods in the main and aggressive modes. How much compromise is made in the aggressive mode with respect to security? What is the gain with respect to efficiency?
32. Compare the digital signature method in the main and aggressive modes. How much compromise is made in aggressive mode with respect to security? What is the gain with respect to efficiency?
33. In the main and aggressive mode, we assume that an intruder cannot calculate the SKEYID. Give the reasoning behind this assumption.

34. In IKE phase I, the identity is usually defined as the IP address. In the preshared key method, the preshared key is also a function of the IP address. Show how this may create a vicious circle.
35. Compare methods for the main mode and show which method exchanges protected IDs.
36. Repeat Exercise 35 for aggressive methods.
37. Show how IKE reacts to the replay attack in the main mode. That is, show how IKE responds to an attacker that tries to replay one or more messages in the main mode.
38. Show how IKE reacts to the replay attack in the aggressive mode. That is, show how IKE responds to an attacker that tries to replay one or more messages in the aggressive mode.
39. Show how IKE reacts to the replay attack in the quick mode. That is, show how IKE responds to an attacker that tries to replay one or more messages in the quick mode.
40. Show how IPSec reacts to a brute-force attack. That is, can an intruder do an exhaustive computer search to find the encryption key for IPSec?