IPSec-1

G.P. Biswas

Prof/CSE, IIT, Dhanbad

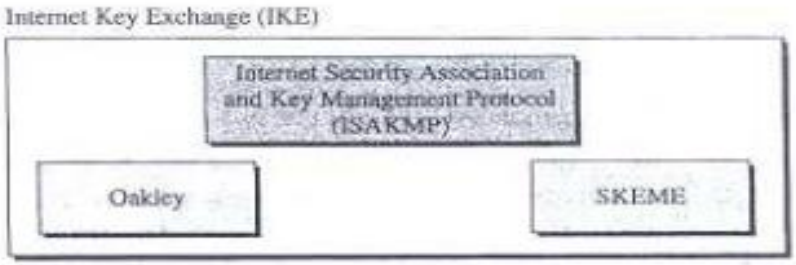
Internet Key Exchange (IKE) Protocol

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.

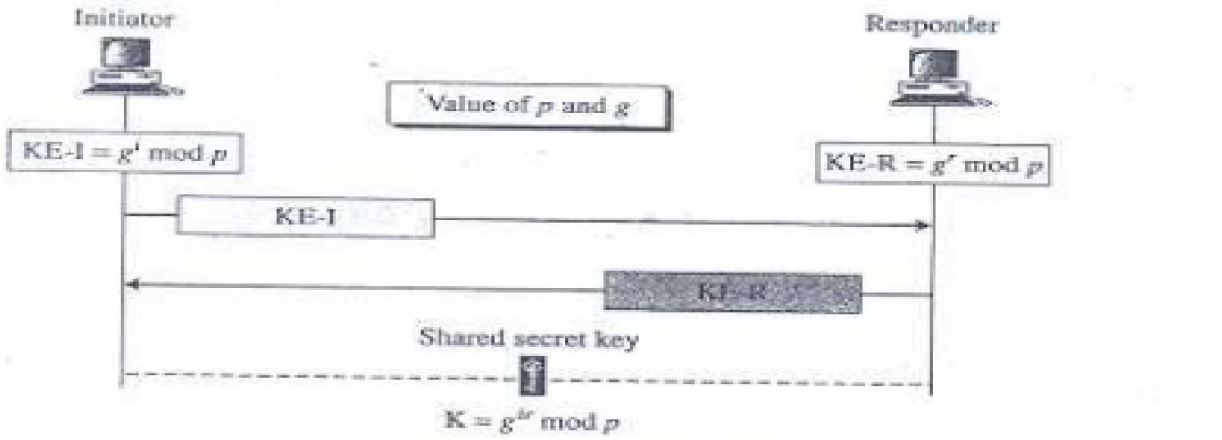
SKEME, designed by Hugo Krawcyzk, is another protocol for key exchar uses public-key encryption for entity authentication in a key-exchange protoc will see shortly that one of the methods used by IKE is based on SKEME.

The **Internet Security Association and Key Management Protocol (ISAK** a protocol designed by the National Security Agency (NSA) that actually implem exchanges defined in IKE. It defines several packets, protocols, and parameters tha the IKE exchanges to take place in standardized, formatted messages to create S will discuss ISAKMP in the next section as the carrier protocol that implements IK

Improved Diffie-Hellman Key Exchange Protocol

- The key-exchange idea in IKE is based o the Diffie-Hellman protocol. It provides a session key between two pairs without the need for the existence of any previous secret.

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

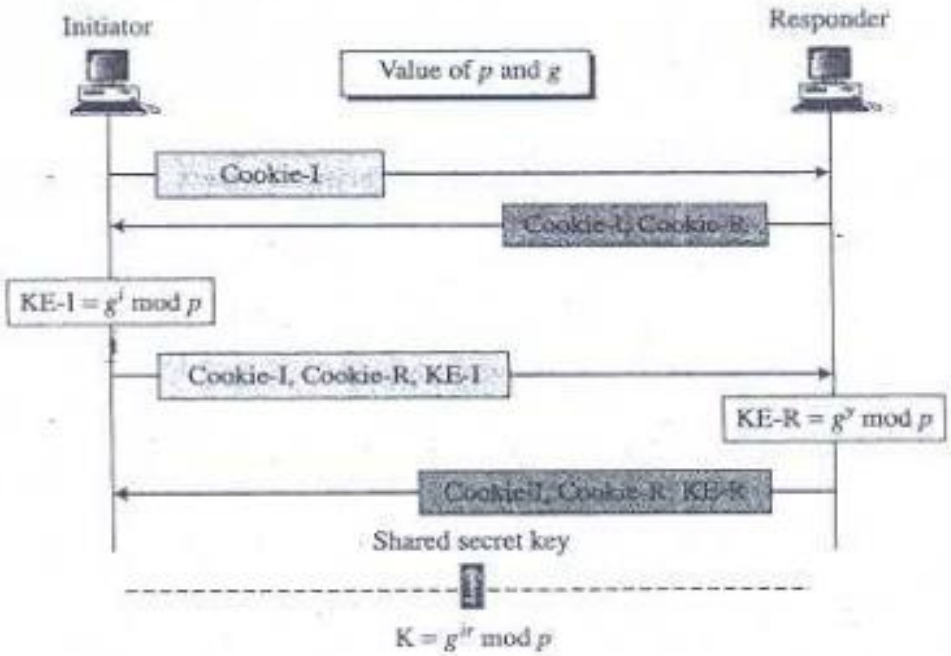
The Diffie-Hellman protocol has some weaknesses that need to be eliminated before it is suitable as an Internet key exchange

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 (Internet Key Exchange Protocol)

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.

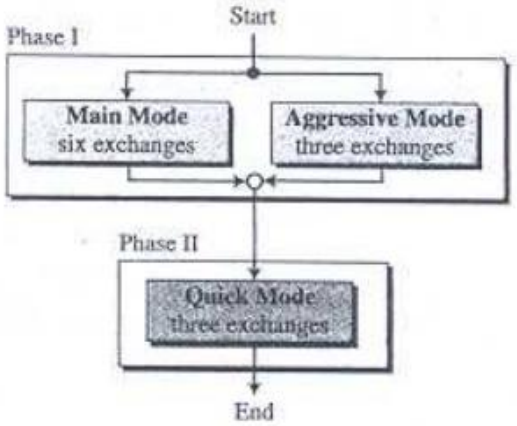
IKE is divided into two phases: phase I and phase II. Phase I creates SAs for phase II; phase II creates SAs for a data exchange protocol such as IPSec.

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

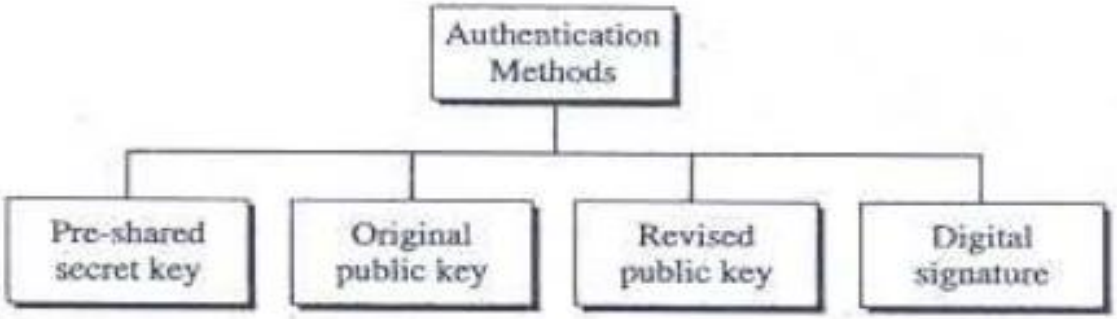
To allow for a variety of exchange methods, IKE has defined modes for the phases. So far, there are two modes for phase I: the *main mode* and the *aggressive mode*. The only mode for phase II is the *quick mode*. Figure 18.18 shows the relationship between phases and modes.

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.

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

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

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

Other common secrets are calculated as follows:

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

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

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

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:

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

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

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*

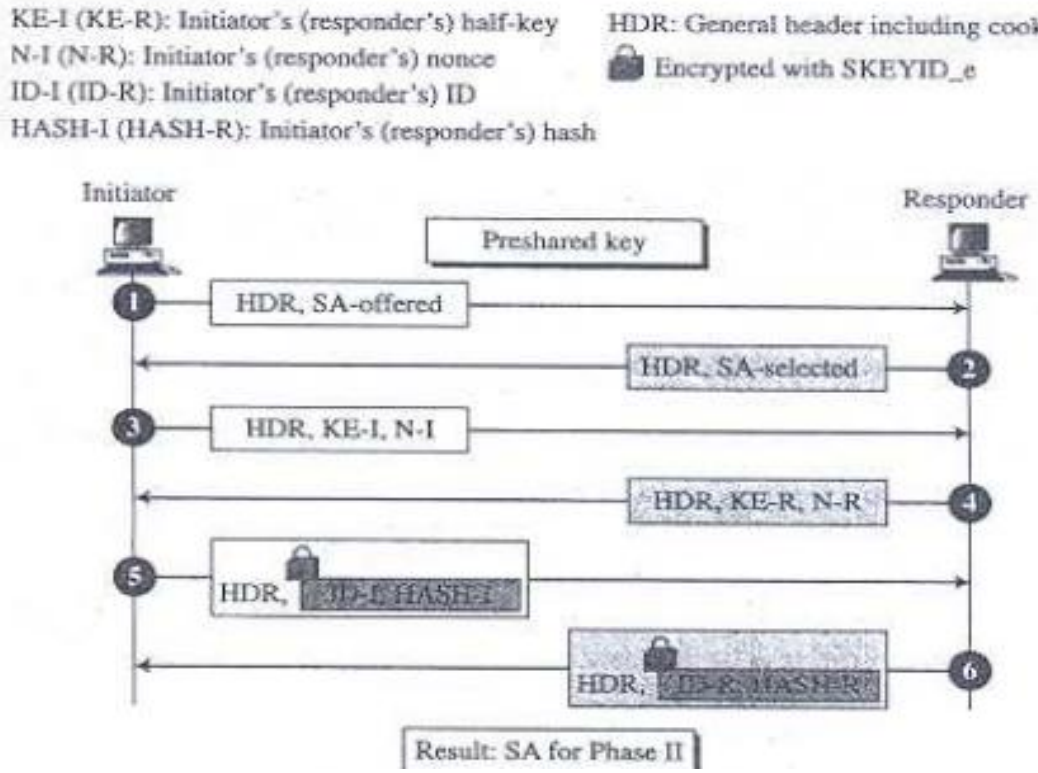
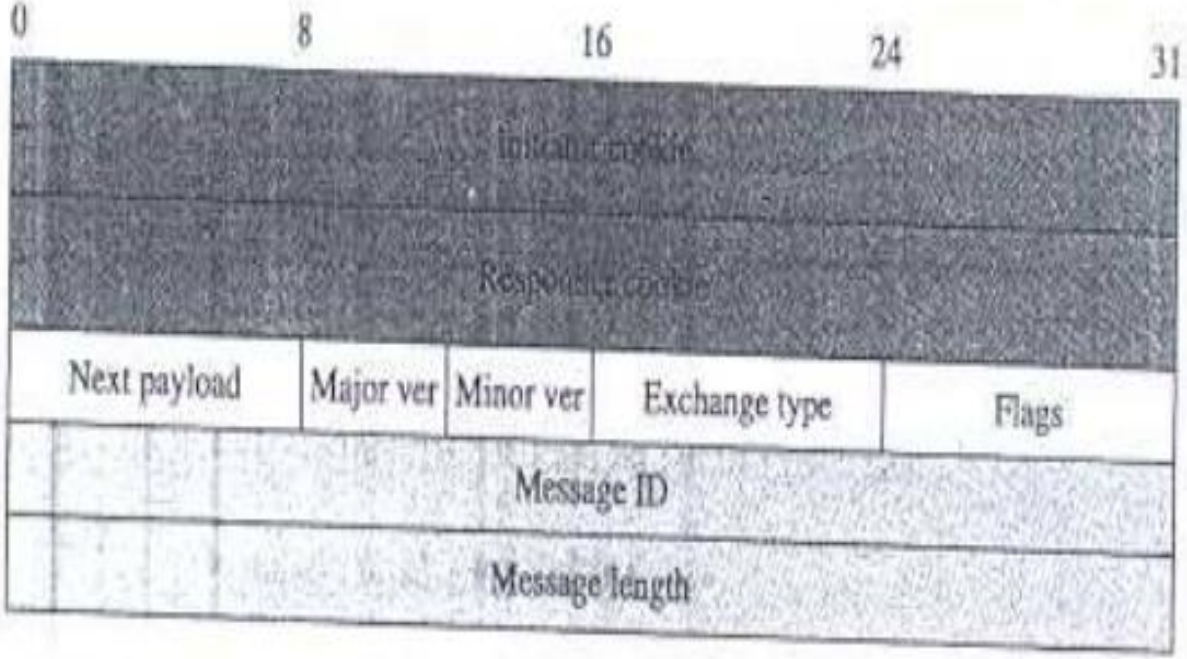


Figure 18.29 *ISAKMP general header*



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*