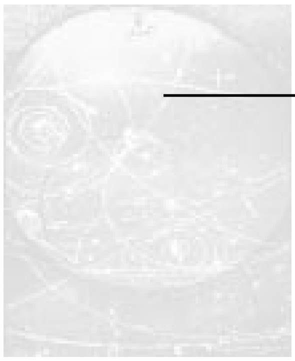

CHAPTER

3

IP Security Overview

IP Packets have no inherent security. It is relatively easy to forge the addresses of IP packets, modify the contents of IP packets, replay old packets, and inspect the contents of IP packets in transit. Therefore, there is no guarantee that IP datagrams received are (1) from the claimed sender (the source address in the IP header); (2) that they contain the original data that the sender placed in them; or (3) that the original data was not inspected by a third party while the packet was being sent from source to destination. IPSec is a method of protecting IP datagrams. This protection takes the form of data origin authentication, connectionless data integrity authentication, data content confidentiality, anti-replay protection, and limited traffic flow confidentiality.

IPSec provides a standard, robust, and extensible mechanism in which to provide security to IP and upper-layer protocols (e.g., UDP or TCP). A default, mandatory-to-implement suite of algorithms is defined to assure interoperability between different implementations, and it is relatively straightforward to add new algorithms without breaking interoperability.

IPSec protects IP datagrams by defining a method of specifying the traffic to protect, how that traffic is to be protected, and to whom the traffic is sent. IPSec can protect packets between hosts, between network security gateways (e.g., routers or firewalls), or between hosts and security gateways. Since an IPSec-protected datagram is, itself, just another IP packet, it is possible to nest security services and provide, for example, end-to-end authentication between hosts and send that IPSec-protected data through a tunnel which is, itself, protected by security gateways using IPSec.

The method of protecting IP datagrams or upper-layer protocols is by using one of the IPSec protocols, the Encapsulating Security Payload (ESP) or the Authentication Header (AH). AH provides proof-of-data origin on received packets, data integrity, and antireplay protection. ESP provides all that AH provides in addition to optional data confidentiality and limited traffic flow confidentiality. Since ESP provides all that AH provides, one may ask, "Why use AH?" That's a good question, and is the topic of debate in the security community. One subtle difference between the two is the scope of coverage of authentication. This will be discussed more fully in later chapters.

It should be noted that the ultimate security provided by AH or ESP is dependent on the cryptographic algorithms applied by them. Mandatory-to-implement algorithms are defined for conformance testing and to insure interoperability among implementations. These algorithms are generally secure, although recent advances in cryptography and the continued demonstration of Moore's law (the observation that every 18 months computing power doubles) make the default encryption algorithm, DES in CBC mode, not suited for highly sensitive data or for data which must remain secure for extended periods of time.

The security services that IPSec provides requires shared keys to perform authentication and/or confidentiality. A mechanism to manually add keys for these services is mandatory to implement. This ensures interoperability of the base IPSec protocols. Of course, manual key addition scales poorly so a standard method of dynamically authenticating IPSec peers, negotiating security services, and generating shared keys is defined. This key management protocol is called IKE—the Internet Key Exchange.

The shared keys used with IPSec are for either a symmetric cipher (when confidentiality is needed) or for a keyed MAC (for data integrity) or for both. IPSec must be fast and existing public key technologies, such as RSA or DSS, are too slow to operate on a packet-by-packet basis. Presently, public key technology is limited to initial authentication during key exchange.

The Architecture

The Architecture Document for IPSec, RFC2401, defines the base architecture upon which all implementations are built. It defines the security services provided by IPSec, how and where they can be used, how packets are constructed and processed, and the interaction of IPSec processing with policy.

The IPSec protocols—AH and ESP—can be used to protect either an entire IP payload or the upper-layer protocols of an IP payload. This distinction is handled by considering two different "modes" of IPSec (Figure 3.1). Transport mode is used to protect upper-layer protocols; tunnel mode is used to protect entire IP datagrams. In transport mode, an IPSec header is inserted between the IP header and the upper-layer protocol header; in tunnel mode the entire IP packet to be protected is encapsulated in another IP datagram and an IPSec header is inserted between the outer and inner IP headers. Both IPSec protocols, AH and ESP, can operate in either transport mode or tunnel mode.

Because of the method of construction, transport mode can only be used to protect packets where the communications endpoint is also the cryptographic endpoint. Tunnel mode may be used in place of transport mode, and in addition may be used by security gateways to provide security services on behalf of other networked entities (for example, a virtual private network). In this latter case, the communications endpoints are those specified in the inner header that's protected and the cryptographic endpoints are those of the outer IP header. A security gateway decapsu-

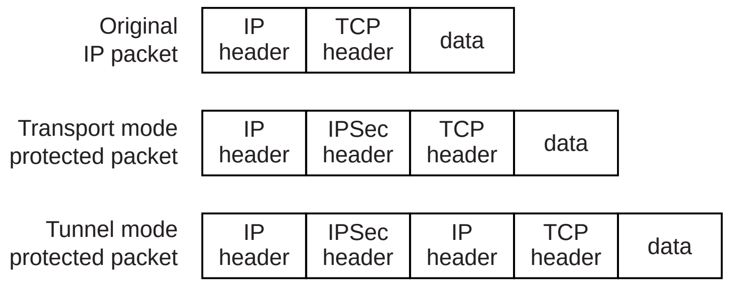

Figure 3.1 IP packets protected by IPSec in transport mode and tunnel mode

lates the inner IP packet upon the conclusion of IPSec processing and forwards the packet to its ultimate destination.

As noted, IPSec may be implemented in end systems or on security gateways such as routers and firewalls. Typically this is done by directly modifying the IP stack to support IPSec natively. When access to the IP stack of a machine is not possible, IPSec may be implemented as a “Bump in the Stack” (BITS) or “Bump in the Wire” (BITW). The former is typically a shim that extracts and inserts packets from the IP stack. The latter is typically an external, dedicated crypto device that may be independently addressable.

To properly encapsulate and decapsulate IPSec packets it is necessary to have a way to associate security services and a key, with the traffic to be protected, and the remote peer with whom IPSec traffic is being exchanged (in other words, how to protect the traffic, what traffic to protect, and with whom the protection is performed). Such a construct is called a “Security Association” (SA). An IPSec SA is unidirectional. That is, it defines security services for one direction, either inbound for packets received by the entity, or outbound, for packets that are secured and sent by the entity. They are identified by a Security Parameter Index (SPI)—which exists in IPSec protocol headers, the IPSec protocol value, and the destination address to which the SA applies—which dictates the direction. Typically, SAs exist in pairs, one in each direction. They may be created manually or dynamically. SAs reside in the Security Association Database (SADB).

When created manually, an SA has no lifetime. It exists until it is manually deleted. When created dynamically, an SA may have a lifetime associated with it. This lifetime is generally negotiated between the IPSec peers by the key management protocol. A lifetime is important because the amount of traffic protected by a key, or similarly the time that a key remains active and in use, must be carefully managed. Excessive use of a key can give an attacker an entry into your work.

The IPSec Architecture defines the granularity by which a user may specify his or her policy. This allows for certain traffic to be identified coarsely and have one level of security applied while allowing other traffic to be identified more finely and have a completely different level of security applied. For example, one may specify IPSec policy on a network security gateway that requires all traffic between its local protected subnet and the subnet of a remote peer be encrypted with DES and authenticated with HMAC-MD5, while all telnet traffic to a mail server from the remote subnet requires encryption with 3DES and authentication with HMAC-SHA, and all Web traffic to another server requires encryption with IDEA and authentication with HMAC-RIPEMD.

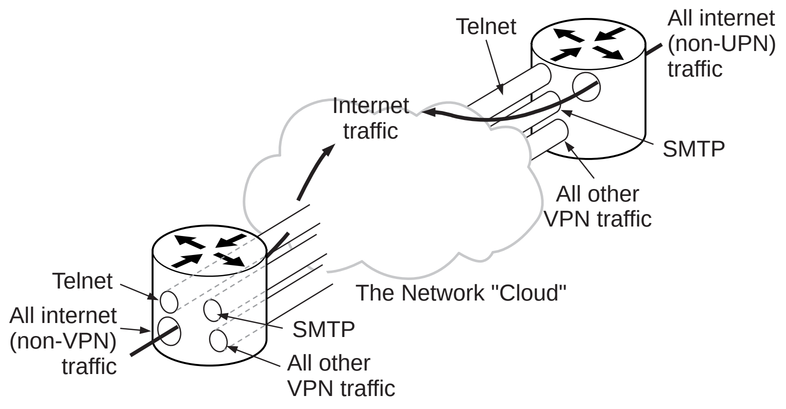

Figure 3.2 IPSec protected flows between separate networks

IPSec policy is maintained in the Security Policy Database (SPD). Each entry of the SPD defines the traffic to be protected, how to protect it, and with whom the protection is shared. For each packet entering or leaving the IP stack, the SPD must be consulted for the possible application of security. An SPD entry may define one of three actions to take upon traffic match: *discard*—do not let this packet in or out; *bypass*—do not apply security services to an outbound packet and do not expect security on an inbound packet; and *apply*—apply security services on outbound packets and require inbound packets to have security services applied. SPD entries that define an action of "apply" will point to an SA or bundle of SAs to apply to the packet.

IP traffic is mapped to IPSec policy by *selectors*. A selector identifies some component of traffic and may be either coarse or fine. IPSec selectors are: destination IP address; source IP address; name; upper-layer protocol; source and destination ports; and a data sensitivity level (if an IPSec system also provides for flow security). The values of these selectors may be specific entries, ranges, or "opaque." A selector in a policy specification may be opaque because that information may not be available to the system at that time. For example, a security gateway that has an IPSec tunnel with a remote security gateway peer may specify that (some of) the traffic that goes through that tunnel is IPSec traffic between two hosts behind the gateways. In this case, neither gateway would have access to, say, the upper-layer protocol or ports, since they would be encrypted by the end hosts. Opaque may also be used as a wild card, indicating the selector applies to any value.

If an SPD entry defines *apply* as an action and does not point to any existing SAs in the SADB, those SAs will have to be created before any traffic may pass. If this rule is for inbound traffic and the SA does not exist, the IPSec Architecture requires the packets to be dropped; if this rule is for outbound traffic the SAs can be created dynamically using the Internet Key Exchange (IKE).

The IPSec Architecture defines the interaction of the SPD, the SADB, with the IPSec processing functions—encapsulate and decapsulate—and defines how various IPSec implementations may exist. It does not, though, define how the base IPSec protocols operate. That is left for two different documents, one to define the Encapsulating Security Payload (RFC2406) and one to describe the Authentication Header (RFC2402).

Both IPSec protocols provide an antireplay service. This is not explicitly part of the architecture but is germane to both protocols and, as such, will be described here. IPSec packets are protected against replay attacks by using a sequence number and a sliding receive window. Each IPSec header contains a unique and monotonically increasing sequence number. When a SA is created, the sequence number is initialized to zero and prior to IPSec output processing the value is incremented. New SAs must be created prior to the sequence number wrapping around back to zero—prior to 2^{32} packets since the sequence number is 32 bits long. The receive window can be any size greater than 32 but 64 is recommended.

For performance reasons, the window size should be a multiple of the size of a word on the computer on which IPSec is being implemented.

The left end of the window represents the sequence number of the beginning of the window and the right end is *window-size* packets in the future. Received packets must be new and must fall either inside the window or to the right of the window, otherwise they are dropped. A packet is new if it has not yet been seen in the window. If a packet is received that is to the right of the window, it may be dropped if it fails an authenticity test (more on that later). If it passes the authenticity check the window is advanced, to the right, to encompass that packet. Note that packets may be received out of order and still be properly processed. Also note that a packet received late—that is, received after a valid packet with a sequence number greater than the size of the window—will be dropped.

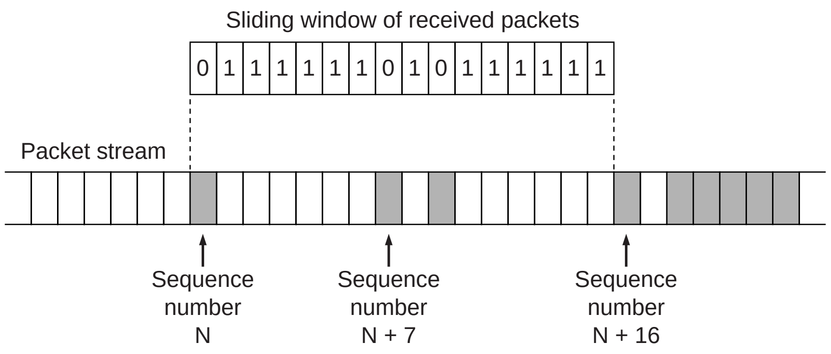

Figure 3.3 A 16 bit sliding replay window

The replay window in Figure 3.3 is only 16 bits and is therefore illegal, but for the sake of illustration will suit us fine. The left end of the window of Figure 3.3 is at sequence number *N*, the right end is therefore at sequence number *N+15*. Packets *N*, *N+7*, *N+9*, *N+16*, and *N+18* onward have not been received. If recently received packet *N+17* is authenticated the window is advanced such that the right end is at *N+17* and the left end is at *N+2*. This would cause packet *N* to be irretrievably lost since it's now to the left of the receive window. Notice, though, that packet *N+7* can still be received provided that packet *N+23* is not received and authenticated first.

It's important to note that the window must not be advanced until the packet that would cause its advancement has been authenticated. Doing otherwise would allow an attacker to generate bogus packets with large sequence numbers that would move the window outside the range of valid sequence numbers and cause us to drop valid packets.

Encapsulating Security Payload (ESP)

ESP is the IPSec protocol that provides confidentiality, data integrity, and data source authentication of IP packets, and also provides protection against replay attacks. It does so by inserting a new header—an ESP header—after an IP header (and any IP options) and before the data to be protected, either an upper-layer protocol or an entire IP datagram, and appending an ESP trailer. ESP is a new IP protocol and an ESP packet is identified by the protocol field of an IP header. If its value is 50 it's an ESP packet and immediately following the IP header is an ESP header. RFC2406 defines ESP.

An older version of ESP from RFC1827 did not provide data integrity and is considered deprecated by the IPSec Working Group. There were several implementations of RFC1827, but they are all being replaced by the current definition of ESP.

Since ESP provides both confidentiality and authentication, it has multiple algorithms defined in its SA—one for confidentiality called a *cipher*; and the other for authentication called the *authenticator*. Each ESP SA will have at most one cipher and one authenticator. It is possible to define NULL ciphers or NULL authenticators and do ESP without encryption or ESP without authentication respectively, but it is illegal to have both a NULL cipher and a NULL authenticator within a single ESP SA. This is illegal because not only is it a pointless burden on the system, it provides no security. One thing that cannot be mentioned enough is that doing something insecure with a security protocol is worse than not doing the security protocol in the first place because of the false sense of security provided. ESP provides security and it should not be used in a patently insecure manner.

The ESP header is not encrypted but a portion of the ESP trailer is. Enough is in clear text, though, to allow for a recipient to process the packet. Since the SPI is used, along with the destination IP address of the IP header of this packet, to identify an SA it must be in the clear. In addition, the sequence number and authentication data are also in the clear. This is due to the specified order of processing of ESP packets: First verify the sequence number, then verify the integrity of the data, then decrypt the data. Since decryption is last, the sequence number and authentication data must be in the clear.

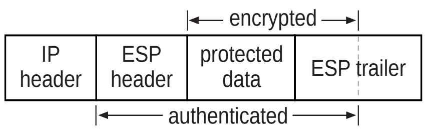

Figure 3.4 An ESP-protected IP packet

All encryption algorithms used with ESP must operate in cipher block chaining (CBC) mode. CBC requires that the amount of data to encrypt be a multiple of the block size of the cipher. This requirement is met by adding padding to the end of the data when necessary to encrypt. The pad becomes part of the cipher text of the packet and is stripped off by the recipient after IPSec processing. If the data is already a multiple of the block size of the cipher, padding need not be added. Compliant implementations are required to support the Data Encryption Standard (DES).

Ciphers in CBC mode also require an initialization vector (IV) to jump-start the encryption process. This IV is contained in the payload field generally as the first bytes, although it is up to a specific algorithm specification to define where it obtains its IV, and a size of the IV. For DES-CBC the IV is the first 8 bytes of the protected data field.

As mentioned, ESP is both a header and a trailer—it encapsulates the data it protects. The header portion contains the SPI and sequence number, the trailer contains the padding (if any), an indicator regarding the length of the pad, the next protocol of the data after ESP, and the authentication data. The size of the authentication data is dependent on the authenticator used. Compliant implementations are required to support both HMAC-MD5 and HMAC-SHA as authenticators with an output of 96 bits. You'll note that these two MACs produce different-sized digests though. HMAC-MD5 produces a 128-bit digest while HMAC-SHA produces a 160-bit digest. This is alright because the high-order 96 bits of the digest are used as ESP's authentication data. Ninety-six bits was chosen because it ensured alignment for IPv6.

There was some discussion about the security of truncating the output of a MAC. It has generally been agreed that such a practice is not inherently insecure and may, in fact, increase security. Regardless of the practice of the two required authenticators, new authenticators may be any length and padding is used to enforce alignment.

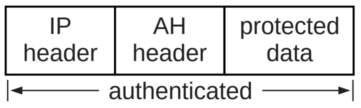

Figure 3.5 An AH-protected IP Packet

The ESP specification defines the format of the ESP header, where that header is placed when doing transport mode or tunnel mode, output data processing, input data processing, and other information such as fragmentation and reassembly. The ESP specification imposes requirements on transforms used with ESP but does not specify what those transforms are. That is left to individual transform specifications. Currently there is one document that describes using DES-CBC as the cipher for ESP, and two documents that describe using the truncated output of HMAC-MD5 and HMAC-SHA as the authenticators for ESP. Other cipher documents include Blowfish-CBC, CAST-CBC, and 3DES-CBC (all optional to implement).

Authentication Header (AH)

Like ESP, AH provides data integrity, data source authentication, and protection against replay attacks. It does not provide confidentiality. Because of this the AH header is much simpler than ESP; it is merely a header and not a header plus trailer. In addition, all of the fields in the AH header are in the clear.

RFC2402 defines the current incarnation of AH while RFC1826 described an older, deprecated version of AH. The important features of AH specified in that RFC remain in the new document—providing data integrity and data source authentication of IP packets—but new features and clarification of some issues raised with RFC1826 were added. For example, antireplay protection is now an integral part of the specification and a definition of using AH in tunnel mode was added. Just as there were several implementations of RFC1827, there were several of RFC1826—usually the same implementations. These deprecated transforms are being replaced by the new suite of IPSec RFCs.

The AH header, like the ESP header, contains an SPI to help locate the SA with which the packet is processed, a sequence number to provide against replay attacks, and an authentication data field to contain the digest from the keyed MAC used to secure the packet. Like ESP, the length of the digest field is defined by the particular transform used. Not too coincidentally, the default, mandatory-to-implement keyed MACs for AH are HMAC-MD5 and HMAC-SHA, both truncated to 96 bits. The same two documents, RFC2403 for HMAC-MD5-96 and RFC2404 for HMAC-SHA-96, used to define how to use these MACs with ESP, are used to define how to use them with AH.

Since AH does not provide confidentiality using a symmetric cipher in CBC mode, there is no explicit padding requirement imposed on it. Some MACs may require padding, for example DES-CBC-MAC, but the technique of addition of the pad is left to the document describing the MAC itself.

The authentication coverage of AH differs from that of ESP. AH authenticates the outer IP header of the IPSec packet. Therefore, the AH document describes the various fields of the IPv4 and IPv6 headers that are mutable—i.e., they may be changed by routers while the packet is in transit from source to destination. These fields must be zeroed prior to computation of the authentication data.

The AH document defines the format of the AH header, where that header is placed when doing transport mode or tunnel mode, output data processing, input data processing, and other information such as handling fragmentation and reassembly.

Internet Key Exchange

Security associations are used with IPSec to define the processing done on a specific IP packet. An outbound packet produces a hit in the SPD and the SPD entry points to one or more SAs—an SA bundle. If there is no SA that instantiates the policy from the SPD it is necessary to create one. That is where the Internet Key Exchange (IKE) comes into play. The whole purpose of IKE is to establish shared security parameters and authenticated keys—in other words, security associations—between IPSec peers.

The IKE protocol is a hybrid of the Oakley and SKEME protocols and operates inside a framework defined by ISAKMP—the Internet Security Association and Key Management Protocol. ISAKMP defines packet formats, retransmission timers, and message construction requirements, in effect, the language. Oakley and SKEME define the steps two peers must take to establish a shared, authenticated key. IKE uses the ISAKMP language to express these and other exchanges.

IKE is actually a general-purpose security exchange protocol and may be used for policy negotiation and establishment of authenticated keying material for a variety of needs—for example, SNMPv3, OSPFv2, etc. The specification of what IKE is being used for is done in a Domain of Interpretation (DOI). There exists a DOI for IPSec, RFC2407, which defines how IKE negotiates IPSec SAs. If and when IKE is used by other protocols, they will each have to define their own DOI.

IKE uses the concept of a security association but the physical construct of an IKE SA is different than an IPSec SA. The IKE SA defines the way in which the two peers communicate; for example, which algorithm to use to encrypt IKE traffic, how to authenticate the remote peer, etc. The IKE SA is then used to produce any number of IPSec SAs between the peers. Therefore, the action that an IPSec implementation takes when an SPD entry has a NULL SADB pointer is to communicate the security requirements from the SPD to IKE and instruct it to establish IPSec SAs.

The IPSec SAs established by IKE may optionally have perfect forward secrecy of the keys and, if desired, also of the peer identity. More than one pair of IPSec SAs may be created at once using a single IKE exchange, and any number of such exchanges may be performed by a single IKE SA. This richness of options makes IKE very extensible but also very complex.

The IKE protocol is performed by each party that will be performing IPSec; the IKE peer is also the IPSec peer. In other words, to create IPSec SAs with a remote entity you speak IKE to that entity not to a different IKE entity. The protocol is a request-response type with an *initiator* and a *responder*. The initiator is the party that is instructed by IPSec to establish some SAs as a result of an outbound packet matching an SPD entry; it initiates the protocol to the responder.

The SPD of IPSec is used to instruct IKE *what* to establish but does not instruct IKE *how* to do so. How IKE establishes the IPSec SAs is based on its own policy settings. IKE defines policy in terms of *protection suites*. Each protection suite must define at least the encryption algorithm, the hash algorithm, the Diffie-Hellman group, and the method of authentication used. IKE's policy database then is the list of all protection suites weighted in order of preference. Since the specific policy suite that the two peers agree upon will dictate how the remainder of their communication is done, this negotiation is the first thing the two IKE peers do.

There is more than one way for two peers to establish a shared secret, but IKE always uses a Diffie-Hellman exchange. The act of doing a Diffie-Hellman exchange is not negotiable, but the parameters to use are.

IKE borrows five groups from the Oakley document; three are traditional exchanges doing exponentiation modulo a large prime, and two are elliptic curve groups. The Diffie-Hellman exchange and the establishment of a shared secret is the second step of the IKE protocol.

Upon completion of the Diffie-Hellman exchange the two peers have a shared secret but it is not authenticated. They may use it—or in the case of IKE, a secret derived from it—to protect their communication, but they have no guarantee that the remote peer is, in fact, someone they trust. The next step in the IKE exchange is authentication of the Diffie-Hellman shared secret and, therefore, authentication of the IKE SA itself. There are five methods of authentication defined in IKE: preshared keys; digital signature using the Digital Signature Standard; digital signature using the RSA public key algorithm; an encrypted nonce exchange using RSA; and a "revised" method of authentication with encrypted nonces that is subtly different than the other encrypted nonce method. (A nonce is merely a random number. Each party in an IKE exchange contributes a nonce to the state of the exchange. This concept will be explained fully in Chapter 7.)

Creation of the IKE SA is referred to as phase one. Once phase one is completed, phase two—creation of IPSec SASs—may commence. There are two exchanges that can be performed for phase one, a Main Mode exchange or an Aggressive Mode exchange. Aggressive Mode is faster but Main Mode is more flexible. There is a single phase two exchange, Quick Mode. This exchange negotiates IPSec SAs under the protection of the IKE SA, which was created from a phase one exchange.

The keys used for the IPSec SAs are, by default, derived from the IKE secret state. Pseudo-random nonces are exchanged in Quick Mode and hashed with the secret state to generate keys and guarantee that all SAs have unique keys. All such keys do not have the property of perfect forward secrecy (PFS) since they're all derived from the same "root" key, the IKE shared secret. To provide PFS, Diffie-Hellman public values, and the group from which they're derived, are exchanged along with the nonces and IPSec SA negotiation parameters. The resultant secret is used to generate the IPSec SA keys to guarantee PFS.

To properly construct the IPSec SA, the initiator of the protocol must specify to IKE which selectors from his SPD matched the traffic. This information is exchanged in Quick Mode using identity payloads and is used to constrain what traffic can be protected by these SAs. At the time of this writing the selector suites in the IPSec Architecture Document was richer than that allowed by the IKE protocol. The IKE protocol cannot express port ranges, nor can it express the “all except” construct—for example, “all TCP ports greater than 1024 except 6000.” It is expected that the specifics of selector indication in Quick Mode exchanges will be changed to allow the full expression of possible selectors.

Upon completion of a Quick Mode the IKE SA returns to a quiescent state and awaits further instruction from IPSec or further communication from the peer. The IKE SA remains active until its lifetime expires or until some external event—such as an operator command to flush the database of IKE SAs— causes the SA to be deleted.

The first two messages in a phase one exchange (either Main Mode or Aggressive Mode) also exchange *cookies*. These resemble pseudo-random numbers but are actually temporal and bound to the peer’s IP address. Cookie creation is done by hashing together a unique secret, the peer’s identity, and a time-based counter. To the casual observer the result of this hash will be a random number, but the recipient of a cookie can quickly determine whether it generated the cookie or not by reconstructing the hash. This binds the cookie to the peer and provides for limited denial of service protection since the real work—the Diffie-Hellman exchange—is not performed until a complete round trip, and an exchange of cookies, has been accomplished.

It would be trivial to write a routine that constructed bogus IKE messages and sent them to a destination with a forged source address. If the responder did some work prior to having a strong belief that it is speaking to a genuine IKE peer and not an attacker forging packets it could easily be overwhelmed. Therefore, in Main Mode, the responder does not do any Diffie-Hellman work until he has received a second message from the initiator and has verified that message contains a cookie that he generated for the initiator.

Aggressive Mode does not have such a protection against denial of service attacks. The parties complete the exchange in three messages (as opposed to Main Mode’s six) and pass more information in each message. Upon receipt of the first Aggressive Mode message the responder must do a Diffie-Hellman exponentiation, this before he has had the chance to check the cookie of the next message that he receives (which is actually the last).

These cookies are used to identify the IKE SA. During a phase one exchange the IKE SA progresses from one state to the next upon processing of received messages and the sending of responses. The state advancement is one way. A phase two exchange is different. A phase two exchange is unique to itself. It is protected by the phase one IKE SA but has its own state. Therefore, it is entirely possible for two or more phase two exchanges to be simultaneously negotiated between the peers and under the protection of the same IKE SA. Each phase two exchange, therefore, creates a transient state machine to track the advancement of the protocol. When the exchange finishes, the state is thrown away. Since each of these transient state machines is protected by the same IKE SA, the messages of the exchanges all have the same cookie pair. An identifier unique to each phase two exchange is used to multiplex these exchanges into a single pipe. This identifier is called a Message ID.

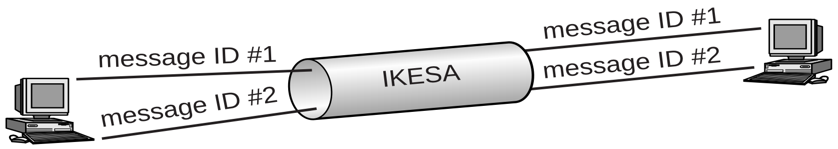

Figure 3.6 An IKE SA from Phase 1 protecting multiple Phase 2 exchanges

Periodically, it is necessary for an IKE process to send a message to his peer outside of any exchange. This could be to notify the peer that some IPSec SAs which it shares are being deleted, or it could be to report some error. Notification messages and delete messages are sent in another unique exchange called an Informational Exchange. This is a one-way message, no retransmission timer is set upon sending such a message, and no response is expected. These Informational exchanges are similar to a phase two exchange in that they're protected by an IKE SA but are unique and have their own state machine (actually a very simple state). Each Informational Exchange therefore has its own unique Message ID to allow it to be multiplexed with Quick Mode Exchanges and possibly other Informational Exchanges through a single IKE SA.

Implementation of a compliant IKE requires adherence to three documents: the base ISAKMP specification (RFC2408), the Domain of Interpretation for IPSec (RFC2407), and the IKE specification itself (RFC2409).