CHAPTER 7

IP Security Architecture

In this chapter, we discuss the following topics:

- What IPSec Does
- How IPSec Works
- Security Association
- Security Association Databases
  - Security Policy Database
  - Security Association Database

In the previous six chapters, our presentation was centered on the following subjects:

- TCP/IP protocol architecture
- Symmetric-key and public-key cryptographic algorithms, digital signature schemes, hash functions, and message authentication code
- Public-key infrastructure (PKI)
- Lightweight Directory Access Protocol (LDAP)

Each of these topics is relevant to a discourse on IPSec. In covering these subject areas in the depth that we did, we laid the foundation for a sound discussion on the actual IPSec protocols. We start by first giving an overview of IPSec.

7.1 What IPSec Does

IP packets are inherently insecure. It is relatively easy to forge IP addresses, modify the contents of IP packets, replay old content, and inspect the content of packets in transit. Therefore, there is no guarantee that an IP datagram originated from the source it claims to originate from, that it will get to the intended destination, or that the contents have not been modified or examined by a third party while it was in transit from the source to the destination.

In Chapter 1, in our presentation on the IP protocol, we mentioned briefly how the 16-bit header checksum is utilized in an effort to validate the integrity of an IP header. To illustrate how relatively easy it is to forge an IP header, let us take a more detailed look at how an IP header checksum is calculated and used to rudimentarily ascertain the integrity of an IP header. Figure 7-1 is the figure we presented in Chapter 1, illustrating an IPv4 header.

The IP header checksum is calculated over the IP header only. It does not include the data field since all the transport protocols (TCP, UDP, ICMP, and IGMP) have a checksum in their header to check their data. To compute the checksum for an outgoing datagram, the 16-bit checksum field is first set to zero, and then the one's complement sum of the header is computed; that is, the entire IP header, excluding the data field, is considered as a sequence 16-bit words, and the one's complement sum of these 16-bit sequences is computed and stored in the checksum field.

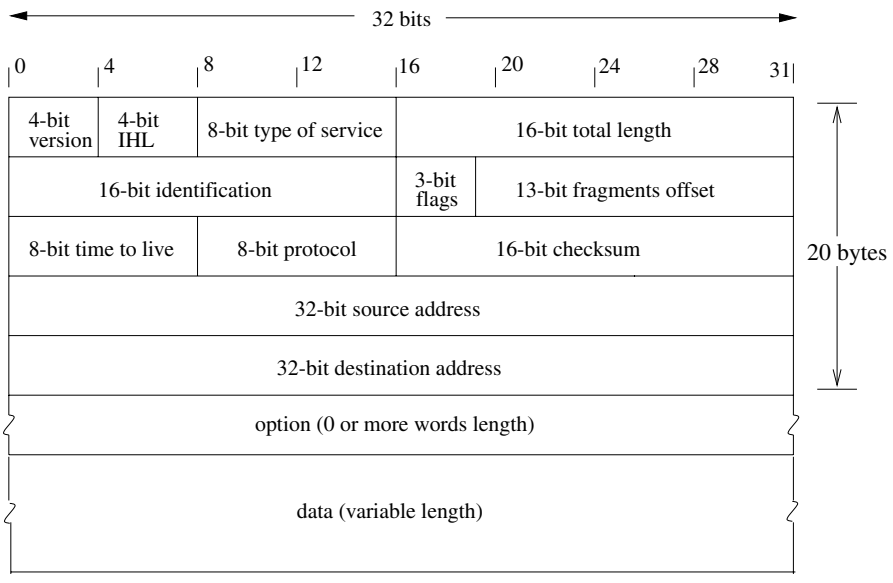

Figure 7-1
IPv4 Datagram

When an IP datagram is received, the receiver calculates the 16-bit one's complement sum of the header. Since the receiver's calculated checksum included the checksum stored by the sender, if the IP header has not been modified, the checksum the receiver computes will be zero. If the receiver's checksum is not zero, the datagram will be discarded. Let us illustrate the calculation of an IP header checksum with an example.

In our example, a computer cracker—we'll call him cracker Joe—is operating from home on a machine that is connected to the Internet via DSL. Assume that the IP address of his machine is 64.143.185.5. (This is a randomly chosen IP address, no malicious intent was intended by choosing this IP address. At the time of writing, an `nslookup` on this IP address indicated that it was either unassigned or it was assigned to a machine that is behind a firewall.) Let us say that Cracker Joe wants to stage a denial of service (d.o.s) attack at the machine hosting the webserver *www.victim.com*. d.o.s attacks are exploits that render a machine unusable until it is rebooted. Assume that the IP address for this machine is 126.30.17.52. Cracker Joe desires to cover his path; that is, he does not want the attack trail to be traceable back to his machine. Cracker Joe did a search on the web and found the source code for a program that launches d.o.s. attacks. He also found another program that is able to intercept IP datagrams immediately after they leave the TCP/IP stack, modify their

contents, and replay the modified datagrams. We will construct an IP datagram and illustrate by way of an example how this program could modify this datagram and replay it.

The first 4 bits of an IP datagram are the version field (see Figure 7-1). Since we are talking about an IPv4 datagram, the version number is 4. The 4-bit binary equivalent of 4 is 0100.

The next field, the IHL (Internet header length) field, is the length of the header in 32-bit words. The minimum value is 5 words (20 bytes), which is the case when no options are present; the maximum value is 15 words (60 bytes), which applies when the options field is 40 bytes. We will assume that there are no options. Therefore, the value of the IHL field is 5; the 4-bit binary number for 5 is 0101.

Next, the 8-bit type of service (TOS) field, is set to zero, since there are no special TOS requirements; thus, the value for the TOS field is 00000000.

The next field is a 16-bit field that stores the total length of the IP datagram, including the data field. In our example, we assume that Cracker Joe is launching a SYN flood attack. In Chapter 1, in our presentation on the TCP protocol, we explained how a three-way handshake is necessary to establish a TCP connection. We recapitulated the information we presented with regard to the initiation of a TCP connection. TCP uses a 32-bit sequence number to uniquely identify each datagram segment that a host transmits. The first sequence number, the initial sequence number (ISN), that a host employs during a TCP session must be randomly generated. The sequence number of the remaining data fragment will be the ISN plus the segment sequence. For example, the first data segment will have sequence number ISN+1 and the second, ISN+2, etc. Note that the fragment with the ISN as the sequence number contains no actual TCP data because the connection has not yet been established. During a TCP session, a host uses the sequence numbers associated with the datagram segments it received from its communicating peer for a number of important functions, such as to reassemble the data fragments and to perform flow control. Therefore, the peer needs to inform the other peer about the ISN it generated for the session, before the TCP connection can be established. The steps involved in exchanging the ISNs are illustrated below (in the illustration, A is the initiator of the TCP session and B is the other peer):

```
1) A --> B:  SYN bit set; sequence no. field contains A's ISN.
2) A <-- B:  SYN and ACK bits set; sequence no. field contains
             B's ISN. Acknowledge no. field contains A's ISN.
3) A --> B:  ACK bit set; Acknowledge no. field contains B's
             ISN.
```

The above is referred to as the three-way handshake. A TCP connection is established only if the three-way handshake is successful.

In a SYN flood attack, the host perpetrating the attack—let us call this host A—floods the victim host B with a large number of TCP connection request packets with the SYN bit set, with no accompanying acknowledgment. If the victim host has not been configured correctly so that it is protected from these kinds of attacks, it will attempt to reply to each of the packets it received as indicated in step 2 of the three-way handshake illustrated above. Meanwhile, there will be no acknowledgment from the perpetrating host; as a result, the victim host will be kept waiting for acknowledgments until the time it has been configured to wait for an acknowledgment expires. If there are enough of these connection requests from A, soon all of B's resources will be used up in responding to these SYN packets, and B will be incapacitated until it is rebooted.

The length of a SYN packet, a TCP packet with the SYN bit set, is 20 bytes. A SYN packet has no data and, typically, no options—Figure 7-2 is the figure illustrating the TCP header that we presented in Chapter 1—therefore, the total length of an IP datagram encapsulating a SYN packet is 20 + 20 = 40 bytes. The 16-bit binary equivalent of 40 is 0000000000101000.

The next field is the 16-bit identification field. The identification field allows a host to determine which datagram a newly arrived fragment

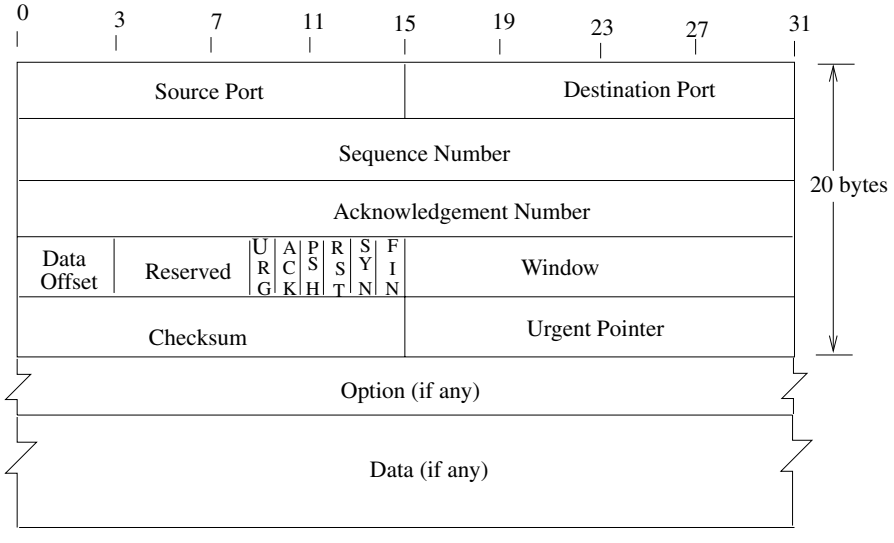

Figure 7-2
TCP Header Format

belongs to. Each datagram has a unique identification number, and each fragment of a datagram has the same identification number. We give this field a 16-bit binary value of 0001110001010100.

For the 3-bit flags field, only 2 of the 3 bits are used. The first of the 2 bits, called the DF (don't fragment) bit, indicates whether or not the IP datagram should be fragmented. If this bit is set, it indicates that the datagram should not be fragmented. In our example, we set the DF bit because we do not want the datagram to be fragmented. The second of the 2 bits, is called the MF (more fragment) bit. It indicates when the last fragment of a datagram arrives. Since the DF bit is set, this bit must be unset. Therefore, the binary value for the 3-bit flag is 100.

The next field, the 13-bit fragment offset field, tells where in the current datagram this fragment belongs. Because the datagram is not fragmented, this 13-bit field will be 0, i.e., 0000000000000.

The 8-bit time-to-live (TTL) field is used to limit the lifetime of a datagram, thus preventing the datagram from looping infinitely within a network segment. The TTL field is set to a default value by the source. Each router that the datagram passes through decrements the TTL value by 1. When a router decrements the TTL value, it recalculates the header checksum and replaces the old checksum with the new value. For the purpose of our example, we will set the TTL field to 12. The 8-bit binary equivalent of 12 is 00001100.

The next field is the 8-bit protocol field. It indicates which transport protocol is used for the data encapsulated within the IP datagram. The packet is a TCP packet; therefore, the protocol number is 6. The 8-bit binary equivalent is 00000110.

The 16-bit checksum field must be set to 0, i.e., 0000000000000000.

The source IP address is the IP address of Cracker Joe's machine, that is, 64.143.185.5. The binary equivalent of this IP address can be computed by calculating the 8-bit binary number for each byte of the IP address and then concatenating the binary numbers. Thus, for this IP address, the binary equivalent of the first byte is 01000000, the second byte is 10001111, the third byte is 10111001, and the last byte is 00000101. The 32-bit representation of this IP address is therefore 01000000 10001111 10111001 00000101.

The destination address is the IP address of *www.victim.com*, which we assume to be 126.30.17.52; the binary equivalent is 01111110 00011110 00010001 00110100.

Table 7-1 shows the binary value we assigned to each field.

The next step involves arranging these values in 16-bit sequences, and summing the sequences. This information is shown in Table 7-2.

TABLE 7-1

Summary of Binary Values for the Fields

Field	Binary Values
Version	0100
Header length	0101
TOS	00000000
total length	0000000000101000
identification	0001110001010100
3-bit flag	100
fragment offset	0000000000000
TTL	00001100
protocol	00000110
checksum	0000000000000000
source IP address	01000000 10001111 10111001 00000101
destination IP address	01111110 00011110 00010001 00110100

As indicated in Table 7-2, the 16-bit calculated sum is 1011 1100 0010 0100. The one's complement of this sum is 0100 1101 1101 1011. Thus, for our example, the binary value 0100 1101 0011 1011 would be placed in the header checksum field.

TABLE 7-2

Summation of the 16-bit Sequences

16-bit Sequences	Fields Composing the Sequence
0100 0101 0000 0000	Values from version, IHL, and TOS fields
0000 0000 0010 1000	16-bit total length field
0001 1100 0101 0100	16-bit identification field
1000 0000 0000 0000	Flags and fragmentation offset fields
0000 1100 0000 0110	TTL and protocol fields
0000 0000 0000 0000	Checksum field
0100 0000 1000 1111	First 2 bytes of source IP address
1011 1001 0000 0101	Last 2 bytes of source IP address
0111 1110 0001 1110	First 2 bytes of destination IP address
0001 0001 0011 0100	Last 2 bytes of destination IP address
1011 1100 0010 0100	16-bit calculated sum

TABLE 7-3

Summation of the 16-bit Sequences at the Receiving Host

16-bit Sequences	Fields Composing the Sequence
0100 0101 0000 0000	Values from version, IHL, and TOS fields
0000 0000 0010 1000	16-bit total length field
0001 1100 0101 0100	16-bit identification field
1000 0000 0000 0000	Flags and fragmentation offset fields
0000 1100 0000 0110	TTL and protocol fields
0100 0011 1101 1011	Checksum field
0100 0000 1000 1111	First 2 bytes of source IP address
1011 1001 0000 0101	Last 2 bytes of source IP address
0111 1110 0001 1110	First 2 bytes of destination IP address
0001 0001 0011 0100	Last 2 bytes of destination IP address
1111 1111 1111 1111	16-bit calculated sum

When the datagram reaches the destination host, the checksum is recomputed as shown in Table 7-3.

In actuality, the TTL field likely would have been changed; however, since a new checksum is calculated each time that the TTL value is decremented, the effect will be canceled out. We are using the checksum that was calculated by the source host, rather than that calculated by an intermediate router; therefore, we need to use the original TTL value.

As indicated in Table 7-3, the computed 16-bit sum at the receiver's end consists of all 1 bits. The complement of this bit-string is a bit-string consisting of 16 zeros. Therefore, the datagram will be accepted by the receiving host as authentic.

The point is that if a datagram is intercepted, modified, and then sent to its intended destination, provided that the header checksum is recalculated correctly after the changes to the desired fields are made, the receiving host likely will not detect that the datagram has been tampered with. We illustrate this by returning to our example involving cracker Joe.

Cracker Joe executed a d.o.s. attack on the machine hosting the *www.victim.com* web server; in order to cover his tracks, he used a spoofing software to capture the packets that are used to launch the d.o.s attack—immediately after they left the TCP/IP stack—and then changed the source destination field, recalculated the header checksum, and relayed the packets to their intended destination. Let us assume that Cracker

Joe wanted the administrator of the *www.victim.com* machine to think that the d.o.s packets originated from a machine with the IP address 161.69.3.155. Therefore, cracker Joe configures his spoofing software to replace the source IP address of the packets that are used to attack the victim computer with 10100001 01000101 00000011 10011011, which is the 32-bit binary equivalent of 161.69.3.155. The new checksum value is as shown in the summation of the 16-bit sequences in Table 7-4.

The new header checksum is the one's complement of 1110 0111 0111 0000, which is 0001 1000 1000 1111. This 16-bit value is placed in the checksum field. When the datagram with the spoofed source IP address arrives at the victim host, it will calculate the one's complement checksum, which will be zero. The summation is shown in Table 7-5.

The one's complement of 1111 1111 1111 1111 is a 16-bit zero string. Therefore, the victim host will accept the packet as being authentic, whereas, as we indicated, the source IP address was in fact spoofed.

The spoofing of IP addresses—particularly when a connection is not required between the spoofed host and the destination host—can be done quite easily using software that is freely available on the Internet. Changing a field in an IP datagram is quite trivial; calculation of the header checksum requires more effort. However, RFC 1071 [BBP88] outlines how it can be done efficiently; this RFC even includes C source code for calculating IP checksums.

TABLE 7-4

Summation of the 16-bit Sequences After the Modification of Destination IP Address

16-bit Sequences	Fields Composing the Sequence
0100 0101 0000 0000	Values from version, IHL, and TOS fields
0000 0000 0010 1000	16-bit total length field
0001 1100 0101 0100	16-bit identification field
1000 0000 0000 0000	Flags and fragmentation offset fields
0000 1100 0000 0110	TTL and protocol fields
0000 0000 0000 0000	Checksum field
1010 0001 0100 0101	First 2 bytes of source IP address
0000 0011 1001 1011	Last 2 bytes of source IP address
0111 1110 0001 1110	First 2 bytes of destination IP address
0001 0001 0011 0100	Last 2 bytes of destination IP address
1110 0111 0111 0000	16-bit calculated sum

TABLE 7-5

Summation of the 16-bit Sequences for the Modified Datagram at the Receiving Host

16-bit Sequences	Fields Composing the Sequence
0100 0101 0000 0000	Values from version, IHL, and TOS fields
0000 0000 0010 1000	16-bit total length field
0001 1100 0101 0100	16-bit identification field
1000 0000 0000 0000	Flags and fragmentation offset fields
0000 1100 0000 0110	TTL and protocol fields
0001 1000 1000 1111	Checksum field
1010 0001 0100 0101	First 2 bytes of source IP address
0000 0011 1001 1011	Last 2 bytes of source IP address
0111 1110 0001 1110	First 2 bytes of destination IP address
0001 0001 0011 0100	Last 2 bytes of destination IP address
1111 1111 1111 1111	16-bit calculated sum

If a connection is desired between the spoofed host and the victim host, much more effort is required since the perpetuation of these types of exploits involves predicting the ISN of the SYN packet used to initiate the connection from the host that is being spoofed. However, this has not been much of an impediment to crackers, who over the years have fine-tuned different techniques for predicting ISNs and consequently routinely spoofed IP addresses.

IPSec was designed to prevent the spoofing of IP addresses, prevent any form of tampering with and replaying of IP datagrams, and provide confidentiality and other security services for IP datagrams. IPSec offers these services at the network layer—the layer in the TCP/IP protocol stack that contains the IP protocol. The security that IPSec affords is provided via combinations of cryptographic protocols and security mechanisms. IPSec enables systems to select required security protocols, choose the cryptographic algorithms that are desired to be used with the selected protocols, and generate and put in place any cryptographic keys that are necessary to provide the requested services.

The security services that IPSec affords include access control to network elements, data origin authentication, connectionless integrity for protocols such as UDP that offer connectionless services, detection and rejection of replayed packets, use of encryption to provide data confiden-

tiality, and limited traffic flow confidentiality. Since the IPSec services are offered at the network layer—layer 2 of the TCP/IP protocol stack—these services can be used by any of the upper-layer protocols such as TCP, UDP, ICMP, and IGMP or any application layer protocol.

7.2 How IPSec Works

IPSec is designed to provide high-quality, interoperable cryptographic-based security for IPv4 and IPv6 datagrams. IPSec achieves these objectives by using two traffic security protocols: authentication header (AH) and encapsulating security payload (ESP), and through the use of cryptographic-key management procedures and protocols such as Internet Key Exchange (IKE) protocol.

The IP AH protocol provides data origin authentication, connectionless integrity, and an optional anti-replay service. The ESP protocol provides data confidentiality, limited traffic flow confidentiality, connectionless integrity, data origin authentication, and anti-replay service. There are two modes of operation of both AH and ESP: transport mode and tunnel mode. We explain these modes of operation, as they apply to AH and ESP, in the next two chapters. The IKE protocol is used to negotiate the cryptographic algorithm choices to be utilized by AH and ESP, and put in place the necessary cryptographic keys that the algorithms require. AH, ESP, and IKE protocols are discussed in more detail in later chapters.

The protocols that IPSec utilizes are designed to be algorithm independent. The choice of algorithms is specified in the Security Policy Database (SPD). The available choice of cryptographic algorithms depends on the IPSec implementation; however, a standard set of default algorithms are specified by IPSec to ensure interoperability on the global Internet.

IPSec allows the user or administrator of a system or a network to control the granularity at which the security service is offered. For example, an organization's policy might specify that data traffic that originated from certain subnets should be protected with both AH and ESP and that the encryption should be done with triple-DES with three different keys. On the other hand, the policy might specify that data traffic from another site should be protected with only ESP and that this traffic should be afforded encryption with AES (Advanced Encryption Standard). IPSec is able to differentiate between the security services it offers to different data traffic by the use of security association (SA).

7.3 Security Association

The concept of SA is fundamental to IPSec. The two protocols that IPSec uses—AH and ESP—both use SA, and a principal function of the IKE protocol, the key management protocol that IPSec uses, is the establishment and maintenance of SA. SA is an agreement between communicating peers on factors such as the IPSec protocol, mode of operation of the protocols (transport mode or tunnel mode), cryptographic algorithms, cryptographic keys, and lifetime of the keys that will be used to protect the traffic between them. If both AH and ESP are desired for protecting the traffic between two peers, then two sets of SAs are required: an SA for AH and one for ESP. SAs that define tunnel mode operation for AH or ESP are called tunnel mode SAs, whereas those that define transport mode are called transport mode SAs. Security associations are simplex (that is, they are unidirectional); therefore, separate SAs are required for outbound and inbound traffic. The term SA bundle is used to describe a set of SAs that are to be applied to data originated from or destined to a given host.

SAs are negotiated between the communicating peers via key management protocols such as IKE. When the negotiation of an SA completes, both peers store the SA parameters in their security association databases (SADs). One of the parameters of an SA is its lifetime, which takes the form of a time interval or a count of the number of bytes to which IPSec protocol will be applied for the SA in question. When the lifetime of an SA expires, this SA is either replaced by a new SA or is terminated. When the SA terminates, its entry is deleted from the SAD.

SAs are uniquely identified by a triplet consisting of a security parameter index (SPI), a destination IP address for outbound SAs or a source IP address for inbound SAs, and a specified protocol (example AH or ESP). The SPI is a unique 32-bit integer that is generated and used as a unique identifier of an SA. It is transported in the AH and ESP headers. Therefore, the recipient of the IPSec datagram can readily identify the SPI and use it along with the source or destination IP address and the protocol to search the SAD to ascertain the SA or SA bundle that is associated with the datagram.

7.4 Security Association Databases

There are two databases that are necessary for the processing of IPSec traffic: security policy database (SPD) and SAD. SPD specifies the policies that are to be applied to the traffic destined to or originated from a

given host or network. SAD contains the active SA parameters. For both SPD and SAD, separate inbound and outbound databases are required.

7.4.1 Security Policy Database

The IPSec protocol mandates that the SPD must be consulted during the processing of all traffic, whether the traffic is inbound or outbound. The SPD contains an order list of policy entries. Each entry is specified by the use of one or more selectors. The selectors that IPSec currently allows are:

- *Destination IP address:* The destination IP address can be a 32-bit IPv4 or a 128-bit IPv6 address. The address can be a host IP address, a broadcast, unicast, anycast, a multicast group, a range of addresses, address plus netmask, or wild card address. The destination IP address is obtained from the *destination IP address* field of the AH, ESP, or—if IPSec is not applied to the packet—IP header.
- *Source IP address:* The source IP address, like the destination IP address, can be a 32-bit IPv4 or a 128-bit IPv6 address. This address, similarly, can be a host IP address, a broadcast, unicast, anycast, a multicast group, a range of addresses, address plus netmask, or wild card address. The destination IP address is obtained from the *source IP address* field of the AH, ESP or IP header.
- *Transport layer protocol:* The transport layer protocol is obtained from the IPv4 *protocol* or the IPv6 *next header fields*.
- *System name:* The system name can be a fully qualified DNS name, or e-mail address, such as bert.cs.mcgill.ca, or an X.500 DN, for example, *cn=bert,ou=Computer Science,o=McGill University, st=Quebec,c=Canada*. Refer to the discussion on the X.500 directory in Chapter 5 for detail on DNs.
- *User ID:* The user ID can be a fully qualified DNS user name such as *foo@cs.mcgill.ca* or an X.500 DN.

Each entry in the SPD consists of one or more selectors, an indication of whether the packets that match the selectors in the the entry should be discarded, be subjected to IPSec processing, or not be subjected to IPSec processing. If the packets are to be subjected to IPSec processing, the entry must also contain a pointer to an SA specification that details the IPSec protocols, modes, and cryptographic algorithms to be applied to the packets matching this policy entry.

The first entry with selectors matching those corresponding to the traffic under consideration will be applied. If no matching entry is found, the packets for the traffic under consideration will be discarded. Therefore, the entries in the SPD should be ordered according to the desired preference of application.

The entries in the SPD determine the granularity with which the traffic is processed. For example, the policies could specify that IPSec service corresponding to a given SA or SA bundle should be applied to all traffic to or from any source or destination, or the policy could specify the application of different SAs or SA bundles based on specified selectors. The SPD plays a very important role in the control of the flow of all traffic through an IPSec system.

7.4.2 Security Association Database

The SAD contains the active SA entries. Each SA entry is indexed by a triplet consisting of an SPI, a source or destination IP address, and an IPSec protocol. In addition, an SAD entry consists of the following fields:

- *Sequence number counter:* This is a 32-bit integer that is used to generate the sequence number field in AH or ESP headers.
- *Sequence counter overflow:* This is a flag indicating whether the overflow of the sequence number counter should be audited and the transmission of additional traffic be blocked for the given SA.
- *Anti-replay window:* A 32-bit counter and a bit-map that are used to ascertain whether an inbound AH or ESP packet is a replay.
- AH authentication cryptographic algorithm and the required key.
- ESP authentication cryptographic algorithm and the required key.
- ESP encryption algorithm, key, initial vector (IV), and IV mode. We discussed the IV and IV modes of operation in Section 2.3.
- *IPSec protocol mode:* This field indicates which IPSec protocol mode (transport, tunnel, or wild card) should be applied to AH and ESP traffic.
- *Path maximum transfer unit (PMTU):* Any observed PMTU and aging variables. The PMTU is the maximum size of an IP datagram that will be allowed to pass through a given network path on route from a source to a destination host, without being subjected to fragmentation.

- *Lifetime of the SA:* This field contains the time interval within which a SA must be replaced by a new SA or be terminated, plus an indication of whether the SA should be replaced or terminated when it expires. The lifetime of a SA takes two forms: a time interval or a byte count that represents the number of bytes that IPSec protocols have been applied to. The parameter that expires first takes precedence.

Separate SADs are kept for inbound and outbound IPSec processing. For inbound or outbound traffic, the respective SADs are searched for selectors matching the SPI, the source or destination IP address, and the IPSec protocol, extracted from the packet header. If a matching entry is found, the parameters for this SA are compared with the appropriate fields in the AH or ESP headers. If the header fields correspond with the SA parameters in the database, the packet is processed. However, if there are any discrepancies, the packet is discarded. If no SA entry matches the selectors, the packet is discarded if it is an inbound packet. However, if the packet is outbound, a new SA or SA bundle will be created and entered in the outbound SAD.

A SA lifetime has two kinds of limits: a soft limit and a hard limit. When the soft limit is reached, the communicating peers must renegotiate a new SA to replace the existing one. However, the existing SA is not deleted from the database until the hard limit expires. Unlike the SPD, the entries in the SAD are not ordered. Nonetheless, as is the case with SPD lookups, the first matching SA entry that is found in the SAD is used for the IPSec processing of the packets that are associated with the given SA.

In the following three chapters, we give detailed descriptions of the three IPSec protocols, AH, ESP and IKE, and explain how they interoperate to secure IP traffic.