INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

IPSec Base Virtual Private Network

Carlton Roy Davis

School of Computer Science

McGill University, Montreal, Canada.

May 2000

A thesis submitted to the faculty of Graduate Studies and Research in partial fulfillment of the requirement for the degree of Master in Computer Science.

Abstract

The Internet evolved from an experiential packet-switching network called the ARPANET. This network has grown exponentially since its conversion from an experimental to an operational network in 1975. However, the need for confidential and secure data channel has dissuaded many enterprises from using this ubiquitous public infrastructure. The IPSec protocol suite developed by the Internet Engineering Task Force (IETF) makes it possible to implement secure communication channels or virtual private network (VPN) over the Internet. Corporations can benefit from substantial financial savings by utilizing VPN for inter-company or intra-company communications rather than using expensive lease or privately own network infrastructure with its associated high maintenance costs. In this thesis, we will discuss the architecture, design and use of IPSec base VPN.

Résumé

L'internet a évolué d'un réseau expérimental de "packet-switching" appelé ARPANET. Ce réseau a grandi exponentiellement depuis sa transformation d'un réseau expérimental à un réseau opérationnel en 1975. Cependant, les besoins de média de transfert de données à la fois confidentiels et sécuritaires ont dissuadé beaucoup d'entreprises d'utiliser cette infrastructure publique omniprésente. La suite de protocoles IPSec dévelopée par la Internet Engeneering Task Force (IETF) rend possible l'implémentation de canaux de communication ou de réseaux privés virtuels (VPN) par-dessus Internet. Les entreprises peuvent bénéficier d'économies monétaires substantielles en utilisant les VPN pour leurs communications internes ou externes plutôt que d'utiliser des infrastructures privées coûteuses qui de plus coûtent très cher à maintenir. Dans ce mémoire, nous discuterons l'architecture, la conception et l'utilisation d'un VPN basé sur le protocole IPSec.

Acknowledgement

I would like to express my appreciation to my supervisor Prof. Ratzer for the help he provided for this research project. I wish also to say a special thank you to Luc Boulianne for spurring my interest initially in this subject area. Thanks also to Prof. Dini and his research colleagues at CRIM who provided some assistance during the early stage of the research.

I am very grateful for the support and encouragement I received from my family and friends.

Finally, I wish to say thanks to McGill University for providing the facilities for me to do the research.

Contents

Chapter 1

Introduction

1.1 Motivation

In the mid 1960's [Tan96] in the height of the cold war, the Department of Defense (DoD) in the USA wanted a command and control network that could survive a nuclear war. The DoD consequently commissioned its research arm— ARPA (Advance Research Project Agency)—to invent the technology that could get data to its destination reliably even if arbitrary part of the network disappeared without warning as a result of a nuclear attack.

Traditional telephone technology called circuit switching would not work because it had serious drawbacks. In circuit switching, a route for data to get from one place to another is set up by using relays to make physical connections among pieces of cable. Therefore, if part of the circuit fails, a new circuit must be set up, which could be quite difficult and time consuming depending on the severity of the damage [CQ93].

ARPA used a different kind of technology called packet switching. The idea of packet switching network was proposed by Paul Baran [Bar64]. In packet switching, data to be sent over the network is divided up into discrete parts called packets. Each packet is routed independently from one computer to the next over the network until it reaches its final destination.

The first experimental network—called the ARPANET—went into being in De-

cember 1969. It consisted of subnets and hosts computers. The subnets consisted of minicomputers called IMPs (Interface Message Processors) connected by transmission lines. This network contained four nodes, one each at UCLA (University of California at Los Angeles), UCSB (University of California at Santa Barbara), SRI (Stanford Research Institute) and University of Utah. Each node of the network composed of an IMP and an host in the same room, connected by wires. These four sites were chosen because all had large ARPA contracts; additionally, all four sites had different and completely incompatible computers. This experimental network grew rapidly: in July 1970 it grew to eight nodes, by March 1971 it expanded to sixteen nodes, in April 1972 it grew to twenty three nodes and by September 1972 it consisted of thirty four nodes [Tan96].

In the 1983 TCP/IP protocols were adopted as the only official protocol of the ARPANET. During the same year the term *Internet* came into common usage. The old ARPANET was divided into MILNET, the unclassified part of the Defense Data Network (DDN), and a new smaller ARPANET. The "Internet" was used to refer to the entire network: the MILNET plus the ARPANET [Hun98]. Many regional networks in the USA joined up to the Internet. In the mid 1980's trans-Atlantic fiber optic cables were laid and consequently, networks in Europe, the Pacific region and Canada were connected to the Internet. Growth of the Internet continued exponentially, and by 1995 there were multiple backbones, hundreds of regional networks, tens of thousands of Local Area Networks (LAN's), millions of hosts, and tens of millions of users. At the close of the twentieth century almost every nation is connected to the Internet.

The ubiquitous nature of the Internet has raised many security concerns. The principal concern over the years is the fact that IP packets are inherently insecure. It is relative 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 the IP datagrams originated from the source it claims to originate from, or 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.

These security concerns have dissuaded many corporation from using the Internet to transmit sensitive electronic data even though the price of bandwidth over the Internet has fallen considerably over the years. These corporations continue to use expensive leased or privately own networks infrastructure with their high associate support and maintenance cost to transmit intra or inter-company electronic data.

IPSec—the IP security protocol developed by the Internet Engineering Task Force (IETF) in the late 1990's—has addressed most of the security issues of the IP datagram. With IPSec authentication, it is possible to detect if an original IP datagram has been modified, thus adding some guarantee to the origin and integrity of the data; it also provides data content confidentially and replay protection.

The main contribution of this thesis is to examine the components of IPSec and present different designs of IPSec based Virtual Private Network (VPN) that enterprises can use to transmit confidential electronic data over the Internet.

1.2 Thesis Organization

This thesis is organized as follows: Chapter 2 gives an overview of TCP/IP. Section 2.1 gives a brief history presenting the motivation for the development of this protocol suite; and Section 2.2 looks at the TCP/IP protocol architecture and the four layers of the TCP/IP protocol stack.

Chapter 3 gives an overview of cryptographic techniques relevant to the IPSec protocol. Section 3.1 examines the Data Encryption Standard (DES) and other private-key cryptographic protocols. Section 3.2 looks at the design principle and application of commonly used public-key cryptosystems. Section 3.3 examines the design of two cryptographic hash functions and a Message Authentication Code (MAC). The final Section in this chapter looks at digital signatures and examines the Digital Signature Standard (DSS).

Chapter 4 examines the IPSec architecture and introduces some of the components of this security protocol. Section 3.1 explains what IPSec does, section 3.2 explains how IPSec works, section 3.3 presents the concept of Security Association

(SA), section 3.4 looks at IPSec Security Association Databases and finally, Section 3.5 looks at SA and key management.

Chapters 5 to 7 give detail on the main components of IPSec. Chapters 5 and 6 look at IPSec Authentication Header (AH) and Encapsulating Security Payload (ESP) respectively; each chapter discusses the the component's format, the different modes of these components, and the processing of the components. Chapter 7 looks at the Internet Key Exchange (IKE) protocol. Section 7.1 examines the components of Internet Security Association and Key Management Protocol (ISAKMP) that IKE utilizes. Section 7.2 discusses the IKE exchange types; and the final section in this chapter presents the groups that can be utilized for the Diffie-Hellman key exchange.

Chapter 8 presents some IPSec based VPN solutions; and discusses relevant issues, such as addressing and routing. Finally, Chapter 9 concludes the research and make some recommendations for further study.

Chapter 2

TCP/IP Overview

2.1 Some History

In December 1969 ARPA (Advance Research Project Agency)—the research arm of the USA Department of Defense (DoD) commissioned the first experimental network. This network—called the ARPANET consisted of four nodes, one each at UCLA (University of California at Los Angeles), UCSB (University of California at Santa Barbara), SRI (Stanford Research Institute) and University of Utah. The network was developed in response to the DoD desire for a network that could withstand a nuclear war. The DoD wanted a network that would keep connections intact provided that the source and the destination machines were functioning, even if some components of the network disappear without warning. This network worked well in its early stage when there were few nodes. However, as the number of nodes increased the network was subjected to a number of system crashes [Ano98]. Additionally, when satellite and radio networks were added to the ARPANET in the early 1970's. Network Control Protocol (NCP) [NKPC70], the existing protocol of the ARPANET, had trouble working with these networks. As a result, research for a new protocol that was robust and able to work well with different kinds of network, started in the early 1970's. The research effort culminated with the development of the TCP/IP protocol suite in 1974 [CK74]. A later perspective of this protocol is given in [LCPM85] and its design philosophy is describe in [Cla88].

The TCP/IP protocol suite proved to be quite robust and was very adaptable to the different networks. This along with the fact that is was an open protocol, it was freely available, it was developed independently of any specific computer hardware or operating system, and it was simple and easy to implement—made it very popular. By 1983 TCP/IP was integrated into release 4.2 of Berkeley Software Distribution (BSD) UNIX, and the same year, the DoD adopted this protocol suite as Military Standard (MIL STD) and it became the standard of the Internet.

Today, TCP/IP continues to be the standard for the Internet. The fact that this protocol was designed to be independent of any specific physical network hardware has allowed it to be integrated into many different kinds of networks. TCP/IP is currently integrated into Ethernets, token ring, dial-up networks and just about every other type of physical transmission medium, and virtually all modern operating systems.

2.2 TCP/IP Protocol architecture

Networking protocols are normally developed in layers, with each layer responsible for a different facet of the communication [Ste94]. The TCP/IP protocol suite consists of four layers. Each layer has distinct functions and consists of a combination of different protocols as shown in Figure 2.1. The functionalities of the layers are discussed below.

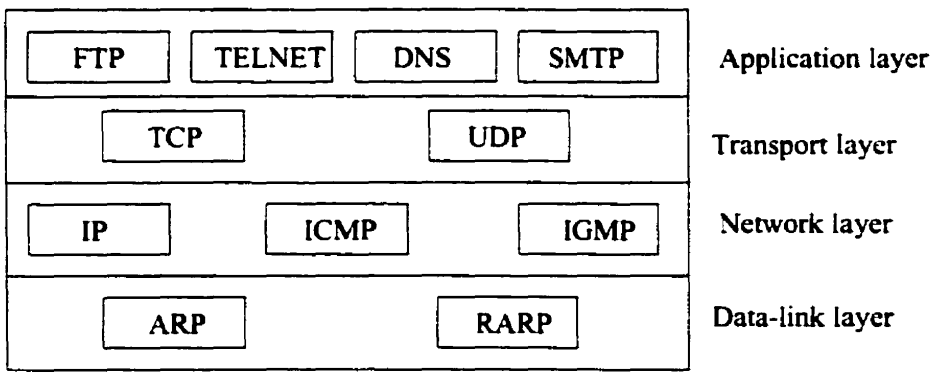

Figure 2.1: Protocols at the four layers of the TCP/IP protocol suite.

2.2.1 Data-link Layer

The *data-link* layer is also called the *link* or *network access layer*. It is the lowest layer of the TCP/IP protocol stack. Example of data-link layers are Ethernet, Token Ring and ATM (Asynchronous Transfer Mode). For information on Ethernet refer to [MB76], for information on Token Ring see [LRA92], and for further detail on ATM see [FWWD94], [Gor95] and [Kya95]. The protocols in data-link layer provide means for the system to deliver data to other devices on the network. These protocols must know details such as the packet structure and addressing scheme of the underlining network. Two of the protocols in this layer are ARP (Address Resolution Protocol), which maps IP address to Ethernet address; and RARP (Reverse Address Resolution Protocol), which maps Ethernet address to IP address. See [Plu82] and [FTMT84] for further information on these protocols.

When the data-link layer receives a packet from the network layer—the layer above it—it encapsulates the packet with the appropriate header then sends it via the physical network to the specified device. The reverse occurs when a packet arrives from the physical network to the data-link layer: the layer removes the data-link header then sends the packet up to the network layer for processing. This is illustrated in Figure 2.2.

2.2.2 Network Layer

The *network* layer is also called the *internet* layer. This layer is responsible for the routing of packets from the source to the destination. The network layer consists of three protocols: Internet Control Protocol (ICMP), Internet Group Management Protocol (IGMP) and Internet Protocol (IP).

The official specification of *ICMP* is outlined in [Pos81b]. ICMP is the protocol that is used to communicate error conditions that occurred during the transmission of IP packets. The ICMP message is usually encapsulated in the IP datagram. Another important role of ICMP is the debug of network connectivity problems. The popular debugging tools: ping and traceroute use this protocol.

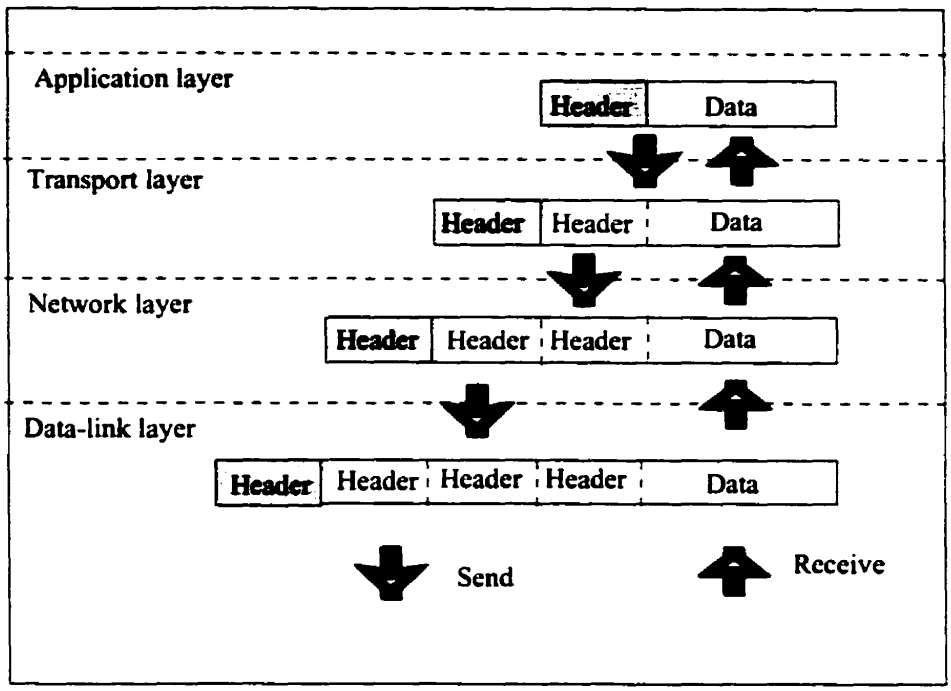

Figure 2.2: Encapsulation and de-capsulation of data as it goes down and up the protocol stack, respectively.

IGMP is the protocol that is used by hosts and routers to ascertain information about the hosts in a multicast groups: groups of hosts to which IP datagrams are to be sent simultaneously. As is the case with ICMP, IGMP messages are encapsulated in IP datagrams. For further detail on IGMP refer to [Dee89] which contains the original specification of IGMP.

IP is the protocol that holds the whole TCP/IP protocol suite architecture together. The data of the protocols in all of the layers of the TCP/IP protocol stack, except the data-link layer, are transmitted as IP datagrams. IP allows hosts to inject packets into the data-link layer—which eventually puts them on the physical network—and have them travel on potentially different networks to their final destination. IP offers a *connectionless* service. In connectionless services, each message or datagram carries the full destination address, and is routed independently of the other datagrams. With connectionless services, it is possible for the datagrams to arrive at

their destination in different order that they were sent. Whereas, this is not possible with *connection-oriented* services, since the latter is like a telephone system which establishes a connection or a path from the host to the destination, uses the path to send data, then releases the connection after the data transmission is completed.

There are two versions of IP that are available for use with the TCP/IP protocol suite: IPv4 and IPv6 [1]. The format of the datagrams of these two protocols are discussed in the following two sections.

IPv4 Datagrams Format

An IP datagram consists of a header portion and a data portion. The header portion consists of a 20 bytes fixed part and a variable length optional part. The data portion is of variable length. Figure 2.3 shows the format of an IPv4 datagram. An IP datagram is transmitted in *big endian* byte order: from left to right, that is, lower order bytes are transmitted first. This is the byte ordering required for all binary integers in the TCP/IP headers as they traverse a network. This is called *network byte order*. Machines such as Pentiums, which uses *little endian* byte ordering format must convert header values to network byte order before transmitting the data.

The *version* field indicates the version of the IP protocol that the data belongs. This field is usually use for backward compatibility.

The *IHL* field is the length of the header in 32-bits words. The minimum value is 5 words (20 bytes) which is the case when no options are present: and the maximum value is 15 words (60 bytes) which applies when the options field is 40 bytes.

The *type of service (TOS)* field indicates the traffic requirement of the datagram in terms of the combination of delay, throughput and reliability.

The *total length* field is the total length of the IP datagram—header and data—in bytes. The maximum length is 65,535 bytes. This field is used in combination with the IHL field to indicate where the data portion of the IP datagram begins.

The *identification* field allows a host to determine which datagram that a newly arrived fragment belongs to. Each datagram has a unique identification number. and

[1] IPv5 is a real-time stream protocol

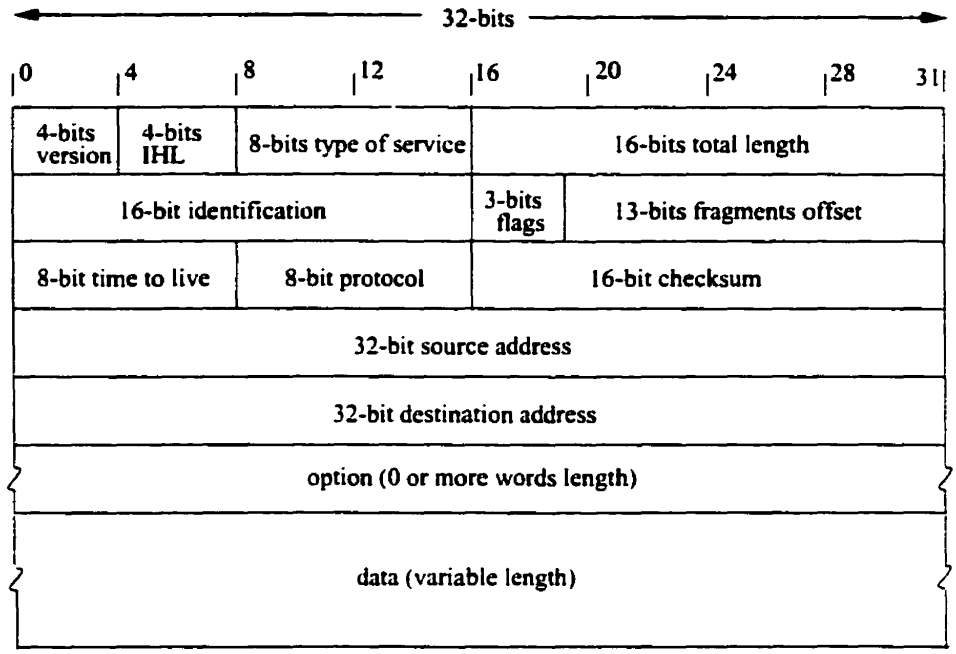

Figure 2.3: IPv4 datagram.

each fragment of a datagram has the same identification number.

Only 2 of the 3 bits *flags* are used. The first of the two bits is called the *DF* (don't fragment) bit; it indicates whether or not the IP datagram should be fragmented. If this bit is set, it indicates that the datagram should not be fragmented. When a router receives a datagram with DF bit set, it usually sends back an ICMP message to the host from which the datagram originated indicating its MTU (Maximum Transfer Unit). This bit is therefore used to determine path MTU (path Maximum Transfer Unit): the maximum size of an IP datagram that will be allowed to pass through a given network path on route to the destination without been subjected to fragmentation. The next bit is called *MF* (more fragment) bit. This bit indicates when the last fragment of a datagram arrives. This bit is set for all of the fragments of a datagram except the last fragment.

The *fragment offset* tells where in the current datagram this fragment belongs. All fragments except the last one in a datagram must be a multiple of 8 bytes.

The *time to live (TTL)* field is used to limit the life time of a datagram, thus

preventing datagram from looping infinitely within a network segment. The TTL field is set to a default value by the host. Each router that the datagram passes through, decrements the TTL value by one. If a router receives a datagram with a TTL value of 1, it discards the datagram and sends an ICMP message to the source indicating that the TTL value of the datagram has expired.

The *protocol* field tells which transport protocol is used for the data encapsulated within the IP datagram. It allows destination hosts to demultiplex IP datagrams among the different transport protocols.

The *header checksum* is computed on the IP header and is used to determine the integrity of the IP header. It should be noted that the checksum is not encrypted and it can easily be forged.

The *source and destination address* fields indicate the 4 bytes IP address of the host that generated the datagram, and the destination host respectively.

The variable length *options* field carries optional information about a datagram such as the security and handling restriction. This field is rarely used and is usually ignored by most routers.

The *data* portion of the IP datagram is of variable length and it contains the IP payload. For further detail regarding the format of the data or header portion of an IPv4 datagram refer to [Pos81] which contains the official specification of IPv4 protocol.

IPv6 Datagram Format

IPv4 suffers from a major limitation: it limits IP address to 32 bits. With the current rate of growth of the Internet, there is a real possibility that if the size of IP addresses does not increase, then there might not be enough IP addresses to meet the demand. The designated successor of IPv4: IPv6, overcomes this limitation as well as simplifies the IP header and adds more flexibility to the IP datagram. Some of the changes from IPv4 to IPv6 are outlined below.

- IPv6 increases the IP address size from 32 bits to 128 bits, thus increasing the addressable nodes by many folds.

- The number of header fields is reduced from 13 in IPv4 to 7 in IPv6. The smaller number of headers allows routers to process packets faster and therefore, increases throughput.

- IPv6 provides better support for options. The options are treated as separate headers instead of being a part to the IP header. This change allows routers to skip over headers that are not intended for them. This feature speeds up packet processing time; it also allows for less stringent limits on the length of options and provides greater flexibility for the introduction of new options in the future.

- IPv6 provides new capability to label packets belonging to particular traffic stream for which the sender requests special handling, such as non-default quality of service.

- IPv6 does not support any fragmentation for packets in transit. The host that generates the packet must perform path MTU to ascertain the maximum size of the IP datagram that will be allowed to pass through the network segment on route to the destination host.

- IPv6 specified extension to support authentication, data integrity and (optional) data confidentiality.

Figure 2.4 shows the required IPv6 fixed headers.

The *version* field is similar to IPv4 version field (see Section 2.2.2).

The *priority* field is used to indicate the quality of service that a packet requires.

The *flow label* field is still experimental, but is likely that it will be used in the future to set up a pseudo-connection with particular properties and requirements [Tan96].

The *payload length* is a 16 bit unsigned integer which indicates the length of the IP payload, that is, the rest of the packet following the IPv6 headers.

The *next header* field identifies the type of header immediately following the the IPv6 header. This field facilitates the reduction in the number of fields in the IPv6

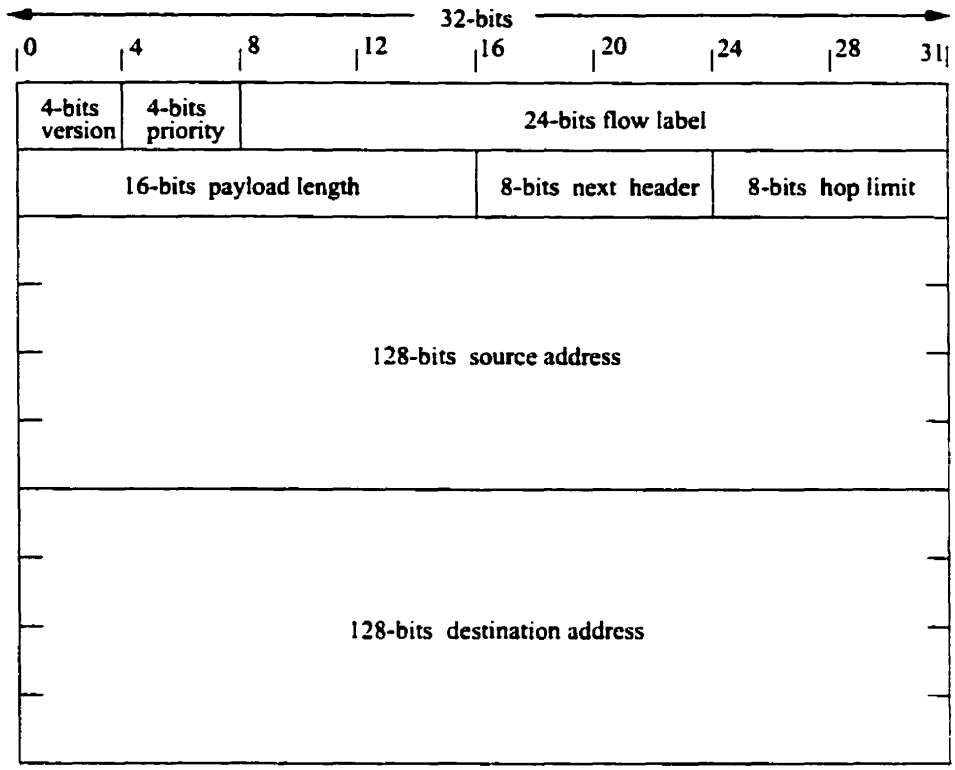

Figure 2.4: IPv6 Header Format.

header compared to that of IPv4. It tells which extension header, if any, follows this one; if this header is the last IP header, it tells which transport protocol the packet should be passed to. For information on the number assigned to each protocol see [RP94].

The *hop limit* field is the same as the TTL field in IPv4; see Section 2.2.2.

The *source and destination address* fields represent the 128-bits IPv6 addresses of the source and intended recipient of the packet, respectively.

Extension Headers

IPv6 introduced the concept of optional *extension headers*. These headers can be supplied to provide additional information. There are currently 6 extension headers defined by IPv6, and each has a unique identification number (described in [RP94]). The extension headers, if present, are inserted between the IPv6 header and the

transport header. If more than one extension header is present, the order of the headers is important and should be as detailed in [DH95] (the original specification of IPv6).

2.2.3 Transport Layer

The layer above the network layer in the TCP/IP protocol stack is the *transport layer*. This layer provides a flow of data between two hosts, for the application layer above. Two protocols are implemented at the transport layer: *TCP (Transmission Control Protocol)* and *UDP (User Datagram Protocol)*.

TCP is a reliable connection-oriented [2] protocol that allows bytes streams from a host to be delivered without error to any other hosts on networks that are reachable. TCP protocol also breaks up packets it received from the application layer into appropriate fragment sizes, acknowledges received packets, re-assembles packet fragments it received from the network layer, and perform flow control to ensure that a fast hosts does not swamp a slower host with more packets that it can handle. See RFC 793 [Pos81c] (the original specification document for TCP) for further detail on this protocol.

UDP, unlike TCP, offers an unreliable connectionless [3] service. It does not perform any error checking or flow control; It just sends datagram from one host to another and does not provide any guarantee that the datagram will get to its destination. There is much less overhead involves in processing UDP datagrams compared to TCP packets. Consequently, the throughput for UDP datagrams is usually greater than that of TCP. UDP protocol is usually use by applications for which prompt delivery is more important than accurate delivery; such as in the transmission of speech or video. RFC 768 [Pos80] outlined the original specification of UDP.

[2] Section 2.2.2 on page 12

[3] Section 2.2.2 on page 11

2.2.4 Application Layer

The top of the TCP/IP protocol stack is the *application layer*. This layer includes all processes that use the transport layer protocols to deliver data. Application layer protocols usually provide services to the users of the system. There are many application protocols. The ones that just about all TCP/IP implementation provides include:

- Telnet: The Network Terminal Protocol, which provides remote login.
- FTP: File Transfer Protocol, which is used for file transfer to or from remote hosts.
- SMTP: Simple Mail Transfer Protocol, which delivers electronic mail.
- SNMP: Simple Network Management Protocol, which is used for monitoring network segments.
- DNS: Domain Name Service, which is used for mapping host names onto their IP addresses.
- HTTP: Hypertext Transfer Protocol, which is used for fetching web pages on the World Wide Web.

Chapter 3

Cryptographic Techniques

3.1 The Data Encryption Standard

In 1973 the USA National Bureau of Standards (NBS), currently known as National Institute of Standards and Technology (NIST), issued a request for proposal for a national cryptosystem standard. A number of cipher systems was proposed. After a review of the proposals NBS adopted the cryptosystem developed at IBM as the Data Encryption Standard (DES) on July 1977. This cipher system is based on a cryptographic algorithm called LUCIFER [Fie73] that a research team at IBM—headed by Horst Feistel—developed in 1971.

DES became the most widely used cipher system in the world. It was reaffirmed as national standard in 1983, 1988, 1993 and on October 25, 1999 NIST adopted triple DES—a more secure variant of DES—as the national standard.

A complete description of DES is given in Federal Information Processing Standards Publication 46-3 (FIPS PUB 46-3) [NIST99]. DES takes a 64-bit block of plaintext x and a 56-bit key K and output 64-bit ciphertext c. The DES algorithm proceeds in three stages:

1. The 64-bit block plaintext x is first ran through to an initial permutation function IP which gives a 64-bit output x_0. We can represent this as $x_0 = IP(x) = L_0R_0$, where L_0 represents the the first 32 bits of x_0, and R_0 represents the last

32 bits. The permutation function IP is shown in Table 3.1(a). It is interpreted as follows: the first three bits of the output from this function are the 58th, 50th and 42nd bits of the input to the this function; similarly, the 62nd, 63rd and 64th bits of the output are the 23rd, 15th and 7th bits respectively of the input bit string.

2. x_0 is then subjected to 16 iterations of key-dependent computations involving a cipher function f and a key scheduling function KS. If we represents the output from each iteration as $x_i = L_i R_i$, where $1 \leq i \leq 16$, then
   $L_i = R_{i-1}$
   $R_i = L_i \oplus f(R_{i-1}, K_i)$
   where $\oplus$ denotes the exclusive-or of two bit strings. The K_i's are 48-bit blocks which are derived from the original 56-bit key using the key scheduling function KS. The key scheduling function, the derivation of the K_i's, and the cipher function f will be discussed later.

3. The inverse permutation function IP^{-1} is then applied to $R_{16}L_{16}$ to give a 64-bit block ciphertext c. that is, $c = IP^{-1}(R_{16}L_{16})$. Note the change in order of R_{16} and L_{16}. The inverse permutation function is shown in Table 3.1(b). It is the inverse of IP: if it is applied to the output of IP, the result would be identical to the bit string inputted to IP; that is, $IP^{-1}(IP(x)) = 1$. Figure 3.1 illustrates the DES algorithm.

The *cipher function* f involves the following steps:

- The R_i's is subjected to an expansion permutation E which takes as its input the 32-bit block R_i and yields a 48-bit block output. The expansion permutation is shown in Table 3.2(a). It is interpreted as follows: The first three bits of the 48-bit output $E(R_i)$ are the bits in positions 32, 1 and 2 of R_i; whereas, the last 3 bits of the output are the bits in positions 31, 32 and 1 of R_i.

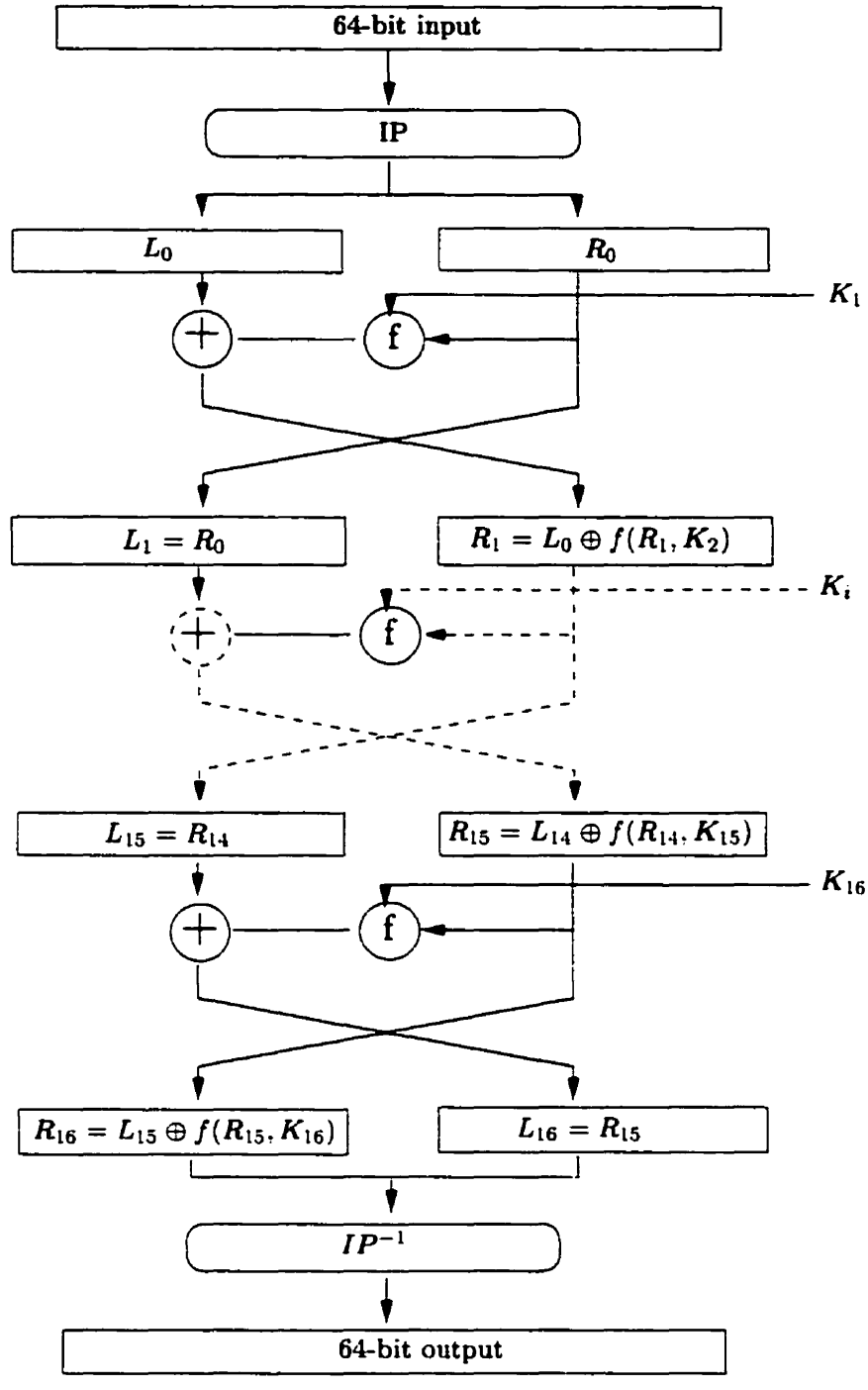

Figure 3.1: Illustration of DES encryption algorithm.

Table 3.1: Initial permutation (IP) and inverse permutation IP^{-1} functions.

(a)				IP			
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

(b)				IP^{-1}			
40	8	18	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	15	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

- The 48-bit output $E(R_i)$ of the expansion permutation is exclusive-or with the 48-bit K_i.

- The 48-bit output that resulted from the $E(R_i) \oplus K_i$ operation is broken up into eight 6-bit blocks and each block is passed through a S-box which gives an output of length 4 bits. The eight S-boxes are shown in Table 3.4. The permutation of the S-boxes can be described as follows: The first and the last bits of a 6-bit input of a given S-box S_i forms a 2-bit binary number to select

Table 3.2: Expansion function and permutation function P.

(a)	E bit-selection table				
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

(d)	P		
16	7 20		21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

one of the four rows in S_i; whereas, the inner four bits form a binary number in the range 0 to 15 to select one of the 16 columns in S_i. For example, if an input to S_1 is 101011. The 2-bit binary number obtained from the first and the last bits is 11, the decimal equivalent is 3; therefore, the row 3 is selected. The inner four bits are 0101, the decimal equivalent is 10; hence, column 10. The number in the 3rd row and 10th column of S_1 is 9; therefore, the output from S_1 for this example is 1001, which is the 4-bit binary representation of 9.

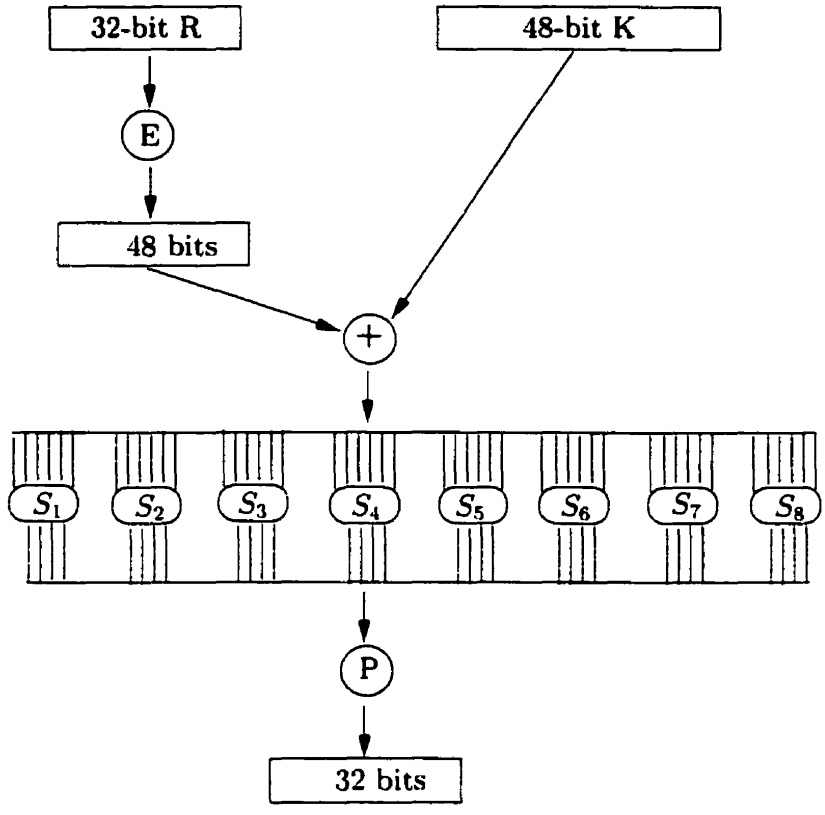

Figure 3.2: Illustration of $f(R, K)$ calculation.

- The 4-bit output of each the eight S-boxes is concatenated as shown in Figure 3.2 to yield a 32-bit output which is then fed into a permutation function P. Finally, P yields an output of 32 bits which is the result of $f(R_i, K_i)$. The permutation function P is shown in Table 3.2(b) and Figure 3.2 illustrates the computation of $f(R, K)$.

The *key scheduling function* KS is used to generate the 48-bit K_i's from the 56-bit original key. Actually, DES keys are 64 bits in length. However, 8 of the bits are used for error detection: the bits in positions 8, 16, 24,..., 64 are used for assuring that each byte has an odd parity, that is, each byte has an odd number of 1's. KS involves two permutation functions: permutation choice one (PC-1) and permutation choice two (PC-2). The functions are shown in Table 3.4. The algorithm for determing the K_i's where, $1 \leq i \leq 16$, can be described as follows:

Table 3.3: Tables used for DES key schedule calculation.

(a) PC-1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	63	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

(b)	Schedule of left shifts															
Iteration number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
left shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

(c) PC-2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
4	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

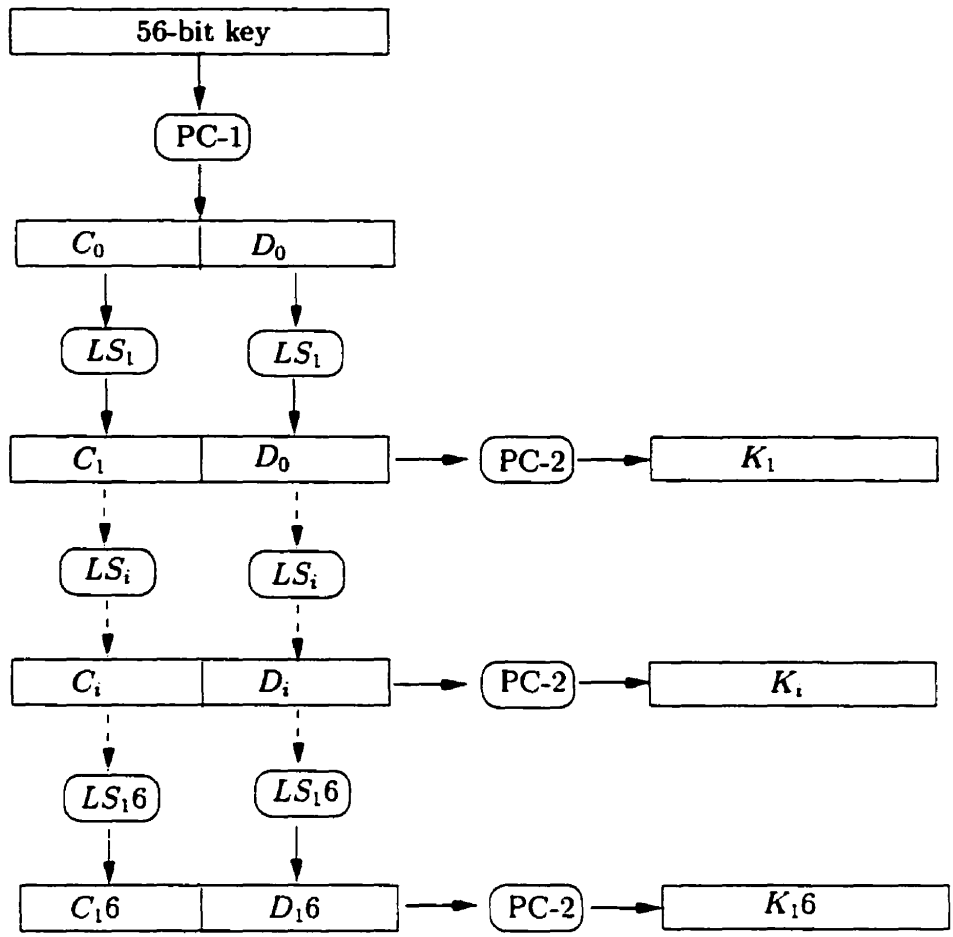

Figure 3.3: Illustration of DES key schedule calculation.

1. Given a 64-bit key K, discards the 8 parity bits and apply the the fixed permutation PC-1 to the remaining 56 bits of K. We can represents this as PC-1(K) = C_0D_0, where C_0 and D_0 represent the first and last 28 bits of K, respectively. In Table 3.4(a), PC-1 is divided into two halves. The first half determines the bits in C_0 and the second half determines the bits in D_0.

2. Compute C_i and D_i, such that

   $C_i = LS_i(C_{i-1})$

   $D_i = LS_i(D_{i-1})$,

   where LS_i is either 1 or 2 and it represents the number of cyclic left shifts that the bits in C_i or D_i are to be shifted by. Table 3.4(c) shows the LS_i's for the 16 iterations.

3. Concatenate the bits of C_i and D_i and apply the fixed permutation PC-2 to the result. The output from PC-2 is the key K_i, ie K_i =PC-2(C_iD_i). The key schedule computations are illustrated in Figure 3.3.

Table 3.4: DES eight S-boxers.

S_1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	14	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

The deciphering of DES utilizes the same key and algorithm as that which is used for enciphering, except that the algorithm is applied in the reverse order.

3.1.1 DES Mode of Operation

The specification of DES specified four possible modes of operation. A brief description of each of the four modes is given below.

- *Electronic Codebook (ECB) mode*: This is the direct application of the DES algorithm as described in the pervious section: given a bitstring of plaintext $x = x_1x_2..$, break up the bitstring in blocks of 64 bits x_i's, and apply the DES algorithm to each x_i to produce a block of ciphertext, c_i. We write $c_i = e_k(x_i)$, where e_k is the DES encipering algorithm. The complete ciphertext c is the concatenation of the c_i's in numerical order. That is, $c = c_1c_2...$

- *Cipher Block Chaining (CBC) mode*: Given a bitstring of plaintext $x = x_1x_2....$ where the x_i's are 64-bit blocks; the first block x_1 is first exclusive-ored with a 64-bit initial vector IV, then the DES encryption algorithm is applied to the result to give the first block of 64-bit ciphertext c_1. The consequent blocks of plaintext x_i's are then exclusive-ored with the previous block of ciphertext c_{i-1} and the encryption algorithm applied. This can be represented as:
  $c_1 = e_k(x_1 \oplus IV)$, $c_2 = e_k(x_2 \oplus c_1)$ and $c_i = e_k(x_i \oplus c_{i-1})$

- *Cipher Feedback (CFB) mode*: This mode uses previosly generated ciphertext as input to the DES algorithm to generate pseudo-random outputs which are exclusive-ored with the plaintext to produce the ciphertext. The steps can be described as follows: apply DES algorithm to an initial vector IV to produce a cipher output z_1, then exclusive-or x_1 with z_1 to produce the first block of ciphertext c_1. The subsequent blocks of ciphertext c_i's are produced by applying the DES algorithm to the previous block of ciphertext, then exclusive-or the output with the corresponding block of plaintext x_i, ie
  $z_1 = e_k(IV)$, $c_1 = x_1 \oplus z_1$, $z_2 = e_k(c_1)$, $c_2 = x_2 \oplus z_2$, $z_i = e_k(c_{i-1})$ and $c_i = x_i \oplus z_i$

- *Output Feedback (OFB) mode*: This mode is similar to the CFB mode, except that OFB does not chain the ciphertext; instead, the initial vector IV is encrypted in turn to produce the z_i's. So,
  $z_1 = e_k(IV)$, $c_1 = x_1 \oplus z_1$, $z_i = e_k(z_{i-1})$ and $c_i = x_i \oplus z_i$

3.1.2 DES implementation

The popularity of DES can be attributed to the fact that the algorithm can be implemented very efficently in either software and hardware. The only computations that are involved in the DES encipher or deciphering are exclusive-or operations. The permutation and expansion functions can be implemented using lookup tables; therefore, these operations can all be performed in constant time. This contrasts with public-key cryptographic algorithms which usually involve computationly intensive operation such as the exponentation of large numbers. DES enciphering and deciphering can therefore be performed much more quickly than all of the current public-key encryption algorithm. As a result, DES or variants of the DES algorithm have enjoyed widespread use, particularly in applications where the speed of encrypting and decrypting the data is of much importance.

3.1.3 Triple DES

Given a 64-bit plaintext x and the corresponding 64-bit ciphertext c that resulted from enciphering x with DES, the 56-bit DES key K can be found within 2^{56} operations. As the speed of computing systems increases it becomes more feasible to perform large number of operations within limited time periods. In 1997, RSA laboratories issued a challenge which involved a reward of $10,000 to find the DES key of a ciphertext which was perceded by a known block of text which contained the the phrase "the unknown message is:". A project, headed by Roche Verse—an independent consultant—which involved over 70,000 computer systems linked over the Internet used a brute-force [1] program to find the correct DES key in approximatly 96 days [RSA97]. This successful attack on DES accelerated the search for a more secure replacement of DES. In October 25, 1999, NIST affirmed triple DES as the new data encryption standard for the Federal Bureau.

The triple DES algorithm can be described as follows: Let $e_k(x)$ and $d_k(x)$ represents the encryption and decryption of the 64-bit bitstring x using the DES algorithm

[1] a brute-force attack involves trying all possible keys until the correct one is found.

with key K, then the 64-bit ciphertext x is obtained by performing the following operation:

$$c = e_{K_3}(d_{K_2}(e_{K_1}(x)))$$

Where K_1, K_2 and K_3 are 56-bit DES keys.

The deciphering of triple DES to derive the plaintext x from the ciphertext c is the reverse of the enciphering process and it can be described as:

$$x = d_{K_1}(e_{K_2}(d_{K_3}(c)))$$

For greater security, the three keys should all be different; this in essence corresponds to enciphering with a key of length 168 bits—which should be relatively secure against brute-force attack for many years. For data requiring a lesser degree of security, K_1 can be equal to K_3. In this case, the key is effectively of length 112 bits.

3.1.4 Other Private Key Cryptosystems

Over the years, a number of private key cryptosystems have been proposed as possible replacement for DES. Some of these cryptosysems will be introduced below.

- **Blowfish**: Blowfish was developed by Bruce Schneier [Sch93]. It uses a key of variable length: from 32 to 448 bits to encrypt blocks of 64-bit plaintext into blocks of 64-bit ciphertext. Blowfish has a simple structure which makes it easy to implement. Currently, Blowfish is one of the fastest encryption system. The variable length key allows for varied degree of security. It is believed that blowfish is very secure, particularly when longer keys are used.

- **CAST-128** CAST was developed by Carlisle Adams and Stafford Tavares [Ada97]. It uses a key that varies for 40 to 128 bits in length, in 8-bits increments; and produce 64-bit blocks ciphertext from blocks of 64-bit plaintext. The CAST algorithm involves 16 rounds of computations which are quite complex. This algorithm has received widespread review and it is believed to be quite safe.

- **International Data Encryption Algorithm (IDEA)**: IDEA was developed by Xuejia Lai and James Massey at the Swiss Institute of Technology. The original specifications and later modifications are outlined in [LM90] and [LM91] respectively. IDEA uses a 128-bit key and and encrypts plaintext in blocks of 64 bits and output 64-bits block ciphertext. IDEA enciphering algorithm has received a fair bit of attention and it is believed that it should to be quite secure for a number of years.

- **RC2** RC2 was developed by Ronald Rivest [Riv98]. It is was designed to be easily implement on 16-bit microprocessors. The algorithm uses a key of variable length: from 8 bytes to 128 bytes and enciphers blocks of 64-bit plaintext and outputs blocks of ciphertext of length 64 bits.

- **RC5** RC5 was developed by Ronald Rivest [Riv94]; a later description is given in [BR96]. RC5, like DES is suitable for both hardware and software implementation. It is adaptable to processor of different word length: the number of bits in the processor's word is one of the input to the RC5 algorithm. RC5 algorithm also has variable number of rounds, which allows tradeoff between higher speed and higher security. The algorithm uses a variable length key to encrypt blocks of 32, 64 or 128 bits plaintext and output blocks of ciphertext of corresponding length. RC5 has a low memory requirement; which therefore makes it suitable to be used in applications such as smart cards which have limited memory.

3.2 Public Key Cryptosystems

Private-key cryptosystems suffer from a major drawback. These cryptosystems are *symmetric*, that is, the same key is use for encrypting and decrypting the data; therefore, the sender and receipent of the message need to exchange the "secret" cryptographic key via a relatively secure channel, which may not be available. *Public-key* cryptosystems address this issue. These systems are *asymmetric* systems: the key that is used to decipher the data is different from that which is used to encipher

it. The encrypting keys—commonly refered to as the public keys—do not need to be kept secret. This therefore eliminates the need for a secure channel to exchange keys.

The ideas on which public-key cryptosystems are based was first published in 1976 by Whitfield Diffie and Martin Hellman [DH76]. These systems are based on the concept of trapdoor one-way functions. One-way functions are functions that are easy to compute but hard to invert. Whereas, trapdoor one-way functions are one-way functions which can be inverted easily with the knowledge of some additional information; this additional information is refered to as the trapdoor.

Over the years a number of public-key cryptosystems have been proposed; three of the more commonly used ones will be presented in this section.

3.2.1 RSA Cryptosystem

The RSA cryptosystem was developed in 1977 by Ronald Rivest, Adi Shamir and Len Adlemar at MIT [RSA78]. This system is based on the difficulty of factoring large integers. The RSA system can be described as follows:

1. Generate two large primes p and q.
2. Compute the product of the primes $n = pq$.
3. Compute the number of integers that are less than n and relatively prime [2] to n. This is equal to the Euler phi function $\phi(n) = (p-1)(q-1)$.
4. Select a random number b such that $1 < b < \phi(n)$ and b is relatively prime to n, ie $gcd(b, \phi(n)) = 1$.
5. Compute $a = b^{-1} \bmod \phi(n)$.
6. Keep a, p and q secret and make n and b available to any one who wishes to send you encrypted messages.

The primes p and q should be large enough such that, given their product n, it should be computationally infeasible to factor n without prior knowledge of either p

[2] Two numbers a and b are relatively prime if $gcd(a, b) = 1$.

or q. For secure systems it is recommended that n be at least 200 digits which is equivalent to $200 \times log_2\ 10 \simeq 664$ bits in length. We will now turn to to question of how are these primes of suitable size generated. Currently, the most practical way of doing this is use a *pseudo-random number generator* to generate sufficiently large add numbers, then use a *probabilistic primality testing* algorithm to determine if the number is a prime. A commonly used algorithm that tests for prime is the *Miller-Rabin* primality testing algorithm. This algorithm along with other probabilistic primality testing algorithms are outlined in [Sti95].

The Encryption Decryption Processes

Plaintext are encrypted in blocks. Each block should be less than the binary representation of n in bits, that is, each block should be less than $log_2\ n$ bits in length. For a block of plaintext x, the public keys (b, n) are used to generate the corresponding block of ciphertext c by computing: $c = x^b \bmod n$. The block of plaintext x can then be generated from the block of ciphertext c by calculating: $x = c^a \bmod n$.

3.2.2 ElGamal Cryptosystem

The ElGamal cryptosystem was developed by ElGamal and published in 1985 in [ElG85]. This system is based on the *Discrete Logarithm* problem. The Discrete Logarithm problem can be described as follows: Consider the equation $\beta = g^x \bmod p$, if p is carefully chosen it is considered to be very difficult to compute the values of x given β, g and p. However, β can be computed quite efficiently if g, x and p are given. In other words, exponentiation modulo p is a one-way function for suitable p.

We will now describe the ElGamal system. First, a suitable prime p is choose such that the Discrete Logarithm problem is difficult for integers less that p. Then suitable β, g and a are chosen such that $\beta \equiv g^a \bmod p$. β, g and p are then made public and a kept private.

Encryption and Decryption Processes

To encipher a plaintext x, Alice chooses a secret random integer k such that $k < (p-1)$; then if e_K is the encryption process, we write:

$$e_K(x,k) = (y_1, y_2)$$

$$\text{where } y_1 = g^k \bmod p$$

$$\text{and } y_2 = x\beta^k \bmod p$$

Note that since the ciphertext depends on the value of k that Alice chooses, a plaintext x can be encrypted into many different ciphertext; therefore, the ElGamal cryptosystem is non-deterministic.

If d_K is the decryption process, we write:

$$d_K(y_1, y_2) = y_2(y_1^a)^{-1} \bmod p$$

In simple terms, a plaintext x is "masked" by multiplying it by β^k; the product y_2 and g^k are then sent to Bob—the holder of the private key. Bob can then use his private key a to compute β^k from g^k; finally, the plaintext x can be determined by dividing y_2 by the "mask", β^k.

3.2.3 Diffie-Hellman Key Exchange

The Diffie-Hellman Key Exchange was the first published public-key cryptosystem. It was published in 1976 in [DH76]. This key exchange protocol is base on the *Discrete Logarithm* problem. See Section 3.2.2 for the discussion on the Discrete Logarithm problem. If Alice and Bob wish to exchange a secret key over an insecure channel using the Difie-Hellman Key Exchange protocol, they would proceed as follows:

1. Alice and Bob decide on a suitable prime p and an integer α such that α is a primitive root of p; α and p can be make public.

2. Alice chooses a secret integer a_A and compute $y_A = \alpha^{a_A} \bmod p$
   and sends y_A to Bob.

3. Bob chooses a secret integer a_B and compute $y_B = \alpha^{a_B} \bmod p$
   and sends y_B to Alice. y_A and y_B are refer to as Diffie-Hellman public values.

4. Alice generates the secret key K by computing $K = (y_B)^{a_A} \bmod p$.

5. Bob generates the secret key K by computing $K = (y_A)^{a_B} \bmod p$.

Alice and Bob will generate the identical key K since:

$$\begin{aligned} K &= (y_B)^{a_A} \bmod p = (\alpha^{a_B} \bmod p)^{a_A} \bmod p \\ &= (\alpha^{a_B})^{a_A} \bmod p = \alpha^{a_B a_A} \bmod p \\ &= (\alpha^{a_A})^{a_B} \bmod p = (\alpha^{a_A} \bmod p)^{a_B} \bmod p \\ &= (y_A)^{a_B} \bmod p \end{aligned}$$

The security of Diffie-Hellman Key Exchange is base on the assumption that it is computationally infeasible to compute a_A or a_B from y_A or y_B and α; since this is equivalent to solving the Discrete Logarithm problem.

Man-in-the-middle Attack

The Diffie-Hellman Key Exchange is vulnerable to a *man-in-the-middle attack* where an intruder—let's call her Eve—intercepted messages, modified them and sent the modified messages to Bob and Alice as illustrated in Figure 3.4. Alice thinks that she is using Bob's y_B to generate the key; whereas, she is actually using Eve's value y_B'. Similarly, Bob thinks that he is using Alice's y_A to generate the key; whereas, he is actually using Eve's value y_A'. Hence, the the key that Alice and Bob generate will—unknowingly by both parties—be shared by Eve. The man-in-the-middle attack can be thwarted by using an authenticated Diffie-Hellman Key exchange.

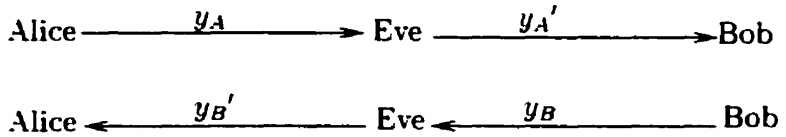

Figure 3.4: Illustration of man-in-the-middle attack.

Authenticated Diffie-Hellman Key Exchange

A possible modification to the Diffie-Hellman Key Exchange that can foil the man-in-the-middle attack is the use of digital signature [3] to sign and authenticate the integers y_A and y_B that Alice and Bob exchange. This scheme is similar to the Diffie-Hellman exchange, except that Alice signs y_A using a digital signature algorithm and generates the signature $sig(y_A)$; she then sends y_A and $sig(y_A)$ to Bob. Bob can then use Alice's public verification algorithm to determine whether $sig(y_A)$ is indeed Alice's signature for y_A. Bob, similarly signs y_B and sends both y_B and $sig(y_B)$ to Alice who will be able to use Bob's public verification algorithm to ascertain whether or not $sig(y_B)$ is Bob's signature for y_B. Both parties will therefore be able to determine whether or not the messages that they received from each other have been tampered with. Thus, Alice or Bob will not be fooled into receiving components for the generation of the key from Eve, thinking that it came from Bob or Alice respectively.

3.2.4 Other Public-key Cryptosystems

Over the years a number of other public-key cryptosystems have been proposed. Some of the more popular ones are base on:

- **Elliptic Curve**: These cryptosystems are based on the difficulty of solving problems involving elliptic curves. These systems are becoming very popular because, compared to other public-key cryptosystems, they appear to offer equal security for much smaller key size.

- **Knapsack Problem**: These cryptosystems are base on the *Subset Sum* problem, which is a member of the group of a large class of problems called *NP-complete*, for which there are no known polynomial-time algorithm to solve them.

For further information on these as well as other public-key cryptosystem, two good source of reference are: [Sti95] and [Sta99].

[3] Digital signature will be discussed in Section 3.4.

3.2.5 Comparison of Private and Public-key Cryptosystems

Public-key cryptosystems such as RSA—as discussed Section 3.2—resolve the problem of having to use a secure channel or a key exchange protocol to exchange secrete keys. However, compared to private-key cryptosystems, public-key systems encrypt data much more slowly than the the former. This is so, since public-key cryptosystems usually involve modular exponential calculation involving large integers—which can be quite computationally intensive. Whereas, the computations involve in private-key systems are usually primitive operations such as exclusive-or of bit stings, which can be done much more quickly. If we compare speed of encryption of RSA using a 512 bit modulo p, RSA is 1500 times slower than DES [Sti95]. As a consequence of this big difference in the speed of encryption of private-key cryptosystems compared to public-key systems, applications—such as IPSec—which require high throughput, currently, almost exclusively, use private-key cryptosystems for enciphering of data. On the other hand, digital signature applications as well as other applications that do not necessarily require high throughput, mainly utilize public-key cryptosystems.

3.3 Hash Functions and MAC

A *cryptographic hash function* is an algorithm that takes a message x of any arbitrary length and produces a fixed length output $h(x)$, called message digest or fingerprint. Hash functions are used to verify the integrity of messages or files. For example, a message digest can be generated for a binary file and compare the digest with that generated for the same file on a secure site or medium; if the file has not been tampered with, the digests will be identical. A another common use of hash functions is with digit signature: the message to be digitally signed is first hashed then the fixed length message digest ran through the digital signature algorithm. This will be discussed further in Section 3.4.

If an hash function is to be useful cryptographically, it must be *strongly collision-free*. That is, it should be computationally infeasible to find two messages x and x' such that $x \neq x'$ and $h(x) = h(x')$. Two hash functions, MD5 and SHA-1 will

be discussed in this section; we are also going to present an example of a *Message Authentication Codes (MAC)*. MAC are similar to hash functions, except that a secret key is required to produce the fixed length message digest.

3.3.1 MD5 Hash Function

The MD5 message digest algorithm [Riv92b] was developed by Ronald Rivest at MIT. It is an extension of an earlier version called MD4 [Riv92a]. MD5 takes a message x of arbitrary length and output a 128-bit message digest $h(x)$. The MD5 algorithm consists of 5 steps. A description of these steps follows.

- *Step 1: Append Padding Bits*

  The message x is padded by adding a single "1" bit followed by an appropriate number of "0" bits, such that the length of the message $|x| \equiv 448 \bmod 512$. That is, the message is extended such that the length of the padded message is 64 bits less that a multiple of 512. Padded bit(s) is/are always added, even if the original message is already 64 bits less than a multiple of 512 in length. So, the number of padded bits added is between 1 and 512 inclusively.

- *Step 2: Append Length*

  The 64-bit representation of the length of the original message is appended to the padded message. If the length of the original message is greater than 2^{64}, then the lower order 64 bits are appended to the message instead. At this point, the message has a length that is an exact multiple of 512 and consequently it is divisible by 16. Let M[0 1 ... n-1] represents the words of the resulting message, where n is divisible by 16.

- *Step 3: Initialize MD buffer*

  a 128-bit buffer is used to compute the message digest. The buffer can be represented by four 32-bits registers A, B, C and D. These registers are initialized to the following hexadecimal values:

  A: 01 23 45 67

B: 89 ab cd ef

C: fe dc ba 98

D: 76 54 32 10

The values are stored in *little-endian* byte format, that is, lower order bytes first.

- *Step 4: Process Message in 16-word (512-bits) Blocks*

  Let X represents a 16-word block of the message. If $X[i]$ and $M[i]$ denotes the word at index i in the 16-word block and n-word message respectively, then process the message according to the follows algorithm.

  For $i = 0$ to $n/16 - 1$ do

  For $j = 0$ to 15 do

  set $X[j]$ to $M[i * 16 + j]$.

  end /* of loop j */

  $A \wedge A = A$

  $B \wedge B = B$

  $C \wedge C = C$

  $D \wedge D = D$

  Round1

  Round2

  Round3

  Round4

  $A = (A + AA) \bmod 2^{32}$

  $B = (B + BB) \bmod 2^{32}$

  $C = (C + CC) \bmod 2^{32}$

  $D = (D + DD) \bmod 2^{32}$

  end /* of loop i */

  The four rounds indicated in the algorithm utilize four auxiliary functions and a 64-element table $T[1...64]$, where $T[i] = 2^{32} \times abs(\sin(i))$, where i is the index in the table and $T[i]$ is element at index i. The auxiliary functions each takes

three 32-bit words as input and output a 32-bit word. The functions are defined below.

$$
\begin{aligned}
F(x,y,z) &= x \wedge y \vee ((\neg x) \wedge z) \\
G(x,y,z) &= x \wedge z \vee (y \vee \neg z) \\
H(x,y,z) &= x \oplus y \oplus z) \\
I(x,y,z) &= y \oplus (x \vee \neg z)
\end{aligned}
$$

Where $x \wedge y$ is the bit-wise "and" of x and y

$x \vee y$ is the bit-wise "or" of x and y

$x \oplus y$ is the bit-wise "exclusive-or" of x and y

$\neg x$ is the bit-wise "complement" of x

A description of the four rounds follows.

Round 1

Let $[abcd\ k\ s\ i]$ represents the operation

$a = b + ((a + F(b,c,d) + X[k] + T[i] <<< s)$.

where $Y <<< s$ represents the circular shift of Y by s positions $(0 \leq s \leq 31)$.

Do the following 16 operations.

$[ABCD\ 0\ 7\ 1]\ [DABC\ 1\ 12\ 2]\ [CDAB\ 2\ 17\ 3]\ [BCDA\ 3\ 22\ 4]$

$[ABCD\ 4\ 7\ 5]\ [DABC\ 5\ 12\ 6]\ [CDAB\ 6\ 17\ 7]\ [BCDA\ 7\ 22\ 8]$

$[ABCD\ 8\ 7\ 9]\ [DABC\ 9\ 12\ 10]\ [CDAB\ 10\ 17\ 11]\ [BCDA\ 11\ 22\ 12]$

$[ABCD\ 12\ 7\ 13]\ [DABC\ 13\ 12\ 14]\ [CDAB\ 14\ 17\ 15]\ [BCDA\ 15\ 22\ 16]$

Round 2

Let $[abcd\ k\ s\ i]$ represents the operation

$a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s)$.

Do the following 16 operations.

$[ABCD\ 1\ 5\ 17]\ [DABC\ 6\ 9\ 18]\ [CDAB\ 11\ 14\ 19]\ [BCDA\ 0\ 20\ 20]$

[*ABCD* 5 5 21] [*DABC* 10 9 22] [*CDAB* 15 14 23] [*BCDA* 4 20 24]
[*ABCD* 9 5 25] [*DABC* 14 9 26] [*CDAB* 3 14 27] [*BCDA* 8 20 28]
[*ABCD* 13 5 29] [*DABC* 2 9 30] [*CDAB* 7 14 31] [*BCDA* 12 20 32]

Round 3

Let [*abcd k s t*] represents the operation

$a = b + ((a + H(b, c, d) + X[k] + T[i]) <<< s)$.

Do the following 16 operations.

[*ABCD* 5 4 33] [*DABC* 8 11 34] [*CDAB* 11 16 35] [*BCDA* 14 23 36]
[*ABCD* 1 4 37] [*DABC* 4 11 38] [*CDAB* 7 16 39] [*BCDA* 10 23 40]
[*ABCD* 13 4 41] [*DABC* 0 11 42] [*CDAB* 3 16 43] [*BCDA* 6 23 44]
[*ABCD* 9 4 45] [*DABC* 12 11 46] [*CDAB* 15 16 47] [*BCDA* 2 23 48]

Round 4

Let [*abcd k s t*] represents the operation

$a = b + ((a + I(b, c, d) + X[k] + T[i]) <<< s)$.

Do the following 16 operations.

[*ABCD* 0 6 49] [*DABC* 7 10 50] [*CDAB* 14 15 51] [*BCDA* 5 21 52]
[*ABCD* 12 6 53] [*DABC* 3 10 54] [*CDAB* 10 15 55] [*BCDA* 1 21 56]
[*ABCD* 8 6 57] [*DABC* 15 10 58] [*CDAB* 6 15 59] [*BCDA* 13 21 60]
[*ABCD* 4 6 61] [*DABC* 11 10 62] [*CDAB* 2 15 63] [*BCDA* 9 21 64]

- *Step 5: Output*

  The 128-bit message digest is the content of the four registers: A, B, C and D appended in little-endian byte format, that is, starts with the lower byte of A and ends with the higher byte of D.

The computations involved in this algorithm are all primitive operations; also, the algorithm does not require any large lookup table; as a result, the MD5 algorithm is quite fast on a 32-bit processor.

3.3.2 Secure Hash Algorithm (SHA-1)

The Secure Hash Algorithm (SHA) was developed by the National Institute of Standard and Technology (NIST) and adopted as the Secure Hash Standard in 1993. The revised version of the the algorithm, SHA-1, was issued as Federal Information Processing Standards Publication 180-1 (FIPS PUB 180-1) [NIST95] in 1995. SHA-1 takes a message of length $< 2^{64}$ bits and output a 160-bit message digest. The design of the SHA-1 algorithm is similar to that of MD5. A description of the five steps involve in the processing of the SHA-1 message digest follows. The first two steps are the same as Steps 1 and 2 in for the MD5 algorithm (see Section 3.3.1).

- *Step 3: Initialize buffers*

  The SHA-1 computation uses three buffers: two buffers each consists of five 32-bit registers and a sequence of eighty 32-bit words; and a single word buffer TEMP. The words of the 80-word sequence are depicted as W_0 , ..., W_{79}. The five registers of the first buffer are labeled A, B, C, D and E; whereas those of the second buffer are designated H_0, H_1, H_2, H_3 and H_4. A, B, C, D and E are initialized to the following hexadecimal values:

  A: 67 45 23 01

  B: ef cd ab 89

  C: 98 ba dc fe

  D: 10 32 54 76

  E: c3 d2 e1 f0

  the initial values of A, B, C, and D are the same as those for the four MD5 registers except that for SHA-1, the values are stored in *big-endian* byte format, that is, higher byte order first; whereas, the values are stored in little-endian byte format for MD5.

- *Step 4: Process message in 16-word (512-bit) blocks*

  SHA-1 uses a sequence of logical functions f_0, f_1, ..., f_{79}. Each of the f_i's takes

three 32-bit words as input and output a 32-bit word. The f_i's are defined as follows:

$$\begin{aligned} f_i(x,y,z) &= (x \wedge y) \vee (\neg y \wedge z) \quad (0 \leq i \leq 19) \\ f_i(x,y,z) &= x \oplus y \oplus z \quad (20 \leq i \leq 39) \\ f_i(x,y,z) &= (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) \quad (40 \leq i \leq 59) \\ f_i(x,y,z) &= x \oplus y \oplus z \quad (60 \leq i \leq 79) \end{aligned}$$

SHA-1 also uses a sequence of constant words K_0, K_1, ... K_{79}. The hexadecimal values for these words are shown below.

$$\begin{aligned} K_i &= 5a827999 \quad (0 \leq i \leq 19) \\ K_i &= 6ed9eba1 \quad (20 \leq i \leq 39) \\ K_i &= 8f1bbcdc \quad (40 \leq i \leq 59) \\ K_i &= ca62c1d6 \quad (60 \leq i \leq 79) \end{aligned}$$

Before we proceed, it is necessary to define some terms.

$X + Y$ represents the addition of modulo 32 of two bit strings, that is,

$X + Y = X + Y \bmod 32$

$S^n(X)$ represents the circular left shift of the bit string X by n bits $(0 \leq n \leq 32)$ that is, $S^n(X) = (X << n) \vee (X >> 32 - n)$; where $<<$ and $>>$ represents left and right shift respectively.

Let M_1, M_2, ..., M_n (n is an integer divisible by 16) represents 16-word block sequences of the message that resulted from Step 2. Process each M_i as follows:

1. Divide M_i into 16 words W_0, W_1, ... W_{15}, where W_0 is the left-most word.
2. for $t = 16$ to 79 do

   let $W_t = S^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$
3. Let $H_0 = A$, $H_1 = B$, $H_2 = C$, $H_3 = D$ and $H_4 = E$

4. for $t = 0$ to 79 do

$$TEMP = S^5(H_0) + f_t(H_1, H_2, H_3) + H_4 + W_t + K_t$$

$$H_4 = H_3 \quad H_3 = H_2 \quad H_2 = S^{30}(H_1) \quad H_1 = H_0 \quad H_0 = TEMP$$

let $A = A + H_0 \quad B = B + H_1 \quad C = C + H_2 \quad D = D + H_3 \quad E = E + H_4$

- *Step 5: Output*

  The 160-bit message digest is the content of 5 words A, B, C, D and E appended in big-endian byte format.

SHA-1 executes more slowly than MD5 since more steps are involved in producing the 160-bit SHA-1 message digest compare to the number of steps executed in the processing of MD5 128-bit message digest. However, the additional 32 bits in length of the message digest of SHA-1 makes it less susceptable to brute force attack than MD5.

3.3.3 HMAC

Message Authentication Codes (MAC) are similar to hash functions (Section 3.3), excepts that a secret key is required to produce the message digest. MAC's are typically used to validate information transmitted between two parties that share a secret key. HMAC are algorithms that perform message authentication using cryptographic hash functions. HMAC can be used in combination with any iterated cryptographic hash function such as MD5 and SHA-1, without modification of the hash function. A brief description of HMAC follows.

Let H denotes a cryptographic hash function,

K denotes a secret key,

B represents the byte length of the block that is used to compute the message, digest (64 bytes for MD5 and SHA-1),

L denotes the byte length of the message digest,

ipad represents the byte 0x36 repeated B times and

opad represents the byte 0x5c repeated B times.

K can be of any length up to B bytes; however, it is recommended that K be not

shorter than L in length. Applications that use key with length greater than B need to hash the key using H then use the resulting L-byte string as the actual key to the HMAC algorithm.

To compute HMAC of a data 'text' do the following:

1. Append zeroes to the end of K to create a B byte string.
2. Exclusive-or the B-byte string obtained from step 1 with ipad.
3. Append the data stream 'text' to the B-byte string resulted from step 2.
4. Apply H to bit string generated in step 3.
5. Exclusive-or the B-byte bit string generated in step 1 with opad.
6. Append the message digest that resulted from step 4 to the B-byte string that resulted in step 5.
7. Apply H to the bit sting generated in step 6 and output the result.

The above can be represented as $H(K\oplus, H(K \oplus ipad, text))$.
The security of HMAC depends on the hash function H that is used. If H is strongly collision-free, then the HMAC will also be strongly collision-free. For further discussion on the security and design of HMAC refer to [KBC97] which contains the original specifications of HMAC.

3.4 Digital Signatures

Digital signatures schemes are methods that are used to sign messages stored in electronic from, which can then be transmitted over a computer network. A signature scheme consist of a signing algorithm and a verification algorithm. The signing algorithm utilizes a private key to generate the signature $sig(m)$ for a messsage m; the pair $(m, sig(m))$ is then sent to the recipient. The verification algorithm, on the other hand, uses the signatory public keys and takes the pair (m, y) as input and return true if $y = sig(m)$ and false if $y \neq sig(m)$. Digital signatures can therefore

be used to detect unauthorized modifications of data and authenticate the identity of the signatory.

3.4.1 Digital Signature Standard

The National Institute of Standards and Technology (NIST) adopted a signature scheme—known as the Digital Signature Algorithm (DSA)—as the Digital Signature Standard (DSS) in December 1994. The DSS specification was revised in 1998 and published as FIPS PUB 186-1 [NIST98] in December 15, 1998. FIPS PUB 186-1 specified that either DSA or RSA signature schemes can be used to generate digital signatures for the Federal agencies. A description of these signature schemes follows.

Digital Signature Algorithm

DSA is a modification of the the ElGamal signature scheme [ElG85] (see Section 3.2.2 for a description of the ElGamal scheme). DSA can be described as follows: Let

- p be a prime such that $2^{L-1} < p < 2^L$ for $512 \leq L \leq 1024$ and L is a multiple of 64
- q be a 160-bit prime that divides $p - 1$
- $g = h^{(p-1)/q} \bmod p$, where h is any integer with $1 < h < p - 1$, such that $h^{(p-1)/q} \bmod p > 1$; that is, g has order $q \bmod p$
- $\beta = g^a \bmod p$, where a is a random or pseudo-randomly generated integer such that $0 < a < q$
- k be a randomly or pseudo-randomly generated integer such that $0 < k < q$

The values p, q, g and β are public and a and k are private. A new k should be generated for each signature.

The signature of a message m for a given k can be defined as:

$$\begin{aligned} sig(m,k) &= (y,s) \quad \text{where} \\ y &= (g^k \bmod p) \bmod q \quad \text{and} \\ s &= (k^{-1}(\text{SHA-1}(m) + ay)) \bmod q \end{aligned}$$

The SHA-1 hash function is used to reduce the variable length message m to a 160-bit message digest which is then signed using the digital signature scheme.
Let $ver(m,y,s)$ be the verification algorithm which takes as input the message m and y and s as defined above.
The verification of a signature is done by performing the following computations:

$$\begin{aligned} d_1 &= ((\text{SHA-1}(m))s^{-1}) \bmod q \\ d_2 &= (ys^{-1}) \bmod q \\ ver(m,y,s) &= true \Leftrightarrow ((g^{d_1}\beta^{d_2}) \bmod p) \bmod q = y \end{aligned}$$

The signature for the message m is valid if and only if $ver(m,y,s)$ returns true. If $ver(m,y,s)$ returns false, then either the message m has been modified or the signature is not that of the signatory.

RSA Signature scheme

The RSA public-key cryptosystem (see Section 3.2.1) can be used to provide digital signatures. A description of the RSA signature scheme follows.
Let $n = pq$, where p and q are primes. Select two integers a and b such that $ab \equiv 1($ mod $\phi(n))$. Let the values of n and b be public and the values p, q and a be private.
The signature $sig(m)$ for a message m is generated by computing:

$$sig(m) = (h(m))^a \bmod n$$

where $h(m)$ is the message digest that resulted from hashing the message m with a cryptographic hash function such as SHA-1 or MD5.
The verification algorithm $ver(m,y)$ which takes as input the message m and the signature y, can be defined as:

$$ver(m, y) = true \Leftrightarrow h(m) \equiv y^b (\mathrm{mod} n)$$

Anyone can verify a signature because the verification algorithm uses the signatory public keys; however, only the signatory will be able to generate a valid signature since this requires the signatory private keys.

Chapter 4

IP Security Architecture

4.1 What IPSec Does

IP packets are inherently insecure. It is relative 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, or 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. IPSec is designed to address these security concerns of the IP protocol.

IPSec addresses security of datagrams at the network layer—the layer in the TCP/IP protocol stack that contains the IP protocol. The security that IPSec affords is provided by the use of a combination of cryptographic protocols and security mechanisms. IPSec enables systems to select required security protocols, select the algorithms that will be used for the security services, and generate and put in place any cryptographic keys that are required to provide the requested services(s).

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, the use of encryption to provide data confidentiality, and limited traffic flow confidentiality. The services that IPSec provides are at the network layer—layer two of the TCP/IP

protocol stack; consequently, these services can be used by any of the upper layer protocols such as TCP, UDP, ICMP, IGMP or any application layer protocol.

4.2 How IPSec Works

IPSec is designed to provide high quality, interoperable cryptographic based security for IPv4 and IPv6 datagrams. IPSec achieves these objectives by use of two traffic security protocols—Authentication Header (AH) and Encapsulation Security Payload (ESP)—and through the use of cryptographic key management procedures and protocols such as Internet Key Exchange protocol (IKE).

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. These modes of operation will be discussed in the chapters on AH and ESP. AH and ESP can be applied, in the desired mode, alone or in combination with each other. The Internet Key Exchange (IKE) protocol is used to negotiate the choices of the cryptographic algorithms to be utilized by AH and ESP, and put in place the necessary cryptographic keys that the algorithms require. AH, ESP and IKE protocols will be discussed in more detail in later chapters.

The protocols that IPSec uses are designed to be algorithm-independent. The choice of algorithms are specified in the Security Policy Database (SPD). The possible choices of algorithms that are available are dependent on the IPSec implementation; however, a standard set of default algorithms are specified by IPSec to ensure interoperability in 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 specified that data traffic that originated from certain subnet(s) should be protected with both AH and ESP and that the encryption should be done with triple-DES with three different keys; whereas, the policy might specified that data

traffic from another site should be protected with only ESP and that these traffic should be afforded encryption with DES. IPSec is able to differentiate between the security service it offers to different data traffic by the use of Security Association (SA).

4.3 Security Association

The concept of "Security Association (SA)" is fundamental to IPSec. The two protocols that IPSec uses—Authentication Header (AH) and Encapsulating Payload (ESP)—both use SA; and a principal function of Internet Key Exchange (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 traffic between two peers then two sets of SAs are required: a SA for AH and a SA for ESP. SAs that defined tunnel mode operation for AH or ESP are called tunnel mode SAs; whereas, the SAs that defined transport mode are called transport mode SAs. Security Associations are simplex, that is, they are uni-directional; 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 that is destined to a given host.

SA are negotiated between the communicating peers via key management protocols such as IKE. When the negotiation of a SA completes, both peers store the SA parameters in their Security Association Databases (SAD). One of the parameters of a SA is its lifetime, which takes the form of a time interval or a count of the number of bytes to which IPSec protocol has been applied. When the lifetime of a SA expires, this SA is either replaced by a new SA or terminated. When the SA terminates its entry is deleted from the SAD.

Security Associations are uniquely identified by a triplet consisting of a Security

Parameter Index (SPI), a destination IP address for outbound SA or a source IP address for inbound SA, and a specified protocol (example AH or ESP). The SPI is a unique 32-bit integer, which is generated and used as a unique identifier of a SA. It is transported in the AH and ESP header. 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 Security Association Database (SAD) to ascertain the SA or SA bundle that is associated with the datagram.

4.4 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 that is destined to, or originated from a given host or network. Whereas, SAD contains the active SA parameters. For both SPD and SAD, separate inbound and outbound databases are required.

4.4.1 Security Policy Database

The IPSec protocol mandates that the Security Policy Database (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 Authentication Header (AH) or the Encapsulating Security Payload (ESP) header(s) or—if IPSec is not applied to the packet—the IP header.

- Source IP address: The source 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 source IP address field of the AH or the ESP header(s) or—if IPSec is not applied to the packet—the 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 such as bert.cs.mcgill.ca, a X.500 distinguished name or a X.500 general name.

- User ID: The user ID can be a fully qualified DNS user name such as foo@cs.mcgill.ca or a X.500 distinguished name.

Each entry in the SPD consists of one or more selectors, an indication of whether the packets that match the selector(s) in the the entry should be discarded, be subject to IPSec processing or not be subject to IPSec processing. If the packets are to be subjected to IPSec processing, the entry also contains a pointer to a Security Association (SA) specification which details the IPSec protocols, modes and algorithms to be applied to the packets matching this policy entry.

The first entry with selector(s) matching that/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. The entries in the SPD should therefore be ordered according to the desired preference of application.

The entries in the SPD determines 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 therefore plays a very important role in the control of the flow of all traffic through an IPSec system.

4.4.2 Security Association Database

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

- Sequence Number Counter: This is a 32-bit integer which 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 is used to ascertain whether an inbound AH or ESP packet is a replay.
- AH authentication algorithms and required keys.
- ESP Authentication algorithms and required keys.
- ESP encryption algorithms, keys, Initial Vector (IV) and IV mode.
- IPSec protocol mode: This field indicates which IPSec protocol mode (transport, tunnel or wild card) is applied to AH and ESP traffic.
- Path Maximum Transfer Unit (PMTU): Any observed PMTU and aging variables.
- 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 byte count which represents the number of bytes that IPSec protocols has been applied to. The parameter that expires first takes precedence.

Separate SAD are kept for inbound and outbound IPSec processing. For inbound or outbound traffic, the respective SAD is searched for selectors matching the SPI, the source or destination address and IPSec protocol, extracted from the packet header. If a matching entry is found, the parameter for this SA are compared with the appropriate fields in the AH or ESP headers. If the header fields corresponds 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. If the packet however is outbound, a new SA or SA bundle will be created and entered in the outbound SAD.

A SA lifetime has two kind of limit: a soft limit and a hard limit. When the soft limit is reached, the communicating peers must re-negotiate a new SA to replace the existing one. However, the existing SA is not deleted form 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.

Chapter 5

Authentication Header

The IP protocol is inherently insecure. The authentication mechanism that is used to provide integrity for the IP datagrams is rather primitive. The IP "header checksum" field was intended to guarantee the integrity of IP datagrams. The checksum is calculated over the IP header. It is computed by first setting the checksum field to 0, then calculates the 16-bit one's complement sum of the IP header and store the result in the checksum field. When the destination host receives the packet, it computes the 16-bit one's complement sum of the IP header. If none of the IP header fields have been modified, the destination host one's complement sum should all be ones since it involves the source one's complement sum. The datagram is therefore discarded if the receiver's one's complement sum is not all ones. This provides very little security however, since the IP header fields can easily be modified and the original checksum can be replaced with a new value that reflects the changes. The receiver will likely not detect the modification, provided that the checksum for the modified datagram was calculated correctly.

The Authentication Header (AH) protocol was designed to improve the security of IP datagrams. AH provides connectionless integrity and data origin authentication and an anti-replay protection service [KA98]. AH uses cryptographic Message Authentication Codes (MAC) (see Section: 3.3) to authenticate the IP datagram. MAC requires a secret key to generate the message digest of a bitstring. IPSec uses Internet Key Exchange (IKE)—will be discussed in Chapter 7—to negotiate, gener-

ate and put in place necessary cryptographic keys for AH or Encapsulated Security Payload (ESP). Since only the source and the destination nodes know the secret key, it should be computationally infeasible for an intruder to modify the datagram and recomputes an authentication data that will be validated by the hosts with the secret key.

In this chapter, we will discussed the format of AH, AH modes of operation and AH processing.

5.1 Authentication Header Format

AH consists of five fixed-length fields and a variable length Authentication Data field. Figure 5.1 shows the relative position of these fields in AH.

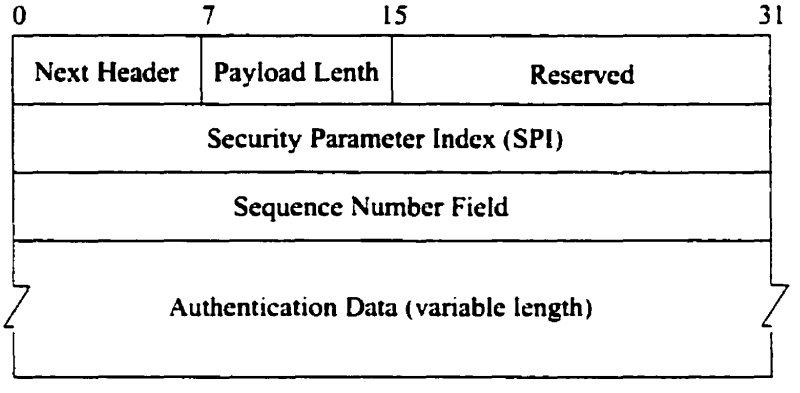

Figure 5.1: Authentication Header Format.

- Next Header: This is a 8-bit field that identifies the next payload after the AH. For example, if the ESP header follows the AH this field will contain the value 50; see [IANA00] for the set of assigned IP Protocol Numbers.
- Payload Length: This 8-bit field contains the length of AH in 32-bit words minus 2.
- Reserved: This 16-bit field is reserved for future use.

- Security Parameter Index (SPI): The SPI is a 32-bit integer that is used in combination with the source or destination address and the IPSec protocol (AH or ESP) to uniquely identify the SA (see Section 4.3) for the datagram. The range of numbers from 1 to 255 are reserved by the Internet Assigned Number Authority (IANA) for future use; and 0 is reserved for local and implementation specific use. Therefore, the current valid SPI values are between 256 and $2^{32} - 1$ inclusively.

- Sequence Number: This field contains a 32-bit unsigned integer which servers as a monotonically increasing counter. The sender's and receiver's sequence number counters are initialized to 0 when the SA is established. The sender consequently increases its sequence number by one for every packet it sends using a given SA. The sequence number is used to prevent intruders from capturing and re-sending previously transmitted datagrams.

- Authentication Data: This is a variable-length field that contains the Integrity Check Value (ICV) for the datagram. For IPv4 datagrams, this field must be an integral multiple of 32; whereas, for IPv6 datagrams, it must be a integral multiple of 64. Padding bits are added if necessary to acquire the desired length for this field.

5.2 Authentication Header Modes

The location of the AH header depends on the mode of operation of AH. There are two modes of operations: *transport mode* and *tunnel mode*. These modes of operation will be discussed in the following two subsections. AH can be applied in a nested fashion through the use of tunnel mode, applied in combination with ESP, or applied alone.

5.2.1 AH Transport Mode

In transport mode, AH is inserted after the IP header and before the transport layer protocol, or before any other IPSec protocol header. So, for IPv4, in transport mode,

AH is inserted after the variable length "options" field, see Figure 2.3 on page 13. Figure 5.2 shows the position of AH—in transport mode—relative to other header fields.

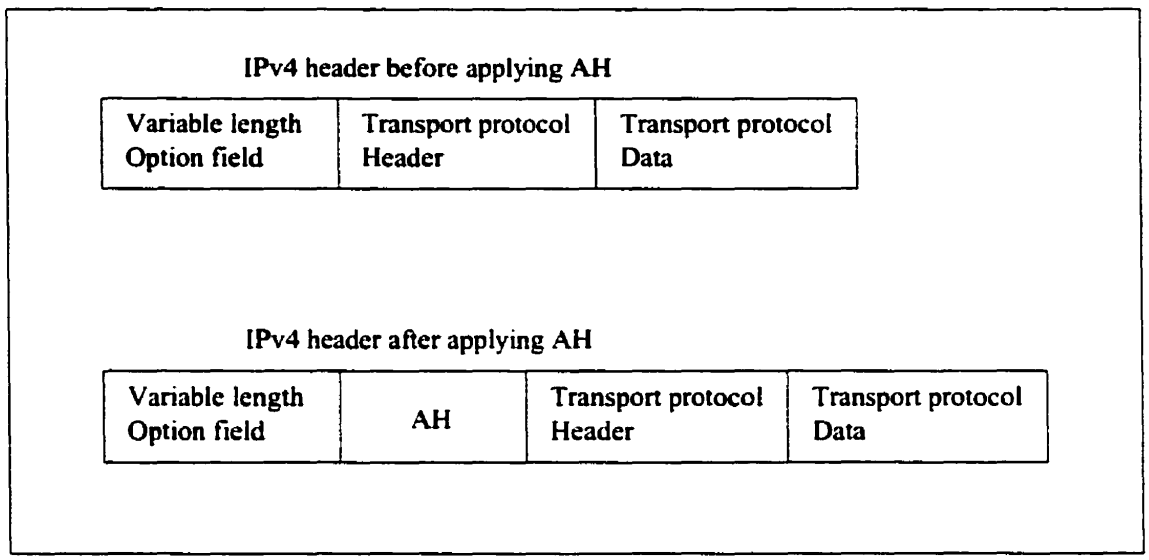

Figure 5.2: AH relative to other IPv4 header fields, in transport mode.

The option field was eliminated from IPv6. The options, in this version of IP, are treated as separate headers—called extension headers—that are inserted after the IP header. This feature speed up packet processing time, in that, except for the Hop-by-Hop extension header which contains routing information that routers need to examine, the extension headers are not examined or processed by any intermediate node on the path from the source to the destination; they are only processed by the destination host. In transport mode, for IPv6, the AH is inserted after the Hop-by-Hop, Routing and Fragmentation extension headers; Figure 5.3 illustrates this.

In AH transport mode, the Integrity Check Value (ICV) is calculated over the entire IP datagram; however, some of the IP header fields are mutable while the datagram is in transit from the source to the destination. In the calculation of ICV, a value of zero is used for the mutable fields. The integrity check is therefore not fully bullet proof since an intruder could modify the mutable fields and this would not be

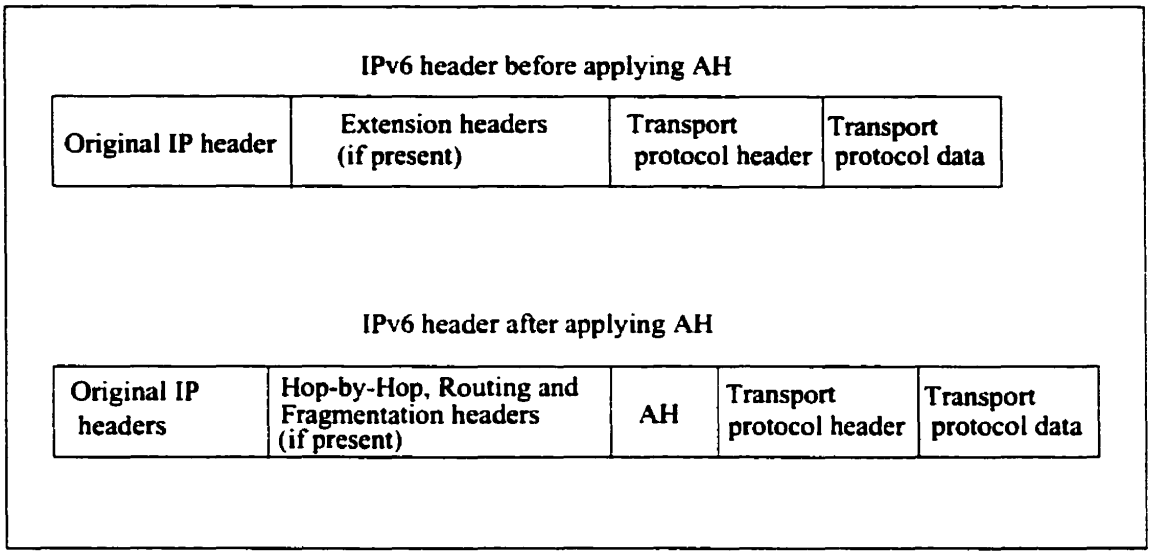

Figure 5.3: AH relative to other IPv6 extension headers, in transport mode.

detected during the authentication check at the receiver node.

5.2.2 AH Tunnel Mode

In tunnel mode, AH is inserted before the original IP header and a new IP header, is consequently inserted in front of the AH. This is illustrated diagrammatically for IPv4 in Figure 5.4 and for IPv6 in Figure 5.5 respectively.

The inner IP header carries the true source (the node that generated the packet) and the final destination address. Whereas, the outer IP header typically carries the sender's security gateway address as the source address, and the recipient's security gateway address as the destination address. Consequently, the source address in the inner and outer IP headers may be different. The same holds for the destination address. As is the case with AH transport mode, in AH tunnel mode, the ICV is computed over the entire IP datagram. However, unlike AH transport mode, all of the fields of the original IP header are immutable, since these fields are not modified by intermediate routers, because they are encapsulated by the outer IP header. The tunnel mode therefore offers added security, particular for hosts that are behind a

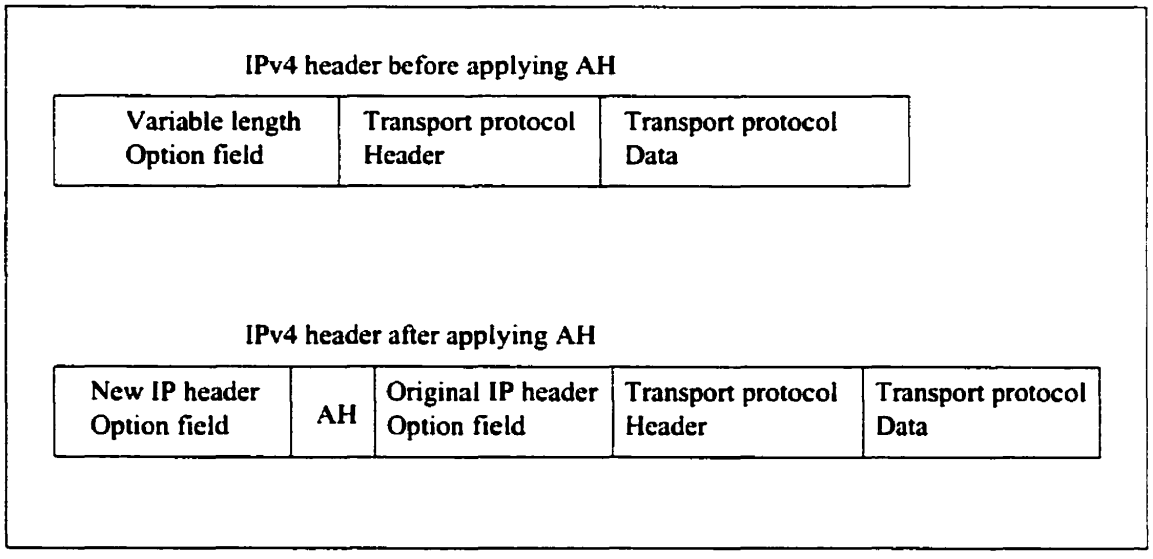

Figure 5.4: AH relative to other IPv4 header fields, in tunnel mode.

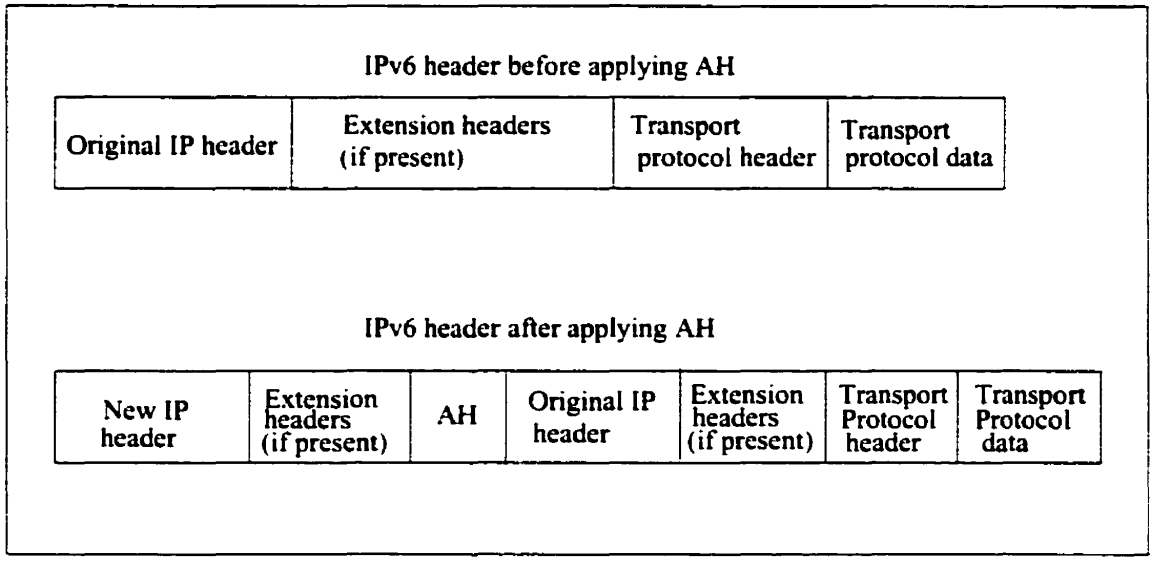

Figure 5.5: AH relative to other IPv6 extension headers, in tunnel mode.

security gateway or firewall.

5.3 AH Processing

5.3.1 Outbound Processing

When an IPSec implementation receives an outbound packet it uses the relevant selectors (destination IP address and port, transport protocol, etc) to search the Security Policy Database (SPD) to ascertain what policy is applicable to the traffic. If IPSec processing is required and a SA or SA bundle has already been established, the SPD entry that matches the selectors in the packet will point to the appropriate SA bundle in the SAD. If a SA has not been established, the IPSec implementation will create a SA and link it to the SPD entry. The SA is then used to process the packet as follows:

1. Generate or Increment Sequence Number: The sequence number is used to prevent replay of previously transmitted packets. When a new SA is established, the sender initializes its sequence number counter to zero. For each packet that the sender transmits, it increases the sequence number by one and insert the resulted value of the sequence number counter into the sequence number field of the AH header.

2. Calculate Integrity Check Value: The ICV is calculated using the values in the immutable fields of the IP header and the values of the mutable fields that are predictable upon arrival at the destination node. A value of zero is assigned to the unpredictable mutable fields for the purpose of the ICV computation. The SA for the datagram stipulates the cryptographic authentication algorithm that should be used to generate ICV. The available choices of authentication algorithms varies with different IPSec implementations. However, for interoperability, all implementations are required to support HMAC with MD5, and HMAC with SHA-1 [1]. The ICV generated is inserted into the variable length

[1] For detail on these cryptographic algorithms, see Chapter 3

Authentication Data field of the AH, then if required, the datagram is fragmented into the appropriate fragment size and sent to the destination node.

5.3.2 Inbound Processing

When a datagram arrives at an IPSec host or security gateway, if the more fragment bit is set, this is an indication that there are other fragments that are yet to arrive. The IPSec application therefore waits until a fragment arrives with a sequence number that is similar to the previous ones, and has the more fragment bit not set. It then reassembles the IP fragments, then performs the following steps:

1. Use the SPI, destination IP address and IPSec protocol in the IP header (outer IP header if tunnel mode) to lookup the SA for this datagram in the inbound SAD. If the lookup fails, it drops the packet and logs the error.

2. Use the SA found in (1) to do the IPSec processing. This involves first checking to determine whether the selectors in the IP headers (inner header if tunnel mode) match those in SA. If the selectors do not match, the application drops the packet and audit the error. If the selectors match, the IPSec application keeps track of the SA and the order in which it is applied relative to the others, and continues to do (1) and (2) until it encounters a non-IPSec extension header or a transport layer protocol.

3. Use the selectors in the packet to find a policy in the inbound SPD whose selectors match those of the packet.

4. Check whether the SA's found in (1) and (2) match the policy specified in (3). If the check fails, repeat steps (4) and (5) until all policy entries have been check or until the check succeeds.

5. If anti-replay is enabled, use the anti-replay window of the SA to determine if the packet is a replay. If the packet is a replay, drop it and log the error.

6. Use the authentication algorithm specified by the SA bundle to calculate the ICV for the packet and compares it with the value stored in the Authentication Data field. If the two values differs, then discard the packet and audit the error.

At the end of these steps, if the packet have not been discarded, it is then passed to the transport layer protocol, or is forwarded.

Chapter 6

Encapsulating Security Payload

As is the case with the Authentication Header (AH), the Encapsulating Security Payload (ESP) is designed to improve the security of the Internet Protocol (IP). ESP provides data confidentiality, data origin authentication, connectionless integrity, an anti-replay service and limited traffic flow confidentiality [SA98]. ESP in essence, provides the services that AH offers plus two additional services: data confidentiality and a limited traffic flow confidentiality. ESP can be applied alone, in a nested fashion or in combination with AH. In this chapter, we will discuss the format of ESP packets, the modes of operation of ESP and the processing of ESP packets.

6.1 ESP Packet Format

ESP packets consist of four fixed-length fields and 3 variable-length fields. Figure 6.1 shows the packet format of this protocol.

- Security Parameter Index (SPI): The SPI is a 32-bit integer that is used in combination with the source or destination address and the IPSec protocol (ESP or AH) to uniquely identify the SA (see Section 4.3) for the datagram. The range of numbers from 1 to 255 are reserved by the Internet Assigned Number Authority (IANA) for future use; and 0 is reserved for local and implementation specific use. Therefore, the current valid SPI values are between 256 and $2^{32} - 1$ inclusively. This field is similar to the AH SPI field.

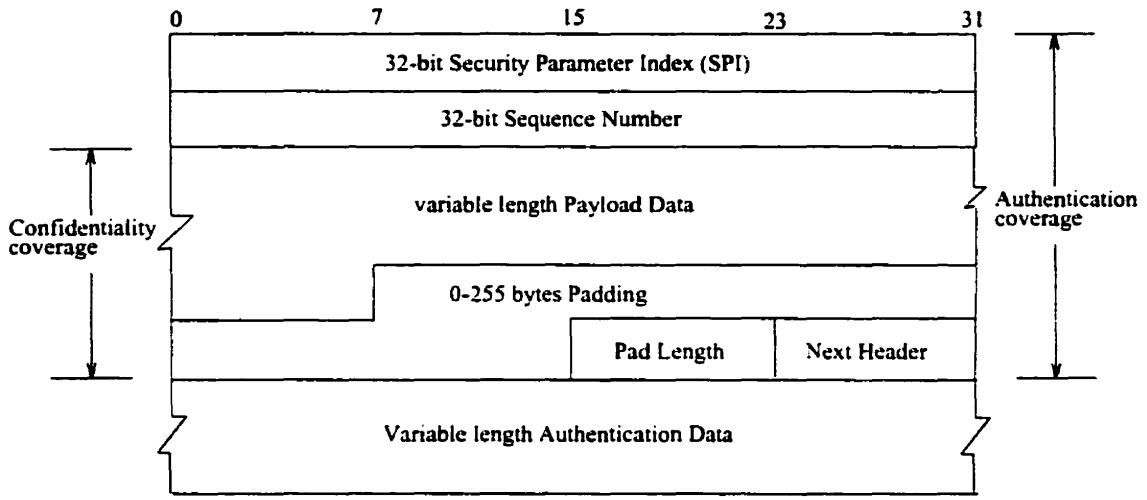

Figure 6.1: ESP Packet Format.

- Sequence Number: As is the case for the AH, this field contains a 32-bit unsigned integer which servers as a monotonically increasing counter. The sender's and receiver's sequence number counters are initialized to 0 when the SA is established. The sender consequently increases it's sequence number by one for every packet it sends using a given SA. The sequence number is used to prevent intruders from capturing and re-sending previously transmitted datagrams.

- Payload Data: This is a variable length field that contains the actual payload data. The length of this field in bits must be an integer multiple of eight.

- Padding: This field contains the padding bits—if any—that are utilized by the encryption algorithm, or that are used to align the Pad Length (see Figure 6.1) field so that it begins at the third byte within the 4-byte word. The length of this field can be between 0 and 255 bytes inclusively.

- Pad Length: The Pad Length field is a 8-bit field which indicates the number of padding bytes in the Padding field. The valid values for this field are integers between 0 and 255 inclusively.

- Next Header: This a 8-bit field that identifies the type of data encapsulated

in the payload. It may indicate an IPv6 extension header or a transport layer protocol; refer to [IANA00] for the set of assigned IP Protocol Numbers.

- Authentication Data: This is a variable-length field that contains the Integrity Check Value (ICV), which as indicated by Figure 6.1, is calculated over the length of the ESP packet minus the Authentication data field. The length of this field is specified by the authentication algorithm employed for the authentication.

6.2 ESP Modes

As is the case for AH, the location of the ESP in the packet depends on the mode of operation of ESP. There are two modes of operations: *transport mode* and *tunnel mode*. These modes of operation will be discussed in the following two subsections.

6.2.1 ESP Transport Mode

In transport mode, ESP is inserted after the IP header and any options it contains but before any transport layer protocol; or before any IPSec protocol that has already been applied. So, for IPv4, in transport mode, ESP is inserted after the variable-length "options" field, see Figure 2.3 on page 13. Figure 5.2 shows the position of ESP—in transport mode—relative to other header fields. In this diagram, the ESP Header field consists of the SPI and Sequence number fields; whereas, the ESP Trailer field consists of the Padding, Pad Length and Next Header fields. Encryption and authentication services are applied as shown in Figure 6.2.

In transport mode, for IPv6, the ESP is inserted after the Hop-by-Hop, Routing and Fragmentation extension headers; Figure 6.3 illustrates this.

6.2.2 ESP Tunnel Mode

In tunnel mode, ESP is inserted before the original IP header and a new IP header, is consequently inserted in front of the ESP header. This is illustrated diagrammatically

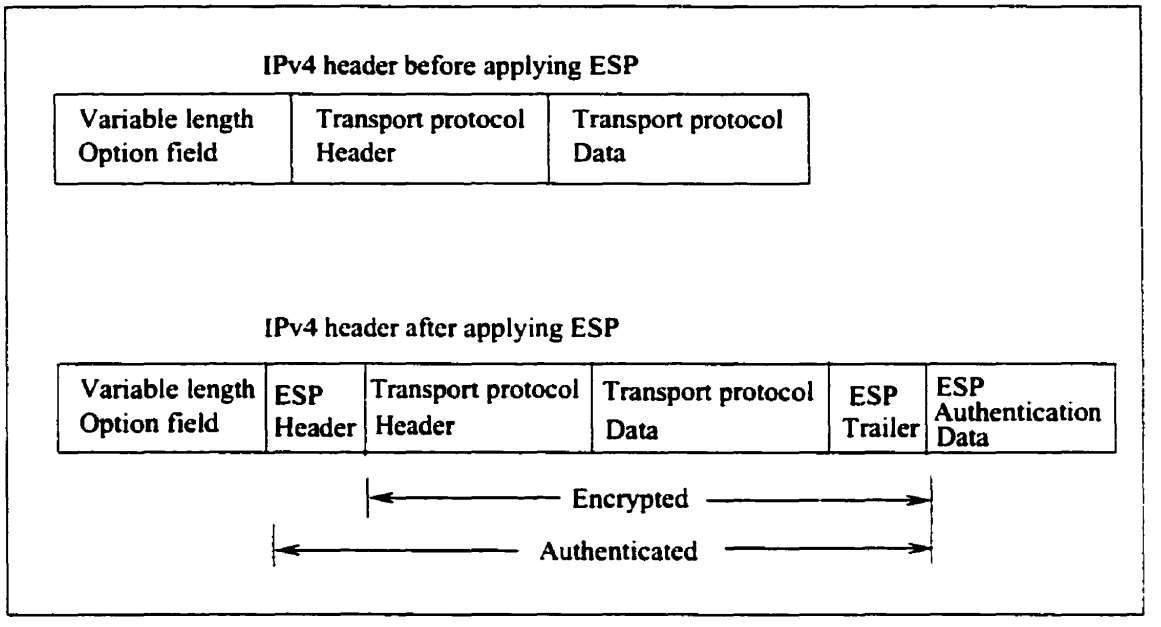

Figure 6.2: ESP relative to other IPv4 header fields, in transport mode.

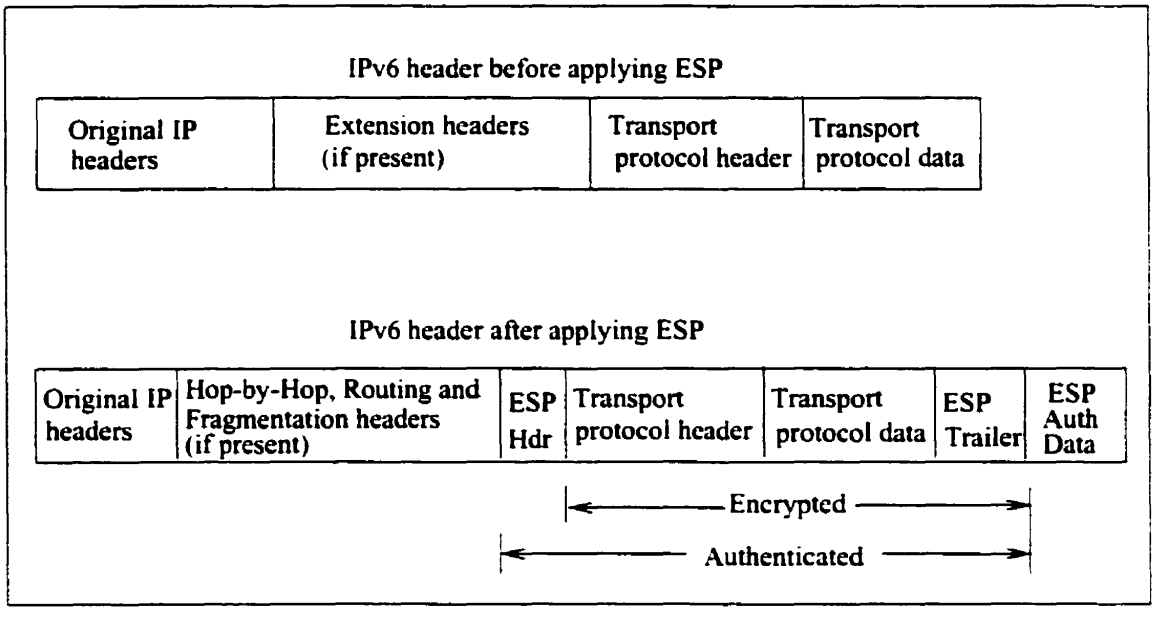

Figure 6.3: ESP relative to other IPv6 extension headers, in transport mode.

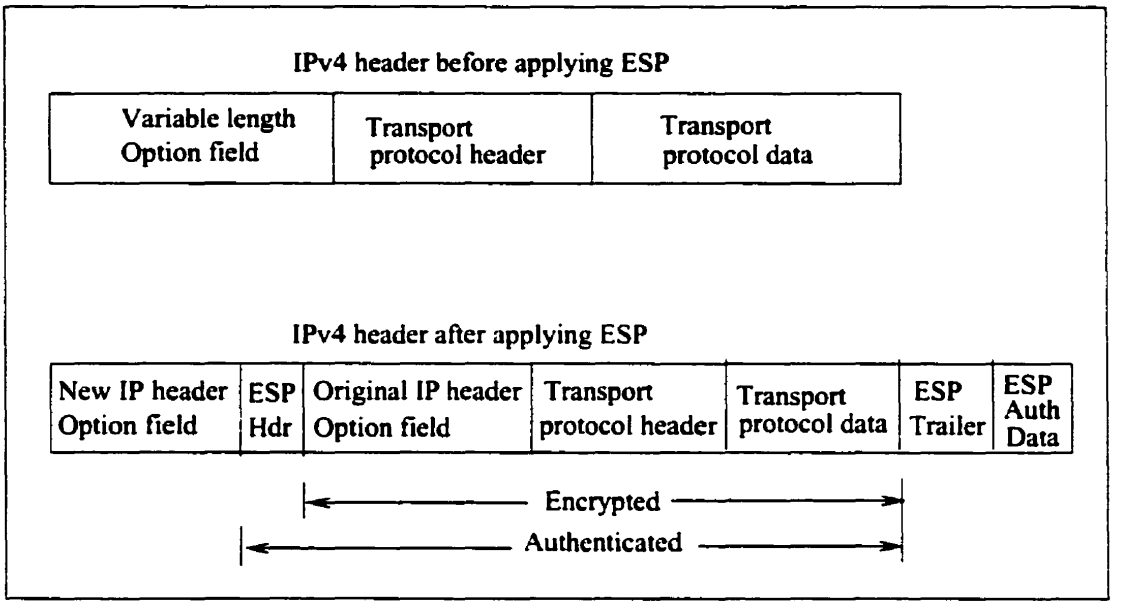

Figure 6.4: ESP relative to other IPv4 header fields, in tunnel mode.

for IPv4 in Figure 6.4 and for IPv6 in Figure 6.5, respectively.

The inner IP header carries the true source (the node that generated the packet) and the final destination address. Whereas, the outer IP header typically carries the sender's security gateway address as the source address, and the recipient's security gateway address as the destination address. Consequently, the source address in the inner and outer IP headers may be different. The same holds for the destination address.

6.3 ESP Processing

6.3.1 Outbound Processing

When an IPSec implementation receives an outbound packet it uses the relevant selectors (destination IP address and port, transport protocol, etc) to search the Security Policy Database (SPD) to ascertain what policy is applicable to the traffic. If IPSec processing is required and a SA or SA bundle has already been established.

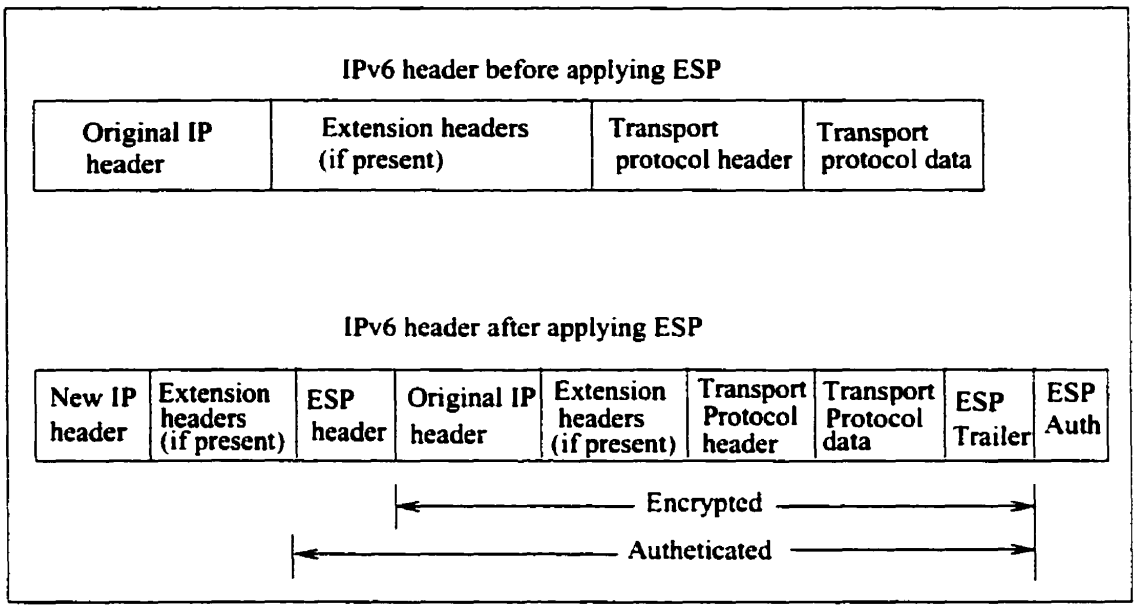

Figure 6.5: ESP relative to other IPv6 extension headers, in tunnel mode.

the SPD entry that matches the selectors in the packet will point to the appropriate SA bundle in the SAD. If a SA has not been established, the IPSec implementation will create a SA and link it to the SPD entry. The SA is then used to process the packet as follows:

1. Generate or Increment Sequence Number: The sequence number is used to prevent replay of previously transmitted packets. When a new SA is established. the sender initializes its sequence number counter to zero. For each packet that the sender transmits, it increases the sequence number by one and insert the resulted value of the sequence number counter into the sequence number field of the ESP packet.

2. Encryption of Packet: If the traffic requires confidentiality services. the SA will specify the encryption algorithm to be used. The available choices of encryption algorithms depends on the IPSec implementation; However. only private-key cryptosystems are currently use because of the slow execution speed of public-key cryptosystems compared to private-key enciphering systems, as discussed in

Chapter 3. For interoperability, all IPSec implementation must provide DES in CBC mode [1] as an encryption algorithm option. When the encryption algorithm requires an Initial Vector (IV) [2]—as it is the case with DES in CBC mode, the IV is carried in the first few bytes of Payload Data field. As illustrated in Figures 3.2 - 3.5, the ESP header: the SPI and the Sequence number, are not encrypted. This is so, because the destination node needs the SPI to look up the SA of the packet, and it requires the Sequence Number to ascertain whether or not the packet is a re-play of a previously transmitted packet. The authentication data is similarly not encrypted because the receiving node must authenticate the packet before it decrypts it. This will be discussed further in the discussion of ESP Inbound Processing. When encryption service is required the packet must be encrypted before the calculation of the ICV.

3. Calculate the Integrity Check Value: If the SA for the packet stipulates that ESP authentication service should be applied, the ICV is calculated using the values in all the fields of the ESP packet except the Authentication Data field, then the ICV is inserted in the Authentication Data field. The SA for the packet specifies the algorithm that should be used to generate ICV. The available choices of authentication algorithms varies with different IPSec implementation. However, for interoperability, all implementations are required to support HMAC with MD5, and HMAC with SHA-1 [3]. If fragmentation is required, the packet is broken into the appropriate sizes then sent on route to the destination node.

6.3.2 Inbound Processing

When a packet arrives at an IPSec host or security gateway, if the more fragment bit is set, this is an indication that there are other fragments that are yet to arrive. The IPSec application therefore waits until a fragment arrives with a sequence number that is similar to the previous ones, and has the more fragment bit not set. It then

[1] See Chapter 3 for a description of DES algorithm.

[2] See Chapter 3 for an explanation of the use of Initial Vectors.

[3] For detail on these cryptographic algorithms, see Chapter 3

reassembles the IP fragments, then performs the following steps:

1. Use the destination IP address and IPSec protocol in the IP header (outer IP header if tunnel mode), and the SPI in the ESP header to lookup the SA for the packet in the inbound SAD. If the lookup fails, it drops the packet and logs the error.

2. Use the SA found in (1) to process the ESP packet. This involves first checking to determine whether the selectors in the IP headers (inner header if tunnel mode) match those in SA. If the selectors do not match, the application drops the packet and audit the error. If the selectors match, the IPSec application keeps track of the SA and the order in which it is applied relative to the others, and continues to do (1) and (2) until it encounters a non-IPSec extension header or a transport layer protocol.

3. Use the selectors in the packet to find a policy in the inbound SPD whose selectors match those of the packet.

4. Check whether the SAs found in (1) and (2) match the policy specified in (3). If the check fails, repeat steps (4) and (5) until all policy entries have been check or until the check succeeds.

5. If anti-replay is enabled, use the anti-replay window of the SA or SA bundle to determine if the packet is a replay. If the packet is a replay, drop it and log the error.

6. If the SA stipulates that authentication service is required, then use the authentication algorithm and the private-key specified by the SA bundle to calculate the ICV for the packet and compares it with the value stored in the ESP Authentication Data field. If the two values differs, then discard the packet and audit the error.

7. If the SA indicates the application of confidentiality service, use the cryptographic algorithm and the private-key that the SA specifies to decrypt the

packet. Decryption processes are in general quite CPU and memory intensive. If the IPSec system is allowed to perform unnecessary decryption or encryption of packets, then the system will be vulnerable to denial of service attack. Consequently, when decryption or encryption is required, this service is applied after the packet has been successfully authenticated.

At the end of these steps, if the packet have not been discarded, it is then passed to the transport layer protocol, or is forwarded.

Chapter 7

Internet Key Exchange

Internet Key exchange (IKE) is a protocol that is use to negotiate and provide authenticated keying materials in a protected manner for Security Associations (SA). IKE is a hybrid protocol that utilizes relevant parts of three different protocols: Internet Security Association and Key Management Protocol (ISAKMP) [MSST98], Oakley Key Determination Protocol [Orm98] and SKEME [Kra96].

7.1 ISAKMP

ISAKMP defines procedures and packet formats to negotiate, establish, modify and delete SAs. ISAKMP provides a common framework for the format of SA attributes, and the methodologies for negotiating, modifying and deleting SAs. that different key exchange protocols can use. ISAKMP defines several payloads. These payloads provides modular building block for constructing ISAKMP messages. We will examine the ISAKMP payloads that IKE utilizes, but before we do, we will discuss the ISAKMP header format and the Generic Payload header.

7.1.1 ISAKMP Header Format

ISAKMP messages consist of a fixed-length header followed by a variable number of payloads. The fixed-length header contains the necessary information for the protocol

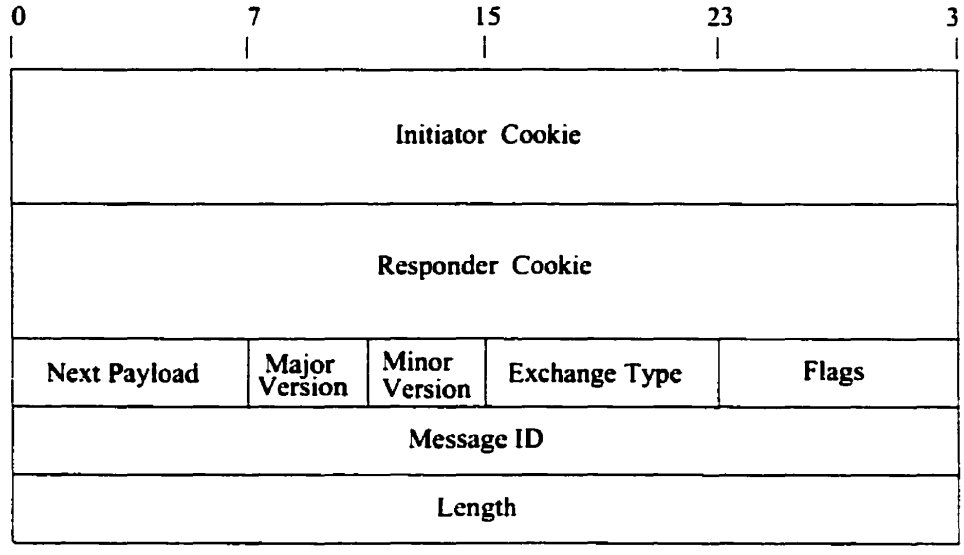

Figure 7.1: ISAKMP Header Format.

to maintain state and process the payloads. The ISAKMP header format is illustrated in Figure 7.1.

A description of the ISAKMP header fields follows.

- Initiator Cookie: This field contains a unique 8-bit integer that the initiator of the ISAKMP exchanges generates. It is used as an anti-clogging protection agent: It helps the communicating peers to ascertain whether or not a particular message indeed originated from the other peer. If the initiator determines that the cookie in the Responder Cookie field does not match the cookie it previously received from the responder, it will discard the message; similarly, if the responder ascertains that the Initiator Cookie field does not match the cookie it previously received from the initiator, it will drop the message without any further processing. The method of generating the cookies varies with different implementations of ISAKMP; however, the protocol specifies that the cookies, whether that of the initiator or responder, should be generated using secret information that are unique to the respective ISAKMP communicating hosts, and it should not be possible to determine the secret information from the cookie. In addition, the cookie for each SA should be unique. A possible method of

Table 7.1: Assigned Values for ISAKMP Payloads.

Next Payload Type	Assigned Value
NONE	0
Security Association	1
Proposal	2
Transform	3
Key Exchange	4
Identification	5
Certificate	6
Certificate Request	7
Hash	8
Signature	9
Nonce	10
Notification	11
Delete	12
Vendor	13
RESERVED	14 - 127
Private USE	128 - 255

generating cookies is to perform a hash (using MD5 or SHA-1 hash function) of the concatenation of the source and destination address, UDP source and destination ports, a locally generated secret random number and the current date and time.

- Responder Cookie: This field contains the responder's 8-bit cookie. The attributes of this cookie are similar to that of the initiator's.
- Next Payload: This is a 8-bit field that indicates the first payload in the message. Table 7.1 shows the payload that ISAKMP currently defines.
- Major Version: This 4-bit field indicates the major version of the ISAKMP

Table 7.2: ISAKMP Exchange Type and Assigned Values.

Exchange Type	Assigned Value
NONE	0
Base	1
Identity Protection	2
Authentication Only	3
Aggressive	4
Informational	5
ISAKMP Future Use	6 - 31
DOI Specific Use	32 - 239
Private Use	240 - 255

protocol in used. The specification for ISAKMP specifies that an ISAKMP implementation should not accept packets with a Major Version or Minor Version that is larger than its own.

- Minor Version: This 4-bit field contains the protocol Minor Version number.
- Exchange Type: This 8-bit field indicates the type of exchange that the message comprises of. Table 7.2 shows the exchange types that ISAKMP currently defines.
- Flags: This 8-bit field indicates specific options that are set for ISAKMP exchanges. The first 3 bits of this field are currently used, the others are set to zero before transmission.

    1. E(ncryption Bit): This is the least significant of the flag bits. When it is set to 1, all payloads following the header are encrypted using the encryption algorithm specified in the ISAKMP SA. However, when this bit is set to 0, the payload is not encrypted.
    2. C(commit bit): This is the second bit of the Flag field. It is used to ensure

that encrypted data are not received before the establishment of the SA is completed. When this bit is set to 1, the communicating peer which did not set the bit must wait until it receives an Information Exchange containing a Notify Payload with the "CONNECTED Notify Message" from the peer which set the commit bit.

3. A(uthentication only Bit): This is the third bit of the Flag bits. It allows the transmission of information with integrity check using the authentication algorithm specified in the SA, but with no encryption.

- Message ID: This is a 4-byte field which contains a random value generated by the initiator of the phase 2 (will be discussed later in this chapter) negotiation. It serves as a unique Message Identifier that is used to identify the protocol state during phase 2 negotiations.

- Length: This is a 4-bytes field which indicates the length of the total message (header + payloads) in bytes.

7.1.2 ISAKMP Payloads Formats

Generic Payload Header

Each ISAKMP payload begins with a generic header. The Generic Payload Header clearly defines the boundaries of the payload and consequently allows the chaining of different payloads. Figure 7.2 illustrates the Generic Payload Header.

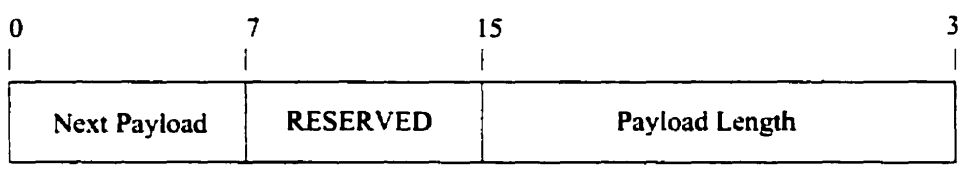

Figure 7.2: Generic Payload Header Format.

A description of the Generic Payload Header fields follows.

- Next Payload: This 8-bit field identifies the payload type that follows this payload. If the current payload is the last payload of the message, the Next Payload field will be set to 0.
- RESERVED This field is unused and it is set to 0
- Payload Length: Contains the length in bytes of the current payload including the Generic Payload Header. This is a 2-byte field.

We will now discuss the ISAKMP payloads that IKE uses.

Security Association Payload

The Security Association Payload is used to negotiate SA and to indicate the Domain of Interpretation (DOI) under which the negotiation takes place. Figure 7.3 illustrates the format of this payload.

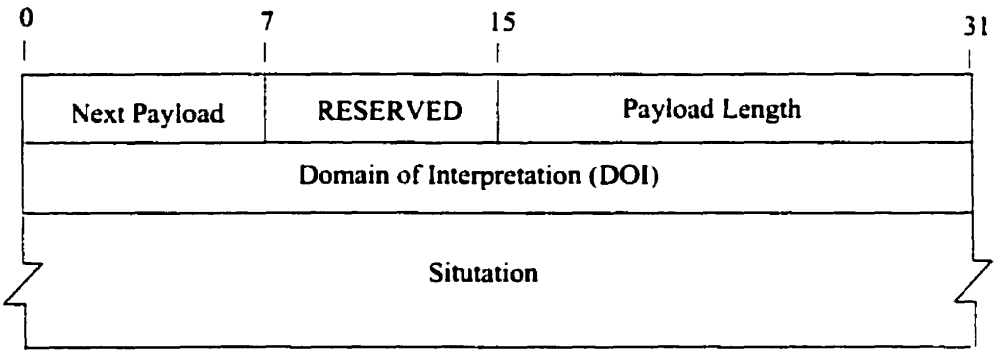

Figure 7.3: Security Association Payload Format.

The "Next Payload", "RESERVED" and "Payload Length" fields are similar to those of Generic Payload Header fields.

- Domain of Interpretation (DOI): This 4-byte field contains a 4-byte unsigned integer which identifies the DOI under which the negotiation is taking place. A DOI defines payload formats, exchange types and convention for naming relevant information such as cryptographic algorithms, modes, etc.

- Situation: This variable length field identifies the Situation under which the negotiation is taking place. A Situation is the set of information that will be used to determine the required security service.

Proposal Payload

The Proposal Payload contains information used during SA negotiation. This payload provides the framework for the initiator of the ISAKMP exchange to present to the recipient the preferred security protocol(s) and associated security mechanisms desired for the SA being negotiated. The Proposal Payload is illustrated in Figure 7.4.

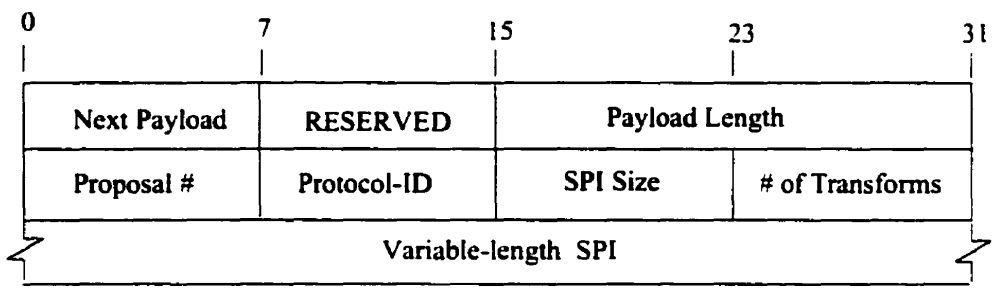

Figure 7.4: Proposal Payload Format.

A description of the Proposal Payload fields follows. The "Next Payload", "RESERVED" and "Payload Length" fields are similar to those described in the payloads that we discussed previously.

- Proposal Number: This 8-bit field contains the identification number of the proposal for the current payload. The Proposal Number allows the peers that are involved in the SA establishment negotiation, to present preferences to its communicating peers in the form of logical AND or OR operations. For example, if the initiator of the exchange wishes to inform the other peer that it desires a combine protection suit consisting of ESP encryption with Triple-DES and authentication with HMAC-SHA-1 and AH authentication with HMAC-MD5, it would send a message that contains three Proposal Payloads, each with the

same Proposal Number. If however, the peer needs to transmit the information that it requires either ESP authentication with HMAC-MD5 or AH authentication with HMAC-SHA-1, it would send a message consisting of two Proposal Payloads, each with monotonically increasing Proposal Number. The Proposal Number here indicates the order of preference of the proposals: The greater the preference, the smaller the proposal number.

- Protocol-ID: This 8-bit field contains the protocol identifier (example 50 for ESP and 51 for AH).

- SPI Size: Contains the length in bytes of the Security Parameter Index (SPI) of the protocol specified by the Protocol-ID field. If the protocol is ISAKMP, the SPI is the initiator-responder cookie pair. This field is a 8-bit field.

- Number of Transform: This 8-bit field gives the number of Transforms for the proposal. Each of the transforms specified is embodied in a Transform Payload.

- SPI: This variable-length field contains the source node SPI.

Transform Payload

The Transform Payload provides the framework for the initiating peer to present different security mechanisms for a given protocol during the negotiation to establish a SA. The Proposal Payload identifies a protocol for which services are being negotiated. The Transform Payload allows the initiating peer to presents the supported transform or mode of operation of the proposed protocol. Figure 7.5 illustrates the format of the Transform Payload.

A description of the Transform Payload fields follows.

- Next Payload: The 8-bit Next Payload field identifies the payload type that follows the current payload. This field will either contains the value 3 or 0. If another Transform Payload follows the current Transform Payload, the value in this field will be 3; if however the current Transform Payload is the last one for the proposed protocol, the value in this field will be 0.

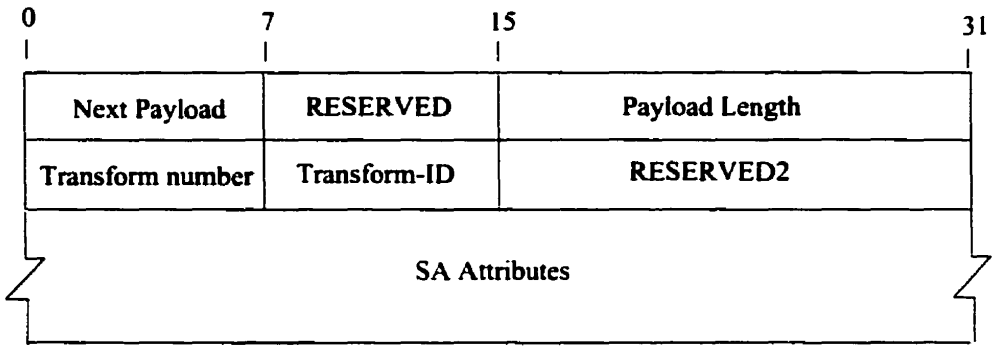

Figure 7.5: Transform Payload Format.

- RESERVED: Similar to the "RESERVED" field in the payloads previously discussed.
- Payload Length: Similar to the "Payload Length" field in the payloads previously discussed.
- Transform number: This 8-bit field identifies the Transform Number for the current payload. If there are more than one transforms for the proposed protocol, each transform will be embodied in a Transform Payload and each transform is identified by a monotonically increasing number. The transforms are ordered according to the initiating peer's preference: the most desired transform has the lowest Transform Number.
- Transform-ID: This 8-bit field specifies the transform identifier for the proposed protocol.
- RESERVED2: This 16-bit field is unused and is set to 0.
- SA Attributes: This variable-length field contains the Security Association attributes for the Transform specified by the Transform-ID field.

Key Exchange Payload

The Key Exchange Payload supports various key exchange protocols. Figure 7.6 illustrates the Key Payload format.

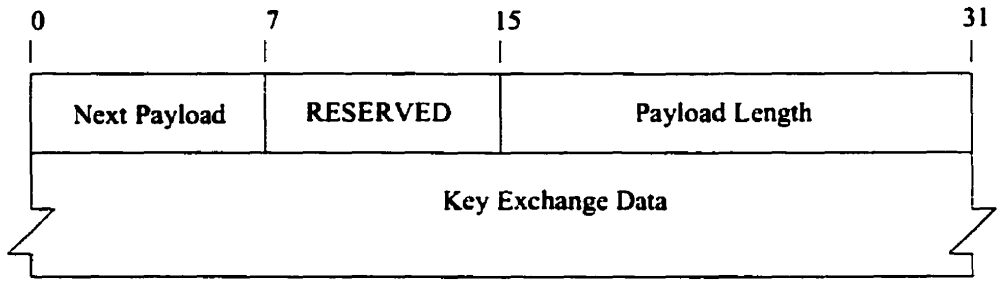

Figure 7.6: Key Exchange Payload Format.

The "Next Payload", "RESERVED" and "Payload Length" fields are similar to those of the payload previously discussed. The Key Exchange Data field is of variable length and it contains the data required to generate a session key. The DOI for the respective key exchange, specifies the format and the interpretation of the data in this field.

Identification Payload

The Identification Payload allows communicating peers to exchange identity information. Figure 7.7 illustrates the format of this payload.

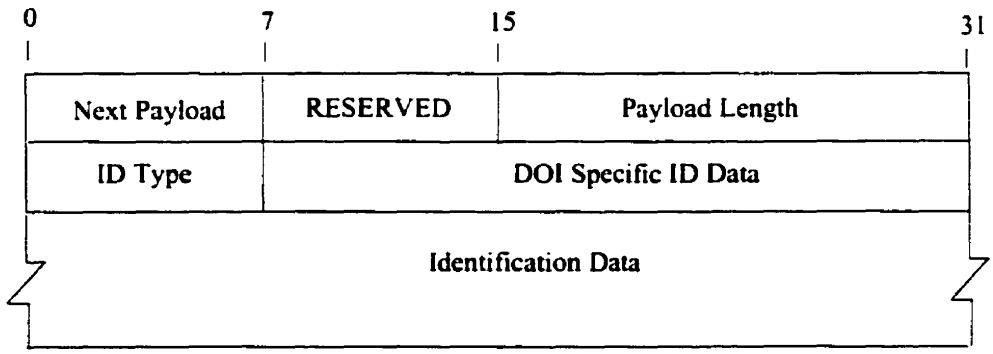

Figure 7.7: Identification Payload Format.

The "Next Payload", "RESERVED" and "Payload Length" fields are similar to

those of the payload previously discussed.

- ID Type: This 8-bit field contains the type of identification being used, and it depends on the DOI of the respective key exchange.
- DOI Specific ID Data: This 24-bit field contains DOI specific identification data.
- Identification Data: This variable-length field contains identification information specified by the DOI of the key exchange. The ID Type field specifies the format of the information in this field.

Certificate Payload

The Certificate Payload allows communicating peers to exchange certificate or certificate-related material. Figure 7.8 illustrates the format of this payload.

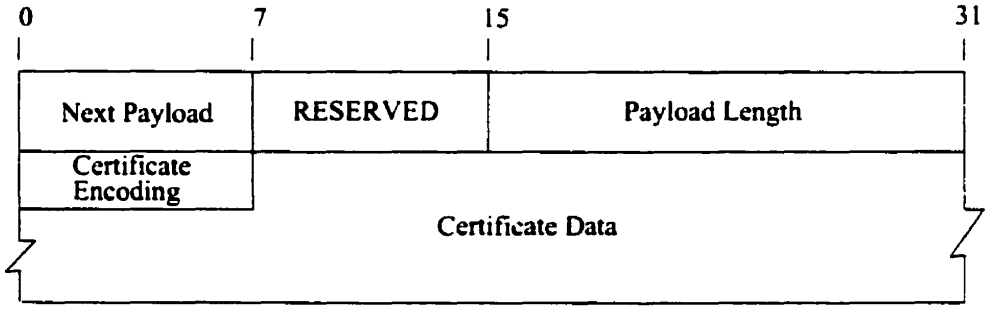

Figure 7.8: Certificate Payload Format.

The "Next Payload", "RESERVED" and "Payload Length" fields are again, similar to those of the payload previously discussed.

- Certificate Encoding: This 8-bit field identifies the kind of certificate or certificate-related information the "Certificate Data" field contains.
- Certificate Data: This variable-length field contains the actual certificate data for the certificate type specified in the "Certificate Encoding" field.

Hash Payload

The Hash Payload contains the data generated by the hash function selected during the SA negotiation. Figure 7.9 illustrates the format of this payload.

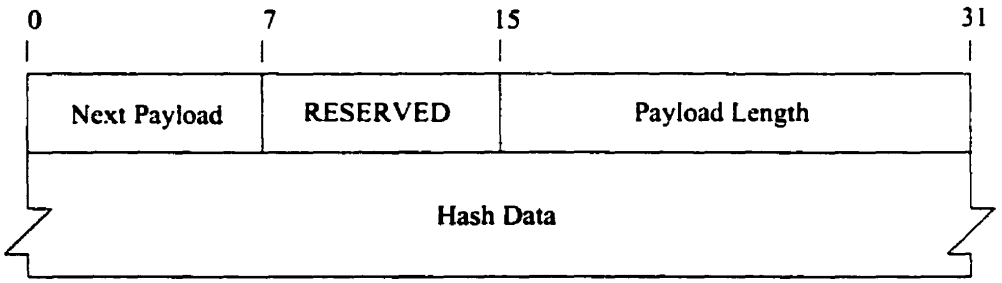

Figure 7.9: Hash Payload Format.

The "Hash Data" field contains the message digest that resulted from the application of the hash function to the input data.

Signature Payload

The Signature Payload contains the data generated by the signature function negotiated for the SA; and it is used to verify the integrity of the data in an ISAKMP message. The format of this payload is illustrated in figure 7.10.

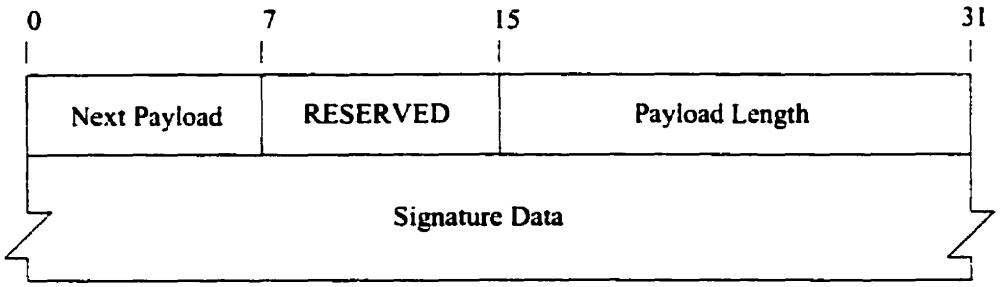

Figure 7.10: Signature Payload Format.

The "Signature Data" field is of variable length and it contains the signature data

that resulted from the signing of the ISAKMP message using the digital signature algorithm negotiated for the SA.

Nonce Payload

The Nonce payload contains random data that is used to protect the exchange data against replay. Figure 7.11 illustrates this payload.

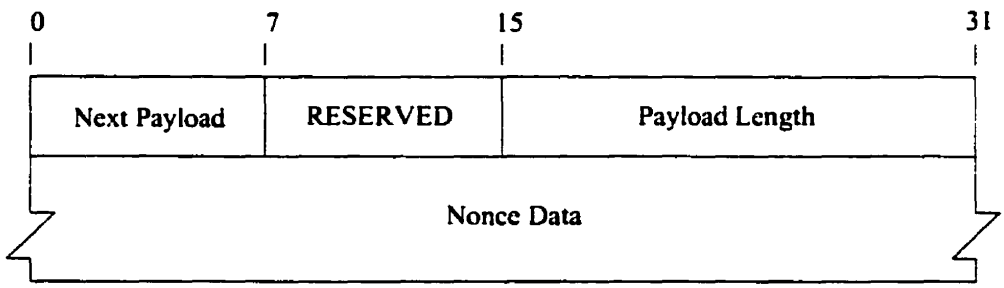

Figure 7.11: Nonce Payload Format.

The variable-length "Nonce Data" field contains the random data generated by the communicating peer that is sending the message.

ISAKMP defines a number of other payloads. For information about these additional payloads or for further detail about the ISAKMP protocol, please refer to [MSST98] which contains the original specification of ISAKMP.

7.2 Exchanges

7.2.1 Exchange Phases

IKE presents different exchanges in modes which operate in one of two ISAKMP negotiation phases.

Phase 1 Negotiation

In this phase, two ISAKMP communicating peers establish an ISAKMP SA which essentially is an agreement on how further negotiation traffic between the peers will be protected. This ISAKMP SA is then used to protect the negotiation of SAs for other protocols, such as ESP or AH.

Phase 2 Negotiation

This phase is used to establish SAs for other security services, such as AH or ESP. A single Phase 1 SA can be used to establish many Phase 2 SAs.

7.2.2 Exchange Modes

IKE currently defines four possible modes of exchange: Main Mode, Aggressive Mode, Quick Mode and New Group Mode. The first three negotiate SAs; whereas, the latter negotiates groups for Diffie-Hellman exchange. SA or group offers take the form of Transform Payload(s) encapsulated in Proposal Payload(s) which is/are encapsulated in Security Association Payload(s). These exchange modes will be discussed further in subsequent subsections; however, before we do, it is necessary to explain some notations that we will employ throughout this chapter.

Notations

HDR: indicates an ISAKMP header whose exchange type is the mode. When it is written as *HDR**, it indicates that the payloads following the ISAKMP headers are encrypted.

SA: is a Security Association Payload with one or more Proposal Payload.

KE: is a Key Exchange Payload.

IDx: denotes the Identification Payload where "x" is either "ii" which stands for ISAKMP initiator, or "ir" signifying ISAKMP responder.

HASH: is the Hash Payload.

SIG: signifies the Signature Payload. The data to be signed is exchange-specific.

AUTH: is a generic authentication mechanism such a SIG or HASH.

CERT: denotes the Certificate Payload.

Nx: represents the Nonce Payload for "x" where "x" is either "i" or "r" for ISAKMP initiator and responder respectively.

<P>_b: denotes the body of the payload <P>, that is, the payload without the ISAKMP Generic Header.

Ck-I: is the Initiator cookie from the ISAKMP header.

Ck-R: is the Responder cookie from the ISAKMP header.

g_i^x: represents the initiator's public Diffie-Hellman value.

g_r^x: represents the responder's public Diffie-Hellman value.

prf(key,msg): indicates a pseudo-random function—such as HMAC— with key "key" and input message "msg".

KEYSTR: represents a key string derived from secret keying material known only to the communicating peers.

KEYSTR_e: denotes the keying material used by ISAKMP to protect the confidentiality of its messages.

KEYSTR_a: denotes the keying material used by ISAKMP to authenticate its messages.

KEYSTR_2: is the keying material used to generate keys for non- ISAKMP SA during Phase 2 negotiation.

<x>y: indicates that x is encrypted using key y.

—>: denotes initiator to responder communication.

<—: denotes responder to initiator communication.

$x \mid y$: denotes that x is concatenated with y.

[x]: indicates that x is optional.

Main Mode

Main Mode is an adaptation of the ISAKMP "Identity Protection Exchange". It involves an authenticated Diffie-Hellman key exchange. It is used to negotiate Phase

1 ISAKMP SA. This exchange mode was designed to separate key exchange information from the identity and authentication information. The separation of these information provides protection for the identity of the communicating peers—since the identity information can be exchanged under the protection of the previously generated Diffie-Hellman shared secret—at the expense of two or three additional messages. Figure 7.12 shows an example of a Main Mode exchange.

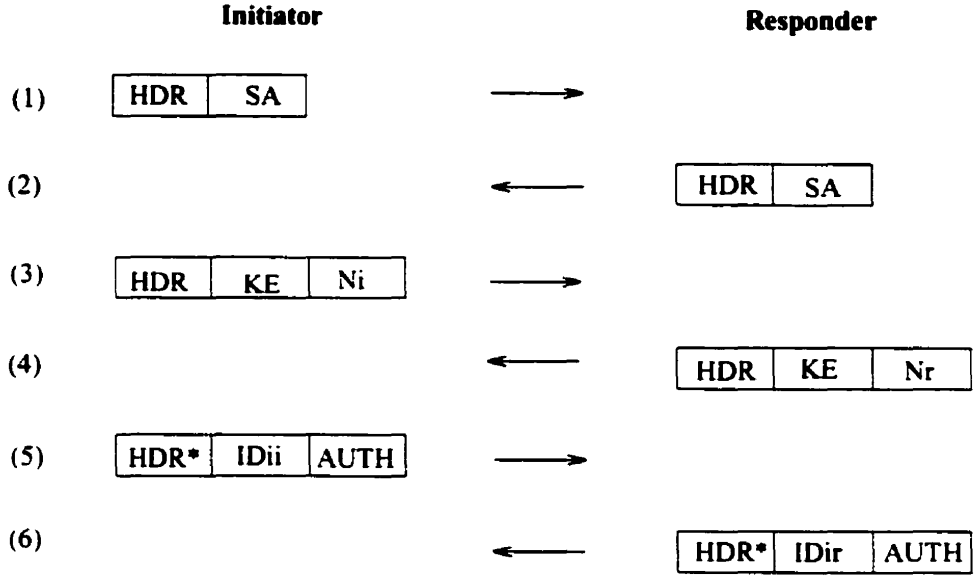

Figure 7.12: Example of a Main Mode Exchange.

- In message (1), the initiator sends the responder a Security Association (SA) Payload which encapsulates Proposal Payload(s) which in turn encapsulates Transform Payload(s).
- In message (2), the responder sends a SA Payload which indicates the proposal it accepted for the SA it is negotiating.
- In the next two messages the initiator and responder exchange Diffie-Hellman public values[1] and auxiliary data, such as nonces, that are necessary for the

[1]Section 3.2.3 explains the derivation of Diffie-Hellman public values.

exchange.

- In the last two messages, the initiator and responder exchange identification data, and authenticate the Diffie-Hellman exchange. The information transmitted during these two messages are encrypted using the secret key generated from the key material exchanged in messages (3) and (4); consequently, the identity information of the communicating peers are protected. Note that the ISAKMP header is included in all messages.

Aggressive Mode

Aggressive Mode is an implementation of the ISAKMP "Aggressive Mode Exchange". It is used to negotiate ISAKMP Phase 1 SA where identity protection of the communicating peers is not required. This exchange mode allows the SA, Key Exchange and Authentication related Payloads to be transmitted together. Combining these payloads into one message reduces the number of round-trips at the expense of not providing protection for the identity of the communicating peers. Figure 7.13 shows an example of an Aggressive Mode exchange.

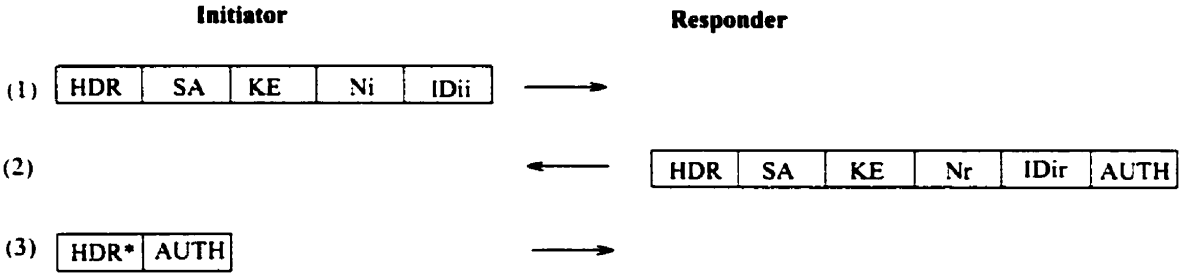

Figure 7.13: Example of an Aggressive Mode Exchange.

- In message (1), the initiator sends the responder a SA Payload which contains a single Proposal Payload encapsulating a Transform Payload: in Aggressive Mode only one proposal with a single transform is offered: the responder has the choice of accepting or rejecting this offer. The Diffie-Hellman public value,

required auxiliary data and identity information are also transmitted in first message.

- In message (2), if responder accepts the initiator's proposal, it sends a SA Payload encapsulating the Proposal Payload containing the proposal and transform the initiator proposed. It also transmit the Diffie-Hellman public value, required auxiliary data and identity information as part of this message. This message is transmitted under the protection of the agreed upon authentication function.

- In message (3), the initiator sends the result of the application of the agreed upon authentication function. This message in effect, authenticates the initiator and provides a proof of its participation in the exchange. The message is encrypted using the key generated from the keying material exchanged in the first two messages. Note, however, that the messages containing the identity information of the peers are not encrypted; consequently, unlike the Main Mode, Aggressive Mode do not provide identity protection for the communicating peers.

Quick Mode

Quick mode is used to negotiate Phase 2 SA under the protection of the negotiated Phase 1 ISAKMP SA. All of the payloads in a Quick Mode exchange are encrypted. Quick Mode exchange can be described as shown below.

	Initiator		Responder
(1)	HDR*, HASH(1), SA, Ni [KE], [IDi, IDr]	--->	
(2)		<---	HDR*, HASH(2), SA, Nr [KE], [IDi, IDr]
(3)	HDR*, HASH(3)	--->	

- In the first message, the initiator sends a Hash, a SA—which encapsulates one or more Proposal Payload(s), which encapsulated one or more Transform Payload(s)—and a Nonce Payload, and optional key exchange material and identification information to the responder. If Perfect Forwarded Secrecy [2] is required then key exchange material must be included in the this message. The Hash Payload contains the message digest—using the negotiated prf— of the Message ID (MsgID) from the ISAKMP header concatenated with the entire message that follows the the Hash Payload, including all payload headers. That is,

  HASH(1) = prf(SKEYSTR_a, MsgID | SA | Ni [| KE] [| IDi | IDr)

  The generation of the keying materials (SKEYSTR or SKEYSTR_*) will be address momentarily.

- The payloads in the second message are similar to those in the first message. The finger print contained in Hash(2) is generated similarly as that for Hash(1), except that the initiator's nonce Ni minus the payload header is added after MsgID but before the complete message.

  HASH(2) = prf(SKEYSTR_a, MsgID | Ni_b | SA | Nr [| KE] [| IDi | IDr)

- The third message is used to authenticate the previous exchanges and it consists of only the ISAKMP header and a Hash Payload. The message digest in the Hash Payload is generated using as input: a byte of 0 concatenated with MsgID,

[2] The term Perfect Forwarded Secrecy is used to describe the phenomenon where the derivation of the keying material is of such that if a single key is comprised, this will not affect the security of the data protected by other keys.

followed by the initiator's nonce minus the payload header, followed by the responder's nonce minus the payload header. That is,
HASH(1) = prf(SKEYSTR_a, 0 | MsgID |Ni_b | Nr_b).

If Perfect Forward Secrecy (PFS) is required, then a KE Payload is included in massage (1) and (2). The new keying material (NewKEYSTR) can be defined as:
$NewKEYSTR = prf(SKEYSTR\_d, g^{xy} \mid protocol \mid SPI \mid Ni\_b \mid Ni\_r)$.
where g^{xy} is the Diffie-Hellman shared secret exchanged in the KE Payload, and "protocol" and "SPI" are obtained from the "Protocol-ID" and "SPI" fields respectively, of the Proposal Payload encapsulated in the SA Payload. When PFS is not required, the KE Payload is not exchanged in Quick Mode messages. In this case, the new keying material can be defined as:
$NewKEYSTR = prf(SKEYSTR\_d, protocol \mid SPI \mid Ni\_b \quad Ni\_r)$.

New Group Mode

The New Group Mode (NGM) is used to negotiate new group for Diffie-Hellman key exchange. NGM is carried out under the protection of ISAKMP Phase 1 exchange. The messages exchange in this exchange mode is shown below.

	Initiator		Responder
(1)	HDR*, HASH(1), SA	--->	
(2)		<---	HDR*, HASH(2), SA

where,
HASH(1) = prf(SKEYSTR_a, MsgID | SA) and
HASH(1) = prf(SKEYSTR_a, MsgID | SA).
The Proposal Payload(s) encapsulated within the SA Payload specify the characteristics of the groups.

Generation of Keying Material

We have discussed the generation of keying material for Quick Mode exchanges; however, we have not yet discussed how keying material are generated for Main and Aggressive Mode exchanges. We will present this information now. Currently, three methods of authentication are allowed for both Main Mode and Aggressive Mode exchanges: digital signature, authentication with public-key encryption and pre-shared key. The value of SKEYSTR depends on the method of authentication.

1. For digital signature
   SKEYSTR = $prf(Ni\_b \mid Nr\_b,\ g^{xy})$
2. For public-key encryption
   SKEYSTR = $prf(Hash(Ni\_b \mid Nr\_b,$ CK-I Ck-R)
3. For pre-shared keys
   SKEYSTR = prf(pre-shared-key, Ni_b | Nr_b)

Both Main Mode and Aggressive Mode exchanges generate three groups of authenticated keying material. These keying material are derived similarly for both exchange modes and they are transported in the "Key Exchange Data" field of the KE Payload. The derivations are as follows:

SKEYSTR_2 = prf(SKEYSTR, g^{xy} | Ck-I | Ck-R | 0)

SKEYSTR_a = prf(SKEYSTR, SKEYSTR_d | g^{xy} | Ck-I | Ck-R | 1)

SKEYSTR_e = prf(SKEYSTR, SKEYSTR_a | g^{xy} | Ck-I | Ck-R | 2)

The values 0, 1 and 2 in the above expressions are the 1-byte representations of these values. The exchanges are authenticated by the generation of HASH_i and HASH_r by the initiator and the responder respectively, of the exchanges. Where

HASH_i = prf(SKEYSTR, $g^x{}_i \mid g^x{}_r$ | Ck-I | Ck-R SAi_b | IDii_b)

HASH_r = prf(SKEYSTR, $g^x{}_r \ \ g^x{}_i$ | Ck-R Ck-I | SAi_b | IDir_b)

The necessary keys for encryption, digital signature and Message Authentication Code algorithms are derived from the keying materials in algorithmic specific man-

ners.

7.3 Oakley Groups

The IKE protocol allows negotiation of the group for Diffie-Hellman exchange. The original specification of IKE [HC98] defined four groups that originated from the Oakley Protocol. These groups are presented below.

First Oakley Group

This group is a modular exponentation group (MEG) and it is the default group for both the Oakley and IKE protocols. The assigned group ID is 1.
The prime is: $2^{768} - 2^{704} - 1 + 2^{64} * \{[2^{638}\pi] + 149686\}$
The generator is : 2

Second Oakley Group

The second group group is also a MEG and it is assigned a group ID of 2.
The prime is: $2^{1024} - 2^{960} - 1 + 2^{64} * \{[2^{894}\pi] + 129093\}$
The generator is : 2

Third Oakley Group

This is an elliptic curve group defined over the Galois Field GF$[2^{155}]$. This group has group ID 3. The field size is 155 and the irreducible polynomial for the field is:
$u^{155} + u^{62} + 1$.
The equation for the elliptic curve is:
$y^2 + xy = x^3 + ax^2 + b$
Group Prime/Irreducible Polynomial: 0x0800000000000000000000004000000000000001
Group Generator 1: 0x7b
Group Curve A: 0x0
Group Curve B: 0x07338f
Group Order: 0X0800000000000000000057db5698537193aef944

When this group is used the, the data in the "Key Exchange Data" field of the KE Payload is the value of x from the solution (x,y), the point on the curve chosen by taking the randomly secret K_a*P, where "*" is the repetition of the group addition and double operations, and P is the point on the curve with x coordinate equals to Generator 1 and the y coordinate determined from the equation of the elliptic curve.

Fourth Oakley Group

This is an elliptic curve group defined over the Galois Field GF[2^{185}], and has group ID 4. The field size is 185. The irreducible polynomial for the field is:

$u^{185} + u^{69} + 1$

The equation for the elliptic curve is:

$y^2 + xy = x^3 + ax^2 + b$

Group Prime/Irreducible Polynomial:

0x020000000000000000000000000020000000000000000001

Group Generator 1: 0x18

Group Curve A: 0x0

Group Curve B: 0x1ee9

Group Order: 0X01ffffffffffffffffffdbf2f889b73e484175f94ebc

When this group is used, the data in the "Key Exchange Data" field of the KE Payload is generated similarly as indicated for the Third Oakley group.

This concludes our discussion of the IKE protocol. For addition information on this protocol, please refer to [HC98] which contains the original specification.

Chapter 8

VPN Solutions

In the previous chapters, we examined the security protocols that comprise the IPSec Protocol suite. In this chapter, we are going to illustrate how these security services can be used to provide different VPN solutions. However, before we proceed, it is necessary to define some terms that we will be using throughout this chapter.

Notations

- VPN: For the purpose of our discussion, Virtual Private Networks (VPN) are secure communication channels which offer data protection via the use of strong cryptographic authentication and/or encryption algorithms. VPN consists of computers (IPSec enabled servers or workstations) and communication links; optionally VPN can also involve routers, switches and security gateways.

- Security Gateway: A security gateway is an access point to a network. A security gateway often includes a firewall (see next entry), and it provides access control; thus only allowing authorized traffic to transverse the network it protects.

- Firewall: A firewall is a packet filtering entity that examines the protocol headers of packets and either accepts the packets and forward them to the destination hosts, or reject them, depending on how the selectors in the packets headers—

source and destination IP address, Transport or Application Layer protocols, source and destination ports, etc.—matches the firewall access control rules.

8.1 Interconnecting Branch Offices

Let us consider a company with has a number of branch offices that are located at different geographical areas. For example, we will assume that the company has offices in Asia, Europe and North America. To interconnect these offices, the company basically has two choices: utilize private network infrastructures or VPN. Let us consider the options for the former: the company will need to either lease dedicated communication channels from telecommunication providers; set up satellite data feed; or installed Trans-Atlantic data channels. The cost of even the most cost-effective implementation of any of these options, is much more prohibitive than a VPN solution; considering that each of these options involves the purchasing or leasing of additional equipment—network cables and supporting infrastructures—plus there are high associated maintenance costs. Whereas, a VPN solution typically involves the purchasing of the VPN software and the necessary licenses and perhaps a few additional computers; and providing the necessary training for Network or Security Administrators who will administer the VPN.

Figure 8.1 illustrates an example of how the branch offices of an enterprise can be interconnected via VPN using the Internet as a backbone network. Each branch office has a security gateway that provides an interface with the Internet and the company's internal network. These security gateways are configured to enforce the access control policies of the respective branch office. There are a number of security service options that can be employed when a host on any of the company's intra-net wishes to communicate with a entity on the intra-net of another branch office. We will examine the option that offers the greatest level of security.

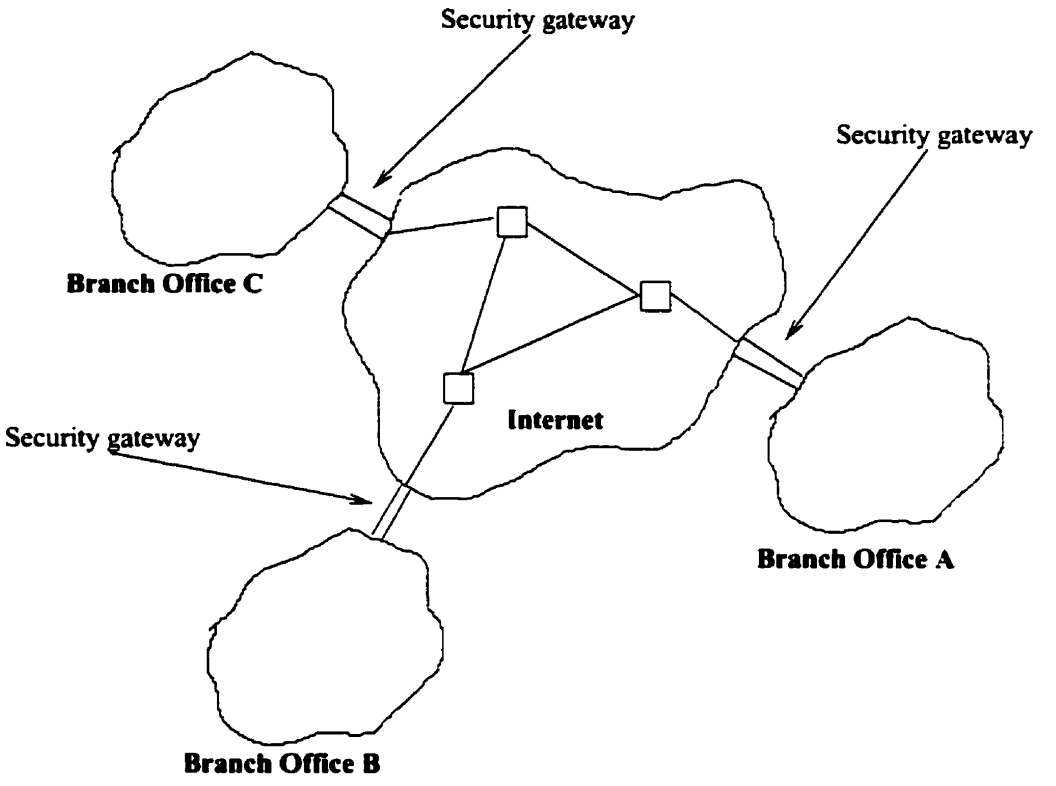

Figure 8.1: VPN interconnecting branches using Internet as backbone network.

End-to-end Authentication and Encryption

This option is employed when a high degree of security for the data traffic is required and the internal network cannot be fully trusted. In employing this option, a host negotiates a Security Association (SA) with a peer on the intra-net of another branch office. The SA will specify the security services required (AH and/or ESP), the mode of operation of these services, and the required authentication and encryption algorithms. Internet Key Exchange protocol (IKE) will then generate and put in place on the communicating peers, the necessary cryptographic keys.

Figure 8.2 illustrates a typical end-to-end secure tunnel connecting communicating peers on the internal network of two branch offices. In this scenario, since there is a security gateway on the intra-net of both branch offices, the hosts will typically negotiate transport mode for the authentication and encryption services that will be

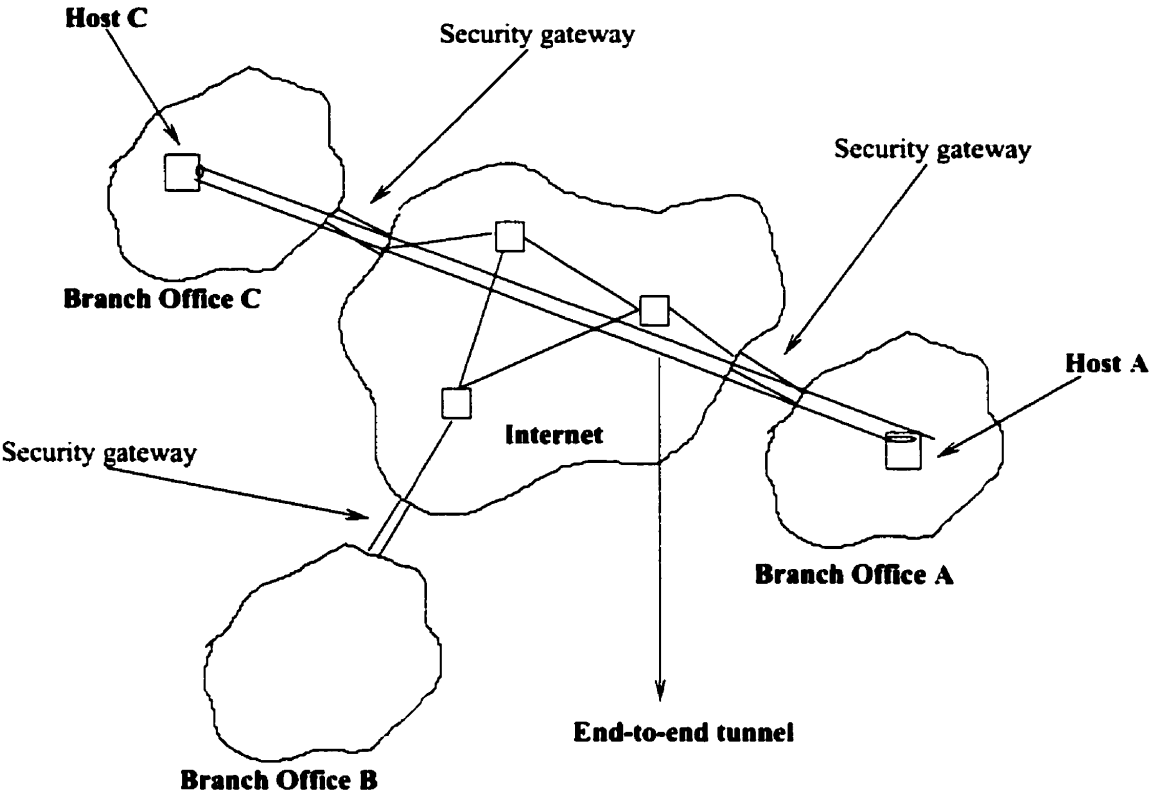

Figure 8.2: Secure end-to-end tunnel connecting communicating peers on internal network of two branch offices.

applied the traffic that they will exchange. As explained in Chapter 5 and 6, transport mode of operation for AH or ESP protects the data that the upper layer protocols encapsulate, but this mode of operation does not protect the IP header. However, when the packet from host A—destined to host C—is forwarded to branch office A's security gate, the security gateway will typically negotiate a SA with the destination host's security gateway, which will involve tunnel mode of operation for the required security services. ESP tunnel mode confidentiality service will protect both the data and the identity of sending entity (host A in our example) while the packet is in transit form one security gateway to the next via the Internet.

When the packet arrives at branch office C's security gateway, it will check its Security Policy Database, first to determine whether or not it should accept the

packet. If the packet should be given access, the security gateway will then consult its SA database to ascertain the SA for the traffic, then authenticates the packet using the authentication algorithm that the SA specifies. If the authentication succeeded, it will forward the packet to the intended destination—host C.

When host C receives the packet it will consult its SA database to identifies the SA that is applicable to this packet, then authenticates the packet to ensure that it has not been modified while it was in transit form the security gateway. Finally, if the authentication succeeded, host C will decrypt the packet using the algorithm specified by the SA for the traffic.

8.2 Interconnecting Different Company's Intra-net

There are situations where it might be necessary for a company to give restrictive access to other company, to its intra-net. Consider for example, a supplier who wishes to give clients access to its database servers that contains relevant information the clients might wish to access. The end-to-end authentication encryption scenario explained in the previous section can also be adopted to this situation as well. In this case, the supplier's security gateway will be configured to accept packets from the domains of its clients, provided that the destination hosts are the entity to which specific access have been given to the respective clients. Apart from addressing issues that might need to be resolved if the companies are using private (global ambiguous) IP addresses, most of the information presented in the previous example apply to this scenario as well.

8.3 Addressing Issues

For both VPN solutions discussed in the previous sections, the enterprises can use either globally unique (public) or globally ambiguous (private) IP addresses. Let us first consider the scenario where VPN is used to interconnect branch offices. If the company uses private IP addresses for the hosts on the intra-nets of the different

branch offices; this will have very little bearing on the effectiveness of the VPN solution, provided that security gateways have unambiguous IP addresses. The security gateways with public IP addresses provide the interface for the hosts on the intra-net with entities on external networks. When a host on any of the internal networks wishes to communicate with an entity outside the company's domain, it forwards the packets it wishes to sent to the its security gateway; the security gateway then modifies the packets' IP header and inserts its IP address in the "source IP address" field, then forward the packet to their intended destination(s). The identities of the private IP addresses for the hosts on the internal network are therefore protected. If an entity on any of the company's intra-nets needs to communicate with another host on another intra-net, and the destination hosts needs to know the identity of the source node, the security gateway can accommodate this requirement, while protecting the identity of the source node from unauthorized entities, by using tunnel mode ESP confidentiality service. As explained in Chapters 6, when ESP is used in tunnel mode, the entire IP datagram is encrypted and a new unencrypted IP header is inserted in front of the original IP header. The new IP header typically contains the security gate IP address as the source IP address.

The same argument can be applied to the case where the hosts on the enterprise's intra-net have unambiguous IP addresses. The identities of the hosts can be protected in a similar manner as outlined above. Similarly, the same holds for the situation where a company needs to give entities on another company's intra-net access to its internal network, except that if either of the enterprises uses ambiguous IP addresses, then the network administrators must ensure the entities that need to communicate do not have the same IP address.

8.4 Routing Issues

Most medium size to large scale VPN will need to employ an IP routing protocol. Routing protocols exchange routing information with neighbouring routers so that each router can learn about the topology of neighbouring networks and be able to

ascertain the most efficient way of reaching hosts that the entities on its network need to communicate with. Consider either of the VPN solutions outlined in this chapter; assume that a routing protocol is deployed on the security gateways. These gateways would need to exchange sensitive information such as the IP addresses of the hosts that are reachable from their network. This information obviously requires protection from unauthorized entities. VPN also provides this protection. Each security gateway can negotiate a SA with its security gateway neighbours and set up a secure communicating channel with each of its neighbours by using ESP security services. Figure 8.3 illustrates how the security gateways can exchange routing information using secure communication channels.

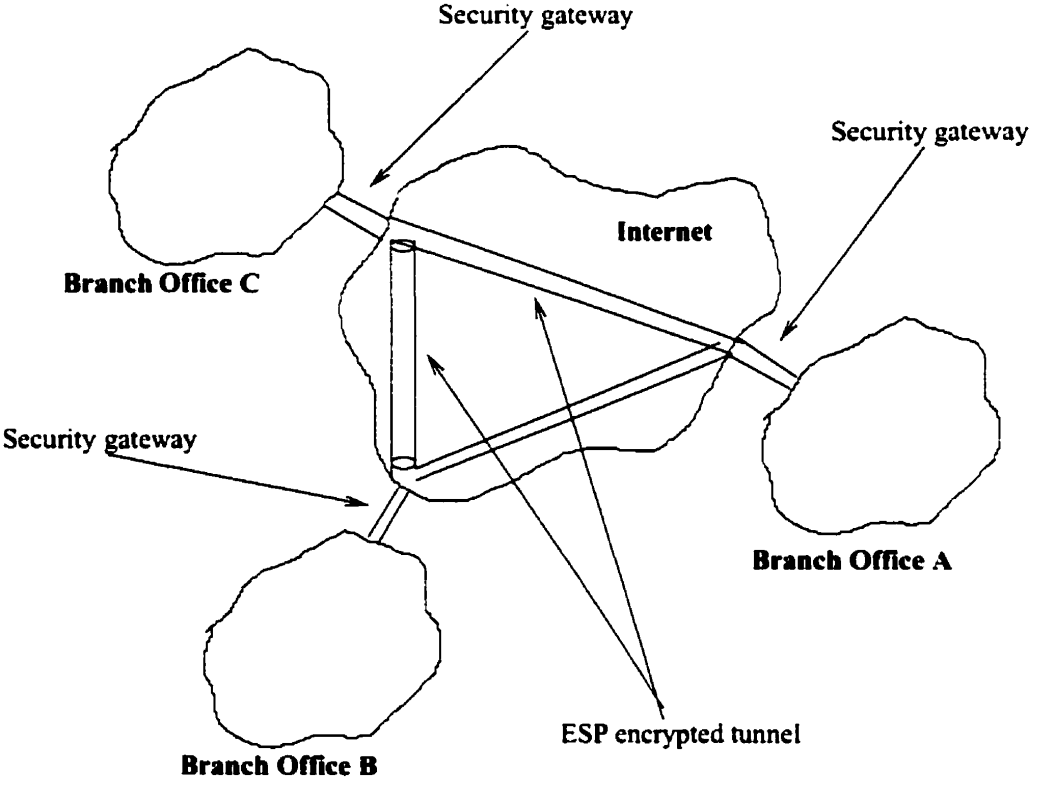

Figure 8.3: Secure communication channels for the exchange of routing information between security gateways.

Since the security gateways are acting as end-hosts, they will typically utilize ESP in transport mode because there is no need to conceal their IP addresses: they are

gateways to their intra-nets, hence their identities must be public. These security gateways can also be configured to exchange non-confidential information with other non-IPSec enabled routers. This can be done by including entries in the Security Policy Database (SPD) that stipulate that packets originated from external networks, whether or not they are protected with any of the IPSec security services. if they are destined to certain service port (such as the ports that routing protocols or Simple Transfer Mail Protocol use), then they can be given restrictive access. This often is necessary in order to facilitate communications such as sending and receiving of electronic mails to and from external hosts.

Evidently, entities that are part of a VPN—whether they are hosts, routers or security gateways—can exchange information in a secure manner while protecting their identities if needs be. The security gateways for VPN can limit access to only traffic that originated from entities that are apart of the VPN. However, as explained above, IPSec VPN are interoperable with non-IPSec enabled networks, since they do not necessarily preclude traffic from the latter.

Chapter 9

Conclusion

The purpose of this research was to present the components of IPSec and to illustrate how these components can be used to provide secure communication channels between communicating entities, using the Internet as backbone network.

First, we gave a brief history of the Internet which covers the period from its inception in the form of an experimental network in 1969, to its current state. We then gave a brief overview of the components of the TCP/IP protocol suite. IPSec can be considered as an extension of the IP protocol. Therefore, in order to understand IPSec, it is necessary to have a firm grasp of the relevant components of the TCP/IP protocol suit.

Next, we presented an overview of cryptographic techniques. The security services that IPSec offer are based on strong cryptographic algorithms; consequently, for an in-depth understanding of IPSec and to be cognizant with the degree of protection that its security service can offer, it is necessary to have a good understanding of the cryptographic techniques that this protocol employs. We gave a detailed presentation of Data Encryption Standard (DES) and Triple-DES, then we introduced some other commonly used private-key cryptographic algorithms and discussed their security. Following this, we presented some public-key encryption algorithms and discussed the performance, in general, of public-key encryption algorithms compared to that of private-key enciphering algorithms. We also presented some commonly used hash functions, message authentication code, and digital signature schemes. We

then examined the IP Security architecture: we explained what IPSec does, we introduced the databases and protocols that IPSec uses, and gave an overview of how the different IPSec entities interact to provide the required security services.

Following this, we presented the AH and ESP protocols. We described their header formats, discussed the modes of operation (transport mode and tunnel mode) and their processing. We then described IKE—the protocol that is used to negotiate SA for the IPSec services; and generates and put in place necessary cryptographic keys that the services require.

Finally, we presented different VPN solutions and illustrated how the IPSec security services can be utilized to provide secure communication channels for data traffic over the Internet backbone.

Currently IPSec base VPN offers excellent protection for unicast hosts, that is, if the address of the host is a single IP address; however, there are more work to be done in order to extend a similar measure of security to hosts in multicast groups. The difficulties lie mainly in the design and implementation of an efficient key exchange protocol for multicast groups. Future research in the domain of IPSec base VPN should address the issue of extending security services currently offered to unicast address to multicast address as well. Also, the processing of IPSec traffic has a relatively high overhead cost. Another possible focus for future IPSec VPN related research, is to find more efficient ways of processing IPSec packets, in order to increase throughput.

Bibliography

[Ada97] Adams, C., "The CAST-128 Encryption Algorithm," Request For Comments (RFC 2144), May 1997.

[Ano98] Anonymous, "Maximum Security A Hacker's Guide to Protect Your Internet and Site Network," Sams Publishing, 2nd ed., August 1998.

[BR96] Baldwin, R., Rivest, R., " The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms," Request for Comments (RFC 2040), October 1996.

[Bar64] Baran. Paul, "On Distributed Communications: I. Introduction to Distributed Communications Network," MEMORANDUM RM-3420-PR. RAND Corporation, August 1964.

[CQ93] Carl-Mitchell, Smoot and Quarterman, John S., "Practical Internetworking with TCP/IP and UNIX," Addison-Wesley Publishing Company, 1993.

[CK74] Cerf, V. and Kahn R., "A Protocol for Packet Network Interconnection," IEEE Trans. on Communication, Vol. COM-22, pp. 637-648, May 1974.

[Cla88] Clark, D.D, "The Design Philosophy of the DARPA Internet Protocols," Proc. SIGCOMM 88 Conf., ACM, pp. 106-114, 1988.

[Dee89] Deering, S.E., "Host Extensions for IP Multicasting," Request For Comments (RFC 1112), August 1989.

[DH95] Deering, S.E. and Hinden, R., "Internet Protocol, Version 6 (IPv6) Specification ," Request For Comments (RFC 1883), December 1995.

[DH76] Diffie, W. and Hellman, M., "Multiusers Cryptographic Techniques," Proceedings of the AFIPS National Computer Conference, June 1976.

[ElG85] ElGamal, T., "A Public Key Cryptosystem and A Signature Scheme Based On Discrete Logarithms," IEE Transactions on Information Theory, Vol. 31, pp. 469-472, 1985.

[Fie73] Feistel, Horst. "Cryptography and Computer Privacy," Scientific America, May 1973.

[FTMT84] Finlayson, R., Timothy, M., Mogul, J., Theimer, M., "A Reverse Address Resolution Protocol," Network Working Group Request For Comments (RFC 903), June 1984.

[FWWD94] ischer, W., Wallmeier, E., Worster, T., Davis, S.P., Hayter, A., "Data Communication Using ATM: Architecture, Protocols and Resource Management." IEEE Communication Magazine, Vol. 32, pp. 24-33, August 1994.

[Gor95] Goralski, W.J., "Introduction to ATM Networking," New York: McGraw-Hill, 1995.

[HC98] Harkins, D., Carrel, D., "Internet Key Exchange (IKE)," Request for Comments (RFC 2409), November 1998.

[Hun98] Hunt, Craig, "TCP/IP Network Administration," O'Reilly and Associates Inc., 2nd ed., 1998.

[IANA00] Internet Assigned Numbers Authority (IANA), "Protocol Numbers and Assigned Services", http://www.iana.org/numbers.html.

[KA98] Kent, S., Atkinson, R., "IP Authentication Header" Request for Comments (RFC 2402), November 1998.

[KBC97] Krawczyk, H., Bellare, M., and Canetti, R., "HMAC: Keyed-Hashing for Message Authentication," Request for Comments (RFC 2104), February 1997.

[Kra96] Krawczyk, H., "SKEME: A versitile Secure Key Exchange Mechanism for Internet", IEEE Proceedings of Symposium on Network and Distributed Systems Security, 1996.

[Kya95] Kyas, O. "ATM Networks," London: International Thompson Publishing, 1995.

[LM90] Lia, X., and Massey, J., "A Proposal for a New Block Encryption Standard," Proceedings, EUROCRYPTO '90, Published by Springer-Verlag, 1990.

[LM91] Lia, X., and Massey, J., "Markov Cipher and Cryptanalysis," Proceedings, EUROCRYPTO '91, Published by Springer-Verlag, 1991.

[LRA92] Latif, A., Rowlance, E.J., and Adams, R.H., "IBM 8209 LAN Bridge," IEEE Network Magazine, Vol. 6, pp. 28-37, May/June 1992.

[LCPM85] Leiner, B.M., Cole, R., Postel, J., and Mills, D., "The DARPA Internet Protocol Suite," IEEE Communication Magazine, Vol 23, pp. 29-34, March 1985.

[MB76] Metcalfe, R. M., and Boggs, D.R., "Ethernet: Distributed Packet Switching for local Computer Networks." Commun. of the ACM, Vol. 19 pp. 395-404. July 1976.

[MSST98] Maughan, D., Schertler, M., Schneider, M., Turner, J., "Internet Security Association and Key Management Protocol." Request for comments (RFC 2408), November 1998.

[NIST93] National Institute of Standard and Technology, "Secure Hash Standard," Federal Information Processing Standards Publication 180-1 (FIPS PUB 180-1), April 17, 1995.

[NIST98] National Institute of Standard and Technology, "Digital Signature Standard (DSS)," Federal Information Processing Standards Publication 186-1 (FIPS PUB 186-1), December 15, 1998.

[NIST99] National Institute of Standard and Technology, "Data Encryption Standard (DES)," Federal Information Processing Standards Publication 46-3 (FIPS PUB 46-3), October 25, 1999.

[NKPC70] Newkirk, J., Kraley, M., Postel, J., and Crocker, S.D, "Prototypical Implementation of the NCP," Network Working Group Request For Comments (RFC 55), June 1970.

[Orm98] Orman, H., "Oakley Key Determination Protocol," Request for comments (RFC 2412), November 1998.

[Plu82] Plummer, David C., "Ethernet Address Resolution Protocol." Network Working Group Request For Comments (RFC826), November 1982.

[Pos80] Postel, J.B., "User Datagram Protocol," Request for comments (RFC 768), August 1980.

[Pos81a] Postel, J.B., "Internet Protocol," Request for comments (RFC 791), September 1981.

[Pos81b] Postel, J.B., "Internet Control Message Protocol." Request for comments (RFC 792), September 1981.

[Pos81c] Postel, J.B., "Transmission Control Protocol," Request for comments (RFC 793), September 1981.

[RP94] Reynolds, J. and Postel, J.B., "Assigned Numbers," Request for comments (RFC 1700), October 1994.

[Riv92a] Rivest, R., "The MD5 Message—Digest Algorithm," Request for comments (RFC 1321), April 1992.

[Riv92b] Rivest, R., "The MD4 Message—Digest Algorithm," Request for comments (RFC 1320), April 1992.

[Riv94] Rivest, R., "RC5 Encryption Algorithm," Proceedings, Second International Workshop on Fast Software Encryption, Published by Springer-Verlag, December 1994.

[RSA78] Rivest, R., Shamir, A., and Adleman, L., "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," Communications of the ACM, February 1978.

[Riv98] Rivest, R., "A Description of the RC2(r) Encryption Algorithm," Request for Comments (RFC 2268), March 1998.

[RSA97] RSA Data Security Inc., "Government Encryption Standard Takes a Fall." RSA Data Press Release, June 17, 1997.

[SA98] Kent, S., Atkinson, R., "IP Encapsulatin Security Payload (ESP)," Request for Comments (RFC 2406), September 1998.

[Sch93] Schneier, B., "Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish)," Proceedings, Workshop on Fast Software Encryption, Published by Springer-Verlag, December 1993.

[Sta99] Stallings, William, "Cryptography And Network Security: Principle and Practice", Prentice Hall Inc., 2nd ed. 1999.

[Ste94] Stevens, W. Richards, "TCP/IP Illustrated, Vol. 1 The Protocols," Addison-Wesley Publishing Company, 1994.

[Sti95] Stinson, D., R., "Cryptography Theory and Practice," CRC Press LLC, 1995.

[Tan96] Tanenbaum, Andrew S., "Computer Networks," Prentice-Hall Inc., 3rd ed., 1996.