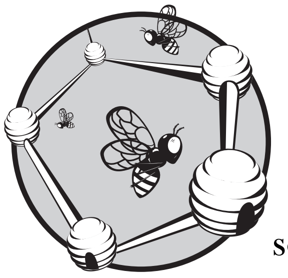
29

IP SECURITY (IPSEC) PROTOCOLS



One of the weaknesses of the original Internet Protocol (IP) is that it lacks any sort of general-purpose mechanism for ensuring the authenticity and privacy of data as it is passed over the internetwork. Since IP datagrams must usually be routed between two devices over unknown networks, any information in them is subject to being intercepted and even possibly changed. With the increased use of the Internet for critical applications, security enhancements were needed for IP. To this end, a set of protocols called *IP Security* or *IPsec* was developed.

In this chapter, I provide a brief description of IPsec concepts and protocols. I begin with an overview of IPsec, including a discussion of the history of the technology and a definition of the standards. I describe the main components and protocols of the IPsec suite and its different architectures and methods for implementation. I then move to actually discussing how IPsec works, beginning with a description of the two IPsec modes (transport and tunnel) and how they differ. I describe security associations and related constructs such as the Security Parameter Index (SPI). The last three topics cover the three main IPsec protocols: IPsec Authentication Header (AH), IPsec Encapsulating Security Payload (ESP), and the IPsec Internet Key Exchange (IKE).

NOTE *IPsec was initially developed with IPv6 in mind, but has been engineered to provide security for both IPv4 and IPv6 networks, and operation in both versions is similar. There are some differences in the datagram formats used for AH and ESP. These differences depend on whether you use IPsec in IPv4 or IPv6, because the two versions have different datagram formats and addressing. I highlight these differences where appropriate.*

IPsec Overview, History, and Standards

The big problem with the original IP version (IPv4) is the pending exhaustion of its address space. This situation arose due to the rapid expansion of the Internet beyond anyone's expectations when IPv4 was developed. This same mismatch between how the Internet was when IPv4 was created and how it is now has led to another major problem with IP: the lack of a definitive means of ensuring security on IP internetworks.

The security problem arose because 25 years ago, the Internet was tiny and relatively private. Today it is enormous and truly public. As the Internet has grown, the need for security has grown with it. Consider that TCP/IP and the early Internet precursors were developed as very small networks used by government researchers at the United States Defense Advanced Research Projects Agency (*DARPA* or *ARPA*). People who were well known and would generally have had security clearance controlled all the hardware. In such a network, you don't need to build security in to the protocols—you build it into the building! It's easier to use locks and guards to ensure security than fancy encryption. The easiest way to keep someone from snooping or tampering with data on the network is simply to deny them access to the hosts that connect to the network.

This worked fine at first when there were only a few dozen machines on the Internet. And even when the Internet first started to grow, it was used pretty much only to connect together researchers and other networking professionals. New sites were added to the network slowly at first, and at least someone knew the identity of each new site added to the growing internetwork. However, as the Internet continued to increase in size and was eventually opened to the public, maintaining security of the network as a whole became impossible. Today, the "great unwashed masses" are on the Internet. Many routers—owned by "who knows" and administered by "who knows"—stand between you and most other devices you want to connect with. You cannot assume that the data you send or receive is secure.

A number of methods have evolved over the years to address the need for security. Most of these are focused at the higher layers of the OSI protocol stack in order to compensate for IP's lack of security. These solutions are valuable for certain situations, but they can't be generalized easily because they are particular to various applications. For example, we can use Secure Sockets Layer (SSL) for certain applications like World Wide Web access or File Transfer Protocol (FTP), but there are dozens of applications that this type of security was never intended to work with.

What was really needed was a solution to allow security at the IP level so all higher-layer protocols in TCP/IP could take advantage of it. When the decision was made to develop a new version of IP (IPv6), this was the golden opportunity to resolve not just the addressing problems in the older IPv4, but the lack of security as well. New security technology was developed with IPv6 in mind, but since IPv6 has taken years to develop and roll out, and the need for security is now, the solution was designed to be usable for both IPv4 and IPv6.

The technology that brings secure communications to the IP is called *IP Security*, commonly abbreviated *IPsec*. The capitalization of this abbreviation is variable, so you'll see IPSec and IPSEC.

Overview of IPsec Services and Functions

IPsec is not a single protocol, but rather a set of services and protocols that provide a complete security solution for an IP network. These services and protocols combine to provide various types of protection. Since IPsec works at the IP layer, it can provide these protections for any higher-layer TCP/IP application or protocol without the need for additional security methods, which is a major strength. Some of the kinds of protection services offered by IPsec include the following:

- Encryption of user data for privacy
- Authentication of the integrity of a message to ensure that it is not changed en route
- Protection against certain types of security attacks, such as replay attacks
- The ability for devices to negotiate the security algorithms and keys required to meet their security needs
- Two security modes, tunnel and transport, to meet different network needs

KEY CONCEPT *IPsec* is a contraction of *IP Security*, and it consists of a set of services and protocols that provide security to IP networks. It is defined by a sequence of several Internet standards.

IPsec Standards

Since IPsec is actually a collection of techniques and protocols, it is not defined in a single Internet standard. Instead, a collection of RFCs defines the architecture, services, and specific protocols used in IPsec. Some of the most important of these are shown in Table 29-1, all of which were published in November 1998.

Table 29-1: Important IP Security (IPsec) Standards

RFC Number	Name	Description
2401	Security Architecture for the Internet Protocol	The main IPsec document, describing the architecture and general operation of the technology, and showing how the different components fit together.
2402	IP Authentication Header	Defines the IPsec Authentication Header (AH) protocol, which is used for ensuring data integrity and origin verification.
2403	The Use of HMAC-MD5-96 within ESP and AH	Describes a particular encryption algorithm for use by the AH and Encapsulation Security Payload (ESP) protocols called Message Digest 5 (MD5), HMAC variant.

(continued)

Table 29-1: Important IP Security (IPsec) Standards (continued)

RFC Number	Name	Description
2404	The Use of HMAC-SHA-1-96 within ESP and AH	Describes a particular encryption algorithm for use by AH and ESP called Secure Hash Algorithm 1 (SHA-1), HMAC variant.
2406	IP Encapsulating Security Payload (ESP)	Describes the IPsec ESP protocol, which provides data encryption for confidentiality.
2408	Internet Security Association and Key Management Protocol (ISAKMP)	Defines methods for exchanging keys and negotiating security associations.
2409	The Internet Key Exchange (IKE)	Describes the IKE protocol that's used to negotiate security associations and exchange keys between devices for secure communications. Based on ISAKMP and OAKLEY.
2412	The OAKLEY Key Determination Protocol	Describes a generic protocol for key exchange.

Deployment of IPsec has only really started to take off in the last few years. A major use of the technology is in implementing virtual private networks (VPNs). It appears that the future is bright for IPsec, as more and more individuals and companies decide that they need to take advantage of the power of the Internet, while also protecting the security of the data they transport over it.

IPsec General Operation, Components, and Protocols

IPsec isn't the only difficult topic in this book, but it is definitely a subject that baffles many. Most discussions of it jump straight to describing the mechanisms and protocols, without providing a general description of what it does and how the pieces fit together. Well, I recognized that IPsec is important, and I don't shy away from a challenge. Thus, here's my attempt to provide a framework for understanding IPsec's various bits and pieces.

So what exactly does IPsec do, and how does it do it? In general terms, it provides security services at the IP layer for other TCP/IP protocols and applications to use. What this means is that IPsec provides the tools that devices on a TCP/IP network need in order to communicate securely. When two devices (either end-user hosts or intermediate devices such as routers or firewalls) want to engage in secure communications, they set up a *secure path* between themselves that may traverse across many insecure intermediate systems. To accomplish this, they must perform (at least) the following tasks:

- They must agree on a set of security protocols to use so that each one sends data in a format the other can understand.
- They must decide on a specific encryption algorithm to use in encoding data.
- They must exchange keys that are used to "unlock" data that has been cryptographically encoded.
- Once this background work is completed, each device must use the protocols, methods, and keys previously agreed upon to encode data and send it across the network.

IPsec Core Protocols

To support these activities, a number of different components make up the total package known as IPsec, as shown in Figure 29-1. The two main pieces are a pair of technologies sometimes called the *core protocols* of IPsec, which actually do the work of encoding information to ensure security:

IPsec Authentication Header (AH) This protocol provides authentication services for IPsec. It allows the recipient of a message to verify that the supposed originator of a message was actually fact the one that sent it. It also allows the recipient to verify that intermediate devices en route haven't changed any of the data in the datagram. It also provides protection against so-called *replay attacks*, whereby a message is captured by an unauthorized user and resent.

Encapsulating Security Payload (ESP) AH ensures the integrity of the data in datagram, but not its privacy. When the information in a datagram is "for your eyes only," it can be further protected using ESP, which encrypts the payload of the IP datagram.

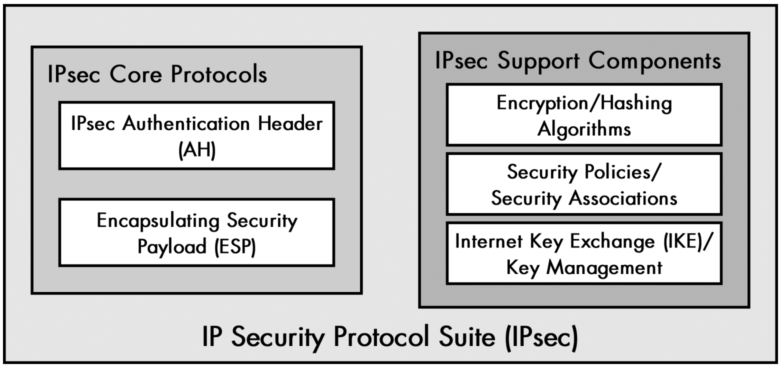

IPsec Core Protocols
IPsec Authentication Header (AH)
Encapsulating Security Payload (ESP)
IPsec Support Components
Encryption/Hashing Algorithms
Security Policies/ Security Associations
Internet Key Exchange (IKE)/ Key Management
IP Security Protocol Suite (IPsec)

Figure 29-1: Overview of IPsec protocols and components *IPsec consists of two core protocols, AH and ESP, and three supporting components.*

IPsec Support Components

AH and ESP are commonly called *protocols*, though this is another case where the use of this term is debatable. They are not really distinct protocols but are implemented as headers that are inserted into IP datagrams, as you will see. They thus do the "grunt work" of IPsec, and can be used together to provide both authentication and privacy. However, they cannot operate on their own. To function properly, they need the support of several other protocols and services (see Figure 29-1). The most important of these include the following:

Encryption/Hashing Algorithms AH and ESP are generic and do not specify the exact mechanism used for encryption. This gives them the flexibility to work with a variety of such algorithms and to negotiate which one to use as needed. Two common ones used with IPsec are *Message Digest 5 (MD5)* and *Secure Hash Algorithm 1 (SHA-1)*. These are also called *hashing* algorithms because they work by computing a formula called a *hash* based on input data and a key.

Security Policies, Security Associations, and Management Methods Since IPsec provides flexibility in letting different devices decide how they want to implement security, they require some means to keep track of the security relationships between themselves. This is done in IPsec using constructs called *security policies* and *security associations*, and by providing ways to exchange security association information.

Key Exchange Framework and Mechanism For two devices to exchange encrypted information, they need to be able to share keys for unlocking the encryption. They also need a way to exchange security association information. In IPsec, a protocol called the *Internet Key Exchange (IKE)* provides these capabilities.

KEY CONCEPT IPsec consists of a number of different components that work together to provide security services. The two main ones are protocols called the *Authentication Header (AH)* and *Encapsulating Security Payload (ESP)*, which provide authenticity and privacy to IP data in the form of special headers added to IP datagrams.

Well, that's at least a start at providing a framework for understanding what IPsec is all about and how the pieces fit together. You'll examine these components and protocols in more detail as you proceed through this chapter.

IPsec Architectures and Implementation Methods

The main reason that IPsec is so powerful is that it provides security to IP, which is the basis for all other TCP/IP protocols. In protecting IP, you are protecting pretty much everything else in TCP/IP as well. An important issue, then, is how exactly do you get IPsec into IP? There are several implementation methods for deploying IPsec. These represent different ways that IPsec may modify the overall layer architecture of TCP/IP.

Three different implementation architectures are defined for IPsec in RFC 2401. The one you use depends on various factors including the version of IP used (IPv4 or IPv6), the requirements of the application, and other factors. These, in turn, rest on a primary implementation decision: Should IPsec be programmed into all hosts on a network, or just into certain routers or other intermediate devices? This is a design decision that must be based on the requirements of the network:

End-Host Implementation Putting IPsec into all host devices provides the most flexibility and security. It enables end-to-end security between any two devices on the network. However, there are many hosts on a typical network, so this means far more work than just implementing IPsec in routers.

Router Implementation This option is much less work because it means you make changes to only a few routers instead of hundreds or thousands of clients. It provides protection only between pairs of routers that implement IPsec, but this may be sufficient for certain applications such as VPNs. The routers can be used to provide protection for just the portion of the route that datagrams take outside the organization, thereby leaving connections between routers and local hosts unsecured (or possibly, secured by other means).

Three different architectures are defined that describe methods for how to get IPsec into the TCP/IP protocol stack: integrated, bump in the stack, and bump in the wire.

Integrated Architecture

Under ideal circumstances, we would integrate IPsec's protocols and capabilities directly into IP itself. This is the most elegant solution, because it allows all IPsec security modes and capabilities to be provided just as easily as regular IP. No extra hardware or architectural layers are needed.

IPv6 was designed to support IPsec. Thus, it's a viable option for hosts or routers. With IPv4, integration would require making changes to the IP implementation on each device, which is often impractical (to say the least!).

Bump in the Stack (BITS) Architecture

In the bump in the stack (BITS) technique, IPsec is made a separate architectural layer between IP and the data link layer. The cute name refers to the fact that IPsec is an extra element in the networking protocol stack, as you can see in Figure 29-2. IPsec intercepts IP datagrams as they are passed down the protocol stack, provides security, and passes them to the data link layer.

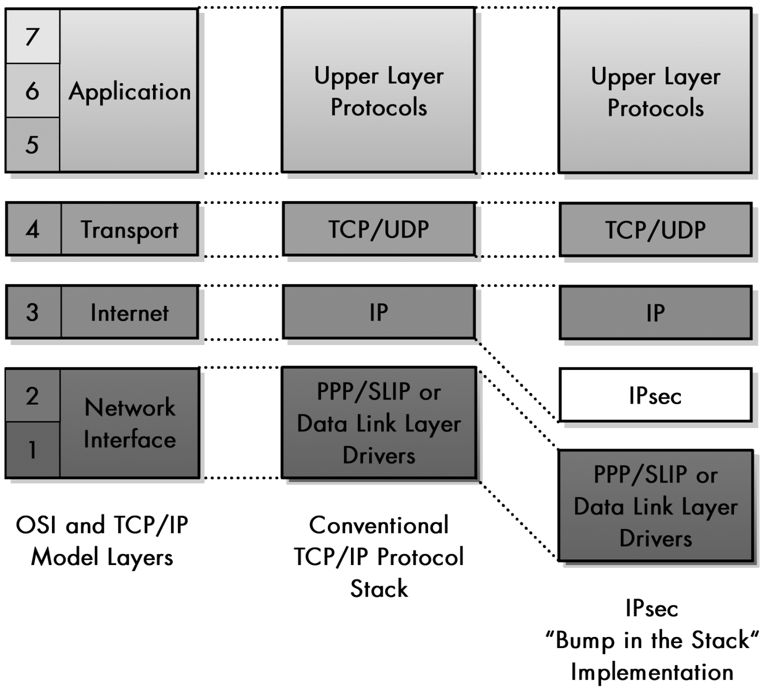

7
6
5
Application
4
Transport
3
Internet
2
1
Network Interface
OSI and TCP/IP Model Layers
Upper Layer Protocols
TCP/UDP
IP
PPP/SLIP or Data Link Layer Drivers
Conventional TCP/IP Protocol Stack
Upper Layer Protocols
TCP/UDP
IP
IPsec
PPP/SLIP or Data Link Layer Drivers
IPsec "Bump in the Stack" Implementation

Figure 29-2: IPsec bump in the stack (BITS) architecture *In this type of IPsec implementation, IPsec becomes a separate layer in the TCP/IP stack. It is implemented as software that sits below IP and adds security protection to datagrams created by the IP layer.*

The advantage of this technique is that IPsec can be retrofitted to any IP device, since the IPsec functionality is separate from IP. The disadvantage is that there is a duplication of effort compared to the integrated architecture. BITS is generally used for IPv4 hosts.

Bump in the Wire (BITW) Architecture

In the bump in the wire (BITW) method, we add a hardware device that provides IPsec services. For example, suppose we have a company with two sites. Each has a network that connects to the Internet using a router that is not capable of IPsec functions. We can interpose a special IPsec device between the router and the Internet at both sites, as shown in Figure 29-3. These devices will then intercept outgoing datagrams, add IPsec protection to them, and strip it off incoming datagrams.

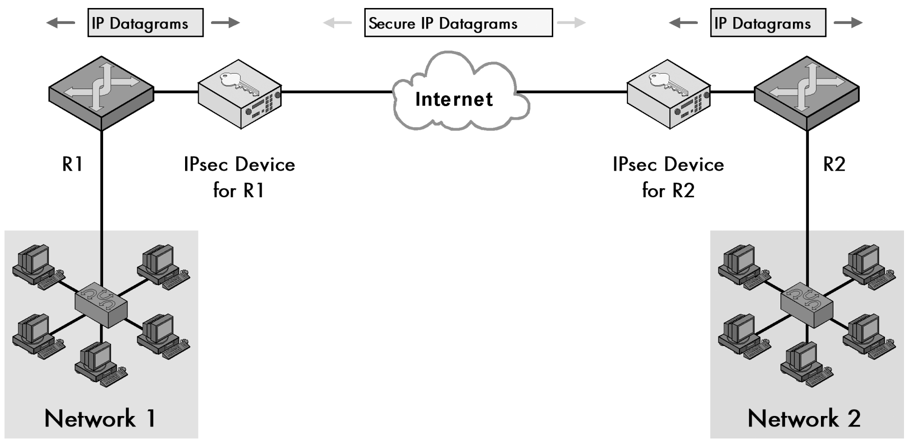


Figure 29-3: IPsec bump in the wire (BITW) architecture *In this IPsec architecture, IPsec is actually implemented in separate devices that sit between the devices that wish to communicate securely. These repackage insecure IP datagrams for transport over the public Internet.*

Just as BITS lets you add IPsec to legacy hosts, BITW can retrofit non-IPsec routers to provide security benefits. The disadvantages are complexity and cost.

KEY CONCEPT Three different architectures or implementation models are defined for IPsec. The best is integrated architecture, in which IPsec is built into the IP layer of devices directly. The other two are *bump in the stack (BITS)* and *bump in the wire (BITW)*, which are ways of layering IPsec underneath regular IP, using software and hardware solutions, respectively.

As you will see in the next section, the choice of architecture has an important impact on which of the two IPsec modes can be used. Incidentally, even though BITS and BITW seem quite different, they are actually do the same thing. In the case of BITS, we have an extra software layer that adds security to existing IP datagrams; in BITW, distinct hardware devices do this same job. In both cases, the result is the same, and the implications on the choice of IPsec mode is likewise the same.

IPsec Modes: Transport and Tunnel

You just saw that three different basic implementation architectures could be used to provide IPsec facilities to TCP/IP networks. The choice of which implementation you use, as well as whether you implement in end hosts or routers, impacts the specific way that IPsec functions. Two specific modes of operation that are related to these architectures are defined for IPsec. They are called *transport mode* and *tunnel mode*.

IPsec modes are closely related to the function of the two core protocols, AH and ESP. Both of these protocols provide protection by adding a header (and possibly other fields) containing security information to a datagram. The choice of mode does not affect the method by which each generates its header, but rather, changes what specific parts of the IP datagram are protected and how the headers are arranged to accomplish this. In essence, the mode really describes, not prescribes, how AH or ESP do their thing. It is used as the basis for defining other constructs, such as security associations (SAs).

Transport Mode

As its name suggests, in transport mode, the protocol protects the message passed down to IP from the transport layer. The message is processed by AH and/or ESP, and the appropriate header(s) are added in front of the transport (UDP or TCP) header. The IP header is then added in front of that by IP.

Another way of looking at this is as follows: Normally, the transport layer packages data for transmission and sends it to IP. From IP's perspective, this transport layer message is the payload of the IP datagram. When IPsec is used in transport mode, the IPsec header is applied only over this IP payload, not the IP header. The AH and ESP headers appear between the original, single IP header and the IP payload. This is illustrated in Figure 29-4.

Tunnel Mode

In tunnel mode, IPsec is used to protect a completely encapsulated IP datagram after the IP header has already been applied to it. The IPsec headers appear in front of the original IP header, and then a new IP header is added in front of the IPsec header. That is to say, the entire original IP datagram is secured and then encapsulated within another IP datagram. This is shown in Figure 29-5.

Comparing Transport and Tunnel Modes

The bottom line in understanding the difference between the two IPsec modes is this: Tunnel mode protects the original IP datagram as a whole, header and all, while transport mode does not. Thus, in general terms, the order of the headers is as follows:

Transport Mode IP header, IPsec headers (AH and/or ESP), IP payload (including transport header)

Tunnel Mode New IP header, IPsec headers (AH and/or ESP), old IP header, IP payload

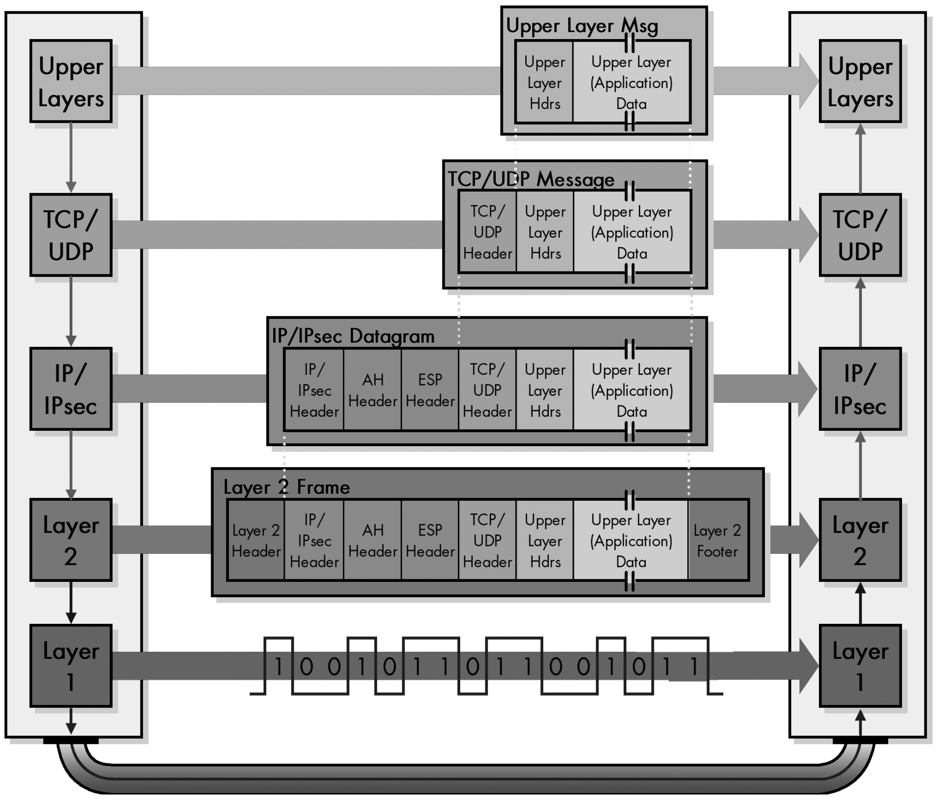


Figure 29-4: IPsec transport mode operation *When IPsec operates in transport mode, it is integrated with IP and used to transport the upper layer (TCP/UDP) message directly. After processing, the datagram has just one IP header that contains the AH and ESP IPsec headers. Contrast this to tunnel mode, shown in Figure 29-5.*

Again, this is a simplified view of how IPsec datagrams are constructed; the reality is significantly more complex. The exact way that the headers are arranged in an IPsec datagram in both transport and tunnel modes depends on which version of IP is being used. IPv6 uses extension headers that must be arranged in a particular way when IPsec is used. The header placement also depends on which IPsec protocol is being used, AH or ESP. Note that it is also possible to apply both AH and ESP to the same datagram; if so, the AH header always appears before the ESP header.

There are thus three variables and eight basic combinations of mode (tunnel or transport), IP version (IPv4 or IPv6) and protocol (AH or ESP). The coming discussions of AH and ESP describe the four format combinations of transport/ tunnel mode and IPv4/IPv6 applicable to each protocol. Note that ESP also includes an ESP trailer that goes after the data protected.

You could probably tell by reading these descriptions how the two modes relate to the choice of IPsec architecture you looked at earlier. Transport mode requires that IPsec be integrated into IP, because AH/ESP must be applied as the original IP packaging is performed on the transport layer message. This is often the choice for implementations requiring end-to-end security with hosts that run IPsec directly.

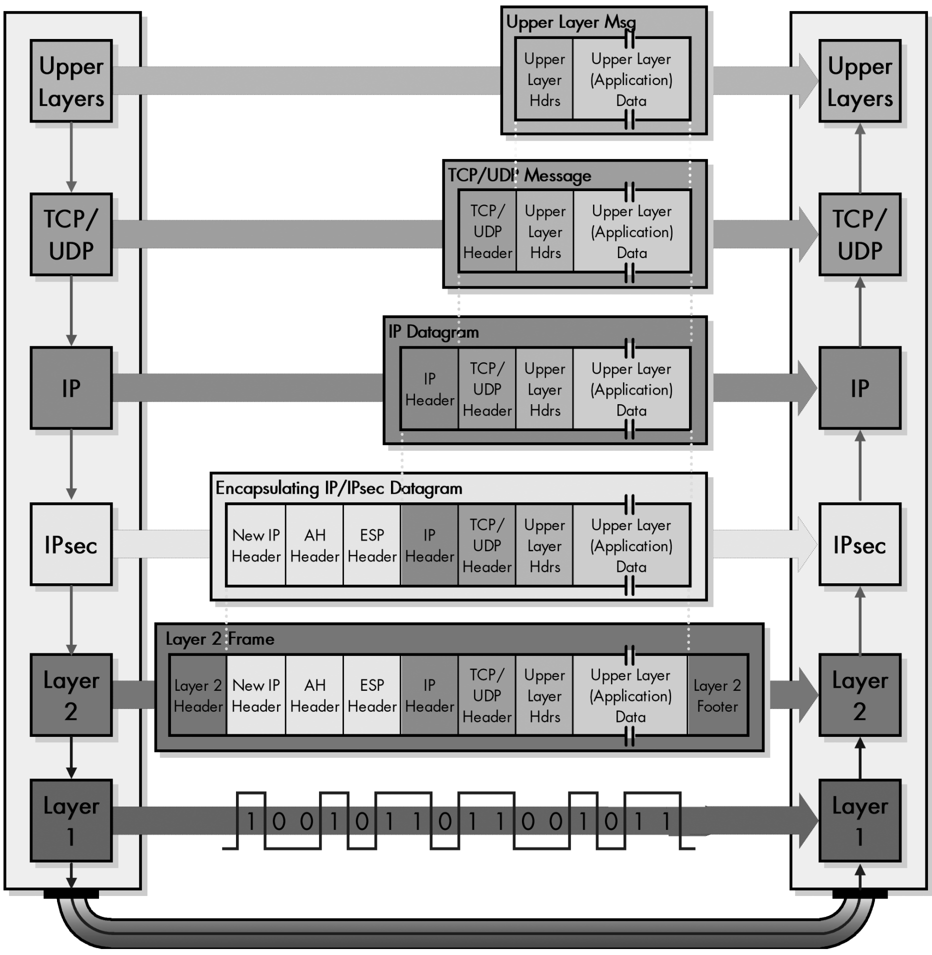


Figure 29-5: IPsec tunnel mode operation *IPsec* tunnel mode *is so named because it represents an encapsulation of a complete IP datagram, thereby forming a virtual tunnel between IPsec-capable devices. The IP datagram is passed to IPsec, where a new IP header is created with the AH and ESP IPsec headers added. Contrast this to transport mode, shown in Figure 29-4.*

Tunnel mode represents an encapsulation of IP within the combination of IP plus IPsec. Thus, it corresponds with the BITS and BITW implementations, where IPsec is applied after IP has processed higher-layer messages and has already added its header. Tunnel mode is a common choice for VPN implementations, which are based on the tunneling of IP datagrams through an unsecured network such as the Internet.

KEY CONCEPT IPsec has two basic modes of operation. In *transport mode*, IPsec AH and ESP headers are added as the original IP datagram is created. Transport mode is associated with integrated IPsec architectures. In *tunnel mode*, the original IP datagram is created normally, and then the entire datagram is encapsulated into a new IP datagram containing the AH/ESP IPsec headers. Tunnel mode is most commonly used with *bump in the stack (BITS)* and *bump in the wire (BITW)* implementations.

IPsec Security Constructs

Important IPsec security constructs include security associations, the security association database, security policies, the security policy database, selectors, and the security parameter index. These items are all closely related and essential to understand before you begin looking at the core IPsec protocols. These constructs are used to guide the operation of IPsec in a general way and particularly to guide exchanges between devices. The constructs control how IPsec works and ensure that each datagram coming into or leaving an IPsec-capable device is treated properly.

Security Policies, Security Associations, and Associated Databases

Let's begin by considering the problem of how to apply security in a device that may be handling many different exchanges of datagrams with others. There is overhead involved in providing security, so you do not want to do it for every message that comes in or out. Some types of messages may need more security; others may need less. Also, exchanges with certain devices may require different processing than others.

To manage all of this complexity, IPsec is equipped with a flexible, powerful way of specifying how different types of datagrams should be handled. To understand how this works, you must first define the following two important logical concepts:

Security Policies and the Security Policy Database (SPD) A *security policy* is a rule that is programmed into the IPsec implementation. It tells the implementation how to process different datagrams received by the device. For example, security policies decide if a particular packet needs to be processed by IPsec or not. AH and ESP entirely bypass those that do not need processing. If security is required, the security policy provides general guidelines for how it should be provided, and if necessary, links to more specific detail. Security policies for a device are stored in the device's *security policy database (SPD).*

Security Associations (SAs) and the Security Association Database (SAD) A security association (SA) is a set of security information that describes a particular kind of secure connection between one device and another. You can consider it a contract, if you will, that specifies the particular security mechanisms that are used for secure communications between the two. A device's security associations are contained in its *security association database (SAD).*

It's often hard to distinguish between the SPD and the SAD, because they are similar in concept. The main difference between them is that security policies are general, while security associations are more specific. To determine what to do with a particular datagram, a device first checks the SPD. The security policies in the SPD may reference a particular SA in the SAD. If so, the device will look up that SA and use it for processing the datagram.

Selectors

One issue I haven't covered yet is how a device determines what security policies or SAs to use for a specific datagram. Again here, IPsec defines a very flexible system that lets each security association define a set of rules for choosing datagrams that the SA applies to. Each of these rule sets is called a *selector*. For example, you might define a selector that says that a particular range of values in the Source Address of a datagram, combined with another value in the Destination Address, means that a specific SA must be used for the datagram.

Security Association Triples and Security Parameter Index (SPI)

Each secure communication that a device makes to another requires that an SA be established. SAs are unidirectional, so each one only handles either inbound or outbound traffic for a particular device. This allows the level of security for a flow from Device A to Device B to be different than the level for traffic coming from Device B to Device A. In a bidirectional communication of this sort, both Device A and Device B would have two SAs; Device A would have SAs that you could call SAdeviceBin and SAdeviceBout. Device B would have SAs SAdeviceAin and SAdeviceAout.

SAs don't actually have names, however. They are instead defined by a set of three parameters, called a *triple*:

Security Parameter Index (SPI) A 32-bit number that is chosen to uniquely identify a particular SA for any connected device. The SPI is placed in AH or ESP datagrams and thus links each secure datagram to the security association. It is used by the recipient of a transmission so it knows what SA governs the datagram.

IP Destination Address The address of the device for which the SA is established.

Security Protocol Identifier Specifies whether this association is for AH or ESP. If both are in use with this device, they have separate SAs.

As you can see, the two security protocols AH and ESP are dependent on SAs, security policies, and the various databases that control the operation of those SAs and policies. Management of these databases is important, but it's another complex subject entirely. Generally, SAs can either be set up manually (which is of course extra work) or you can deploy an automated system using a protocol like IKE (discussed near the end of this chapter).

Confused? I don't blame you, despite my best efforts, and remember that this is all highly simplified. Welcome to the wonderful world of networking security. If you are ever besieged by insomnia, I highly recommend RFC 2401!

IPsec Authentication Header (AH)

As I mentioned earlier in this chapter, AH is one of the two core security protocols in IPsec. This is another protocol whose name has been well chosen. It provides *authentication* of either all or part of the contents of a datagram through the addition of a *header* that is calculated based on the values in the datagram. The parts of the datagram that are used for the calculation, and the placement of the header, depend on the mode (tunnel or transport) and the version of IP (IPv4 or IPv6).

The operation of AH is surprisingly simple, especially for any protocol that has anything to do with network security. The simplicity is analogous to the algorithms used to calculate checksums or perform cyclic redundancy (CRC) checks for error detection. In those cases, the sender uses a standard algorithm to compute a checksum or CRC code based on the contents of a message. This computed result is transmitted along with the original data to the destination, which repeats the calculation and discards the message if any discrepancy is found between its calculation and the one done by the source.

This is the same idea behind AH, except that instead of using a simple algorithm known to everyone, it uses a special hashing algorithm and a specific key known only to the source and the destination. An SA between two devices specifies these particulars, so that the source and destination know how to perform the computation but nobody else can. On the source device, AH performs the computation and puts the result (called the *integrity check value*, or *ICV*) into a special header with other fields for transmission. The destination device does the same calculation using the key that the two devices share. This enables the device to see immediately if any of the fields in the original datagram were modified (due to either error or malice).

Just as a checksum doesn't change the original data, neither does the ICV calculation change the original data. The presence of the AH header allows us to verify the integrity of the message, but doesn't encrypt it. Thus, AH provides *authentication* but not *privacy* (that's what ESP is for).

AH Datagram Placement and Linking

The calculation of AH is similar for both IPv4 and IPv6. One difference is in the exact mechanism used for placing the header into the datagram and for linking the headers together. I'll describe IPv6 first because it is simpler, and because AH was really designed to fit into its mechanism for this.

IPv6 AH Placement and Linking

In IPv6, the AH is inserted into the IP datagram as an extension header, following the normal IPv6 rules for extension header linking. It is linked by the previous header (extension or main), which puts the assigned value for the AH header (51) into its Next Header field. The AH header then links to the next extension header or the transport layer header using its Next Header field.

In transport mode, the AH is placed into the main IP header and appears before any Destination Options header that contains options intended for the final destination, and before an ESP header if present, but after any other extension headers. In tunnel mode, it appears as an extension header of the new IP datagram that encapsulates the original one being tunneled. This is shown graphically in Figure 29-6.

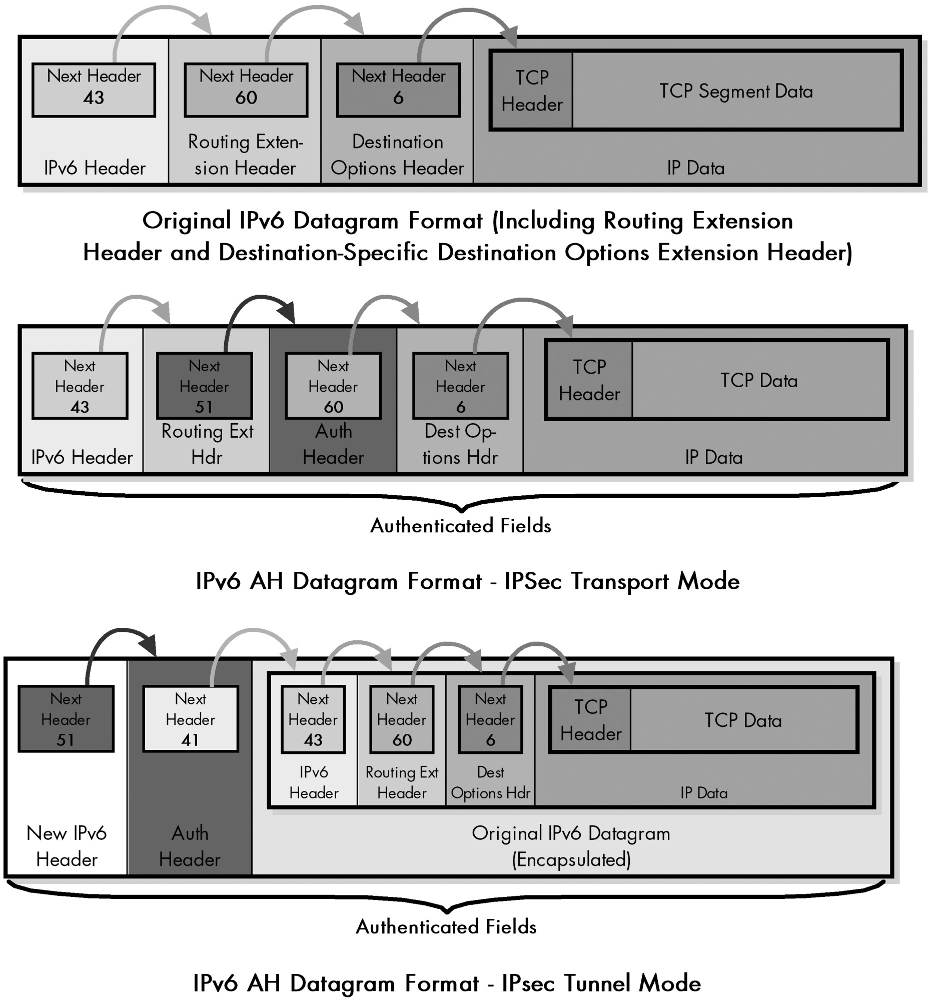


Figure 29-6: IPv6 datagram format with IPsec Authentication Header (AH) *This is an example of an IPv6 datagram with two extension headers that are linked using the standard IPv6 mechanism (see Figure 26-3 in Chapter 26). When AH is applied in transport mode, it is simply added as a new extension header (as shown in dark shading) that goes between the Routing extension header and the Destination Options header. In tunnel mode, the entire original datagram is encapsulated into a new IPv6 datagram that contains the AH header. In both cases, the Next Header fields are used to link each header one to the next. Note the use of Next Header value 41 in tunnel mode, which is the value for the encapsulated IPv6 datagram.*

IPv4 AH Placement and Linking

In IPv4, a method that is similar to the IPv6 header-linking technique is employed. In an IPv4 datagram, the Protocol field indicates the identity of the higher-layer protocol (typically TCP or UDP) that's carried in the datagram. As such, this field points to the next header, which is at the front of the IP payload. AH takes this value and puts it into its Next Header field, and then places the protocol value for AH itself (51 in dotted decimal) into the IP Protocol field. This makes the IP header point to the AH, which then points to whatever the IP datagram pointed to before.

Again, in transport mode, the AH header is added after the main IP header of the original datagram; in tunnel mode it is added after the new IP header that encapsulates the original datagram that's being tunneled. This is shown in Figure 29-7.

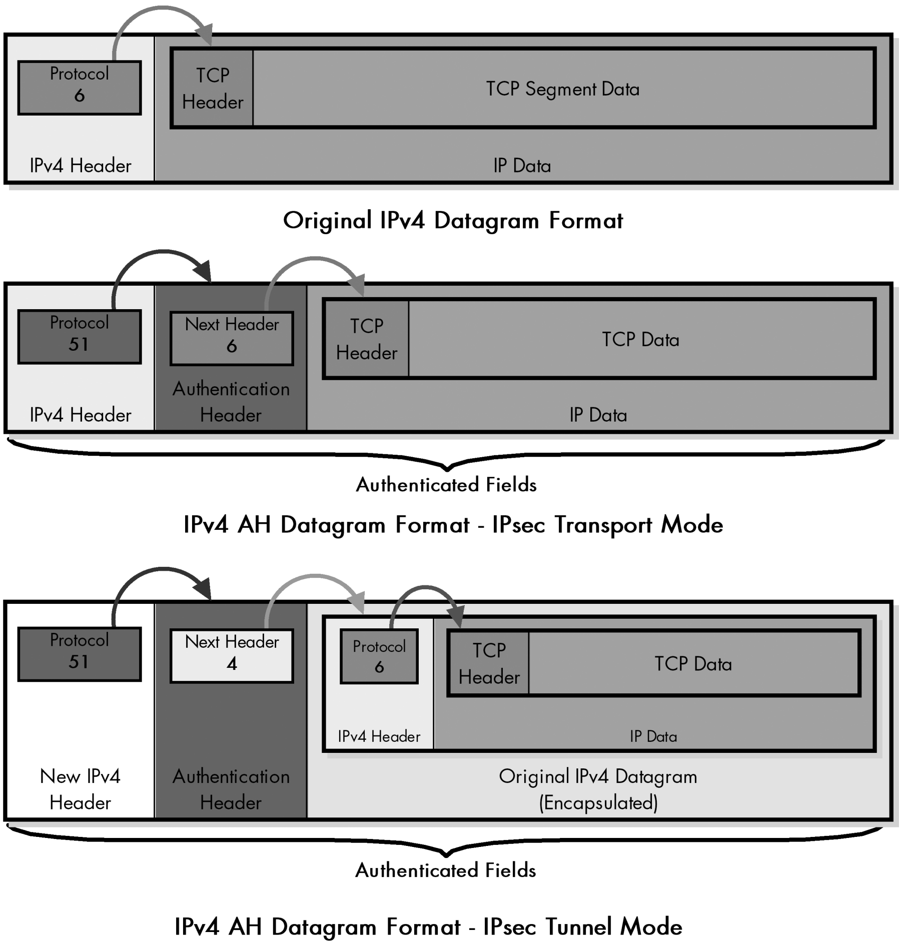


Figure 29-7: IPv4 datagram format with IPsec AH *Here is an example of an IPv4 datagram; it may or may not contain IPv4 options (which are not distinct entities as they are in IPv6). In transport mode, the AH header is added between the IP header and the IP data; the Protocol field of the IP header points to it, while its Next Header field contains the IP header's prior protocol value (in this case 6, for TCP). In tunnel mode, the IPv4 datagram is encapsulated into a new IPv4 datagram that includes the AH header. Note that in tunnel mode, the AH header uses the value 4 (which means IPv4) in its Next Header field.*

KEY CONCEPT The IPsec *Authentication Header (AH)* protocol allows the recipient of a datagram to verify its authenticity. It is implemented as a header that's added to an IP datagram that contains an *integrity check value (ICV)*, which is computed based on the values of the fields in the datagram. The recipient can use this value to ensure that the data has not been changed in transit. AH does not encrypt data and thus does not ensure the privacy of transmissions.

AH Format

The format of AH is described in Table 29-2 and illustrated in Figure 29-8.

Table 29-2: IPsec Authentication Header (AH) Format

Field Name	Size (Bytes)	Description
Next Header	1	Contains the protocol number of the next header after the AH. Used to link headers together.
Payload Len	1	Despite its name, this field measures the length of the authentication header itself, not the payload. (I wonder what the history is behind that!) It is measured in 32-bit units, with 2 subtracted for consistency with how header lengths are normally calculated in IPv6.
Reserved	2	Not used; set to zeros.
SPI	4	A 32-bit value that, when combined with the destination address and security protocol type (which is obviously the one for AH here), identifies the security association (SA) that will be used for this datagram. (SAs are discussed earlier in this chapter.)
Sequence Number	4	A counter field that is initialized to zero when an SA is formed between two devices, and then incremented for each datagram sent using that SA. This uniquely identifies each datagram on an SA and is used to provide protection against replay attacks by preventing the retransmission of captured datagrams.
Authentication Data	Variable	Contains the result of the hashing algorithm, called the integrity check value (ICV), performed by the AH protocol.

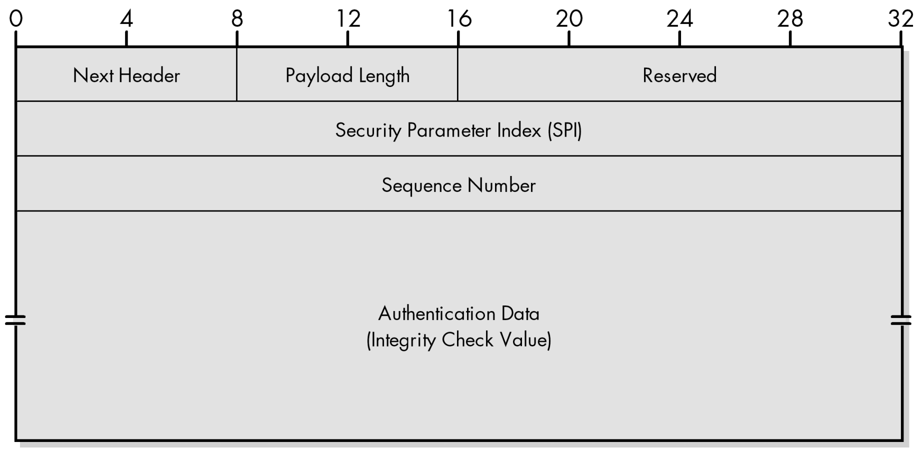


Figure 29-8: *IPsec Authentication Header (AH) format*

The size of the Authentication Data field is variable to support different datagram lengths and hashing algorithms. Its total length must be a multiple of 32 bits. Also, the entire header must be a multiple of either 32 bits (for IPv4) or 64 bits (for IPv6), so additional padding may be added to the Authentication Data field if necessary.

You may also notice that no IP addresses appear in the header, which is a prerequisite for it being the same for both IPv4 and IPv6.

IPsec Encapsulating Security Payload (ESP)

The IPsec AH provides integrity authentication services to IPsec-capable devices so that they can verify that messages are received intact from other devices. For many applications, however, this is only one piece of the puzzle. We want to not only protect against intermediate devices changing the datagrams, but also to protect against them examining their contents as well. For this level of private communication, AH is not enough; we need to use the ESP protocol.

The main job of ESP is to provide the privacy we seek for IP datagrams by encrypting them. An encryption algorithm combines the data in the datagram with a key to transform it into an encrypted form. This is then repackaged using a special format that you will see shortly, and then transmitted to the destination, which decrypts it using the same algorithm. ESP also sports its own authentication scheme like the one used in AH, or it can be used in conjunction with AH.

ESP Fields

ESP has several fields that are the same as those used in AH, but it packages its fields in a very different way. Instead of having just a header, it divides its fields into three components:

ESP Header This contains two fields, SPI and Sequence Number, and comes before the encrypted data. Its placement depends on whether ESP is used in transport mode or tunnel mode, as explained earlier in this chapter.

ESP Trailer This section is placed after the encrypted data. It contains padding that is used to align the encrypted data through a Padding and Pad Length field. Interestingly, it also contains the Next Header field for ESP.

ESP Authentication Data This field contains an ICV that's computed in a manner that's similar to how the AH protocol works. The field is used when ESP's optional authentication feature is employed.

There are two reasons why these fields are broken into pieces like this. The first is that some encryption algorithms require the data to be encrypted to have a certain block size, and so padding must appear after the data and not before it. That's why padding appears in the ESP Trailer field. The second is that the ESP Authentication Data appears separately because it is used to authenticate the rest of the encrypted datagram after encryption. This means that it cannot appear in the ESP Header or ESP Trailer.

ESP Operations and Field Use

This is still a bit boggling so I'm going to try to explain this procedurally by considering three basic steps performed by ESP: calculating the header, then the trailer, and then the Authentication field.

Header Calculation and Placement

The first thing to consider is how the ESP header is placed. This is similar to how AH works and depends on the IP version, as follows:

IPv6 The ESP Header field is inserted into the IP datagram as an extension header, following the normal IPv6 rules for extension-header linking. In transport mode, it appears before a Destination Options header that contains options intended for the final destination, but after any other extension headers, if present. In tunnel mode, it appears as an extension header of the new IP datagram that encapsulates the original one being tunneled. This is shown in Figure 29-9.

IPv4 As with AH, the ESP Header field is placed after the normal IPv4 header. In transport mode, it appears after the IP header of the original datagram; in tunnel mode, it appears after the IP header of the new IP datagram that's encapsulating the original one. You can see this in Figure 29-10.

Trailer Calculation and Placement

The ESP Trailer field is appended to the data that will be encrypted. ESP then performs the encryption. The payload (TCP/UDP message or encapsulated IP datagram) and the ESP trailer are both encrypted, but the ESP header is not. Note again that any other IP headers that appear between the ESP header and the payload are also encrypted. In IPv6, this can include a Destination Options extension header.

Normally, the Next Header field would appear in the ESP Header and would be used to link the ESP Header to the header that comes after it. However, the Next Header field in ESP appears in the trailer and not the header, which makes the linking seem a bit strange in ESP. The method is basically the same as what's used in AH and in IPv6 in general, with the Next Header and Protocol fields being used to tie everything together. However, in ESP the Next Header field appears *after* the encrypted data, and so it points back to one of the following: a Destination Options extension header (if present), a TCP/UDP header (in transport mode), or an IPv4/IPv6 header (in tunnel mode). This is also shown in Figures 29-9 and 29-10.

ESP Authentication Field Calculation and Placement

If the optional ESP authentication feature is being used, it is computed over the entire ESP datagram (except the Authentication Data field itself, of course). This includes the ESP header, payload, and trailer.

KEY CONCEPT The IPsec ESP protocol allows the contents of a datagram to be encrypted, which ensures that only the intended recipient is able to see the data. ESP is implemented using three components: an *ESP Header* that's added to the front of a protected datagram, an *ESP Trailer* that follows the protected data, and an optional *ESP Authentication Data* field that provides authentication services similar to those provided by AH.

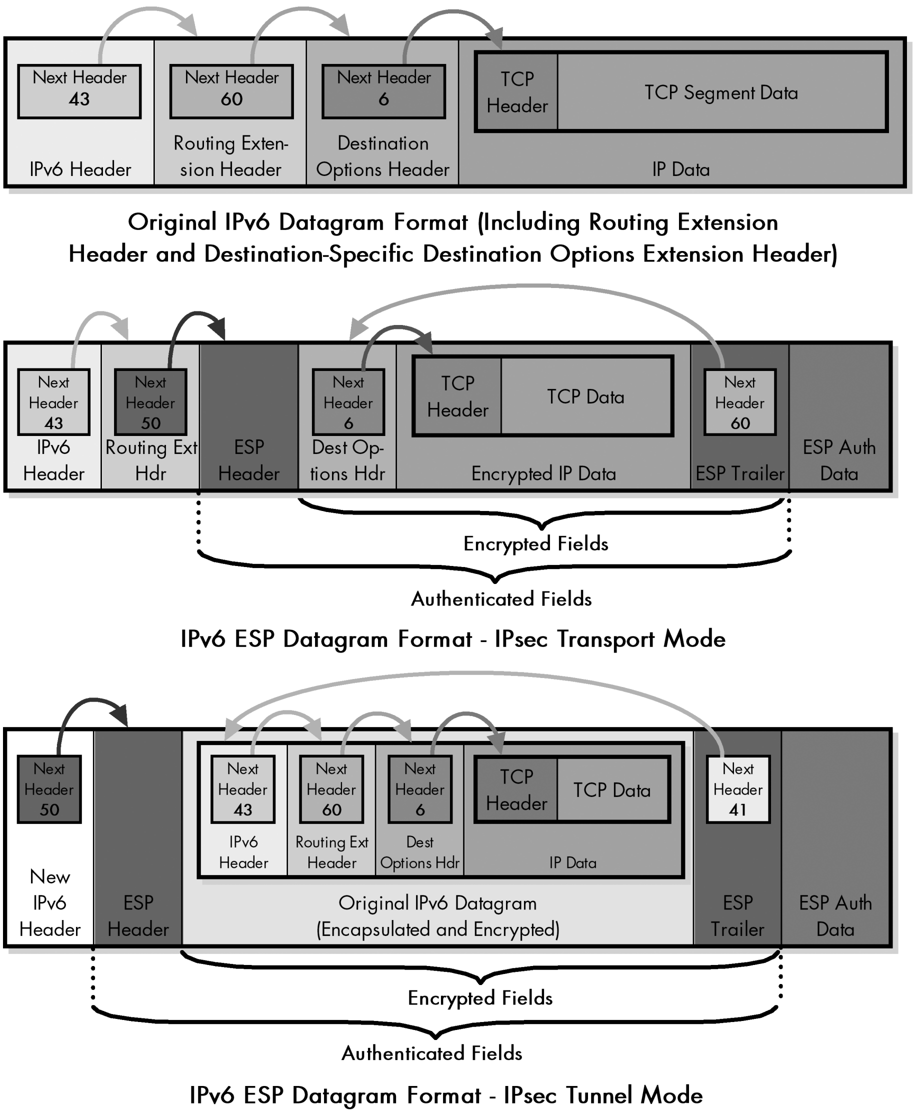


Figure 29-9: IPv6 datagram format with IPsec ESP *Here is the same example of an IPv6 datagram with two extension headers that you saw in Figure 29-6. When ESP is applied in transport mode, the ESP Header field is added to the existing datagram as in AH, and the ESP Trailer and ESP Authentication Data fields are placed at the end. In tunnel mode, the ESP Header and Trailer fields bracket the entire encapsulated IPv6 datagram. Note the encryption and authentication coverage in each case, and also how the Next Header field points back into the datagram since it appears in the ESP Trailer.*

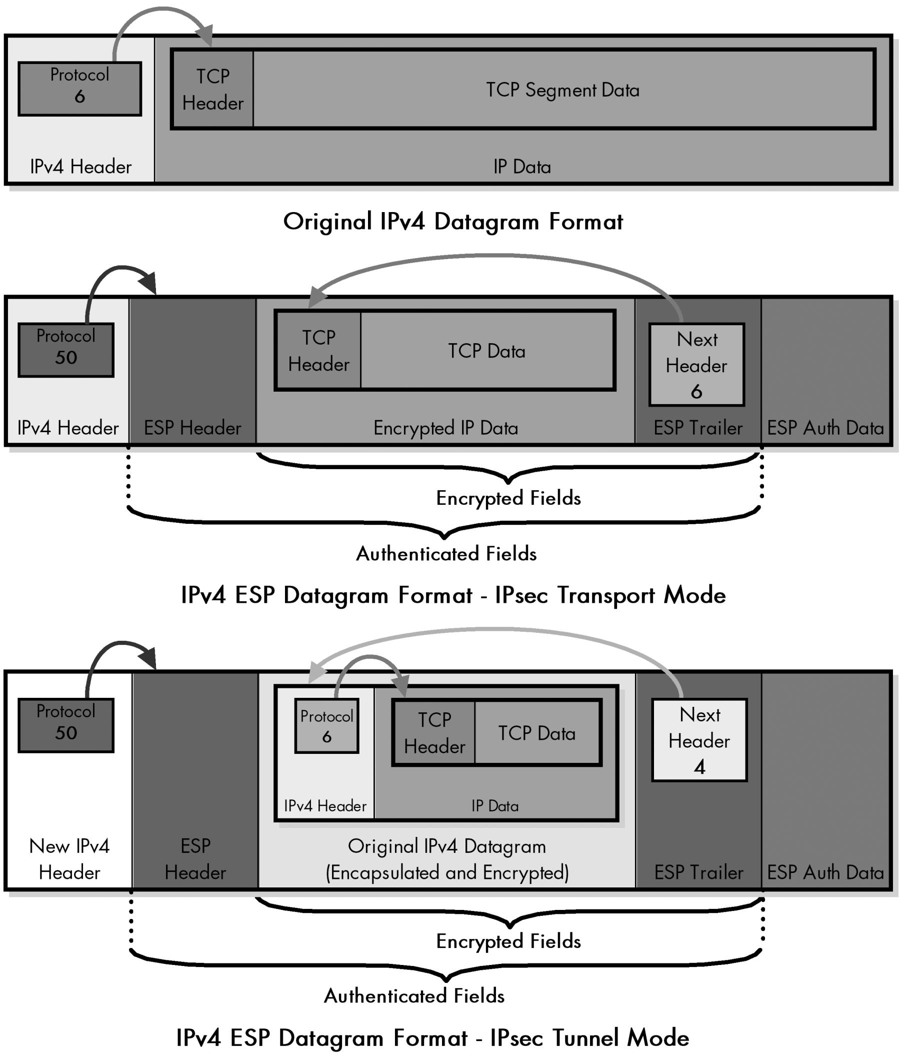


Figure 29-10: IPv4 datagram format with IPsec ESP *Here is the same sample IPv4 datagram that you saw in Figure 29-7. When ESP processes this datagram in transport mode, the ESP Header field is placed between the IPv4 header and data, with the ESP Trailer and ESP Authentication Data fields following. In tunnel mode, the entire original IPv4 datagram is surrounded by these ESP components, rather than just the IPv4 data. Again, as in Figure 29-9, note the encryption and authentication coverage, and how the Next Header field points back to specify the identity of the encrypted data or datagram.*

ESP Format

The format of the ESP sections and fields is described in Table 29-3 and illustrated in Figure 29-11. In both the figure and the table, I have shown the encryption and authentication coverage of the fields explicitly, to clarify how it all works.

Table 29-3: IPsec Encapsulating Security Payload (ESP) Format

Section	Field Name	Size (Bytes)	Description	Encryption Coverage	Authentication Coverage
ESP Header	SPI	4	A 32-bit value that is combined with the destination address and security protocol type to identify the SA that will be used for this datagram. (SAs are discussed earlier in this chapter.)		↕
	Sequence Number	4	A counter field initialized to zero when an SA is formed between two devices, and then incremented for each datagram that's sent using that SA. This is used to provide protection against replay attacks.		↕
Payload	Payload Data	Variable	The encrypted payload data, which consists of a higher-layer message or encapsulated IP datagram. It may also include support information such as an initialization vector that's required by certain encryption methods.	↕	↕
ESP Trailer	Padding	Variable (0 to 255)	Additional padding bytes are included as needed for encryption or for alignment.	↕	↕
	Pad Length	1	The number of bytes in the preceding Padding field.	↕	↕
	Next Header	1	Contains the protocol number of the next header in the datagram. Used to chain together headers.	↕	↕
ESP Authentication Data		Variable	Contains the ICV resulting from the application of the optional ESP authentication algorithm.		

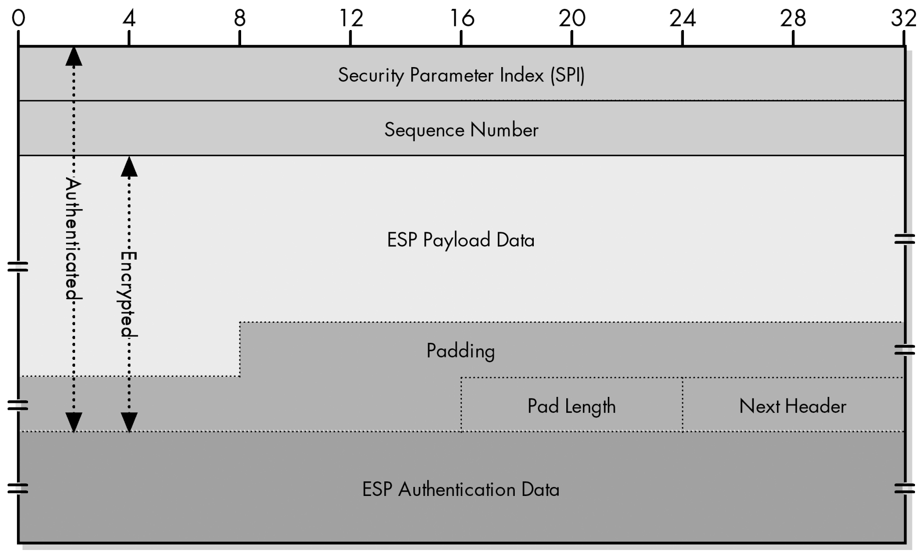


Figure 29-11: IPsec ESP format *Note that most of the fields and sections in this format are variable length. The exceptions are the SPI and Sequence Number fields, which are four bytes long, and the Pad Length and Next Header fields, which are one byte each.*

The Padding field is used when encryption algorithms require it. Padding is also used to make sure that the ESP Trailer field ends on a 32-bit boundary. That is, the size of the ESP Header field plus the Payload field, plus the ESP Trailer field must be a multiple of 32 bits. The ESP Authentication Data field must also be a multiple of 32 bits.

IPsec Internet Key Exchange (IKE)

IPsec, like many secure networking protocol sets, is based on the concept of a shared secret. Two devices that want to send information securely encode and decode it using a piece of information that only the devices know. Anyone who isn't in on the secret is able to intercept the information but is prevented either from reading it (if ESP is used to encrypt the payload) or from tampering with it undetected (if AH is used). Before either AH or ESP can be used, however, it is necessary for the two devices to exchange the secret that the security protocols themselves will use. The primary support protocol used for this purpose in IPsec is called *Internet Key Exchange (IKE)*.

IKE is defined in RFC 2409, and it is one of the more complicated of the IPsec protocols to comprehend. In fact, it is simply impossible to truly understand more than a real simplification of its operation without significant background in cryptography. I don't have a background in cryptography, and I must assume that you, my reader, do not either. So rather than fill this topic with baffling acronyms and unexplained concepts, I will just provide a brief outline of IKE and how it is used.

IKE Overview

The purpose of IKE is to allow devices to exchange information that's required for secure communication. As the title suggests, this includes cryptographic keys that are used for encoding authentication information and performing payload encryption. IKE works by allowing IPsec-capable devices to exchange SAs, which populate their SADs. These SADs are then used for the actual exchange of secured datagrams with the AH and ESP protocols.

IKE is considered a hybrid protocol because it combines (and supplements) the functions of three other protocols. The first of these is the *Internet Security Association and Key Management Protocol (ISAKMP)*. This protocol provides a framework for exchanging encryption keys and security association information. It operates by allowing security associations to be negotiated through a series of phases.

ISAKMP is a generic protocol that supports many different key exchange methods. In IKE, the ISAKMP framework is used as the basis for a specific key exchange method that combines features from two key exchange protocols:

OAKLEY Describes a specific mechanism for exchanging keys through the definition of various key exchange modes. Most of the IKE key exchange process is based on OAKLEY.

SKEME Describes a different key exchange mechanism than OAKLEY. IKE uses some features from SKEME, including its method of public key encryption and its fast rekeying feature.

IKE Operation

IKE doesn't strictly implement either OAKLEY or SKEME but takes bits of each to form its own method of using ISAKMP. Clear as mud, I know. Because IKE functions within the framework of ISAKMP, its operation is based on the ISAKMP phased-negotiation process. There are two phases, as follows:

ISAKMP Phase 1 The first phase is a setup stage where two devices agree on how to exchange further information securely. This negotiation between the two units creates an SA for ISAKMP itself: an *ISAKMP SA*. This security association is then used for securely exchanging more detailed information in Phase 2.

ISAKMP Phase 2 In this phase, the ISAKMP SA established in Phase 1 is used to create SAs for other security protocols. Normally, this is where the parameters for the "real" SAs for the AH and ESP protocols would be negotiated.

An obvious question is why IKE bothers with this two-phased approach. Why not just negotiate the SA for AH or ESP in the first place? Well, even though the extra phase adds overhead, multiple Phase 2 negotiations can be conducted after one Phase 1, which amortizes the extra cost of the two-phase approach. It is also possible to use a simpler exchange method for Phase 2 once the ISAKMP SA has been established in Phase 1.

The ISAKMP SA negotiated during Phase 1 includes the negotiation of the following attributes used for subsequent negotiations:

- An encryption algorithm, such as the *Data Encryption Standard (DES)*
- A hash algorithm (MD5 or SHA, as used by AH or ESP)
- An authentication method, such as authentication using previously shared keys
- A *Diffie-Hellman* group

NOTE *Diffie and Hellman were two pioneers in the industry who invented public-key cryptography. In this method, instead of encrypting and decrypting with the same key, data is encrypted using a public key that anyone can know, and decrypted using a private key that is kept secret. A Diffie-Hellman group defines the attributes of how to perform this type of cryptography. Four predefined groups derived from OAKLEY are specified in IKE, and provision is allowed for defining new groups as well.*

Note that even though SAs in general are unidirectional, the ISAKMP SA is established bidirectionally. Once Phase 1 is complete, either device can set up a subsequent SA for AH or ESP using the ISAKMP SA.