This chapter covers the following topics:

- **Types of IPsec VPNs**—This section lays out the two main types of IPsec VPNs.
- **Composition of IPsec**—This section introduces the protocols that are combined to create the IPsec protocol suite.
- **Introduction to IKE**—This section discusses some of the basic information about IKE and how it works in the context of the IPsec protocol.
- **IPsec Negotiation Using the IKE Protocol**—This section talks about how IKE is used to negotiate IPsec tunnels between two peers. Packet-by-packet details of all transactions are given for main mode, aggressive mode, and quick mode negotiations.
- **IKE Authentication Methods**—This section discusses the IKE authentication methods while focusing on the most important means of authentication available in IKE– Digital Certificates.
- **Encryption and Integrity Checking Mechanisms in IPsec**—This section covers the details of the various hashing and encryption mechanisms used in IPsec.
- **Packet Encapsulation in IPsec**—This section talks about how packets are encapsulated in IPsec using ESP or AH in tunnel or transport mode.
- **IKE Enhancements for Remote-Access Client IPsec**—This section talks about the mode config, x-auth, and NAT transparency features implemented to facilitate remote-access client IPsec.
- **IPsec Dead Peer Discovery Mechanism**—This section talks about the keepalive mechanism implemented in IPsec.
- **Case Studies**—This section discusses case studies involving real-life IPsec implementations. Configurations and debugs are included to aid in your understanding of each case study.

CHAPTER 13

IPsec

IPsec (IP Security) is a suite of protocols used to create virtual private networks (VPNs). According to the IETF, it is defined as follows:

A security protocol in the network layer will be developed to provide cryptographic security services that will flexibly support combinations of authentication, integrity, access control, and confidentiality.

In the most commonly used scenario, IPsec allows an encrypted tunnel to be created between two private networks. It also allows for authenticating the two ends of the tunnel. However, the IPsec protocol allows only for the encapsulation and encryption of IP data (unlike GRE, which can tunnel but not encrypt non-IP traffic), so to create a tunnel for non-IP-based traffic, IPsec must be used in conjunction with a protocol such as GRE, which allows for the tunneling of non-IP protocols.

IPsec is becoming the de facto standard for creating VPNs in the industry. It has been implemented by a large number of vendors, and interoperability between multivendor devices makes IPsec a very good option for building VPNs. Due to IPsec's growing acceptance and importance in the VPN world, we will discuss it in detail in this chapter.

We will start this chapter by discussing the types of setups IPsec is used to implement. We will then take that basic knowledge and go into a very detailed discussion of the three main protocols in IPsec—IKE, ESP, and AH. We will do a packet-by-packet analysis of how IKE and its component protocols are used to negotiate IPsec tunnels. We will discuss IKE in its various forms, as well as the different authentication methods. This will lead us into a more detailed and separate discussion of the authentication mechanisms available to IPsec. We will look at the advantages and disadvantages of preshared keys, digital signatures, and encrypted nonces. We will then go into a detailed discussion of how encryption and integrity checking (two of the cornerstones of IPsec) are built into IPsec. We will conclude our discussion of IPsec by talking about special enhancements made to IPsec to take care of remote-access VPN client environments.

Types of IPsec VPNs

There are a number of ways to categorize IPsec VPNs. However, it is instructive to look at IPsec in light of the two main VPN design issues it tries to resolve:

- The seamless connection of two private networks to form one combined virtual private network
- The extension of a private network to allow remote-access users (also known as *road warriors*) to become part of the trusted network

Based on these two designs, IPsec VPNs can be divided into two main categories:

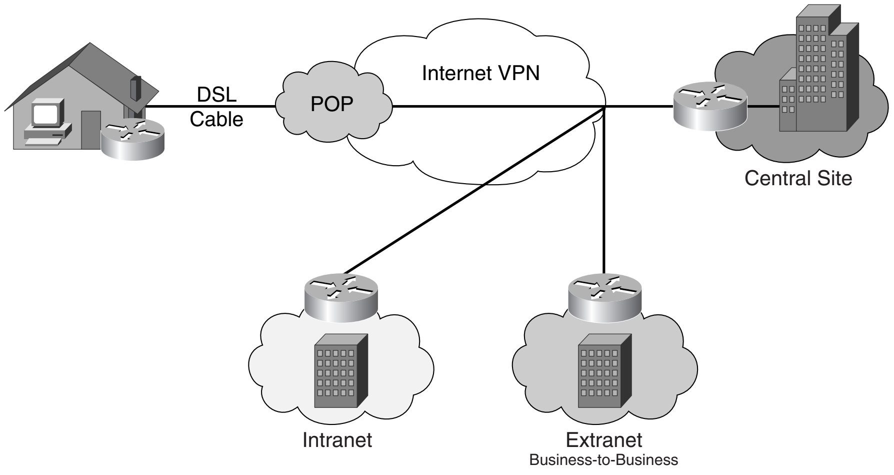
- LAN-to-LAN IPsec implementations (also known as *site-to-site VPNs*)
- Remote-access client IPsec implementations

LAN-to-LAN IPsec Implementations

LAN-to-LAN IPsec is a term often used to describe an IPsec tunnel created between two LANs. These are also called *site to site* IPsec VPNs. LAN-to-LAN VPNs are created when two private networks are merged across a public network such that the users on either of these networks can access resources on the other network as if they were on their own private network.

IPsec provides a means of negotiating and establishing an encrypted tunnel between two sites. Generally, tunneling is important, because the two private networks often use RFC 1918 addressing, which needs to be tunneled to traverse the public network. IPsec lets you define what traffic is to be encrypted and how it is to be encrypted. We will look at the details of how IPsec achieves this in the next few sections. Some examples of LAN-to-LAN and site-to-site IPsec setups are shown in Figure 13-1.

Figure 13-1 *Types of IPsec VPNs: LAN-to-LAN and Site-to-Site IPsec*

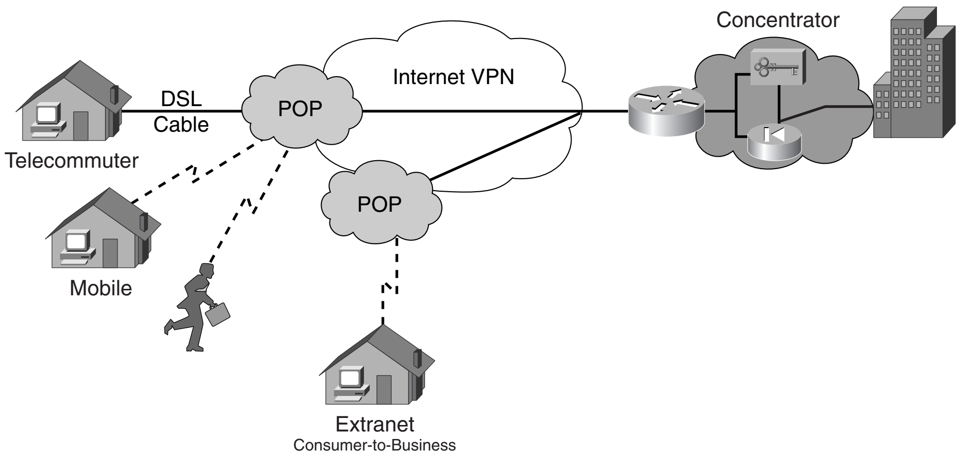


Remote-Access Client IPsec Implementations

Remote-access client IPsec VPNs are created when a remote user connects to an IPsec router or access server using an IPsec client installed on the remote user's machine. Generally, these remote-access machines connect to the public network or the Internet using dialup or some other similar means of connectivity. As soon as basic connectivity to the Internet is established, the IPsec client can set up an encrypted tunnel across the pubic network or the Internet to an IPsec termination device located at the edge of the private network to which the client wants to connect and be a part of. These IPsec termination devices are also known as IPsec remote-access *concentrators*.

Remote-access implementation of IPsec comes with its own unique set of challenges. In the site-to-site scenario, the number of IPsec peers, meaning the devices that terminate IPsec tunnels, may be limited. However, in the case of remote-access IPsec VPNs, the number of peers can be substantial, even running into the tens of thousands for larger implementations. These scenarios require special scalable mechanisms for authentication key management because maintaining all the keys for all the users might become an impossible task. Later in this chapter we will look at some of the important features that have been added to IPsec to handle remote-access clients setups. Some examples of remote-access IPsec setups are shown in Figure 13-2.

Figure 13-2 *Types of IPsec VPNs: Remote-Access IPsec*



Composition of IPsec

IPsec combines three main protocols to form a cohesive security framework:

- Internet Key Exchange (IKE) protocol

- Encapsulating Security Payload (ESP) protocol
- Authentication Header (AH) protocol

Of these three protocols, IKE and ESP are the ones that are mostly deployed together. Although AH is also an important component of the IPsec protocol suite, not that many deployments of IPsec have this protocol turned on for use. In general, much of AH's functionality is embedded in ESP. Therefore, in our discussions in the rest of this chapter, we will focus our attention on ESP, and much of the discussion will assume that we are talking about ESP unless otherwise stated. For example, while discussing quick-mode exchanges in the following sections, we will assume that the goal is to do ESP.

IKE is used to negotiate the parameters between two IPsec peers for setting up a tunnel between them. We will look in detail at the workings of IKE in the next section. ESP provides the encapsulation mechanism for IPsec traffic. We will go into a more detailed discussion of ESP later in this chapter as well.

Table 13-1 describes the three main components of the IPsec protocol suite.

Table 13-1 *IPsec Combines Three Main Protocols to Form a Cohesive Security Framework*

Protocol	Description
IKE	Provides a framework for negotiating security parameters and establishing authenticated keys.
ESP	Provides a framework for encrypting, authenticating, and securing data.
AH	Provides a framework for authenticating and securing data.

Introduction to IKE

IKE or Internet key exchange is the protocol responsible for negotiating the IPsec tunnel characteristics between two IPsec peers. IKE's responsibilities in the IPsec protocol include

- Negotiating protocol parameters
- Exchanging public keys
- Authenticating both sides
- Managing keys after the exchange

IKE solves the problems of manual and unscalable IPsec implementation by automating the entire key-exchange process. This is one of IPsec's critical requirements.

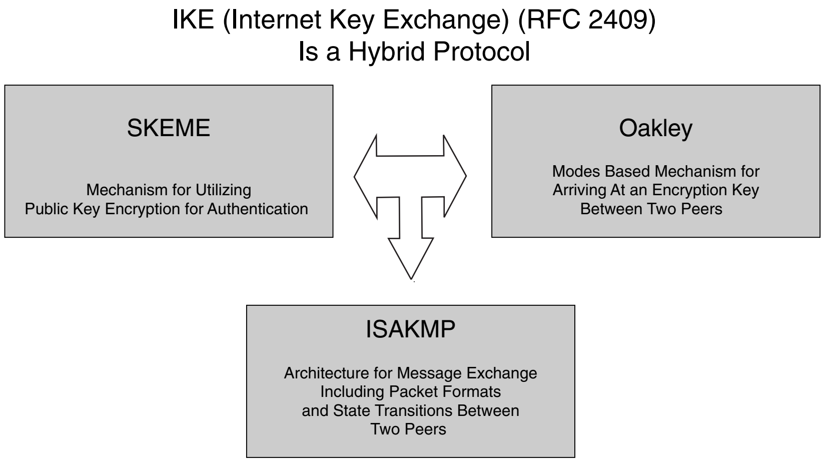
IKE, like IPsec, is also a combination of three different protocols:

- **SKEME**—Provides a mechanism for using public key encryption for authentication purposes.

- **Oakley**—Provides a mode-based mechanism for arriving at an encryption key between two IPsec peers.
- **ISAKMP**—Defines the architecture for message exchange, including packet formats and state transitions between two IPsec peers.

IKE combines these three protocols in one framework to provide IPsec the facilities just discussed. Figure 13-3 shows these three main components of IKE.

Figure 13-3 *Composition of the IKE Protocol*


IKE (Internet Key Exchange) (RFC 2409)
Is a Hybrid Protocol
SKEME
Mechanism for Utilizing Public Key Encryption for Authentication
Oakley
Modes Based Mechanism for Arriving At an Encryption Key Between Two Peers
ISAKMP
Architecture for Message Exchange Including Packet Formats and State Transitions Between Two Peers

IKE is defined as a standard in RFC 2409. Although IKE does provide a great deal of functionality to the IPsec protocol, some shortcomings in the protocol structure make it difficult to implement in code and scale to new challenges. Work is under way to improve the workings of the IKE protocol and its restandardization in an improved format. This is called the "son-of-IKE" or IKE v2 initiative. See the IETF website for more information on this initiative.

IKE is a two phase protocol. An IPsec tunnel is set up between two peers through the following sequence of events:

Step 1 Interesting traffic is received or generated by one of the IPsec peers on an interface that has been configured to initiate an IPsec session for this traffic.

Step 2 Main mode or aggressive mode negotiation using IKE results in the creation of an IKE Security Association (SA) between the two IPsec peers.

Step 3 Quick mode negotiation using IKE results in the creation of two IPsec SAs between the two IPsec peers.

Step 4 Data starts passing over an encrypted tunnel using the ESP or AH encapsulation techniques (or both).

Based on Steps 2 and 3, you can see that IKE is a two-phase protocol. Phase 1 is accomplished using main mode or aggressive mode exchanges between the peers, and Phase 2 is accomplished using quick mode exchanges.

This list shows how IKE behaves in a two-phase mechanism:

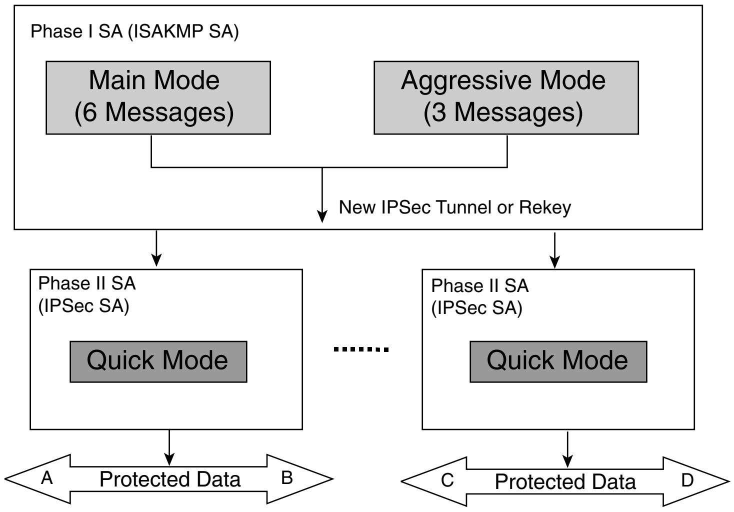
Step 1 In a Phase 1 exchange, peers negotiate a secure, authenticated channel with which to communicate. Main mode or aggressive mode accomplishes a Phase I exchange.

Step 2 In a Phase 2 exchange, security associations are negotiated on behalf of IPsec services. Quick mode accomplishes a Phase II exchange.

Main-mode exchange takes place through the exchange of a total of six messages between the two IPsec peers. If aggressive mode is used, only three messages complete Phase 1 of the exchange. Quick-mode exchange is done using an additional three messages exchanged between the two IPsec peers.

Figure 13-4 shows main mode, aggressive mode, and quick mode in IKE.

Figure 13-4 *IKE Main Mode, Aggressive Mode, and Quick Mode*



Goals of Main Mode (or Aggressive Mode)

The primary goals of main mode (or aggressive mode) are

- Agreeing on a set of parameters that are to be used to authenticate the two peers and to encrypt a portion of the main mode and all of the quick mode exchange. None of the aggressive mode is encrypted if it is used as the method for negotiation.

- Authenticate the two peers to each other.
- Generate keys that can be used to generate keying material for actual encryption of data as soon as the negotiations have been completed.

All the information negotiated in main mode or aggressive mode, including the keys that are later used to generate the keys to encrypt the data, is stored as what is known as IKE or ISAKMP security association (SA). Any two IPsec peers have only one ISAKMP security association between them.

Goals of Quick Mode

The primary role of quick mode is allow the two peers to agree on a set of attributes for creating the IPsec security associations that will be used to encrypt (in the case of ESP) the data between the two hosts. Also, in the case of PFS (Perfect Forward Secrecy, which you'll read about later), quick mode is responsible for redoing the Diffie-Hellman (DH) exchange so that new keying material is available before the IPsec data encryption keys are generated.

The next section analyzes how IKE is used to negotiate the IPsec parameters between IPsec peers using main (or aggressive) and quick mode.

IPsec Negotiation Using the IKE Protocol

IKE negotiates IPsec tunnels between two IPsec peers. This negotiation can be done using a combination of main-mode and quick-mode exchanges or a combination of aggressive-mode and quick-mode exchanges. This section looks at the various packets and message types that are used in these exchanges to do the negotiation. We will look at three types of negotiations that IKE carries out:

- Main mode using preshared key authentication followed by quick-mode negotiation
- Main mode using digital signature authentication followed by quick-mode negotiation
- Aggressive mode using preshared key authentication followed by quick-mode negotiation

In addition to these types, the following types of negotiations can also take place:

- Main mode using encrypted nonces authentication followed by quick-mode negotiation
- Aggressive mode using digital signature authentication followed by quick-mode negotiation

However, we will not go into the details of the last two types because the first three are by far the most common permutations used. We will focus on the most commonly used types of negotiations to aid your understanding.

Main Mode Using Preshared Key Authentication Followed by Quick Mode Negotiation

As stated earlier, this negotiation takes place with the exchange of a total of six messages between the two IPsec peers in main mode followed by three messages exchanged during quick mode.

In the following sections we will walk through the preparation for, as well as the actual exchange of, each message involved in this negotiation.

IKE Phase 1 (Main Mode): Preparation for Sending Messages 1 and 2

The first two messages in the IKE main mode negotiation are used to negotiate the various values, hash mechanisms, and encryption mechanisms to use for the later half of the IKE negotiations. In addition, information is exchanged that will be used to generate keying material for the two peers.

However, before the first two messages can be exchanged, the negotiation's initiator and responder must calculate what are known as *cookies*. These cookies are used as unique identifiers of a negotiation exchange.

The two peers generate a pseudo-random number that is used for anticlogging purposes. These cookies are based on a unique identifier for each peer (src and destination IP addresses) and therefore protect against replay attacks. The ISAKMP RFC states that the method of creating the cookie is implementation-dependent but suggests performing a hash of the IP source and destination address, the UDP source and destination ports, a locally generated random value, time, and date. The cookie becomes a unique identifier for the rest of the messages that are exchanged in IKE negotiation.

The following list shows how each peer generates its cookie:

- **Generation of the initiator cookie**—An 8-byte pseudo-random number used for anti-clogging

  CKY-I = md5{(src_ip, dest_ip), random number, time, and date}

- **Generation of the responder cookie**—An 8-byte pseudo-random number used for anti-clogging

  CKY-R = md5{(src_ip, dest_ip), random number, time, and date}

IKE uses payloads and packet formats defined in the ISAKMP protocol to do the actual exchange of information. The packets exchanged consist of the ISAKMP header and a series of *payloads* that are used to carry the information needed to carry out the negotiation.

IKE Phase 1 (Main Mode): Sending Message 1

Let's look at the first message sent across by the IPsec initiator to the respondent and look at the various fields in this message.

Figure 13-5 shows the first IKE message being sent.

Figure 13-5 *Sending IKE Main Mode Message 1*

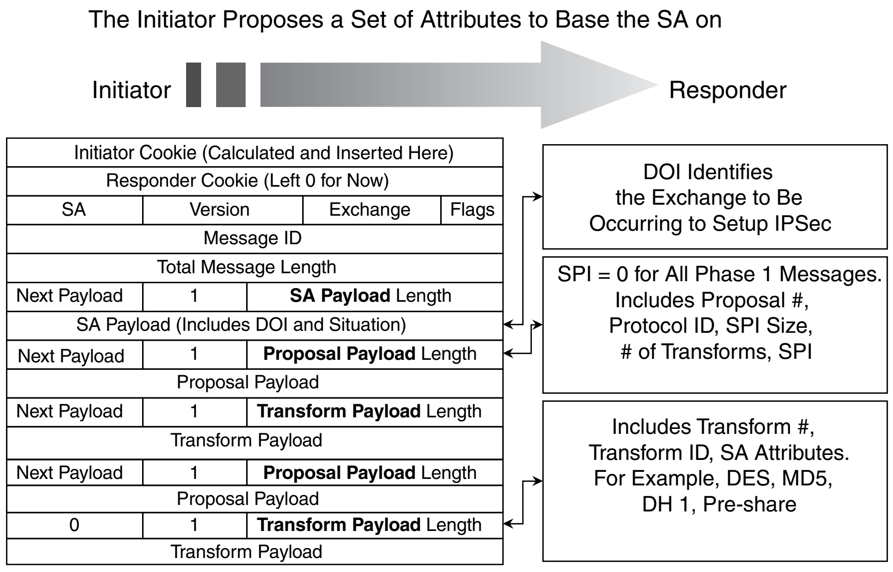

The Initiator Proposes a Set of Attributes to Base the SA on
Initiator
Responder
Initiator Cookie (Calculated and Inserted Here)
Responder Cookie (Left 0 for Now)
SA
Version
Exchange
Flags
Message ID
Total Message Length
Next Payload
1
SA Payload Length
SA Payload (Includes DOI and Situation)
Next Payload
1
Proposal Payload Length
Proposal Payload
Next Payload
1
Transform Payload Length
Transform Payload
Next Payload
1
Proposal Payload Length
Proposal Payload
0
1
Transform Payload Length
Transform Payload
DOI Identifies the Exchange to Be Occurring to Setup IPSec
SPI = 0 for All Phase 1 Messages. Includes Proposal #, Protocol ID, SPI Size, # of Transforms, SPI
Includes Transform #, Transform ID, SA Attributes. For Example, DES, MD5, DH 1, Pre-share

Figure 13-5 shows the ISAKMP header and five payloads. There is one SA payload and two pairs of proposal and transform payloads.

NOTE All payloads have a field that defines the length of that particular payload. In all the figures in this chapter depicting packet formats, the name of the payload in this field is highlighted to clearly distinguish between the various payloads.

Let's look at each of these in turn:

- **ISAKMP header**—The ISAKMP header contains the initiator's cookie, and the responder's cookie is left at 0 for the responder to calculate and fill in. The next payload field contains a value signifying that the next payload is the SA payload.

- **SA payload**—The SA payload contains two important pieces of information. Because ISAKMP is a generic protocol with packet and message formats that can be used to negotiate any number of protocols, it is important to specify that this particular ISAKMP exchange is taking place for IPsec negotiation. Therefore, the SA payload contains a Domain of Interpretation (DOI), which says that this message exchange is for IPsec. The other important piece of information is the situation. The situation definition is a 32-bit bitmask that represents the environment under which the IPsec SA proposal and negotiation are carried out. The situation provides information that the responder can use to make a policy determination about how to process the incoming security association request.
- **Proposal payload**—The proposal payload contains a proposal number, protocol ID, SPI (security parameter index) size, number of transforms, and SPI. The proposal number is used to differentiate the various proposals being sent in the same ISAKMP packet. The protocol ID is set to ISAKMP, and the SPI is set to 0. The SPI is not used to identify an IKE phase 1 exchange. Rather, the pair of cookies is used to identify the IKE exchange. We will discuss SPI again during phase 2 of IKE where its use is of more significance. The number of transforms indicates the number of transforms that are associated with this particular proposal payload (only one in this case). Note that the packet contains two pairs of proposal and transform payloads.
- **Transform payload**—Includes transform number, transform ID, and IKE SA attributes. The transform number and the ID are used to uniquely identify the transform from among the rest of the transforms offered in this ISAKMP packet (this sample packet has only one other transform, though). The IKE SA attributes include the attributes that the initiator wants the responder to agree on. These include the type of hash to use in the calculation of various keying materials later in the negotiation, the encryption mechanism to use to encrypt IKE negotiation messages as soon as a shared secret has been established, the Diffie-Hellman exchange mechanism to use, the method of authentication to use, and the timeout values of the IKE SAs negotiated. In this example, assume that the method of authentication being offered is preshared, meaning that a preshared key has been defined on the initiator and responder out of band.

For the sake of this example, assume that the initiator offered the responder the following transform attributes in the first payload. Don't worry about what it offered in the second payload, because we will assume that the responder agrees to what is offered in the first pair of transform and proposal payloads and does not have to worry about the second pair:

- **Encryption mechanism**—DES
- **Hashing mechanism**—MD5-HMAC
- **Diffie-Hellman group**—1
- **Authentication mechanism**—Preshared

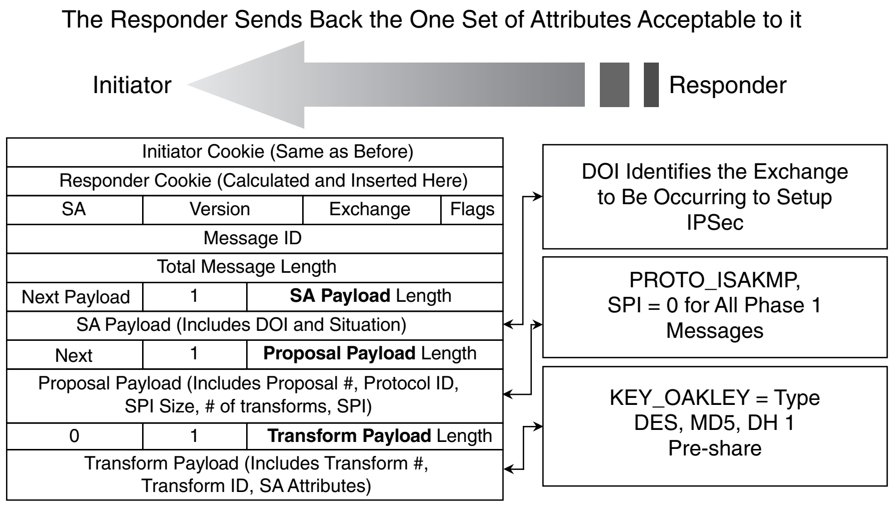
All ISAKMP messages are carried in a UDP packet with destination port 500.

IKE Phase 1 (Main Mode): Sending Message 2

Message 2 is the response from the responder to the packet that was sent by the initiator. Most of the fields are the same as in the packet sent by the initiator, so we will discuss the differences only. Note that only one proposal and transform payload is included in the response because the responder agrees to only one proposal/transform pair and returns that as the agreed-on pair.

Figure 13-6 shows the second IKE message being sent.

Figure 13-6 *Sending IKE Main Mode Message 2*


The Responder Sends Back the One Set of Attributes Acceptable to it
Initiator
Responder
Initiator Cookie (Same as Before)
Responder Cookie (Calculated and Inserted Here)
SA
Version
Exchange
Flags
Message ID
Total Message Length
Next Payload
1
SA Payload Length
SA Payload (Includes DOI and Situation)
Next
1
Proposal Payload Length
Proposal Payload (Includes Proposal #, Protocol ID, SPI Size, # of transforms, SPI)
0
1
Transform Payload Length
Transform Payload (Includes Transform #, Transform ID, SA Attributes)
DOI Identifies the Exchange to Be Occurring to Setup IPSec
PROTO_ISAKMP, SPI = 0 for All Phase 1 Messages
KEY_OAKLEY = Type DES, MD5, DH 1 Pre-share

Here are the payloads included in this message:

- **ISAKMP header**—You can see that the ISAKMP header now has both cookie fields set to their respective values.
- **SA payload**—The SA payload contains pretty much the same material that was sent by the initiator.
- **Proposal payload**—The proposal payload contains the information for the proposal that the responder has decided to accept.
- **Transform payload**—The transform payload contains the elements of the transform that the responder has decided to accept.

For the sake of our example, assume that the responder accepted the first proposal described in the preceding section:

- **Encryption mechanism**—DES
- **Hashing mechanism**—MD5-HMAC

- **Diffie-Hellman group**—1
- **Authentication mechanism**—Preshared

NOTE IKE SA timeouts are also negotiated using the transform payloads. The acceptable transform payload must not only have all the other IKE attributes contained in it that are agreeable to the responder, but it also must have an IKE SA timeout value that is equal to or less than the responder that it is set up to accept. If no such transform is included, the negotiation fails.

IKE Phase 1 (Main Mode): Preparation for Sending Messages 3 and 4

The next step that both the initiator and the responder must carry out is to generate material that will be used for the production of a Diffie-Hellman shared secret between the two. Because DH is a critical component of the negotiation taking place between the two peers, we will here discuss how it works.

Diffie-Hellman Algorithm

The Diffie-Hellman algorithm is used in IKE negotiations to allow the two peers to agree on a shared secret, to generate keying material for subsequent use, without knowing any secrets beforehand. (Note that although the preshared secret in this example is already defined on the two peers, the DH secret is used in conjunction with that preshared secret to authenticate the two peers to each other.)

The DH algorithm relies on the following property:

There exists a DH public value = X_a

such that

$X_a = g^a \bmod p$

where

g is the generator

p is a large prime number

a is a private secret known only to the initiator

And there exists another DH public value = X_b

such that

$X_b = g^b \bmod p$

where

g is the generator

p is a large prime number

b is a private secret known only to the responder

Then the initiator and the responder can generate a shared secret known only to the two of them by simply exchanging the values X_a and X_b with each other. This is true because

initiator secret = $(X_b)^a \bmod p = (X_a)^b \bmod p$ = responder secret

This value is the shared secret between the two parties and is also equal to g^{ab}.

Coming back to IKE, in order to calculate the DH secret between the two peers, the two peers calculate the DH public values and send them to each other. In addition, a value known as a *nonce* is also generated and exchanged. A nonce is a very large random number generated using certain mathematical techniques. It is used in later calculations of the keying material. The following lists describe the preparation for sending message 3 of the IKE.

First, the two peers independently generate a DH public value:

- Generation of the DH public value by the initiator

  DH public value = X_a

  $X_a = g^a \bmod p$

  where

  g is the generator

  p is a large prime number

  a is a private secret known only to the initiator

- Generation of the DH public value by the responder

  DH public value = X_b

  $X_b = g^b \bmod p$

  where

  g is the generator

  p is a large prime number

  b is a private secret known only to the responder

As soon as the DH public values have been calculated, the two peers also independently calculate the nonces:

- Generation of a nonce by the initiator

  initiator nonce = N_i

- Generation of a nonce by the responder

  responder nonce = N_r

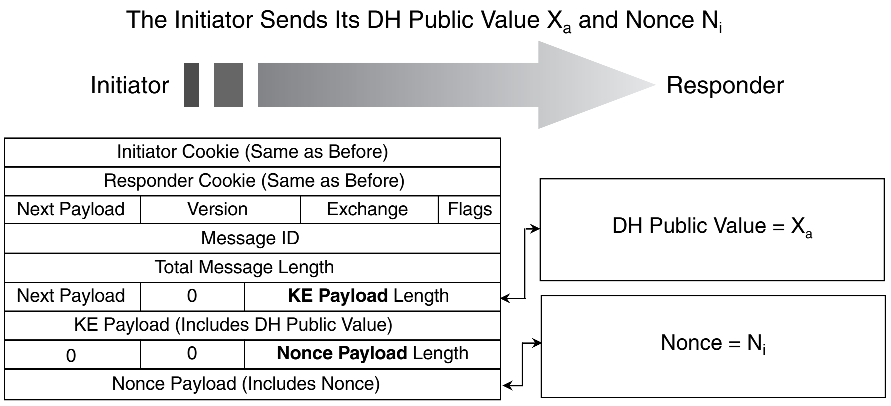
IKE Phase 1 (Main Mode): Sending Message 3

The third message is sent from the initiator to the responder. The ISAKMP header is similar to the one contained in the earlier messages, complete with cookies. However, two new payloads are introduced in this message—the key exchange payload and the nonce payload.

- **Key exchange (KE) payload**—The KE payload is used to carry the DH public value $X_{a \text{ generated for doing DH.}}$
- **Nonce payload**—The nonce payload contains the nonce generated in accordance with the preceding discussion.

Figure 13-7 shows message 3 of IKE being sent.

Figure 13-7 *Sending Message 3 of IKE*

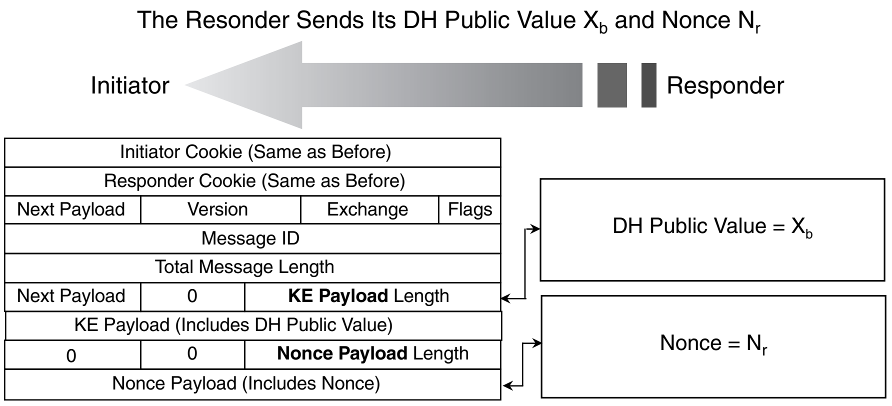

The Initiator Sends Its DH Public Value X_a and Nonce N_i
Initiator
Responder
Initiator Cookie (Same as Before)
Responder Cookie (Same as Before)
Next Payload
Version
Exchange
Flags
Message ID
Total Message Length
Next Payload
0
KE Payload Length
KE Payload (Includes DH Public Value)
0
0
Nonce Payload Length
Nonce Payload (Includes Nonce)
DH Public Value = X_a
Nonce = N_i

IKE Phase 1 (Main Mode): Sending Message 4

The fourth message is sent from the responder to the initiator. It is very similar to message 3 in that it contains the corresponding DH public value and the nonce for the responder.

Figure 13-8 shows message 4 of IKE being sent.

Figure 13-8 *Sending Message 4 of IKE*


The Resonder Sends Its DH Public Value X_b and Nonce N_r
Initiator
Responder
Initiator Cookie (Same as Before)
Responder Cookie (Same as Before)
Next Payload
Version
Exchange
Flags
Message ID
Total Message Length
Next Payload
0
KE Payload Length
KE Payload (Includes DH Public Value)
0
0
Nonce Payload Length
Nonce Payload (Includes Nonce)
DH Public Value = X_b
Nonce = N_r

IKE Phase 1 (Main Mode): Preparation for Sending Messages 5 and 6

Before messages 5 and 6 can be sent, the two peers must calculate the DH shared secret. In addition, based on the exchanged nonce, the DH secret calculated, and the preshared keys stored on the two peers, three sets of keys must be generated that will be used to authenticate the two peers to each other as well as to encrypt subsequent IKE exchanges.

Three keys are generated by each peer. These keys come out to be the same on both ends due to the nature of the DH exchange and the rest of the elements used to generate the keys. These keys are called session keys (SKEYs). The three keys being generated are as follows:

- **SKEYID_d**—This key is used to calculate subsequent IPsec keying material.
- **SKEYID_a**—This key is used to provide data integrity and authentication to subsequent IKE messages.
- **SKEYID_e**—This key is used to encrypt subsequent IKE messages.

Now that both ends have the keying material generated, the rest of the IKE exchange can occur confidentially using encryption done using **SKEYID_e**.

Note that for the keys to be generated correctly, each peer must find the corresponding preshared key for its peer. A number of preshared keys might be configured on both the peers for each of the many peers they communicate with. However, the ID payload identifying the peer's IP address or the host name does not arrive until the next message is exchanged. Therefore, each peer must find the preshared key for its peer by using the source IP address from which it is receiving the ISAKMP packets. The reason for sending the ID payload later is to hide it by using encryption afforded by the keys that have been generated in this step. We will look at the problems this method can pose in the discussion of aggressive mode.

For the two peers to generate their SKEYIDs, they must first calculate their shared DH secret:

- Calculation of the shared DH secret by the initiator

  shared secret = $(X_b)^a$ mod p

- Calculation of the shared DH secret by the responder

  shared secret = $(X_a)^b$ mod p

These two values end up being the same, g^{ab}, as discussed earlier.

After the DH secret has been calculated, the SKEYIDs can be generated. Figure 13-9 shows the session keys being generated by the initiator, and Figure 13-10 shows the session keys being generated by the responder.

Figure 13-9 *Session Keys Being Generated by the Initiator*

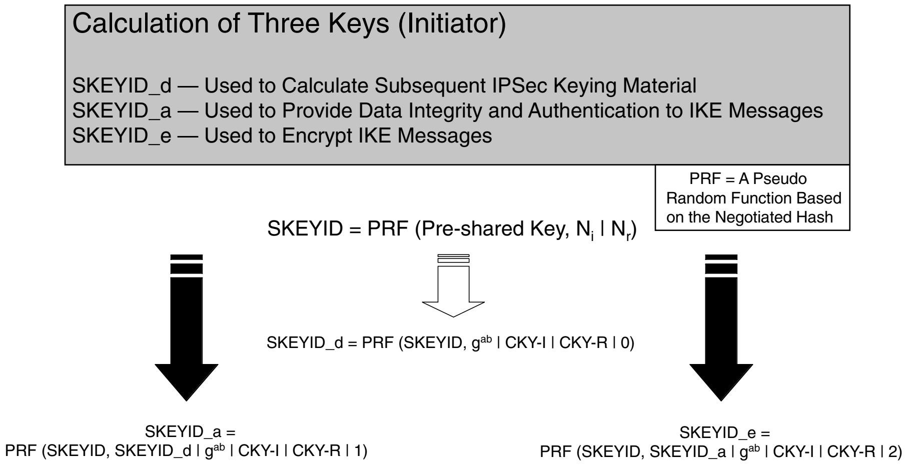
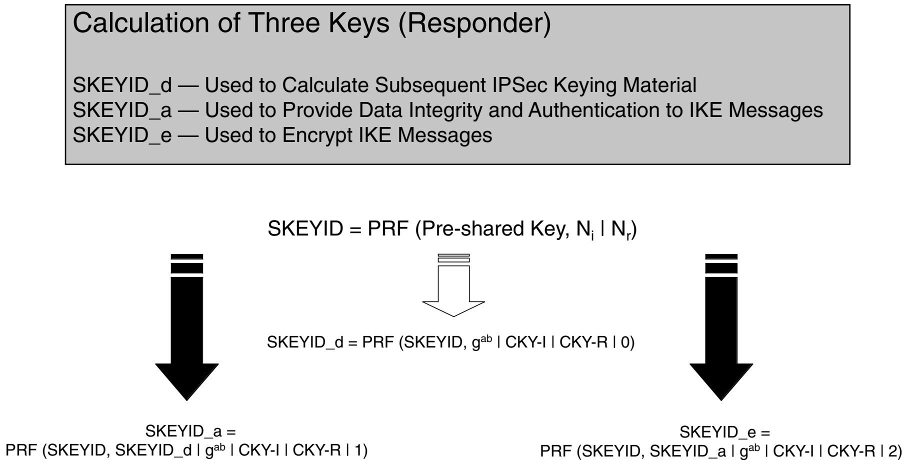

Calculation of Three Keys (Initiator)
SKEYID_d — Used to Calculate Subsequent IPSec Keying Material
SKEYID_a — Used to Provide Data Integrity and Authentication to IKE Messages
SKEYID_e — Used to Encrypt IKE Messages
PRF = A Pseudo Random Function Based on the Negotiated Hash
SKEYID = PRF (Pre-shared Key, N_i | N_r)
SKEYID_d = PRF (SKEYID, g^{ab} | CKY-I | CKY-R | 0)
SKEYID_a = PRF (SKEYID, SKEYID_d | g^{ab} | CKY-I | CKY-R | 1)
SKEYID_e = PRF (SKEYID, SKEYID_a | g^{ab} | CKY-I | CKY-R | 2)

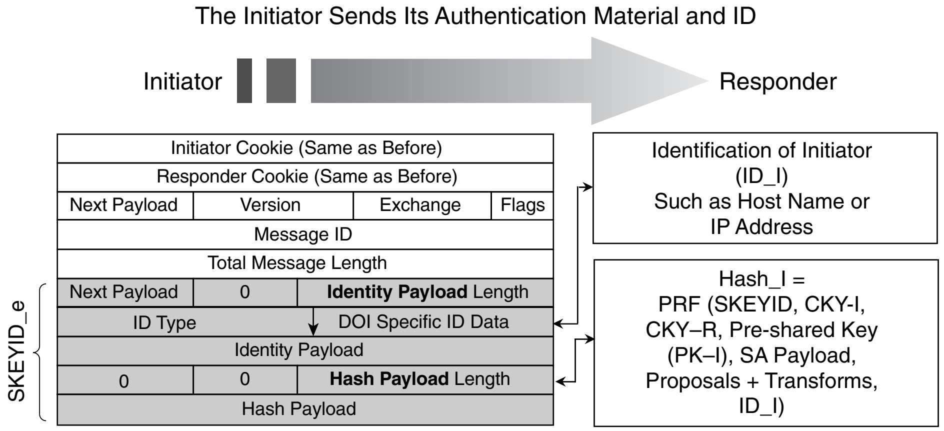
Figure 13-10 *Session Keys Being Generated by the Responder*


Calculation of Three Keys (Responder)
SKEYID_d — Used to Calculate Subsequent IPSec Keying Material
SKEYID_a — Used to Provide Data Integrity and Authentication to IKE Messages
SKEYID_e — Used to Encrypt IKE Messages
SKEYID = PRF (Pre-shared Key, N_i | N_r)
SKEYID_d = PRF (SKEYID, g^ab | CKY-I | CKY-R | 0)
SKEYID_a = PRF (SKEYID, SKEYID_d | g^ab | CKY-I | CKY-R | 1)
SKEYID_e = PRF (SKEYID, SKEYID_a | g^ab | CKY-I | CKY-R | 2)

IKE Phase 1 (Main Mode): Sending Message 5

Message 5 is sent with the usual ISAKMP header, but it contains two new payloads—the identity payload and the hash payload. Figure 13-11 shows message 5 of IKE being sent.

Figure 13-11 *Sending Message 5 of IKE*


The Initiator Sends Its Authentication Material and ID
Initiator
Responder
Initiator Cookie (Same as Before)
Responder Cookie (Same as Before)
Next Payload
Version
Exchange
Flags
Message ID
Total Message Length
Next Payload
0
Identity Payload Length
ID Type
DOI Specific ID Data
Identity Payload
0
0
Hash Payload Length
Hash Payload
SKEYID_e
Identification of Initiator (ID_I) Such as Host Name or IP Address
Hash_I = PRF (SKEYID, CKY-I, CKY–R, Pre-shared Key (PK–I), SA Payload, Proposals + Transforms, ID_I)

Here are the various payloads included in the message:

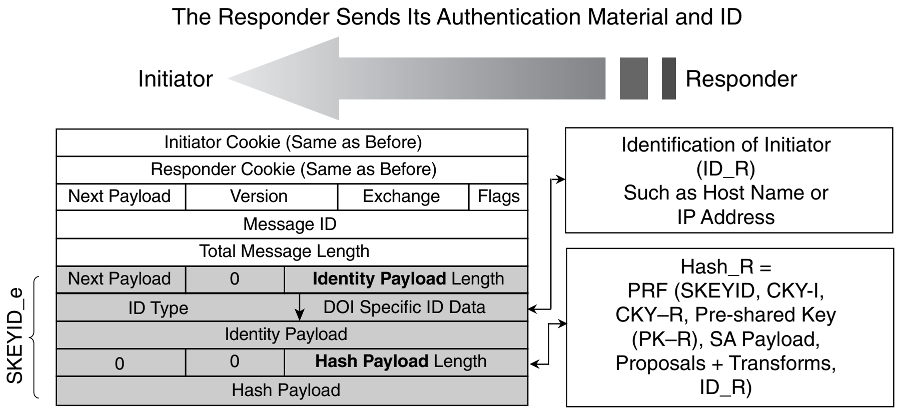
- **Identity payload**—This payload contains information about the identity of the initiator. This is in the form of the initiator's IP address or host name.
- **Hash payload**—The hash payload is calculated as shown in Figure 13-11. The hash is used for authentication purposes. The responder calculates the same hash on its end. If the two hashes come out to be the same, authentication is said to have taken place.

The two payloads are encrypted using SKEYID_E.

IKE Phase 1 (Main Mode): Sending Message 6

Message 6 is very similar to message 5 in that the responder sends the corresponding authentication material and its ID. Figure 13-12 shows IKE message 6 being sent.

Figure 13-12 *Sending Message 6 of IKE*


The Responder Sends Its Authentication Material and ID
Initiator
Responder
Initiator Cookie (Same as Before)
Responder Cookie (Same as Before)
Next Payload
Version
Exchange
Flags
Message ID
Total Message Length
Next Payload
0
Identity Payload Length
ID Type
DOI Specific ID Data
Identity Payload
0
0
Hash Payload Length
Hash Payload
SKEYID_e
Identification of Initiator (ID_R) Such as Host Name or IP Address
Hash_R = PRF (SKEYID, CKY-I, CKY–R, Pre-shared Key (PK–R), SA Payload, Proposals + Transforms, ID_R)

The contents of the packets are encrypted using SKEYID_E.

Completion of IKE Phase 1 (Main Mode) Using Preshared Keys

The main mode of IKE is completed using the material received in the last two messages of IKE exchange. The two sides authenticate each other by recalculating the hash and comparing it to the one they received in the hash payload. If the two values are the same, the two sides are considered to have successfully authenticated.

The following list shows how the two peers use the authentication hashes received from each other to verify each other's authenticity:

- Initiator authenticates the responder

  Step 1 Decrypt the message using SKEYID_E.

  Step 2 Find the configured PK-R (pre-shared key or responder) using ID_R.

  Step 3 Calculate Hash_R on its own.

  Step 4 If received Hash_R = self-generated Hash_R, authentication is successful!

- Responder authenticates the initiator

  Step 1 Decrypt the message using SKEYID_E.

  Step 2 Find the configured PK-I (pre-shared key or initiator) using ID_I.

  Step 3 Calculate Hash_I on its own.

  Step 4 If received Hash_I = self-generated Hash_I, authentication is successful!

At this point, the IKE SA is established, and main mode using preshared key authentication is complete. The next step, quick mode, is discussed in the next section.

IKE Phase 2 (Quick Mode): Preparation for Sending Messages 1 and 2

Just as the main mode was used to agree upon parameters for the IKE SA, quick mode is used to negotiate some of the parameters of IPsec security association. In our example, we will assume that the initiator has decided to use a property known as Perfect Forward Secrecy (PFS).

PFS

PFS is a property that the initiator of an IKE negotiation can "suggest" to the responder. This suggestion is made by sending a key exchange attribute payload during the first message of quick mode. If the responder agrees to do PFS, it continues the quick mode exchange. Otherwise, it returns "Attributes Not Supported," and the initiator can continue without PFS if it is so configured.

PFS is a property that forces the peers (if they agree to it) to generate a new DH secret during the quick mode exchange. This allows the encryption keys used to encrypt data to be generated using a new DH secret. This ensures added secrecy. If the original DH secret were to somehow get compromised, this new secret can ensure privacy for the data. Also, because the negotiations for the new DH are carried out using encrypted traffic, this new DH secret has an added element of secrecy.

Note that it is not possible to negotiate the DH group the way it was offered and accepted in main mode, because quick mode has only one exchange in which the DH exchange must take place. Therefore, the PFS DH group configured on both ends must match.

To ensure PFS, the two peers generate the numbers needed to perform DH exchange once again. This is done in exactly the same manner as was done in phase 1 of IKE. The following list describes how this is done by the two peers. The apostrophe on top of the various values is used to signify the fact that these values are different from the DH values calculated in phase 1 of IKE:

- Execution of DH by the initiator again to ensure PFS:

  New nonce generated: N_i'

  New DH public value = X_a'

  $X_a' = g^a \bmod p$

  where

  g is the generator

  p is a large prime number

  a is a private secret known only to the initiator

- Execution of DH by the responder again to ensure PFS:

  New Nonce Generated: N_r'

  New DH public value = X_b'

  $X_b' = g^b \bmod p$

  where

  g is the generator

  p is a large prime number

  b is a private secret known only to the responder

IKE Phase 2 (Quick Mode): Sending Message 1

Message 1 from the initiator to the responder contains payloads you have already seen in the main mode negotiation. The contents of the payloads are, of course, different. Note the presence of the key exchange payload for requesting PFS. In addition, the new nonce is also sent across.

The hash payload contains a hash calculated on some of the elements that were agreed on in phase 1, as well as some of the payloads contained in this message itself. The purpose of the hash is to authenticate the peer again.

In addition, the message contains proposals and transform pairs for the IPsec SAs to be formed. The proposal payload includes the type of encapsulation to use, AH or ESP, and an SPI. The SPI is an interesting element. It is a 32-bit number that is chosen by the initiator to uniquely identify the outgoing IPsec SA that is generated as a result of this negotiation in its database of security associations (it might be talking to a lot more than one peer at a time). The responder, upon seeing the SPI, makes sure that it is not the same as one of the SPIs it is using and starts using it for its incoming IPsec SA. It also proposes an SPI for its outgoing (and the initiator's incoming) SA, which the initiator agrees to after checking.

The associated transform payload includes parameters such as tunnel or transport mode, SHA or MD5 for integrity checking in ESP or AH, and lifetimes for the IPsec security association.

A new payload type introduced in this message is the ID payload. The ID payload contains the proxy identities on whose behalf the initiator does the negotiation. These are generally IP address subnets, but they can have more fields, such as port, too. In the case of a site-to-site IPsec set up with two gateways doing IPsec negotiations with each other, the proxy IDs are based on rules defined on the gateways that define what type of traffic is supposed to be encrypted by the peers.

Note that most of the message is still encrypted using one of the keys generated in phase 1.

For the sake of our example, assume that the initiator offers the following transform attributes for the IPsec SA in one of the transform and proposal payload sets:

- **Encapsulation**—ESP
- **Integrity checking**—SHA-HMAC
- **DH group**—2
- **Mode**—Tunnel

Figure 13-13 shows the first message of IKE quick mode being sent.

NOTE Unlike IKE timeout negotiation, which is very strict, timeout negotiation in quick mode for the IPsec SA is more lenient. If the initiator proposes a transform that, all other things being acceptable, contains a timeout value that is higher than the timeout configured on the responder, the responder does not fail the negotiation. Instead, it simply returns the transform payload with the timeout modified to the lower value configured on it. The initiator then sets up the IPsec with this lower value.

Note that this behavior is different from the behavior shown in the negotiation of the IKE lifetime. A transform with a timeout higher than the one configured on the responder is unacceptable. If none of the transforms carry acceptable attributes, including an equal or lower timeout, the negotiation simply fails.

The difference in the behaviors is due to the fact that the IPsec SA timeout negotiation is done after authentication has taken place, whereas the IKE timeout negotiation is done between unauthenticated peers with no trust established.

Figure 13-13 *Sending the First Message of IKE Quick Mode*

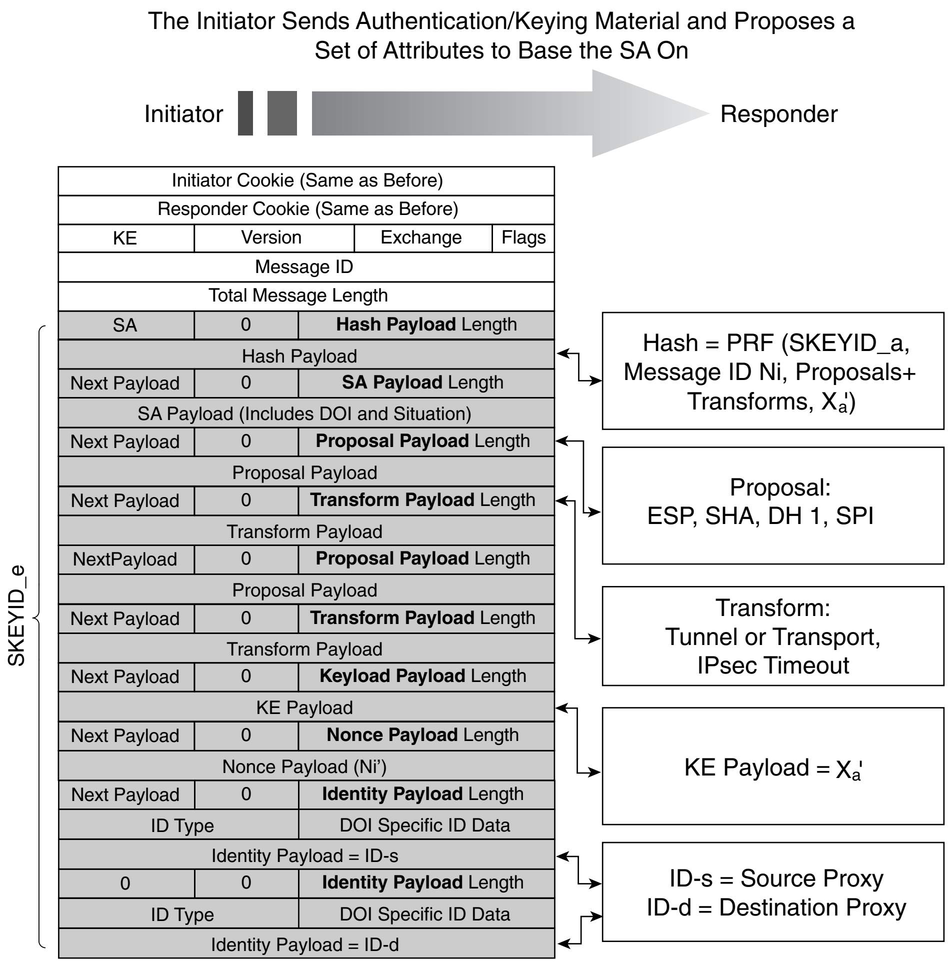
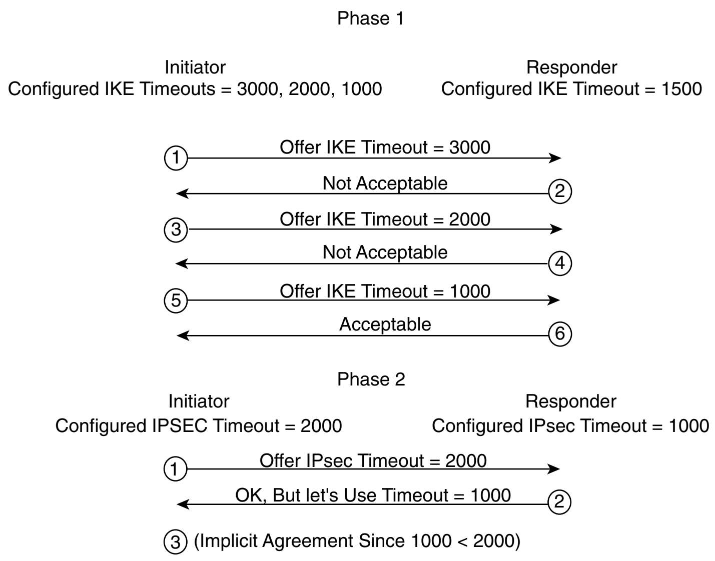


Figure 13-14 shows how the lifetimes are negotiated between two peers.

Figure 13-14 *IKE and IPsec Lifetime Negotiation*


Phase 1
Initiator
Configured IKE Timeouts = 3000, 2000, 1000
Responder
Configured IKE Timeout = 1500
1 Offer IKE Timeout = 3000
2 Not Acceptable
3 Offer IKE Timeout = 2000
4 Not Acceptable
5 Offer IKE Timeout = 1000
6 Acceptable
Phase 2
Initiator
Configured IPSEC Timeout = 2000
Responder
Configured IPsec Timeout = 1000
1 Offer IPsec Timeout = 2000
2 OK, But let's Use Timeout = 1000
3 (Implicit Agreement Since 1000 < 2000)

IKE Phase 2 (Quick Mode): Sending Message 2

The second message is sent by the responder in response to the initiator's first message. The fields are pretty much the same, with a few changes. Most notably, only the proposal and the transform payloads that were acceptable are returned to the initiator.

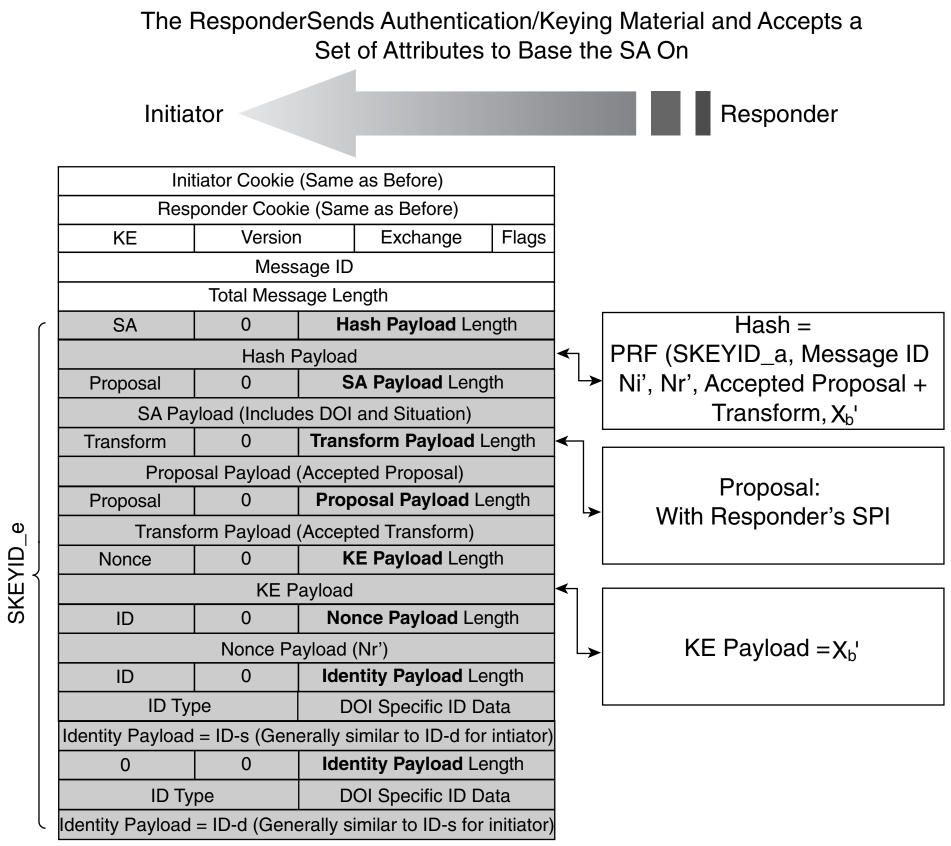
The proposal payload contains the SPI of the outgoing SA for the responder rather than the SPI that was sent by the initiator.

The hash payload contains the hash of only the payloads that are contained in this message. This hash acts as proof of the responder's liveness because it contains a hash of the two latest nonces as well the message ID, proving to the initiator that the responder has indeed received its initial packet and is ready to receive its encrypted messages.

The responder looks at the IDs offered by the initiator in the ID payloads and compares them to the proxy IDs defined on itself. If its policy permits the IDs that are being offered by the initiator, the negotiation continues. Otherwise, a failure occurs.

Figure 13-15 shows message 2 of quick mode being sent.

Figure 13-15 *Sending Message 2 of Quick Mode*


The ResponderSends Authentication/Keying Material and Accepts a Set of Attributes to Base the SA On
Initiator
Responder
Initiator Cookie (Same as Before)
Responder Cookie (Same as Before)
KE
Version
Exchange
Flags
Message ID
Total Message Length
SA
0
Hash Payload Length
Hash Payload
Proposal
0
SA Payload Length
SA Payload (Includes DOI and Situation)
Transform
0
Transform Payload Length
Proposal Payload (Accepted Proposal)
Proposal
0
Proposal Payload Length
Transform Payload (Accepted Transform)
Nonce
0
KE Payload Length
KE Payload
ID
0
Nonce Payload Length
Nonce Payload (Nr')
ID
0
Identity Payload Length
ID Type
DOI Specific ID Data
Identity Payload = ID-s (Generally similar to ID-d for intiator)
0
0
Identity Payload Length
ID Type
DOI Specific ID Data
Identity Payload = ID-d (Generally similar to ID-s for initiator)
SKEYID_e
Hash = PRF (SKEYID_a, Message ID Ni', Nr', Accepted Proposal + Transform, X_b')
Proposal: With Responder's SPI
KE Payload =X_b'

IKE Phase 2 (Quick Mode): Preparation for Sending Message 3

Before the last message can be sent for quick mode, the two ends must use the information sent relative to DH to generate a new DH secret and use this secret in conjunction with SKEYID_d and some of the other parameters to generate the IPsec encryption/decryption keys. The following are the steps taken by the two peers to generate the keys:

- Initiator generates IPsec keying material

  Step 1 Generate new DH shared secret = $(X_b')^a$ mod p

  Step 2 IPsec session key for incoming IPsec SA = PRF (SKEYID_d, protocol (ISAKMP), new DH shared secret, SPI_r, N_i', N_r')

  Step 3 IPsec session key for outgoing IPsec SA = PRF (SKEYID_d, protocol (ISAKMP), new DH shared secret, SPI_i, N_i', N_r')

- Responder generates IPsec keying material

  Step 1 Generate new DH shared secret = $(X_a')^b$ mod p

  Step 2 IPsec session key for incoming IPsec SA = PRF (SKEYID_d, protocol (ISAKMP), new DH shared secret, SPI_i, N_i', N_r')

  Step 3 IPsec session key for outgoing IPsec SA = PRF (SKEYID_d, protocol (ISAKMP), new DH shared secret, SPI_r, N_i', N_r')

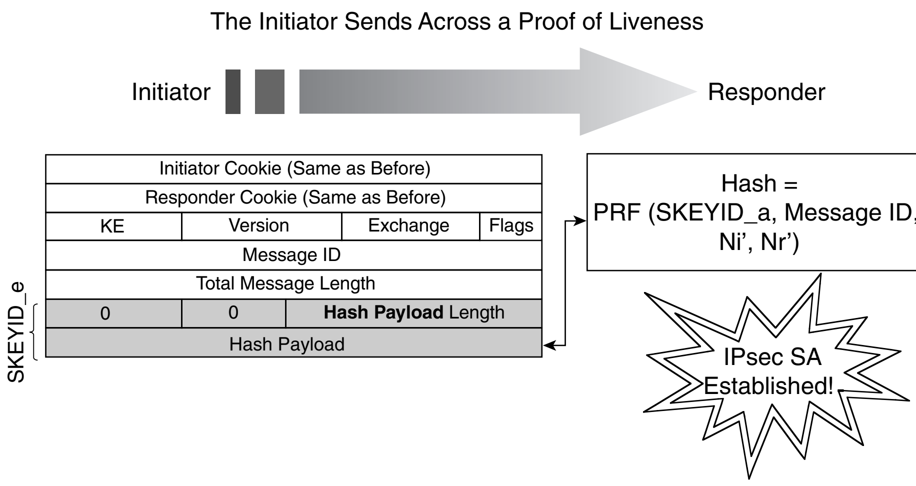
As soon as the IPsec key is generated, the tunnel is pretty much ready to become operational. Only one additional message is sent which concludes the quick mode negotiation.

Please note that two keys are generated on the initiator and the responder: one for the outgoing SA (which matches the key for the incoming SA key on the peer) and one for the incoming SA (which matches the key for the outgoing SA key on the peer).

IKE Phase 2 (Quick Mode): Sending Message 3

The last message of the IKE exchange is sent to verify the liveness of the responder. This is necessary for two reasons. The first reason is that the responder needs some way to know that the initiator received its first and only message of quick mode and correctly processed it. The second reason is to avoid a limited denial of service attack orchestrated by an attacker by replaying a first message of the quick mode exchange from the initiator to the receiver. By sending another message with the correct message ID and latest nonces hashed, the initiator proves to the responder that it has not only received its nonces but also is a live and current peer. Figure 13-16 shows how the final message is sent to the responder.

Figure 13-16 *Sending Message 3 of Quick Mode*


The Initiator Sends Across a Proof of Liveness
Initiator
Responder
Initiator Cookie (Same as Before)
Responder Cookie (Same as Before)
KE
Version
Exchange
Flags
Message ID
Total Message Length
0
0
Hash Payload Length
Hash Payload
SKEYID_e
Hash = PRF (SKEYID_a, Message ID, Ni', Nr')
IPsec SA Established!

After message 3 has been received by the receiver, IPsec's quick mode is concluded. All the goals of IKE—authentication, negotiation of the encryption algorithm, and other attributes needed to generate the encrypted packets—have been negotiated. Now the agreed-on IPsec SAs can be used to exchange encrypted traffic.

NOTE Note that while it is not a requirement per the RFC, the responder can also send the initiator a message similar to the message 3 discussed above to verify its liveliness.

Main Mode Using Digital Signature Authentication Followed by Quick Mode Negotiation

Another important method of authentication used in IKE is digital signatures. We will discuss the use and workings of digital signatures in a later section. Here the changes in main mode of IKE negotiation are described. You might want to read the section on digital signatures (PKI) first and then come back to this section.

The only difference in the preshared and digital signatures method of authentication occurs in the fifth and sixth IKE messages that are sent. The first four messages of main mode and all the quick mode messages remain the same as in preshared key-based IKE negotiation.

Let's look at the preparation and sending of messages 5 and 6 for digital signatures to appreciate the difference.

IKE Phase 1 (Main Mode): Preparation for Sending Messages 5 and 6 Using Digital Signatures

The first difference that is quite evident is the way the SKEYs are generated when using digital signatures. Instead of generating PRF outputs with the preshared secret as one of the components, the PRF output is generated using the DH secret g^{ab} instead, along with the rest of the parameters, which are the same as the ones used in the preshared method. Obviously, at this point, anyone who knows how to do IPsec can negotiate (using a hit-and-miss method to discover configured transforms and proposals) and have the keys generated, which would be valid on both ends. The effect that the preshared secret has, of limiting successful negotiations to those peers that have the same preshared key configured, is not visible in this method up until this point. In order to achieve the same level of control when using digital certificates, a successful negotiation is restricted by either manually setting up each peer with the public key of the peers to which it is allowed to connect or enrolling each peer in a certificate authority (CA) server with a certain organization name. All peers to which the peer is allowed to connect must enroll with the same certificate authority server and belong to the same organization. Messages 5 and 6 of the exchange contain the certificates of the two peers. These certificates contain the identity of the two peers, including the organization ID.

Figure 13-17 shows the mechanism through which the keys are generated on the initiator in the digital signatures method of conducting main mode negotiation.

Figure 13-17 *How Keys Are Generated on the Initiator in the Digital Signatures Method of Main Mode Negotiation*

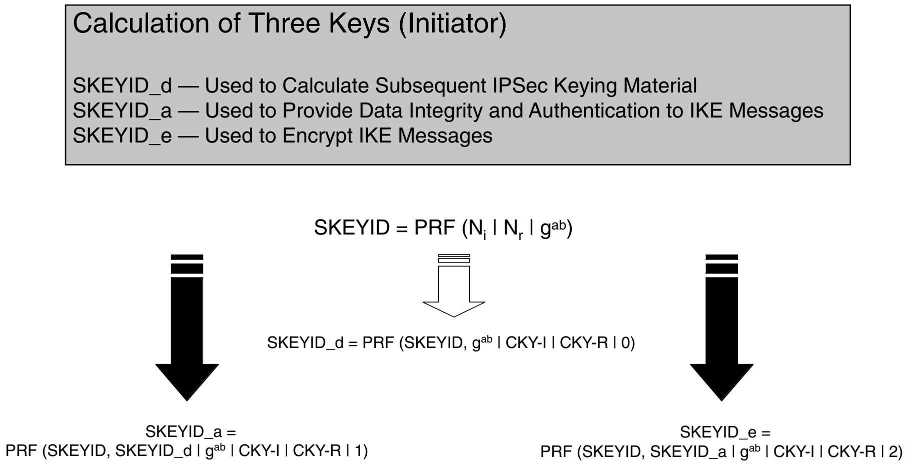
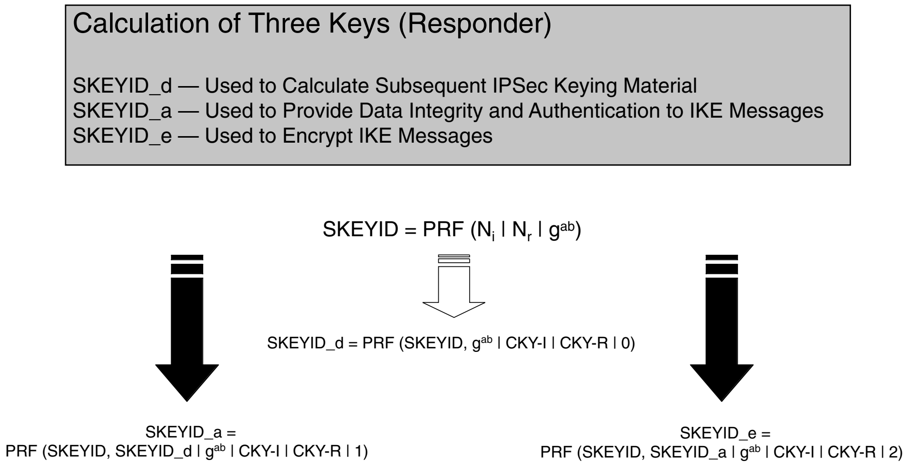

Calculation of Three Keys (Initiator)
SKEYID_d — Used to Calculate Subsequent IPSec Keying Material
SKEYID_a — Used to Provide Data Integrity and Authentication to IKE Messages
SKEYID_e — Used to Encrypt IKE Messages
SKEYID = PRF (N_i I N_r I g^ab)
SKEYID_d = PRF (SKEYID, g^ab I CKY-I I CKY-R I 0)
SKEYID_a = PRF (SKEYID, SKEYID_d I g^ab I CKY-I I CKY-R I 1)
SKEYID_e = PRF (SKEYID, SKEYID_a I g^ab I CKY-I I CKY-R I 2)

Figure 13-18 shows the mechanism through which the keys are generated on the responder in the digital signatures method of conducting main mode negotiation.

Figure 13-18 *How Keys Are Generated on the Responder in the Digital Signatures Method of Main Mode Negotiation*


Calculation of Three Keys (Responder)
SKEYID_d — Used to Calculate Subsequent IPSec Keying Material
SKEYID_a — Used to Provide Data Integrity and Authentication to IKE Messages
SKEYID_e — Used to Encrypt IKE Messages
SKEYID = PRF (N_i I N_r I g^ab)
SKEYID_d = PRF (SKEYID, g^ab I CKY-I I CKY-R I 0)
SKEYID_a = PRF (SKEYID, SKEYID_d I g^ab I CKY-I I CKY-R I 1)
SKEYID_e = PRF (SKEYID, SKEYID_a I g^ab I CKY-I I CKY-R I 2)

IKE Phase 1 (Main Mode): Sending Message 5 Using Digital Signatures

The message 5 sent by the initiator contains two new payloads in addition to the ones we have already looked at—signature and certificate.

Signature Payload

This payload contains the initiator's signature. In general, a signature is a known value encrypted with an individual's private key. A signature provides nonrepudiation, meaning that because data encrypted using a private key can only be decrypted using its corresponding public key, the sender of the data cannot back out of admitting that he or she sent the data. No one else but that person possesses the private key (corresponding to the public key) that was used to decrypt the data. Obviously, the assumption is that the data is known a priori to both the sender and the receiver. In the case of message 5, this information is indeed a combination of information that has either already been exchanged between the two parties or that has been generated based on exchanged information and is the same on both sides (for example, the SKEYID).

The hash that is encrypted using the private key is generated by hashing the accepted proposal and transform payloads along with certain other values, as shown in Figure 13-19.

Certificate Payload

We will discuss the format of a certificate in detail in the section on IKE authentication methods. For now, it is sufficient to say that the certificate contained in the certificate payload is tied to a unique host name (or some other similar attribute) that is the sender's host name. The certificate also contains the sender's public key, which is used to decrypt the signature sent in the same message. The certificate is issued for a certain organization ID. Both the IPsec peers must belong to the same organization, and both must trust the CA issuing the certificate for the certificate to be acceptable to both of them.

Figure 13-19 shows the initiator sending message 5 in the digital signatures method of conducting main mode negotiation.

IKE Phase 1 (Main Mode): Sending Message 6 Using Digital Signatures

The message 6 sent by the responder to the initiator is pretty much the same as the message sent by the initiator. It contains the certificate of the responder and the signature generated using the private key of the responder and the corresponding values stored on it through the IKE exchanges so far.

Figure 13-20 shows the responder sending message 6 in the digital signatures method of conducting main mode negotiation.

Figure 13-19 *Initiator Sending Message 5 in the Digital Signatures Method of Main Mode Negotiation*

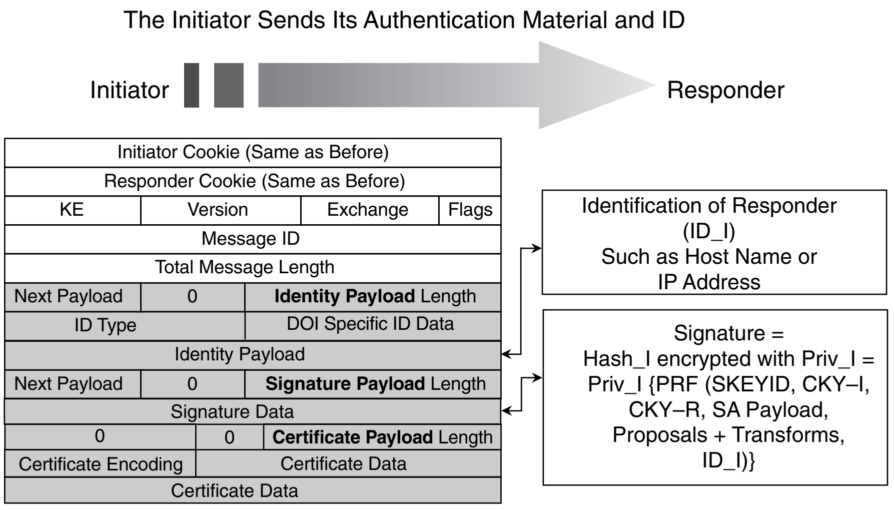
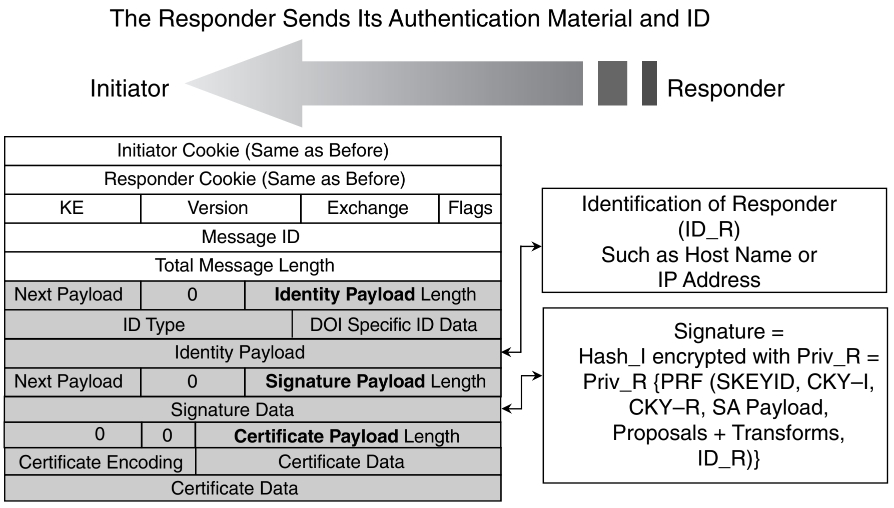

The Initiator Sends Its Authentication Material and ID
Initiator
Responder
Initiator Cookie (Same as Before)
Responder Cookie (Same as Before)
KE
Version
Exchange
Flags
Message ID
Total Message Length
Next Payload
0
Identity Payload Length
ID Type
DOI Specific ID Data
Identity Payload
Next Payload
0
Signature Payload Length
Signature Data
0
0
Certificate Payload Length
Certificate Encoding
Certificate Data
Certificate Data
Identification of Responder (ID_I) Such as Host Name or IP Address
Signature = Hash_I encrypted with Priv_I = Priv_I {PRF (SKEYID, CKY–I, CKY–R, SA Payload, Proposals + Transforms, ID_I)}

Figure 13-20 *Responder Sending Message 6 in the Digital Signatures Method of Main Mode Negotiation*


The Responder Sends Its Authentication Material and ID
Initiator
Responder
Initiator Cookie (Same as Before)
Responder Cookie (Same as Before)
KE
Version
Exchange
Flags
Message ID
Total Message Length
Next Payload
0
Identity Payload Length
ID Type
DOI Specific ID Data
Identity Payload
Next Payload
0
Signature Payload Length
Signature Data
0
0
Certificate Payload Length
Certificate Encoding
Certificate Data
Certificate Data
Identification of Responder (ID_R) Such as Host Name or IP Address
Signature = Hash_I encrypted with Priv_R = Priv_R {PRF (SKEYID, CKY–I, CKY–R, SA Payload, Proposals + Transforms, ID_R)}

IKE Phase 1 (Main Mode): Completion of Authentication Using Digital Signatures

As soon as both the initiator and responder have compared the ID of the peer with the ID set up in its configuration by the system administrator as the ID to which it is allowed to establish a peering relationship, the authentication process using digital signatures is completed by both the initiator and the responder. This is done by decrypting the encrypted Hash_I using the public key of the other side received through its certificate. The hashes are then compared in a manner similar to the method used in preshared authentication to verify that they are the same. If they are, the two sides are assumed to be authenticated.

- Initiator authenticates the responder

  Step 1 Decrypt the message using SKEYID_E.

  Step 2 Decrypt Hash_R using Pub_R.

  Step 3 Calculate Hash_R on its own.

  Step 4 If received Hash_R = self-generated Hash_R, authentication is successful!

- Responder authenticates the initiator

  Step 1 Decrypt the message using SKEYID_E.

  Step 2 Decrypt Hash_I using Pub_I.

  Step 3 Calculate Hash_I on its own.

  Step 4 If received Hash_I = self-generated Hash_I, authentication is successful!

As soon as main mode has been completed as just described, quick mode goes through exactly the same steps as in the preshared authentication method.

Aggressive Mode Using Preshared Key Authentication

This section discusses aggressive mode, an alternative for main mode functions. Aggressive mode is completed using only three messages instead of the six used in main mode. However, the speed comes at a cost. We will discuss the shortcomings at the end of this section, after you have seen how aggressive mode works.

IKE Phase 1 (Aggressive Mode): Preparation for Sending Messages 1 and 2

The preparation for sending messages 1 and 2 in aggressive mode is the same as the combination of the preparations done in main mode prior to sending messages 1, 2, 3, and 4.

Essentially, all the information needed to generate the DH secret is exchanged in the first two messages exchanged between the two peers. No opportunity is given to negotiate the DH group by offering a series of DH group values by the initiator to the receiver. Instead, both have to agree on a DH group immediately or fail negotiation.

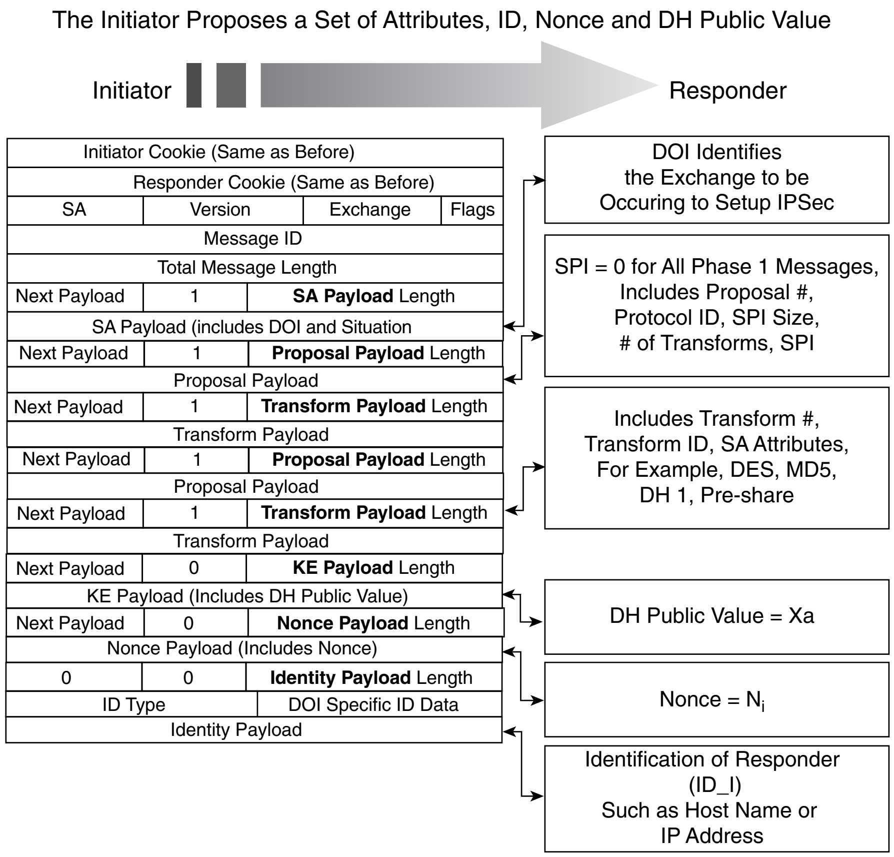
In addition, if the preshared secret method of authentication is being used in aggressive mode, the identity of the peer that is also exchanged in the first two packets is sent in the clear. This is unlike main mode negotiation. However, the advantage of this is that the ID can now be used to search for the pre-shared key belonging to the peer. As mentioned in the discussion of main mode, this ID does not arrive until after the pre-shared key has been found by using the source IP address of the negotiation IP packets.

IKE Phase 1 (Aggressive Mode): Sending Message 1

The first message sent by the initiator contains all the material needed for the responder to generate a DH secret. This means that the initiator sends the key exchange payload and the nonce payload with the appropriate values generated in them. In addition, an ID payload is sent. The message also includes one or more pairs of proposal and transform payloads for the responder to chose from.

Figure 13-21 shows the first message of IKE aggressive mode being sent from the initiator to the responder.

Figure 13-21 *Sending the First Message of IKE Aggressive Mode from the Initiator to the Responder*


The Initiator Proposes a Set of Attributes, ID, Nonce and DH Public Value
Initiator
Responder
Initiator Cookie (Same as Before)
Responder Cookie (Same as Before)
SA
Version
Exchange
Flags
Message ID
Total Message Length
Next Payload
1
SA Payload Length
SA Payload (includes DOI and Situation
Next Payload
1
Proposal Payload Length
Proposal Payload
Next Payload
1
Transform Payload Length
Transform Payload
Next Payload
1
Proposal Payload Length
Proposal Payload
Next Payload
1
Transform Payload Length
Transform Payload
Next Payload
0
KE Payload Length
KE Payload (Includes DH Public Value)
Next Payload
0
Nonce Payload Length
Nonce Payload (Includes Nonce)
0
0
Identity Payload Length
ID Type
DOI Specific ID Data
Identity Payload
DOI Identifies the Exchange to be Occuring to Setup IPSec
SPI = 0 for All Phase 1 Messages, Includes Proposal #, Protocol ID, SPI Size, # of Transforms, SPI
Includes Transform #, Transform ID, SA Attributes, For Example, DES, MD5, DH 1, Pre-share
DH Public Value = Xa
Nonce = N_i
Identification of Responder (ID_I) Such as Host Name or IP Address

All the payloads sent in message 1 were described in detail in the section on main mode.

IKE Phase 1 (Aggressive Mode): Preparation for Sending Message 2

Upon the receipt of the first packet, the responder has all the material it needs to generate the DH secret. It goes ahead and generates the DH secret. If the preshared secret is being used, the responder finds the preshared secret by using the ID from the ID payload it received in the previous message. It then uses the DH secret and the preshared secret along with other attributes to calculate the three keys, just as is done in main mode. Having created the keys, it uses the SKEYID to generate Hash_R for authentication purposes.

IKE Phase 1 (Aggressive Mode): Sending Message 2

Message 2, sent by the responder to the initiator, contains the proposal and transform pair to which the responder has agreed, as well as the hash for authentication purposes generated by the responder. It also contains all the material the initiator needs to generate its DH secret. The message also contains the responder's ID.

Figure 13-22 shows the second message of IKE aggressive mode being sent from the responder to the initiator.

IKE Phase 1 (Aggressive Mode): Preparation for Sending Message 3

Upon receipt of message 2, the initiator calculates the DH secret as well, finds the preshared key, generates the SKEYS, and proceeds to generate Hash_R on its own. If the hash it generates matches the one that is sent by the responder, authentication is supposed to have occurred. The initiator now generates Hash_I to allow the responder to do the authentication as well.

IKE Phase 1 (Aggressive Mode): Sending Message 3

The initiator sends message 3 of the exchange, containing a hash payload loaded with the Hash_I it has generated.

Figure 13-23 shows the third message of IKE aggressive mode being sent from the initiator to the responder.

Figure 13-22 *Sending the Second Message of IKE Aggressive Mode from the Responder to the Initiator*

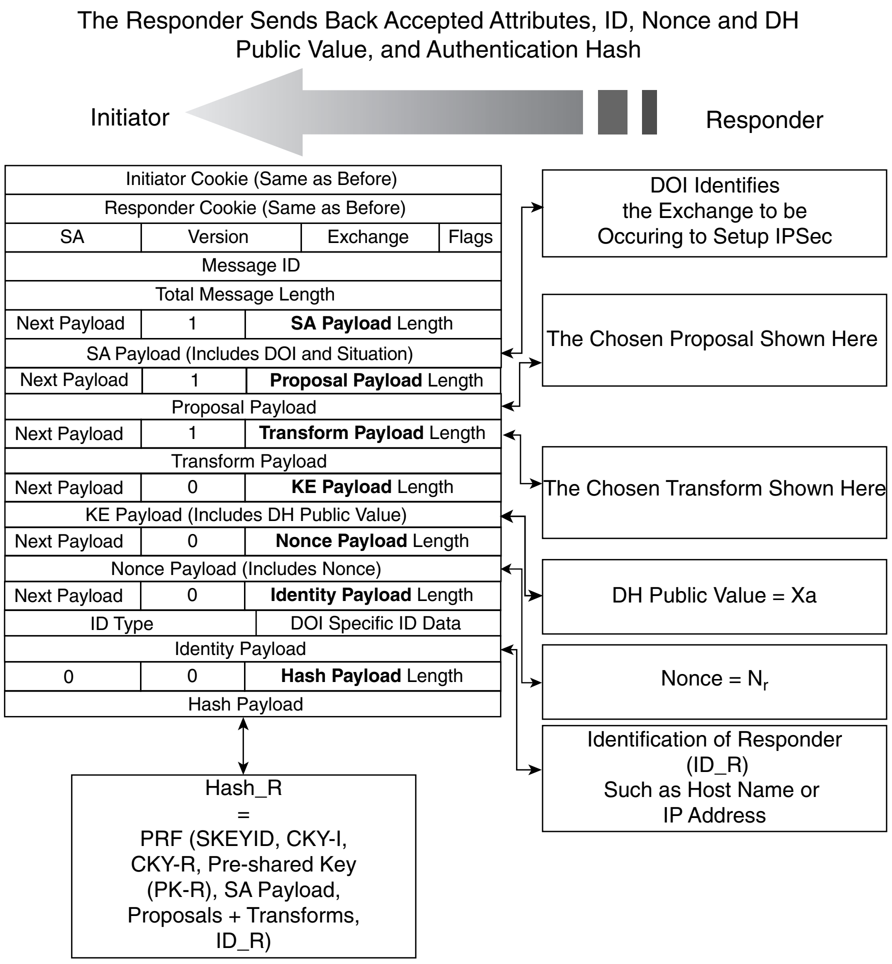
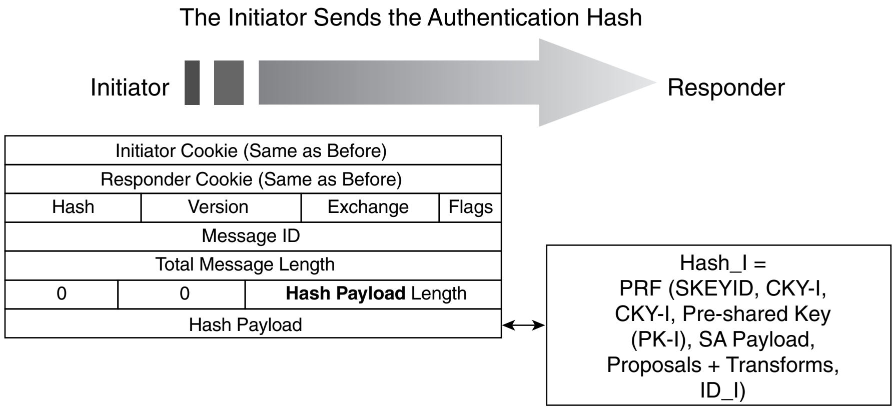

The Responder Sends Back Accepted Attributes, ID, Nonce and DH Public Value, and Authentication Hash
Initiator
Responder
Initiator Cookie (Same as Before)
Responder Cookie (Same as Before)
SA
Version
Exchange
Flags
Message ID
Total Message Length
Next Payload
1
SA Payload Length
SA Payload (Includes DOI and Situation)
Next Payload
1
Proposal Payload Length
Proposal Payload
Next Payload
1
Transform Payload Length
Transform Payload
Next Payload
0
KE Payload Length
KE Payload (Includes DH Public Value)
Next Payload
0
Nonce Payload Length
Nonce Payload (Includes Nonce)
Next Payload
0
Identity Payload Length
ID Type
DOI Specific ID Data
Identity Payload
0
0
Hash Payload Length
Hash Payload
DOI Identifies the Exchange to be Occuring to Setup IPSec
The Chosen Proposal Shown Here
The Chosen Transform Shown Here
DH Public Value = Xa
Nonce = N_r
Identification of Responder (ID_R) Such as Host Name or IP Address
Hash_R = PRF (SKEYID, CKY-I, CKY-R, Pre-shared Key (PK-R), SA Payload, Proposals + Transforms, ID_R)

Figure 13-23 *Sending the Third Message of IKE Aggressive Mode from the Initiator to the Responder*


The Initiator Sends the Authentication Hash
Initiator
Responder
Initiator Cookie (Same as Before)
Responder Cookie (Same as Before)
Hash
Version
Exchange
Flags
Message ID
Total Message Length
0
0
Hash Payload Length
Hash Payload
Hash_I = PRF (SKEYID, CKY-I, CKY-I, Pre-shared Key (PK-I), SA Payload, Proposals + Transforms, ID_I)

Upon receipt of this hash, the responder generates the hash on its own and compares it to the Hash_I it has received. If they match, authentication is supposed to have occurred on the responder end as well.

This concludes aggressive mode. The keying material has been generated, and the peers have been authenticated. The quick mode that follows is exactly the same as the one that follows main mode.

IKE Authentication Mechanisms

Primarily three methods of authentication can be used to authenticate peers that want to establish an IPsec tunnel between themselves. Note that this is called *device authentication* rather than user authentication. The difference is that in device authentication, the devices that are actually doing IPsec communication on behalf of a user authenticate each other. User authentication (also called extended authentication in IPsec) is done in addition to device authentication and is for specific users who are making use of the IPsec services offered by the IPsec-enabled devices. We will talk about extended authentication in a later section of this chapter.

The three main mechanisms of device authentication are

- Preshared keys
- Digital signatures
- Encrypted nonces

Preshared Keys

Preshared keys can be used to authenticate peers by defining the same key on both the IPsec peers. This method of authentication is obviously very simple to set up. However, this is not a very scalable method. For example, if an IPsec device has 100 peers, 100 preshared keys need to be set up manually—one for each. This process has to be repeated on each of the 101 devices.

Also, this method does not scale when the remote clients are trying to authenticate to an IPsec gateway using main mode authentication. These clients are assigned IP addresses from their ISPs unknown to the IPsec gateway. Because in main mode authentication the peer's IP address is used to find its key to do the authentication, it is not possible to define a key on the gateway that can be used to authenticate these clients. One way around this problem is to use aggressive mode, which exchanges the ID of the IPsec peers in the first message exchanged. This allows the ID, if it is sent as the name of the remote client, to be used to find the preshared key for it. However, this is not a very secure method, because the ID is sent in the clear. We will discuss an enhancement to this method using extended authentication in a later section.

What Are Public and Private Keys?

Both of the methods that we will discuss next use the concept of a public/private key pair. Public/private key pairs are an important concept in security. Basically, the public/private key pair provides the following important properties:

- Data encrypted using the public key can be decrypted only using the corresponding private key.
- Data encrypted using the private key can be decrypted only using the corresponding public key.

The first property is used for confidentiality purposes. The private key of a public/private pair is known only to the owner of the peer and no one else. On the other hand, the public key can be known to as many individuals or devices that need to send data confidentially to the owner. So, data sent to the owner of the pair encrypted using the public key can be decrypted only by the owner, because no one else but the owner possesses the private key to decrypt the data. This makes the data confidential.

The second property is used for authentication purposes. To authenticate himself to someone, the owner can send data encrypted using his private key to the other party. He also sends the original data along with the encrypted data. If the receiver can decrypt the data using the sender's public key, it knows the sender is who he says he is, because no one else could have correctly encrypted the data.

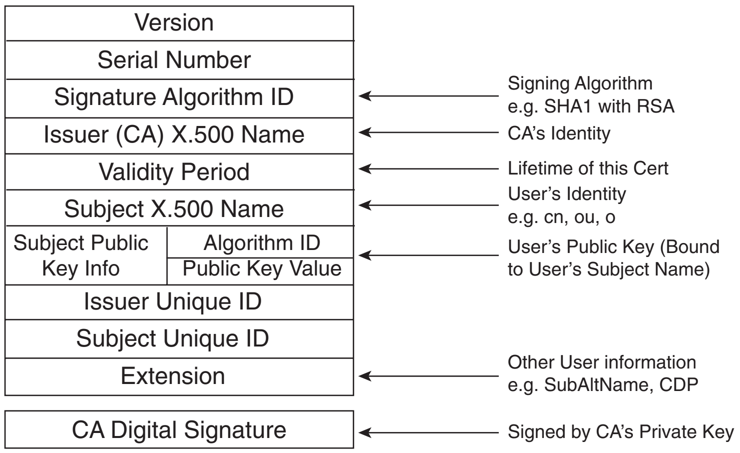
You will see the use of these properties in the methods discussed next.

Digital Signatures

Digital signatures, one of the methods you saw used in the earlier description of IKE, is a method that lends itself to being scalable when used in conjunction with a certificate authority. The authentication relies on the use of public/private key pairs being generated on both the IPsec peers. The public keys are exchanged either out of band the way preshared keys are exchanged, by the system administrators of the two IPsec peers, or during the IKE negotiation (using digital certificates). When the keys are exchanged during the IKE negotiation, they are contained in a certificate payload. The certificate, in addition to holding the public key, has information about the sender, such as the sender's name and the organization he belongs to.

Figure 13-24 shows the contents of a digital certificate.

Figure 13-24 *Contents of a Digital Certificate*



For the certificate to be acceptable to the receiver, it must be issued by a certificate authority server that is trusted by both the sender and the receiver. Also, the sender and the receiver must belong to the same organizational unit. This ensures that only the IPsec peers who are administratively allowed to speak to each other can do so.

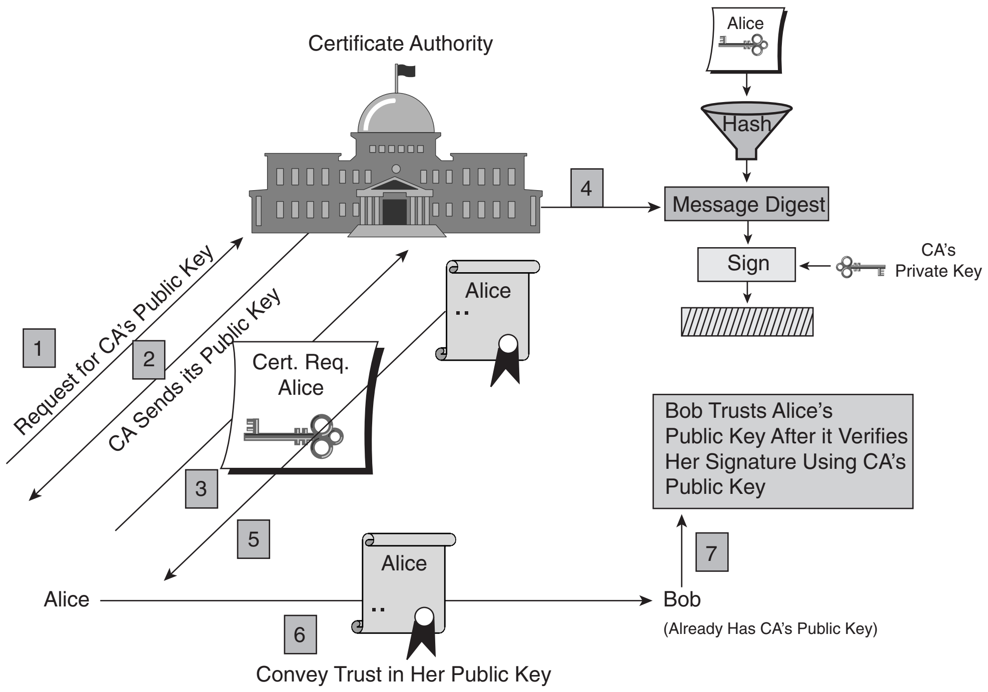
Another important benefit that digital signatures provide is that of nonrepudiation. Nonrepudiation means that the sender cannot back out of the claim that he or she initiated an authentication exchange. This happens because the digital signature sent during the IKE negotiation is encrypted using the sender's private key. It can be decrypted only using the sender's public key. Because the contents of the signature or the clear-text info (Hash_I or Hash_R) is known to both peers, the peer can say for sure that the sender sent the signature, because it could only have been encrypted using the sender's private key.

The process of using digital certificates starts with each of the IPsec peers sending a request for the CA certificate (containing the CA's public key) to the CA. The CA responds with the certificate. The peer then sends a certificate request encrypted using the CA's public key to the CA. The request contains the peer's public key. The CA sends back a certificate containing information about the peer, such the peer's name, IP address, and the organization it belongs to. The certificate also contains the peer's public key and a signature generated by encrypting the public key and the rest of the information in the certificate using the CA's private key.

When a certificate is offered by a peer to another peer, the receiving party can decrypt the signature using the CA's public key and thus verify that the public key contained in the certificate was actually issued to the sender by the CA.

Figure 13-25 shows the process of using digital certificates.

Figure 13-25 *Using Digital Certificates*

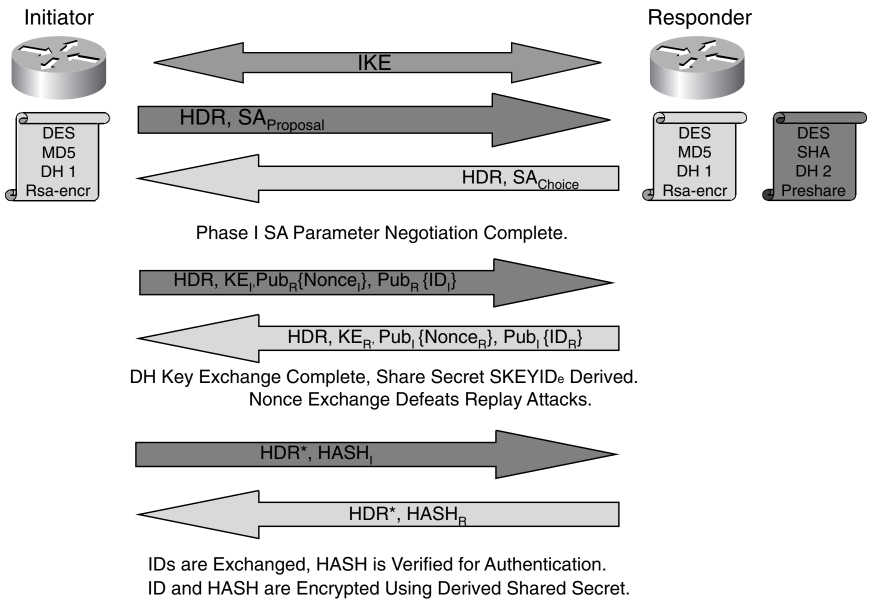

Certificate Authority
Alice
Hash
4
Message Digest
Sign
CA's Private Key
1
Request for CA's Public Key
2
CA Sends its Public Key
Cert. Req. Alice
Alice
3
5
Bob Trusts Alice's Public Key After it Verifies Her Signature Using CA's Public Key
7
Alice
Alice
Bob
(Already Has CA's Public Key)
6
Convey Trust in Her Public Key

Encrypted Nonces

Another authentication method that can be used in IKE is encrypted nonces. As the name suggests, this method involves encrypting the nonces that are sent during the DH information exchange. Nonces, as you might recall, are pseudo-random numbers exchanged between the peers and used in the generation of the SKEYs. Because encrypting the nonces can protect against some man-in-the-middle attacks that can be staged against the DH algorithm, this is a useful technique to employ. The nonces are encrypted by the initiator using the receiver's public keys. The public keys need to be exchanged between the peers before the IKE negotiation begins. This is necessary because the third and fourth packets of the exchange, which carry the nonces, need to have the nonces encrypted via the peer's public key, and there is no mechanism for certificate exchange before the third message of the exchange. When encrypted nonces are used, the peers also send their IDs encrypted with their respective public keys.

Figure 13-26 shows IKE main mode negotiation using encrypted nonces.

Figure 13-26 *IKE Main Mode Negotiation Using Encrypted Nonces*


Initiator
Responder
IKE
DES
MD5
DH 1
Rsa-encr
HDR, SA_Proposal
HDR, SA_Choice
DES
MD5
DH 1
Rsa-encr
DES
SHA
DH 2
Preshare
Phase I SA Parameter Negotiation Complete.
HDR, KE_I, Pub_R{Nonce_I}, Pub_R {ID_I}
HDR, KE_R, Pub_I {Nonce_R}, Pub_I {ID_R}
DH Key Exchange Complete, Share Secret SKEYIDe Derived.
Nonce Exchange Defeats Replay Attacks.
HDR*, HASH_I
HDR*, HASH_R
IDs are Exchanged, HASH is Verified for Authentication.
ID and HASH are Encrypted Using Derived Shared Secret.

Encryption and Integrity-Checking Mechanisms in IPsec

As soon as the IPsec SAs are established, the next step is to start performing the encryption using DES or 3DES encryption if ESP was set up as the encapsulation mechanism. In addition to encryption, both ESP and AH require that integrity checking hashes be included in the packets. This section discusses how encryption and integrity checking work.

Encryption

Encryption is performed using a symmetric key algorithm known as Data Encryption Standard (DES) or its stronger cousin, Triple DES (3DES). *Symmetric key algorithm* means that a single key is used to encrypt as well as decrypt data. Although the keys for the incoming and outgoing IPsec SAs on a given peer are different, the key used by the sender to encrypt the data and the key used by the receiver to decrypt the data are the same. This key is the one that is established during IKE's quick mode.

DES is used in its Cipher Block Chaining (CBC) mode of operation. Cipher blocking chaining connects a series of cipher blocks to encrypt the clear text. A cipher block is basically the DES encryption algorithm that converts a fixed-length block of clear text into a block of cipher text of the same length using a key that is known to the encryptor as well as the decryptor. Decryption is done applying the same algorithm in reverse using the same key. The fixed length of the clear text is called the *block size*. The block size for DES is 64 bits. The key used is 56 bits in length.

To encrypt a clear-text message that might be longer than the 64-bit block size, the individual algorithm blocks are chained. CBC is the most commonly deployed method of chaining. In CBC mode, each clear-text block is XORed with the encrypted output from the operation done on the previous block of clear text. An initialization vector (IV) is used as the "seed" for providing the input for the first block of data. This ensures that the same information fed into the CBC chain does not repeatedly produce the same output.

Figure 13-27 shows the workings of the DES CBC mode.

The initialization vector for any given packet is different from the initialization vector used for any other packet. The IV is a random sequence of bits. It is sent to the receiver of the encrypted packet in the packet's ESP header. The IV is encrypted to further protect the data's confidentiality. Because it is only 64 bits in length, it can be encrypted using a single cipher block, allowing it to be decrypted on the packet's receipt and then used in the chaining to decrypt the rest of the packet.

3DES encryption is done by chaining the CBC blocks, as shown in Figure 13-28. 3DES encryption is achieved by first encrypting a 64-bit block of data using a 56-bit key and a CBC DES block. The encrypted data is then decrypted using a different 56-bit key. The result is then encrypted using another 56-bit key. The resultant total key length is 56 * 3 = 168 bits.

Figure 13-27 *DES CBC Mode*

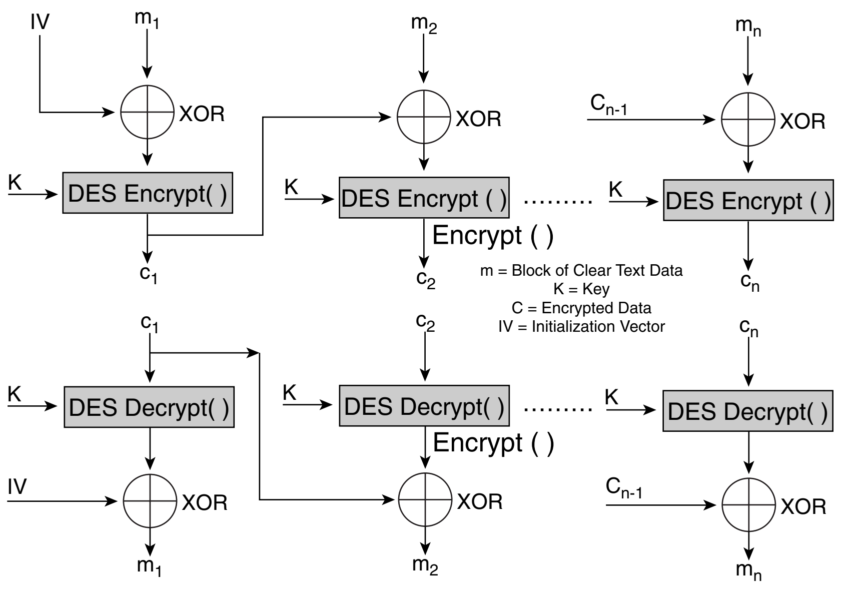
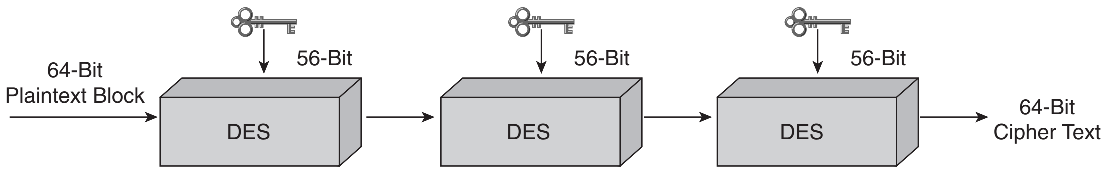


Figure 13-28 *3DES Encryption*

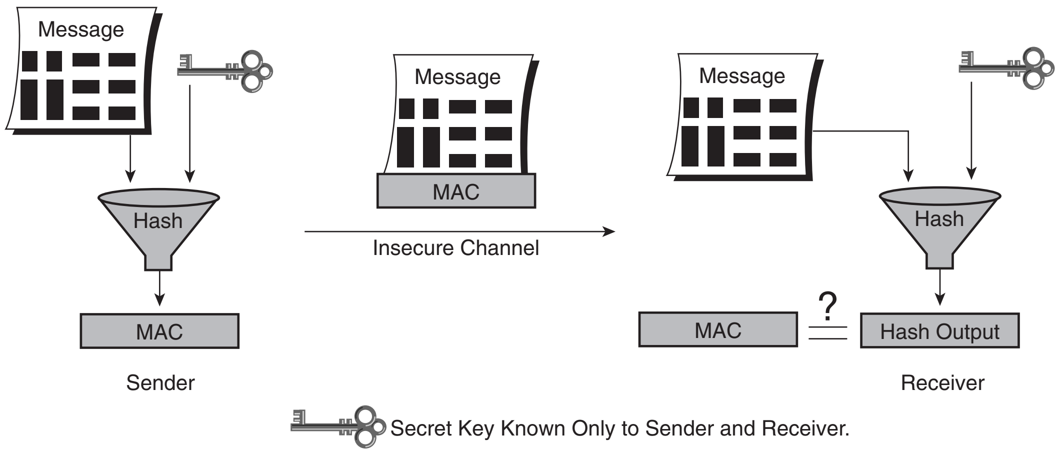


Integrity Checking

The integrity checking in ESP and AH is done using HMAC (Message Authentication Codes using Hashing), a mechanism for message authentication using cryptographic hash functions. HMAC can be used with any iterative cryptographic hash function in combination with a shared secret key. IPsec uses the MD5 or SHA-1 hashing algorithm. Hashing (as opposed to encryption) is a nonreversible function, meaning that as soon as a hash has been created from clear text, it is impossible or extremely difficult to retrieve the clear text from the cipher text. The hashes in ESP and AH work by calculating a hash on the packet whose integrity is to be ensured and attaching this hash to the end of the packet. When the receiver gets the packet, it again calculates the hash on the packet and finds out if the hash it received and the one it calculated, match. If they do, the data was not tampered with in transit.

Figure 13-29 shows integrity checking using hashes.

Figure 13-29 *Integrity Checking Using Hashes*



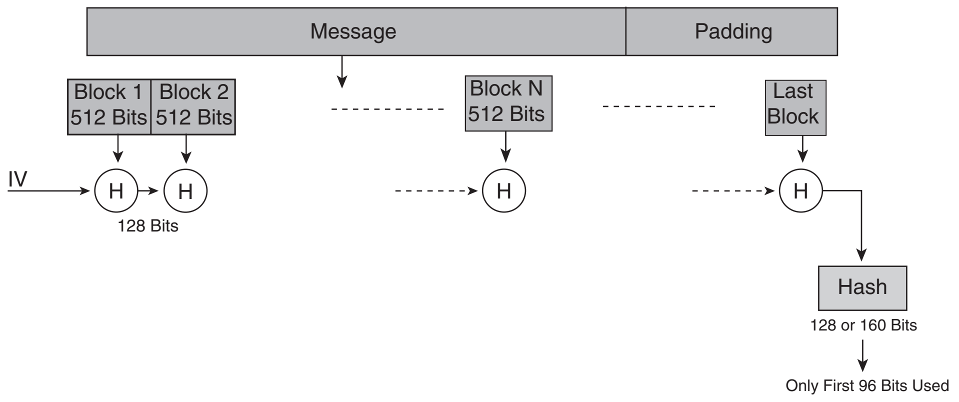
Integrity checking is done on the entire original packet in ESP, including the ESP header. In AH, not only the original packet and the AH header are included in the integrity check, but the new IP header attached to the packet is included as well. Although this can be useful against some types of attacks involving modification of packet headers in transit, it can be very troublesome in environments where NAT occurs on the packets. Because NAT modifies the IP header, AH considers the integrity checking to have failed for that reason. ESP does not have this problem, because its integrity check does not include the new IP header.

MD5 provides 128 bits of output, and SH provides 160 bits of output. However, both of these outputs are concatenated at the 96th bit, and only the first 96 bits are included as the integrity hash in the packets being sent. SHA, although computationally slower than MD5, is considered the stronger of the two hashes.

Figure 13-30 shows how hashes are used in ESP and AH for integrity checking.

The next section contains more details on how ESP and AH encapsulation occur in IPsec.

Figure 13-30 *Use of Hashes in ESP and AH*



Packet Encapsulation in IPsec

Packet encapsulation is handled by ESP or AH or both for an IPsec tunnel. Encapsulation includes encrypting the data portion of the header if ESP is being used, adding the appropriate header to provide the IPsec peer with information on how to decrypt the data (for ESP), and generating hashes to be used by the peer for verifying that the data (and the IP header in the case of AH) was not tampered with in transit.

Encapsulation can occur in two main ways:

- Transport mode
- Tunnel mode

Transport Mode

In transport mode, the original IP header of the packet that is being encrypted is used to transport the packet. An additional header for ESP or AH (or both) is inserted between the packet's IP header and its IP payload. This mode of operation requires that the original IP header contain addresses that can be routed over the public network.

Tunnel Mode

In tunnel mode, the original IP header is not used to transport the packet. Instead, a new IP header is tagged in front of the ESP or AH (or both) header. This IP header contains the IP addresses of the two IPsec peers as the source and destination IP addresses rather than the IP addresses of the originating host and the destination host. This mode allows end hosts with RFC 1918 IP addressing to participate in a virtual private network set up across the Internet, because the IP addresses of the IPsec gateways are used to transport and route the packets rather than the IP addresses of the end hosts. Tunnel mode is by far the most widely used mode in IPsec deployments.

Figure 13-31 shows the format of the packet that is created using tunnel and transport modes using AH.

Figure 13-31 *Packet Format Using AH in Tunnel and Transport Modes*

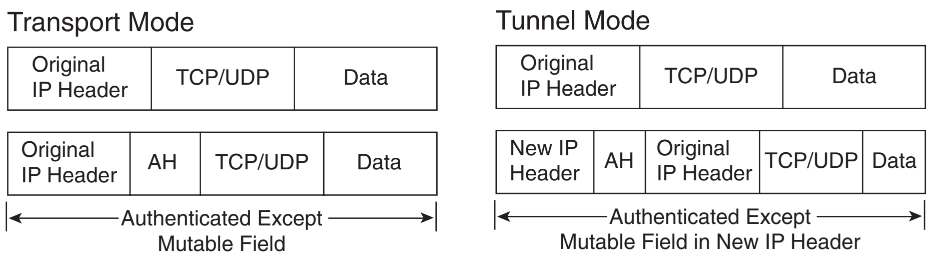
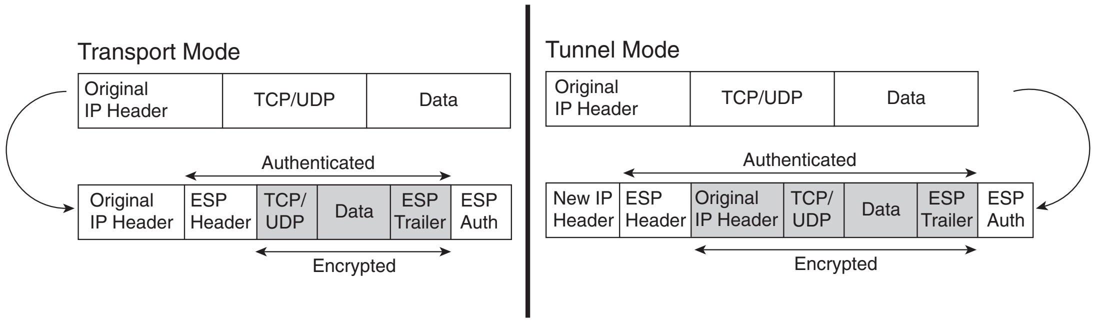


Figure 13-32 shows the format of the packet that is created using tunnel and transport modes using ESP.

Figure 13-32 *Packet Format Using ESP in Tunnel and Transport Modes*



Note the difference in the amount of the packet that AH and ESP provide authentication or integrity checking for. Although ESP provides integrity checking for all of the packet expect the IP header, AH provides integrity checking for the IP header as well. This means that, for example, if the IP header is modified during transit due to NAT, the hash checking fails at the receiving IPsec peer. This can be a source of problems in environments that do static NAT translation in the path of the IPsec tunnel.

Also note that AH and ESP can be used together as well. In that case, the AH header is tagged on, and the associated processing is done first, followed by the ESP processing. Obviously, the order is reversed at the receiving IPsec peer.

ESP (Encapsulating Security Payload)

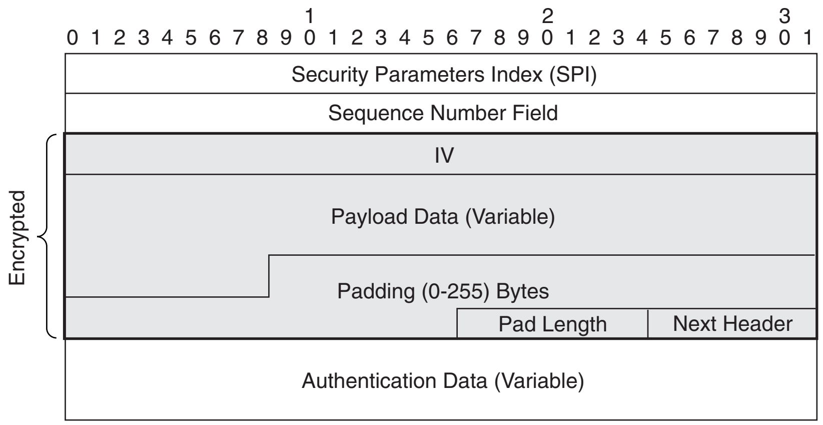
Encapsulating Security Payload (ESP) is the protocol that defines how the IPsec protocol encapsulates the original IP packet and how it provides encryption and authentication for the data therein. ESP also protects against anti-replay attacks.

ESP uses DES or 3DES to provide data confidentiality by encrypting the packet's contents. The keying material is derived through the IKE negotiation process discussed earlier in this chapter. Data integrity is provided by using MD5 or SHA hashes to calculate hashes on the data that is included in the packet such that in case the packet's contents are changed, the hash does not come out the same when recalculated by the receiver. The replay detection is achieved by using sequence numbers in the packets that ESP sends out and then implementing a sliding window on each IPsec peer that tracks which sequence numbers have been received. The RFC recommends a 32-packet window, but the PIX and the router implement a 32- and 64-packet window, respectively.

ESP is classified as IP protocol 50, and its format is defined in RFC 2406.

Figure 13-33 shows the format of the ESP header. Most of this header is sent in the clear to the IPsec peer because it contains information needed to decrypt the packet for the peer. However, a portion of the header containing the initialization vector is sent encrypted to the remote peer for security reasons.

Figure 13-33 *ESP Header Format*



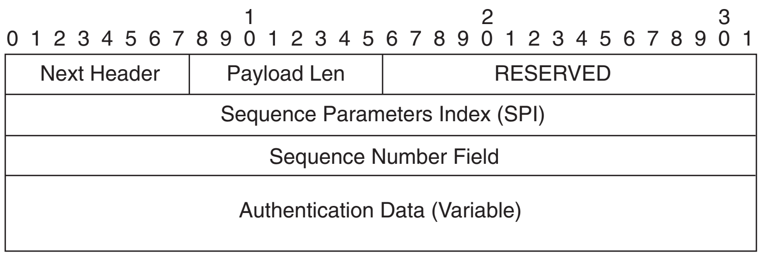
AH (Authentication Header)

AH is the protocol used within the IPsec suite that authenticates the data as well as the IP header. It is a form of encapsulation not used very often in IPsec deployments. AH provides no mechanisms for encrypting the data. All it provides is a hash that allows a check to be made on the data and the packet's IP header to ensure that the data was not tampered with in transit. Checking the IP header for integrity sets AH apart from ESP, which checks only the packet's data portion, not its IP header. However, as discussed earlier, this property of AH can cause problems in a NAT environment. AH, like ESP, also allows for anti-replay protection by using sequence numbers and sliding windows to keep track of packets received and packets that are yet to be sent. Replay detection works with each IPsec peer by keeping track of the sequence numbers of the packets it has received and advancing a window as it receives newer sequence numbers. It drops any packets that arrive outside this window or that are repeated.

AH is classified as IP protocol 51 and is defined in RFC 2402.

Figure 13-34 shows the format of the AH header.

Figure 13-34 *AH Header Format*



IKE Enhancements for Remote-Access Client IPsec

Remote Access IPsec provided through VPN clients connecting to IPsec gateways is a very important IPsec scenario. However, remote-access clients provide some particularly difficult challenges for IKE. We will list these challenges first and then talk about how IKE has been extended to meet these challenges:

- IPsec clients use unknown-to-gateway IP addresses to connect to the gateway, making it impossible to define unique address-based preshared keys. These are the IP addresses that have been assigned to these clients by their ISPs.
- For the IPsec clients to be treated as part of the private network to which they are connected, they must enter the private network with known IP addresses and not the IP addresses assigned to them by the ISP.

- The IPsec clients must use the DNS server, DHCP server, and other such servers provided on the private network as the primary sources of information rather than use the ISP's server, which does not have information regarding the internal resources.
- IPsec clients often connect from behind Port Address Translation (PAT) devices (such as DSL or cable routers doing PAT). Because ESP encrypts all the port information in the TCP or UDP header, PAT can no longer function as normal.

These four challenges have resulted in some extensions being implemented in the IKE protocol that allow the IPsec client implementations to overcome these challenges and provide maximum benefit to the users while ensuring security.

NOTE Currently, most of these techniques are still in the Internet draft stage at IETF and have not been converted into full-fledged RFCs.

Next we will discuss the four most important features that have been added to IKE to get around the issues just outlined.

Extended Authentication

Extended authentication allows the user of the IPsec client to authenticate itself, rather than the IPsec client software, to the IPsec gateway. This allows what is known as a *wildcard* preshared key to be used for authenticating all IPsec clients connecting to the IPsec gateway using the same preshared secret. The insecurity in allowing this to happen is overcome by doing one more stage of authentication—extended authentication. Extended authentication is done on a per-user basis, generally by the IPsec gateway in collaboration with a TACACS+ or RADIUS server.

Extended authentication (also called x-auth) occurs after IKE's phase 1 (the authentication phase) has been completed. This is why it is also said to occur in phase 1.5 of IKE.

Figure 13-35 shows the placement of x-auth and mode config in IKE negotiation.

X-auth takes place with the gateway sending the client an *attributes payload*. The attributes payload contains zero-length attributes for the attributes that the gateway wants the client to provide. An example is username(), password(). On receiving this message, the client either responds with a fail if it does not know how to provide these, which fails the negotiation, or responds by putting values into the attribute payload. An example is username(john), password(D04). After receiving the attributes, the gateway checks the credentials' validity and notifies the client of the success or failure. The client responds with an attributes payload of type config_ack. Quick mode is started at this point if the authentication was successful.

Figure 13-35 *Placement of X-Auth and Mode Config in IKE Negotiation*

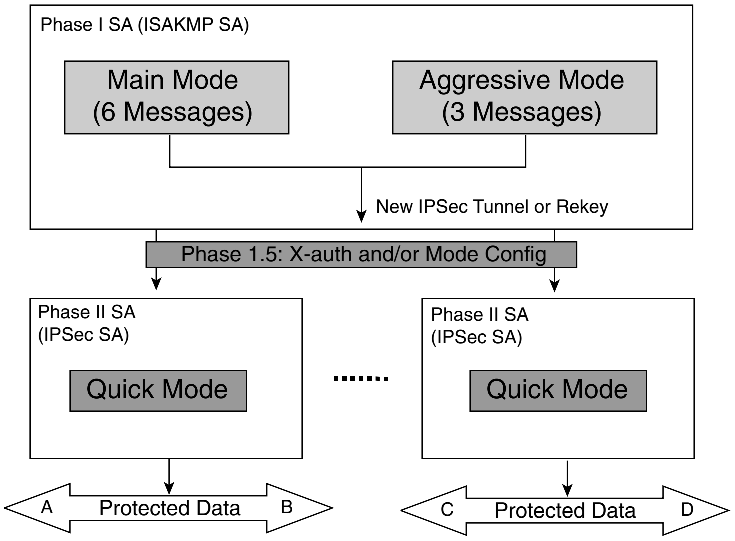


Figure 13-36 shows the negotiation of x-auth during IKE negotiation. Figures 13-37 and 13-38 show how attribute request and reply payloads are exchanged between the peers to complete x-auth.

Figure 13-36 *Negotiation of X-Auth During IKE Negotiation*

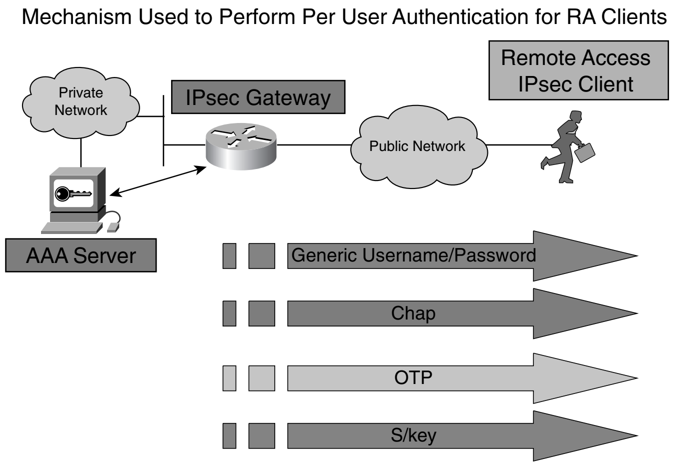

Mechanism Used to Perform Per User Authentication for RA Clients
Remote Access IPsec Client
Private Network
IPsec Gateway
Public Network
AAA Server
Generic Username/Password
Chap
OTP
S/key

Figure 13-37 *Start of Extended Authentication with the Exchange of Attribute Payloads Using ISAKMP Messages*

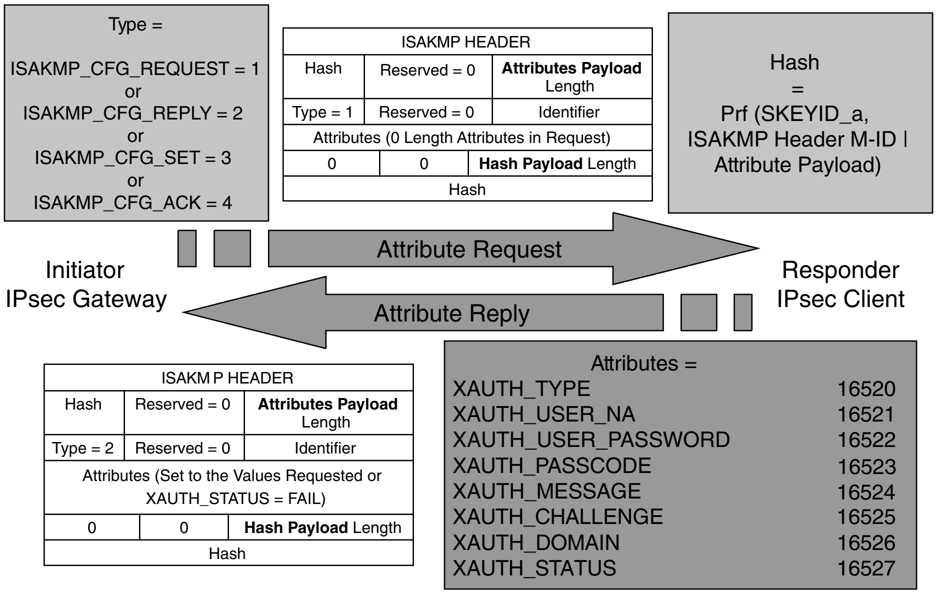
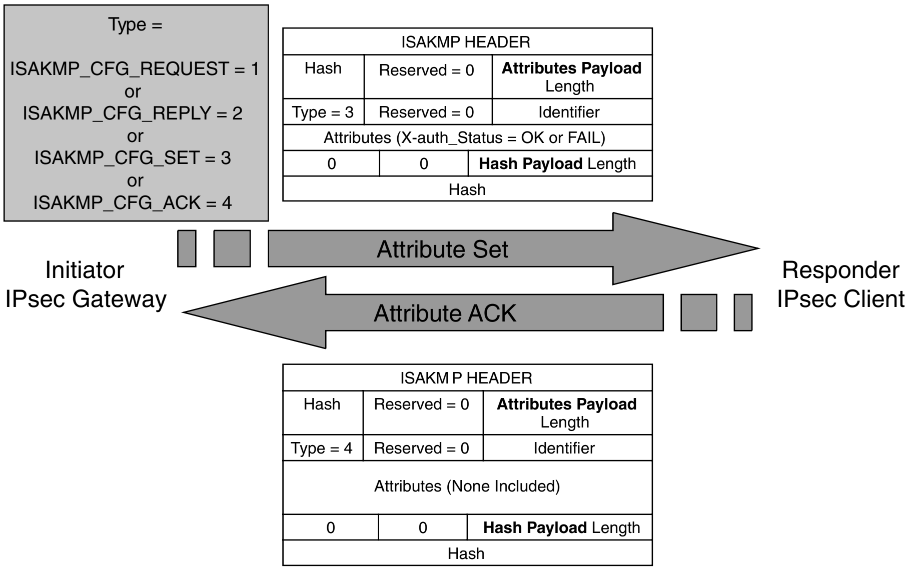


Figure 13-38 *Completion of Extended Authentication with the Exchange of Attribute Payloads Using ISAKMP Messages*



Mode Configuration

Mode configuration is a method employed to take care of the second and third challenges listed at the start of this section. It allows an IP address to be assigned to the client as well as DNS and other server IP addresses to be pushed to the client.

The IP address pushed to the client is called the *Internal IP address*. When this feature is not turned on, the IP header of the packet that is encrypted and encapsulated in the ESP payload contains the same source IP address as the outside IP header: the address assigned to the client by the ISP. However, with this feature turned on, the source address in the encapsulated packet is changed to this Internal IP address in all packets that are sent using ESP. This way, when the gateway decapsulates and decrypts the ESP packet, the packet that comes out has a predictable Internal IP address assigned to it.

In addition to the internal IP address, mode configuration pushes the other parameters, such as the DNS server IP address, the DHCP server IP address, and the NetBIOS name server IP address, to the client as well. The client installs these servers as its primary DNS and also installs other types of servers. This allows the client to resolve the names of the machines on the private network, among other things.

Mode config works using the same type of payload that x-auth uses. It occurs after x-auth has occurred, still in phase 1.5. However, unlike x-auth, in this case it is the client that requests the attributes from the gateway. (In some older versions of IOS, this was not necessarily the case, meaning that the client did not ask for the attributes but simply waited for them to be pushed.) The client may ask for whatever attributes it wants, and the gateway can reply with whatever attributes it deems appropriate to push and ignore others. In addition, the gateway can also push attributes the client didn't request.

Figure 13-39 shows the negotiation of mode configuration during IKE negotiation. Figure 13-40 shows the packet format of the attributes payload exchanged between the two peers for completing mode config.

NAT Transparency

NAT transparency is a mechanism introduced in IPsec to get around the problem of the encryption of TCP/UDP ports in ESP from stopping PAT from occurring. This problem is overcome by encapsulating the ESP packet in a UDP header with the necessary port information in it to allow the PAT to work correctly. Upon receiving these packets, the gateway strips the UDP header and processes the rest of the packet normally. IKE negotiation, which occurs using UDP port 500, does not have this problem; only ESP has this issue.

Figure 13-41 shows the tagging of a UDP header on top of the original ESP header.

Figure 13-39 *Negotiation of Mode Configuration During IKE Negotiation*

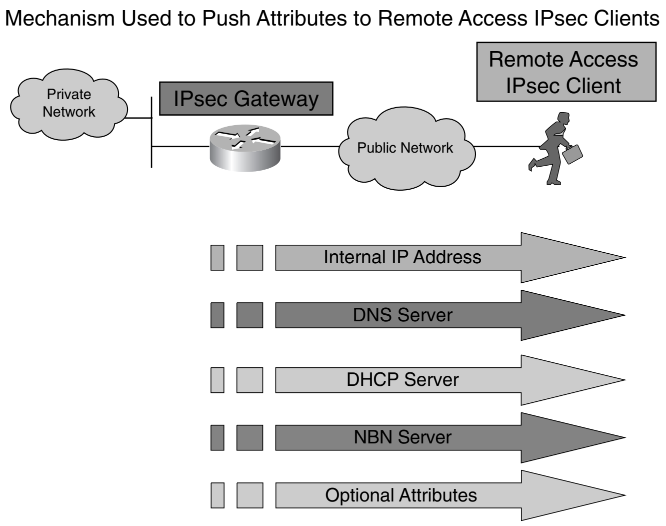


Figure 13-40 *Mode Config Is Completed by Exchanging Attribute Payloads Between the Gateway and the IPsec Client*

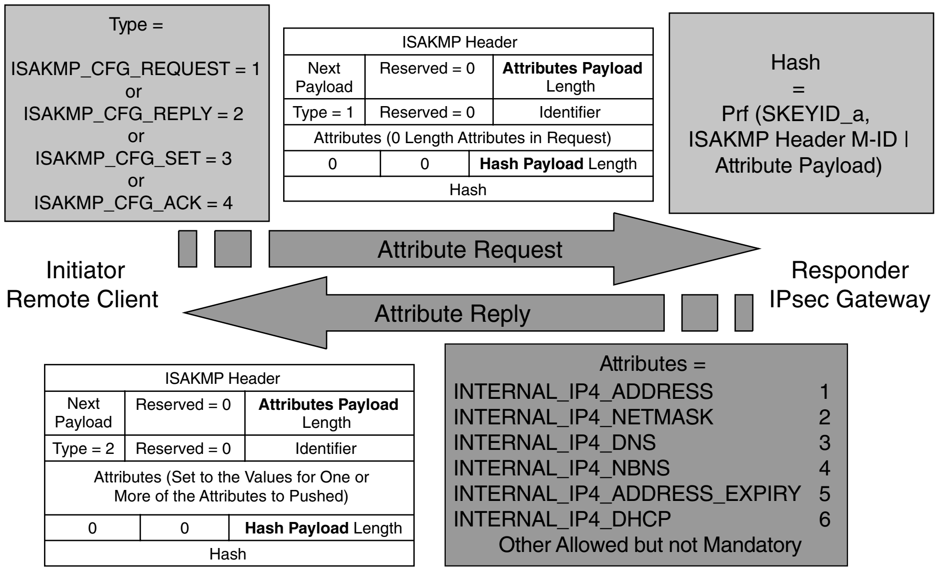
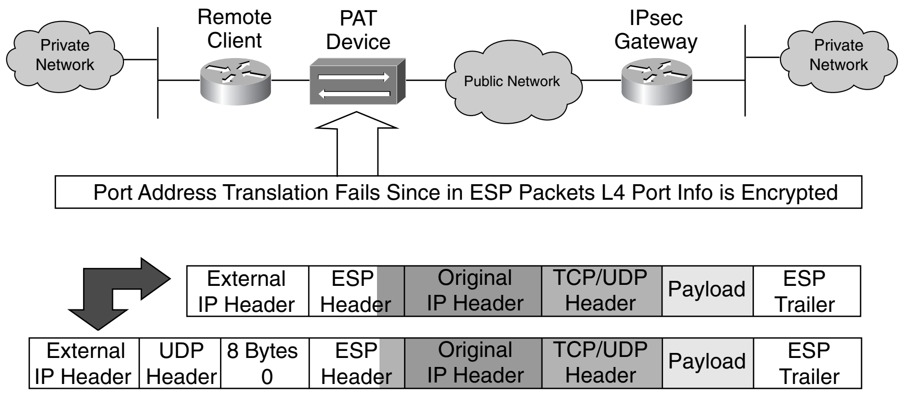


Figure 13-41 *Tagging on a UDP Header to Traverse PAT*



The UDP header is created based on a port number that is pushed from the gateway to the client upon being requested in the mode config request by the client. Currently, the PAT "detection" must be done by the person who is using the IPsec client. In other words, the person has to configure the client to let it know that it needs to ask the gateway for UDP header information. Various Internet drafts currently under consideration at IETF allow for automatic detection of PAT in the IPsec path using methods that depend on the same lines of reasoning due to which AH breaks in the presence of NAT—meaning using a hash to see if the packet's IP header gets modified in transit.

Current implementations of IOS and PIX IPsec do not support PAT traversal. However, support for this feature is expected soon.

IPsec Dead Peer Discovery Mechanism

IPsec provides a mechanism for a peer to send a delete notification payload via IKE to its peers when it is disconnecting an IPsec SA. However, in many cases this notification payload never gets sent, either because the peer gets disconnected too abruptly (a system crash) or due to network issues (someone pulls a laptop's Ethernet cable). In these cases, it is important to have a dead peer discovery mechanism that can allow for the discovery of such peers so that data loss does not occur when a peer sends packets to a peer that is no longer alive. (In effect, an IPsec peer can keep sending traffic to a dead peer for extended periods of time.) This mechanism is implemented using a technique called Dead Peer Discovery (DPD).

DPD works through the use of a notify payload, which is sent to the peer before new data is about to be sent after a period of inactivity longer than a configured "worry metric" has elapsed. Passage of IPsec traffic is considered proof of liveness. If it is alive, the peer acknowledges the notify payload by sending back one of it own.

Figure 13-42 shows the basic elements of the DPD protocol.

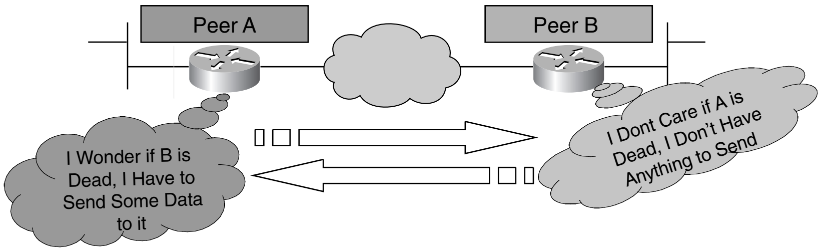

Peer A
Peer B
I Wonder if B is Dead, I Have to Send Some Data to it
I Dont Care if A is Dead, I Don't Have Anything to Send

Figure 13-42 *Basic Functionality of the DPD Protocol*

Figure 13-43 shows how the DPD protocol is used to gather information regarding a peer's liveness.

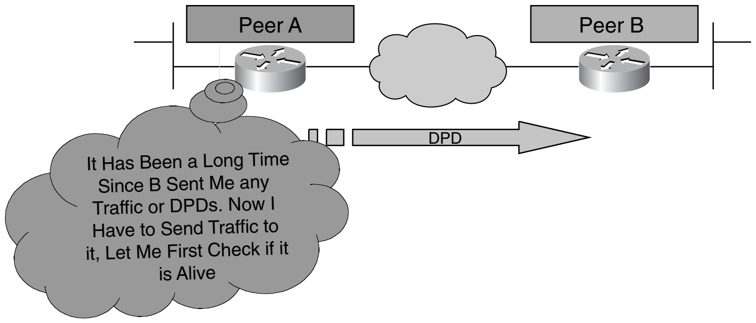
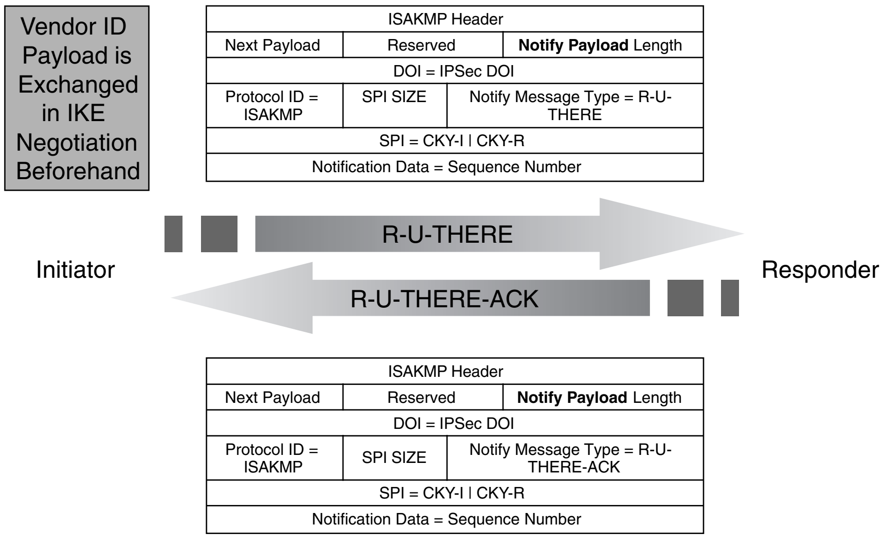

Peer A
Peer B
DPD
It Has Been a Long Time Since B Sent Me any Traffic or DPDs. Now I Have to Send Traffic to it, Let Me First Check if it is Alive

Figure 13-43 *DPD Protocol Operation*

For this mechanism to work, a vendor ID payload must be exchanged in the IKE main mode exchange, signifying that the two hosts support this type of mechanism. The worry metric is defined locally. This type of need-only-based mechanism greatly reduces the load on the peer by *not* using periodic keepalives which might or might not be needed. Also, the DPD mechanism allows the sending of R-U-THERE messages when a gateway wants to clean up its resources after a while and wants to know if some of the idle clients are still alive.

Figure 13-44 shows how the DPD notifications just discussed are exchanged.

Figure 13-44 *Exchange of DPD Notifications*



Case Studies

This section discusses a number of case studies showing the implementation of the concepts discussed in this chapter. Configurations, debugs, and **show** command output are included to help you understand the contents better.

It would be very educational for you to compare the debug output with the earlier step-by-step explanation of IKE to try to understand where each IKE negotiation step takes place.

Router-to-Router IPsec Using Preshared Keys as the Authentication Mechanism

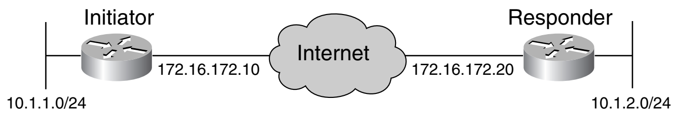
This is the most basic and common type of IPsec VPN. This VPN falls under the category of LAN-to-LAN VPNs. The authentication method used here is preshared keys. The following case studies have examples of more-secure authentication methods.

We will use this case study to do a detailed analysis of a basic configuration. Then, based on this analysis, we will talk about add-ons and changes in the rest of the case studies.

Examples 13-1 and 13-2 show configurations of routers acting as initiators and responders of IPsec negotiations respectively. Examples 13-3 and 13-4 show the **debug** output for these two routers. Examples 13-5 and 13-6 are used to discuss the output of the **show** commands for these two routers.

Figure 13-45 shows the network diagram for this case study.

Figure 13-45 *Network Diagram for This Case Study*



Example 13-1 *Configuration of the Router Acting as the Initiator of the IPsec Negotiation*

```
hostname Initiator
!The ISAKMP policy defines the attributes which will be negotiated with peers for
!the IKE SA.
crypto isakmp policy 1
!The encryption method defined below is used for encrypting the IKE negotiation
!packets using SKEYID_e
 encr 3des
!The hash algorithm defined below is used to generate the hashes which are used for
!IKE authetnication purposes.
 hash sha
!The line below defines the authentication method as pre-shared key authentication
 authentication pre-share
!
!
!The line below defines the pre-shared key for the peer at the IP address
!172.16.172.20. Please note that the initiator will search through its config for
!this key using the source IP address of the IKE negotiation packets it is receiving.
crypto isakmp key jw4ep9846804ijl address 172.16.172.20
!The following line defines the transform set for use with IPsec. This transform set
!specifies the name of the transform set as 'myset'. The encapsulation method defined
!is ESP and the encryption algorithm to use is 3DES  (triple DES). The last part of
!this command specifies MD5 as the ESP integrity checking hash.
crypto ipsec transform-set myset esp-3des esp-md5-hmac
!
!
!The following configuration is for the crypto map named 'vpn'. Crypto maps
!essentially bind the entire IPsec configuration together. Various elements of IPsec
!defined in various places in the configuration are tied together using the crypto
!map. 10 is the instance number for the map here. Instance numbers are used to
!specify the order in which multiple crypto maps are parsed in a config. The key
!word 'ipsec-isakmp' is used to specify that this particular crypto map is to be
!used for IPsec rather than CET.
crypto map vpn 10 ipsec-isakmp
!The command line below defines the IP address of the IPsec peer.
 set peer 172.16.172.20
!The line below defines the transform set to use for this crypto map.
 set transform-set myset
!The line below specifies the access list which will be define traffic which will
!either trigger IKE negotiation or be used to verify that the proxy Ids being offered
!during an IKE negotiation are valid.
 match address 101
```

Example 13-1 *Configuration of the Router Acting as the Initiator of the IPsec Negotiation (Continued)*

```
!
!
interface Ethernet0/0
 ip address 10.1.1.1 255.255.255.0
!

interface Ethernet1/0
 ip address 172.16.172.10 255.255.255.0
!The line below is used as a toggle switch to turn on IPsec functionality as defined
!by the crypto map vpn.
 crypto map vpn
!
!
!The access list define below is used to specify interesting traffic for IPsec.
access-list 101 permit ip 10.1.1.0 0.0.0.255 10.1.2.0 0.0.0.255
!
```

Example 13-2 *Configuration of the Router Acting as the Responder of the IPsec Negotiation*

```
hostname Responder

crypto isakmp policy 1
 encr 3des
 hash sha
 authentication pre-share
crypto isakmp key jw4ep9846804ijl address 172.16.172.10
!
crypto ipsec transform-set myset esp-3des esp-md5-hmac
!
crypto map vpn 10 ipsec-isakmp
 set peer 172.16.172.10
 set transform-set myset
 match address 101

interface Ethernet0/0
 ip address 10.1.2.1 255.255.255.0
!

interface Ethernet1/0
 ip address 172.16.172.20 255.255.255.0
 crypto map vpn
!
access-list 101 permit ip 10.1.2.0 0.0.0.255 10.1.1.0 0.0.0.255
```

NOTE In Example 13-3, the explanation of the debug is followed by the actual debugs for that explanation.

Example 13-3 *Debugs of the Router Acting as the Initiator of the IPsec Negotiation*

```
Initiator#show debug
Cryptographic Subsystem:
  Crypto ISAKMP debugging is on
  Crypto Engine debugging is on
  Crypto IPSEC debugging is on

A#ping
Protocol [ip]:
Target IP address: 10.1.2.1
Repeat count [5]:
Datagram size [100]:
Timeout in seconds [2]:
Extended commands [n]: y
Source address or interface: 10.1.1.1
Type of service [0]:
Set DF bit in IP header? [no]:
Validate reply data? [no]:
Data pattern [0xABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.1.2.1, timeout is 2 seconds:

!The ping source and destination addresses matched the match address access list
!for the crypto map VPN. 'local' is the local tunnel endpoint, and 'remote' is the
!remote crypto endpoint as configured in the map. src proxy is the src interesting
!traffic as defined by the match address access list. dst proxy is the destination
!interesting traffic as defined by the match address access list.

00:04:10: IPSEC(sa_request): ,
  (key eng. msg.) OUTBOUND local= 172.16.172.10, remote= 172.16.172.20,
    local_proxy= 10.1.1.0/255.255.255.0/0/0 (type=4),
    remote_proxy= 10.1.2.0/255.255.255.0/0/0 (type=4),

!The protocol and the transforms are specified by the crypto map that has been
!hit, as are the lifetimes

    protocol= ESP, transform= esp-3des esp-md5-hmac ,
    lifedur= 3600s and 4608000kb,
    spi= 0x8EAB0B22(2393574178), conn_id= 0, keysize= 0, flags= 0x400C

!Begins main mode exchange. The first two packets negotiate phase I SA parameters.
00:04:10: ISAKMP: received ke message (1/1)
00:04:10: ISAKMP: local port 500, remote port 500
00:04:10: ISAKMP (0:1): Input = IKE_MESG_FROM_IPSEC, IKE_SA_REQ_MM
```

Example 13-3 *Debugs of the Router Acting as the Initiator of the IPsec Negotiation (Continued)*

```
!MM stands for main mode, and QM stands for quick mode. The IKE debugs show which
!stage of IKE the negotiation is in, such as MM1. As you saw in the discussion of
!IKE, main mode is divided into six portions or messages, and quick mode into
!three.
Old State = IKE_READY  New State = IKE_I_MM1

00:04:10: ISAKMP (0:1): beginning Main Mode exchange
00:04:10: ISAKMP (0:1): sending packet to 172.16.172.20 (I) MM_NO_STATE
00:04:10: ISAKMP (0:1): received packet from 172.16.172.20 (I) MM_NO_STATE
00:04:10: ISAKMP (0:1): Input = IKE_MESG_FROM_PEER, IKE_MM_EXCH
Old State = IKE_I_MM1  New State = IKE_I_MM2

00:04:10: ISAKMP (0:1): processing SA payload. message ID = 0

!The preshared key is searched for and found based on the source IP address of IKE
!negotiation packets.

00:04:10: ISAKMP (0:1): found peer pre-shared key matching 172.16.172.20
00:04:10: ISAKMP (0:1): Checking ISAKMP transform 1 against priority 1 policy

!These are the parameters offered by the other side. Policy 1 is the policy set up
!on this router.

00:04:10: ISAKMP:      encryption 3DES-CBC
00:04:10: ISAKMP:      hash SHA
00:04:10: ISAKMP:      default group 1
00:04:10: ISAKMP:      auth pre-share
00:04:10: ISAKMP:      life type in seconds
00:04:10: ISAKMP:      life duration (VPI) of  0x0 0x1 0x51 0x80

!The policy 1 on this router and the attributes offered by the other side matched.

00:04:10: ISAKMP (0:1): atts are acceptable. Next payload is 0
00:04:10: ISAKMP (0:1): Input = IKE_MESG_INTERNAL, IKE_PROCESS_MAIN_MODE
Old State = IKE_I_MM2  New State = IKE_I_MM2

!The third and fourth packets complete Diffie-Hellman exchange.

00:04:10: ISAKMP (0:1): sending packet to 172.16.172.20 (I) MM_SA_SETUP
00:04:10: ISAKMP (0:1): Input = IKE_MESG_INTERNAL, IKE_PROCESS_COMPLETE
Old State = IKE_I_MM2  New State = IKE_I_MM3
00:04:10: ISAKMP (0:1): received packet from 172.16.172.20 (I) MM_SA_SETUP
00:04:10: ISAKMP (0:1): Input = IKE_MESG_FROM_PEER, IKE_MM_EXCH
Old State = IKE_I_MM3  New State = IKE I_MM4
00:04:10: ISAKMP (0:1): processing KE payload. message ID = 0
00:04:10: ISAKMP (0:1): processing NONCE payload. message ID = 0
00:04:10: ISAKMP (0:1): found peer pre-shared key matching 172.16.172.20
00:04:10: ISAKMP (0:1): SKEYID state generated
00:04:10: ISAKMP (0:1): processing vendor id payload
```

continues

Example 13-3 *Debugs of the Router Acting as the Initiator of the IPsec Negotiation (Continued)*

```
!Note below that some vendor ID payloads are being exchanged. These are necessary
!to gauge the peer's ability to do the things described in the vendor payload. For
!example, below, VID payloads for Unity Protocol Support (a new protocol
!introduced by Cisco for its newer version of VPN clients) and dead peer discovery
!are received.
00:04:10: ISAKMP (0:1): vendor ID is Unity
00:04:10: ISAKMP (0:1): processing vendor id payload
00:04:10: ISAKMP (0:1): vendor ID is DPD
00:04:10: ISAKMP (0:1): processing vendor id payload
00:04:10: ISAKMP (0:1): speaking to another IOS box!
00:04:10: ISAKMP (0:1): processing vendor id payload
00:04:10: ISAKMP (0:1): Input = IKE_MESG_INTERNAL, IKE_PROCESS_MAIN_MODE
Old State = IKE_I_MM4  New State = IKE_I_MM4

!The fifth and sixth packets complete IKE authentication. Phase I SA is
!established.

00:04:10: ISAKMP (0:1): SA is doing pre-shared key authentication using id type
ID_IPV4_ADDR
00:04:10: ISAKMP (1): ID payload
        next-payload : 8
        type         : 1
        protocol     : 17
        port         : 500
        length       : 8
00:04:10: ISAKMP (1): Total payload length: 12
00:04:10: ISAKMP (0:1): sending packet to 172.16.172.20 (I) MM_KEY_EXCH
00:04:10: ISAKMP (0:1): Input = IKE_MESG_INTERNAL, IKE_PROCESS_COMPLETE
Old State = IKE_I_MM4  New State = IKE_I_MM5

00:04:10: ISAKMP (0:1): received packet from 172.16.172.20 (I) MM_KEY_EXCH
00:04:10: ISAKMP (0:1): Input = IKE_MESG_FROM_PEER, IKE_MM_EXCH
Old State = IKE_I_MM5  New State = IKE_I_MM6

00:04:10: ISAKMP (0:1): processing ID payload. message ID = 0
00:04:10: ISAKMP (0:1): processing HASH payload. message ID = 0
00:04:10: ISAKMP (0:1): SA has been authenticated with 172.16.172.20
00:04:10: ISAKMP (0:1): Input = IKE_MESG_INTERNAL, IKE_PROCESS_MAIN_MODE
Old State = IKE_I_MM6  New State = IKE_I_MM6

00:04:10: ISAKMP (0:1): Input = IKE_MESG_INTERNAL, IKE_PROCESS_COMPLETE
Old State = IKE_I_MM6  New State = IKE_P1_COMPLETE

!Begin quick mode exchange. IPsec SA will be negotiated in quick mode.

00:04:10: ISAKMP (0:1): beginning Quick Mode exchange, M-ID of 965273472
00:04:10: ISAKMP (0:1): sending packet to 172.16.172.20 (I) QM_IDLE
00:04:10: ISAKMP (0:1): Node 965273472, Input = IKE_MESG_INTERNAL, IKE_INIT_QM
Old State = IKE_QM_READY  New State = IKE_QM_I_QM1

00:04:10: ISAKMP (0:1): Input = IKE_MESG_INTERNAL, IKE_PHASE1_COMPLETE
```

Example 13-3 *Debugs of the Router Acting as the Initiator of the IPsec Negotiation (Continued)*

```
Old State = IKE_P1_COMPLETE  New State = IKE_P1_COMPLETE

00:04:10: ISAKMP (0:1): received packet from 172.16.172.20 (I) QM_IDLE

!The IPsec SA proposal offered by the far end is checked against the local crypto
!map configuration


00:04:10: ISAKMP (0:1): processing HASH payload. message ID = 965273472
00:04:10: ISAKMP (0:1): processing SA payload. message ID = 965273472
00:04:10: ISAKMP (0:1): Checking IPsec proposal 1
00:04:10: ISAKMP: transform 1, ESP_3DES
00:04:10: ISAKMP:   attributes in transform:
00:04:10: ISAKMP:      encaps is 1
00:04:10: ISAKMP:      SA life type in seconds
00:04:10: ISAKMP:      SA life duration (basic) of 3600
00:04:10: ISAKMP:      SA life type in kilobytes
00:04:10: ISAKMP:      SA life duration (VPI) of  0x0 0x46 0x50 0x0
00:04:10: ISAKMP:      authenticator is HMAC-MD5

!The proposal 1 and transform 1 offered by the other end are found to be
!acceptable.

00:04:10: ISAKMP (0:1): atts are acceptable.
00:04:10: IPSEC(validate_proposal_request): proposal part #1,
  (key eng. msg.) INBOUND local= 172.16.172.10, remote= 172.16.172.20,
    local_proxy= 10.1.1.0/255.255.255.0/0/0 (type=4),
    remote_proxy= 10.1.2.0/255.255.255.0/0/0 (type=4),
    protocol= ESP, transform= esp-3des esp-md5-hmac ,
    lifedur= 0s and 0kb,
    spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x4
00:04:10: ISAKMP (0:1): processing NONCE payload. message ID = 965273472
00:04:10: ISAKMP (0:1): processing ID payload. message ID = 965273472
00:04:10: ISAKMP (0:1): processing ID payload. message ID = 965273472

!Two IPsec SAs have been negotiated--an incoming SA with the SPI generated by the
!local machine, and an outbound SA with the SPIs proposed by the remote end.


00:04:10: ISAKMP (0:1): Creating IPsec SAs
00:04:10:         inbound SA from 172.16.172.20 to 172.16.172.10
        (proxy 10.1.2.0 to 10.1.1.0)
00:04:10:         has spi 0x8EAB0B22 and conn_id 2029 and flags 4
00:04:10:         lifetime of 3600 seconds
00:04:10:         lifetime of 4608000 kilobytes
00:04:10:         outbound SA from 172.16.172.10   to 172.16.172.20
        (proxy 10.1.1.0        to 10.1.2.0       )
00:04:10:         has spi -343614331 and conn_id 2030 and flags C
00:04:10:         lifetime of 3600 seconds
00:04:10:         lifetime of 4608000 kilobytes
```

continues

Example 13-3 *Debugs of the Router Acting as the Initiator of the IPsec Negotiation (Continued)*

```
!The IPsec SA info negotiated by IKE is populated into the router's SADB.


00:04:10: IPSEC(key_engine): got a queue event...
00:04:10: IPSEC(initialize_sas): ,
  (key eng. msg.) INBOUND local= 172.16.172.10, remote= 172.16.172.20,
    local_proxy= 10.1.1.0/255.255.255.0/0/0 (type=4),
    remote_proxy= 10.1.2.0/255.255.255.0/0/0 (type=4),
    protocol= ESP, transform= esp-3des esp-md5-hmac ,
    lifedur= 3600s and 4608000kb,
    spi= 0x8EAB0B22(2393574178), conn_id= 2029, keysize= 0, flags= 0x4
00:04:10: IPSEC(initialize_sas): ,
  (key eng. msg.) OUTBOUND local= 172.16.172.10, remote= 172.16.172.20,
    local_proxy= 10.1.1.0/255.255.255.0/0/0 (type=4),
    remote_proxy= 10.1.2.0/255.255.255.0/0/0 (type=4),
    protocol= ESP, transform= esp-3des esp-md5-hmac ,
    lifedur= 3600s and 4608000kb,
    spi= 0xEB84DC85(3951352965), conn_id= 2030, keysize= 0, flags= 0xC

!IPsec SA created in SADB, sent out last packet with commit bit set. IPsec tunnel
!established.



00:04:10: IPSEC(create_sa): sa created,
  (sa) sa_dest= 172.16.172.10, sa_prot= 50,
    sa_spi= 0x8EAB0B22(2393574178),
    sa_trans= esp-3des esp-md5-hmac , sa_conn_id= 2029
00:04:10: IPSEC(create_sa): sa created,
  (sa) sa_dest= 172.16.172.20, sa_prot= 50,
    sa_spi= 0xEB84DC85(3951352965),
    sa_trans= esp-3des esp-md5-hmac , sa_conn_id= 2030
00:04:10: ISAKMP (0:1): sending packet to 172.16.172.20 (I) QM_IDLE
00:04:10: ISAKMP (0:1): deleting node 965273472 error FALSE reason ""
00:04:10: ISAKMP (0:1): Node 965273472, Input = IKE_MESG_FROM_PEER, IKE_QM_EXCH
Old State = IKE_QM_I_QM1  New State = IKE_QM_PHASE2_COMPLETE
```

Example 13-4 *Output of* **show** *Commands on the Router Acting as the Initiator of the IPsec Negotiation*

```
!The command below shows the state of the crypto ISAKMP SA. It is shown here in
!QM IDLE, meaning that quick mode has completed successfully.
Initiator#show crypto isakmp sa
dst             src             state           conn-id    slot
172.16.172.20   172.16.172.10   QM_IDLE               1       0

!The command below gives details on both the incoming and outgoing IPsec SAs. It
!gives information on the attributes negotiated during the exchange as well as
!statistics for how many packets have been exchanged via each of these SAs.
Initiator#show crypto ipsec sa
```

Example 13-4 *Output of* **show** *Commands on the Router Acting as the Initiator of the IPsec Negotiation (Continued)*

```
interface: Ethernet1/0
    Crypto map tag: vpn, local addr. 172.16.172.10

   local  ident (addr/mask/prot/port): (10.1.1.0/255.255.255.0/0/0)
   remote ident (addr/mask/prot/port): (10.1.2.0/255.255.255.0/0/0)
   current_peer: 172.16.172.20
     PERMIT, flags={origin_is_acl,}
    #pkts encaps: 4, #pkts encrypt: 4, #pkts digest 4
    #pkts decaps: 4, #pkts decrypt: 4, #pkts verify 4
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0, #pkts decompress failed: 0
    #send errors 6, #recv errors 0

     local crypto endpt.: 172.16.172.10, remote crypto endpt.: 172.16.172.20
     path mtu 1500, media mtu 1500
     current outbound spi: EB84DC85

     inbound esp sas:
      spi: 0x8EAB0B22(2393574178)
        transform: esp-3des esp-md5-hmac ,
        in use settings ={Tunnel, }
        slot: 0, conn id: 2029, flow_id: 1, crypto map: vpn
        sa timing: remaining key lifetime (k/sec): (4607998/3347)
        IV size: 8 bytes
        replay detection support: Y

     inbound ah sas:

     inbound pcp sas:

     outbound esp sas:
      spi: 0xEB84DC85(3951352965)
        transform: esp-3des esp-md5-hmac ,
        in use settings ={Tunnel, }
        slot: 0, conn id: 2030, flow_id: 2, crypto map: vpn
        sa timing: remaining key lifetime (k/sec): (4607999/3347)
        IV size: 8 bytes
        replay detection support: Y

     outbound ah sas:

     outbound pcp sas:

!The command below basically prints the configuration of the crypto map on the
!router

Initiator#show crypto map
Crypto Map "vpn" 10 ipsec-isakmp
        Peer = 172.16.172.20
```

continues

Example 13-4 *Output of* **show** *Commands on the Router Acting as the Initiator of the IPsec Negotiation (Continued)*

```
        Extended IP access list 101
            access-list 101 permit ip 10.1.1.0 0.0.0.255 10.1.2.0 0.0.0.255
        Current peer: 172.16.172.20
        Security association lifetime: 4608000 kilobytes/3600 seconds
        PFS (Y/N): N
        Transform sets={ myset, }
        Interfaces using crypto map vpn:
                Ethernet1/0
```

Example 13-5 *Debugs of the Router Acting as the Responder of the IPsec Negotiation*

```
Responder#show debug
Cryptographic Subsystem:
  Crypto ISAKMP debugging is on
  Crypto Engine debugging is on
  Crypto IPSEC debugging is on

1w1d: ISAKMP (0:0): received packet from 172.16.172.10 (N) NEW SA
1w1d: ISAKMP: local port 500, remote port 500
1w1d: ISAKMP (0:1): Input = IKE_MESG_FROM_PEER, IKE_MM_EXCH
Old State = IKE_READY  New State = IKE_R_MM1

1w1d: ISAKMP (0:1): processing SA payload. message ID = 0
1w1d: ISAKMP (0:1): found peer pre-shared key matching 172.16.172.10
1w1d: ISAKMP (0:1): Checking ISAKMP transform 1 against priority 1 policy
1w1d: ISAKMP:      encryption 3DES-CBC
1w1d: ISAKMP:      hash SHA
1w1d: ISAKMP:      default group 1
1w1d: ISAKMP:      auth pre-share
1w1d: ISAKMP:      life type in seconds
1w1d: ISAKMP:      life duration (VPI) of  0x0 0x1 0x51 0x80
1w1d: ISAKMP (0:1): atts are acceptable. Next payload is 0
1w1d: ISAKMP (0:1): Input = IKE_MESG_INTERNAL, IKE_PROCESS_MAIN_MODE
Old State = IKE_R_MM1  New State = IKE_R_MM1

1w1d: ISAKMP (0:1): sending packet to 172.16.172.10 (R) MM_SA_SETUP
1w1d: ISAKMP (0:1): Input = IKE_MESG_INTERNAL, IKE_PROCESS_COMPLETE
Old State = IKE_R_MM1  New State = IKE_R_MM2

1w1d: ISAKMP (0:1): received packet from 172.16.172.10 (R) MM_SA_SETUP
1w1d: ISAKMP (0:1): Input = IKE_MESG_FROM_PEER, IKE_MM_EXCH
Old State = IKE_R_MM2  New State = IKE_R_MM3

1w1d: ISAKMP (0:1): processing KE payload. message ID = 0
1w1d: ISAKMP (0:1): processing NONCE payload. message ID = 0
1w1d: ISAKMP (0:1): found peer pre-shared key matching 172.16.172.10
1w1d: ISAKMP (0:1): SKEYID state generated
1w1d: ISAKMP (0:1): processing vendor id payload
1w1d: ISAKMP (0:1): vendor ID is Unity
1w1d: ISAKMP (0:1): processing vendor id payload
```

Example 13-5 *Debugs of the Router Acting as the Responder of the IPsec Negotiation (Continued)*

```
1w1d: ISAKMP (0:1): vendor ID is DPD
1w1d: ISAKMP (0:1): processing vendor id payload
1w1d: ISAKMP (0:1): speaking to another IOS box!
1w1d: ISAKMP (0:1): processing vendor id payload
1w1d: ISAKMP (0:1): Input = IKE_MESG_INTERNAL, IKE_PROCESS_MAIN_MODE
Old State = IKE_R_MM3  New State = IKE_R_MM3

1w1d: ISAKMP (0:1): sending packet to 172.16.172.10 (R) MM_KEY_EXCH
1w1d: ISAKMP (0:1): Input = IKE_MESG_INTERNAL, IKE_PROCESS_COMPLETE
Old State = IKE_R_MM3  New State = IKE_R_MM4

1w1d: ISAKMP (0:1): received packet from 172.16.172.10 (R) MM_KEY_EXCH
1w1d: ISAKMP (0:1): Input = IKE_MESG_FROM_PEER, IKE_MM_EXCH
Old State = IKE_R_MM4  New State = IKE_R_MM5

1w1d: ISAKMP (0:1): processing ID payload. message ID = 0
1w1d: ISAKMP (0:1): processing HASH payload. message ID = 0
1w1d: ISAKMP (0:1): SA has been authenticated with 172.16.172.10
1w1d: ISAKMP (0:1): Input = IKE_MESG_INTERNAL, IKE_PROCESS_MAIN_MODE
Old State = IKE_R_MM5  New State = IKE_R_MM5

1w1d: ISAKMP (0:1): SA is doing pre-shared key authentication using id type ID
 _IPV4_ADDR
1w1d: ISAKMP (1): ID payload
        next-payload : 8
        type         : 1
        protocol     : 17
        port         : 500
        length       : 8
1w1d: ISAKMP (1): Total payload length: 12
1w1d: ISAKMP (0:1): sending packet to 172.16.172.10 (R) QM_IDLE
1w1d: ISAKMP (0:1): Input = IKE_MESG_INTERNAL, IKE_PROCESS_COMPLETE
Old State = IKE_R_MM5  New State = IKE_P1_COMPLETE

1w1d: ISAKMP (0:1): Input = IKE_MESG_INTERNAL, IKE_PHASE1_COMPLETE
Old State = IKE_P1_COMPLETE  New State = IKE_P1_COMPLETE

1w1d: ISAKMP (0:1): received packet from 172.16.172.10 (R) QM_IDLE
1w1d: ISAKMP (0:1): processing HASH payload. message ID = 965273472
1w1d: ISAKMP (0:1): processing SA payload. message ID = 965273472
1w1d: ISAKMP (0:1): Checking IPsec proposal 1
1w1d: ISAKMP: transform 1, ESP_3DES
1w1d: ISAKMP:   attributes in transform:
1w1d: ISAKMP:      encaps is 1
1w1d: ISAKMP:      SA life type in seconds
1w1d: ISAKMP:      SA life duration (basic) of 3600
1w1d: ISAKMP:      SA life type in kilobytes
1w1d: ISAKMP:      SA life duration (VPI) of  0x0 0x46 0x50 0x0
1w1d: ISAKMP:      authenticator is HMAC-MD5
1w1d: ISAKMP (0:1): atts are acceptable.
```

continues

Example 13-5 *Debugs of the Router Acting as the Responder of the IPsec Negotiation (Continued)*

```
1w1d: IPSEC(validate_proposal_request): proposal part #1,
  (key eng. msg.) INBOUND local= 172.16.172.20, remote= 172.16.172.10,
    local_proxy= 10.1.2.0/255.255.255.0/0/0 (type=4),
    remote_proxy= 10.1.1.0/255.255.255.0/0/0 (type=4),
    protocol= ESP, transform= esp-3des esp-md5-hmac ,
    lifedur= 0s and 0kb,
    spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x4
1w1d: ISAKMP (0:1): processing NONCE payload. message ID = 965273472
1w1d: ISAKMP (0:1): processing ID payload. message ID = 965273472
1w1d: ISAKMP (0:1): processing ID payload. message ID = 965273472
1w1d: ISAKMP (0:1): asking for 1 spis from ipsec
1w1d: ISAKMP (0:1): Node 965273472, Input = IKE_MESG_FROM_PEER, IKE_QM_EXCH
Old State = IKE_QM_READY  New State = IKE_QM_SPI_STARVE

1w1d: IPSEC(key_engine): got a queue event...
1w1d: IPSEC(spi_response): getting spi 3951352965 for SA
        from 172.16.172.20   to 172.16.172.10   for prot 3
1w1d: ISAKMP: received ke message (2/1)
1w1d: ISAKMP (0:1): sending packet to 172.16.172.10 (R) QM_IDLE
1w1d: ISAKMP (0:1): Node 965273472, Input = IKE_MESG_FROM_IPSEC, IKE_SPI_REPLY
Old State = IKE_QM_SPI_STARVE  New State = IKE_QM_R_QM2

1w1d: ISAKMP (0:1): received packet from 172.16.172.10 (R) QM_IDLE
1w1d: ISAKMP (0:1): Creating IPsec SAs
1w1d:         inbound SA from 172.16.172.10 to 172.16.172.20
        (proxy 10.1.1.0 to 10.1.2.0)
1w1d:         has spi 0xEB84DC85 and conn_id 2029 and flags 4
1w1d:         lifetime of 3600 seconds
1w1d:         lifetime of 4608000 kilobytes
1w1d:       outbound SA from 172.16.172.20   to 172.16.172.10   (proxy 10.1.2.0
to 10.1.1.0       )
1w1d:         has spi -1901393118 and conn_id 2030 and flags C
1w1d:         lifetime of 3600 seconds
1w1d:         lifetime of 4608000 kilobytes
1w1d: ISAKMP (0:1): deleting node 965273472 error FALSE reason "quick mode done
  (await()"
1w1d: ISAKMP (0:1): Node 965273472, Input = IKE_MESG_FROM_PEER, IKE_QM_EXCH
Old State = IKE_QM_R_QM2  New State = IKE_QM_PHASE2_COMPLETE

1w1d: IPSEC(key_engine): got a queue event...
1w1d: IPSEC(initialize_sas): ,
  (key eng. msg.) INBOUND local= 172.16.172.20, remote= 172.16.172.10,
    local_proxy= 10.1.2.0/255.255.255.0/0/0 (type=4),
    remote_proxy= 10.1.1.0/255.255.255.0/0/0 (type=4),
    protocol= ESP, transform= esp-3des esp-md5-hmac ,
    lifedur= 3600s and 4608000kb,
    spi= 0xEB84DC85(3951352965), conn_id= 2029, keysize= 0, flags= 0x4
1w1d: IPSEC(initialize_sas): ,
  (key eng. msg.) OUTBOUND local= 172.16.172.20, remote= 172.16.172.10,
    local_proxy= 10.1.2.0/255.255.255.0/0/0 (type=4),
    remote_proxy= 10.1.1.0/255.255.255.0/0/0 (type=4),
    protocol= ESP, transform= esp-3des esp-md5-hmac ,
```

Example 13-5 *Debugs of the Router Acting as the Responder of the IPsec Negotiation (Continued)*

```
    lifedur= 3600s and 4608000kb,
    spi= 0x8EAB0B22(2393574178), conn_id= 2030, keysize= 0, flags= 0xC
1w1d: IPSEC(create_sa): sa created,
  (sa) sa_dest= 172.16.172.20, sa_prot= 50,
    sa_spi= 0xEB84DC85(3951352965),
    sa_trans= esp-3des esp-md5-hmac , sa_conn_id= 2029
1w1d: IPSEC(create_sa): sa created,
  (sa) sa_dest= 172.16.172.10, sa_prot= 50,
    sa_spi= 0x8EAB0B22(2393574178),
    sa_trans= esp-3des esp-md5-hmac , sa_conn_id= 2030
1w1d: ISAKMP (0:1): purging node 965273472
```

Example 13-6 *Output of* **show** *Commands on the Router Acting as the Responder of the IPsec Negotiation*

```
Responder#show cry isa sa
dst             src             state           conn-id    slot
172.16.172.10   172.16.172.20   QM_IDLE               1       0

Responder#show crypto ipsec sa

interface: Ethernet1/0
    Crypto map tag: vpn, local addr. 172.16.172.20

   local  ident (addr/mask/prot/port): (10.1.2.0/255.255.255.0/0/0)
   remote ident (addr/mask/prot/port): (10.1.1.0/255.255.255.0/0/0)
   current_peer: 172.16.172.10
     PERMIT, flags={origin_is_acl,}
    #pkts encaps: 4, #pkts encrypt: 4, #pkts digest 4
    #pkts decaps: 4, #pkts decrypt: 4, #pkts verify 4
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0, #pkts decompress failed: 0
    #send errors 0, #recv errors 0

     local crypto endpt.: 172.16.172.20, remote crypto endpt.: 172.16.172.10
     path mtu 1500, media mtu 1500
     current outbound spi: 8EAB0B22

     inbound esp sas:
      spi: 0xEB84DC85(3951352965)
        transform: esp-3des esp-md5-hmac ,
        in use settings ={Tunnel, }
        slot: 0, conn id: 2029, flow_id: 1, crypto map: vpn
        sa timing: remaining key lifetime (k/sec): (4607998/3326)
        IV size: 8 bytes
        replay detection support: Y

     inbound ah sas:

     inbound pcp sas:
```

continues

Example 13-6 *Output of* **show** *Commands on the Router Acting as the Responder of the IPsec Negotiation*

```
     outbound esp sas:
      spi: 0x8EAB0B22(2393574178)
        transform: esp-3des esp-md5-hmac ,
        in use settings ={Tunnel, }
        slot: 0, conn id: 2030, flow_id: 2, crypto map: vpn
        sa timing: remaining key lifetime (k/sec): (4607999/3326)
        IV size: 8 bytes
        replay detection support: Y

     outbound ah sas:

     outbound pcp sas:

Initiator#show cry map
Crypto Map "vpn" 10 ipsec-isakmp
        Peer = 172.16.172.10
        Extended IP access list 101
            access-list 101 permit ip 10.1.2.0 0.0.0.255 10.1.1.0 0.0.0.255
        Current peer: 172.16.172.10
        Security association lifetime: 4608000 kilobytes/3600 seconds
        PFS (Y/N): N
        Transform sets={ myset, }
        Interfaces using crypto map vpn:
                Ethernet1/0
```

Router-to-Router IPsec Using Digital Signatures with Digital Certificates

This case study uses digital signatures instead of preshared keys, making it a more secure and scalable VPN setup. Various CA servers work with Cisco routers. Among them are CA servers from vendors, Microsoft, VeriSign, and Baltimore.

A router administrator needs to go through the following steps to obtain a certificate for the router:

Step 1 Generate a public/private key pair.

(a) Generate a general-purpose key pair.

This pair is generated using this command:

crypto key generate rsa

In this case, the pair of public and private keys is used for both the signature mode of authentication and the encrypted nonce form of authentication.

(b) Generate special-purpose key pairs.

This pair is generated using this command:

crypto key generate rsa usage-keys

In this case, two pairs of public and private keys are generated—one to be used for encrypted nonce authentication (if used), and one for the signature mode of authentication (if configured). Generating two pairs of keys reduces the keys' exposure during negotiation, thus reducing the risk of their compromise.

Step 2 Enroll with a certificate authority server.

To properly authenticate each other, both IPsec peers must enroll to obtain digital certificates. Both the IPsec peers must obtain certificates from the same CA server (or different CA servers belonging to the same root CA). They must also belong to the same organization. Often this organization name is specified as a domain name that both routers agree on.

Upon enrollment, each router gets one certificate for each of its public keys. If a single general-purpose key was generated, only one certificate is issued. If usage keys are configured, two certificates are obtained—one for each public key.

In addition to the certificates for the router's public key, a certificate for the CA itself is also downloaded to the router.

If a registration authority (RA) is used, two additional RA certificates are downloaded on the router.

NOTE The CA server administrator must ensure that a certificate is issued only to routers and devices that genuinely belong to an organization or domain. As discussed earlier, this ensures that only devices that are meant to administratively talk to each other can do so. Routers reject certificates that do not belong to the domain or organization they are allowed to interoperate with.

Cisco routers have an automatic mechanism for enrolling themselves in CA servers—Simple Certificate Enrollment Protocol (SCEP). SCEP allows the router to be enrolled in a CA using the following simple commands:

```
test1-isdn(config)#crypto ca identity testCA
test1-isdn(ca-identity)#enrollment url http://cisco-CA
```

```
test1-isdn(ca-identity)#exit
test1-isdn(config)#crypto ca authenticate testCA
test1-isdn(config)#crypto ca enroll testCA
```

The **crypto ca identity** command defines a tag to be used to define the CA server configuration on the router. **enrollment url** specifies the CA server's URL that is accessed using SCEP to obtain the certificate. The command **crypto ca authenticate** is used to obtain the CA server's certificate, and the command **crypto ca enroll** is used to obtain certificates for each public key on the router.

Some CAs have an RA as part of their implementation. An RA is essentially a server that acts as a proxy for the CA so that CA functions can continue when the CA is offline. In this case, the CA signs all the certificates, and a separate RA acts as the PKI server for the enrollment transactions.

If an RA is in use, an additional command is configured in crypto ca-identity CLI mode:

```
test1-isdn(ca-identity)#enrollment mode ra
```

CA servers can also export the certificates to an LDAP server for distribution. If that is the case, the URL for the LDAP server must also be specified using the following command:

```
test1-isdn(ca-identity)#query url ldap://server-ldap
```

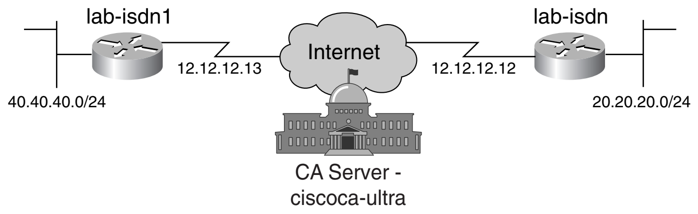
Another important thing to note is that when certificates are used during negotiation, the certificate's distinguished name is used as the identity rather than the router's configured ISAKMP identity.

In the following configuration examples, it is interesting to note that no ISAKMP policy is configured. In this situation, the router uses the default ISAKMP policy set up in its internal code. The default ISAKMP policy, which is numbered 65535, is as follows:

Encryption algorithm—DES
Hash algorithm—Secure Hash Algorithm (SHA)
Authentication method—Rivest Shamir Adleman (RSA) signature
Diffie-Hellman group—#1 (768-bit)
Lifetime—86400 seconds, no volume limit

Figure 13-46 shows the network diagram for this case study. Examples 13-7 and 13-8 show the configuration on the two routers used in this case study while Example 13-9 shows the debug output on the lab-isdn router.

Figure 13-46 *Network Diagram for This Case Study*



Example 13-7 *Configuration of the Router lab-isdn1*

```
lab-isdn1#wr t
Building configuration...

Current configuration:
!
version 11.3
service timestamps debug datetime msec
!
hostname lab-isdn1
!
enable secret 5 <removed>
!
username lab-isdn password 0 cisco
ip host ciscoca-ultra 171.69.54.46
ip host lab-isdn 12.12.12.12
ip domain-name cisco.com
ip name-server 171.68.10.70
ip name-server 171.68.122.99
isdn switch-type basic-ni1
!
crypto ipsec transform-set mypolicy ah-sha-hmac esp-des esp-sha-hmac
!
crypto map test 10 ipsec-isakmp
 set peer 12.12.12.12
 set transform-set mypolicy
 match address 144
!
!The configuration below defines the CA server to be used by the router. The
!configuration also contains the certificates downloaded by the router from the CA
!server.

crypto ca identity bubba
 enrollment url http://ciscoca-ultra

crypto ca certificate chain bubba
!Below is the router's certificate for the signature public key.
 certificate 3E1ED472BDA2CE0163FB6B0B004E5EEE
  308201BC 30820166 A0030201 0202103E 1ED472BD A2CE0163 FB6B0B00 4E5EEE30
  0D06092A 864886F7 0D010104 05003042 31163014 06035504 0A130D43 6973636F
  20537973 74656D73 3110300E 06035504 0B130744 65767465 73743116 30140603
  55040313 0D434953 434F4341 2D554C54 5241301E 170D3938 30343038 30303030
  30305A17 0D393930 34303832 33353935 395A303B 31273025 06092A86 4886F70D
  01090216 18737461 6E6E6F75 732D6973 646E312E 63697363 6F2E636F 6D311030
  0E060355 04051307 35363739 39383730 5C300D06 092A8648 86F70D01 01010500
  034B0030 48024100 D2D125FF BBFC6E56 93CB4385 5473C165 BC7CCAF6 45C35BED
  554BAA0B 119AFA6F 0853F574 5E0B8492 2E39B5FA 84C4DD05 C19AA625 8184395C
  6CBC7FA4 614F6177 02030100 01A33F30 3D300B06 03551D0F 04040302 05203023
  0603551D 11041C30 1A821873 74616E6E 6F75732D 6973646E 312E6369 73636F2E
  636F6D30 09060355 1D130402 3000300D 06092A86 4886F70D 01010405 00034100
  04AF83B8 FE95F5D9 9C07C105 F1E88F1A 9320CE7D 0FA540CF 44C77829 FC85C94B
```

continues

Example 13-7 *Configuration of the Router lab-isdn1 (Continued)*

```
  8CB4CA32 85FF9655 8E47AC9A B9D6BF1A 0C4846DE 5CB07C8E A32038EC 8AFD161A
  quit
!Below is the CA's certificate for the CA server's public key
 certificate ca 3051DF7169BEE31B821DFE4B3A338E5F
  30820182 3082012C A0030201 02021030 51DF7169 BEE31B82 1DFE4B3A 338E5F30
  0D06092A 864886F7 0D010104 05003042 31163014 06035504 0A130D43 6973636F
  20537973 74656D73 3110300E 06035504 0B130744 65767465 73743116 30140603
  55040313 0D434953 434F4341 2D554C54 5241301E 170D3937 31323032 30313036
  32385A17 0D393831 32303230 31303632 385A3042 31163014 06035504 0A130D43
  6973636F 20537973 74656D73 3110300E 06035504 0B130744 65767465 73743116
  30140603 55040313 0D434953 434F4341 2D554C54 5241305C 300D0609 2A864886
  F70D0101 01050003 4B003048 024100C1 B69D7BF6 34E4EE28 A84E0DC6 FCA4DEA8
  04D89E50 C5EBE862 39D51890 D0D4B732 678BDBF2 80801430 E5E56E7C C126E2DD
  DBE9695A DF8E5BA7 E67BAE87 29375302 03010001 300D0609 2A864886 F70D0101
  04050003 410035AA 82B5A406 32489413 A7FF9A9A E349E5B4 74615E05 058BA3CE
  7C5F00B4 019552A5 E892D2A3 86763A1F 2852297F C68EECE1 F41E9A7B 2F38D02A
  B1D2F817 3F7B
  Quit
!Below is the router's certificate for the encryption public key.
 certificate 503968D890F7D409475B7280162754D2
  308201BC 30820166 A0030201 02021050 3968D890 F7D40947 5B728016 2754D230
  0D06092A 864886F7 0D010104 05003042 31163014 06035504 0A130D43 6973636F
  20537973 74656D73 3110300E 06035504 0B130744 65767465 73743116 30140603
  55040313 0D434953 434F4341 2D554C54 5241301E 170D3938 30343038 30303030
  30305A17 0D393930 34303832 33353935 395A303B 31273025 06092A86 4886F70D
  01090216 18737461 6E6E6F75 732D6973 646E312E 63697363 6F2E636F 6D311030
  0E060355 04051307 35363739 39383730 5C300D06 092A8648 86F70D01 01010500
  034B0030 48024100 BECE2D8C B32E6B09 0ADE0D46 AF8D4A1F 37850034 35D0C729
  3BF91518 0C9E4CF8 1A6A43AE E4F04687 B8E2859D 33D5CE04 2E5DDEA6 3DA54A31
  2AD4255A 756014CB 02030100 01A33F30 3D300B06 03551D0F 04040302 07803023
  0603551D 11041C30 1A821873 74616E6E 6F75732D 6973646E 312E6369 73636F2E
  636F6D30 09060355 1D130402 3000300D 06092A86 4886F70D 01010405 00034100
  B3AF6E71 CBD9AEDD A4711B71 6897F2CE D669A23A EE47B92B B2BE942A 422DF4A5
  7ACB9433 BD17EC7A BB3721EC E7D1175F 5C62BC58 C409F805 19691FBD FD925138
  quit
!
interface Ethernet0
 ip address 40.40.40.40 255.255.255.0
 no ip mroute-cache
!
interface BRI0
 ip address 12.12.12.13 255.255.255.0
 encapsulation ppp
 no ip mroute-cache
 dialer idle-timeout 99999
 dialer map ip 12.12.12.12 name lab-isdn 4724171
 dialer hold-queue 40
 dialer-group 1
 isdn spid1 919472411800 4724118
 isdn spid2 919472411901 4724119
 ppp authentication chap
 crypto map test
```

Example 13-7 *Configuration of the Router lab-isdn1 (Continued)*

```
!
ip classless
ip route 0.0.0.0 0.0.0.0 12.12.12.12
access-list 144 permit ip 40.40.40.0 0.0.0.255 20.20.20.0 0.0.0.255
dialer-list 1 protocol ip permit
!
line con 0
 exec-timeout 0 0
line vty 0 4
 password ww
 login
!
end
```

Example 13-8 *Configuration of the Router lab-isdn*

```
lab-isdn#write terminal
Building configuration...

Current configuration:
!
version 11.3
service timestamps debug datetime msec

!
hostname lab-isdn
!
enable secret 5 <removed>
!
username lab-isdn1 password 0 cisco
ip host ciscoca-ultra 171.69.54.46
ip host lab-isdn1 12.12.12.13
ip domain-name cisco.com
ip name-server 171.68.10.70
ip name-server 171.68.122.99
isdn switch-type basic-ni1
!
crypto ipsec transform-set mypolicy ah-sha-hmac esp-des esp-sha-hmac
!
crypto map test 10 ipsec-isakmp
 set peer 12.12.12.13
 set transform-set mypolicy
 match address 133
!
crypto ca identity lab
 enrollment url http://ciscoca-ultra

crypto ca certificate chain lab
```

continues

Example 13-8 *Configuration of the Router lab-isdn (Continued)*

```
!Below is the router's certificate for the signature public key.
 certificate 44FC6C531FC3446927E4EE307A806B20

  308201E0 3082018A A0030201 02021044 FC6C531F C3446927 E4EE307A 806B2030
  0D06092A 864886F7 0D010104 05003042 31163014 06035504 0A130D43 6973636F
  20537973 74656D73 3110300E 06035504 0B130744 65767465 73743116 30140603
  55040313 0D434953 434F4341 2D554C54 5241301E 170D3938 30343038 30303030
  30305A17 0D393930 34303832 33353935 395A305A 31263024 06092A86 4886F70D
  01090216 17737461 6E6E6F75 732D6973 646E2E63 6973636F 2E636F6D 311E301C
  060A2B06 0104012A 020B0201 130E3137 312E3638 2E313137 2E313839 3110300E
  06035504 05130735 36373939 3139305C 300D0609 2A864886 F70D0101 01050003
  4B003048 024100B8 F4A17A70 FAB5C2E3 39186513 486779C7 61EF0AC1 3B6CFF83
  810E6D28 B3E4C034 CD803CFF 5158C270 28FEBCDE CB6EF2D4 83BDD9B3 EAF915DB
  78266E96 500CD702 03010001 A3443042 300B0603 551D0F04 04030205 20302806
  03551D11 0421301F 82177374 616E6E6F 75732D69 73646E2E 63697363 6F2E636F
  6D8704AB 4475BD30 09060355 1D130402 3000300D 06092A86 4886F70D 01010405
  00034100 BF65B931 0F960195 ABDD41D5 622743D9 C12B5499 B3A8EB30 5005E6CC
  7FDF7C5B 51D13EB8 D46187E5 A1E7F711 AEB7B33B AA4C6728 7A4BA692 00A44A05 C5CF973F
  quit
!Below is the CA's certificate for the CA server's public key
 certificate ca 3051DF7169BEE31B821DFE4B3A338E5F
  30820182 3082012C A0030201 02021030 51DF7169 BEE31B82 1DFE4B3A 338E5F30
  0D06092A 864886F7 0D010104 05003042 31163014 06035504 0A130D43 6973636F
  20537973 74656D73 3110300E 06035504 0B130744 65767465 73743116 30140603
  55040313 0D434953 434F4341 2D554C54 5241301E 170D3937 31323032 30313036
  32385A17 0D393831 32303230 31303632 385A3042 31163014 06035504 0A130D43
  6973636F 20537973 74656D73 3110300E 06035504 0B130744 65767465 73743116
  30140603 55040313 0D434953 434F4341 2D554C54 5241305C 300D0609 2A864886
  F70D0101 01050003 4B003048 024100C1 B69D7BF6 34E4EE28 A84E0DC6 FCA4DEA8
  04D89E50 C5EBE862 39D51890 D0D4B732 678BDBF2 80801430 E5E56E7C C126E2DD
  DBE9695A DF8E5BA7 E67BAE87 29375302 03010001 300D0609 2A864886 F70D0101
  04050003 410035AA 82B5A406 32489413 A7FF9A9A E349E5B4 74615E05 058BA3CE
  7C5F00B4 019552A5 E892D2A3 86763A1F 2852297F C68EECE1 F41E9A7B 2F38D02A
  B1D2F817 3F7B
  Quit
!Below is the router's certificate for the encryption public key.
 certificate 52A46D5D10B18A6F51E6BC735A36508C
  308201E0 3082018A A0030201 02021052 A46D5D10 B18A6F51 E6BC735A 36508C30
  0D06092A 864886F7 0D010104 05003042 31163014 06035504 0A130D43 6973636F
  20537973 74656D73 3110300E 06035504 0B130744 65767465 73743116 30140603
  55040313 0D434953 434F4341 2D554C54 5241301E 170D3938 30343038 30303030
  30305A17 0D393930 34303832 33353935 395A305A 31263024 06092A86 4886F70D
  01090216 17737461 6E6E6F75 732D6973 646E2E63 6973636F 2E636F6D 311E301C
  060A2B06 0104012A 020B0201 130E3137 312E3638 2E313137 2E313839 3110300E
  06035504 05130735 36373939 3139305C 300D0609 2A864886 F70D0101 01050003
  4B003048 024100D7 71AD5672 B487A019 5ECD1954 6F919A3A 6270102E 5A9FF4DC
  7A608480 FB27A181 715335F4 399D3E57 7F72B323 BF0620AB 60C371CF 4389BA4F
  C60EE6EA 21E06302 03010001 A3443042 300B0603 551D0F04 04030207 80302806
  03551D11 0421301F 82177374 616E6E6F 75732D69 73646E2E 63697363 6F2E636F
  6D8704AB 4475BD30 09060355 1D130402 3000300D 06092A86 4886F70D 01010405
  00034100 8AD45375 54803CF3 013829A8 8DB225A8 25342160 94546F3C 4094BBA3
```

Example 13-8 *Configuration of the Router lab-isdn (Continued)*

```
  F2F5A378 97E2F06F DCFFC509 A07B930A FBE6C3CA E1FC7FD9 1E69B872 C402E62A A8814C09
  quit
!
interface Ethernet0
 ip address 20.20.20.20 255.255.255.0
!
interface BRI0
 description bri to rtp
 ip address 12.12.12.12 255.255.255.0
 no ip proxy-arp
 encapsulation ppp
 no ip mroute-cache
 bandwidth 128
 load-interval 30
 dialer idle-timeout 99999
 dialer hold-queue 40
 dialer-group 1
 isdn spid1 919472417100 4724171
 isdn spid2 919472417201 4724172
 ppp authentication chap
 crypto map test
!
ip classless
ip route 0.0.0.0 0.0.0.0 12.12.12.13
access-list 133 permit ip 20.20.20.0 0.0.0.255 40.40.40.0 0.0.0.255
dialer-list 1 protocol ip permit
!
line con 0
 exec-timeout 0 0
line vty 0 4
 password ww
 login
!
end
```

Example 13-9 *Debugs on the Router lab-isdn*

```
lab-isdn#show debug
Cryptographic Subsystem:
  Crypto ISAKMP debugging is on
  Crypto Engine debugging is on
  Crypto IPSEC debugging is on
lab-isdn#

lab-isdn#
*Mar 21 20:16:50.871: ISAKMP (4): processing SA payload. message ID = 0
*Mar 21 20:16:50.871: ISAKMP (4): Checking ISAKMP transform 1 against priority
  65535
        policy
```

continues

Example 13-9 *Debugs on the Router lab-isdn (Continued)*

```
*Mar 21 20:16:50.875: ISAKMP:      encryption DES-CBC
*Mar 21 20:16:50.875: ISAKMP:      hash SHA
*Mar 21 20:16:50.875: ISAKMP:      default group 1
!The authentication method shown below is RSA sig. This is the authentication
!method being offered by the other side.
*Mar 21 20:16:50.875: ISAKMP:      auth RSA sig
*Mar 21 20:16:50.879: ISAKMP (4): atts are acceptable. Next payload is 0
*Mar 21 20:16:50.879: Crypto engine 0: generate alg param

*Mar 21 20:16:54.070: CRYPTO_ENGINE: Dh phase 1 status: 0
!The RSA signature authentication process starts here
*Mar 21 20:16:54.090: ISAKMP (4): SA is doing RSA signature authentication
*Mar 21 20:16:57.343: ISAKMP (4): processing KE payload. message ID = 0
*Mar 21 20:16:57.347: Crypto engine 0: generate alg param

*Mar 21 20:17:01.168: ISAKMP (4): processing NONCE payload. message ID = 0
*Mar 21 20:17:01.176: Crypto engine 0: create ISAKMP SKEYID for conn id 4
*Mar 21 20:17:01.188: ISAKMP (4): SKEYID state generated
*Mar 21 20:17:07.331: ISAKMP (4): processing ID payload. message ID = 0
!The cert payload contains the certificate of the other side
*Mar 21 20:17:07.331: ISAKMP (4): processing CERT payload. message ID = 0
*Mar 21 20:17:07.497: ISAKMP (4): cert approved with warning
!The signature payload contains a hash encrypted with the sender's private key
*Mar 21 20:17:07.600: ISAKMP (4): processing SIG payload. message ID = 0
!The encrypted hash is decrypted using the public key in the certificate, bringing
!the hash out for use in the authentication process.
*Mar 21 20:17:07.608: Crypto engine 0: RSA decrypt with public key
*Mar 21 20:17:07.759: generate hmac context for conn id 4
!Based on comparing hashes, the authentication is found to be successful.
*Mar 21 20:17:07.767: ISAKMP (4): SA has been authenticated
*Mar 21 20:17:07.775: generate hmac context for conn id 4
!This router is now encrypting Hash_I with its private key to send to the receiver
!as its signature. This allows the receiver to authenticate it, just as it has
!authenticated the receiver.
*Mar 21 20:17:07.783: Crypto engine 0: RSA encrypt with private key
*Mar 21 20:17:08.672: CRYPTO_ENGINE: key process suspended and continued
*Mar 21 20:17:08.878: CRYPTO_ENGINE: key process suspended and continued
*Mar 21 20:17:09.088: CRYPTO_ENGINE: key process suspended and continued
*Mar 21 20:17:09.291: CRYPTO_ENGINE: key process suspended and continued
*Mar 21 20:17:09.493: CRYPTO_ENGINE: key process suspended and continued
*Mar 21 20:17:09.795: CRYPTO_ENGINE: key process suspended and continued
*Mar 21 20:17:10.973: generate hmac context for conn id 4
*Mar 21 20:17:10.981: ISAKMP (4): processing SA payload. message ID =
  -538880964*Mar 21 20:17:10.981: ISAKMP (4): Checking IPsec proposal 1
*Mar 21 20:17:10.981: ISAKMP: transform 1, AH_SHA_HMAC
*Mar 21 20:17:10.985: ISAKMP:   attributes in transform:
*Mar 21 20:17:10.985: ISAKMP:      encaps is 1
*Mar 21 20:17:10.985: ISAKMP:      SA life type in seconds
*Mar 21 20:17:10.985: ISAKMP:      SA life duration (basic) of 3600
*Mar 21 20:17:10.989: ISAKMP:      SA life type in kilobytes
*Mar 21 20:17:10.989: ISAKMP:      SA life duration (VPI) of
  0x0 0x46 0x50 0x0
```

Example 13-9 *Debugs on the Router lab-isdn (Continued)*

```
*Mar 21 20:17:10.993: ISAKMP (4): atts are acceptable.
*Mar 21 20:17:10.993: ISAKMP (4): Checking IPsec proposal 1
*Mar 21 20:17:10.993: ISAKMP: transform 1, ESP_DES
*Mar 21 20:17:10.997: ISAKMP:   attributes in transform:
*Mar 21 20:17:10.997: ISAKMP:      encaps is 1
*Mar 21 20:17:10.997: ISAKMP:      SA life type in seconds
*Mar 21 20:17:10.997: ISAKMP:      SA life duration (basic) of 3600
*Mar 21 20:17:11.001: ISAKMP:      SA life type in kilobytes
*Mar 21 20:17:11.001: ISAKMP:      SA life duration (VPI) of
  0x0 0x46 0x50 0x0
*Mar 21 20:17:11.001: ISAKMP:      HMAC algorithm is SHA
*Mar 21 20:17:11.005: ISAKMP (4): atts are acceptable.
*Mar 21 20:17:11.005: IPSEC(validate_proposal_request): proposal part #1,
  (key eng. msg.) dest= 12.12.12.12, SRC= 12.12.12.13,
    dest_proxy= 20.20.20.0/0.0.0.0/0/0,
    src_proxy= 40.40.40.0/0.0.0.16/0/0,
    protocol= AH, transform= ah-sha-hmac ,
    lifedur= 0s and 0kb,
    spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x4
*Mar 21 20:17:11.013: IPSEC(validate_proposal_request): proposal part #2,
  (key eng. msg.) dest= 12.12.12.12, SRC= 12.12.12.13,
    dest_proxy= 20.20.20.0/0.0.0.0/0/0,
    src_proxy= 40.40.40.0/0.0.0.16/0/0,
    protocol= ESP, transform= esp-des esp-sha-hmac ,
    lifedur= 0s and 0kb,
    spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x4
*Mar 21 20:17:11.021: ISAKMP (4): processing NONCE payload. message ID = -538880964
*Mar 21 20:17:11.021: ISAKMP (4): processing ID payload. message ID = -538880964
*Mar 21 20:17:11.021: ISAKMP (4): processing ID payload. message ID = -538880964
*Mar 21 20:17:11.025: IPSEC(key_engine): got a queue event...
*Mar 21 20:17:11.029: IPSEC(spi_response): getting spi 112207019 for SA
        from 12.12.12.13     to 12.12.12.12     for prot 2
*Mar 21 20:17:11.033: IPSEC(spi_response): getting spi 425268832 for SA
        from 12.12.12.13     to 12.12.12.12     for prot 3
*Mar 21 20:17:11.279: generate hmac context for conn id 4
*Mar 21 20:17:11.612: generate hmac context for conn id 4
*Mar 21 20:17:11.644: ISAKMP (4): Creating IPsec SAs
*Mar 21 20:17:11.644:         inbound SA from 12.12.12.13     to 12.12.12.12
        (proxy 40.40.40.0      to 20.20.20.0     )
*Mar 21 20:17:11.648:         has spi 112207019 and conn_id 5 and flags 4
*Mar 21 20:17:11.648:         lifetime of 3600 seconds
*Mar 21 20:17:11.648:         lifetime of 4608000 kilobytes
*Mar 21 20:17:11.652:         outbound SA from 12.12.12.12     to 12.12.12.13
        (proxy 20.20.20.0      to 40.40.40.0     )
*Mar 21 20:17:11.652:         has spi 83231845 and conn_id 6 and flags 4
*Mar 21 20:17:11.656:         lifetime of 3600 seconds
*Mar 21 20:17:11.656:         lifetime of 4608000 kilobytes
*Mar 21 20:17:11.656: ISAKMP (4): Creating IPsec SAs
*Mar 21 20:17:11.656:         inbound SA from 12.12.12.13     to 12.12.12.12
        (proxy 40.40.40.0      to 20.20.20.0     )
```

continues

Example 13-9 *Debugs on the Router lab-isdn (Continued)*

```
*Mar 21 20:17:11.660:         has spi 425268832 and conn_id 7 and flags 4
*Mar 21 20:17:11.660:         lifetime of 3600 seconds
*Mar 21 20:17:11.664:         lifetime of 4608000 kilobytes
*Mar 21 20:17:11.664:         outbound SA from 12.12.12.12     to 12.12.12.13
        (proxy 20.20.20.0      to 40.40.40.0     )
*Mar 21 20:17:11.668:         has spi 556010247 and conn_id 8 and flags 4
*Mar 21 20:17:11.668:         lifetime of 3600 seconds
*Mar 21 20:17:11.668:         lifetime of 4608000 kilobytes
*Mar 21 20:17:11.676: IPSEC(key_engine): got a queue event...
*Mar 21 20:17:11.676: IPSEC(initialize_sas): ,
  (key eng. msg.) dest= 12.12.12.12, SRC= 12.12.12.13,
    dest_proxy= 20.20.20.0/255.255.255.0/0/0,
    src_proxy= 40.40.40.0/255.255.255.0/0/0,
    protocol= AH, transform= ah-sha-hmac ,
    lifedur= 3600s and 4608000kb,
    spi= 0x6B024AB(112207019), conn_id= 5, keysize= 0, flags= 0x4
*Mar 21 20:17:11.680: IPSEC(initialize_sas): ,
  (key eng. msg.) SRC= 12.12.12.12, dest= 12.12.12.13,
    src_proxy= 20.20.20.0/255.255.255.0/0/0,
    dest_proxy= 40.40.40.0/255.255.255.0/0/0,
    protocol= AH, transform= ah-sha-hmac ,
    lifedur= 3600s and 4608000kb,
    spi= 0x4F60465(83231845), conn_id= 6, keysize= 0, flags= 0x4
*Mar 21 20:17:11.687: IPSEC(initialize_sas): ,
  (key eng. msg.) dest= 12.12.12.12, SRC= 12.12.12.13,
    dest_proxy= 20.20.20.0/255.255.255.0/0/0,
    src_proxy= 40.40.40.0/255.255.255.0/0/0,
    protocol= ESP, transform= esp-des esp-sha-hmac ,
    lifedur= 3600s and 4608000kb,
    spi= 0x19591660(425268832), conn_id= 7, keysize= 0, flags= 0x4
*Mar 21 20:17:11.691: IPSEC(initialize_sas): ,
  (key eng. msg.) SRC= 12.12.12.12, dest= 12.12.12.13,
    src_proxy= 20.20.20.0/255.255.255.0/0/0,
    dest_proxy= 40.40.40.0/255.255.255.0/0/0,
    protocol= ESP, transform= esp-des esp-sha-hmac ,
    lifedur= 3600s and 4608000kb,
    spi= 0x21240B07(556010247), conn_id= 8, keysize= 0, flags= 0x4
*Mar 21 20:17:11.699: IPSEC(create_sa): sa created,
  (sa) sa_dest= 12.12.12.12, sa_prot= 51,
    sa_spi= 0x6B024AB(112207019),
    sa_trans= ah-sha-hmac , sa_conn_id= 5
*Mar 21 20:17:11.703: IPSEC(create_sa): sa created,
  (sa) sa_dest= 12.12.12.13, sa_prot= 51,
    sa_spi= 0x4F60465(83231845),
    sa_trans= ah-sha-hmac , sa_conn_id= 6
*Mar 21 20:17:11.707: IPSEC(create_sa): sa created,
  (sa) sa_dest= 12.12.12.12, sa_prot= 50,
    sa_spi= 0x19591660(425268832),
    sa_trans= esp-des esp-sha-hmac , sa_conn_id= 7
*Mar 21 20:17:11.707: IPSEC(create_sa): sa created,
  (sa) sa_dest= 12.12.12.13, sa_prot= 50,
```

Example 13-9 *Debugs on the Router lab-isdn (Continued)*

```
    sa_spi= 0x21240B07(556010247),
    sa_trans= esp-des esp-sha-hmac , sa_conn_id= 8
*Mar 21 20:18:06.767: ISADB: reaper checking SA, conn_id = 4
lab-isdn#
```

Using digital certificates allows IPsec implementations to scale very well. However, using digital certificates requires that a proper CA server system be set up to distribute the certificates. This can be somewhat intensive at the beginning, but after it is done, the rewards are security and scalability.

Router-to-Router IPsec Using RSA Encrypted Nonces

The RSA encrypted keys method is another way to authenticate two end devices using IPsec. This method involves generating RSA keys, as done when using certificates. However, one of the issues with using RSA encrypted nonces is that this method requires the public key of the two peers to be available on the two peers before the third message of the IKE exchange is sent. This is because the third and fourth messages of IKE exchange include nonces, which this method of authentication needs to protect by encrypting them with the receiver's public key. Because the first two messages do not allow a certificate payload to be exchanged, the certificates containing the public keys must be exchanged out of band between the two peers before the IKE process can begin.

The RSA public keys can be exchanged by first generating them and then copying and pasting them into the peer's configuration.

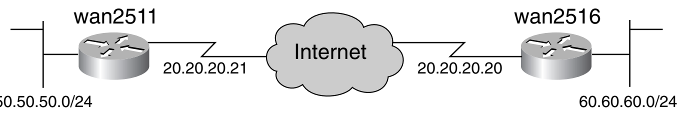
In the following example, the public key for the 2516 router is being copied manually to the 2511 router:

```
wan2511(config)#crypto key pubkey-chain rsa
wan2511(config-pubkey-chain)#named-key wan2516.cisco.com
wan2511(config-pubkey-key)#key-string
Enter a public key as a hexidecimal number ....

wan2511(config-pubkey)#$86F70D 01010105 00034B00 30480241 00DC3DDC 59885F14
wan2511(config-pubkey)#$D918DE FC7ADB76 B0B9DD1A ABAF4884 009E758C 4064C699
wan2511(config-pubkey)#$220CB9 31E267F8 0259C640 F8DE4169 1F020301 0001
wan2511(config-pubkey)#quit
wan2511(config-pubkey-key)#^Z
wan2511#
```

Another interesting thing to note is the way the ID is encrypted during the IKE authentication process. The identity might be too large for all of it to be encrypted in the peer's public key. So the order in which the ID is used in this method of authentication is distinguished name, FQDN, IP address.

Examples 13-10 and 13-11 show the configuration on the two routers used in this case study while example 13-12 shows the debug output on the 'wan2511' router. Figure 13-47 shows the network diagram for this case study.

Figure 13-47 *Network Diagram for This Case Study*



Example 13-10 *Configuration of the Router wan2511*

```
wan2511#show run
Building configuration...

Current configuration:
!
version 11.3
service timestamps debug datetime msec
!
hostname wan2511
!
enable password <removed>
!
no ip domain-lookup
!The command below is used to define the name resolution for the peer to which
!this router will be talking IPsec
ip host wan2516.cisco.com 20.20.20.20
ip domain-name cisco.com
!
crypto isakmp policy 1
!The command below sets up RSA encrypted nonces as the method of authentication.
 authentication rsa-encr
 group 2

crypto isakmp identity hostname
!
crypto ipsec transform-set auth2 ah-sha-hmac esp-des esp-sha-hmac
!
crypto map test 10 ipsec-isakmp
 set peer 20.20.20.20
 set transform-set auth2
 match address 133
!
!Below is the public key for the peer router, wan2516. This key was received by
!this router via a key-exchange mechanism. However, this can be done manually as
!well (cut and paste).

crypto key pubkey-chain rsa
 named-key wan2516.cisco.com
```

Example 13-10 *Configuration of the Router wan2511 (Continued)*

```
  key-string
   305C300D 06092A86 4886F70D 01010105 00034B00 30480241 00DC3DDC 59885F14
   1AB30DCB 794AB5C7 82D918DE FC7ADB76 B0B9DD1A ABAF4884 009E758C 4064C699
   3BC9D17E C47581DC 50220CB9 31E267F8 0259C640 F8DE4169 1F020301 0001
  quit
!
interface Ethernet0
 ip address 50.50.50.50 255.255.255.0
!
interface Serial0
 ip address 20.20.20.21 255.255.255.0
 encapsulation ppp
 no ip mroute-cache
 crypto map test
!
interface Serial1
 no ip address
 shutdown
!
ip classless

ip route 60.0.0.0 255.0.0.0 20.20.20.20
access-list 133 permit ip 50.50.50.0 0.0.0.255 60.60.60.0 0.0.0.255
!
line con 0
 exec-timeout 0 0
 password ww
 login

line vty 0 4
 password ww
 login
!
end
```

Example 13-11 *Configuration of the Router wan2516*

```
wan2516#write terminal

Building configuration...

Current configuration:
!
version 11.3
no service pad
service timestamps debug datetime msec
!
hostname wan2516
```

continues

Example 13-11 *Configuration of the Router wan2516 (Continued)*

```
!
enable password ww
!
no ip domain-lookup

ip host wan2511.cisco.com 20.20.20.21
ip domain-name cisco.com
!
crypto isakmp policy 1authentication rsa-encr
 group 2

crypto isakmp identity hostname
!
crypto ipsec transform-set auth2 ah-sha-hmac esp-des esp-sha-hmac
!
crypto map test 10 ipsec-isakmp
 set peer 20.20.20.21
 set transform-set auth2
 match address 144
!
!The key below is the public key of the peer
crypto key pubkey-chain rsa
 named-key wan2511.cisco.com
  key-string
   305C300D 06092A86 4886F70D 01010105 00034B00 30480241 00E9007B E5CD7DC8
   6E1C0423 92044254 92C972AD 0CCE9796 86797EAA B6C4EFF0 0F0A5378 6AFAE43B
   3A2BD92F 98039DAC 08741E82 5D9053C4 D9CFABC1 AB54E0E2 BB020301 0001
  quit
!
!
interface Ethernet0
 ip address 60.60.60.60 255.255.255.0
!
interface Serial0
 ip address 20.20.20.20 255.255.255.0
 encapsulation ppp
 clockrate 2000000
 crypto map test
!
ip default-gateway 20.20.20.21
ip classless
ip route 0.0.0.0 0.0.0.0 20.20.20.21
access-list 144 permit ip 60.60.60.0 0.0.0.255 50.50.50.0 0.0.0.255
!
line con 0
 exec-timeout 0 0
 password ww
 login
line aux 0
 password ww
 login
```

Example 13-11 *Configuration of the Router wan2516 (Continued)*

```
line vty 0 4
 password ww
 login
!
end
```

Example 13-12 *Debugs on the Router wan2511*

```
wan2511#show debug
Cryptographic Subsystem:
  Crypto ISAKMP debugging is on

wan2511#
*Mar  1 00:27:23.279: ISAKMP (6): processing SA payload. message ID = 0
*Mar  1 00:27:23.279: ISAKMP (6): Checking ISAKMP transform 1 against priority 1
  policy
*Mar  1 00:27:23.283: ISAKMP:      encryption DES-CBC
*Mar  1 00:27:23.283: ISAKMP:      hash SHA
*Mar  1 00:27:23.283: ISAKMP:      default group 2
!The authentication method offered by the peer is RSA encrypted nonces
*Mar  1 00:27:23.287: ISAKMP:      auth RSA encr
*Mar  1 00:27:23.287: ISAKMP:      life type in seconds
*Mar  1 00:27:23.287: ISAKMP:      life duration (basic) of 240
*Mar  1 00:27:23.291: ISAKMP (6): atts are acceptable. Next payload is 0
*Mar  1 00:27:32.055: ISAKMP (6): Unable to get router cert to find DN!
!RSA encrypted nonce authentication has started
*Mar  1 00:27:32.055: ISAKMP (6): SA is doing RSA encryption authentication
*Mar  1 00:27:41.183: ISAKMP (6): processing KE payload. message ID = 0
!Both the ID payload and the nonce payload are encrypted using the peer's public
!key
*Mar  1 00:27:51.779: ISAKMP (6): processing ID payload. message ID = 0
*Mar  1 00:27:54.507: ISAKMP (6): processing NONCE payload. message ID = 0
*Mar  1 00:27:57.239: ISAKMP (6): SKEYID state generated
*Mar  1 00:27:57.627: ISAKMP (6): processing KE payload. message ID = 0
*Mar  1 00:27:57.631: ISAKMP (6): processing ID payload. message ID = 0
*Mar  1 00:28:00.371: ISAKMP (6): processing NONCE payload. message ID = 0
*Mar  1 00:28:13.587: ISAKMP (6): processing HASH payload. message ID = 0
!Authentication has succeeded at this point
*Mar  1 00:28:13.599: ISAKMP (6): SA has been authenticated
*Mar  1 00:28:13.939: ISAKMP (6): processing SA payload. message ID = -161552401
*Mar  1 00:28:13.943: ISAKMP (6): Checking IPsec proposal 1
*Mar  1 00:28:13.943: ISAKMP: transform 1, AH_SHA_HMAC
*Mar  1 00:28:13.943: ISAKMP:   attributes in transform:
*Mar  1 00:28:13.947: ISAKMP:      encaps is 1
*Mar  1 00:28:13.947: ISAKMP:      SA life type in seconds
*Mar  1 00:28:13.947: ISAKMP:      SA life duration (basic) of 3600
*Mar  1 00:28:13.951: ISAKMP:      SA life type in kilobytes
*Mar  1 00:28:13.951: ISAKMP:      SA life duration (VPI) of
  0x0 0x46 0x50 0x0
```

continues

Example 13-12 *Debugs on the Router wan2511 (Continued)*

```
*Mar  1 00:28:13.955: ISAKMP (6): atts are acceptable.
*Mar  1 00:28:13.959: ISAKMP (6): Checking IPsec proposal 1
*Mar  1 00:28:13.959: ISAKMP: transform 1, ESP_DES
*Mar  1 00:28:13.959: ISAKMP:   attributes in transform:
*Mar  1 00:28:13.963: ISAKMP:      encaps is 1
*Mar  1 00:28:13.963: ISAKMP:      SA life type in seconds
*Mar  1 00:28:13.963: ISAKMP:      SA life duration (basic) of 3600
*Mar  1 00:28:13.967: ISAKMP:      SA life type in kilobytes
*Mar  1 00:28:13.967: ISAKMP:      SA life duration (VPI) of
  0x0 0x46 0x50 0x0
*Mar  1 00:28:13.971: ISAKMP:      HMAC algorithm is SHA
*Mar  1 00:28:13.971: ISAKMP (6): atts are acceptable.
*Mar  1 00:28:13.975: ISAKMP (6): processing NONCE payload. message ID = -161552401
*Mar  1 00:28:13.979: ISAKMP (6): processing ID payload. message ID = -161552401
*Mar  1 00:28:13.979: ISAKMP (6): processing ID payload. message ID = -161552401
*Mar  1 00:28:14.391: ISAKMP (6): Creating IPsec SAs
*Mar  1 00:28:14.391:         inbound SA from 20.20.20.20     to 20.20.20.21
        (proxy 60.60.60.0      to 50.50.50.0     )
*Mar  1 00:28:14.395:         has spi 437593758 and conn_id 7 and flags 4
*Mar  1 00:28:14.399:         lifetime of 3600 seconds
*Mar  1 00:28:14.399:         lifetime of 4608000 kilobytes
*Mar  1 00:28:14.403:         outbound SA from 20.20.20.21     to 20.20.20.20
        (proxy 50.50.50.0      to 60.60.60.0     )
*Mar  1 00:28:14.403:         has spi 411835612 and conn_id 8 and flags 4
*Mar  1 00:28:14.407:         lifetime of 3600 seconds
*Mar  1 00:28:14.407:         lifetime of 4608000 kilobytes
*Mar  1 00:28:14.411: ISAKMP (6): Creating IPsec SAs
*Mar  1 00:28:14.411:         inbound SA from 20.20.20.20     to 20.20.20.21
        (proxy 60.60.60.0      to 50.50.50.0     )
*Mar  1 00:28:14.415:         has spi 216990519 and conn_id 9 and flags 4
*Mar  1 00:28:14.415:         lifetime of 3600 seconds
*Mar  1 00:28:14.419:         lifetime of 4608000 kilobytes
*Mar  1 00:28:14.419:         outbound SA from 20.20.20.21     to 20.20.20.20
        (proxy 50.50.50.0      to 60.60.60.0     )
*Mar  1 00:28:14.423:         has spi 108733569 and conn_id 10 and flags 4
*Mar  1 00:28:14.423:         lifetime of 3600 seconds
*Mar  1 00:28:14.427:         lifetime of 4608000 kilobytes
wan2511#
```

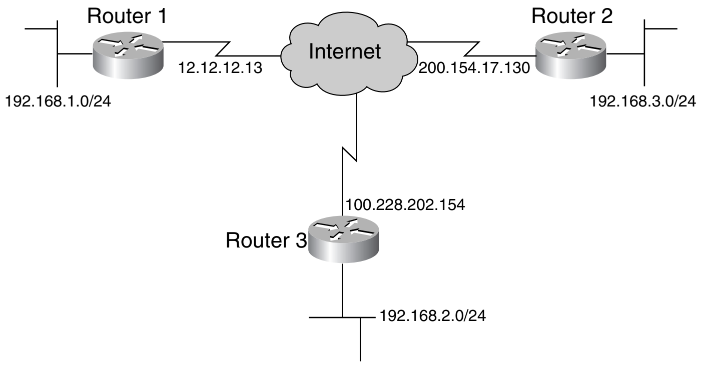
The encrypted nonce IKE authentication method does not scale as well as the digital signatures method of authentication using digital certificates. This is because the peers' public keys have to be manually exchanged. However, if scalability is not a concern, and if the other device can be trusted to provide the correct public key, this method can provide a level of security comparable to digital certificates. In fact, there is additional security, because the nonces are exchanged encrypted rather than in the clear, providing an added level of security against some DH attacks.

One-to-Many Router IPsec

This is a common scenario often used when a company has multiple offices connecting to each other. The setup in this case study is fully meshed, meaning that each site can talk to every other site directly rather than having to go through a hub in the middle. While you can apply only one crypto map to an interface, a crypto map can have multiple instances, each instance being used for a unique site.

Examples 13-13, 13-14 and 13-15 show the configurations on each of the three routers used in this case study. Figure 13-48 shows the network diagram for this case study.

Figure 13-48 *Network Diagram for This Case Study*



Example 13-13 *Configuration of Router1*

```
router1#write terminal
Building configuration...
Current configuration:
!
version 12.0
service timestamps debug uptime
service timestamps log uptime
!
hostname router1
!
logging buffered 4096 debugging
enable password
!
clock timezone est 8
ip subnet-zero
!
crypto isakmp policy 4
  authentication pre-share
crypto isakmp key xxxxxx1234 address 100.228.202.154
```

continues

Example 13-13 *Configuration of Router1 (Continued)*

```
crypto isakmp key xxxxxx5678 address 200.154.17.130
!
!
crypto ipsec transform-set encrypt-des esp-des esp-sha-hmac
!
!
crypto map combined local-address Serial0

!Instance 20 of the crypto map below is used for the first peer, and instance 30
!is for the second peer. The router uses whichever instance of the crypto map has
!the access list matching the traffic coming through.
crypto map combined 20 ipsec-isakmp
  set peer 100.228.202.154
  set transform-set encrypt-des
  match address 106
crypto map combined 30 ipsec-isakmp
  set peer 200.154.17.130
  set transform-set encrypt-des
  match address 105
!
!
interface Serial0
  ip address 100.232.202.210 255.255.255.252
  no ip directed-broadcast
  ip nat outside
  no ip route-cache
  no ip mroute-cache
  no fair-queue
  no cdp enable
  crypto map combined
!
interface FastEthernet0
  ip address 192.168.1.1 255.255.255.0
  no ip directed-broadcast
  ip nat inside
  no cdp enable
!
!
ip nat inside source route-map nonat interface Serial0 overload
ip classless
ip route 0.0.0.0 0.0.0.0 100.232.202.209
ip route 192.168.2.0 255.255.255.0 100.232.202.209
ip route 192.168.3.0 255.255.255.0 100.232.202.209
no ip http server
!
access-list 105 permit ip 192.168.1.0 0.0.0.255 192.168.3.0 0.0.0.255
access-list 106 permit ip 192.168.1.0 0.0.0.255 192.168.2.0 0.0.0.255
access-list 150 deny ip 192.168.1.0 0.0.0.255 192.168.2.0 0.0.0.255
access-list 150 deny ip 192.168.1.0 0.0.0.255 192.168.3.0 0.0.0.255
access-list 150 permit ip 192.168.1.0 0.0.0.255 any
no cdp run
```

Example 13-13 *Configuration of Router1 (Continued)*

```
route-map nonat permit 10
  match ip address 150
!
!
line con 0
  transport input none
  line aux 0
  password 7 aaaaaaa
  login local
  modem InOut
  transport input all
  speed 38400
  flowcontrol hardware
line vty 0 4
  exec-timeout 30 0
  password aaaaaa
login
!
end
```

Example 13-14 *Configuration of Router2*

```
router2#write terminal
Current configuration:
!
version 12.0
service timestamps debug uptime
service timestamps log uptime
service password-encryption
!
hostname router2
!
enable secret 5 <removed>
enable password 7 <removed>
!
clock timezone EST 8
ip subnet-zero
!
crypto isakmp policy 4
  authentication pre-share
crypto isakmp key xxxxxx101112 address 100.228.202.154
crypto isakmp key xxxxxx5678 address 100.232.202.210
!
!
crypto ipsec transform-set 1600_box esp-des esp-sha-hmac
!
!
crypto map combined local-address Ethernet1
```

continues

Example 13-14 *Configuration of Router2 (Continued)*

```
crypto map combined 7 ipsec-isakmp
  set peer 100.232.202.210
  set transform-set 1600_box
  match address 105
crypto map combined 8 ipsec-isakmp
  set peer 100.228.202.154
  set transform-set 1600_box
  match address 106
!
!
!
interface Ethernet0
  ip address 192.168.3.1 255.255.255.0
  no ip directed-broadcast
  ip nat inside
!
interface Ethernet1
  ip address 200.154.17.130 255.255.255.224
  no ip directed-broadcast
  ip nat outside
  crypto map combined
!
ip nat inside source route-map nonat interface Ethernet1 overload
ip classless
ip route 0.0.0.0 0.0.0.0 200.154.17.129

no ip http server
!

access-list 105 permit ip 192.168.3.0 0.0.0.255 192.168.1.0 0.0.0.255
access-list 106 permit ip 192.168.3.0 0.0.0.255 192.168.2.0 0.0.0.255
access-list 150 deny ip 192.168.3.0 0.0.0.255 192.168.1.0 0.0.0.255
access-list 150 deny ip 192.168.3.0 0.0.0.255 192.168.2.0 0.0.0.255
access-list 150 permit ip any any
route-map nonat permit 10
  match ip address 150
!

line con 0
  password 7 aaaaa
  transport input none
line vty 0 4
  exec-timeout 0 0
  password 7 aaaaaa
login
!
end
```

Example 13-15 *Configuration of Router3*

```
router3#write terminal
Current configuration:
!
version 12.0
service timestamps debug uptime
service timestamps log uptime

!
hostname router3
!
logging buffered 4096 debugging
enable secret 5 <removed>
enable password <removed>
!
clock timezone est 8
ip subnet-zero
!
!
crypto isakmp policy 4
  authentication pre-share
crypto isakmp key xxxxxx1234 address 100.232.202.210
crypto isakmp key xxxxxx101112 address 200.154.17.130
!
crypto map combined local-address Serial0
crypto map combined 7 ipsec-isakmp
  set peer 100.232.202.210
  set transform-set encrypt-des
  match address 106
crypto map combined 8 ipsec-isakmp
  set peer 200.154.17.130
  set transform-set 1600_box
  match address 105
!
!
interface Serial0
  ip address 100.228.202.154 255.255.255.252
  no ip directed-broadcast
  ip nat outside
  crypto map combined
!
interface FastEthernet0
  ip address 192.168.2.1 255.255.255.0
  no ip directed-broadcast
  ip nat inside
  no cdp enable
!
!
ip nat inside source route-map nonat interface Serial0 overload
ip classless
ip route 0.0.0.0 0.0.0.0 100.228.202.153
```

continues

Example 13-15 *Configuration of Router3 (Continued)*

```
no ip http server
!
access-list 105 permit ip 192.168.2.0 0.0.0.255 192.168.3.0 0.0.0.255
access-list 106 permit ip 192.168.2.0 0.0.0.255 192.168.1.0 0.0.0.255
access-list 150 deny ip 192.168.2.0 0.0.0.255 192.168.3.0 0.0.0.255
access-list 150 deny ip 192.168.2.0 0.0.0.255 192.168.1.0 0.0.0.255
access-list 150 permit ip 192.168.2.0 0.0.0.255 any
no cdp run
route-map nonat permit 10
  match ip address 150
!
!
line con 0
  transport input none
line aux 0
  password aaaa
  login local
  modem InOut
  transport input all
  speed 115200
  flowcontrol hardware
line vty 0 4
  exec-timeout 30 0
  password aaaaa
login
!
no scheduler allocate
end
```

Although the configurations shown here are for a setup in which a fully meshed environment is desirable, a partially meshed setup can be arranged as well. The accessibility of peers is determined by the crypto maps set up on the peers.

In addition to the mesh setup seen in this case study, another type of setup is a hub-and-spoke setup. In this type of setup, a hub site routes all the encrypted traffic to the peers.

High-Availability-IPsec-Over-GRE Setup

IPsec over GRE is a popular implementation mechanism often used in conjunction with IPsec. It is especially useful in some cases in which IPsec alone cannot fulfill the setup's requirements, such as when multicast traffic needs to be tunneled. This list summarizes the cases in which IPsec over GRE is often used:

- To send multicast or broadcast traffic over the VPN link.
- To send non-IP protocol-based traffic over the VPN link.

- To have high availability. By participating in routing protocols, GRE tunnels can allow IPsec tunnels to stay up even when a primary path goes down. This happens when the routing protocol allows a peer to realize that its peer cannot be reached via a given path and then converges on a different path to access the peer.
- To scale an IPsec implementation. GRE allows a single IPsec access control element (ACE) to specify the traffic that needs to be encrypted. This is done with the use of a "GRE any any" ACE defining IPsec interesting traffic.

  Note that although "any any" in an ACE is not recommended to be used to define IPsec interesting traffic, it is acceptable to use such an ACE in conjunction with using GRE with IPsec. The reason the use of an ACE with plain IPsec is discouraged is that it results in *all* traffic to and from the router being encrypted, which is often an undesirable outcome. However, with GRE, the amount of traffic that is to be encrypted is not all the traffic, just the traffic that has already been GRE-encapsulated, thereby reducing the all-encompassing behavior of the "any any" ACE.

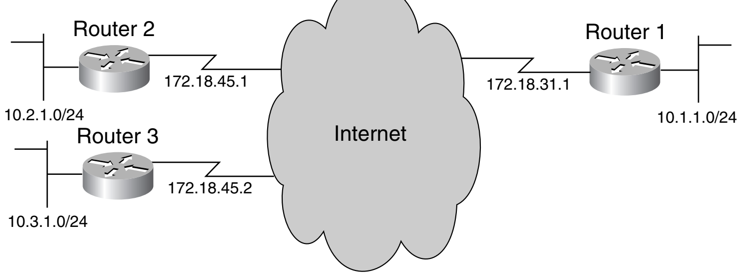
Using GRE with IPsec to encapsulate and encrypt multicast traffic is very useful in setups in which there is a need to route routing protocols such as EIGRP over the tunnel. Using GRE allows two networks to be seamlessly connected in terms of routing.

IPsec is often used in transport mode with GRE because GRE can provide the tunneling that IPsec tunnel mode provides. Therefore, not using IPsec tunnel mode saves some amount of packet overhead.

GRE can carry this kind of traffic, but it does not have the IPsec protocol's encryption-handling capabilities. Therefore, a combination of the two works well. The overhead to the packet added by GRE is often insignificant as compared to IPsec. The sequence of events that takes place in this setup is as follows: The plain-text packet is first encapsulated using the GRE protocol. After this is done, IPsec takes over and encrypts the GRE packet. The crypto map needs to be applied to the physical interface and the tunnel interface. This statement is true for any logical interface you might be using in an IPsec setup (such as BVI interfaces). In this case study we see IPX traffic sent encrypted across the VPN link.

Examples 13-16, 13-17 and 13-18 show the configurations on each of the three routers used in this case study. Figure 13-49 shows the network diagram for this case study.

Figure 13-49 *Network Diagram for This Case Study*



Example 13-16 *Configuration of Router1*

```
router1#write terminal
Current configuration:
!
version 12.0
service timestamps debug uptime
service timestamps log uptime
!
hostname router1
!
logging buffered 4096 debugging
enable secret 5 <removed>
enable password <removed>
!
clock timezone est 8
ip subnet-zero
!

crypto isakmp policy 10
 authentication pre-share
crypto isakmp key cisco123 address 172.18.45.1
crypto isakmp key cisco345 address 172.18.45.2
!
crypto ipsec transform-set one esp-des esp-md5-hmac
mode transport
!
!Crypto map for router 2
crypto map gre 10 ipsec-isakmp
 set peer 172.18.45.1
 set transform-set one
 match address gre1
!Crypto map for router 3
crypto map gre 20 ipsec-isakmp
 set peer 172.18.45.2
 set transform-set one
 match address gre2
!
!Tunnel to router 1
interface Tunnel0
 ip address 10.4.1.1 255.255.255.0
 tunnel source 172.18.31.1
 tunnel destination 172.18.45.1
 crypto map gre
!
!Tunnel to router 2
interface Tunnel1
 ip address 10.4.2.1 255.255.255.0
 tunnel source 172.18.31.1
 tunnel destination 172.18.45.2
 crypto map gre
!
```

Example 13-16 *Configuration of Router1 (Continued)*

```
!
interface Ethernet0
 ip address 10.1.1.1 255.255.255.0
!
interface Serial0
 ip address 172.18.31.1 255.255.255.0
 crypto map gre
!
!
ip classless
ip route 172.18.0.0 255.255.0.0 serial0
!Below is the configuration for GRE used to do the routing over the GRE tunnels
ip eigrp 100
 network 10.0.0.0
!
!
!To reduce the number of access lists, in case a large number of peers are set up,
!a simple GRE any any access list can be used to define the interesting traffic.
!Only one such access list is needed.

ip access-list extended gre1
 permit gre host 172.18.31.1 host 172.18.45.1
!
ip access-list extended gre2
 permit gre host 172.18.31.1 host 172.18.45.2
```

Example 13-17 *Configuration of Router2*

```
router2#write terminal
Current configuration:
!
version 12.0
service timestamps debug uptime
service timestamps log uptime
!
hostname router2
!
logging buffered 4096 debugging
enable secret 5 <removed>
enable password <removed>
!
clock timezone est 8
ip subnet-zero
!

crypto isakmp policy 10
 authentication pre-share
crypto isakmp key cisco123 address 172.18.31.1
```

continues

Example 13-17 *Configuration of Router2 (Continued)*

```
!
crypto ipsec transform-set one esp-des esp-md5-hmac
 mode transport
!
crypto map gre 10 ipsec-isakmp
 set peer 172.18.31.1
 set transform-set one
 match address gre1
!
!
interface Tunnel0
 ip address 10.10.1.1 255.255.255.0
 tunnel source 172.18.45.1
 tunnel destination 172.18.31.1
 crypto map gre
!
!
interface Ethernet0
 ip address 10.2.1.1 255.255.255.0
!
interface Serial0
 ip address 172.18.45.1 255.255.255.0
 crypto map gre
!
!
ip classless
ip route 172.18.0.0 255.255.0.0 serial0

ip eigrp 100
 network 10.0.0.0
!
!
ip access-list extended gre1
 permit gre host 172.18.45.1 host 172.18.31.1
!
```

Example 13-18 *Configuration of Router3*

```
router3#write terminal
Current configuration:
!
version 12.0
service timestamps debug uptime
service timestamps log uptime
!
hostname router3
!
logging buffered 4096 debugging
enable secret 5 <removed>
enable password <removed>
```

Example 13-18 *Configuration of Router3 (Continued)*

```
!
clock timezone est 8
ip subnet-zero
!

crypto isakmp policy 10
 authentication pre-share
crypto isakmp key cisco345 address 172.18.31.1

!
crypto ipsec transform-set one esp-des esp-md5-hmac
 mode transport
!
crypto map gre 10 ipsec-isakmp
 set peer 172.18.31.1
 set transform-set one
 match address gre1
!
!
interface Tunnel0
 ip address 10.5.1.1 255.255.255.0
 tunnel source 172.18.45.2
 tunnel destination 172.18.31.1
 crypto map gre
!
!

interface Ethernet0
 ip address 10.3.1.1 255.255.255.0
!
interface Serial0
 ip address 172.18.45.2 255.255.255.0
 crypto map gre
!
!
ip classless
ip route 172.18.0.0 255.255.0.0 serial0

ip eigrp 100
 network 10.0.0.0
!
!
ip access-list extended gre1
 permit gre host 172.18.45.2 host 172.18.31.1
!
```

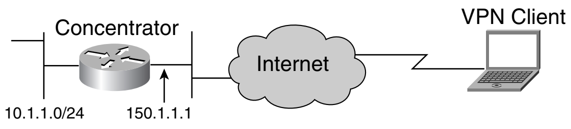
Remote-Access IPsec Using X-Auth, Dynamic Crypto Maps, Mode Config, and Preshared Keys

This case study is essentially a compilation of a number of features that can be used with remote-access IPsec VPNs.

- As discussed earlier, x-auth (extended authentication) is the method used to allow users to authenticate themselves on a per-user basis rather than use only a wildcard preshared secret shared by all the users.
- Dynamic crypto maps allow the router to accept connections from VPN clients whose IP addresses it does not know.
- Mode configs allow the router to assign IP addresses to the VPN clients after they have connected to it. This address, known as the internal IP address, is different from the IP address assigned to the clients when they connect to the ISP. The ISP address is used to carry the IPsec packet over the Internet. The internal IP address, often a private address, is embedded in the IPsec packet and is used to communicate with the private network behind the router.

The configuration in this case study is based on Cisco Secure VPN Client 1.1. The router configuration is included in Example 13-19 while the configuration for the VPN client is shown in Example 13-20. Figure 13-50 shows the network diagram for this case study.

Figure 13-50 *Network Diagram for This Case Study*



Example 13-19 *Configuration of the Router Called Concentrator*

```
Concentrator#show running-config
Building configuration...

Current configuration:
!
version 12.0
service timestamps debug uptime
service timestamps log uptime
!
hostname Concentrator
!
enable secret 5 <removed>
enable password ww
!
aaa new-model
```

Example 13-19 *Configuration of the Router Called Concentrator (Continued)*

```
!The command below defines the mechanism to use for extended authentication

aaa authentication login xauth local
!The user defined below will be used to authenticate the x-auth clients
username ciscouser password 0 cisco1234

!
ip subnet-zero
!
crypto isakmp policy 10
 hash md5
 authentication pre-share

!The wildcard preshared key is configured because the client's IP address is
!unknown

crypto isakmp key cisco123 address 0.0.0.0

!Here we identify the pool to use to assign addresses to the VPN clients

crypto isakmp client configuration address-pool local mypool

!Binding the extended authentication list to the crypto map being used

crypto map REMOTEACCESS client authentication list xauth
!


crypto ipsec transform-set RTP-TRANSFORM esp-des esp-md5-hmac
!
crypto dynamic-map vpn 1
 set transform-set RTP-TRANSFORM
!
!The following commands request that IP addresses be assigned to the VPN clients

crypto map REMOTEACCESS client configuration address initiate
crypto map REMOTEACCESS client configuration address respond

!Below is the static crypto map that points to a dynamic crypto map. This is
!applied to the router's outside interface. Note that a dynamic crypto map cannot
!be applied directly to an interface.

crypto map REMOTEACCESS 1 ipsec-isakmp dynamic vpn
!
interface Ethernet0/0
ip address 150.1.1.1 255.255.255.0
crypto map REMOTEACCESS
!
```

continues

Example 13-19 *Configuration of the Router Called Concentrator (Continued)*

```
!
interface Ethernet0/1
ip address 11.10.1.1 255.255.255.0
no ip directed-broadcast
!
!This is the pool from which we assign addresses to the clients

ip local pool mypool 10.1.10.0 10.1.10.254
ip nat translation timeout never
ip nat translation tcp-timeout never
ip nat translation udp-timeout never
ip nat translation finrst-timeout never
ip nat translation syn-timeout never
ip nat translation dns-timeout never
ip nat translation icmp-timeout never
ip classless
ip route 0.0.0.0 0.0.0.0 10.103.1.1
no ip http server
!
dialer-list 1 protocol ip permit
dialer-list 1 protocol ipx permit
!
line con 0
transport input none
line aux 0
line vty 0 4
password <removed>
login
!
end
```

Example 13-20 *Configuration of a VPN Client Connecting to the Router Called Concentrator*

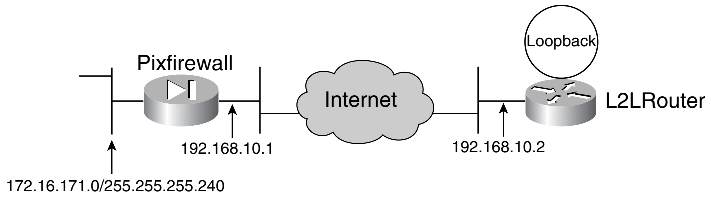
```
VPN CLIENT CONFIGURATION

VPNClient - Pre-shared, Wild-card, Mode-config
Name of connection:
Remote party address = IP_Subnet = 10.1.1.0, Mask 255.255.255.0
Connect using Secure Gateway Tunnel to 150.1.1.1
My Identity:
Select certificate = None
ID_Type = ip address (171.68.118.143), pre-shared key and fill in key ('cisco123')
Security policy = defaults
Proposal 1 (Authen) = DES, MD5
Proposal 2 (Key Exchange) = DES, MD5, Tunnel
```

PIX IPsec Combining LAN-to-LAN and Remote-Access Setups

The PIX Firewall can also terminate IPsec tunnels. This case study shows a setup in which the PIX terminates a LAN-to-LAN and remote-access IPsec tunnel. The same crypto map is used for both the VPN clients and the LAN-to-LAN connection. Different instance numbers are applied to distinguish between the crypto maps used for the two types of tunnels. The VPN-relevant configuration is shown in bold.

Example 13-21 shows the configuration on the PIX firewall for this case study. Example 13-22 shows the setup of the router connecting to the PIX using IPsec. Figure 13-51 shows the network diagram for this case study.

Figure 13-51 *Network Diagram for This Case Study*



Example 13-21 *Configuration of the PIX Called Pixfirewall*

```
pixfirewall#write terminal
PIX Version 5.2
nameif ethernet0 outside security0
nameif ethernet1 inside security100
nameif ethernet2 dmz security50
access-list bypassingnat permit ip 172.16.0.0 255.255.0.0 10.1.100.0 255.255.255.0
access-list bypassingnat permit ip 172.16.0.0 255.255.0.0 10.1.1.0 255.255.255.0
access-list bypassingnat permit ip host 20.1.1.1 host 10.1.1.1
access-list inboundtraffic permit icmp any any
access-list inboundtraffic permit tcp any host 192.168.10.100 eq telnet
access-list 101 permit ip host 20.1.1.1 host 10.1.1.1
enable password <removed> encrypted
passwd <removed> encrypted
hostname pixfirewall
fixup protocol ftp 21
fixup protocol http 80
fixup protocol smtp 25
fixup protocol h323 1720
fixup protocol rsh 514
fixup protocol sqlnet 1521
names
pager lines 24
no logging timestamp
no logging standby
no logging console debugging
```

continues

Example 13-21 *Configuration of the PIX Called Pixfirewall (Continued)*

```
no logging monitor
no logging buffered
no logging trap
no logging history
logging facility 20
logging queue 512
interface ethernet0 auto
interface ethernet1 auto
interface ethernet2 auto
mtu outside 1500
mtu inside 1500
mtu dmz 1500
ip address outside 192.168.10.1 255.255.255.0
ip address inside 172.16.171.5 255.255.255.240
ip address dmz 127.0.0.1 255.255.255.255
!The command below defines the pool of addresses from which addresses get assigned
!to the clients connecting to the PIX via mode config
ip local pool vpnpool 10.1.100.240-10.1.100.250
no failover
failover timeout 0:00:00
failover ip address outside 0.0.0.0
failover ip address inside 0.0.0.0
failover ip address dmz 0.0.0.0
arp timeout 14400
global (outside) 1 192.168.10.20-192.168.10.30 netmask 255.255.255.0
global (outside) 1 192.168.10.19 netmask 255.255.255.0
!The NAT command configured below stops NAT from occurring for IP addresses pairs
!that the PIX considers IPsec interesting traffic. This includes packets destined
!for the remote LAN as well as pool addresses assigned to the VPN clients. See the
!access list bypassingnat above to see which addresses are defined to bypass NAT.
nat (inside) 0 access-list bypassingnat
nat (inside) 1 172.16.0.0 255.255.0.0 0 0
static (inside,outside) 192.168.10.100 172.16.171.13 netmask 255.255.255.255 0 0
static (inside,outside) 20.1.1.1 20.1.1.1 netmask 255.255.255.255 0 0
access-group inboundtraffic in interface outside
route outside 0.0.0.0 0.0.0.0 192.168.10.2 1
route inside 20.0.0.0 255.0.0.0 172.16.171.13 1
route inside 171.68.0.0 255.255.0.0 172.16.171.1 1
route inside 172.16.0.0 255.255.0.0 172.16.171.1 1
timeout xlate 3:00:00 conn 1:00:00 half-closed 0:10:00 udp 0:02:00
timeout rpc 0:10:00 h323 0:05:00
timeout uauth 0:05:00 absolute
aaa-server TACACS+ protocol tacacs+
aaa-server RADIUS protocol radius
aaa-server myserver protocol tacacs+
aaa-server myserver (inside) host 171.68.178.124 cisco timeout 5
no snmp-server location
no snmp-server contact
snmp-server community public
no snmp-server enable traps
floodguard enable
```

Example 13-21 *Configuration of the PIX Called Pixfirewall (Continued)*

```
!The sysopt command below allows IPsec traffic to bypass the security checks
!performed by the PIX on incoming traffic from a low- to a high-security
!interface. The assumption is that IPsec takes care of nongenuine packets and
!drops them.
sysopt connection permit-ipsec
!The command below defines the transform set to use for IPsec.
crypto ipsec transform-set mysetdes esp-des esp-md5-hmac
!The command below defines the dynamic crypto map used to accept VPN client
!traffic.
crypto dynamic-map mydynmap 10 set transform-set mysetdes
!The crypto map below is used for the LAN-to-LAN tunnels that will be established
!by the PIX.
crypto map newmap 20 ipsec-isakmp
crypto map newmap 20 match address 101
crypto map newmap 20 set peer 192.168.10.2
crypto map newmap 20 set transform-set mysetdes
!The crypto map below is used to tie to the dynamic crypto map defined above.
!Note that this is simply another instance of the crypto map configured for the
!LAN-to-LAN tunnel, because only one crypto map can be applied to a PIX interface.
This !crypto map points to the dynamic crypto map defined above.
crypto map newmap 30 ipsec-isakmp dynamic mydynmap
!The two commands below set up the PIX to do mode configuration for the remote-
!access clients, allowing it to initiate as well as accept requests for mode
!configuration
crypto map newmap client configuration address initiate
crypto map newmap client configuration address respond
!The command below ties the authentication to be done for extended authentication
!to the server defined as 'myserver'
crypto map newmap client authentication myserver
!The command below applies the crypto map to the outside interface
crypto map newmap interface outside
!The isakmp enable command in PIX is an additional command needed for ISAKMP
!configs
!isakmp enable outside
!The preshared secret defined below is the wildcard key for the remote-access
!clients.
isakmp key mysecretkey address 0.0.0.0 netmask 0.0.0.0
!The command below defines the preshared secret for the router. It also has the
!keywords 'no-xauth' and 'no-config-mode' at the end, which specify that x-auth and
!config mode are not to be done with this peer
isakmp key myotherkey address 192.168.10.2 netmask 255.255.255.255 no-xauth no-
config-mode
isakmp identity address
!The command below is used to define the pool that will be used to assign
!addresses via mode config to the VPN clients
isakmp client configuration address-pool local vpnpool outside
!The commands below define the parameters of the ISAKMP policy on the PIX
isakmp policy 10 authentication pre-share
isakmp policy 10 encryption des
isakmp policy 10 hash md5
isakmp policy 10 group 1
```

continues

Example 13-21 *Configuration of the PIX Called Pixfirewall (Continued)*

```
telnet timeout 5
terminal width 80

: end
```

Example 13-22 *Configuration of the Router L2LRouter*

```
L2LRouter#write terminal
Router Configuration
Current configuration:
!
version 12.0
service timestamps debug uptime
service timestamps log uptime

!
hostname L2LRouter
!
aaa new-model
!
ip subnet-zero
!
!
crypto isakmp policy 10
 hash md5
 authentication pre-share
 lifetime 1000
crypto isakmp key myotherkey address 192.168.10.1
!
crypto ipsec transform-set strong esp-des esp-md5-hmac
!
crypto map topix 10 ipsec-isakmp
 set peer 192.168.10.1
 set transform-set strong
 match address 101
!
interface Loopback0
 ip address 10.1.1.1 255.255.255.255
 no ip directed-broadcast
!
interface Ethernet0/0
 ip address 192.168.10.2 255.255.255.0
 no ip directed-broadcast
 crypto map topix

!
ip classless
ip route 0.0.0.0 0.0.0.0 192.168.10.1
no ip http server
```

Example 13-22 *Configuration of the Router L2LRouter (Continued)*

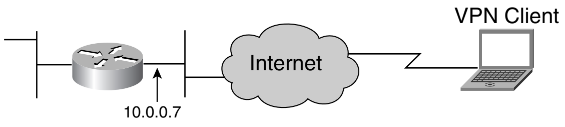
```
!
access-list 101 permit ip host 10.1.1.1 host 20.1.1.1
!
line con 0
 transport input none
line aux 0
line vty 0 4
!
end
```

IPsec Over L2TP Using Voluntary Tunneling

This case study describes the use of the Windows 2000 built-in L2TP and IPsec client to connect to a router. The L2TP tunnel is encrypted using IPsec. In the preceding chapter we looked at an example of compulsory tunneling in which the LAC did the tunneling on behalf of the client. In this case study, L2TP is done by the client itself. This type of scenario is known as *voluntary tunneling*. In addition, IPsec is also run between the client and the IPsec gateway, ensuring that L2TP is protected via the security features offered by IPsec.

Figure 13-52 shows the network diagram for this case study.

Figure 13-52 *Network Diagram for This Case Study*



Setting Up IPsec on the Windows 2000 Host

A description of how to setup Windows 2000 to connect to a router follows. However, the reader is advised to read the following detailed documents on the Microsoft website for more detailed information on how to setup Windows 2000 to do L2TP/IPsec:

"How to Configure a L2TP/IPsec Connection Using Pre-shared Key Authentication" (Q240262)

Although IPsec is part of the Windows 2000 installation, you must enable it via the Microsoft Management Console (MMC) to configure the IPsec service:

Enabling IPsec Service

Step 1 Select **Start > Run** and type **mmc**.

Step 2 The window Console1 opens. From here, select **Add/Remove Snap-in**.

Step 3 Click the **Add** button. A list of snap-ins appears.

Step 4 Go down the list and select **IP Security Policy Management**, and then click **Add**.

Step 5 Close the Add window and return to the console.

Defining an IPsec Policy and associated rules

Three default policies are available by default on the console:

- **Client Respond Only**—Allows unsecured communications but negotiates security if requested by the server.
- **Secure Server**—For all IP traffic, always require security using Kerberos.
- **Server (Request Security)**—For all IP traffic, always request security using Kerberos.

Note that these are defaults only. You can change the information. For example, you can use IKE authentication methods other than Kerberos, but the best thing to do is create your own policy.

To create a customized policy, follow this procedure:

Step 1 Right-click the IP Security Policies icon and select Create IP Security Policy.

Step 2 Use the wizard to create your first policy. Select the defaults until you are asked about the authentication method for IKE. Select either Digital certificates or Pre-Shared key. If you select certificates, you must have access to the certificate authority you want to use, you must have already retrieved the CA's certificate, and you must have enrolled your client with the CA. If you have not done these things, you cannot browse for a CA. You can select something else and then come back later and change it. For this case study assume that pre-shared key was chosen.

Step 3 After creating the policy, you can right-click it and select **properties** to edit it. The default rule is already applied to the policy. You can edit this rule shown as '<dynamic>', or, for simplicity, create a new one.

Step 4 Click **Add**, and go to the **New Rule Properties** window. A number of tabs will be present on this window. In the following steps we will go through each of these tabs to setup the rules needed to configure IPsec to occur for the L2TP traffic.

Step 5 Select **IP Filter List**, click **Add**, name the list, and either use the wizard or add the parameters manually. For L2TP you must configure three lists. The concept is that all traffic in the L2TP tunnel is protected by your IPsec policy. All three lists will be for UDP traffic. The first one will specify traffic going from UDP source port 1701 to UDP destination port 1701. The second one will specify traffic going from UDP source port 1701 to any UDP destination port and the last list will be for traffic coming from any UDP source port and going to the UDP destination port 1701. When adding the filters, remember that you are specifying what traffic you are applying your IPsec policy to. In this case, the policy is applied to the L2TP tunnel, so you are setting up a transport mode IPsec connection between the client and the gateway. Your tunnel endpoints are usually the client's IP address and the gateway's IP address. Follow these steps:

Step 6 Apply the filter list, and move on to filter action.

Step 7 The **Filter action** tab is where you specify your IPsec policy. You can use the wizard or create it manually. You may add more than one combination of algorithms (this is analogous to creating multiple transform sets in Cisco IOS). All combinations are proposed to the gateway.

Step 8 You use the **Authentication Methods** tab to edit the authentication method used during IKE main mode peer authentication. You can define multiple authentication methods. In our example we have defined the preshared key 'ciscosys' on this screen.

Step 9 The **Tunnel Setting** tab is used to define IPsec's transport and tunnel modes. For L2TP, you are setting up a transport mode IPsec connection, so you must select **This rule does not specify an IPsec Tunnel**.

Step 10 The **Connection Type** tab allows you to apply this policy to individual adapters or to all adapters in your PC. You are now finished defining your rule. Make sure you select it from the list and apply it.

Step 11 The actual IKE parameters are accessed via the **General** tab, next to the **Rules** tab. You can change the IKE lifetime. If you click **Advanced**, you may edit the attributes in the IKE proposals.

Step 12 After you have finished defining your customized policy, return to the Console screen. Right-click the new policy and select **assign**. By doing this, you assign this policy to any adapter or connection you specified during the creation of the policy rules. For example, if you applied this rule to all remote-access connections, when you assign the policy, that rule is assigned to any L2TP connections initiated by this PC.

Configuring the Windows 2000 Client for L2TP

To complete the setup for L2TP over IPsec, you must set up L2TP on the Windows 2000 client. You must define and configure your L2TP connection by doing the following:

Step 1 Select either **Start > Settings > Control Panel > Network > Dial-up Connections** or **Start > Settings > Network > Dial-up Connections > Make New Connection**.

Step 2 Use the wizard to create a connection called L2TP. This connection connects to a private network through the Internet. You also need to specify the L2TP tunnel gateway's IP address or name (10.0.0.7 in this example).

Step 3 The new connection appears in the Network and Dial-up Connections window in Control Panel. Right-click it to edit its properties.

Step 4 Under the **Networking** tab, make sure that the 'Type of Server I am calling' is set to L2TP. If you plan on allocating a dynamic internal address to this client from the gateway, via either a local pool or DHCP, select TCP/IP protocol, thenmake sure that the client is configured to obtain an IP address automatically. You may also issue DNS information automatically. The **Advanced** button lets you define static WINS and DNS information, and the **Options** tab lets you turn off IPsec or assign a different policy to the connection.

Step 5 Under the **Security** tab, you can define the user authentication parameters, such as PAP, CHAP, MS-CHAP, or Windows domain logon.

Step 6 As soon as the connection is configured, double-click it to pop up the login screen, and then select **connect**.

During the connection lifetime, statistics are available via the VPN connection icon in the bottom-right corner of the toolbar. IPsec status is available through Ipsecmon.

Example 13-23 outlines the commands required for L2TP and IPsec on the router.

Example 13-23 *Configuration of the Router LNS*

```
LNS#write terminal
version 12.0
service timestamps debug uptime
service timestamps log uptime

!
hostname LNS
```

Example 13-23 *Configuration of the Router LNS (Continued)*

```
!
boot system flash c4500-is56-mz

!Enable AAA services here
aaa new-model
aaa authentication ppp default radius local
enable secret 5 <removed>
enable password cisco
!
username munchkin password 7 <removed>
ip subnet-zero
ip domain-name cisco.com
ip name-server 10.0.0.4
!
!Enable VPN/VPDN services and define groups and their particular variables
vpdn enable
!
vpdn-group 1
!Default L2TP VPDN group
!Allow the router to accept incoming requests.
 accept dialin l2tp virtual-template 1
!Tunnel authentication is used in the LNS to LAC, aka NAS-initiated L2TP tunnel.
!Users are authenticated at the NAS or LNS before the tunnel is established.
!So it is not required for client-initiated tunnels.
 no l2tp tunnel authentication
!

!These are the IKE policies the router will accept
crypto isakmp policy 4
!
crypto isakmp policy 5
 hash md5
!
crypto isakmp policy 6
 hash md5
 group 2
!
!Use this for IKE and wildcard preshared keys
crypto isakmp key ciscosys address 0.0.0.0
!
!
!Define the IPsec policies the router will accept/propose. For L2TP connections,
!the mode should be transport
crypto IPsec transform-set desmd5tr esp-des esp-md5-hmac
 mode transport
crypto IPsec transform-set ahshatr ah-sha-hmac
 mode transport
crypto IPsec transform-set ahmd5tr ah-md5-hmac
 mode transport
!
```

continues

Example 13-23 *Configuration of the Router LNS (Continued)*

```
!Set up a dynamic crypto map template
crypto dynamic-map dyna 1

set transform-set ahmd5tr
 match address 110
!
!If you are using RSA signatures, include your CA identity. This identity is
!configured to use the Microsoft CA with SCEP support.
!
crypto ca identity cisco.com
 enrollment retry count 100
 enrollment mode ra
 enrollment url http://cisco-b0tpppy88:80/certsrv/mscep/mscep.dll
 crl optional
!
!Certs removed for keeping the configs short
!
crypto ca certificate chain cisco.com
 certificate 615D1CBC000000000006
 certificate ra-sign 710685AF000000000004
 certificate 615D299B000000000007
 certificate ca 578242EB5428E68A4D02516EF449B266
 certificate ra-encrypt 71068B79000000000005
 !
!Apply the dynamic crypto map template to an actual crypto map
crypto map iosmsdyn 1 IPsec-isakmp dynamic dyna
!
!
interface Ethernet0
 ip address 10.0.0.7 255.255.255.0
 no ip redirects
 no ip directed-broadcast
 crypto map iosmsdyn
!
!
interface Virtual-Template1
 ip unnumbered Ethernet0
 no ip directed-broadcast
 peer default ip address pool vpdn
 ppp authentication ms-chap
!
router rip
 redistribute connected
 redistribute static
network 192.168.0.0
!
!Define your local address pool and any other dynamically assigned information
ip local pool vpdn 192.168.2.10 192.168.2.15
no ip classless
ip route 0.0.0.0 0.0.0.0 192.168.0.100
no ip http server
```

Example 13-23 *Configuration of the Router LNS (Continued)*

```
!
access-list 110 permit udp host 192.168.0.7 any eq 1701
access-list 110 permit udp any eq 1701 host 192.168.0.7
access-list 110 permit udp host 192.168.0.7 eq 1701 any
!
!
radius-server host 192.168.0.4 auth-port 1645 acct-port 1646
radius-server key kitchen
!
line con 0
 transport input none
line aux 0
line vty 0 4
 password cisco
!
end
```

Tunnel Endpoint Discovery (TED) with IPsec

This case study covers a special feature available in Cisco router implementations. TED allows a router to dynamically configure its IPsec peer address without your having to configure it manually in the router configs. This is a useful scalability feature that lets you set up a large number of peers simply by defining an access list for their interesting traffic and then allowing TED to find out who the peers are. The catch is that the interesting traffic must be defined using globally routable addresses if the VPN is being set up over the Internet. This is necessary because TED uses normal routing to figure out where the IPsec peers are.

TED works by using a probe packet that is sent with the destination address set to the IP address defined as the destination IP in the packet received on the router from a network behind the router. This probe is intercepted by the IPsec router sitting in front of the destination IP address. This router gleans the necessary information about the proxy IDs that prompted the first router to send out a probe. Then it sends back a probe reply with proxy IDs that are either the same as the ones that were sent by the initiator or are a subset of the those. These proxy IDs are the ones that are used by the initiator to negotiate the IPsec tunnel.

Figure 13-53 shows the process through which a router discovers its IPsec peer after being configured with only an access list specifying the interesting traffic for IPsec.

Figure 13-54 shows the format of the TED probe that a router sends to try and discover its peer and the probe response that is sent in response by the discovered peer. The ID payloads contain the source addresses configured on the router as interesting traffic (source proxy ID) and the ID of the initiator coded as its IP address. This allows the peer intercepting the probe to know which address it is supposed to send the probe reply to. Examples 13-24 and 13-25 show how to set up routers to do TED discovery. Examples 13-26 through 13-29 show various debugs and **show** commands for the successful setup. Figure 13-55 shows the network diagram for this case study.

Figure 13-53 *TED Discovery Process*

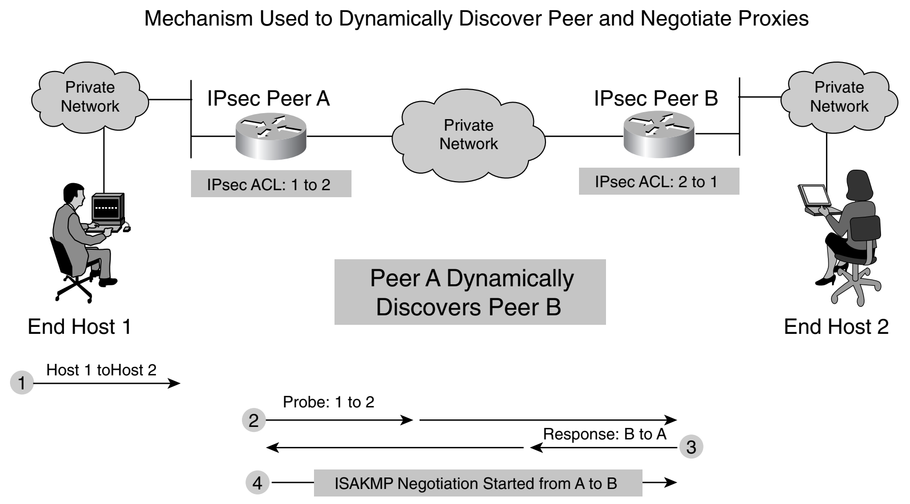


Figure 13-54 *TED Probe and TED Probe Response Packet Structure*

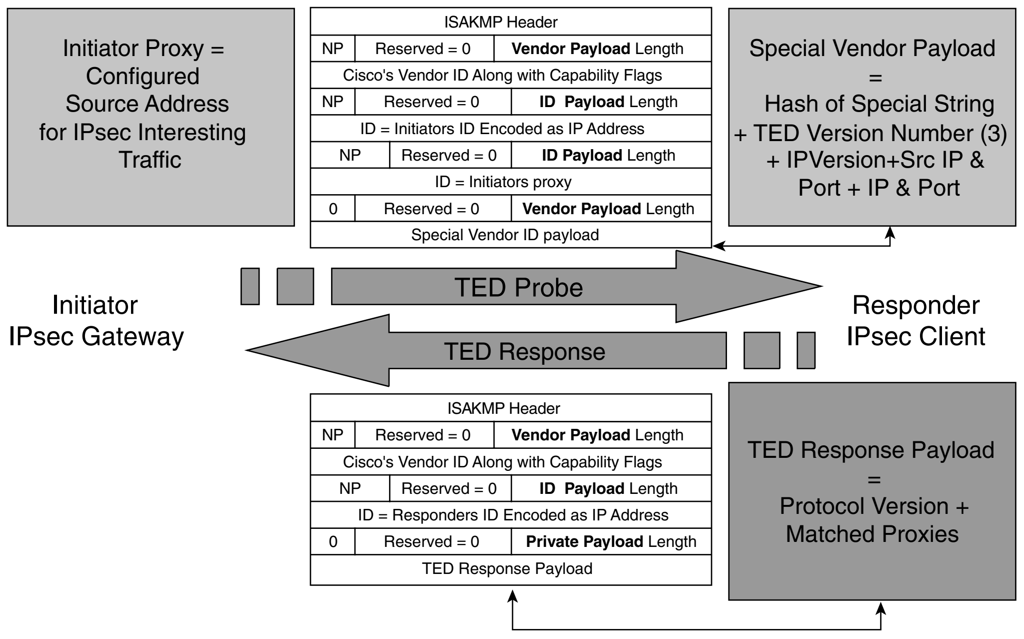
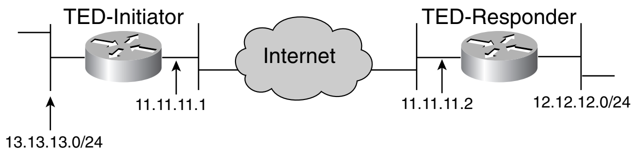

Initiator Proxy = Configured Source Address for IPsec Interesting Traffic
ISAKMP Header
NP | Reserved = 0 | Vendor Payload Length
Cisco's Vendor ID Along with Capability Flags
NP | Reserved = 0 | ID Payload Length
ID = Initiators ID Encoded as IP Address
NP | Reserved = 0 | ID Payload Length
ID = Initiators proxy
0 | Reserved = 0 | Vendor Payload Length
Special Vendor ID payload
Special Vendor Payload = Hash of Special String + TED Version Number (3) + IPVersion+Src IP & Port + IP & Port
Initiator IPsec Gateway
TED Probe
TED Response
Responder IPsec Client
ISAKMP Header
NP | Reserved = 0 | Vendor Payload Length
Cisco's Vendor ID Along with Capability Flags
NP | Reserved = 0 | ID Payload Length
ID = Responders ID Encoded as IP Address
0 | Reserved = 0 | Private Payload Length
TED Response Payload
TED Response Payload = Protocol Version + Matched Proxies

Figure 13-55 *Network Diagram for This Case Study*



Example 13-24 *Configuration of the Router TED-Initiator*

```
TED-Initiator#show running-config
Building configuration...

Current configuration:
!
version 12.0
service timestamps debug uptime
service timestamps log uptime

!
hostname TED-Initiator
!
enable secret 5 <removed>
enable password <removed>
!
ip subnet-zero
!
crypto isakmp policy 10
 authentication pre-share
!One of the issues with using a preshared key with TED is the need to use a
!wildcard preshared key because the peer's address is not known beforehand. A
!resolution to this is to use digital certificate-based digital signatures as the
!authentication method.
crypto isakmp key abc123 address 0.0.0.0
!
crypto ipsec transform-set ted-transforms esp-des esp-md5-hmac
!Note that no peer address has been configured in the crypto map below.
crypto dynamic-map ted-map 10
 set transform-set ted-transforms
 match address 101
!
!The keyword discover in the crypto map below triggers the use of TED

crypto map tedtag 10 ipsec-isakmp dynamic ted-map discover
!
interface Ethernet0/0
 ip address 13.13.13.13 255.255.255.0
 no ip directed-broadcast
 no mop enabled
```

continues

Example 13-24 *Configuration of the Router TED-Initiator (Continued)*

```
!
interface Ethernet0/1
 ip address 11.11.11.1 255.255.255.0
 crypto map tedtag
!
ip classless
ip route 0.0.0.0 0.0.0.0 11.11.11.2
no ip http server
!
access-list 101 permit ip 13.13.13.0 0.0.0.255 12.12.12.0 0.0.0.255
access-list 101 permit icmp 13.13.13.0 0.0.0.255 12.12.12.0 0.0.0.255
!
line con 0
 transport input none
line aux 0
line vty 0 4
 password ww
 login
!
end
```

Example 13-25 *Configuration of the Router TED-Responder*

```
TED-Responder#show running-config
Building configuration...

Current configuration:
!
version 12.0
service timestamps debug uptime
service timestamps log uptime

!
hostname TED-Responder
!
enable secret 5 <removed>
enable password <removed>
!

!
crypto isakmp policy 10
 authentication pre-share
crypto isakmp key abc123 address 0.0.0.0
!
crypto ipsec transform-set ted-transforms esp-des esp-md5-hmac
!
crypto dynamic-map ted-map 10
 set transform-set ted-transforms
 match address 101
```

Example 13-25 *Configuration of the Router TED-Responder (Continued)*

```
!
crypto map tedtag 10 ipsec-isakmp dynamic ted-map discover
!
interface Ethernet0/0
 ip address 12.12.12.12 255.255.255.0
 no ip directed-broadcast
 no mop enabled
!
interface Ethernet0/1
 ip address 11.11.11.2 255.255.255.0
 crypto map tedtag
!
ip classless
ip route 0.0.0.0 0.0.0.0 11.11.11.1
no ip http server
!
access-list 101 permit ip 12.12.12.0 0.0.0.255 13.13.13.0 0.0.0.255
access-list 101 permit icmp 12.12.12.0 0.0.0.255 13.13.13.0 0.0.0.255

!
line con 0
 transport input none
line aux 0
line vty 0 4
 password ww
 login
!
no scheduler allocate
end
```

Example 13-26 *Debugs on the Router TED-Initiator*

```
TED-Initiator#show debug
Cryptographic Subsystem:
  Crypto ISAKMP debugging is on
  Crypto Engine debugging is on
  Crypto IPSEC debugging is on
TED-Initiator#
!The TED process has started. The proxy IDs are shown in the message below.
01:33:56: IPSEC(tunnel discover request): ,
  (key eng. msg.) src= 13.13.13.14, dest= 12.12.12.13,
    src_proxy= 13.13.13.0/255.255.255.0/0/0 (type=4),
    dest_proxy= 11.11.11.1/255.255.255.255/0/0 (type=1),
    protocol= ESP, transform= esp-des esp-md5-hmac ,
    lifedur= 3600s and 4608000kb,
    spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x4044
01:33:56: GOT A PEER DISCOVERY MESSAGE FROM THE SA MANAGER!!!
01:33:56: src = 13.13.13.14 to 12.12.12.13, protocol 3, transform 2, hmac 1
```

continues

Example 13-26 *Debugs on the Router TED-Initiator (Continued)*

```
!The TED process below determines which address is the source address in the TED
!packet and which addresses are the proxy IDs.
01:33:56: proxy source is 13.13.13.0     /255.255.255.0   and my address (not used
  now) is 11.11.11.1
01:33:56: ISAKMP (1): ID payload
        next-payload : 5
        type         : 1
        protocol     : 17
        port         : 500
        length       : 8
01:33:56: ISAKMP (1): Total payload length: 12
!The initiator determines below that the first ID payload will be its own IP
!address, 11.11.11.1, and the second payload contains its source IPsec proxy:
!13.13.13.0/24
01:33:56: 1st ID is 11.11.11.1
01:33:56: 2nd ID is 13.13.13.0     /255.255.255.0
01:33:56: ISAKMP (0:1): beginning peer discovery exchange
!The TED probe is being sent to the destination IP address found in the original
!packet that was received on the initiator and that matched the IPsec interesting
!traffic access list.
01:33:56: ISAKMP (1): sending packet to 12.12.12.13 (I) PEER_DISCOVERY
!The peer has been discovered to be 12.12.12.13, and it responds as shown below
01:33:56: ISAKMP (1): received packet from 12.12.12.13 (I) PEER_DISCOVERY
!Upon processing the vendor ID payload, the initiator ascertains that the
!responder does indeed understand what was sent to it.
01:33:56: ISAKMP (0:1): processing vendor id payload
01:33:56: ISAKMP (0:1): speaking to another IOS box!
01:33:56: ISAKMP (0:1): processing ID payload. message ID = 0
!The responder's IP address is encoded in the ID payload. It is equal to 11.11.11.2
01:33:56: ISAKMP (0:1): processing ID payload. message ID = 1168952014
!Upon looking at the ID payload sent by the responder, the initiator finds that
!the responder's proxy ID indeed matches the proxy configured on itself.
01:33:56: ISAKMP (1): ID_IPV4_ADDR_SUBNET dst 12.12.12.0/255.255.255.0 prot 0 port 0
01:33:56: ISAKMP (1): received response to my peer discovery probe!
!Normal IKE processing starts at this point to the IP address discovered through
!TED
01:33:56  ISAKMP: initiating IKE to 11.11.11.2 in response to probe.
01:33:56: ISAKMP (2): sending packet to 11.11.11.2 (I) MM_NO_STATE
01:33:56: ISAKMP (0:1): deleting SA
01:33:56: ISAKMP (2): received packet from 11.11.11.2 (I) MM_NO_STATE
01:33:56: ISAKMP (0:2): processing SA payload. message ID = 0
01:33:56: ISAKMP (0:2): Checking ISAKMP transform 1 against priority 10 policy
01:33:56: ISAKMP:      encryption DES-CBC
01:33:56: ISAKMP:      hash SHA
01:33:56: ISAKMP:      default group 1
01:33:56: ISAKMP:      auth pre-share
01:33:56: ISAKMP (0:2): atts are acceptable. Next payload is 0
01:33:56: CryptoEngine0: generate alg parameter
01:33:56: CRYPTO_ENGINE: Dh phase 1 status: 0
01:33:56: CRYPTO_ENGINE: Dh phase 1 status: 0
01:33:56: ISAKMP (0:2): SA is doing pre-shared key authentication
```

Example 13-26 *Debugs on the Router TED-Initiator (Continued)*

```
01:33:56: ISAKMP (2): SA is doing pre-shared key authentication using id type
  ID_IPV4_ADDR
01:33:56: ISAKMP (2): sending packet to 11.11.11.2 (I) MM_SA_SETUP
01:33:56: ISAKMP (2): received packet from 11.11.11.2 (I) MM_SA_SETUP
01:33:56: ISAKMP (0:2): processing KE payload. message ID = 0
01:33:56: CryptoEngine0: generate alg parameter
01:33:57: ISAKMP (0:2): processing NONCE payload. message ID = 0
01:33:57: CryptoEngine0: create ISAKMP SKEYID for conn id 2
01:33:57: ISAKMP (0:2): SKEYID state generated
01:33:57: ISAKMP (0:2): processing vendor id payload
01:33:57: ISAKMP (0:2): speaking to another IOS box!
01:33:57: ISAKMP (2): ID payload
        next-payload : 8
        type         : 1
        protocol     : 17
        port         : 500
        length       : 8
01:33:57: ISAKMP (2): Total payload length: 12
01:33:57: CryptoEngine0: generate hmac context for conn id 2
01:33:57: ISAKMP (2): sending packet to 11.11.11.2 (I) MM_KEY_EXCH
01:33:57: ISAKMP (2): received packet from 11.11.11.2 (I) MM_KEY_EXCH
01:33:57: ISAKMP (0:2): processing ID payload. message ID = 0
01:33:57: ISAKMP (0:2): processing HASH payload. message ID = 0
01:33:57: CryptoEngine0: generate hmac context for conn id 2
01:33:57: ISAKMP (0:2): SA has been authenticated with 11.11.11.2
01:33:57: ISAKMP (0:2): beginning Quick Mode exchange, M-ID of 474637101
01:33:57: CryptoEngine0: clear dh number for conn id 1
01:33:57: IPSEC(key_engine): got a queue event...
01:33:57: IPSEC(spi_response): getting spi 348588451 for SA
        from 11.11.11.2      to 11.11.11.1      for prot 3
01:33:57: CryptoEngine0: generate hmac context for conn id 2
01:33:57: ISAKMP (2): sending packet to 11.11.11.2 (I) QM_IDLE
01:33:57: ISAKMP (2): received packet from 11.11.11.2 (I) QM_IDLE
01:33:57: CryptoEngine0: generate hmac context for conn id 2
01:33:57: ISAKMP (0:2): processing SA payload. message ID = 474637101
01:33:57: ISAKMP (0:2): Checking IPsec proposal 1
01:33:57: ISAKMP: transform 1, ESP_DES
01:33:57: ISAKMP:   attributes in transform:
01:33:57: ISAKMP:      encaps is 1
01:33:57: ISAKMP:      SA life type in seconds
01:33:57: ISAKMP:      SA life duration (basic) of 3600
01:33:57: ISAKMP:      SA life type in kilobytes
01:33:57: ISAKMP:      SA life duration (VPI) of  0x0 0x46 0x50 0x0
01:33:57: ISAKMP:      authenticator is HMAC-MD5
01:33:57: validate proposal 0
01:33:57: ISAKMP (0:2): atts are acceptable.
01:33:57: IPSEC(validate_proposal_request): proposal part #1,
  (key eng. msg.) dest= 11.11.11.2, src= 11.11.11.1,
    dest_proxy= 12.12.12.0/255.255.255.0/0/0 (type=4),
    src_proxy= 13.13.13.0/255.255.255.0/0/0 (type=4),
```

continues

Example 13-26 *Debugs on the Router TED-Initiator (Continued)*

```
    protocol= ESP, transform= esp-des esp-md5-hmac ,
    lifedur= 0s and 0kb,
    spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x4
01:33:57: validate proposal request 0
01:33:57: ISAKMP (0:2): processing NONCE payload. message ID = 474637101
01:33:57: ISAKMP (0:2): processing ID payload. message ID = 474637101
01:33:57: ISAKMP (0:2): processing ID payload. message ID = 474637101
01:33:57: CryptoEngine0: generate hmac context for conn id 2
01:33:57: ipsec allocate flow 0
01:33:57: ipsec allocate flow 0
01:33:57: ISAKMP (0:2): Creating IPsec SAs
01:33:57:         inbound SA from 11.11.11.2 to 11.11.11.1      (proxy 12.12.12.0
  to 13.13.13.0)
01:33:57:         has spi 348588451 and conn_id 2000 and flags 4
01:33:57:         lifetime of 3600 seconds
01:33:57:         lifetime of 4608000 kilobytes
01:33:57:         outbound SA from 11.11.11.1 to 11.11.11.2      (proxy 13.13.13.0
  to 12.12.12.0)
01:33:57:         has spi 132187477 and conn_id 2001 and flags 4
01:33:57:         lifetime of 3600 seconds
01:33:57:         lifetime of 4608000 kilobytes
01:33:57: ISAKMP (2): sending packet to 11.11.11.2 (I) QM_IDLE
01:33:57: ISAKMP (0:2): deleting node 474637101
01:33:57: IPSEC(key_engine): got a queue event...
01:33:57: IPSEC(initialize_sas): ,
  (key eng. msg.) dest= 11.11.11.1, src= 11.11.11.2,
    dest_proxy= 13.13.13.0/255.255.255.0/0/0 (type=4),
    src_proxy= 12.12.12.0/255.255.255.0/0/0 (type=4),
    protocol= ESP, transform= esp-des esp-md5-hmac ,
    lifedur= 3600s and 4608000kb,
    spi= 0x14C709A3(348588451), conn_id= 2000, keysize= 0, flags= 0x4
01:33:57: IPSEC(initialize_sas): ,
  (key eng. msg.) src= 11.11.11.1, dest= 11.11.11.2,
    src_proxy= 13.13.13.0/255.255.255.0/0/0 (type=4),
    dest_proxy= 12.12.12.0/255.255.255.0/0/0 (type=4),
    protocol= ESP, transform= esp-des esp-md5-hmac ,
    lifedur= 3600s and 4608000kb,
    spi= 0x7E10555(132187477), conn_id= 2001, keysize= 0, flags= 0x4
01:33:57: IPSEC(create_sa): sa created,
  (sa) sa_dest= 11.11.11.1, sa_prot= 50,
    sa_spi= 0x14C709A3(348588451),
    sa_trans= esp-des esp-md5-hmac , sa_conn_id= 2000
01:33:57: IPSEC(create_sa): sa created,
  (sa) sa_dest= 11.11.11.2, sa_prot= 50,
    sa_spi= 0x7E10555(132187477),
    sa_trans= esp-des esp-md5-hmac , sa_conn_id= 2001
```

Example 13-27 *Output of the* **show** *Commands on the Router TED-Initiator*

```
TED-Initiator#show crypto ipsec sa

interface: Ethernet0/1
    Crypto map tag: tedtag, local addr. 11.11.11.1

   local  ident (addr/mask/prot/port): (13.13.13.0/255.255.255.0/0/0)
   remote ident (addr/mask/prot/port): (12.12.12.0/255.255.255.0/0/0)
   current_peer: 11.11.11.2
     PERMIT, flags={}
    #pkts encaps: 9, #pkts encrypt: 9, #pkts digest 9
    #pkts decaps: 9, #pkts decrypt: 9, #pkts verify 9
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0, #pkts decompress failed: 0
    #send errors 0, #recv errors 0

     local crypto endpt.: 11.11.11.1, remote crypto endpt.: 11.11.11.2
     path mtu 1500, media mtu 1500
     current outbound spi: 7E10555

     inbound esp sas:
      spi: 0x14C709A3(348588451)
        transform: esp-des esp-md5-hmac ,
        in use settings ={Tunnel, }
        slot: 0, conn id: 2000, flow_id: 1, crypto map: tedtag
        sa timing: remaining key lifetime (k/sec): (4607998/3557)
        IV size: 8 bytes
        replay detection support: Y

     inbound ah sas:

     inbound pcp sas:

     outbound esp sas:
      spi: 0x7E10555(132187477)
        transform: esp-des esp-md5-hmac ,
        in use settings ={Tunnel, }
        slot: 0, conn id: 2001, flow_id: 2, crypto map: tedtag
        sa timing: remaining key lifetime (k/sec): (4607998/3557)
        IV size: 8 bytes
        replay detection support: Y

     outbound ah sas:

     outbound pcp sas:
```

Example 13-28 *Debugs on the Router TED-Responder*

```
TED-Responder#show debug
Cryptographic Subsystem:
  Crypto ISAKMP debugging is on
  Crypto Engine debugging is on
  Crypto IPSEC debugging is on
TED-Responder#
!The responder has just received the TED probe and is trying to figure out if it
!is an ISAKMP initialization offer.
00:40:16: %CRYPTO-4-IKMP_NO_SA: IKE message from 1.204.0.4 has no SA and is not an
initialization offer
00:40:16: ISAKMP (0): received packet from 13.13.13.14 (N) NEW SA
00:40:16: ISAKMP (0:1): processing vendor id payload
!The responder has figured out by looking at the vendor ID payload that the other
!side is an IOS box. Now it also understands how to respond to the message.
00:40:16: ISAKMP (0:1): speaking to another IOS box!
00:40:16: ISAKMP (0:1): processing ID payload. message ID = 0
00:40:16: ISAKMP (0:1): processing ID payload. message ID = -1418395956
!By looking at the ID payload, the responder finds out that the source proxy for
!the initiator is 13.13.13.0
00:40:16: ISAKMP (1): ID_IPV4_ADDR_SUBNET src 13.13.13.0/255.255.255.0 prot 0 port 0
!The responder sends back a response to the TED probe, putting its IP address and
!its proxy in it. Below, the responder formulates the message to send to the
!initiator.
00:40:16: ISAKMP (1): responding to peer discovery probe!
00:40:16: peer's address is 11.11.11.1
00:40:16: src (him) 4, 13.13.13.0     /255.255.255.0   to dst (me) 4,
  12.12.12.0     /255.255.255.0
00:40:16: ISAKMP (1): ID payload
        next-payload : 5
        type         : 4
        protocol     : 17
        port         : 500
        length       : 12
00:40:16: ISAKMP (1): Total payload length: 16
00:40:16: ISAKMP (1): sending packet to 13.13.13.14 (R) PEER_DISCOVERY
00:40:16: ISAKMP (0:1): deleting SA
!Normal IKE processing starts on receiving a request from the initiator.
00:40:16: ISAKMP (0): received packet from 11.11.11.1 (N) NEW SA
00:40:16: ISAKMP (0:2): processing SA payload. message ID = 0
00:40:16: ISAKMP (0:2): Checking ISAKMP transform 1 against priority 10 policy
00:40:16: ISAKMP:      encryption DES-CBC
00:40:16: ISAKMP:      hash SHA
00:40:16: ISAKMP:      default group 1
00:40:16: ISAKMP:      auth pre-share
00:40:16: ISAKMP (0:2): atts are acceptable. Next payload is 0
00:40:16: CryptoEngine0: generate alg parameter
00:40:17: CRYPTO_ENGINE: Dh phase 1 status: 0
00:40:17: CRYPTO_ENGINE: Dh phase 1 status: 0
00:40:17: ISAKMP (0:2): SA is doing pre-shared key authentication
00:40:17: ISAKMP (2): SA is doing pre-shared key authentication using id
  type ID_IPV4_ADDR
```

Example 13-28 *Debugs on the Router TED-Responder (Continued)*

```
00:40:17: ISAKMP (2): sending packet to 11.11.11.1 (R) MM_SA_SETUP
00:40:17: ISAKMP (2): received packet from 11.11.11.1 (R) MM_SA_SETUP
00:40:17: ISAKMP (0:2): processing KE payload. message ID = 0
00:40:17: CryptoEngine0: generate alg parameter
00:40:17: ISAKMP (0:2): processing NONCE payload. message ID = 0
00:40:17: CryptoEngine0: create ISAKMP SKEYID for conn id 2
00:40:17: ISAKMP (0:2): SKEYID state generated
00:40:17: ISAKMP (0:2): processing vendor id payload
00:40:17: ISAKMP (0:2): speaking to another IOS box!
00:40:17: ISAKMP (2): sending packet to 11.11.11.1 (R) MM_KEY_EXCH
00:40:17: ISAKMP (2): received packet from 11.11.11.1 (R) MM_KEY_EXCH
00:40:17: ISAKMP (0:2): processing ID payload. message ID = 0
00:40:17: ISAKMP (0:2): processing HASH payload. message ID = 0
00:40:17: CryptoEngine0: generate hmac context for conn id 2
00:40:17: ISAKMP (0:2): SA has been authenticated with 11.11.11.1
00:40:17: ISAKMP (2): ID payload
        next-payload : 8
        type         : 1
        protocol     : 17
        port         : 500
        length       : 8
00:40:17: ISAKMP (2): Total payload length: 12
00:40:17: CryptoEngine0: generate hmac context for conn id 2
00:40:17: CryptoEngine0: clear dh number for conn id 1
00:40:17: ISAKMP (2): sending packet to 11.11.11.1 (R) QM_IDLE
00:40:17: ISAKMP (2): received packet from 11.11.11.1 (R) QM_IDLE
00:40:17: CryptoEngine0: generate hmac context for conn id 2
00:40:18: ISAKMP (0:2): processing SA payload. message ID = 474637101
00:40:18: ISAKMP (0:2): Checking IPsec proposal 1
00:40:18: ISAKMP: transform 1, ESP_DES
00:40:18: ISAKMP:   attributes in transform:
00:40:18: ISAKMP:      encaps is 1
00:40:18: ISAKMP:      SA life type in seconds
00:40:18: ISAKMP:      SA life duration (basic) of 3600
00:40:18: ISAKMP:      SA life type in kilobytes
00:40:18: ISAKMP:      SA life duration (VPI) of  0x0 0x46 0x50 0x0
00:40:18: ISAKMP:      authenticator is HMAC-MD5
00:40:18: validate proposal 0
00:40:18: ISAKMP (0:2): atts are acceptable.
00:40:18: IPSEC(validate_proposal_request): proposal part #1,
  (key eng. msg.) dest= 11.11.11.2, src= 11.11.11.1,
    dest_proxy= 12.12.12.0/255.255.255.0/0/0 (type=4),
    src_proxy= 13.13.13.0/255.255.255.0/0/0 (type=4),
    protocol= ESP, transform= esp-des esp-md5-hmac ,
    lifedur= 0s and 0kb,
    spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x4
00:40:18: validate proposal request 0
00:40:18: ISAKMP (0:2): processing NONCE payload. message ID = 474637101
00:40:18: ISAKMP (0:2): processing ID payload. message ID = 474637101
00:40:18: ISAKMP (2): ID_IPV4_ADDR_SUBNET src 13.13.13.0/255.255.255.0 prot 0
  port 0
```

continues

Example 13-28 *Debugs on the Router TED-Responder (Continued)*

```
00:40:18: ISAKMP (0:2): processing ID payload. message ID = 474637101
00:40:18: ISAKMP (2): ID_IPV4_ADDR_SUBNET dst 12.12.12.0/255.255.255.0 prot 0
  port 0
00:40:18: IPSEC(key_engine): got a queue event...
00:40:18: IPSEC(spi_response): getting spi 132187477 for SA
        from 11.11.11.1      to 11.11.11.2      for prot 3
00:40:18: CryptoEngine0: generate hmac context for conn id 2
00:40:18: ISAKMP (2): sending packet to 11.11.11.1 (R) QM_IDLE
00:40:18: ISAKMP (2): received packet from 11.11.11.1 (R) QM_IDLE
00:40:18: CryptoEngine0: generate hmac context for conn id 2
00:40:18: ipsec allocate flow 0
00:40:18: ipsec allocate flow 0
00:40:18: ISAKMP (0:2): Creating IPsec SAs
00:40:18:         inbound SA from 11.11.11.1  to 11.11.11.2      (proxy 13.13.13.0
  to 12.12.12.0)
00:40:18:         has spi 132187477 and conn_id 2000 and flags 4
00:40:18:         lifetime of 3600 seconds
00:40:18:         lifetime of 4608000 kilobytes
00:40:18:         outbound SA from 11.11.11.2 to 11.11.11.1      (proxy 12.12.12.0
  to 13.13.13.0)
00:40:18:         has spi 348588451 and conn_id 2001 and flags 4
00:40:18:         lifetime of 3600 seconds
00:40:18:         lifetime of 4608000 kilobytes
00:40:18: ISAKMP (0:2): deleting node 474637101
00:40:18: IPSEC(key_engine): got a queue event...
00:40:18: IPSEC(initialize_sas): ,
  (key eng. msg.) dest= 11.11.11.2, src= 11.11.11.1,
    dest_proxy= 12.12.12.0/255.255.255.0/0/0 (type=4),
    src_proxy= 13.13.13.0/255.255.255.0/0/0 (type=4),
    protocol= ESP, transform= esp-des esp-md5-hmac ,
    lifedur= 3600s and 4608000kb,
    spi= 0x7E10555(132187477), conn_id= 2000, keysize= 0, flags= 0x4
00:40:18: IPSEC(initialize_sas): ,
  (key eng. msg.) src= 11.11.11.2, dest= 11.11.11.1,
    src_proxy= 12.12.12.0/255.255.255.0/0/0 (type=4),
    dest_proxy= 13.13.13.0/255.255.255.0/0/0 (type=4),
    protocol= ESP, transform= esp-des esp-md5-hmac ,
    lifedur= 3600s and 4608000kb,
    spi= 0x14C709A3(348588451), conn_id= 2001, keysize= 0, flags= 0x4
00:40:18: IPSEC(create_sa): sa created,
  (sa) sa_dest= 11.11.11.2, sa_prot= 50,
    sa_spi= 0x7E10555(132187477),
    sa_trans= esp-des esp-md5-hmac , sa_conn_id= 2000
00:40:18: IPSEC(create_sa): sa created,
  (sa) sa_dest= 11.11.11.1, sa_prot= 50,
    sa_spi= 0x14C709A3(348588451),
    sa_trans= esp-des esp-md5-hmac , sa_conn_id= 2001
```

Example 13-29 *Output of the* **show** *Commands on the Router TED-Responder*

```
TED-Responder#show crypto ipsec sa

interface: Ethernet0/1
    Crypto map tag: tedtag, local addr. 11.11.11.2

   local  ident (addr/mask/prot/port): (12.12.12.0/255.255.255.0/0/0)
   remote ident (addr/mask/prot/port): (13.13.13.0/255.255.255.0/0/0)
   current_peer: 11.11.11.1
     PERMIT, flags={}
    #pkts encaps: 9, #pkts encrypt: 9, #pkts digest 9
    #pkts decaps: 9, #pkts decrypt: 9, #pkts verify 9
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0, #pkts decompress failed: 0
    #send errors 0, #recv errors 0

     local crypto endpt.: 11.11.11.2, remote crypto endpt.: 11.11.11.1
     path mtu 1500, media mtu 1500
     current outbound spi: 14C709A3

     inbound esp sas:
      spi: 0x7E10555(132187477)
        transform: esp-des esp-md5-hmac ,
        in use settings ={Tunnel, }
        slot: 0, conn id: 2000, flow_id: 1, crypto map: tedtag
        sa timing: remaining key lifetime (k/sec): (4607998/3451)
        IV size: 8 bytes
        replay detection support: Y

     inbound ah sas:

     inbound pcp sas:

     outbound esp sas:
      spi: 0x14C709A3(348588451)
        transform: esp-des esp-md5-hmac ,
        in use settings ={Tunnel, }
        slot: 0, conn id: 2001, flow_id: 2, crypto map: tedtag
        sa timing: remaining key lifetime (k/sec): (4607998/3433)
        IV size: 8 bytes
        replay detection support: Y

     outbound ah sas:
     outbound pcp sas:
```

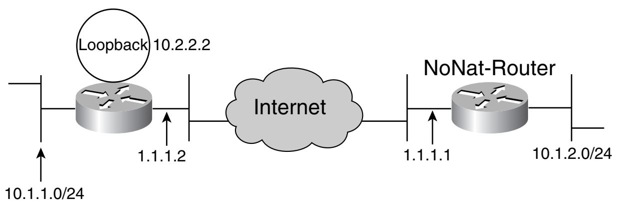
NAT's Interaction with IPsec

Because IPsec in tunnel mode hides private IP addresses, the need to do NAT for traffic going into an IPsec tunnel does not arise. Indeed, sometimes it is required that you *not* do NAT for traffic entering the tunnel. This is true in situations where the network administrator of two networks connected via a VPN wants to allow the users of both the networks to access all the resources on both networks using the same IP addresses. This means that if the users on Network A are accessing a server on Network A using the IP address 10.1.1.1, the users on Network B connected to Network A via a VPN should also be able to connect to this server using the IP address 10.1.1.1, despite the fact that the IPsec router in front of Network A has a static NAT translation for the server in place to allow connectivity to it from the Internet.

The configuration in Example 13-30 (which is stripped of unnecessary information) shows a method of bypassing NAT entering an IPsec tunnel. The goal of bypassing IPsec is achieved using a special policy-routing trick. It uses the rule that NAT, whether static or dynamic, kicks in only when NAT interesting traffic goes from an interface configured with the **ip nat inside** command directly to an interface set up with the **ip nat outside** command. In Example 13-30, this flow is broken by policy-routing the traffic headed for the other end of the IPsec tunnel to a loopback interface and then out the egress interface. This forces NAT not to occur for this traffic.

Example 13-30 shows the configuration needed to setup the 'NATRTR' for bypassing NAT. Figure 13-56 shows the network diagram for this case study.

Figure 13-56 *Network Diagram for This Case Study*



Example 13-30 *Configuration of the Router NATRTR Used in This Case Study*

```
Router#write terminal

hostname NATRTR

crypto map test 10 IPsec-isakmp
  set peer 1.1.1.1
 set transform-set transform
 match address 100
!This is the loopback the traffic will be routed to in order to change the order
!of events on the router
interface Loopback1

ip address 10.2.2.2 255.255.255.252

interface Ethernet0/0
 ip address 1.1.1.2 255.255.255.0
```

Example 13-30 *Configuration of the Router NATRTR Used in This Case Study (Continued)*

```
 ip nat outside
 crypto map test

interface Ethernet0/1
 ip address 10.1.1.1 255.255.255.0
 ip nat inside
 ip route-cache policy
!The policy route map below is used to force the IPsec interesting traffic to the
!loopback interface.
 ip policy route-map nonat
!This is the dynamic NAT configuration we are trying to bypass.
ip nat inside source access-list 1 interface Ethernet0/0 overload
!This is the static NAT entry we are trying to bypass.
ip nat inside source static 10.1.1.2 100.1.1.3
access-list 1 permit 10.0.0.0 0.255.255.255
!The access list below defines IPsec interesting traffic.
access-list 100 permit ip 10.1.1.0 0.0.0.255 10.1.2.0 0.0.0.255
!The access list below defines the traffic that is to be used by the route map
!nonat to route to the loopback interface.
access-list 120 permit ip 10.1.1.0 0.0.0.255 10.1.2.0 0.0.0.255
route-map nonat
  permit 10
!Below is the route map used to route the traffic matching access list 120 to the
!loopback interface.
route-map nonat permit 10
 match ip address 120
 set ip next-hop 10.2.2.1
```

As shown in Example 13-21, the PIX Firewall allows NAT to be bypassed using the **nat 0** command.

Interaction of Firewalls with IPsec

Two holes need to be opened to allow IPsec to work through a firewall:

- ESP or/and AH (protocols 50 and 51, respectively)
- UDP port 500 (for ISAKMP)

In addition, if UDP encapsulation is being used for NAT traversal, the UDP port being used for this purpose also needs to be allowed through the firewall. UDP encapsulation is currently not a feature supported on Cisco routers or PIX firewalls.

In case the firewall is configured on the router terminating the IPsec tunnel itself, the firewall needs to be opened for ESP/AH and ISAKMP traffic as well as the IP address pair in the decrypted IP traffic. The reason for this is that the incoming access list is processed twice for the incoming IPsec traffic. It is processed once for the encrypted traffic in the form of an ESP/AH packet, and then after the packet is decrypted, the access list is processed again. Because the packet has been decrypted, a hole in the access list needs to be opened for the IP addresses of the decrypted IP packet. This should not represent a security hole, because the IPsec routine drops any packets arriving at the incoming interface with the IP addresses in the header matching the IPsec interesting traffic access list. So, although there is a hole in the access list for these IP addresses, the IPsec routine guards it.

Summary

This chapter discussed in detail the workings and implementation of the IPsec protocol. IPsec is fast becoming the de facto VPN standard for the industry. The various ways it can be implemented to suit customer needs are aiding its popularity. We discussed in detail how the three main protocols in IPsec function. In addition, we looked at the authentication mechanism employed by IKE and how encryption and integrity checking in IPsec take place. It is important to look at the case studies presented in this chapter in light of the explanations given for the functioning of IKE and the other protocols in IPsec. You saw some of the most common scenarios in which IPsec is deployed. Because IPsec is becoming very widely deployed, its high availability, its interaction with other protocols, and its ability to seamlessly integrate with the existing network environment are constantly being tested. This is why it is important to understand the basic protocol and the workings of IPsec in addition to knowing how the configurations work. Only then can you have a complete understanding of the involved protocol that IPsec is. We will revisit IPsec in Chapter 24, "Troubleshooting Network Security Implementations," to look at some common troubleshooting techniques to use in IPsec implementations.

Review Questions

1. What does IPsec stand for?
2. What three main protocols form IPsec?
3. What is the main difference between AH and ESP's capabilities?
4. Which phase does ISAKMP go through in negotiating IPsec SAs?
5. Can an IPsec crypto map point to more than one peer?
6. What is the purpose of TED?
7. What is the purpose of the **group** command?
8. Which command defines the encryption method to be used?
9. What does IPsec use to identify L2TP as interesting traffic?
10. Does GRE encapsulate IPsec traffic, or the other way around?