CHAPTER 7

Advanced Security Services, Part I: IPsec

The Internet provides a wealth of services for any organization that chooses to connect to it. No longer are organizations confined to the connectivity and information available in their private networks. Instead, a huge infrastructure and vast repositories of data can be tapped into simply by connecting to a local service provider. In a sense, the Internet becomes an extension, a very large extension, of the private network.

On the other hand, the Internet is a very public place. Data exchanged between the private network and the Internet is generally nonconfidential, public information. Internal data generally stays confined to the private network because to send it through the Internet would make it available for the public to see.

But what if you could securely send internal data *through* the Internet to another private location (like a branch office)? Then, you could capitalize on the size and reach of the Internet and make the Internet *your* infrastructure for interconnecting your private locations. Instead of building your own worldwide network, you could use the worldwide network of the Internet. You would then be using the Internet as a *virtual* private network or VPN. This can significantly save on monthly line costs, offload operations to the service provider, and free up resources for more strategic projects.

This chapter covers the IP Security (IPsec) standard that enables such a solution: the capability to connect your private offices and users, using any untrusted IP network for the secure interconnections.

The main topics of this chapter are

- IPsec Enables Virtual Private Networks
- Benefits of IPsec's Layer 3 Service
- Basic IPsec Security Concepts and Cryptography
- IPsec Concepts
- Internet Key Exchange
- Tying All of the Pieces Together: A Comprehensive Example with IPsec and IKE
- Configuring IKE
- Configuring IPsec
- Troubleshooting IPsec and IKE

NOTE You might notice from the preceding topic list that configuring IKE (Internet Key Exchange) is covered *before* configuring IPsec—even though the background on IKE follows IPsec. This is because IKE is a parent protocol of IPsec. This will be more apparent after the section "Tying All of the Pieces Together: A Comprehensive Example with IPsec and IKE."

IPsec Enables Virtual Private Networks

The IPsec standard is an architecture for transporting data securely on an untrusted network, using encryption and other services. Although IPsec can be used on any IP network, its most popular use by far is for building virtual private networks (VPNs) over the public Internet. RFC 2401 defines the general architecture of IPsec, and RFCs 2402 through 2412 define specific technologies within the IPsec framework.

A VPN is a network service over a public infrastructure (like the Internet or other untrusted network) with the privacy and security policies of a private network. There are three main categories of VPNs:

- *Remote access VPNs* or *access VPNs* provide connectivity for remote users such as road warriors and telecommuters. They are an alternative to direct-dial access and ISDN. The benefits include universal access, remote access outsourcing, and lower costs.
- *Intranet VPNs* provide internal, site-to-site connectivity for organizations—connections between private branch offices, for example. They are an alternative to leased lines and other WAN links. The benefits include wide-reaching connectivity and lower costs.
- *Extranet VPNs* provide connectivity for business partners, customers, and suppliers. They are an alternative to FAX, mail, and electronic data interchange (EDI). They enable collaboration, application sharing, and electronic commerce.

To achieve a level of privacy and security on a public infrastructure equal to that of a private network, many things need to happen:

- You need to identify and authenticate trusted parties (network devices) with whom you want to communicate.
- Data must be exchanged so that it is infeasible for an unauthorized person to intercept, record, and extract the contents of the data.
- The data should not be altered in transit and trusted parties on a VPN must be able to detect this. An attacker should not be able to change your bank deposit of $1000 to $100, for example.
- Data exchanged between trusted parties must be protected against replay attacks. That is, an attacker should not be able to record you withdrawing $1000 from your bank account and then replay that transaction at a later time so that an additional $1000 gets withdrawn.

To address these requirements and make VPNs possible, IPsec provides encryption, authentication, and integrity-checking services. These services are covered in this chapter.

Benefits of IPsec's Layer 3 Service

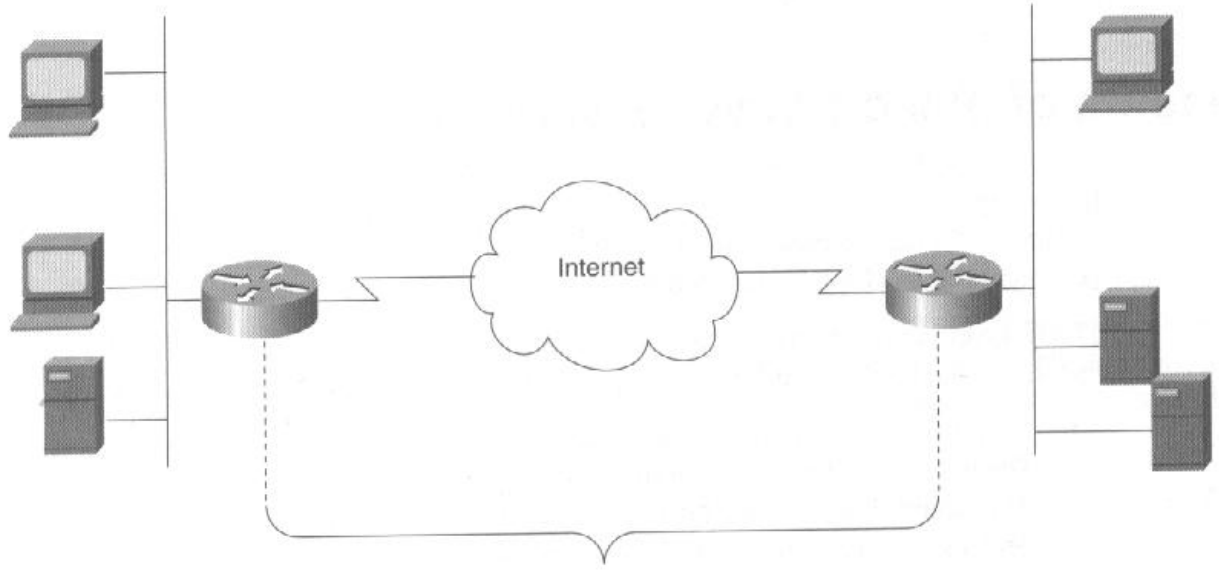
IPsec works at the IP layer (Layer 3) and integrates with your existing IP network. You might, for example, replace a costly leased line connection between two international offices with an intranet VPN that connects the offices through an Internet Service Provider (ISP). The rest of your network can be left unchanged.

The IP layer approach gives IPsec advantages over more traditional security strategies such as link-layer and application-layer encryption. The following are some key IPsec benefits:

- To transfer data securely between any two sites, you need IPsec only at the endpoints: The network in between (the Internet, for example) does not have to be IPsec-aware, just IP-enabled. This differs from the link-layer (Layer 2) approach, which requires encryption devices on every network link between the sites.
- IPsec can be deployed transparent of clients and servers. You do not have to reconfigure workstations or applications to work with IPsec. This differs from application-layer encryption. With application-layer encryption, each client program must have its own security implementation, and little can be done to secure a program that does not have such built-in services as encryption.
- Offloading IPsec to routers (and other networking devices) simplifies deployment: The routers are responsible for applying such services as encryption, and the clients and servers remain as they are for easier management.
- Applying IPsec services at key network points makes it easier to enforce and maintain a consistent security policy across the organization.
- IPsec is flexible. With IPsec you can selectively protect certain types of traffic and let other non-secret traffic circulate in the clear (without any IPsec service) to save system resources.
- From a QoS standpoint, IPsec packets are just like regular IP packets: They carry an IP precedence value. This means you can apply QoS to IPsec packets with the same services used to control normal packets (see Chapters 4, "Deploying Basic Quality of Service Features," and 5, "Deploying Advanced Quality of Service Features").

Figure 7-1 depicts a simple intranet VPN with routers providing IPsec secure networking on behalf of clients and servers at two sites.

Figure 7-1 *IPsec-Enabled Routers Provide IP Layer Security for Clients and Servers*



In Figure 7-1, the routers apply IPsec services (encryption, authentication, and so on) to traffic that is exchanged between the two sites. Because IPsec is an IP layer function, no modification is required for the Internet (it simply forwards IP packets) or to the clients and servers at each site. Because it is a trusted network, the traffic on the LAN within each site is not protected, but data exchanged between the sites is protected to maintain privacy over the Internet. Finally, rules are configured on each router so that traffic not destined to the other site—for example, traffic to public Web sites on the Internet—is transmitted normally, that is, without any IPsec handling.

NOTE To *protect* data with IPsec means to apply encryption, integrity checking, authentication, and other security services to the data so it can be transported securely over an untrusted network.

Basic IPsec Security Concepts and Cryptography

IPsec is a fairly large collection of technologies that encompasses network and security protocols, cryptographic algorithms, and recommendations. IPsec is an architecture for building secure communications over untrusted networks and provides the security services listed in the following sections. These services are *confidentiality*, *integrity*, *origin authentication*, and *anti-replay*. The following sections cover these services and introduce basic security and cryptographic principles as they apply to IPsec.

Confidentiality (Encryption)

Confidentiality provides privacy for the data being exchanged— only the intended recipient can easily read the data. This is achieved through encryption algorithms that scramble plaintext data into an unintelligible form called *ciphertext*. As part of the scrambling process, an encryption algorithm requires a *key* (or more than one key in the case of public key cryptography). The key is nothing more than a string of bits usually written in ASCII or hexadecimal notation. For example, the key 01111000011001100011010101110111011111101000110011 is written as xf5wz3 (ASCII) or 786635777A33 (hexadecimal). Keys are used by algorithms to encrypt plaintext and subsequently decrypt ciphertext. When encryption is used effectively, it is infeasible (see "Cryptographic Security: Infeasible, Not Impossible") to deduce the original plaintext from the ciphertext without the proper key. The key is like a password: If you know the key, you can decrypt the message and read the data, but if you don't know the key, you'll have great difficulty discovering the original message.

Some popular encryption algorithms include DES (Data Encryption Standard), Triple-DES (often written as 3DES), IDEA (International Data Encryption Algorithm), and Blowfish. The IPsec standard defines mandatory encryption algorithms (DES and null) and allows for 3DES, IDEA, Blowfish, and future algorithms. Consult RFC 2406 for more details.

Cryptographic Security: Infeasible, Not Impossible

Cryptography, a branch of mathematics and the science of secret writing, often calls a trusted security service *infeasible* to subvert (rather than *impossible* to subvert) because mathematically, any algorithm can be compromised if an attacker has enough computing power, know-how, and time to crack it. The security of an algorithm lies in the *amount* of computing power and time required to break it. If descrambling a ciphertext without the key takes years and millions of dollars of computing power, an algorithm is generally considered secure—that is, secure enough to protect most kinds of data for most users.

Another popular phrase in cryptographic circles is *believed to be secure* or *considered to be secure*. A security algorithm is believed to be secure (and *believed to be difficult to subvert*) if there is no easy way to crack it at the present time. The possibility always exists, however, that some smart person will come up with a mathematical breakthrough that makes an algorithm easy to compromise—a way to solve a difficult mathematical problem that is a feature and the basis of a security algorithm. If this happens, everyone will have to cease use of the once-believed secure algorithm and resort to other algorithms that are believed to be secure.

Applying Cryptographic Security to Your Data

There's simply no such thing as a security algorithm that is impossible to break. There are only algorithms that in practice, and depending on the type of data that needs to be protected, are considered secure and secure enough for general VPN applications.

At the time of this writing, Triple-DES is considered a secure encryption algorithm. DES (a precursor of Triple-DES), on the other hand, has reportedly been cracked within a few hours or less with specialized cracking hardware—hardware designed by and accessible to the crypto-elite. More information on the security provided by DES can be found in RFC 2405.

NOTE Governments such as the United States of America often restrict the exportation and use of encryption algorithms. Violations of such laws can be a serious offense, so it is important to be familiar with the laws and involve the exportation policy expert in your organization (if you have one).

Ultimately, you and your organization determine which algorithms are secure enough for your application. Concluding that an algorithm is secure enough for your data depends on the sensitivity of data you are protecting. Protecting top-secret corporate strategies or formulas for example, might require a more secure encryption algorithm than one for encrypting general office e-mail.

Typically, encryption algorithms follow one of two approaches: shared key encryption or public key encryption. Shared key encryption algorithms are the most familiar and are covered in the following section. IPsec employs both encryption strategies.

Shared Key Encryption

In shared key encryption, a single key is used for both encryption and decryption of the data. Only the trusted parties must know the shared key (also called a *shared secret* or *secret key*). DES, 3DES, IDEA, and Blowfish are examples of shared key encryption.

Consider the following communication sequence between two parties, Alice and Bob, using shared key encryption:

1. Alice and Bob agree on a shared key encryption algorithm and a shared key to use for their session.
2. Alice takes the plaintext she wants to send to Bob and encrypts it into ciphertext by using the algorithm and the shared key.
3. Alice sends the ciphertext to Bob.
4. Bob decrypts the ciphertext by using the algorithm and shared key. This results in the original plaintext.
5. Bob can now read the original message from Alice.

Alice and Bob represent any two parties that need to communicate securely over an untrusted infrastructure. Depending on the implementation, they can be people with PCs (or other client devices) or they can represent networking devices such as servers, routers, and firewalls (for instance, router *Alice* communicating with firewall *Bob*).

NOTE Shared key algorithms are also called *symmetric algorithms* because the same key is used for both encryption and decryption.

Agreeing on a Shared Key

Confidentiality ensures that communication is private between the trusted parties, Alice and Bob, but in a shared key encryption scenario, Alice and Bob must agree to use the same key before secure communication can commence. This is a problem, because Alice has to tell Bob what the shared key is without an attacker eavesdropping on the conversation and learning the shared key. Alice could tell Bob what the shared key is over a prior-existing secure session, but if Alice and Bob already have a secure means of communicating, they could use that to communicate and not bother with the shared key. Another thing Alice can do is communicate the shared key to Bob over a secure out-of-band channel—a direct telephone call or a registered letter. This might be feasible if Alice and Bob only have to do this a few times, but if Alice and Bob need to change keys frequently, or if Alice has to communicate with many more people than Bob, the out-of-band method becomes cumbersome.

Using Diffie-Hellman to Agree on a Shared Key

Clearly, the out-of-band method of establishing shared keys does not scale to the size needed for large networks and does not allow easy renegotiation of shared keys that have grown stale. To address this, the *Diffie-Hellman key exchange* algorithm allows two parties to exchange non-secret (publicly readable) data and calculate a unique shared key that is only known by them.

NOTE The Internet Key Exchange (IKE) protocol, used in conjunction with IPsec, leverages Diffie-Hellman key exchange to establish shared keys. See section "Internet Key Exchange" later in this chapter for more information. IKE is defined in RFC 2409.

Diffie-Hellman is not an encryption algorithm but an algorithm for establishing a shared key over an unsecured medium. After a shared key is established with Diffie-Hellman, that shared key can be used with the shared key encryption algorithms mentioned earlier: DES, Triple-DES, IDEA, Blowfish, and so on.

It is outside the scope of this book to explain exactly how Diffie-Hellman works—to do so requires a fair bit of mathematics involving discrete logarithms, modulus arithmetic, prime numbers, and other good things (see Bibliography for references). Without getting into too much math, Diffie-Hellman gets its security from the property that raising a number to a power (exponentiation) is much easier than calculating the logarithm (the reverse operation of exponentiation). With Diffie-Hellman, the calculations are performed with large numbers and modulus arithmetic. This does not make exponentiation more difficult, but it makes the reverse operation, calculating the logarithm and cracking the security, infeasible. That is, it's *believed* to be infeasible. The difficult problem of calculating the logarithm is a security feature of Diffie-Hellman and is known as the problem of finding a *discrete logarithm in a finite field.*

NOTE You do not have to know the mathematical gears of Diffie-Hellman to successfully deploy IPsec or IKE. Diffie-Hellman is working behind-the-scenes and is automatically invoked in IPsec/IKE implementations like Cisco's. It does not hurt to understand what is going on, though. Thus, the information in this section is provided for understanding how Diffie-Hellman solves the problem of establishing shared keys.

In brief, a Diffie-Hellman key exchange between two parties, Alice and Bob, looks like this (here comes some math):

- Alice and Bob are given numbers g and n. These are non-secret, publicly available numbers.
- Alice picks a large random number, x, calculates $A=g^x \bmod n$, and sends this value, A, to Bob over an untrusted network. The value x is known only to Alice and is called Alice's *secret*.
- Bob picks a large random number, y, calculates $B=g^y \bmod n$, and sends this value, B, to Alice over the untrusted network. The value y is known only to Bob and is Bob's secret.
- Alice computes $K_a = B^x \bmod n$.
- Bob computes $K_b = A^y \bmod n$.
- By virtue of an algebraic property of exponents, K_a and K_b are equal because
  - $B^x \bmod n = g^{xy} \bmod n = A^y \bmod n$
- Alice and Bob have successfully negotiated a shared key $K=K_a=K_b$ that is known only to them.

Diffie-Hellman Security Details: Discrete Logarithms

An attacker knows g and n and can intercept non-secret values A and B, but will find it infeasible to calculate secret values x or y (assuming numbers g, n, x, and y are selected properly). Either x or y is needed to arrive at the shared key, K, but to find x or y the attacker must calculate a discrete logarithm in a finite field. This is difficult to do when x, y, and n are very large numbers (several hundred bits long).

The following is a simple example in which the attacker must find secret value y. This is for illustration purposes. In real-world Diffie-Hellman implementations, larger numbers are used to make the problem infeasible to solve.

$g^y \bmod n = B$

$g = 2$

$n = 124798787687687643873593$

$B = 87733227500518975093493$

Attacker must find secret value y (the discrete logarithm of B, modulus n). This is hard to do with large numbers.

Answer: $y = 23874239847329847238472398$

To find the answer, the attacker cannot simply calculate the logarithm of B base g and arrive at secret value y because of complications introduced by modulus arithmetic. For more information, see the sources listed in the Bibliography.

Subverting Diffie-Hellman

Although it is infeasible to find the secret values (x and y) owned by Alice and Bob, an attacker could launch a different attack against Diffie-Hellman called a *man-in-the-middle* attack. The following describes the attack:

- The attacker intercepts a Diffie-Hellman exchange between Alice and Bob.
- The attacker pretends to be Bob and exchanges Diffie-Hellman messages with Alice. Alice is unaware that she is communicating with the attacker.
- Alice and the attacker complete their Diffie-Hellman exchange and compute a shared secret.
- The attacker turns to Bob, pretends to be Alice, and exchanges Diffie-Hellman messages with Bob. Bob is unaware that he is communicating with the attacker.
- Bob and the attacker complete their Diffie-Hellman exchange and compute a shared secret. This is a different shared secret than the one the attacker shares with Alice.

- The attacker now acts as an intermediary in the transfer of data between Alice and Bob by decrypting from one side and encrypting to the other. All the while, Alice and Bob are unaware of the fact that an attacker exists and is reading the data transferred between them.

As a countermeasure to the man-in-the-middle attack, the IKE protocol uses digital signatures to authenticate the origin of Diffie-Hellman exchanges. See "Digital Signatures," later in this chapter.

Public Key Encryption

With public key encryption, the parties that wish to communicate (Alice and Bob) each create a pair of keys and each own the key pair they create. One key, the *private key*, is kept secret and known only by the owner. The other key, the *public key*, is non-secret information and is given to anyone who wishes to send confidential information to the owner of the public key.

NOTE One of the most popular public key encryption algorithms is *RSA*, which is named after its inventors: Rivest, Shamir, and Adleman. In the Cisco implementation, RSA encryption is part of the IKE protocol that works in conjunction with IPsec. IKE is covered in the "Internet Key Exchange" section later in this chapter.

The advantage of using public key encryption is that you do not have to transmit any secret information to have someone communicate with you securely. You can openly publish your public key and anyone can use your public key to send you an encrypted message. You keep your private key secret (known only to you) and use it to decrypt messages that were encrypted with your public key. Likewise, to send an encrypted message to someone, all you need is that person's public key.

The following sequence illustrates how public key encryption works. Bob wishes to send Alice an encrypted message:

1. Alice sends Bob her public key.
2. Bob encrypts the message destined for Alice, using Alice's public key.
3. Bob sends the encrypted data to Alice.
4. Alice receives the encrypted data and, through some fancy math built into the encryption algorithm, decrypts the data by using her private key.
5. Alice and Bob switch roles for communication in the opposite direction (when Alice sends an encrypted message to Bob).

Unlike shared key encryption, in which there is one key known by Alice and Bob, public key encryption involves four keys (assuming bidirectional communication): Alice's public and private keys, and Bob's public and private keys.

This system has scaling advantages because people only need to exchange public keys to securely communicate with one another. Public keys can also be held in a widely accessible, central repository (because they are non-secret, public information) for better management.

The following are some notable properties of public key encryption:

- Decrypting by using the public key is infeasible. Because of clever mathematical tricks (called *one-way trapdoor functions*) built into the system, the public key is useful for only encryption, not decryption. (The logic is reversed with digital signatures: The private key can be used to encrypt/sign and the public key used to decrypt/verify. See "Digital Signatures.")
- Deducing the private key by knowing only the public key and the encrypted data is infeasible. As long as the private key is known only by the owner, only the owner can decrypt the data.
- Public key encryption is slow—public key algorithms are commonly 1,000 times slower than shared key encryptions. As such, public key encryption is often used to exchange small pieces of information and shared key encryption is used to encrypt bulk data such as traffic flows between two points on the Internet.

NOTE Public key encryption is one application of *public key cryptography*, which is the broader study of systems that use the concept of a public and private key. Another popular application of public key cryptography is *digital signatures*, which are covered in the "Digital Signatures" section later in this chapter.

Integrity

Integrity (also called *data integrity*) provides a guarantee that data has not been altered in transit. Two parties that are communicating over an untrusted network must be able to verify that the data received is exactly the same as the data originally sent.

Integrity is achieved through cryptographic hashing algorithms that calculate a unique value when given a piece of data or *message* as input. The unique value (called a *hash value* or *message authentication code*) is like a fingerprint. Given a message and its calculated hash value, it is infeasible to find another message with the same hash value (assuming the implementation is done properly). This is analogous to the idea that it is infeasible to find another person with a fingerprint that matches one on your hand.

Therefore, if someone sends you a message and its associated hash value (fingerprint), you can do your own calculation of what the hash value for the message should be and compare it to the hash value from the other person. If the hash values match, you can be reasonably certain that the message was not altered in transit, because any alteration would change the message and result in a different hash value.

The following sequence illustrates how Bob can validate the integrity of a message sent by Alice:

1. Alice and Bob agree to use a hashing algorithm for integrity checking of data sent between them.
2. Alice and Bob agree on a shared key known only to them. See the "Agreeing on a Shared Key" earlier in this chapter for more information.
3. Alice needs to send Bob a message, "Hello!"
4. Alice calculates a hash value for the message. This is done by inputting the message "Hello!" and the shared key into the hashing algorithm. The output of the algorithm is the hash value.
5. Alice sends the message and her calculated hash value to Bob.
6. Bob calculates his own hash value for the received message. This is done by inputting the message and the shared key into the hashing algorithm. The output of the algorithm is Bob's hash value.
7. Bob compares his hash value with the hash value from Alice. If the hash values match, Bob knows the message was not altered in transit. Any alteration of the data would cause Bob's hash value to be different from Alice's hash value.

If the hash values match, Bob is also certain that the message came from Alice because they are the only ones who know the shared key. A different key would produce a different hash value even if the message were the same. Both the key and the message affect the outputted hash value because they are inputs to the hashing algorithm.

The use of a shared key with a hashing algorithm is a form of authentication (as described previously) and is called *keyed-hashing for message authentication,* or *HMAC* for short (MAC stands for Message Authentication Code and H signifies the use of a hashing function). HMAC is detailed in RFC 2104.

Hashing Algorithms: Examples with Message Digest 5

To illustrate the security provided by cryptographic hashing algorithms, consider the hash values computed by a popular algorithm, Message Digest 5 (MD5), in the following examples.

NOTE Another popular hashing algorithm used with IPsec is Secure Hash Algorithm (SHA-1), which was designed by the National Institute of Standards and Technology (NIST), a division of the U.S. Department of Commerce, and the National Security Agency (NSA), the official security organization of the U.S. government.

The MD5 hashing algorithm takes an input message and outputs a 128-bit hash value. The resulting hash value, written in hexadecimal format, is considered a unique fingerprint of the message. Consider the following input messages and their calculated hash values:

Case 1 MD5 Input: The quick brown fox jumped over the lazy dog.

MD5 Output: 5c6ffbdd40d9556b73a21e63c3e0e904

Case 2 MD5 Input: the quick brown fox jumped over the lazy dog.

MD5 Output: bb0fa6eff92c305f166803b6938dd33a

Although the two message inputs for Case 1 and Case 2 are very similar (only differing by an uppercase *T* and a lowercase *t* in the first word), their hash values are very different. This is a feature of a good hashing algorithm: For all practical purposes, every input (message) should have a unique output (hash value).

A value of 128 bits provides for 2^{128} unique combinations. The security of MD5 lies in infeasibility of altering a message (or substituting the message with a different one) so it passes undetected through the recipient's integrity check. This means, for the attack to succeed, the altered message has to produce the same hash value as the original message—a very difficult thing to do when a good hashing algorithm is used.

No matter how big (or how small) the input message is, the output of MD5 is always a 128-bit hash value. Consider the following examples where a single word is the input (Case 3) and a 100 KB file is the input (Case 4):

Case 3 MD5 Input: Shirley

MD5 Output: 257f4d1a26d393d41524ad7e83e7c524

Case 4 MD5 Input: <100 KB file>

MD5 Output: 5058f1af8388633f609cadb75a75dc9d

The preceding cases highlight the compressive nature of hashing algorithms like MD5. Hashing algorithms are flexible enough to generate hash values for short and long messages alike.

Origin Authentication

Origin authentication provides the assurance that the person you are communicating with is who he or she claims to be. Consider again Bob and Alice: An attacker could masquerade as Bob and attempt to communicate with Alice. If Alice has no means of authenticating Bob, Alice might then communicate with the attacker, thinking she's communicating with Bob. Alice needs a way to know for certain that a message claiming to be from Bob was indeed sent by Bob.

Digital signatures and digital certificates provide the origin authentication for which Alice is looking. IPsec leverages both digital signatures and digital certificates.

Digital Signatures

Digital signatures provide a mechanism for guaranteeing the authenticity of a message. A digital signature is the electronic equivalent of a written signature in that the digital signature can be used to prove that the message was indeed signed by the originator.

RSA signatures and *Digital Signature Standard (DSS)* are two common digital signature algorithms used in IPsec and IKE. Both are based on public key cryptography: An owner keeps a private key confidential and openly publishes a public key (see the related section on "Public Key Encryption" earlier in this chapter). An originator signs a message using his or her private key and sends the message, along with its associated signature, to the recipient. The recipient then verifies the signature by using the originator's public key.

The following sequence describes the basic idea behind digital signatures, using an exchange between Alice and Bob. In this example, Bob sends Alice a signed message using an RSA signature (other algorithms vary, but the concept is the same):

1. Bob sends Alice his public key.
2. Bob encrypts the message with his private key. This produces a unique, encrypted message that only Bob could have made, because he alone is the keeper of his private key. The encrypted message is considered a digitally signed message.
3. Bob sends the signed message to Alice.
4. Alice decrypts the message with Bob's public key. This verifies the digitally signed message. If Alice cannot decrypt the message, the digital signature is invalid. Only the private key can properly sign (encrypt) a message that can be verified (decrypted) with the matching public key.

Digital Certificates and Certification Authorities

A *digital certificate* is an electronic message that binds a public key to a real person (or organization). This association is needed to address some potential problems with public key cryptography (signatures or encryption), namely:

- When Alice receives Bob's public key, how does she know with certainty that the key really belongs to Bob? An attacker could easily generate a bogus private/public key pair and send the public key to Alice, pretending to be Bob.
- Bob could send a legitimately signed message to Alice but repudiate at a later time that the transaction took place. That is, he could deny signing the message that Alice received. Bob could do this by generating a temporary private/public key pair just for his transaction with Alice. When Alice attempts to prove she has a message signed by Bob, Bob could claim he never owned that particular public key.

To address these problems, a third party that Alice trusts needs to be involved. The trusted third party, called a *certification authority (CA)*, creates and signs a digital certificate containing Bob's name (and other identification data) and Bob's public key. The CA is responsible for verifying that the public key belongs to Bob. This is usually done through some manual, out-of-band (non-network) procedure. After a digital certificate is created and signed by the CA, Bob can distribute his certificate to anyone who wants his public key. Digital certificates can also be stored in centralized repositories (centralized servers or *CA servers*) because they are non-secret, public information. This helps to scale the distribution of public keys in large networks.

Obtaining a Digital Certificate

The following is a typical sequence of how Bob obtains a digital certificate from a CA:

1. Bob creates his key pair (private and public keys).
2. Bob sends his public key to the CA over the network and includes personal information (his name, his IP address, the serial number of his device, and so on). This is a request for certificate.
3. The request sits on the CA's server in a pending status until someone at the CA begins processing Bob's request.
4. A person at the CA verifies Bob's identity and confirms that Bob was the one who submitted the public key. For a trustworthy binding of Bob to the key, this verification step is done via some person-to-person, out-of-band procedure.
5. Bob periodically queries the CA server, hoping his certificate is complete and ready for retrieval.
6. The CA creates and signs a certificate containing Bob's public key and personal information, thereby guaranteeing the authenticity of the key.
7. Bob queries the CA server, discovers the certificate is ready, downloads the certificate, and stores the certificate.
8. Bob can now distribute his public key, using the certificate, and other people can validate the authenticity of the certificate by checking the CA's signature (checking the signature requires the CA's public key).

Trust and Nonrepudiation

If Alice trusts the CA who signed Bob's certificate, she has an assurance that the key contained in Bob's certificate has a binding to Bob's identity. Armed with Bob's certificate, Alice can prove that data she received with Bob's signature is authentic. The digital certificate (and the trusted third party behind it) can vouch for the fact that the key contained in the certificate belongs to Bob. This security feature is called *nonrepudiation*; in other words, Bob cannot repudiate or dismiss a digital signature as having no binding force. This addresses the problem of repudiation and public key cryptography stated earlier in this section.

Security of Digital Certificates

The binding of Bob's identity to the public key is only as good as the CA's manual verification step: A CA that requires Bob to visit one of its offices and present a valid passport along with his public key has a stronger binding certificate than a CA that merely requires Bob to submit his public key in an e-mail application. A digital certificate is like an ID card or passport—the level of trust you have in the document is determined by the reputation of the issuer and the lengths to which the issuer validates the identity of the individual.

What if a certificate is altered in transit? The CA's digital signature on the certificate protects against such an attack. If the certificate is modified in any way, a check of the CA's signature will fail. This is a property of digital signatures: The signature is derived from a combination of the certificate's contents *and* the CA's private key. Anyone can check the signature with the CA's public key and detect an alteration. (See also "Digital Signatures," earlier in this chapter.)

Anti-Replay

Anti-replay provides the assurance that past transactions cannot be replayed by an attacker. In spite of taking the precautions mentioned earlier (encryption, hashing, digital signatures, and certificates), an attacker might simply try to record and replay a transaction and not bother with subverting the security defenses.

IKE provides anti-replay security for IPsec by using sequence numbers combined with authentication. With anti-replay, senders and receivers can detect and reject data that is old or duplicate.

Consider Bob and Alice again: Bob invests in the stock market and places an order with Alice to buy 100 shares of some company. Somewhere between Bob and Alice, a lurking attacker records this transaction (with a packet analyzer or other capture device) and replays the captured transaction as another order from Bob to Alice. If Alice has anti-replay protection, she will detect and discard the replayed order from the attacker. On the other hand, if Alice does not have anti-replay protection, she might think Bob placed two orders—each for 100 shares, or 200 shares combined.

IPsec Concepts

The preceding sections of this chapter covered some basic principles of security and cryptography; the following sections introduce concepts specific to IPsec. These are

- Peers
- Transform Sets
- Security Associations
- Transport and Tunnel Modes
- Authentication Header (AH) and Encapsulating Security Payload (ESP)

Peers

A *peer* of an IPsec device is another device participating in IPsec. A peer can be a router, a firewall, a server, or a remote access device such as a PC with IPsec support. Peering between two IPsec devices is typically a point-to-point relationship. Going back to the example of two IPsec devices, Alice and Bob, Bob is a peer of Alice (and Alice a peer of Bob) when the two of them communicate with IPsec.

Transform Sets

A *transform set* is a list of IPsec protocols and cryptographic algorithms that a peer can accept. Because IPsec allows for the use of different protocols and algorithms, a peer needs to declare and negotiate with other peers what it can support. Peers communicate the protocols and algorithms they support by exchanging transform sets. For two peers to communicate successfully, they must share a common transform set. If they do not, their attempt to establish a peering will fail and they will not be able to communicate.

The following situation highlights transform sets in action. Consider a case where Alice and Bob are two IPsec devices on the Internet, but they were made by different manufacturers. Suppose Alice's implementation of IPsec supports an optional encryption algorithm (one that is supported but not mandated by the standard) that Bob does not support. If Alice sends Bob a transform set that includes the optional encryption algorithm, Bob will reject it because he has no way of encrypting or decrypting with that algorithm. What Alice could do to make things easier is send Bob two or three transform sets she thinks Bob might accept and have Bob pick a transform set acceptable to him.

A transform set typically contains the following information:

- An IPsec security protocol (AH or ESP—see "Authentication Header and Encapsulating Security Payload," later in this chapter) that is supported by the peer. Using AH and ESP together is also supported.

- An integrity/authentication algorithm supported by the peer (a hashing algorithm such as MD5 HMAC or SHA-1 HMAC, for example).
- An encryption algorithm supported by the peer (DES or Triple-DES, for example). A *null* encryption algorithm (no encryption) is also supported.

NOTE You may use authentication alone if encryption is not required. Encryption without authentication is also supported but is not recommended because of a potential security risk (see Bibliography for a reference).

An important point to know is that a transform set defines a set of protocols and algorithms to be used for *peering* with another device. It does not define a list of *all* the protocols and algorithms a peer supports. In other words, a transform set lists the rules for a session and is session-focused, not device-focused. A transform set is a proposal for communication: Alice might support DES, Triple-DES, IDEA, and Blowfish, but *propose* to Bob by way of a transform set that their session use DES.

Security Associations

A *security association (SA)* is a logical connection that protects data flowing from one peer to another by using a transform set. Security associations are like logical tunnels between peers: Traffic entering an SA is protected and transported to the other side (the other peer).

SAs are unidirectional—an SA protects data flowing in one direction only. Therefore, for secure bidirectional communication between peers, a pair of SAs is required.

IPsec maintains many pieces of data needed to support an SA between two peers. These parameters include

- The identity of the remote peer participating in IPsec (an IP address or hostname).
- The security protocol (AH or ESP), hashing algorithm (if one is used), and encryption algorithm (if ESP is used). This information is negotiated when the peers exchange transform sets.
- The shared keys used by the hashing and encryption algorithms for the duration of the SA (called the *lifetime* of the SA).
- A description of the traffic flow protected by the SA. Typically, this specifies the IP addresses and port numbers protected by the SA. The description can be fine-grained, such as a single TCP session between two hosts, or it can be broad, such as all traffic flowing from Subnet X to Subnet Y.

Note In the Cisco implementation, this flow description is equivalent to an extended access list (see Chapter 6, "Deploying Basic Security Services," for access list information).

- A unique number that identifies the SA (called the *Security Parameter Index*, or *SPI*).
- Timers and counters that record the lifetime of an SA. This is used to detect when an SA and its associated keys get old and need to be refreshed. SAs and keys can be refreshed only when IKE is used with IPsec.
- Sequence numbers for detecting replay attacks when IKE is used.

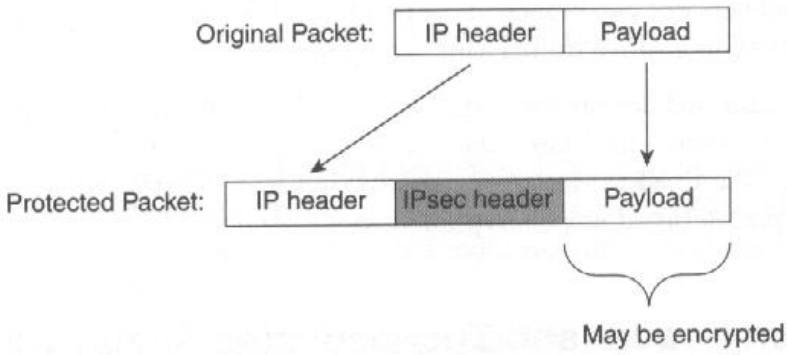
Multiple SA pairs are allowed between peers. For example, one pair of SAs can protect Telnet traffic with Triple-DES and MD5, and another pair of SAs can protect all other traffic with DES and SHA-1.

SAs are established by manual user configuration or dynamically by IKE. IKE is the recommended method, especially for large implementations, because of its scalability and enhanced security services (dynamic rekeying and anti-replay). See "Internet Key Exchange," later in this chapter.

Transport and Tunnel Modes

IPsec defines two kinds of SAs: transport and tunnel mode SAs. A *transport mode SA* is an association between two hosts. In transport mode, the IP payload is protected by IPsec and the original IP header is left intact. Additionally, an IPsec header is inserted after the IP header. This is illustrated in Figure 7-2.

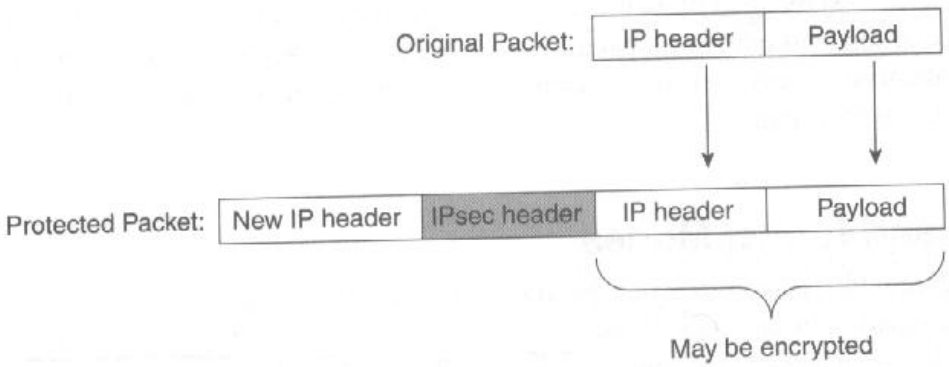
Figure 7-2 *Transport Mode SA*



Transport mode protects traffic between two IPsec hosts (between a PC and a server, for example) and does not afford any *traffic flow confidentiality*. That is, the volume of traffic transmitted from one host to another can easily be observed, even if encryption is used, because the original source and destination addresses are left intact. An attacker could use this data to determine where servers are located, with the assumption that servers transmit and receive more data than clients.

A tunnel mode SA is an association between two routers (also called *security gateways*) or between a router and a host. In tunnel mode, the entire IP packet is protected by and becomes the payload of a new packet. The IPsec header is inserted after the IP header of the new packet. This is illustrated in Figure 7-3.

Figure 7-3 *Tunnel Mode SA*



Tunnel mode is the basis for enabling dedicated VPNs between routers, as well as VPDNs that involve remote users terminating their SAs on routers to gain access to hosts within a network. With tunnel mode you do not have to equip every PC and server with IPsec; instead, you can activate IPsec on routers and use the routers to provide IPsec services on behalf of those computers. For example, you might establish an IPsec peering over the Internet between two branch office routers and use tunnel mode SAs between the routers to protect all interoffice traffic. This is a scenario depicted in Figure 7-1.

The source and destination addresses contained in the new IP header are that of the tunnel mode SA endpoints. Thus, tunnel mode SAs provide a level of traffic flow confidentiality because the IP addresses of the original packet are carried within the secure payload of an IPsec packet (assuming encryption is used).

Authentication Header and Encapsulating Security Payload

IPsec defines two security protocols called *Authentication Header (AH)* (RFC 2402) and *Encapsulating Security Payload (ESP)* (RFC 2406). Each protocol defines its own format for the IPsec header that follows the IP header of an IPsec packet (see Figures 7-2 and

7-3). Both protocols use the concept of an SA; therefore, SAs can be either AH SAs or ESP SAs (an SA cannot be both an AH *and* ESP SA). Additionally, both AH and ESP support transport and tunnel mode.

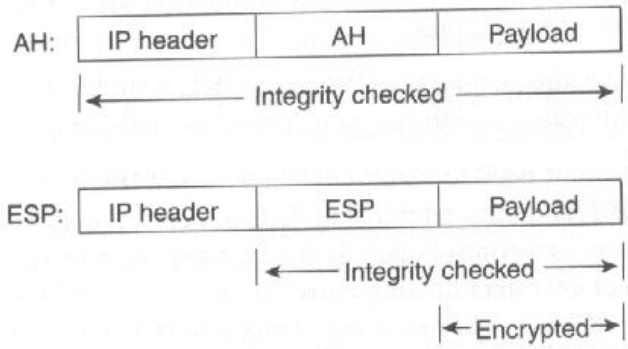
NOTE A device might send a transform set to its peer that specifies use of both AH *and* ESP. If the peer agrees, four SAs will be established: two for AH and two for ESP (assuming bidirectional traffic flow).

AH provides integrity and authentication, using shared key hashing algorithms such as MD5 HMAC and SHA-1 HMAC. AH does not provide confidentiality (encryption).

ESP provides confidentiality and, optionally, integrity and authentication. For confidentiality, ESP supports shared key encryption algorithms such as DES and Triple-DES. Like AH, ESP supports shared key hashing algorithms such as MD5 HMAC and SHA-1 HMAC for integrity and authentication.

Because ESP provides everything that AH does (integrity and authentication), you might not have a need for AH at all. However, there is a slight difference between the integrity and authentication provided by AH and the integrity and authentication provided by ESP. This is illustrated in Figure 7-4.

Figure 7-4 *Differences Between AH and ESP Integrity Checking*



ESP does not check the integrity of the entire IP packet—it protects everything but the IP header. AH on the other hand, checks the integrity of the entire IPsec packet, including the IP header (technically, some fields in the IP header are subject to change during transit and AH cannot protect these values).

Is this difference between AH and ESP significant enough to dismiss ESP's integrity and authentication service? Probably not for most implementations; however, if the integrity of the IP header is important, you can use AH and ESP together. This comes at the price of having twice as many SAs as when using ESP alone: An SA can support either AH or ESP, but not both.

Internet Key Exchange

The Internet Key Exchange (IKE) protocol (RFC 2409) is a management protocol used in conjunction with IPsec and is based on three key management protocols: Internet Security Association and Key Management Protocol (ISAKMP) (RFC 2408), Oakley (RFC 2412), and SKEME (simply called "A Versatile Secure Key Exchange Mechanism for Internet" in a paper by Hugo Krawczyk).

IKE is an important protocol that provides IPsec with the following services:

- Establishes IPsec SAs dynamically as they are needed. Without IKE, you must manually configure all SAs required between all of the peers.
- Enables dynamic rekeying. With IKE, keys are expired after a period of time and new keys are dynamically created. This enhances IPsec security by limiting the amount of data protected by any one key and ever increasing the number of keys an attacker has to crack. Some security aspects of key lifetimes are discussed in RFC 2405.
- Enables anti-replay protection. Without IKE, IPsec peers cannot detect a replay attack. See "Anti-Replay," earlier in this chapter.
- Enables origin authentication with digital certificates and CA servers. Without IKE, there is no CA support, and origin authentication must be done person-to-person (manually and out-of-band).
- Optionally provides *Perfect Forward Secrecy (PFS)*. This is a cryptographic characteristic of shared keys and means that the compromise of a key does not help an attacker discover other keys. In other words, each key is secure on its own merits and is not derived from any other key. Use PFS for enhanced security. (See also "Configuring Perfect Forward Secrecy," later in this chapter.)

For the preceding reasons, deploying IPsec with IKE is highly recommended. IKE adds security features, increases scalability, and simplifies configuration.

When two IPsec devices need to communicate by using IPsec, they first authenticate one another by using IKE and then establish an IKE SA between them for management. The IKE SA is secure and acts as a control channel for exchanging keys and negotiating IPsec SAs (the SAs that protect *user* data flowing between the peers). Unlike IPsec SAs, which are unidirectional, the IKE SA is bidirectional. Only one IKE SA is needed between two IPsec peers to support multiple IPsec SAs, as shown in Figure 7-5.

Figure 7-5 *A Single IKE SA Acts as a Secure Management Channel for IPsec SAs*

Tying All of the Pieces Together: A Comprehensive Example with IPsec and IKE

The sections up to this point have covered the building blocks of IPsec, but how do all of these technologies work together? The following sequence between Alice and Bob incorporates the concepts covered in the preceding sections and describes how IPsec's cryptographic techniques can achieve secure communication over an untrusted network.

In this example, assume that no prior communication has occurred between Alice and Bob. To make the example more concrete, further assume that Alice needs to establish a secure FTP session with Bob:

1. Alice initiates an IKE SA with Bob and proposes a transform set they should use. The transform set specifies the encryption, hashing, and authentication algorithm for the IKE SA as well as such other information as the SA lifetime (the lifetime sets an expiration time for the SA). If Bob can support the transform set, the negotiation continues. Otherwise, the IKE SA fails and no other communication occurs. See "Transform Sets," earlier in this chapter.
2. Alice sends her digital certificate to Bob. The certificate contains her public key and guarantees the authenticity of the key. Bob does the same thing so that Alice has his public key. See "Digital Certificates," earlier in this chapter.
3. Alice and Bob exchange digitally signed Diffie-Hellman numbers for the purpose of establishing a shared secret. The Diffie-Hellman numbers are signed so that each peer can validate the identity of the other peer. Otherwise, an attacker could masquerade as Bob and perform a Diffie-Hellman exchange with Alice—a man-in-the-middle attack (see "Subverting Diffie-Hellman," earlier in this chapter).
4. Alice verifies the signature on Bob's Diffie-Hellman number with Bob's public key (this authenticates Bob). Bob does the same thing and verifies the signature on Alice's Diffie-Hellman number, using Alice's public key.
5. Alice and Bob calculate a shared key known only to them by using the Diffie-Hellman key exchange algorithm. See "Using Diffie-Hellman to Agree on a Shared Key," earlier in this chapter.
6. At this point, the IKE SA is established between Alice and Bob: Data can be sent securely between the peers, using the shared key from Step 5 and the algorithms specified in the IKE transform set.
7. Alice initiates an IPsec SA, which is needed to support the packets for the FTP session (recall that the IKE SA is used for management only, not for user data). Using the IKE SA as a secure channel, Alice proposes one or more transform sets for the IPsec SA. Each of her transform sets specifies a security protocol (AH or ESP) and algorithms (hashing and/or encryption) for the SA. If Bob can support a transform set, negotiation continues. Otherwise, the IPsec SA fails and no other communication occurs (no FTP packets can be sent).

8 Alice and Bob calculate a shared key for the IPsec SA, using the Diffie-Hellman key exchange algorithm. This provides PFS: The key for the IPsec SA is not derived from the key for the IKE SA.

9 Alice can now send FTP packets securely to Bob over the IPsec SA, using the shared key from Step 8 and the algorithms specified in the transform set for the IPsec SA (Step 7).

10 The IPsec SA is maintained by IKE. A short time before the IPsec SA expires, a new SA with new keys is created and communication is rolled over to the new SA transparently. This is the rekeying service provided by IKE.

11 When Bob needs to send data, he must initiate and establish an IPsec SA with Alice because IPsec SAs are unidirectional. See "Security Associations," earlier in this chapter.

Configuring IKE

Cisco's implementation of IKE supports three authentication methods. They are

- **Pre-shared keys**—Two peers are configured with a predetermined shared key. Peers are authenticated if they both possess the same key. This is a simple configuration, does not use public key cryptography or digital certificates, and might be suitable for small networks.
- **RSA encryption**—RSA encryption uses public key cryptography to authenticate peers but does not use digital certificates (does not provide nonrepudiation). Each device must be manually configured with the public key of its peer. This has the advantage of not relying on a shared key and might be suitable for small networks.
- **RSA signatures with digital certificates**—This uses public key cryptography and provides nonrepudiation in conjunction with a CA. A peer obtains a digital certificate from a CA server and distributes its public keys, using the digital certificate. This provides the scalability needed for larger networks.

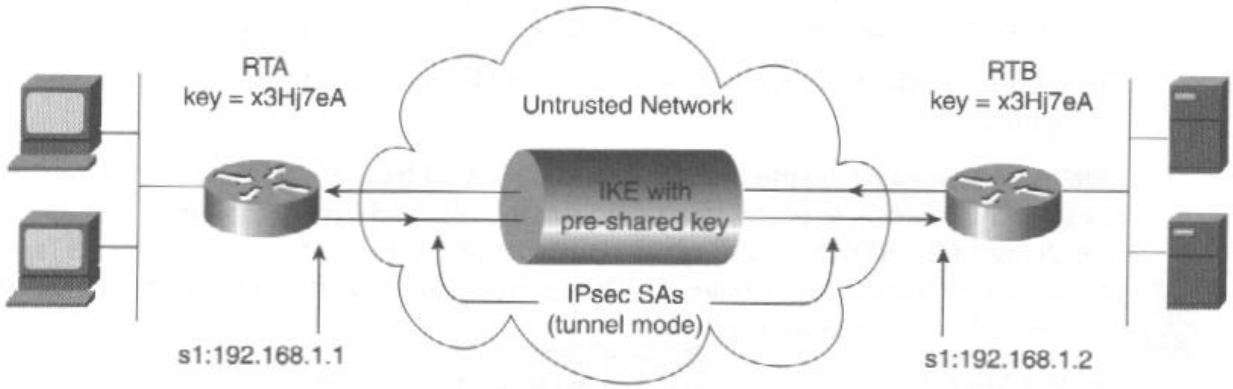
Configuring IKE with Pre-Shared Keys

Authentication with pre-shared keys is the simplest implementation of IKE and might be suitable for small networks (10 routers and fewer, perhaps). As mentioned previously, this configuration does not have the scaling advantages of public key cryptography and digital certificates. Instead, two peers are manually configured with the same key (a shared key). How the keys are determined and programmed into the routers is the responsibility of the router administrators. When a pre-shared key needs to change, human intervention is needed again.

NOTE The shared key is confidential information and therefore needs to be exchanged and configured out-of-band, never across an untrusted network. This makes key management more cumbersome than IKE using RSA encryption and IKE using RSA signatures with digital certificates.

Consider the network in Figure 7-6 with two routers that peer over an untrusted network and provide tunnel mode SAs for their local clients.

Figure 7-6 *Two Routers Peering with IKE and Pre-Shared Keys*



The following IOS commands configure Router A so it can establish an IKE SA with Router B, using pre-shared keys. For clarity, these commands include the context of the IOS prompt.

```
RTA#conf t
Enter configuration commands, one per line. End with CNTL/Z.
RTA(config)#crypto isakmp policy 10
RTA(config-isakmp)#authentication pre-share
RTA(config-isakmp)#hash md5
RTA(config-isakmp)#encryption des
RTA(config-isakmp)#lifetime 43200
RTA(config-isakmp)#exit
RTA(config)#crypto isakmp key x3Hj7eA address 192.168.1.2
```

The command **conf t** (short for **configure terminal**) changes the prompt from enable mode to configure mode.

The command **crypto isakmp policy 10** creates an IKE transform set (called a *policy*) whose priority is 10 and changes the command mode to ISAKMP policy configuration. The keyword **isakmp** in IOS commands refers to IKE (recall that IKE is a hybrid protocol of ISAKMP, Oakley, and SKEME). A router can have multiple IKE transform sets. When a router peers with another IKE device, it negotiates an IKE transform set. It checks each of its IKE transform sets in the order of priority (1 is the highest priority) until a match with the other device is found.

The next four lines configure the parameters of the IKE transform set:

- **authentication pre-share** dictates that the transform set use pre-shared keys for authenticating the remote peer.
- **hash md5** configures MD5 hashing as the transform set's integrity-checking algorithm. Instead of **md5**, you can use the keyword **sha** to define SHA-1 (Secure Hash Algorithm).
- **encryption des** configures the transform set for DES (56-bit) encryption. At the time of this writing, the other option available to most routers is Triple-DES (keyword **3des**). More algorithms might be added by Cisco in the future.
- **lifetime 43200** sets the lifetime for the IKE SA to 43,200 seconds (12 hours). The default is 86,400 seconds (24 hours). After the IKE SA reaches the age specified by this command, it is removed until IKE is needed again to establish new IPsec SAs (existing IPsec SAs have their own lifetimes and are not disturbed when the IKE SA expires).

The command **exit** changes the mode from ISAKMP policy configuration back to global config mode.

Finally, the command **crypto isakmp key x3Hj7eA address 192.168.1.2** sets the pre-shared key used by Router A to peer with Router B to the string **x3Hj7eA**. The string can be any user-defined text (no spaces) as long as it matches the string configured in Router B (see the following configuration for Router B). The keywords **address 192.168.1.2** specify the IP address of Router B's Serial1 interface.

The following is the matching configuration for Router B. For brevity, only the output of **show running-config** relevant to IKE is shown.

```
RTB#sh run

crypto isakmp policy 10
 hash md5
 authentication pre-share
 lifetime 43200
crypto isakmp key x3Hj7eA address 192.168.1.1
```

where **address 192.168.1.1** specifies the IP address of Router A's Serial1 interface.

The command **encryption des** does not appear in the output of **show running-config** because it is the default encryption algorithm for IKE.

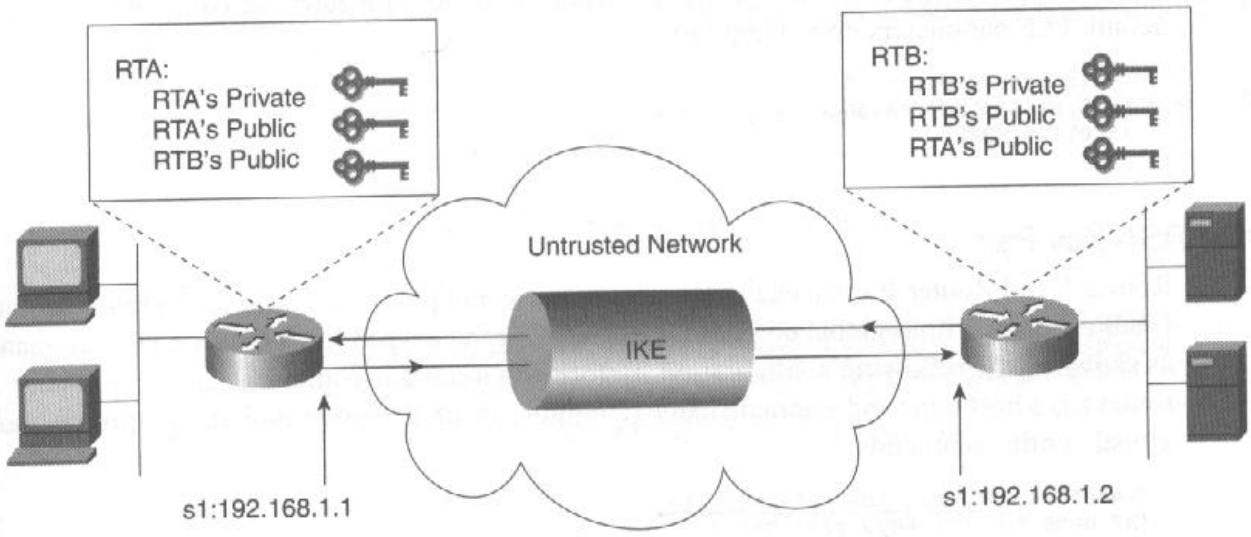
NOTE Router A and Router B must share at least one common IKE transform set; otherwise, IKE negotiation will fail. The pre-shared keys must also match, or authentication will fail.

Configuring IKE with RSA Encryption

A router using IKE with RSA encryption (RSA public key cryptography) is configured with the non-secret, public keys of its peers. This makes the exchanging of keys less problematic than authentication with secret, pre-shared keys. However, a device must be manually configured with the public key of every peer with which it builds an IKE SA. This means RSA encryption does not scale well in large networks. Also, RSA encryption lacks the nonrepudiation that is available when using digital certificates (see "Configuring IKE with RSA Signatures and Digital Certificates," later in this chapter).

Consider the network in Figure 7-7 with two routers peering over an untrusted network with IKE and RSA encryption.

Figure 7-7 *Two Routers Peering with IKE and RSA Encryption*



In the preceding figure, Router A owns a private and public key pair and is configured with Router B's public key. Router A uses Router B's public key to authenticate IKE sessions with Router B. Likewise, Router B owns its key pair and is configured with Router A's public key. The IP addresses assigned to the serial interfaces of Router A and Router B are 192.168.1.1 and 192.168.1.2, respectively.

Configure IKE Transform Sets

The following IOS commands configure Router A so that it can establish an IKE SA with Router B, using RSA encryption (for clarity, these commands include the context of the IOS prompt):

```
RTA#conf t
Enter configuration commands, one per line. End with CNTL/Z.
RTA(config)#crypto isakmp policy 9
RTA(config-isakmp)#authentication rsa-encr
```

continues

```
RTA(config-isakmp)#hash sha
RTA(config-isakmp)#encryption des
RTA(config-isakmp)#lifetime 43200
RTA(config-isakmp)#exit
RTA(config)#_
```

The preceding commands create an IKE transform set with the following characteristics:

- Priority is 9.
- Authentication method is RSA encryption (**rsa-encr**).
- Hashing algorithm is SHA-1 (**sha**).
- Encryption algorithm is 56-bit DES (**des**).
- Lifetime is 43,200 seconds (12 hours).

You must configure Router B with the same IKE transform set; otherwise, authentication with RSA encryption will fail. The following is a partial output of Router B's configuration (the default IKE parameters do not appear):

```
crypto isakmp policy 9
 authentication rsa-encr
 lifetime 43200
```

Generate RSA Key Pair

Router A and Router B must each generate a private and public key pair. To do this, you run a small program within global config mode by issuing the **crypto key generate rsa** command, as shown in the following configuration. Before you issue this command, ensure that your router has a hostname and a domain name (configured with the **hostname** and **ip domain-name** global config commands).

```
RTA(config)#crypto key generate rsa
The name for the keys will be: RTA.cisco.com
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1536
Generating RSA keys ...
<This takes some time>
[OK]

RTA(config)#_
```

As shown in the preceding output, the key generation program requires you to input the modulus (or key length). Longer key lengths provide greater security but take more time to generate.

NOTE During key generation you might see system messages that begin with **%SYS-3-CPUHOG**. These messages indicate that the router is experiencing high CPU utilization—expected, because key generation is CPU-intensive. Other tasks in the router might be adversely affected by the high utilization; therefore, generate the keys before you put the router into service.

Depending on the size of the key and the CPU power of your router, the time to generate the key pair can range from 1 second (Cisco 4700 router and a 512-bit length) to over an hour (Cisco 2500 router and a 2048-bit length). RSA keys smaller than 512 bits are not recommended. Bruce Schneier (see Bibliography) has theorized that a 1536-bit public key length will be reasonably secure for protecting the data of a typical corporation through the year 2010. This takes into account some assumptions on the advance of computing power and mathematics.

After the keys are generated, you can view the public key with the enable command **show crypto key mypubkey rsa** (this is Router A)**:**

```
RTA#sh crypto key mypubkey rsa
% Key pair was generated at: 10:06:46 PST Mar 2 1999
Key name: RTA.cisco.com
 Usage: General Purpose Key
 Key Data:
  305C300D 06092A86 4886F70D 01010105 00034B00 30480241 00CE23D4 4C3CB0E9
  4C44DB1B E29F1F20 53E64550 1A832B8A A439EBFE ACD24C5B 6E6CAB70 CE03048F
  8BFA0F00 C87CD8A7 756EA2D4 A387CA30 628DBC14 22903286 D3020301 0001
```

The hexadecimal data after the line **Key Data**: is the public key.

NOTE For security purposes, you cannot view the private key. The private key is stored in a nonviewable portion of the router's nonvolatile RAM (NVRAM) and is not stored when the configuration is backed up to another device.

Here's the key generation step and display of the public key for Router B (refer to Figure 7-7):

```
RTB(config)#crypto key generate rsa
The name for the keys will be: RTB.cisco.com
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1536
Generating RSA keys ...
[OK]

RTB(config)#^Z
```

continues

```
RTB#sh cry key mypub rsa
% Key pair was generated at: 19:08:18 UTC Mar 2 1999
Key name: RTB.cisco.com
 Usage: General Purpose Key
 Key Data:
  305C300D 06092A86 4886F70D 01010105 00034B00 30480241 00A62A0B 2376E4EC
  CB8E21B7 FFD405E8 BE36BC5B AE39FE9F BB57A830 CAFE6A51 E6A14427 191056F1
  5611F624 081CCD3F 73FCD337 8EA14F27 95634B9F 367BDCF4 E5020301 0001
```

Configure the Router with the Public Key of the Other Device

Router A must now be configured with the public key of Router B (and vice versa). If you own Router A and someone else owns Router B, you might not have visibility to Router B's public key. In such a case, the owner of Router B has to send you the output of Router B's **show crypto key mypubkey rsa**. Again, public keys are non-secret information so they do not have to be sent out-of-band—e-mail and Telnet are viable methods. However, you must be certain that the public key you are given is genuine (owned by the remote peer, not by an attacker pretending to be the remote peer).

The following configuration commands configure Router A with Router B's public key (Router B's configuration is similar):

```
RTA(config)#crypto key pubkey-chain rsa
RTA(config-pubkey-chain)#addressed-key 192.168.1.2
RTA(config-pubkey-key)#key-string
Enter a public key as a hexadecimal number ....

RTA(config-pubkey)# 305C300D 06092A86 4886F70D 01010105 00034B00 30480241 00A62A0B
     2376E4EC
RTA(config-pubkey)# CB8E21B7 FFD405E8 BE36BC5B AE39FE9F BB57A830 CAFE6A51 E6A14427
     191056F1
RTA(config-pubkey)# 5611F624 081CCD3F 73FCD337 8EA14F27 95634B9F 367BDCF4 E5020301
     0001
RTA(config-pubkey)#quit
RTA(config-pubkey-key)#^Z
RTA#
```

The command **crypto key pubkey-chain rsa** changes the command mode from global config mode to public key chain configuration mode (indicated by prompt changing to **config-pubkey-chain**). The public key *chain* is the set of all public keys this router possesses—it's similar to a real-world key chain.

The command **addressed-key 192.168.1.2** tells the router that you wish to enter a public key for remote peer 192.168.1.2 (Router B's Serial1 interface). This command changes the command mode to public key configuration mode as indicated by the next prompt **config-pubkey-key**.

You use the command **key-string** (followed by a carriage return) to begin entry of the remote peer's public key. The hexadecimal data of the public key (the next three lines) is

cut and pasted into the router to avoid making typographical mistakes. After the key data is entered, the command **quit** is issued on its own line and the mode reverts to public key configuration mode.

Finally, typing **Ctrl-Z** (or **end**) exits configure mode completely and returns the prompt to enable mode.

To verify that the public key has been configured correctly, issue **show running-config** on Router A:

```
RTA#sh run
Building configuration...

Current configuration:
<lines deleted for brevity>
!
crypto key pubkey-chain rsa
 addressed-key 192.168.1.2
  address 192.168.1.2
  key-string
   305C300D 06092A86 4886F70D 01010105 00034B00 30480241 00A62A0B 2376E4EC
   CB8E21B7 FFD405E8 BE36BC5B AE39FE9F BB57A830 CAFE6A51 E6A14427 191056F1
   5611F624 081CCD3F 73FCD337 8EA14F27 95634B9F 367BDCF4 E5020301 0001
  quit
```

TIP When there are multiple peers, add the public keys of the other devices.

After Router B is configured with Router A's public key (similar method), the IKE configuration for the network in Figure 7-7 is complete.

Configuring IKE with RSA Signatures and Digital Certificates

IKE authentication with digital certificates uses RSA digital signatures and provides scalability for larger networks. As mentioned previously, digital certificates provide nonrepudiation through the service of a CA.

NOTE Your organization might administer a CA server and act as the CA for all of the devices and people that belong to your organization. Also, you might be the CA for third parties (suppliers, partners, customers) that do business with your organization.

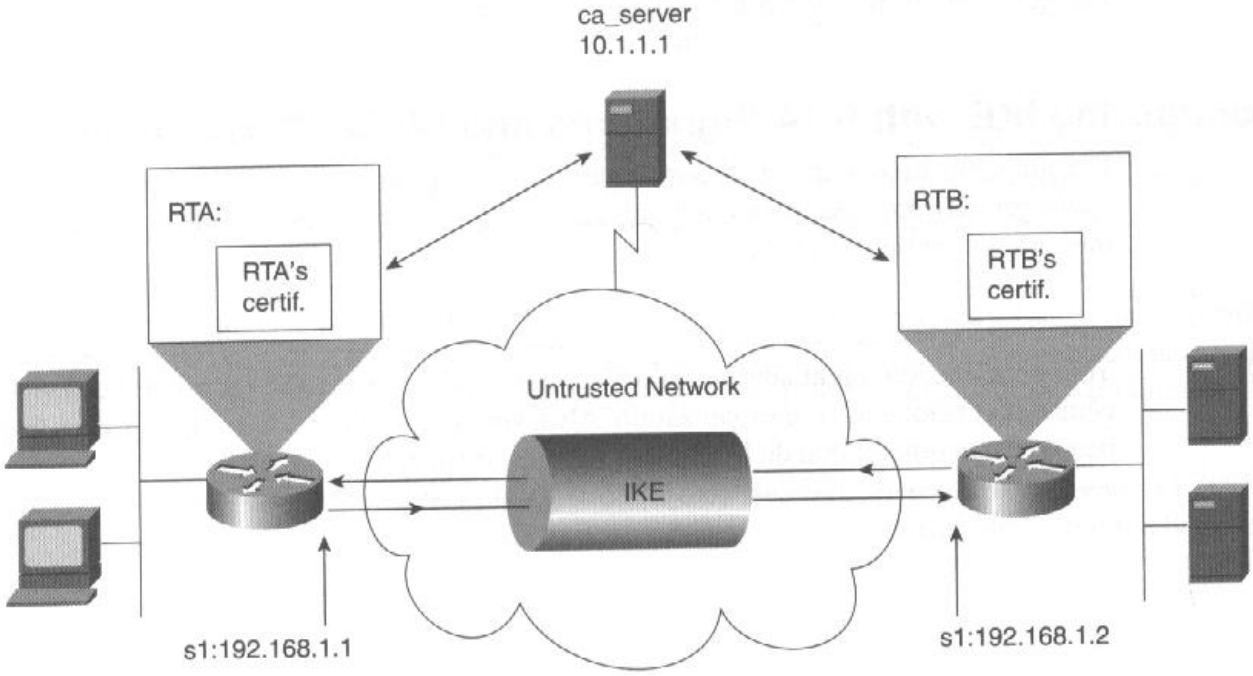
With digital certificate support, a router does not need to be configured with the public keys of each of its peers; instead, a router only needs to obtain its digital certificate from the CA. The digital certificate contains the router's public key (and other identity information) and the CA's signature for the certificate. When building an IKE SA with a peer, the router offers its digital certificate to the peer. The peer can then validate the authenticity of the certificate (by checking the CA's signature) and extract the public key contained within the certificate. See "Digital Signatures" and "Digital Certificates," earlier in this chapter.

CAs and supporting systems that provide digital certificates and public key management are collectively called the *public key infrastructure (PKI)*. PKI standards come from the IETF, International Telecommunication Union (ITU), RSA Data Security, and other organizations. IOS supports standards for PKI elements such as certificates, keys, and CAs. Consult Cisco
for a current list of supported PKI systems and standards.

NOTE A particularly relevant PKI document is RFC 2510.

The example configurations presented in this section on certificates are based on the network depicted in Figure 7-8. This figure describes two routers peering over an untrusted network with IKE and digital certificates with RSA signatures.

Figure 7-8 *Two Routers Peering with IKE and Digital Certificates with RSA Signatures*



In the preceding figure, both Router A and Router B store their own certificates locally in their IOS configurations. Both have already obtained their certificates at a prior time by following the certificate request, verification, and retrieval process with the CA server called **ca_server**. When Router A and Router B need to build an IKE SA, they exchange certificates as part of the IKE authentication process. Refer to "Tying All of the Pieces Together: A Comprehensive Example with IPsec and IKE" for a sequence including IKE and digital certificates.

Configure the IKE Transform Set for RSA Signature

As with pre-shared keys and RSA encryption, you must configure matching IKE transform sets in the routers to support RSA signature. The following IOS commands configure Router A so it can establish an IKE SA with Router B using RSA signature (for clarity, these commands include the context of the IOS prompt):

```
RTA#conf t
Enter configuration commands, one per line. End with CNTL/Z.
RTA(config)#crypto isakmp policy 7
RTA(config-isakmp)#authentication rsa-sig
RTA(config-isakmp)#hash sha
RTA(config-isakmp)#encryption des
RTA(config-isakmp)#lifetime 43200
RTA(config-isakmp)#exit
RTA(config)#_
```

The preceding commands create an IKE transform set with the following characteristics:

- Priority is 7.
- Authentication method is RSA signature (**rsa-sig**).
- Hashing algorithm is SHA-1 (**sha**).
- Encryption algorithm is 56-bit DES (**des**).
- Lifetime is 43,200 seconds (12 hours).

You must configure Router B with the same IKE transform set; otherwise, IKE negotiation will fail.

Configure CA Information

Both Router A and Router B need to be configured with the identity of the CA server so that they can communicate with it. The following commands configure the CA server information in Router A (Router B's configuration is similar):

```
RTB(config)#ip host ca_server 10.1.1.1
RTB(config)#crypto ca identity myca
RTB(ca-identity)#enrollment url http://ca_server
RTB(ca-identity)#enrollment mode ra
RTB(ca-identity)#query url ldap://ca_server
RTB(ca-identity)#crl optional
RTB(ca-identity)#exit
```

The command **ip host ca_server 10.1.1.1** adds an entry in the router's host table so that it can resolve the name **ca_server** to **10.1.1.1**. This command can be skipped if the router is configured with the **ip name-server** command and can resolve the name with DNS.

The command **crypto ca identity myca** adds an entry for the CA, names the CA **myca** (this can be any string), and begins CA identity config mode as indicated when the prompt changes to **ca-identity** in the next line.

The command **enrollment url http://ca_server** configures the router with the Uniform Resource Locator (URL) of the CA. This is required and should match the hostname configured by the preceding **ip host** command. The URL in this example assumes the script directory of the CA server is in a default location. This should apply to most cases; however, if the script directory is in a nonstandard location, include the full path to the script directory in the URL (for example, **enrollment url http://ca_server/cgi-bin/subdir1/subdir2**). Consult your CA for assistance in determining the correct URL.

The command **enrollment mode ra** is necessary only if the router is communicating with a *Registration Authority (RA)*. Some PKI systems split administration tasks across two servers: a CA that signs certificates and an RA that handles the certificate enrollment transactions. Consult your CA. If your CA does not use an RA, you should skip this command.

The command **query url ldap://ca_server** configures the router to use the Lightweight Directory Access Protocol (LDAP) for retrieving certificates from this server. This is usually configured in conjunction with an RA that supports the LDAP protocol. If your CA does not use LDAP, you should skip this step.

NOTE At the time of this writing, IOS interoperates with CA servers from Entrust Technologies and Verisign, using the Certificate Enrollment Protocol (CEP). The Entrust CA server requires the commands **enrollment mode ra** and **query url**; the Verisign CA server does not. See a bulletin at http://www.cisco.com/warp/public/778/security/821_pp.htm or search for "certificate authority support" on Cisco's Web site. Consult Cisco for a current list of supported PKI systems.

The command **crl optional** allows the router to accept a peer's certificate even if the *certificate revocation list (CRL)* is not available. Routers and other devices use the CRL to check for certificates that have been revoked. Certificates expire over time and can be retracted by the owner of the certificate or the CA. To ensure that a certificate is still valid, a device downloads a CRL from its CA and rejects certificates that are on the CRL. If the CRL is not available (if the CA server is down and the CRL is not stored on the router), **crl optional** enables the router to accept a peer's certificate and continue.

Without the **crl optional** command, the security policy is more strict: The router must possess and check the CRL before accepting a peer's certificate. This might be desirable for enhanced security, but it adds an additional step that might require troubleshooting.

Finally, **exit** ends CA identity mode and returns to global configure mode.

Retrieve the Certificate of the CA

Each device participating in authentication with digital certificates needs to retrieve the CA's certificate. This provides the device with the CA's public key that is used to verify certificates the CA has signed.

The following command instructs Router A to contact the CA and retrieve the CA's certificate (Router B also requires this command). The administrator commands are highlighted in boldface. Refer to Figure 7-8 for the topology.

```
RTA(config)#crypto ca authenticate myca
Certificate has the following attributes:
Fingerprint: B60065B9 63EC9373 D33B4548 CAEF52B9

% Do you accept this certificate? [yes/no]: y
RTA(config)#
```

The command **crypto ca authenticate myca** is a global config command and tells a router to get the certificate of the server **myca**.

In the next lines, the router retrieves the *fingerprint* of the CA's certificate and asks you to verify it. The fingerprint is a cryptographic number calculated by the CA and is used to verify the integrity of the certificate. Check the fingerprint your router receives against the fingerprint provided by your CA. Matching fingerprints validates the CA's certificate and ensures that the certificate was not altered in transit.

After accepting the certificate, you can view it with the **show crypto ca certificates** command:

```
RTA#sh cry ca certificates

CA Certificate
  Status: Available
  Certificate Serial Number: 3654B2FF
  Key Usage: Not Set
```

Generate Public and Private Keys

After configuring the CA information, you need to generate the router's public and private keys. This is performed with the command **crypto key generate rsa** introduced in the section "Configuring IKE with RSA Encryption" earlier in this chapter.

Some PKI servers (CA or RA) require each device to have two public/private key pairs—a total of four keys. One key pair is used for encryption and the other key pair is used for digital signatures. The following command generates two such key pairs. Before you issue this command, ensure that your router has a hostname and a domain name (configured with the **hostname** and **ip domain-name** global config commands):

```
RTA(config)#crypto key generate rsa usage-keys
The name for the keys will be: RTA.cisco.com
Choose the size of the key modulus in the range of 360 to 2048 for your
  Signature Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1536
Generating RSA keys ...

%SYS-3-CPUHOG: Task ran for 2812 msec (109/109), process = Key Proc, PC = 72B1EE.
[OK]
Choose the size of the key modulus in the range of 360 to 2048 for your
  Encryption Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1536
Generating RSA keys ...

%SYS-3-CPUHOG: Task ran for 2048 msec (17/9), process = Key Proc, PC = 72A3A8.
[OK]

RTA(config)#_
```

The command **crypto key generate rsa usage-keys** runs a small program within global configure mode that creates *two* RSA key pairs. These key pairs are submitted to the PKI server (CA or RA) with a request for a certificate. See "Send Public Keys to the CA and Get Your Certificate," later in this chapter.

If your PKI server (CA or RA) requires only one public/private key pair, use the command **crypto key generate rsa** instead of **crypto key generate rsa usage-key**.

NOTE The system message **%SYS-3-CPUHOG** in the preceding output indicates that the router is experiencing high CPU utilization. This is expected because key generation is CPU-intensive. As mentioned in an earlier note, generate keys before putting the router into service.

Send Public Keys to the CA and Get Your Certificate

To submit the router's public key (or keys) to the PKI server and request a certificate, issue the **crypto ca enroll** command:

```
RTA(config)#crypto ca enroll myca
% Start certificate enrollment ..
% Create a challenge password. You will need to verbally provide this
   password to the CA Administrator in order to revoke your certificate.
   For security reasons your password will not be saved in the
configuration.
   Please make a note of it.

Password: <type a password here>
Re-enter password: <re-enter password>

% The subject name in the certificate will be: RTA.cisco.com
% Include the router serial number in the subject name? [yes/no]: yes
% Include an IP address in the subject name? [yes/no]: no
Request certificate from CA? [yes/no]: yes
% Certificate request sent to Certificate Authority
% The certificate request fingerprint will be displayed.
% The 'show crypto ca certificate' command will also show the
fingerprint.

RTA(config)#
    Signing Certificate Request Fingerprint:
    AABB1FD9 594B3209 39D9058C 2A19205C
  Encryption Certificate Request Fingerprint:
    6289344E AEFDC0AE BCB06E1B ECD62171
```

The command **crypto ca enroll myca** initiates a request for a certificate and prompts you for the following (indicated by boldface in the preceding output):

- **Password**—You need to supply a password to the CA with your request. If you ever have to revoke a certificate, the CA will ask for this password to prevent fraudulent revocation requests.
- **Include router serial number**—You may include your router's serial number in the certificate. This is not required by IPsec or IKE, but might be required by your CA.
- **Include IP address**—You may include your router's IP address in the certificate. Generally, you will not include this because IP addresses can change and such a change would require a new certificate.
- **Request certificate from CA**—When you are ready to submit the request, answer **yes** to this prompt to send the request.

Next, the router displays the fingerprints associated with the certificate requests and periodically queries the PKI server until the CA fulfills the request. By default, the router queries the server every minute until the certificate is ready for retrieval. The query interval can be adjusted by the **enrollment retry-period** command (CA identity config mode).

The following output of **show crypto ca certificates** indicates that a certificate request is still pending:

```
RTA#sh cr ca certificates
Certificate
  Subject Name
    Name: RTA.cisco.com
    IP Address: 192.168.1.1
  Status: Pending
  Key Usage: Signature
    Fingerprint: AABB1FD9 594B3209 39D9058C 2A19205C
```

A status of **Pending** means the CA server has not yet completed the processing of the certificate request. As mentioned in the earlier section, "Digital Certificates and Certification Authorities," the CA should manually verify that the public key and the owner are legitimate before issuing a certificate.

When the router queries the server, discovers that the certificate is ready, and downloads the certificate, the following system message is displayed:

```
RTA(config)#
%CRYPTO-6-CERTRET: Certificate received from Certificate Authority
RTA(config)#
```

The router now has its certificate. You can verify this by issuing the **show crypto ca certificates** command:

```
RTA#show crypto ca certificates
Certificate
  Subject Name
    Name: RTA.cisco.com
    IP Address: 192.168.1.1
  Status: Available
  Certificate Serial Number: 3654B3B6
  Key Usage: Signature
```

A status of **Available** confirms that the router has its certificate and is ready to distribute it to peers.

When both routers, Router A and Router B in Figure 7-8, have their certificates, IKE configuration for this example is complete.

Additional Commands for IKE

The following paragraphs provide some additional commands for IKE configuration.

The command **group 2** configures the IKE transform set for Diffie-Hellman group 2 (1024-bit numbers). The default is group 1 (768-bit numbers). Group 2 provides greater security but takes longer to compute (the computational time can make a difference on routers with smaller CPUs):

```
RTA(config)#crypto isakmp policy 10
RTA(config-isakmp)#group 2
```

The command **crypto isakmp identity hostname** applies to IKE with pre-shared keys and tells the router to use its hostname (RTA.cisco.com, for example) when it identifies itself to peers:

```
RTA(config)#crypto isakmp identity hostname
```

This might be necessary if the router uses multiple interfaces to peer with other devices. By default, routers identify themselves with their IP address (**crypto isakmp identity address**). It is recommended that peers identify themselves to each other the same way: with addresses or with hostnames.

The command **crypto key zeroize rsa** erases the router's RSA keys (public and private) from the configuration:

```
RTA(config)#crypto key zeroize rsa
```

The command **no addressed-key** removes a peer's public key from the router's public key chain. The following example removes the public key for peer 192.168.1.2:

```
RTA(config)#cry key pubkey-chain rsa
RTA(config-pubkey-chain)#no addressed-key 192.168.1.2
```

The command **no certificate** removes a certificate from the router's configuration. The following example removes a certificate with serial number **3654B30A**:

```
RTA(config)#cry ca certificate chain myca
RTA(config-cert-chain)#no certificate 3654B30A
```

NOTE The enable mode command **show crypto ca certificate** displays certificates and their associated serial numbers.

The command **crypto ca crl request myca** tells the router to contact the CA server **myca** (configured by **crypto ca identity myca**) and download a CRL:

```
RTA(config)#crypto ca crl request myca
```

Skip this command if your CA does not support CRLs. See also "Configure CA Information," earlier in this chapter.

With the **crypto ca certificate query** command, the router downloads certificates and CRLs directly from the CA as they are needed and does not store the data to NVRAM:

```
RTA(config)#crypto ca certificate query
```

By default, when a router initially receives certificates and CRLs, it stores them locally in NVRAM (data in NVRAM is always available to the router even after a power-down). The **crypto ca certificate query** command conserves NVRAM space, but might result in loss of service if the CA is unavailable and the router cannot download certificates and CRLs.

Validating IKE Configuration

To view the IKE transform sets (policies) and verify IKE configuration, issue the **show crypto isakmp policy** command. The following is an example output:

```
RTA#sh cry isakmp pol
Protection suite of priority 7
        encryption algorithm:   DES - Data Encryption Standard (56 bit keys).
        hash algorithm:         Secure Hash Standard
        authentication method:  Rivest-Shamir-Adleman Signature
        Diffie-Hellman group:   #1 (768 bit)
        lifetime:               43200 seconds, no volume limit
Protection suite of priority 9
        encryption algorithm:   DES - Data Encryption Standard (56 bit keys).
        hash algorithm:         Secure Hash Standard
        authentication method:  Rivest-Shamir-Adleman Encryption
        Diffie-Hellman group:   #1 (768 bit)
        lifetime:               43200 seconds, no volume limit
Protection suite of priority 10
        encryption algorithm:   DES - Data Encryption Standard (56 bit keys).
        hash algorithm:         Message Digest 5
        authentication method:  Pre-Shared Key
        Diffie-Hellman group:   #1 (768 bit)
        lifetime:               180 seconds, no volume limit
Default protection suite
        encryption algorithm:   DES - Data Encryption Standard (56 bit keys).
        hash algorithm:         Secure Hash Standard
        authentication method:  Rivest-Shamir-Adleman Signature
        Diffie-Hellman group:   #1 (768 bit)
        lifetime:               86400 seconds, no volume limit
```

The default transform set (line **Default protection suite**) is always present, is listed last, and has the lowest priority.

When Are IKE SAs Established?

After you configure IKE, the router does not initiate an IKE SA until it is needed for establishing the IPsec SAs that protect user traffic. This means you need to configure IPsec before you can see an IKE SA and know with certainty that IKE is working. The following section covers how to configure IPsec.

NOTE IKE uses UDP port number 500. Make sure you do not have any access lists configured on interfaces that might be undesirably blocking UDP port 500.

Configuring IPsec

After IKE is configured, the next step is to configure policies called *crypto maps* that tell a router how to build IPsec SAs within the IKE SA.

This section covers

- Crypto Maps
- Crypto Map Configuration Overview
- Configuring Crypto Access Lists
- Crypto Access Lists: An Example
- Configuring IPsec Transform Sets
- Configuring and Applying Crypto Maps
- When Are SAs Established?
- Configuring IPsec SA Lifetimes
- Configuring Perfect Forward Secrecy
- Configuring Dynamic Crypto Maps
- Tunnel Endpoint Discovery
- Validating IPsec Configuration

Crypto Maps

A crypto map defines the IPsec policies on a router. These policies include:

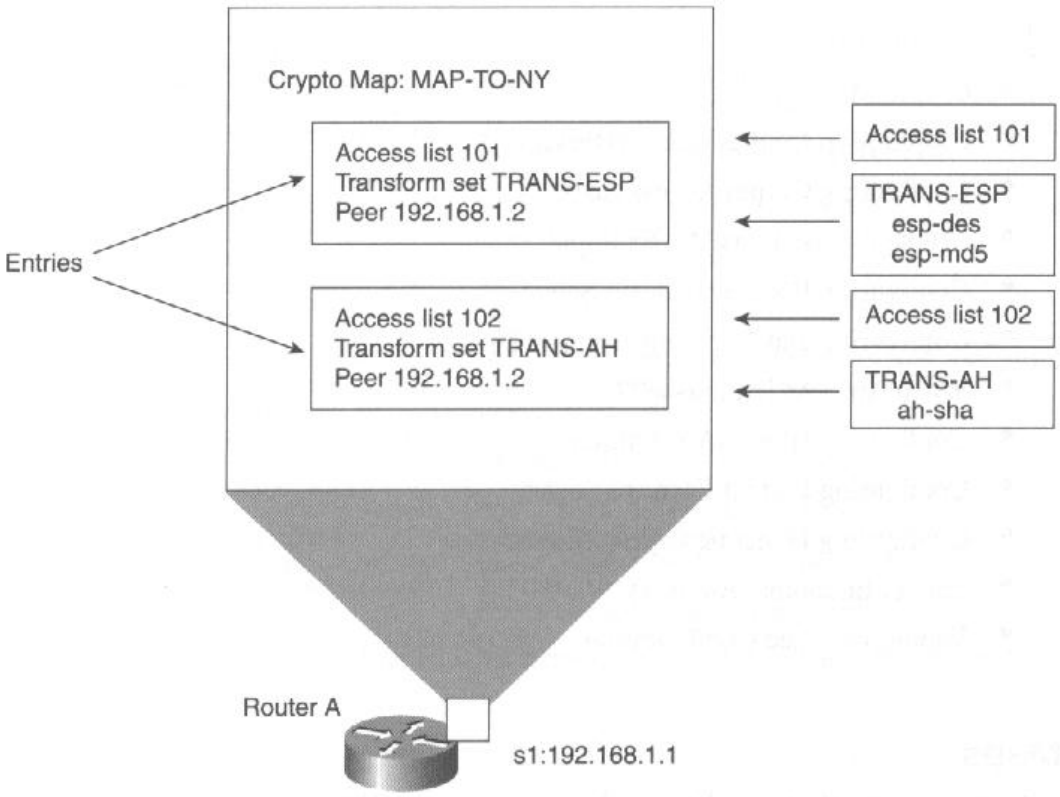
- The traffic that is protected by IPsec
- The transforms that are applied to the IPsec SAs
- The identity of peers to which the protected traffic should be sent
- Additional rules, such as SAs lifetimes and PFS

Crypto maps are applied to router interfaces and can contain one or more *entries*. An entry defines a policy for packets that match an extended access list (called a *crypto access list*). By configuring multiple entries, you can define different protection schemes (IPsec transform sets) for different types of traffic, based on IP address, application port, or anything else that can be matched with an extended access list (see also Chapter 6 on access lists).

NOTE An interface can have only one crypto map. Use multiple entries within a crypto map to assign different policies for different types of traffic.

Figure 7-9 is a conceptual view of a crypto map.

Figure 7-9 *Conceptual View of a Crypto Map Applied to Router A's Serial1 Interface*



As depicted in Figure 7-9, each crypto map entry contains the following elements:

- **Crypto access list**—An extended access list that defines the packets protected by the entry.
- **Transform set**—A list of one or more transforms considered acceptable for protecting the traffic covered by the crypto access list. See "IPsec Concepts" earlier in this chapter for more information.
- **Peer's identity**—The IP address or hostname of the remote peer that the router sends the protected traffic to.
- **Additional parameters**—The lifetime of the IPsec SAs and PFS (on or off). If IKE is not implemented (not recommended), the session's keys and SPIs are manually configured in the entry as well.

Crypto Map Configuration Overview

Your goal is to configure the router so it protects the data you want with the security protocols and algorithms you want. This entails the following:

- Creating the crypto map elements (crypto access lists and transform sets)

- Configuring the crypto map
- Applying the crypto map to a router interface
- Validating the configuration

The following sections describe these steps.

Configuring Crypto Access Lists

As mentioned previously, crypto access lists define the scope or set of packets that need protection of an IPsec SA.

You configure crypto access lists no differently than regular extended access lists. However, crypto access lists have a different purpose and meaning than regular extended access lists. The following list describes the unique characteristics of crypto access lists:

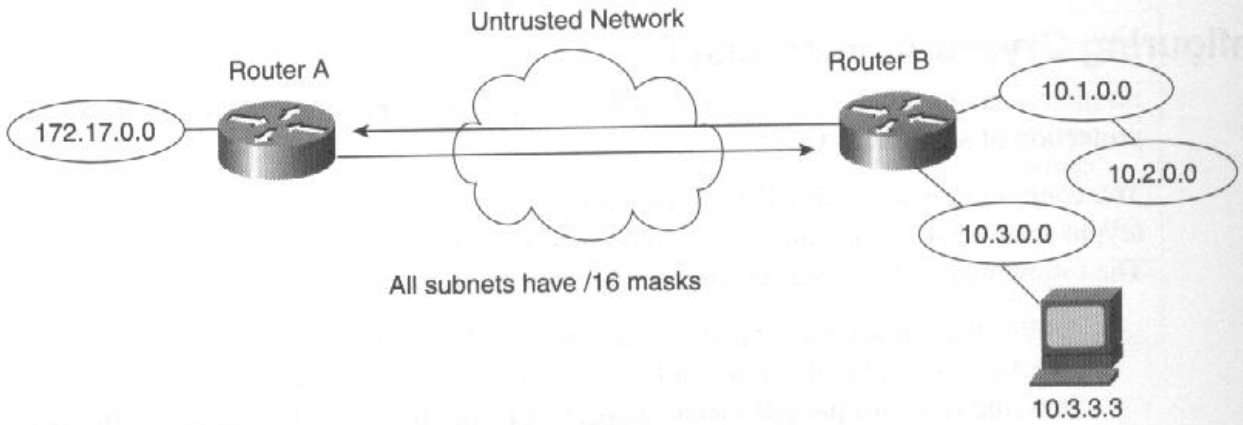
- Unlike regular access lists, crypto access lists do not specify the traffic to filter on an interface, but rather the outbound traffic that is to be protected with IPsec. In crypto access lists, the keyword **permit** means protect. For example, the rule **access-list 101 permit ip host 192.168.10.3 host 10.1.1.4** means "protect all IP traffic from 192.168.10.3 to 10.1.1.4" (when this access list is used as a crypto access list).
- IPsec processing is skipped when a packet is denied by a crypto access list. This means the router simply processes the packet normally, as if IPsec never existed. Recall from Chapter 6 that there is an invisible deny-any rule at the end of every access list. This includes crypto access lists.
- Each rule in a crypto access list defines the traffic to be protected by a single IPsec SA. This means a single SA will protect a large set of packets if the rule is broad (permit all traffic from Subnet X to Subnet Y, for example). On the other hand, an SA might protect a small set of packets if the rule is narrow (permit Telnet traffic from Host A to Host B).
- A crypto access list not only defines the outbound traffic that requires IPsec protection but also checks *inbound* traffic to enforce consistent policies. When a router sends a protected flow to a peer, it expects the flow in the return direction to also be protected. The router checks this by comparing inbound packets to its crypto access lists but with the logic reversed. The reversed logic means the source and destination criteria are swapped. Packets that arrive from a remote peer and match the reverse logic of a crypto access list will be dropped if they are *not* protected by IPsec. This is a safety measure and ensures that traffic the router expects to be protected is indeed protected.

NOTE Because a router checks inbound traffic against its crypto access list, configuring peers with matching, mirror-image access lists is crucial. See the following section, "Crypto Access Lists: An Example."

Crypto Access Lists: An Example

Consider the scenario depicted in Figure 7-10 with two routers that must peer across an untrusted network and provide IPsec services on behalf of devices located in multiple subnets.

Figure 7-10 *Scenario for Configuring Crypto Access Lists*

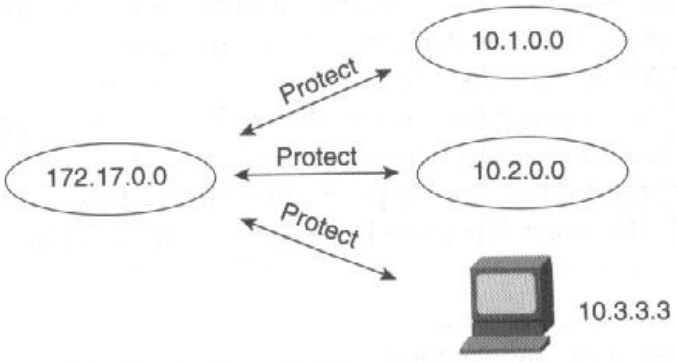


Suppose all subnets have a /16 mask and you are given the following requirements:

- Traffic between subnets 172.17.0.0 and 10.1.0.0 requires IPsec.
- Traffic between subnets 172.17.0.0 and 10.2.0.0 requires IPsec.
- Traffic between subnet 172.17.0.0 and host 10.3.3.3 requires IPsec.
- All other traffic between subnets 172.17.0.0 and 10.3.0.0 does not require IPsec.

To simplify the overall task at hand, you can draw a simple logical diagram like Figure 7-11. This figure depicts the protection required between the subnets.

Figure 7-11 *Protection Required Between the Subnets in Figure 7-10*



The following crypto access lists for Router A and Router B define the traffic that needs to be protected based on the stated requirements.

For Router A:

```
hostname RTA
<lines deleted for brevity>
!
access-list 101 permit ip 172.17.0.0 0.0.255.255 10.1.0.0 0.0.255.255
access-list 101 permit ip 172.17.0.0 0.0.255.255 10.2.0.0 0.0.255.255
access-list 101 permit ip 172.17.0.0 0.0.255.255 host 10.3.3.3
```

For Router B:

```
hostname RTB
<lines deleted for brevity>
!
access-list 102 permit ip 10.1.0.0 0.0.255.255 172.17.0.0 0.0.255.255
access-list 102 permit ip 10.2.0.0 0.0.255.255 172.17.0.0 0.0.255.255
access-list 102 permit ip host 10.3.3.3 172.17.0.0 0.0.255.255
```

First, notice that crypto access lists are nothing more than extended access lists—except their purpose is different and they are applied with crypto maps. As mentioned previously, the router applies IPsec services (encryption, integrity, and the like) to outbound traffic that is permitted by the crypto access list. *Permit* means *protect*.

You write crypto access lists from the perspective of traffic exiting the router and destined to the remote peer. Therefore, Router A matches all packets *from* subnet 172.17.0.0/16 *to* destination subnets 10.1.0.0/16 and 10.2.0.0/16, plus the destination host 10.3.3.3.

Router B matches the same traffic as Router A but has the reverse criteria. From Router B's perspective, the source traffic is subnets 10.1.0.0/16 and 10.2.0.0/16 plus host 10.3.3.3 and the destination subnet is 172.17.0.0/16.

Notice that every access list rule in Router B is a mirror-image of a corresponding rule in Router A. The two access lists 101 and 102 are called *mirror-image crypto access lists*. Mirror-image crypto access lists are crucial to the proper operation of IPsec. If Router B's crypto access list is not a mirror-image of Router A's list, communication problems might occur.

Without mirror-image crypto access lists, problems occur because the access lists do not agree and do not protect the same set of traffic. IPsec packets might flow in one direction but not the other. A modified (and incorrect) configuration for Router B follows and describes the problem (Router A's list is unchanged):

```
hostname RTB
<lines deleted for brevity>
!
access-list 102 permit ip 10.1.0.0 0.0.255.255 172.17.0.0 0.0.255.255
access-list 102 permit ip 10.2.0.0 0.0.255.255 172.17.0.0 0.0.255.255
access-list 102 permit ip 10.3.0.0 0.0.255.255 172.17.0.0 0.0.255.255
```

Notice that the last rule in access list 102 is not a mirror-image of the last rule in access list 101. The criteria **10.3.0.0 0.0.255.255** in list 102 matches all addresses in subnet 10.3.0.0/16, but the criteria **host 10.3.3.3** in list 101 matches just one address in the subnet. Access list 102 covers a wider range of addresses than list 101. This will cause problems when Router A sends packets destined for 10.3.0.0/16 other than 10.3.3.3.

To demonstrate the problem with Router B's modified and incorrect list, suppose a host in subnet 172.17.0.0/16 sends a packet to host 10.3.3.10 (not 10.3.3.3). The following sequence assumes that the crypto access lists have been added to active crypto maps.

NOTE Like regular access lists, crypto access lists do not do anything until they are applied to interfaces. The section "Configuring and Applying Crypto Maps" later in this chapter covers how to add crypto access lists to crypto maps that then get applied to interfaces.

1. Router A receives the packet and determines that packets to 10.3.3.10 need to be forwarded to Router B.
2. Router A checks its crypto access list and finds that the packet does not match any of the permit statements. Router A's list matches packets from 172.17.0.0/16 to host 10.3.3.3, but not to host 10.3.3.10.
3. Because the packet does not match the crypto access list, the router transmits the packet normally without any IPsec services (no AH or ESP headers, no encryption).
4. The packet arrives at Router B.
5. Router B checks the inbound packet against every rule in its crypto access list while reversing the logic of each rule (recall that reversing the logic means the source and destination criteria are swapped). The packet does not match the first two rules of the list but *does* indeed match the third rule (source criteria is 172.17.0.0 0.0.255.255 and destination criteria is 10.3.0.0 0.0.255.255 by reverse logic).
6. Router B drops the packet, because it expects the packet to be protected with IPsec but it is not.
7. The unfortunate result: Hosts in 172.17.0.0/16 cannot contact hosts in 10.3.0.0/16 with or without IPsec. The exception is host 10.3.3.3, which is included in the lists of both Router A and Router B.

NOTE The use of the keyword **any** in crypto access lists can cause problems and is strongly discouraged. The **any** criterion is broad and requires protection for all inbound packets. Thus, inbound packets that lack protection (such as routing protocol updates and other control packets) might be undesirably dropped. You should always explicitly define the traffic that needs IPsec protection.

Configuring IPsec Transform Sets

A transform set (also called a *transform proposal*) defines the security protocols and algorithms that protect traffic for a given IPsec SA. Before two devices can establish an IPsec SA, they must negotiate and agree on a common transform set.

To configure a transform set, use the **crypto ipsec transform-set** global configuration command:

```
RTA(config)#crypto ipsec transform-set TRANS-ESP esp-des esp-md5-hmac
RTA(cfg-crypto-trans)#mode tunnel
RTA(cfg-crypto-trans)#exit
```

The command **crypto ipsec transform-set TRANS-ESP esp-des esp-md5-hmac** creates a transform set called **TRANS-ESP**. This transform set includes two ESP transforms: 56-bit DES encryption (**esp-des**) and MD5 HMAC integrity/authentication (**esp-md5-hmac**).

The command **mode tunnel** dictates that all SAs created with this transform set will be tunnel mode SAs. The other choice is **mode transport**. See "Transport and Tunnel Modes," earlier in this chapter.

NOTE The command **mode transport** is useful for secure router management—that is, for communicating with the router as an IPsec host. You can, for example, use a transport mode SA to Telnet to the router securely. Transport mode can also be used to secure SNMP traffic to and from the router.

NOTE After creating the transform set, you must add it to a crypto map to put it to use. See "Configuring and Applying Crypto Maps," later in this chapter.

Here's another example. The following commands create a transform set that uses both the AH and ESP protocols:

```
RTA(config)#crypto ipsec transform-set TRANS-AH-ESP ah-sha-hmac esp-des
RTA(cfg-crypto-trans)#mode tunnel
RTA(cfg-crypto-trans)#exit
```

The command **crypto ipsec transform-set TRANS-AH-ESP ah-sha-hmac esp-des** creates a transform set called **TRANS-AH-ESP**. This transform set includes an AH transform and an ESP transform: AH with SHA-1 hashing for integrity and authentication and ESP with 56-bit DES for encryption.

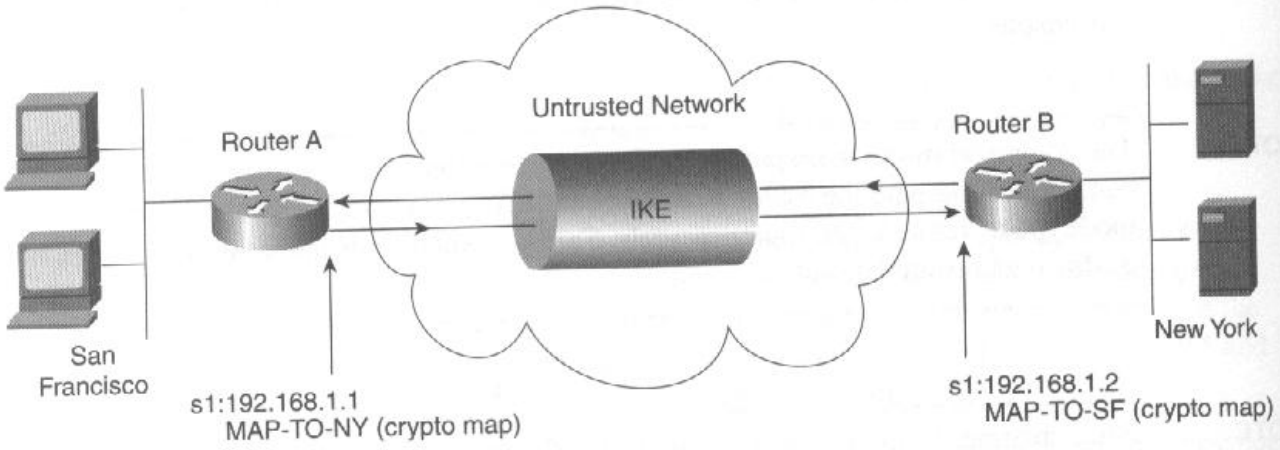
NOTE As mentioned earlier in "Authentication Header and Encapsulating Security Payload," protecting bidirectional traffic with both AH and ESP requires two pairs of SAs: an SA pair for AH and another SA pair for ESP (a total of four unidirectional IPsec SAs).

Configuring and Applying Crypto Maps

After configuring crypto access lists and transform sets, you can add them to a crypto map.

Consider the network in Figure 7-12 with two routers that peer over an untrusted network. Assume that IKE, crypto access lists, and transform sets are configured and a crypto map is now needed.

Figure 7-12 *A Network with a Basic Crypto Map Configuration*



In the preceding diagram, Router A's serial interface to the untrusted network is 192.168.1.1.
A crypto map named MAP-TO-NY is applied to this interface (the configuration commands follow). Likewise, Router B's serial interface is 192.168.1.2 and has a crypto map called
MAP-TO-SF.

The following commands create a crypto map on Router A (for clarity, the context of the IOS prompt is included):

```
RTA#conf t
Enter configuration commands, one per line. End with CNTL/Z.
RTA(config)#crypto map MAP-TO-NY 20 ipsec-isakmp
RTA(config-crypto-map)#match address 101
RTA(config-crypto-map)#set transform-set TRANS-ESP TRANS-AH-ESP
RTA(config-crypto-map)#set peer 192.168.1.2
RTA(config-crypto-map)#exit
RTA(config)#int s1
RTA(config-if)#crypto map MAP-TO-NY
```

The command **crypto map MAP-TO-NY 20 ipsec-isakmp** creates a crypto map entry with a sequence of 20 for a crypto map called **MAP-TO-NY** (the crypto map is created when its first entry is created). Although this example contains just one entry, crypto maps may contain multiple entries to designate multiple peers, transform sets, and access lists. The sequence number prioritizes the crypto map entries. As the router compares packets to the crypto map, it examines entries in the order of their sequence number (lower sequence numbers are examined first). For this example, a sequence of **20** was chosen so that future entries may be placed before or after this entry. The keyword **ipsec-isakmp** indicates that IKE is used to manage the SAs for this entry.

NOTE In addition to IKE, which is specified by the **ipsec-isakmp** keyword, crypto maps support two other options: **ipsec-manual** (IPsec without IKE) and **cisco** (Cisco's pre-IPsec encryption feature called Cisco Encryption Technology, or CET). Consult the IOS documentation for configuring **ipsec-manual** or **cisco**.

The command **match address 101** assigns crypto access list 101 to this entry. Outbound packets that match this list are protected with IPsec. Inbound packets that match the reverse logic of the list are expected to be protected.

The command **set transform-set TRANS-ESP TRANS-AH-ESP** defines the transform sets that are acceptable for protecting the traffic covered by the crypto access list. When negotiating IPsec SAs with the remote peer (Router B), the router proposes transform sets in the order listed by this command (this router's first choice is the transform set TRANS-ESP). Router A and Router B must agree to use a common transform set (a common set of protocols and algorithms) before an SA can be established. **TRANS-ESP** and **TRANS-AH-ESP** are the names of transform sets previously created by the **crypto ipsec transform-set** command. The transform set *names* (TRANS-ESP, TRANS-AH-ESP) are locally significant and do not have to be the same on both routers.

The command **set peer 192.168.1.2** defines the remote peer, Router B, with which this router builds the IPsec SA and to which it subsequently sends the protected traffic. Multiple peers can be configured by repeating the **set peer** command. This provides a level of redundancy for when SAs are established: If the first peer is not reachable, the router attempts to establish the SA with the next peer in the entry.

The interface configuration command **crypto map MAP-TO-NY** applies the crypto map to the router's Serial1 interface (selected by the command **int s1**). Like access lists, crypto maps do not do anything until you apply them to an interface. The proper place to apply the crypto map is the interface where the protected traffic *exits* the router: the interface that points in the direction of the remote peer. In this example, Router A's Serial1 interface is the exit point (refer to Figure 7-12).

The following is the corresponding configuration on Router B (only the relevant crypto map lines are shown):

```
RTB#sh run
Current configuration:
hostname RTB
<lines deleted for brevity>
!
crypto map MAP-TO-SF 20 ipsec-isakmp
 match address 102
 set transform-set B-TRANS1 B-TRANS2
 set peer 192.168.1.1
!
interface Serial1
 ip address 192.168.1.2 255.255.255.0
 crypto map MAP-TO-SF
!
```

The crypto access list **102** must be a mirror image of list 101 on Router A, and at least one of the transform sets (**B-TRANS1** or **B-TRANS2**) must match one of Router A's transform sets (TRANS-ESP and TRANS-AH-ESP). A match means the transform sets share the same protocols (AH, ESP) and algorithms (DES or MD5, for example).

NOTE Crypto access lists are crypto map elements and interoperate with regular packet-filtering access lists that might exist on an interface. Packets blocked by regular access lists are not processed by IPsec.

When Are SAs Established?

The following list describes the logic of a router as it pertains to the setup of IPsec and IKE SAs:

- When a packet requiring IPsec needs to be transmitted, the router checks for an existing IPsec SA suitable for protecting the packet. An SA is suitable if its source and destination criteria (as defined by a crypto access list rule) match the packet. If a suitable IPsec SA exists, the packet is sent using the transform set and peer parameters associated with the SA.
- If a suitable IPsec SA does not exist for the packet, the router needs to set up a new SA. When IKE is configured, the router first checks for an existing IKE SA before creating the IPsec SA. If the router has an existing IKE SA with the destination peer, the IPsec SA is created, using the IKE SA as a secure channel.
- If the router does not have an existing IKE SA with the destination peer, the router needs to set it up. After the IKE SA is in place, the IPsec SA is created, using the IKE SA as a secure channel.

TIP IKE uses UDP port number 500. Make sure you do not have any regular, packet filtering access lists that might be undesirably blocking UDP port 500.

Configuring IPsec SA Lifetimes

The following commands modify the lifetimes associated with IPsec SAs:

```
RTA(config)#crypto map MAP-TO-NY 20 ipsec-isakmp
RTA(config-crypto-map)#set security-association lifetime seconds 2700
RTA(config-crypto-map)#set security-association lifetime kilobytes 2000000
```

The command **set security-association lifetime seconds 2700** sets the lifetime of IPsec SAs created by this crypto map entry to 2700 seconds (45 minutes). The default is 3600 seconds (60 minutes).

The command **set security-association lifetime kilobytes 2000000** sets the *volume* lifetime of IPsec SAs created by this crypto map entry to 2,000,000 kilobytes (approximately 10 Mbps per second for one half hour). This means that after 2,000,000 kilobytes have been transmitted over an SA, the SA will expire. The default is 4,608,000 kilobytes (10 Mbps per second for one hour).

An IPsec SA expires when the first of the two lifetimes (seconds or kilobytes) is reached.

NOTE Shorter lifetimes provide better security because the keys associated with the SAs change more frequently. However, rekeying more frequently results in an increased load on the router's CPU.

Shortly before an IPsec SA expires, the router builds a new SA (called a *rollover SA*) to ensure an SA is ready to use when the old one expires. This provides a smooth transition and minimal disruption to the users. Rollover SAs are created approximately 30 seconds before the **seconds** lifetime or approximately 256 kilobytes less than the **kilobytes** lifetime (whichever comes first).

NOTE When two Cisco routers propose different lifetimes during IPsec SA negotiation, they will agree to use the shorter lifetime.

Configuring Perfect Forward Secrecy

The following commands configure a crypto map entry for PFS:

```
RTA(config)#crypto map MAP-TO-NY 20 ipsec-isakmp
RTA(config-crypto-map)#set pfs group1
```

The command **set pfs group1** tells the router to use PFS on all IPsec SAs created with this entry. By default, PFS is off. The keyword **group1** specifies Diffie-Hellman group 1 (768-bit numbers). The other option, **group2**, specifies Diffie-Hellman group 2 (1024-bit numbers). Group 2 provides greater security but requires more time to compute. See "Internet Key Exchange" earlier in this chapter for more on PFS.

NOTE PFS provides better security than the alternative, which is the exchange of encrypted *nonces*. A nonce is a randomly generated number meant for one-time use. For instance, Alice and Bob can establish a new shared key by exchanging nonces that are encrypted with a shared key they already know. This is considered less secure than PFS because the new key is derived from the old key—discovering the old key helps an attacker find the new key too. With PFS, on the other hand, the new key is created on its own with the Diffie-Hellman algorithm. The downside of PFS is the computational overhead required to generate a new Diffie-Hellman key each time.

Configuring Dynamic Crypto Maps

When a router has numerous remote peers, configuring a crypto map entry for every peer can be laborious. This is especially true when remote access users dial into a central router: Manually configuring each user as a peer in the router's crypto map is impractical. Remote users typically have dynamically assigned IP addresses, so there's no way to predict a remote peer's address and program that into a crypto map.

Dynamic crypto maps simplify large peering configurations by providing templates of basic IPsec requirements. The dynamic crypto map mandates a set of basic requirements and leaves other parameters, such as the peers' IP addresses, undefined. If a peer can authenticate and establish an IKE SA, and if the peer meets the basic requirements defined by the dynamic crypto map, the peer is allowed an IPsec SA with the router.

NOTE Dynamic crypto maps require IKE. IKE establishes dynamic IPsec SAs and authenticates the remote peer.

Dynamic crypto maps are nothing more than crypto maps that are missing some parameters. The missing parameters represent the information that the router does not know about the other peer and does not require from the peer to successfully establish an IPsec SA. Typically, the missing parameter is the peer's IP address (normally configured with the **set peer** command). This provides scalability when there are many peers because the router does not need to know and does not require the peers' IP addresses ahead of time.

The following is an example configuration of a dynamic crypto map:

```
crypto dynamic-map DYN-MAP-DIALIN 20
 match address 101
 set transform-set TRANS-ESP TRANS-AH-ESP
!
crypto map MYMAP 500 ipsec-isakmp dynamic DYN-MAP-DIALIN
!
interface Serial1
 ip address 192.168.1.1 255.255.255.0
 crypto map MYMAP
```

The command **crypto dynamic-map DYN-MAP-DIALIN 20** creates an entry with a sequence of **20** for a dynamic crypto map called **DYN-MAP-DIALIN**. As with regular crypto maps, the sequence number prioritizes the map's entries.

The command **match address 101** assigns crypto access list 101 to this entry. As with regular crypto maps, the list defines the traffic that requires IPsec protection and checks inbound packets to ensure consistent policy. Inbound packets that match the reverse logic of the list are expected to be protected—if they are not, the packets are dropped.

The command **set transform-set TRANS-ESP TRANS-AH-ESP** defines the transform sets accepted by this router. When a remote peer initiates an IPsec SA with this router, it must propose a matching transform set or the negotiation will fail.

Notice that the dynamic crypto map lacks the **set peer** command found in regular crypto maps. This means the map accepts any peer that passes IKE negotiation (the authentication step) and proposes a matching transform set. This eliminates the task of having to configure each peer manually (the main benefit of dynamic crypto maps).

NOTE Recall that the choices for IKE authentication are pre-shared keys, RSA encryption, and RSA signatures with digital certificates.

The command **crypto map MYMAP 500 ipsec-isakmp dynamic DYN-MAP-DIALIN** binds the dynamic crypto map to an entry (sequence of 500) in a regular crypto map called MYMAP. This syntax allows you to configure multiple dynamic crypto maps in a single crypto map or to mix dynamic crypto maps with regular, static map entries.

NOTE When mixing dynamic crypto map entries with regular entries in a crypto map, set the dynamic crypto map entries to be the highest sequence numbers (lowest priority). This is why the example uses a sequence of 500 for the dynamic crypto map entry.

The command **crypto map MYMAP** applies MYMAP, which includes the dynamic crypto map, to interface Serial1.

NOTE By default, dynamic crypto maps can only answer *incoming* peer requests for IKE and IPsec SAs. They cannot initiate outbound SAs to remote peers. The exception to this is dynamic crypto maps with Cisco's *Tunnel Endpoint Discovery* service (covered in the following section).

Tunnel Endpoint Discovery

Tunnel Endpoint Discovery (TED) is a Cisco feature that improves the scalability and availability of IPsec VPNs by extending the capabilities of dynamic crypto maps. As mentioned in the preceding section, "Configuring Dynamic Crypto Maps," dynamic crypto maps greatly reduce your work by eliminating the configuration of specific IPsec peers. However, dynamic crypto maps (by default) are only *receivers* of IKE negotiation requests. That is, unlike regular crypto maps, they cannot initiate outbound SAs to remote peers—dynamic crypto maps rely on other peers to first contact them.

The listen-only behavior of dynamic crypto maps is acceptable in most *access* VPN environments where a central router accepts IKE requests from lots of dial-in users. However, in an *intranet* VPN environment, the listen-only approach isn't very useful because you want to give each peer the ability to initiate IKE with any other peer. This means you have to use regular, static crypto maps.

When you have a large intranet VPN—say over 20 or 30 routers—configuring static crypto maps to connect to all possible peers becomes laborious and difficult to manage. When it's time to add a new peer, you have to revisit all other peers and modify each configuration to accommodate the addition—that's a lot of work.

TED Improves IPsec Scalability

So here comes TED. With TED, a dynamic crypto map can probe for and discover remote peers dynamically. With TED, this means that a dynamic crypto map *can* be used to initiate SAs to multiple peers automatically, on demand, and with minimal configuration (the remote peers must also have TED enabled). Thus, many-to-many (mesh topology) peering is possible without a need to configure huge static crypto maps.

TED works by sending a probe packet and listening for a response to determine the other end of an IKE SA. When an outbound packet requires IPsec protection, TED dispatches a probe packet. The probe gets routed over the untrusted network toward the ultimate destination of the data until it reaches the remote, TED-enabled peer. When the remote peer responds to the originator of the probe, the remote peer is "discovered" and the two devices establish an IKE SA. The rest of the IPsec process then continues as usual.

TIP TED is handy when you're deploying backup peers for high-availability VPNs. If the remote peer goes down, TED will dynamically discover a new remote peer (assuming you deployed one as a backup). When you add a backup peer, you do not have to reconfigure the other routers in the network—the new backup is automatically discovered by TED.

Configuring TED

To enable TED, you simply configure dynamic crypto maps as described in the preceding section, "Configuring Dynamic Crypto Maps," and add the keyword **discover** to the **crypto map ipsec-isakmp dynamic** command. The following is an example:

```
crypto dynamic-map DYN-TED-MAP 20
 match address 101
 set transform-set TRANS-ESP TRANS-AH-ESP
!
crypto map MYMAP 500 ipsec-isakmp dynamic DYN-TED-MAP discover
!
interface Serial1
 ip address 192.168.1.1 255.255.255.0
 crypto map MYMAP
```

The preceding TED-enabled configuration is syntactically the same as the configuration for dynamic crypto maps except the keyword **discover** is appended to the crypto map entry.

NOTE IPsec in IOS releases before 12.0(5)T and 12.0(5)XE do not support TED.

Validating IPsec Configuration

The following enable mode commands are useful for validating the IPsec configuration:

- **show crypto isakmp policy** returns the router's active IKE transform sets (policies) in order of priority.
- **show crypto isakmp sa** displays the status of the router's IKE SAs. A state of **QM_IDLE** means the IKE SA is up and functioning properly. Recall that both IKE and IPsec SAs are built only when they are needed and are triggered by traffic that matches a crypto map.

- **show crypto map** displays the crypto maps configured in the router. Here you can find the details of a crypto map, including its elements (crypto access lists, transform sets, peers, PFS setting, and so on).
- **show crypto ipsec transform-set** displays the transform sets configured in the router.
- **show crypto ipsec sa** displays a detailed list of the router's active IPsec SAs. Here you can find information on each SA, including the lifetime remaining, transforms, mode (tunnel or transport), SPI, and packet counters. Clear the packet counters with **clear crypto sa counters**.
- **show crypto engine connections active** displays a list of active SAs with their associated interfaces, transforms, and counters.
- **show crypto dynamic-map** displays the dynamic crypto maps in the router.

Troubleshooting IPsec and IKE

To communicate successfully with IPsec, two devices must successfully negotiate many steps. The slightest misconfiguration of a protocol, key, lifetime, transform set, or other parameter will result in failed IKE or IPsec negotiations and SAs will not be established.

The next couple of sections offer some hints for isolating and troubleshooting IKE and IPsec.

Check Configurations and Show Commands

Examine the configuration of each peer (issue the **show running-config** command) and double-check the following:

- IKE SA transform sets (policies) agree on both peers including the lifetime parameter.
- IKE pre-shared keys, public keys, or certificates are correct.
- Crypto maps on both routers share a common IPsec transform set.
- Crypto access lists on both routers are mirror images of each other and are included in the crypto maps.
- Crypto maps are configured with the peer's correct and reachable IP address.
- Crypto maps are applied to the proper interfaces (the exiting interface that points to the remote peer).

Also, use the **show** commands from the section "Validating IPsec Configuration" earlier in this chapter and check if any IKE or IPsec SAs have been successfully established (**show crypto isakmp sa** and **show crypto ipsec sa**). If the routers are not establishing SAs when they should, investigate the situation in more detail with the debugging commands in the following section.

NOTE Before you configure any IPsec, it's a good idea to verify you have normal IP connectivity between peers—a regular ping or Telnet from one peer to the other does the trick. After you know the IP layer works, you can configure IPsec.

Enable Debugging and Clearing Existing SAs

To get more detailed information and observe IKE and IPsec negotiations, enable debugging with these commands:

```
RTA#debug crypto isakmp
RTA#debug crypto ipsec
```

With debugging enabled, the router displays the status of IKE and IPsec events in detail. See the following section "Messages for IKE Negotiation and CA Servers."

To debug CA events, issue these additional commands:

```
RTA#debug crypto pki messages
RTA#debug crypto pki transactions
```

To observe IKE negotiation, you might want to clear any existing IKE SAs with the command **clear crypto isakmp**. This allows you to observe IKE negotiation on the router from the beginning.

To clear existing IPsec SAs, issue the command **clear crypto sa**. This clears all IPsec SAs on the router and might be undesirable if there are active SAs transporting live traffic. Alternatively, you can clear existing IPsec SAs by crypto map name, peer, or SPI (issue the command **clear crypto sa ?** for help).

Messages for IKE Negotiation and CA Servers

The following messages indicate that IKE negotiation failed and an IKE SA cannot be established. This is caused by one or more mismatching IKE parameters (lifetime, hashing algorithm, encryption algorithm, authentication method, or Diffie-Hellman group). *Main mode* is the negotiation step that establishes the IKE SA.

```
%CRYPTO-6-IKMP_MODE_FAILURE: Processing of Informational mode failed with peer at
    192.168.1.1
ISAKMP (215): no offers accepted!
ISAKMP (212): SA not acceptable!
%CRYPTO-6-IKMP_MODE_FAILURE: Processing of Main mode failed with peer at 192.168.1.1
```

The following messages appear when a peer is configured with the wrong pre-shared key (IKE authentication with pre-shared keys):

```
ISAKMP: reserved not zero on payload 5!
%CRYPTO-4-IKMP_BAD_MESSAGE: IKE message from 192.168.1.2 failed its sanity check or is
    malformed
```

Any one of the following messages appears when a peer is configured with the wrong public key of the remote peer (IKE with RSA encryption and manually configured public keys):

```
%CRYPTO-6-IKMP_CRYPT_FAILURE: IKE (connection id 127) unable to encrypt (w/peers RSA
      public key) packet
%CRYPTO-6-IKMP_MODE_FAILURE: Processing of Main mode failed with peer at 192.168.1.1
%CRYPTO-4-IKMP_BAD_MESSAGE: IKE message from 192.168.1.1 failed its sanity check or is
      malformed
```

The following messages indicate a communication problem between the router and the CA server:

```
CRYPTO_PKI: socket connect error.
CRYPTO_PKI: 0, failed to open http connection
CRYPTO_PKI: 65535, failed to send out the pki message
```

Ensure that nothing on the network is preventing the router from communicating with the CA server (perhaps an access list).

The following messages indicate that the router submitted a certificate request to the CA server and is waiting for the CA to grant the certificate:

```
CRYPTO_PKI: status = 102: certificate request pending
CRYPTO_PKI: resend GetCertInitial
```

The following messages indicate that the CA granted the certificate and the router received it:

```
CRYPTO_PKI: status = 100: certificate is granted
%CRYPTO-6-CERTRET: Certificate received from Certificate Authority
```

The following messages indicate that IKE negotiation was successful. *Quick mode* is the negotiation step that establishes IPsec SAs.

```
ISAKMP (96): SA has been authenticated with 192.168.1.1
ISAKMP (96): beginning Quick Mode exchange
```

After the IKE SA Is in Place: Messages for IPsec Negotiation

The following messages appear when the two peers cannot negotiate a common IPsec transform set:

```
IPSEC(key_engine): request timer fired: count = n,
ISAKMP (269): SA not acceptable!
%CRYPTO-6-IKMP_MODE_FAILURE: Processing of Quick mode failed with peer at 192.168.1.2
```

Check the transform sets in the crypto maps of both peers and ensure that they propose the same protocols and algorithms (**esp-des** with **esp-md5-hmac**, for example).

The following messages indicate that the peers have successfully negotiated an IPsec transform set:

```
ISAKMP (288): Checking IPSec proposal n
ISAKMP: transform 1, <name>
<lines deleted for brevity>
ISAKMP (288): atts are acceptable.
```

The following messages appear when IPsec SAs are starting up:

```
ISAKMP (264): Creating IPSec SAs
IPSEC(initialize_sas):
```

The following message indicates that an IPsec SA was successfully created:

```
IPSEC(create_sa): sa created
```

The following messages indicate that the router received an IPsec packet but is unable to decrypt it because it does not match a crypto access list:

```
%CRYPTO-4-RECVD_PKT_INV_IDENTITY_ACL: ipsec check access: identity not allowed by
     ACL
00:31:27: IPSEC(epa_des_crypt): decrypted packet failed SA identity check
00:31:41: IPSEC(epa_des_crypt): decrypted packet failed SA identity check
```

This usually happens when the router and its remote peer are not configured with mirror-image access lists. See "Crypto Access Lists: An Example" earlier in this chapter for more information on mirror-image access lists.

The following messages appear when a router receives a non-IPsec packet and expects the packet to be protected with IPsec:

```
%CRYPTO-4-RECVD_PKT_NOT_IPSEC: Rec'd packet not an IPSEC packet.
        (ip) dest_addr= 172.16.2.137, src_addr= 192.168.10.4, prot= 6
```

The packet matches a crypto access list in the crypto map but is not an IPsec packet. This usually happens when the router and its peer are not configured with mirror-image access lists (or when the router receives a non-IPsec packet for some other reason). See "Crypto Access Lists: An Example" earlier in this chapter for more information on mirror-image access lists.

Summary

This chapter covered the basic principles of security and cryptography necessary for understanding and implementing VPNs with IPsec. The IPsec standard embodies a broad range of security protocols, algorithms, and practices that work together to provide secure communication over an untrusted network.

The following are the key concepts of this chapter:

- IPsec is a security architecture that uses many technologies to secure IP traffic across an untrusted network.
- The most popular use of IPsec is for building VPNs.
- Confidentiality is achieved by encrypting plaintext with an algorithm to create ciphertext.

- Shared key encryption requires trusted parties to agree on a common, confidential key. Some common algorithms are DES, Triple-DES, IDEA, and Blowfish. The Diffie-Hellman key exchange algorithm provides a way of establishing a shared key by exchanging non-secret information.
- With public key encryption (or public key cryptography), each party generates and owns a pair of keys: a private key that is kept secret and a public key (non-secret information) that is given to anyone who wishes to send confidential data to the owner of the public key. RSA is a common algorithm.
- Integrity provides a guarantee that data has not been altered in transit. Hashing algorithms such as MD5 and SHA-1 provide the mechanics for delivering integrity.
- Origin authentication provides an assurance that the person (or device) you are communicating with is who he or she claims to be. Digital signatures guarantee authenticity of messages using public key cryptography. Digital certificates (issued by CAs) bind a public key to an identity, thereby establishing trust and nonrepudiation.
- Anti-replay provides the assurance that past transactions cannot be replayed by an attacker. IKE enables this service.
- The main elements of IPsec are peers, transform sets, security associations, modes (transport and tunnel), and protocols (AH and ESP).
- The IKE protocol works hand-in-hand with IPsec and performs many important functions. IKE dynamically establishes IPsec SAs as they are needed, supports the exchange of digital certificates, and enables dynamic rekeying, anti-replay, and perfect forward secrecy.
- In the Cisco implementation, IKE can authenticate peers by using pre-shared keys, RSA encryption, and RSA signatures with digital certificates.
- A router does not initiate an IKE SA until it is needed to establish the IPsec SAs that protect user traffic.
- A crypto map can contain multiple entries. Each entry contains a crypto access list, a list of acceptable transform sets, the address of the remote peer (or multiple addresses if redundancy is desired), and other information (lifetime and PFS).
- Crypto access lists do not specify the traffic to filter on an interface, but rather the outbound traffic that is to be protected with IPsec.
- Crypto access lists also check inbound traffic to enforce consistent security policies.
- Mirror-image crypto access lists are crucial to the proper operation of IPsec. Without them, problems occur because the peers' access lists do not agree and do not protect the same set of traffic.
- Dynamic crypto maps simplify large peering configurations by providing templates of basic IPsec requirements.

- TED allows a dynamic crypto map to probe for and discover remote peers. With TED, a dynamic crypto map can be used to initiate SAs to multiple peers automatically, on demand, and with minimal configuration. This increases IPsec scalability in large intranet and extranet VPNs.
- Many steps have to happen before two peers can successfully communicate with IPsec. Typically, you will need to enable IOS debugging to observe and validate IKE/IPsec negotiations.