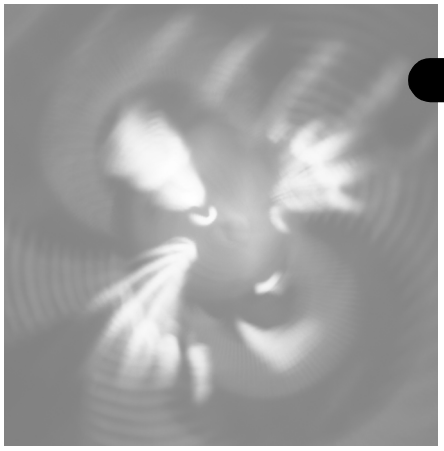

E I G H T

Introduction to Encryption Techniques

In This Chapter

- Symmetric Key Encryption
- Asymmetric Key Encryption
- Comparing Symmetric versus Asymmetric Methods
- The Diffie-Hellman Algorithm
- RSA Public-Key Encryption
- Message Authentication Codes
- Summary

The subject of encryption is fundamental to a complete network security strategy. With increased use of public facilities such as the Internet comes a need to guarantee message confidentiality and integrity for many applications. In the past, security was a matter of physically securing the data and limiting access to the data. Where data communications was involved from site to site, the data often traveled over point-to-point dedicated circuits, which afforded a reasonably degree of privacy and integrity since the actual path traveled by the data was fixed and determinable, and relatively secure from casual eavesdropping.

Encryption has primarily been used to prevent the disclosure of confidential information, but can also be used to provide authenticity of the source of the message, verify the integrity of received data, provide the digital equivalent of a handwritten signature, and nonrepudiation. Nonrepudiation assures that a transacting party cannot deny that the transaction took place.

Cryptography is the name for the study of procedures, algorithms, and methods to encode and decode information. *Cryptanalysis* is the study of methods and means to defeat

or compromise encryption techniques. *Cryptology* is the study of both of cryptology and cryptanalysis combined, and is derived from the Greek *kryptos logos*, which translates into "hidden word."

Encryption usually requires the use of a hidden transformation that requires a *secret key* to encrypt, as well as to reverse the process or decrypt. With some encryption methods, the same key is used to both encrypt and decrypt the information. This form of encryption is known as *symmetric* encryption, which is also known as *single-key* or *secret-key* encryption. Another form of encryption uses two keys: one key to encrypt and a different key to decrypt. These systems are referred to as *asymmetric* encryption, also referred to as *public-key* encryption, since one of the two keys is publicly known (and the other is kept secret).

Messages or data are referred to as *clear text* or *plain text* before encryption is applied, and *cipher text* to describe text or data that has been encrypted. When the key is used to reverse the process (i.e., transform cipher text back to the original clear text), the decoding process is known as *decryption*.

Symmetric Key Encryption

Symmetric or *secret-key* systems have been around for a very long time. Caesar used one of the earliest known single-key systems when communicating with his generals. The Caesar code used a key of three, that is, each letter in the alphabet was shifted by three positions.

a=d, b=e, c=f, d=g, and so on

With the Caesar code, the word c-a-t would be transformed into f-d-w. For letters at the end of the alphabet, the code would wrap around to the beginning of the alphabet:

s=v, t=w, u=x, v=y, w=z, x=a, y=b, z=c

The Caesar code is easy to "break" since the key is a single digit from 1 to 9. While this code may be confusing, with paper and pencil, a little trial and error and a lot of patience, this code is breakable by mere mortals. Puzzles using a similar "code" are often included as brainteasers in popular magazines.

We could make the process a little more difficult by making the key a two-digit number (e.g., 1 to 99) and successively cycling multiple times through the alphabet, making it approximately 10 times more difficult. A third digit would make it approximately 100 times more difficult, with each additional digit increasing the number of possible combinations by a factor of 10. While this would make it much more difficult to try all the combinations by hand, this is a trivial exercise for a modern computer. The method of attempt-

ing all possible combinations is referred to as the *brute force* method. If all possible combinations of the key are attempted, eventually a message that makes sense will appear. We need some method of increasing the complexity beyond what is possible to defeat by use of brute force methods and state-of-the-art computers.

These more complex code systems will require computers to encrypt and decrypt, so we will need a method of describing our code as a mathematical expression. Fortunately, the field of mathematics has such an expression using *modulus* arithmetic. The above Caesar code can be expressed in modulus arithmetic as:

x+3 modulo 26 (where x is the character in plain text, before transform)

The limitation with simple codes like the Caesar code is not only the key length, but also the algorithm it is based upon. There is a higher frequency of occurrence for certain letters in most languages. This fact suggests a shortcut might be found using statistical analysis of individual letter frequencies, so that it might not even be necessary to attempt all possible combinations. What is also need is a method to randomize the distribution, so that these patterns are disguised. Any encryption algorithm that we chose must randomize the data so that the encrypted result (i.e., *ciphertext*) appears random, so that statistical analysis of word and grammar patterns is not possible. When brute force is the only method available to perform *cryptanalysis* (i.e., break the code), the length of the key becomes the critical issue. Key lengths that were once assumed to result in "unbreakable" code may now be considered inadequate.

Data Encryption Standard

The Data Encryption Standard (DES) was formulated by the U.S. Government in the late 1970s, and uses a 56-bit (binary) key. The standard is defined in Federal Information Processing Standard (FIPS) 46.

The clear text is processed in blocks of 64 bits (therefore, DES is considered a block-cipher) and subject to various permutations involving the key. The algorithm used is an improvement of the Lucifer algorithm developed by IBM, with enhancements done by the National Security Agency (NSA) and National Institute of Standards and Technology (NIST). The NIST (formerly known as the National Bureau of Standards) is a division of the U.S. Commerce Department, and is the author of various FIPS publications. NIST issues the standards that are used by all nonmilitary federal agencies, and many private sector organizations choose to follow their lead, as well. NIST declared DES the official standard in early 1977, and it soon became the de facto standard in the private sector, as well.

As has been the practice of NIST, the DES standard is reassessed on a periodic basis. In 1994, NIST recertified DES for an additional five years. The

DES standard has been considered adequate until fairly recently when a collaborative effort, involving numerous machines, proved that brute force attacks can be successful against 56-bit DES, if the attacker can harness sufficient computing power (well within the affordable reach of any sufficiently motivated organization such as a national government).

The "breaking of DES" came as a result of a challenge issued in January of 1997 by RSA® Laboratories that offered a $10,000 reward. The challenge was to discover the 56-bit DES key used to encrypt a message. An independent consultant, Rocke Verser, developed a program that used the brute force method, and made the program available on the Internet, along with a request for volunteers to join in a collaborative effort to discover the secret key. The effort received wide support involving over 70,000 individual machines, with each volunteer assigned a portion of the 2^{56} number range. Over a period of slightly more than three months, the group had searched approximately 25 percent of the possible combinations when they "hit pay dirt." Although this took a relatively long time, with a large number of machines, it did reaffirm the use of modest computing platforms operating in a distributed computing environment to solve large-scale problems.

Bear in mind that this was not a trivial exercise, since even when one combination results in a clear text message from the cipher text, someone or some process must recognize that it is in fact readable clear text, which assumes that 1) the language of the clear text is known and/or 2) the clear text had not been compressed before being encrypted, thereby obscuring the clear text message.

Since DES had been proven to be inadequate for at least some application environments, the NIST began a search for a new standard algorithm to replace the aging DES. The replacement has been named the *Advanced Encryption Standard* (AES), and has been in the works for nearly four years.

Advanced Encryption Standard and Others

The AES standard defined multiple key lengths of 128, 192, and 256 bits, in order to provide some longevity to the standard as computing technologies constantly improve. The NIST criteria had several objectives, including flexibility, ability to implement in hardware and software, and various other factors. They invited proposals and went through a series of evaluations of the various proposals before deciding upon one.

In late 2000, the NIST tentatively selected the Rijndael (pronounced like rine doll) algorithm as the basis for the new standard that should be ratified sometime in 2001. Until the AES standard is finalized, DES continues to be adequate for many applications. Where there is concern regarding the strength of DES, triple-DES (3DES) is a more than adequate substitute, in that 3DES is significantly stronger. (Cisco currently supports both DES and 3DES and will likely support AES once the standard has been ratified).

Although "triple" sounds like it is three times stronger, it is many orders of magnitude stronger. With 3DES, data is encrypted three times, with a 56-bit key applied each time, for an effective key length of 168 bits. The total number of combinations with 56-bit DES is 2^{56}, while 3DES increases the total number of combinations to 2^{168}. This indicates that 3DES is 2^{112} times (168-56=112) stronger than DES, and is considered "unbreakable."

A footnote is in order when we use the term "unbreakable." Mathematicians consider an encryption algorithm unbreakable when there is "no known method other than brute force," and the brute force method is not feasible with known computer technologies. There is always a caveat that unpublished methods that are unknown in the academic community, could be theoretically possible. (Note: The U.S. National Security Agency [NSA] may be the world's foremost authority on encryption, but obviously do not publish all of their research.)

Bear in mind that this does not mean that 3DES is always warranted. The computing power necessary to perform encryption and decryption is always significant. With 3DES, the computational power required is approximately three times that required by DES. For many applications it is not always necessary that encryption be unbreakable, only that it be infeasible. Infeasibility implies that the time and/or effort to apply brute force to break the code exceeds the timeliness or value of the information. Even if it only took one day (24 hours) to brute force a 56-bit key, if the key were changed once per hour, the decoding process would always be behind, since the 24 keys used during any one day would take 24 days to break. If the value of intercepting the data decreases over time, the information may be "secure enough." Common symmetric encryption methods and their key lengths are:

- DES — 56 bit
- IDEA — 128 bit; the International Data Encryption algorithm, developed by Lai and Massey of the Swiss Federal Institute of Technology. This encryption algorithm is included in PGP (Pretty Good Privacy), and has therefore been widely distributed.
- 3DES — 168 bit
- AES — 128/192/256 bit

In Chapter 7 of *Applied Cryptography* by Bruce Schneier (2^{nd} edition, John Wiley & Sons, Inc., 1996), the author provides a thorough discussion of the computational effort required to perform a brute force attack on various key lengths, along with estimates of equipment cost. The author also provides a very comprehensive view of various other encryption methods and algorithms that are beyond the scope of this book.

DES and the other encryption methods listed above are described as *block-ciphers*, meaning that the clear text data (transformed into binary format) is encrypted in blocks of a given length (commonly 64 bit blocks) and produces a cipher text message of approximately the same length. Another

excellent text that describes the detailed internals and mathematical foundation of various encryption methods is *Cryptography and Network Security* by William Stallings (2nd edition, Prentice Hall, 1999). The interested reader is referred to either or both of these texts for a very thorough, detailed discussion of encryption techniques, algorithms, and methods.

Key Management

One of the drawbacks with symmetric encryption is key distribution and management. For each pair of encrypting devices, a separate key should be used. If a given device has many encryption partners, distribution of the keys becomes logistically inconvenient. For each pair of devices, a key must be generated then transported via some secure means to each device. Even if one device generates the key, the problem remains of delivering or communicating the key to the companion device via special courier or other secure transport method. When large numbers of devices are involved, this burden becomes unwieldy very quickly. Consider the case where it might be desirable to encrypt email to multiple parties or casual parties on a one-time basis, but key distribution must be performed first.

The Kerberos protocol partially solves this problem by using the concept of a key distribution center (KDC), but Kerberos is more appropriately used for a distributed computing environment that has a central management. Kerberos is also notorious for the administrative effort required to manage and maintain the environment. A complete discussion of Kerberos is beyond the scope of this material, and will not be presented here.

Fortunately, there is another solution to the key distribution dilemma in the form of asymmetric encryption, as will be explained below.

Asymmetric Key Encryption

As described above, asymmetric encryption involves a two-key set. Data encrypted with one key can only be decrypted with the other in the set. This type of encryption is often described as *public-key* encryption, to emphasize that one of the keys is made publicly available. Even though the other remains secret, this fact should not be confused with the single secret key that is used in symmetric encryption, which we have also referred to as *secret-key* encryption, since the single key is always kept secret. When describing the undisclosed, nonpublic key used with public-key systems, we will refer to it as the *private* key to differentiate from the *secret* key associated with symmetric encryption.

The asymmetric method of encryption has only been around for a short time, when compared to classical symmetric encryption that predates the Romans. Public-key encryption was “discovered” by mathematicians Whitfield

Diffie and Martin Hellman of Stanford University, and described in a paper they published in 1976. It evolved as a direct result of work done in search of a better method to manage key distribution for classic secret key environments. The result of that research is covered in more detail below.

The primary difference between asymmetric (public key) and symmetric (secret key) encryption is that public-key encryption is based upon mathematical functions. By contrast symmetric encryption relies upon clever rearrangement of the bits through successive substitutions and permutations. In layman's terms — the bits are "scrambled." The fact that two keys, one public and one private, are used allows public-key encryption to be applied in creative ways to provide additional functionality beyond the fundamental confidentiality associated with encryption.

Various forms of *asymmetric* or *public-key* encryption exist that serve different purposes, including the Diffie-Hellman method, named after the creators, and the RSA method, named after mathematicians Ron Rivest, Adir Shamir and Leonard Adelman of MIT.

The two keys are used in different ways to achieve different purposes. Used in one fashion, they can provide confidentiality, and in another can provide authentication of the sender. A by-product of public key systems can also be nonrepudiation, by proving that one party *did* send a communication even though they may claim that they did not.

In a public-key system, each individual party generates two related keys: a *private key* and a *public key*. They keep the private key secret, but publish or otherwise make the public key readily available to anyone who needs it. This eliminates one of the shortcomings of symmetric key systems, namely that of secure key distribution. The individual public keys can be obtained by anyone who has a need for them from a conveniently located (public) server, eliminating the necessity for a secure means to deliver the keys, and without compromising the private key in any way. Although the public and private keys are mathematically related, it is not possible to determine one key of the set from knowledge of the other key.

The administrative burden to establish and maintain public key distribution is minimal when compared to the comparable process for symmetric key methods, especially the KDCs used by Kerberos. Further, the function can be distributed across multiple servers, if desired. There is also no requirement to send the public key any individual encryption partner; instead, the partner can the key from a server on an as-needed basis.

How Public-Key Encryption Works

Public-Key encryption proves a number of capabilities, depending upon how it is applied. The major features that are most commonly sought are discussed in the sections that follow.

CONFIDENTIALITY

The relationship between the two keys (private and public) is that whatever is encrypted by one key can only be decrypted by the other key of the set. To illustrate their use to provide confidentiality or authenticity or both, consider the following case.

Alice and Bob each generate a set of keys. Alice generates keys A_{Public} and $A_{Private}$, while Bob generates B_{Public} and $B_{Private}$. Each party keeps their private key, and they exchange their public keys. The only requirement for exchange of public keys is that each party be assured that the public key they receive is, in fact, the true public key of the other party (more on this aspect later).

The end result is that Alice now possesses $A_{private}$ and B_{public}, while Bob has $B_{private}$ and A_{public}. If Alice wants to send a confidential message to Bob, she will use Bob's public key, B_{public}, to encrypt the message and send it to Bob. Since only Bob has the companion private key, $B_{private}$, only he will be able to successfully decrypt the message. This solves the problem of confidentiality between these two parties, without requiring that they share a common key, and is illustrated in Figure 8-1.

However, if we add a third-party, Carol, things get a little more complicated. Carol generates her own set of keys, $C_{private}$ and C_{public}, and each of the three parties receives the appropriate public keys from the other parties. Each party will now have the following keys:

- Alice $A_{private}$, B_{public}, C_{public}
- Bob $B_{private}$, A_{public}, C_{public}
- Carol $C_{private}$, A_{public}, B_{public}

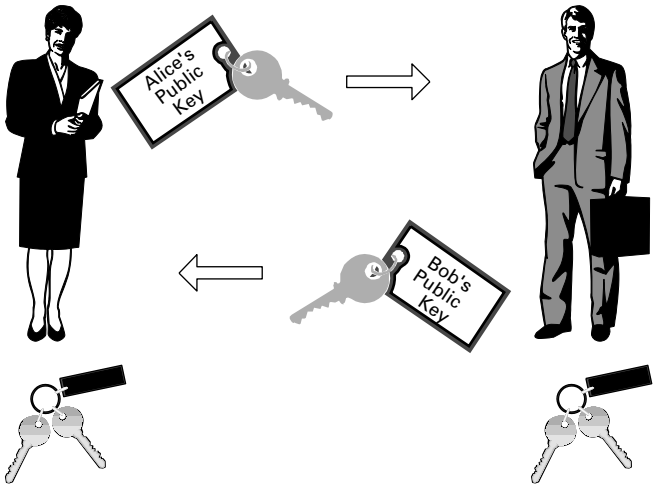

FIGURE 8–1 Public key exchange.

Using the same scenario described earlier, Alice sends a confidential message to Bob encrypted with Bob's public key, B_{public}. Only Bob will be able to decrypt the message, since he has the complementary private key, $B_{private}$. Carol will still be unable to decrypt the message since she has the same key (B_{public}) that was used to encrypt, and only the companion key ($B_{private}$) can be used to decrypt the message.

Bob now has a confidential message that he is able to decrypt, claiming to be from Alice; *however,* Carol could also have created the message, pretending to be Alice, since she also has Bob's public key, B_{public}. Although we have solved the confidentiality problem, we now have an authenticity problem, since anyone with Bob's public key could have generated the message.

Before we solve this particular problem, let's see how public-key systems might be used alternatively to provide authenticity, alone.

AUTHENTICITY

If the communication from Alice to Bob did not require confidentiality, but did require that Bob be assured the message genuinely came from Alice, we can alter the scenario slightly. Alice would now encrypt the message with her *own private* key, $A_{private}$. Anyone who possesses Alice's public key (including Bob) can decrypt and read the message, but Bob (and any others) would know that *only* Alice could have sent the message, since only Alice has the appropriate companion private key. We have now solved the authenticity problem, but we still have not solved the confidentiality problem, since anyone with Alice's public key can decrypt the message she sent encrypted with her private key.

COMBINING CONFIDENTIALITY AND AUTHENTICITY

The challenge is to simultaneously provide confidentiality and authenticity. A little bit of creativity, patience, and logic provides us with an answer.

For Alice to send a confidential message to Bob and provide assurance that the message really came from Alice, we apply a double encryption.

Alice encrypts the message by first using her private key, then encrypts the result using Bob's public key. No matter who receives this double encrypted message, only Bob can decrypt the contents.

To make this procedure easier to understand, think of an envelope inside of an envelope. The inner envelope is the message encrypted with Alice's private key. This inner envelope is then encrypted using Bob's public key, B_{public}, creating the outer envelope.

Only Bob can "open" the outer envelope since only he has the companion key, $B_{private}$, Bob's private key. Any others who may intercept the message will be unable to open the outer envelope, since they could only have Bob's public key, B_{public}, which was used to encrypt the message, and, therefore, cannot also be used to decrypt the same message. As stated earlier, any-

thing encrypted with one key can *only* be decrypted with the other from that public/private "set." Public-key methods never allow the same key to be used for both encryption and decryption.

Bob now has received a confidential message, but the contents and authenticity are not yet determined. If Bob now applies the second stage of decryption to the "inner" envelope, using Alice's public key, he can read the message. If he can successfully decrypt the inner envelope using Alice's public key, he also knows that only Alice could have generated the message with her own private key, providing the authentication of the sender.

In addition, Alice cannot deny sending the message since only Alice could have encrypted the message with her own secret key, $A_{private}$, which also provides nonrepudiation. This procedure could be generalized by rephrasing the steps:

- First, encrypt the message with your *own private* key, providing the authentication and nonrepudiation since only you possess your secret key.
- Second, encrypt the message using the *public key of the recipient* since only the recipient will be able to decrypt the received message, providing confidentiality.
- Send the message.
- The recipient (only) can decrypt the received message with his or her *own private key*, maintaining confidentiality.
- The recipient can then decrypt the contents using the *sender's public key*, verifying the origin of the message.
- If either party doesn't have the other's public key, it must be obtained before the procedure can be completed.

HASHES AND MESSAGE DIGESTS

Although the procedure outlined above would work in theory, in practice a modified approach is used. As described, the procedure consists of two encryptions and two decryptions of the entire message. Since encryption, especially using asymmetric methods, is a very compute-intensive process to begin with, it would be desirable to reduce the volume of data that needs to be encrypted. Compression of the data before encryption would be one means, although it not often used. A simpler process could be achieved through the use of an algorithm known as a *one-way hash* or *message digest*.

A message digest is a short, unique representation of your message; a fingerprint. A message digest (sometimes also referred to as a one-way hash or cryptographic checksum) does not, by itself, require an encryption key. Instead, it is a computation or formula performed on the message, treating the message as a long string of bits representing a very large binary number. The resulting output is a *digest* of a predictable fixed length. The important char-

acteristic of a one-way hash or message digest function is that each message produces its own unique digest.

The digest may be transmitted along with the message as an integrity check or error detection mechanism. However, if the message is intercepted and intentionally altered, a new digest could be computed and delivered with the message. To provide the authentication function and integrity check on the message, the *digest* may be encrypted using the sender's private key. This digest is calculated before any encryption is applied to the message.

Rather than encrypting the entire message twice, only the message digest needs to be encrypted to provide authentication of the sender, reducing the computational requirements. This encrypted message digest is described as a *digital signature*.

(Note: A similar process will be introduced in a later section, in which *symmetric* key encryption will be applied to the message digest. When symmetric key encryption is applied to a message digest, the result is referred to as a *message authentication code* (MAC), rather than a *digital signature*.)

If we reexamine the procedure that we referred to earlier as "double encryption," we see that the computational requirements can be cut nearly in half, since we only encrypt the message once with the receiver's public key (for confidentiality), and encrypt the much shorter digest (typically 128 or 160 bits; 16 or 20 bytes) with the sender's private key. The improved procedure operates as follows:

- The sender calculates the message digest and encrypts the digest using the sender's own private key (insuring authenticity of the sender), thereby creating a *digital signature*.
- The sender encrypts the message (once) using the receiver's public key, insuring confidentiality, since only the receiver possesses the receiver's private key.
- The encrypted message is transmitted along with the encrypted message digest (i.e., digital signature).
- Upon receipt, the receiver decrypts the message using the receiver's private key, which only the receiver possesses.
- The receiver calculates the digest on the decrypted message.
- The receiver decrypts the digital signature using the sender's public key. The decrypted message digest should match the calculated message digest from the decrypted message.
- If the message digest received (from the decrypted digital sign) matches the message digest calculated from the decrypted message, authentication is verified. The receiver then takes whatever action on the message as is appropriate (commonly passing the decrypted message on to the application or process indicated as the destination).
- If the calculated digest and received digest do not match, discard the packet.

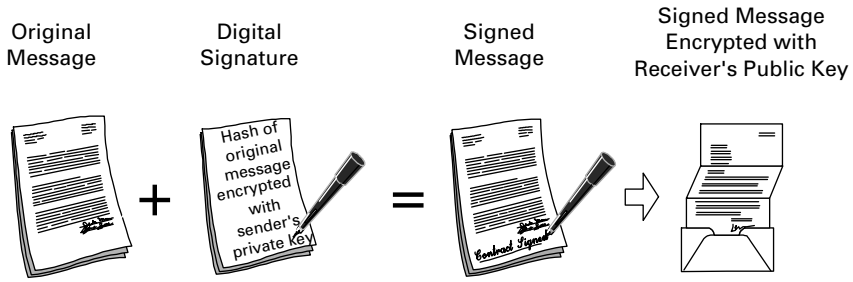

FIGURE 8–2 Digital signatures.

There are several indirect benefits of this approach. We have seen that with use of a message digest, the effort to encrypt and decrypt is reduced. The bandwidth required to transmit the message is also reduced, but also the storage requirements of the receiver are reduced, since the digest can be stored along with the message.

Remember that *without* the message digest, authentication was dependant upon the "inner" level of encryption, of the double encryption (i.e., the first encryption done by the sender and the last decryption done by the receiver). If it were necessary to store the "authentication" along with the message, the receiver would have to save a copy of the "inner" encrypted message, before performing the second decryption. Once the second decryption was performed (using the sender's public key), the authentication vanishes with the decryption. The act of decryption provides the authenticity, and once performed there is no storable evidence of the authentication. The only way to "store" the authentication would be to keep a copy of the message before the final decryption is performed, effectively doubling the storage requirements. The two copies refer to the copy that had been decrypted twice, and a second copy that had been decrypted only once using the receiver's private key. With this approach, the second copy would also require an additional decryption each time the message needed to be reauthenticated. Fortunately, use of the digital signature eliminates this duplication.

Although not specifically stated, the private/public key pair used for digital signatures doesn't have to be the same pair that is used for confidentiality. Any user can have multiple sets, as long as they can identify the appropriate set to potential receivers. In fact, it may be advantageous to have separate pairs so that replacing one would not require the other to change, as well.

Although we now have an improved process to provide authentication, in addition to confidentiality, it is based on the assumption that the public keys we use actually belong to the parties that are claimed. Public-key systems are susceptible to forgeries and/or imposters. We need some mechanism to guarantee that the public keys we receive and use truly belong to the parties they claim to be, rather than a clever imposter.

PUBLIC KEY INFRASTRUCTURE (PKI) AND DIGITAL CERTIFICATES

When you meet a stranger for the first time, they may introduce themselves, but may give a false name or claim to be someone else. A more reliable means would be for a trusted friend to introduce them, and vouch for their identity. A similar process might be used to exchange public keys through a trusted intermediary. The trusted intermediary would require that each individual party adequately identify themselves, and once the intermediary is satisfied that each party is who they claim to be, the intermediary can verify their identity.

If the intermediary is trustworthy and their verification process is thorough, it can serve as a reliable source to verify the identity of any party to any other party. Each party could provide "certificates of identity," issued by and signed by the intermediary vouching for the identity of that party.

In public-key encryption, a Certificate Authority (CA) fills this role of trusted intermediary, vouching for the identity of all participating members. Any individual can "enroll" with the CA by providing satisfactory proof of identity, along with the individual's public key. The CA issues a *certificate* that includes the following:

- Identity of the "entity" that is the owner of the public key; name, serial number, IP address, company and/or department name; or other criteria identifying an individual user or network device, such as a host, router, firewall, et al.
- The public key of this entity, which can be an individual user or a network device, such as a host or router
- Digital signature and identity of the CA issuing the certificate
- Dates of validity for this certificate
- Class of certificate — multiple levels; higher-class levels require enrollee provide more substantial proof of identity
- Digital certificate ID# for this certificate

The current standard for certificates is X.509 v3, as defined by the International Telecommunications Union (ITU). Notice that the identity of the issuing CA must be contained in the certificate. Since the CA digitally signs the certificate, the CA's public key must be used to validate the digital signature (which was signed using the CA's private key), and it may be necessary to retrieve the CA's public key before using the entity's public key that the certificate includes and guarantees.

There are a number of companies that develop CA software, including VersiSign®, Entrust® Technologies, Baltimore™ Technologies, and several others. These companies typically will provide the CA as a for-fee service, or will sell the software allowing an organization to manage its own CA. At this time, cross-certification among different CAs is incomplete, although efforts to provide this capability have been proceeding for several years.

The term *Public Key Infrastructure* (PKI) was intended to describe this *ultimate* goal, of complete cross-certification; however, the term is most often used when referring to the partial functionality that is currently available. Current references to PKI should usually be understood to mean that the "infrastructure" is only available within an individual organization that manages their own CA and builds their own "infrastructure," or between organizations that subscribe to a common CA service provider. There is no universality to the infrastructure at this time. Additional information regarding CAs will be included in the IPSec chapter.

Comparing Symmetric versus Asymmetric Methods

As you may have suspected by now, there is a downside to public-key systems. From the above discussion it might appear that asymmetric, public-key systems are superior to symmetric, secret-key systems. If that were that case, there would be no need for symmetric, secret-key systems at all, but things are rarely that simple.

One shortcoming of asymmetric, public-key methods is that they have an entirely different mathematical basis, and require key lengths that are significantly longer that a comparably strong symmetric, secret-key method. Although the ratio varies by key length, asymmetric keys must be roughly 7 to 18 times longer than an equivalent symmetric key to result in the same resistance to brute force attacks. As mentioned earlier in this chapter, Chapter 7 of *Applied Cryptography* by Bruce Schneier (2nd edition, John Wiley & Sons, Inc., 1996) provides several charts comparing key lengths between symmetric and asymmetric encryption methods. In addition to being the author of several books, Mr. Schneier is also the founder of Counterpane™ Labs (formerly Counterpane Systems). The Counterpane Web site contains a wealth of information related to encryption technologies, as well as some excellent downloadable freeware, and can be found at *http://www.counterpane.com/labs.html*.

Due to the increased key lengths required and other factors, asymmetric methods make require as much as 1,000 times more processing power and are, therefore, considered too computationally intensive for general-purpose encryption. Instead, public-key methods are used primarily to distribute or "negotiate" symmetric secret-keys, or other "low-volume" encryption tasks, with symmetric methods used for most bulk, high-volume encryption tasks.

Public-key encryption may be suitable for tasks such as user encryption or decryption of individual email messages, since the time required to encrypt the message can be treated as part of the overall message composition or reading time. Similarly, the CPU utilization is a less significant issue,

since the workstation may otherwise be idle while encryption or decryption is performed.

Conversely, public-key encryption would add significantly to the delay and load, if performed by a router or firewall acting as an encryption and decryption gateway for transit traffic.

The Diffie-Hellman Algorithm

The Diffie-Hellman algorithm is the oldest of several variants of public-key algorithms, and is based upon the original work done by Whitfield Diffie and Martin Hellman in the late 1970s. It is used today primarily for what is described as "key exchange," but could more accurately be described as "key creation." It is a special use of asymmetric, public-key techniques to negotiate and generate a symmetric, secret-key for use by a symmetric encryption method, such as DES or 3DES. It cannot be used to encrypt and decrypt messages.

The Diffie-Hellman algorithm provides a mechanism for two parties to dynamically establish a single, shared secret key, known only to them, although they are communicating over an unsecured channel. Neither party creates the key; instead, each makes a contribution to the process, sometimes described as a "half-key." The resulting secret key may be used for DES, 3DES or other symmetric encryption method. The strength of the Diffie-Hellman algorithm lies in the difficulty of computing discrete logarithms in a finite field.

For those readers who are allergic to math, ignore the stuff that looks like math in the section below then continue reading where the text starts again.

For those readers who have an appetite for math, the procedure is described below.

Our old friends, Alice and Bob, want to use the Diffie-Hellman algorithm to derive a secret key known only to them, but need to communicate over an unsecured channel to do so. They begin by agreeing on two numbers: **p** (a very large prime number) and **g** (a number that is less than, and related to, **p**). These two numbers do not have to be kept secret.

Next, each party chooses a random integer value that is less than **p**, which we will call X_A (Alice) and X_B (Bob). Each party keeps this number secret but uses it in an equation:

Alice's Equation: $Y_A = g^{X_A} \bmod p$

Bob's Equation: $Y_B = g^{X_B} \bmod p$

Alice computes Y_A and sends the result to Bob.

Bob computes Y_B and sends the result to Alice.

Alice and Bob use their chosen secret values X_A and X_B in a new equation that includes the values Y_A and Y_B that each received from the other.

Alice: $K= (Y_B{}^{X}{}_{A}) \bmod p$

Bob: $K= (Y_A{}^{X}{}_{B}) \bmod p$

They now both have the computed value for K, which is the shared secret key. Any other party that may know **p** or **g**, or who intercepts the values Y_A and/or Y_B that are exchanged, will still be unable to determine K, the secret key, due to the difficulty in computing discrete logarithms.

Diffie-Hellman functions because exponentiation does not depend upon on the order that you do the exponentiation. For example: $2^3x2^2= 32 = 2^2x2^3$

It should be noted that the Diffie-Hellman method does not, by itself, provide authentication of the parties. However, Alice and Bob could use public-key encryption with digital signatures when exchanging the values of Y_A and Y_B, thereby providing authentication.

Perfect Forward Secrecy

The Diffie-Hellman method is particularly useful in generating temporary session keys that are used for the duration of one exchange "session" between two parties, then discarded. A new session requires the negotiation of a new session key. If each new session key is independent of previous session keys, Perfect Forward Secrecy (PFS) can be achieved.

PFS is considered very beneficial since the compromise of any given key will not allow any advantage in determining additional keys. In the absence of PFS, the compromise of one key can provide insight into the values of other keys, making them more vulnerable to attack. This topic will reappear in a later chapter on configuration of IPSec, since Cisco provides a configuration *option* for PFS or no PFS.

RSA Public-Key Encryption

The other significant implementation of public-key encryption is RSA, which derives its name from the developers of the method; Rivest, Shamir, and Adelman. The basis for the strength of this algorithm comes from the fact that it is very easy to multiply two large prime numbers together, but it is almost impossible to determine what those numbers were by factoring the resultant product of their multiplication.

The RSA algorithm was first published in 1978, and since that time has been the most commonly used example of public-key encryption. The description of public-key systems in the earlier part of this chapter are based primarily upon the RSA implementation, and it is by far the most widely

deployed of the public-key encryption methods. Over the past 20 years, a number of other public-key methods have been proposed, yet none have received the acceptance enjoyed by RSA.

Message Authentication Codes

A message authentication code (MAC) is another variation on a one-way hash or message digest that has been described previously. It is a procedure is similar to the procedure to create a digital signature, but the purpose is different. With a digital signature, the message digest was encrypted with the sender's private key to provide authentication and integrity check on the message sent. The primary difference is that the hash is encrypted with a secret key to accomplish a similar purpose. MACs will be revisited during the discussion of IPSec, since they are a required element in the configuration of IPSec on Cisco routers and firewalls.

Summary

In this chapter, a broad overview of encryption technologies was presented, focusing upon those elements that are present and/or supported on the Cisco router and firewall product lines. This chapter was intended as a primer for those topics that will be the subject of further discussion in the next two chapters. The goal of this chapter was to provide a foundation for the discussion of IPSec, an adjunct to the IP network layer protocol.

We discussed the secret-key encryption methods, DES and 3DES, that are currently supported by IPSec, and the public-key methods, the Diffie-Hellman and RSA algorithms, that provide the foundation for the key exchange mechanism found in IPSec.

We made no attempt to cover a number of networking and encryption related topics, such as SSL, S/HTTP, S/MIME, PGP, or other application-layer security protocols, since the emphasis of this book is security applied at the Network Layer.

Likewise, this material is not intended to provide as thorough a discussion of encryption-related topics as may be found in the texts referenced earlier in this chapter, *Cryptography and Network Security* by William Stallings, and *Applied Cryptography* by Bruce Schneier. For the reader who is interested in more detail, these two books are excellent sources.