CHAPTER 31

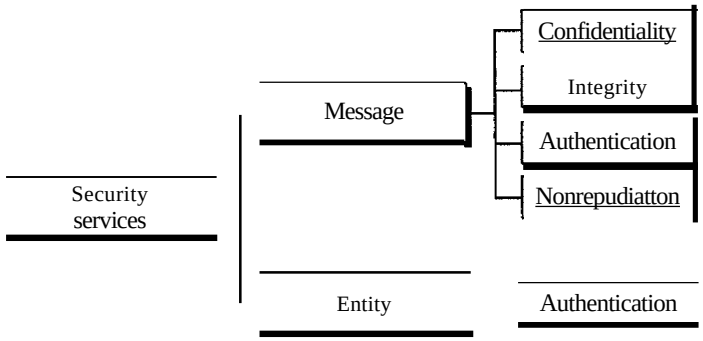
Network Security

In Chapter 30, we introduced the science of cryptography. Cryptography has several applications in network security. **In** this chapter, we first introduce the security services we typically expect in a network. We then show how these services can be provided using cryptography. At the end of the chapter, we also touch on the issue of distributing symmetric and asymmetric keys. The chapter provides the background necessary for Chapter 32, where we discuss security in the Internet.

31.1 SECURITY SERVICES

Network security can provide one of the five services as shown in Figure 31.1. Four of these services are related to the message exchanged using the network: message confidentiality, integrity, authentication, and nonrepudiation. The fifth service provides entity authentication or identification.

Figure 31.1 *Security services related to the message or entity*



Message Confidentiality

Message confidentiality or privacy means that the sender and the receiver expect confidentiality. The transmitted message must make sense to only the intended receiver. To all others, the message must be garbage. When a customer communicates with her bank, she expects that the communication is totally confidential.

Message Integrity

Message integrity means that the data must arrive at the receiver exactly as they were sent. There must be no changes during the transmission, neither accidentally nor maliciously. As more and more monetary exchanges occur over the Internet, integrity is crucial. For example, it would be disastrous if a request for transferring $100 changed to a request for $10,000 or $100,000. The integrity of the message must be preserved in a secure communication.

Message Authentication

Message authentication is a service beyond message integrity. In message authentication the receiver needs to be sure of the sender's identity and that an imposter has not sent the message.

Message Nonrepudiation

Message nonrepudiation means that a sender must not be able to deny sending a message that he or she, in fact, did send. The burden of proof falls on the receiver. For example, when a customer sends a message to transfer money from one account to another, the bank must have proof that the customer actually requested this transaction.

Entity Authentication

In entity authentication (or user identification) the entity or user is verified prior to access to the system resources (files, for example). For example, a student who needs to access her university resources needs to be authenticated during the logging process. This is to protect the interests of the university and the student.

31.2 MESSAGE CONFIDENTIALITY

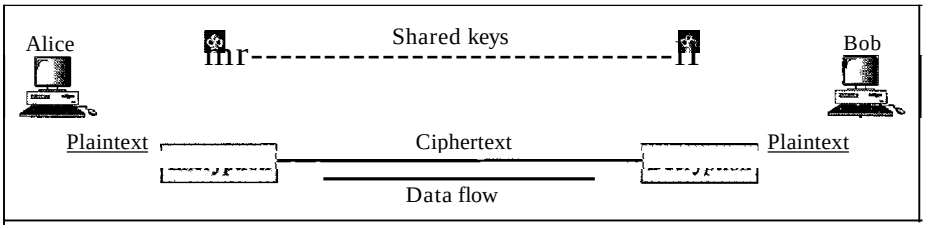
The concept of how to achieve message confidentiality or privacy has not changed for thousands of years. The message must be encrypted at the sender site and decrypted at the receiver site. That is, the message must be rendered unintelligible to unauthorized parties. A good privacy technique guarantees to some extent that a potential intruder (eavesdropper) cannot understand the contents of the message. As we discussed in Chapter 30, this can be done using either symmetric-key cryptography or asymmetric-key cryptography. We review both.

Confidentiality with Symmetric-Key Cryptography

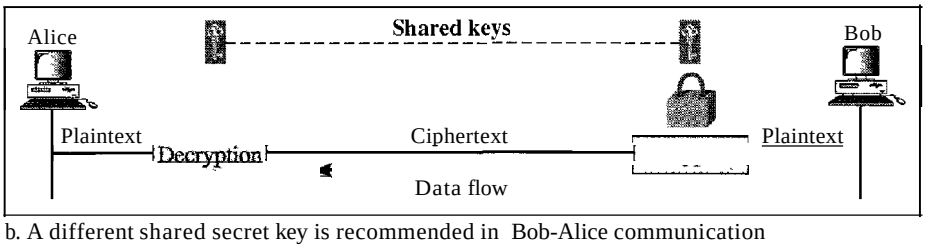
Although modern symmetric-key algorithms are more complex than the ones used through the long history of the secret writing, the principle is the same. To provide confidentiality with symmetric-key cryptography, a sender and a receiver need to share a secret key. In the past when data exchange was between two specific persons (for example, two friends or a ruler and her army chief), it was possible to personally exchange the secret keys. Today's communication does not often provide this opportunity. A person residing in the United States cannot meet and exchange a secret key with a person living in China. Furthermore, the communication is between millions of people, not just a few.

To be able to use symmetric-key cryptography, we need to find a solution to the key sharing. This can be done using a session key. A session key is one that is used only for the duration of one session. The session key itself is exchanged using asymmetric-key cryptography as we will see later. Figure 31.2 shows the use of a session symmetric key for sending confidential messages from Alice to Bob and vice versa. Note that the nature of the symmetric key allows the communication to be carried on in both directions although it is not recommended today. Using two different keys is more secure, because if one key is compromised, the communication is still confidential in the other direction.

Figure 31.2 *Message confidentiality using symmetric keys in two directions*



a. A shared secret key can be used in Alice-Bob communication



b. A different shared secret key is recommended in Bob-Alice communication

The reason symmetric-key cryptography is still the dominant method for confidentiality of the message is its efficiency. For a long message, symmetric-key cryptography is much more efficient than asymmetric-key cryptography.

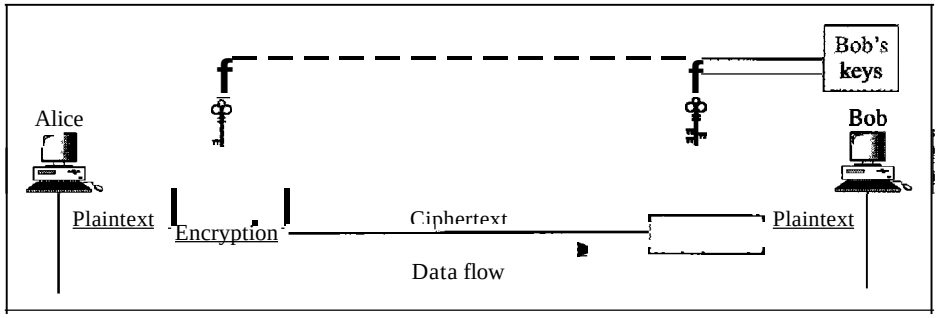
Confidentiality with Asymmetric-Key Cryptography

The problem we mentioned about key exchange in symmetric-key cryptography for privacy culminated in the creation of asymmetric-key cryptography. Here, there is no key sharing; there is a public announcement. Bob creates two keys: one private and one

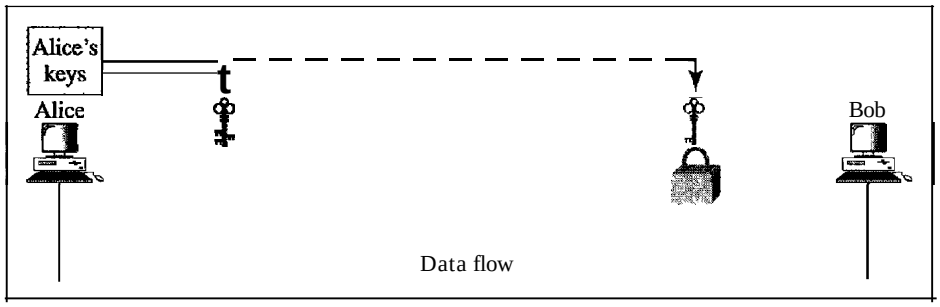
public. He keeps the private key for decryption; he publicly announces the public key to the world. The public key is used only for encryption; the private key is used only for decryption. The public key locks the message; the private key unlocks it.

For a two-way communication between Alice and Bob, two pairs of keys are needed. When Alice sends a message to Bob, she uses Bob's pair; when Bob sends a message to Alice, he uses Alice's pair as shown in Figure 31.3.

Figure 31.3 *Message confidentiality using asymmetric keys*



a. Bob's keys are used in Alice-Bob communication



b. Alice's keys are used in Bob·Alice communication

Confidentiality with asymmetric-key cryptosystem has its own problems. First, the method is based on long mathematical calculations using long keys. This means that this system is very inefficient for long messages; it should be applied only to short messages. Second, the sender of the message still needs to be certain about the public key of the receiver. For example, in Alice-Bob communication, Alice needs to be sure that Bob's public key is genuine; Eve may have announced her public key in the name of Bob. A system of trust is needed, as we will see later in the chapter.

31.3 MESSAGE INTEGRITY

Encryption and decryption provide secrecy, or confidentiality, but not **integrity.** However, on occasion we may not even need secrecy, but instead must have integrity. For example, Alice may write a will to distribute her estate upon her death. The will does not need to be encrypted. After her death, anyone can examine the will. The integrity of the will, however, needs to be preserved. Alice does not want the contents of the will to

be changed. As another example, suppose Alice sends a message instructing her banker, Bob, to pay Eve for consulting work. The message does not need to be hidden from Eve because she already knows she is to be paid. However, the message does need to be safe from any tampering, especially by Eve.

Document and Fingerprint

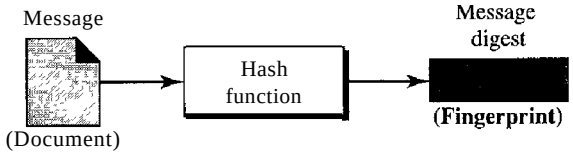
One way to preserve the integrity of a document is through the use of a fingerprint. If Alice needs to be sure that the contents of her document will not be illegally changed, she can put her fingerprint at the bottom of the document. Eve cannot modify the contents of this document or create a false document because she cannot forge Alice's fingerprint. To ensure that the document has not been changed, Alice's fingerprint on the document can be compared to Alice's fingerprint on file. If they are not the same, the document is not from Alice.

To preserve the integrity of a **document,**
both the document and the fingerprint are needed.

Message and Message Digest

The electronic equivalent of the document and fingerprint pair is the message and message digest pair. To preserve the integrity of a message, the message is passed through an algorithm called a hash function. The hash function creates a compressed image of the message that can be used as a fingerprint. Figure 31.4 shows the message, hash function, and the message digest.

Figure 31.4 *Message and message digest*



Difference

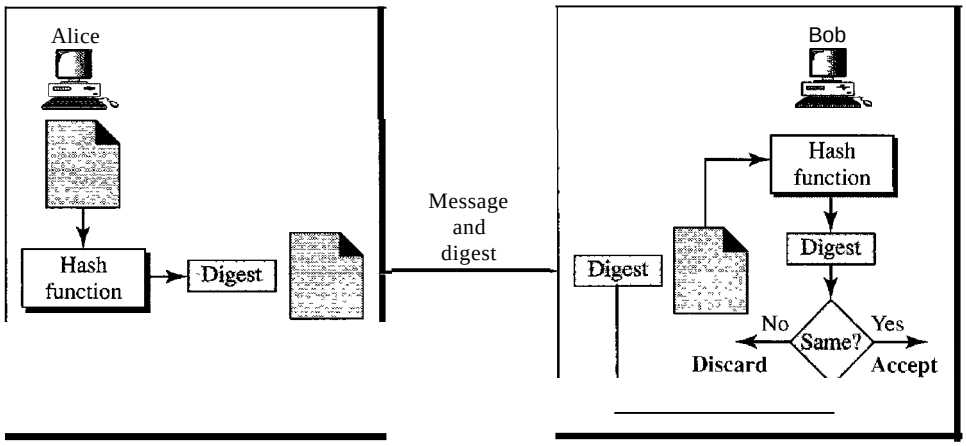
The two pairs document/fingerprint and message/message digest are similar, with some differences. The document and fingerprint are physically linked together; also, neither needs to be kept secret. The message and message digest can be unlinked (or sent) separately and, most importantly, the message digest needs to be kept secret. The message digest is either kept secret in a safe place or encrypted if we need to send it through a communications channel.

The message digest needs to be kept secret.

Creating and Checking the Digest

The message digest is created at the sender site and is sent with the message to the receiver. To check the integrity of a message, or document, the receiver creates the hash function again and compares the new message digest with the one received. If both are the same, the receiver is sure that the original message has not been changed. Of course, we are assuming that the digest has been sent secretly. Figure 31.5 shows the idea.

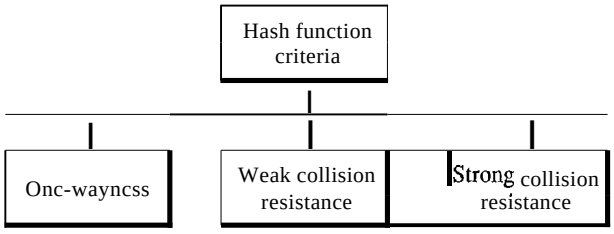
Figure 31.5 *Checking integrity*



Hash Function Criteria

To be eligible for a hash, a function needs to meet three criteria: one-wayness, resistance to weak collision, and resistance to strong collision as shown in Figure 31.6.

Figure 31.6 *Criteria of a hash function*



One-wayness

A hash function must have **one-wayness;** a message digest is created by a one-way hashing function. We must not be able to recreate the message from the digest. Sometimes it is difficult to make a hash function 100 percent one-way; the criteria state that it must be extremely difficult or impossible to create the message if the message digest is given. This is similar to the document/fingerprint case. No one can make a document from a fingerprint.

Example 31.1

Can we use a conventional lossless compression method as a hashing function?

Solution

We cannot. A lossless compression method creates a compressed message that is reversible. You can uncompress the compressed message to get the original one.

Example 31.2

Can we use a checksum method as a hashing function?

Solution

We can. A checksum function is not reversible; it meets the first criterion. However, it does not meet the other criteria.

Weak Collision Resistance

The second criterion, weak collision resistance, ensures that a message cannot easily be forged. **If** Alice creates a message and a digest and sends both to Bob, this criterion ensures that Eve cannot easily create another message that hashes exactly to the same digest. In other words, given a specific message and its digest, it is impossible (or at least very difficult) to create another message with the same digest.

When two messages create the same digest, we say there is a collision. In a week collision, given a message digest, it is very unlikely that someone can create a message with exactly the same digest. A hash function must have weak collision resistance.

Strong Collision Resistance

The third criterion, strong collision resistance, ensures that we cannot find two messages that hash to the same digest. This criterion is needed to ensure that Alice, the sender of the message, cannot cause problems by forging a message. **If** Alice can create two messages that hash to the same digest, she can deny sending the first to Bob and claim that she sent only the second.

This type of collision is called strong because the probability of collision is higher than in the previous case. An adversary can create two messages that hash to the same digest. For example, if the number of bits in the message digest is small, it is likely Alice can create two different messages with the same message digest. She can send the first to Bob and keep the second for herself. Alice can later say that the second was the original agreed-upon document and not the first.

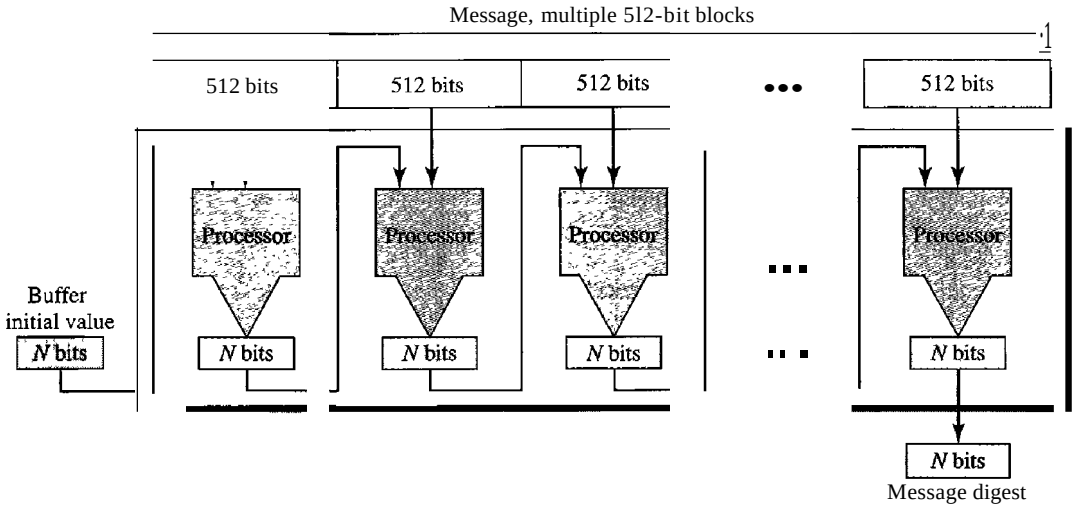
Suppose two different wills can be created that hash to the same digest. When the time comes for the execution of the will, the second will is presented to the heirs. Since the digest matches both wills, the substitution is successful.

Hash Algorithms: SHA-l

While many hash algorithms have been designed, the most common is SHA-l. SHA-1 (Secure Hash Algorithm 1) is a revised version of SHA designed by the National Institute of Standards and Technology (NIST). It was published as a Federal Information Processing Standard (PIPS).

A very interesting point about this algorithm and others is that they all follow the same concept. Each creates a digest of length *N* from a multiple-block message. Each block is 512 bits in length, as shown in Figure 31.7.

Figure 31.7 *Message digest creation*



A buffer of *N* bits is initialized to a predetermined value. The algorithm mangles this initial buffer with the first 512 bits of the message to create the first intermediate message digest of *N* bits. This digest is then mangled with the second 512-bit block to create the second intermediate digest. The *(n* - l)th digest is mangled with the *nth* block to create the *nth* digest. If a block is not 512 bits, padding (Os) is added to make it so. When the last block is processed, the resulting digest is the message digest for the entire message. SHA-l has a message digest of 160 bits (5 words, each of 32 bits).

SHA-l hash algorithms create anN-bit message digest out of a message of 512-bit blocks.

SHA-l has a message digest of 160 bits (5 words of 32 bits).

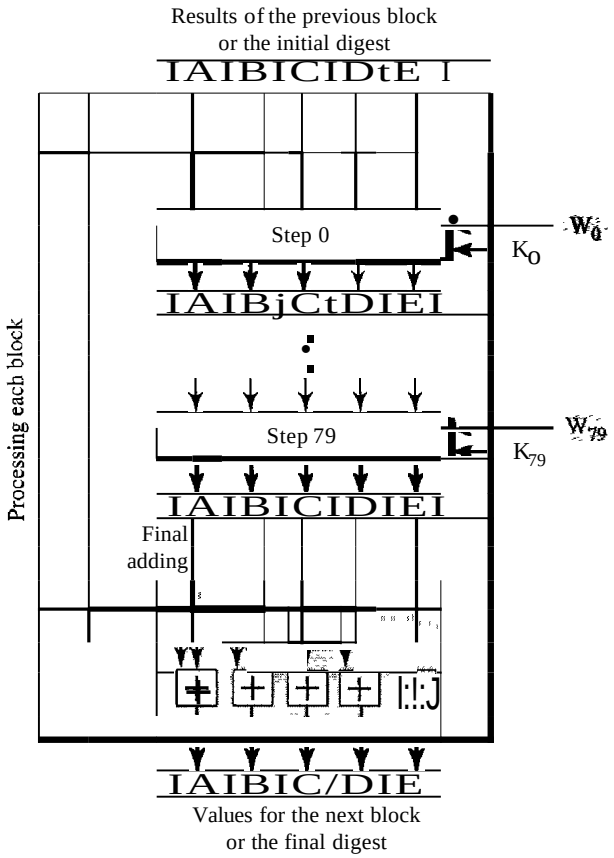
Word Expansion

Before processing, the block needs to be expanded. A block is made of 512 bits or 16 32-bit words, but we need 80 words in the processing phase. So the 16-word block needs to be expanded to 80 words, wordO to word79.

Processing Each Block

Figure 31.8 shows the general outline for the processing of one block. There are 80 steps in block processing. In each step, one word from the expanded block and one 32-bit constant are mangled together and then operated on to create a new digest. At the beginning of processing, the values of digest words (A, B, C, D, and E) are saved into five temporary variables. At the end of the processing (after step 79), these values are

Figure 31.8 *Processing ofone block in SHA-1*



added to the values created from step 79. The detail of each step is complex and beyond the scope of this book. The only thing we need to know is that each step mangles a word of data and a constant to create a result that is fed to the next step.

31.4 MESSAGE AUTHENTICATION

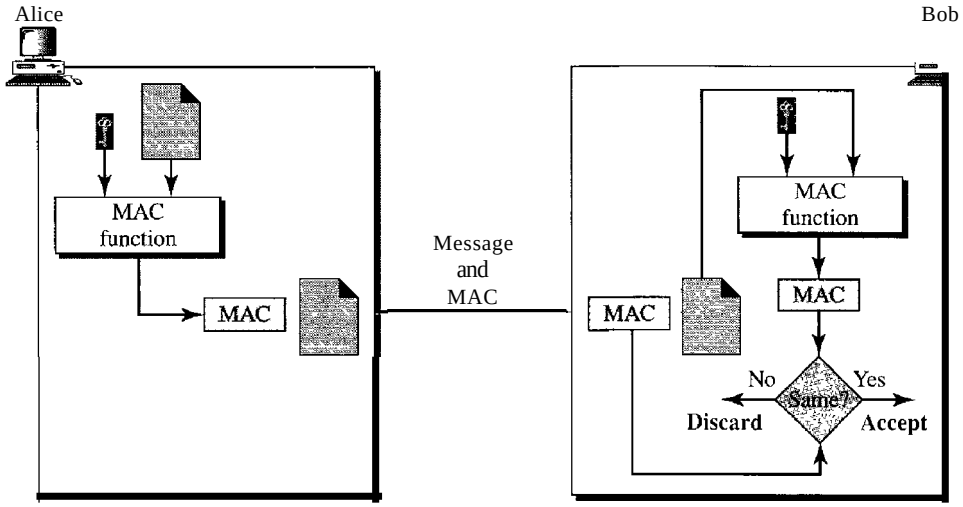
A hash function guarantees the integrity of a message. It guarantees that the message has not been changed. A hash function, however, does not authenticate the sender of the message. When Alice sends a message to Bob, Bob needs to know if the message is coming from Alice or Eve. To provide message authentication, Alice needs to provide proof that it is Alice sending the message and not an imposter. A hash function per se cannot provide such a proof. The digest created by a hash function is normally called a modification detection code (MDC). The code can detect any modification in the message.

MAC

To provide message authentication, we need to change a modification detection code to a message authentication code (MAC). An MDC uses a keyless hash function; a MAC uses a keyed hash function. A keyed hash function includes the symmetric key

between the sender and receiver when creating the digest. Figure 31.9 shows how Alice uses a keyed hash function to authenticate her message and how Bob can verify the authenticity of the message.

Figure 31.9 *MAC, created by Alice and checked by Bob*



Alice, using the symmetric key between herself and Bob (K_{AB}) and a keyed hash function, generates a MAC. She then concatenates the MAC with the original message and sends the two to Bob. Bob receives the message and the MAC. He separates the message from the MAC. He applies the same keyed hash function to the message using the symmetric key K_{AB} to get a fresh MAC. He then compares the MAC sent by Alice with the newly generated MAC. If the two MACs are identical, the message has not been modified and the sender of the message is definitely Alice.

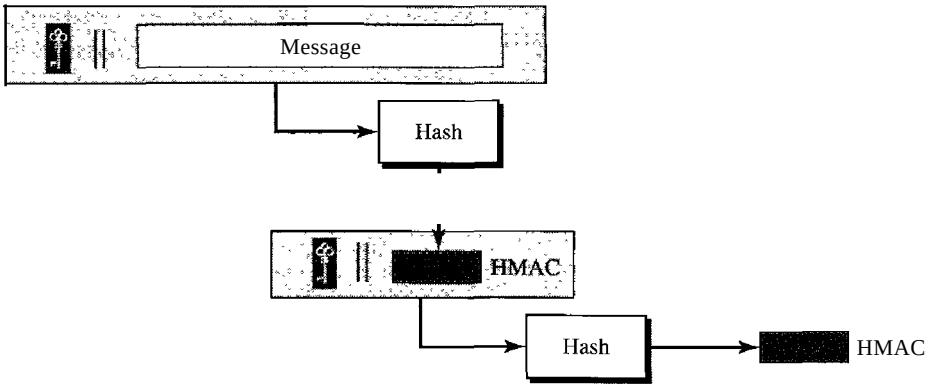
HMAC

There are several implementations of MAC in use today. However, in recent years, some MACs have been designed that are based on keyless hash functions such as SHA-1. This idea is a hashed MAC, called HMAC, that can use any standard keyless hash function such as SHA-1. HMAC creates a nested MAC by applying a keyless hash function to the concatenation of the message and a symmetric key. Figure 31.10 shows the general idea.

A copy of the symmetric key is prepended to the message. The combination is hashed using a keyless hash function, such as SHA-1. The result of this process is an intermediate HMAC which is again prepended with the key (the same key), and the result is again hashed using the same algorithm. The final result is an HMAC.

The receiver receives this final HMAC and the message. The receiver creates its own HMAC from the received message and compares the two HMACs to validate the integrity of the message and authenticate the data origin. Note that the details of an HMAC can be more complicated than what we have shown here.

Figure 31.10 *HMAC*



31.5 DIGITAL SIGNATURE

Although a MAC can provide message integrity and message authentication, it has a drawback. It needs a symmetric key that must be established between the sender and the receiver. A digital signature, on the other hand, can use a pair of asymmetric keys (a public one and a private one).

We are all familiar with the concept of a signature. We sign a document to show that it originated from us or was approved by us. The signature is proof to the recipient that the document comes from the correct entity. When a customer signs a check to himself, the bank needs to be sure that the check is issued by that customer and nobody else. In other words, a signature on a document, when verified, is a sign of authentication; the document is authentic. Consider a painting signed by an artist. The signature on the art, if authentic, means that the painting is probably authentic.

When Alice sends a message to Bob, Bob needs to check the authenticity of the sender; he needs to be sure that the message comes from Alice and not Eve. Bob can ask Alice to sign the message electronically. In other words, an electronic signature can prove the authenticity of Alice as the sender of the message. We refer to this type of signature as a digital signature.

Comparison

Before we continue any further, let us discuss the differences between two types of signatures: conventional and digital.

Inclusion

A conventional signature is included in the document; it is part of the document. When we write a check, the signature is on the check; it is not a separate document. On the other hand, when we sign a document digitally, we send the signature as a separate document. The sender sends two documents: the message and the signature. The recipient receives both documents and verifies that the signature belongs to the supposed sender. If this is proved, the message is kept; otherwise, it is rejected.

Verification Method

The second difference between the two types of documents is the method of verifying the signature. In conventional signature, when the recipient receives a document, she compares the signature on the document with the signature on file. If they are the same, the document is authentic. The recipient needs to have a copy of this signature on file for comparison. In digital signature, the recipient receives the message and the signature. A copy of the signature is not stored anywhere. The recipient needs to apply a verification technique to the combination of the message and the signature to verify the authenticity.

Relationship

In conventional signature, there is normally a one-to-many relationship between a signature and documents. A person, for example, has a signature that is used to sign many checks, many documents, etc. In digital signature, there is a one-to-one relationship between a signature and a message. Each message has its own signature. The signature of one message cannot be used in another message. If Bob receives two messages, one after another, from Alice, he cannot use the signature of the first message to verify the second. Each message needs a new signature.

Duplicity

Another difference between the two types of signatures is a quality called duplicity. In conventional signature, a copy of the signed document can be distinguished from the original one on file. In digital signature, there is no such distinction unless there is a factor of time (such as a timestamp) on the document. For example, suppose Alice sends a document instructing Bob to pay Eve. If Eve intercepts the document and the signature, she can resend it later to get money again from Bob.

Need for Keys

In conventional signature a signature is like a private "key" belonging to the signer of the document. The signer uses it to sign a document; no one else has this signature. The copy of the signature is on file like a public key; anyone can use it to verify a document, to compare it to the original signature.

In digital signature, the signer uses her private key, applied to a signing algorithm, to sign the document. The verifier, on the other hand, uses the public key of the signer, applied to the verifying algorithm, to verify the document.

Can we use a secret (symmetric) key to both sign and verify a signature? The answer is no for several reasons. First, a secret key is known only between two entities (Alice and Bob, for example). So if Alice needs to sign another document and send it to Ted, she needs to use another secret key. Second, as we will see, creating a secret key for a session involves authentication, which normally uses digital signature. We have a vicious cycle. Third, Bob could use the secret key between himself and Alice, sign a document, send it to Ted, and pretend that it came from Alice.

A digital signature needs a public-key system.

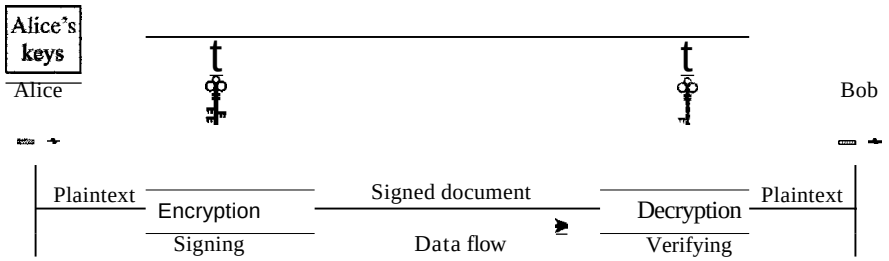
Process

Digital signature can be achieved in two ways: signing the document or signing a digest of the document.

Signing the Document

Probably, the easier, but less efficient way is to sign the document itself. Signing a document is encrypting it with the private key of the sender; verifying the document is decrypting it with the public key of the sender. Figure 31.11 shows how signing and verifying are done.

Figure 31.11 *Signing the message itself in digital signature*



We should make a distinction between private and public keys as used in digital signature and public and private keys as used for confidentiality. In the latter, the private and public keys of the receiver are used in the process. The sender uses the public key of the receiver to encrypt; the receiver uses his own private key to decrypt. In digital signature, the private and public keys of the sender are used. The sender uses her private key; the receiver uses the public key of the sender.

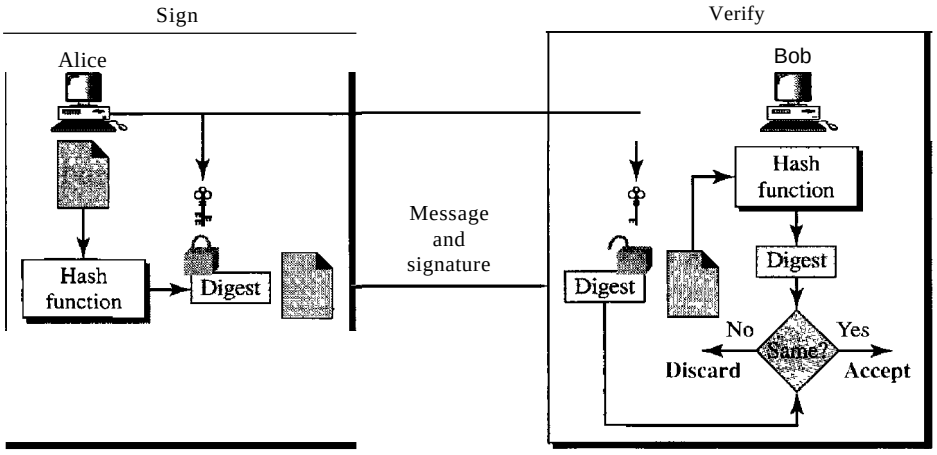
> In a cryptosystem, we use the private and public keys of the receiver; in digital signature, we use the private and public key of the sender.

Signing the Digest

We mentioned that the public key is very inefficient in a cryptosystem if we are dealing with long messages. In a digital signature system, our messages are normally long, but we have to use public keys. The solution is not to sign the message itself; instead, we sign a digest of the message. As we learned, a carefully selected message digest has a one-to-one relationship with the message. The sender can sign the message digest, and the receiver can verify the message digest. The effect is the same. Figure 31.12 shows signing a digest in a digital signature system.

A digest is made out of the message at Alice's site. The digest then goes through the signing process using Alice's private key. Alice then sends the message and the signature to Bob. As we will see later in the chapter, there are variations in the process that are dependent on the system. For example, there might be additional calculations before the digest is made or other secret keys might be used. In some systems, the signature is a set of values.

Figure 31.12 *Signing the digest in a digital signature*



At Bob's site, using the same public hash function, a digest is first created out of the received message. Calculations are done on the signature and the digest. The verifying process also applies criteria on the result of the calculation to detennine the authenticity of the signature. If authentic, the message is accepted; otherwise, it is rejected.

Services

A digital signature can provide three out of the five services we mentioned for a security system: message integrity, message authentication, and nonrepudiation. Note that a digital signature scheme does not provide confidential communication. If confidentiality is required, the message and the signature must be encrypted using either a secret-key or public-key cryptosystem.

Message Integrity

The integrity of the message is preserved even if we sign the whole message because we cannot get the same signature if the message is changed. The signature schemes today use a hash function in the signing and verifying algorithms that preserve the integrity of the message.

A digital signature today provides message integrity.

Message Authentication

A secure signature scheme, like a secure conventional signature (one that cannot be easily copied), can provide message authentication. Bob can verify that the message is sent by Alice because Alice's public key is used in verification. Alice's public key cannot create the same signature as Eve's private key.

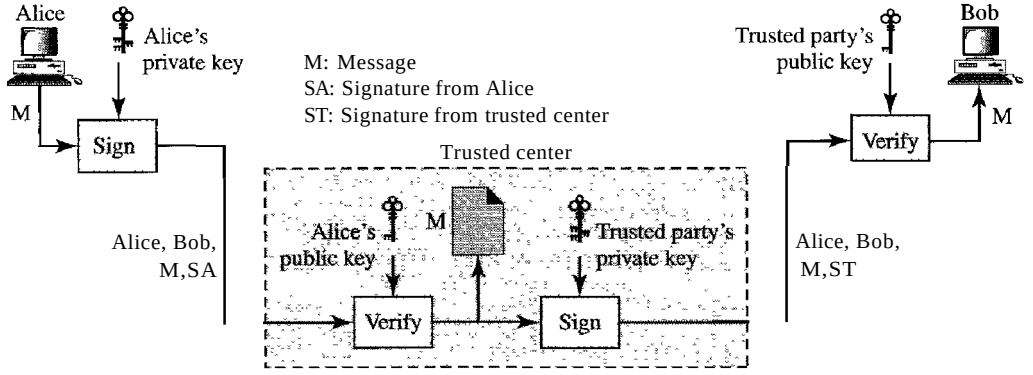
Digital signature provides message authentication.

Message Nonrepudiation

If Alice signs a message and then denies it, can Bob later prove that Alice actually signed it? For example, if Alice sends a message to a bank (Bob) and asks to transfer $10,000 from her account to Ted's account, can Alice later deny that she sent this message? With the scheme we have presented so far, Bob might have a problem. Bob must keep the signature on file and later use Alice's public key to create the original message to prove the message in the file and the newly created message are the same. This is not feasible because Alice may have changed her private/public key during this time; she may also claim that the file containing the signature is not authentic.

One solution is a trusted third party. People can create a trusted party among themselves. In Chapter 32, we will see that a trusted party can solve many other problems concerning security services and key exchange. Figure 31.13 shows how a trusted party can prevent Alice from denying that she sent the message.

Figure 31.13 *Using a trusted center for nonrepudiation*



Alice creates a signature from her message (SA) and sends the message, her identity, Bob's identity, and the signature to the center. The center, after checking that Alice's public key is valid, verifies through Alice's public key that the message comes from Alice. The center then saves a copy of the message with the sender identity, recipient identity, and a timestamp in its archive. The center uses its private key to create another signature (ST) from the message. The center then sends the message, the new signature, Alice's identity, and Bob's identity to Bob. Bob verifies the message using the public key of the trusted center.

If in the future Alice denies that she has sent the message, the center can show a copy of the saved message. If Bob's message is a duplicate of the message saved at the center, Alice will lose the dispute. To make everything confidential, a level of encryption! decryption can be added to the scheme as discussed in the next section.

Nonrepudiation can be provided using a trusted party.

Signature Schemes

Several signature schemes have evolved during the last few decades. Some of them have been implemented. Such as RSA and DSS (Digital Signature Standard) schemes. The latter will probably become the standard. However, the details of these schemes are beyond the scope of this book.

31.6 ENTITY AUTHENTICATION

Entity authentication is a technique designed to let one party prove the identity of another party. An *entity* can be a person, a process, a client, or a server. The entity whose identity needs to be proved is called the claimant; the party that tries to prove the identity of the claimant is called the verifier. When Bob tries to prove the identity of Alice, Alice is the claimant, and Bob is the verifier.

There are two differences between message authentication and entity authentication. First, message authentication may not happen in real time; entity authentication does. In the former, Alice sends a message to Bob. When Bob authenticates the message, Alice mayor may not be present in the communication process. On the other hand, when Alice requests entity authentication, there is no real message communication involved until Alice is authenticated by Bob. Alice needs to be online and takes part in the process. Only after she is authenticated can messages be communicated between Alice and Bob. Message authentication is required when an e-mail is sent from Alice to Bob. Entity authentication is required when Alice gets cash from an automatic teller machine. Second, message authentication simply authenticates one message; the process needs to be repeated for each new message. Entity authentication authenticates the claimant for the entire duration of a session.

In entity authentication, the claimant must identify herself to the verifier. This can be done with one of three kinds of witnesses: *something known, something possessed,* or *something inherent.*

- *Something known.* This is a secret known only by the claimant that can be checked by the verifier. Examples are a password, a PIN number, a secret key, and a private key.
- *Something possessed.* This is something that can prove the claimant's identity. Examples are a passport, a driver's license, an identification card, a credit card, and a smart card.
- *Something inherent.* This is an inherent characteristic of the claimant. Examples are conventional signature, fingerprints, voice, facial characteristics, retinal pattern, and handwriting.

Passwords

The simplest and the oldest method of entity authentication is the password, something that the claimant *possesses.* A password is used when a user needs to access a system to use the system's resources (log-in). Each user has a user identification that is public and a password that is private. We can divide this authentication scheme into two separate groups: the fixed password and the one-time password.

Fixed Password

In this group, the password is fixed; the same password is used over and over for every access. This approach is subject to several attacks.

- Eavesdropping. Eve can watch Alice when she types her password. Most systems, as a security measure, do not show the characters a user types. Eavesdropping can take a more sophisticated form. Eve can listen to the line and then intercept the message, thereby capturing the password for her own use.
- Stealing a Password. The second type of attack occurs when Eve tries to physically steal Alice's password. This can be prevented if Alice does not write down the password; instead, she just commits it to memory. Therefore, a password should be very simple or else related to something familiar to Alice, which makes the password vulnerable to other types of attacks.
- Accessing a file. Eve can hack into the system and get access to the file where the passwords are stored. Eve can read the file and find Alice's password or even change it. To prevent this type of attack, the file can be read/write protected. However, most systems need this type of file to be readable by the public.
- Guessing. Eve can log into the system and try to guess Alice's password by trying different combinations of characters. The password is particularly vulnerable if the user is allowed to choose a short password (a few characters). It is also vulnerable if Alice has chosen something unimaginative, such as her birthday, her child's name, or the name of her favorite actor. To prevent guessing, a long random password is recommended, something that is not very obvious. However, the use of such a random password may also create a problem; Alice might store the password somewhere so as not to forget it. This makes the password subject to stealing.

A more secure approach is to store the hash of the password in the password file (instead of the plaintext password). Any user can read the contents of the file, but, because the hash function is a one-way function, it is almost impossible to guess the value of the password. The hash function prevents Eve from gaining access to the system even though she has the password file. However, there is a possibility of another type of attack called the dictionary attack. In this attack, Eve is interested in finding one password, regardless of the user ID. For example, if the password is 6 digits, Eve can create a list of 6-digit numbers (000000 to 999999), and then apply the hash function to every number; the result is a list of 1 million hashes. She can then get the password file and search the second-column entries to find a match. This could be programmed and run offline on Eve's private computer. After a match is found, Eve can go online and use the password to access the system. We will see how to make this attack more difficult in the third approach.

Another approach is called salting the password. When the password string is created, a random string, called the salt, is concatenated to the password. The salted password is then hashed. The rD, salt, and the hash are then stored in the file. Now, when a user asks for access, the system extracts the salt, concatenates it with the received password, makes a hash out of the result, and compares it with the hash stored in the file. If there is a match, access is granted; otherwise, it is denied. Salting makes the dictionary attack more difficult. If the original password *is* 6 digits and *the* salt *is* 4 digits, then hashing *is*

done over a lO-digit value. This means that Eve now needs to make a list of 10 million items and create a hash for each of them. The list of hashes has 10 million entries and the comparison takes much longer. Salting is very effective if the salt is a very long random number. The UNIX operating system uses a variation of this method.

In another approach, two identification techniques are combined. A good example of this type of authentication is the use of an ATM card with a PIN (personal identification number). The card belongs to the category "something possessed" and the PIN belongs to the category "something known." The PIN is actually a password that enhances the security of the card. If the card is stolen, it cannot be used unless the PIN is known. The PIN, however, is traditionally very short so it is easily remembered by the owner. This makes it vulnerable to the guessing type of attack.

One-Time Password

In this type of scheme, a password is used only once. It is called the one-time password. A one-time password makes eavesdropping and stealing useless. However, this approach is very complex, and we leave its discussion to some specialized books.

Challenge-Response

In password authentication, the claimant proves her identity by demonstrating that she knows a secret, the password. However, since the claimant reveals this secret, the secret is susceptible to interception by the adversary. In challenge-response authentication, the claimant proves that she *knows* a secret without revealing it. In other words, the claimant does not reveal the secret to the verifier; the verifier either has it or finds it.

In challenge-response authentication,
the claimant proves that she knows a secret without revealing it.

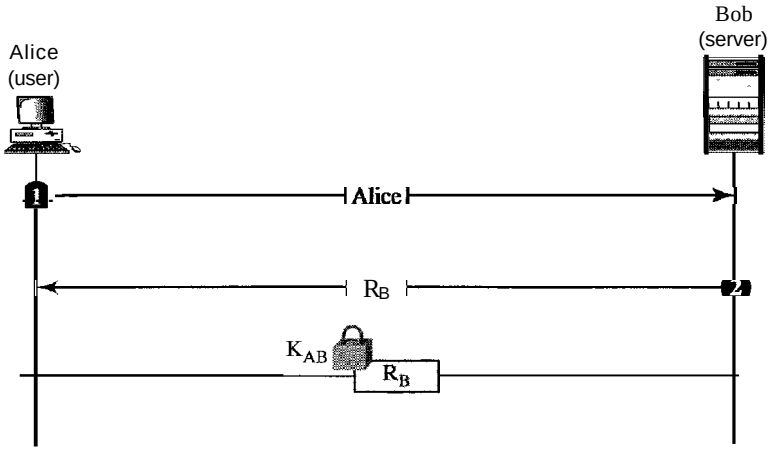
The challenge is a time-varying value such as a random number or a timestamp which is sent by the verifier. The claimant applies a function to the challenge and sends the result, called a *response,* to the verifier. The response shows that the claimant knows the secret.

The challenge is a time-varying value sent by the verifier;
the response is the result of a function applied on the chanenge.

Using a Symmetric-Key Cipher

In the first category, the challenge-response authentication is achieved using symmetric-key encryption. The secret here is the shared secret key, known by both the claimant and the verifier. The function is the encrypting algorithm applied on the challenge. Figure 31.14 shows one approach. The first message is not part of challenge-response, it only informs the verifier that the claimant wants to be challenged. The second message is the challenge. And R_B is the nonce randomly chosen by the verifier to challenge the claimant. The claimant encrypts the nonce using the shared secret key known only to the claimant and the verifier and sends the result to the verifier. The verifier decrypts

Figure 31.14 *Challenge/response authentication using a nonce*



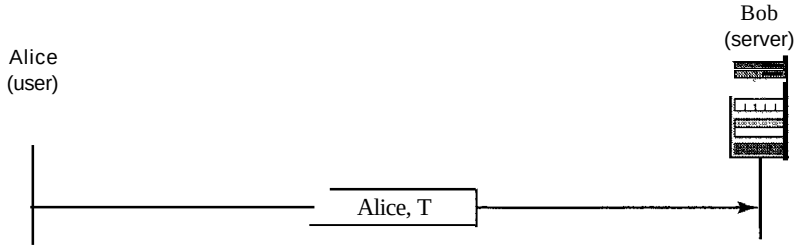
the message. If the nonce obtained from decryption is the same as the one sent by the verifier, Alice is granted access.

Note that in this process, the claimant and the verifier need to keep the symmetric key used in the process secret. The verifier must also keep the value of the nonce for claimant identification until the response is returned.

The reader may have noticed that use of a nonce prevents a replay of the third message by Eve. Eve cannot replay the third message and pretend that it is a new request for authentication by Alice because once Bob receives the response, the value of R_B is not valid any more. The next time a new value is used.

In the second approach, the time-varying value is a timestamp, which obviously changes with time. **In** this approach the challenge message is the current time sent from the verifier to the claimant. However, this supposes that the client and the server clocks are synchronized; the claimant knows the current time. This means that there is no need for the challenge message. The first and third messages can be combined. The result is that authentication can be done using one message, the response to an implicit challenge, the current time. Figure 31.15 shows the approach.

Figure 31.15 *Challenge-response authentication using a timestamp*



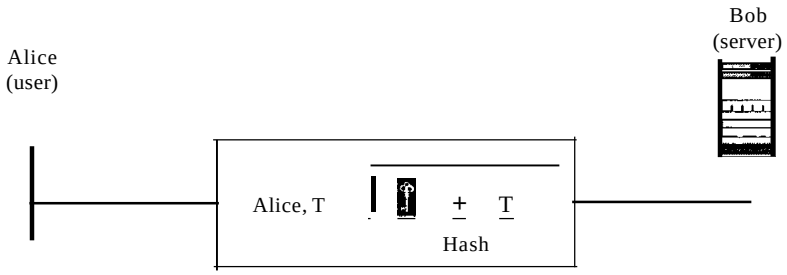
Using Keyed-Hash Functions

Instead of using encryption and decryption for entity authentication, we can use a keyed-hash function (MAC). There are two advantages to this scheme. First, the

encryption/decryption algorithm is not exportable to some countries. Second, in using a keyed-hash function, we can preserve the integrity of challenge and response messages and at the same time use a secret, the key.

Let us see how we can use a keyed-hash function to create a challenge response with a timestamp. Figure 31.16 shows the scheme.

Figure 31.16 *Challenge-response authentication using a keyed-hash function*

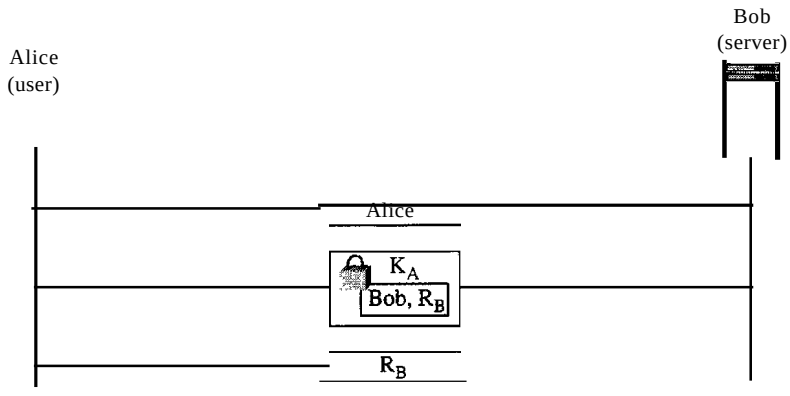


Note that in this case, the timestamp is sent both as plaintext and as text scrambled by the keyed-hash function. When Bob receives the message, he takes the plaintext T, applies the keyed-hash function, and then compares his calculation with what he received to determine the authenticity of Alice.

Using an Asymmetric-Key Cipher

Instead of a symmetric-key cipher, we can use an asymmetric-key cipher for entity authentication. Here the secret must be the private key of the claimant. The claimant must show that she owns the private key related to the public key that is available to everyone. This means that the verifier must encrypt the challenge using the public key of the claimant; the claimant then decrypts the message using her private key. The response to the challenge is the decrypted challenge. We show two approaches: one for unidirectional authentication and one for bidirectional authentication. In one approach, Bob encrypts the challenge using Alice's public key. Alice decrypts the message with her private key and sends the nonce to Bob. Figure 31.17 shows this approach.

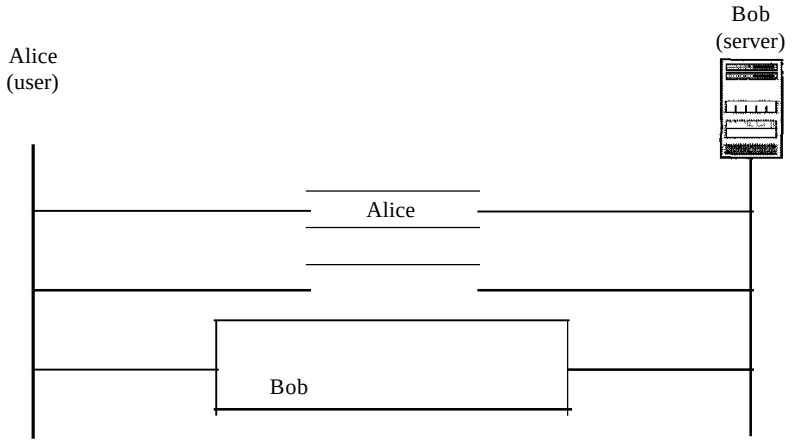
Figure 31.17 *Authentication, asymmetric-key*



Using Digital Signature

We can use digital signature for entity authentication. In this method, we let the claimant use her private key for signing instead of using it for decryption. In one approach shown in Figure 31.18, Bob uses a plaintext challenge. Alice signs the response.

Figure 31.18 *Authentication, using digital signature*



31.7 KEY MANAGEMENT

We have used symmetric-key and asymmetric-key cryptography in our discussion throughout the chapter. However, we never discussed how secret keys in symmetric-key cryptography and how public keys in asymmetric-key cryptography are distributed and maintained. In this section, we touch on these two issues. We first discuss the distribution of symmetric keys; we then discuss the distribution of asymmetric keys.

Symmetric-Key **Distribution**

We have learned that symmetric-key cryptography is more efficient than asymmetric-key cryptography when we need to encrypt and decrypt large messages. Symmetric-key cryptography, however, needs a shared secret key between two parties.

If Alice needs to exchange confidential messages with N people, she needs N different keys. What if N people need to communicate with one another? A total of $N(N - 1)/2$ keys is needed. Each person needs to have $N - 1$ keys to communicate with each of the other people, but because the keys are shared, we need only $N(N - 1)12$. This means that if 1 million people need to communicate with one another, each person has almost 0.5 million different keys; in total, almost 1 billion keys are needed. This is normally referred to as the N^2 problem because the number of required keys for N entities is close to N^2.

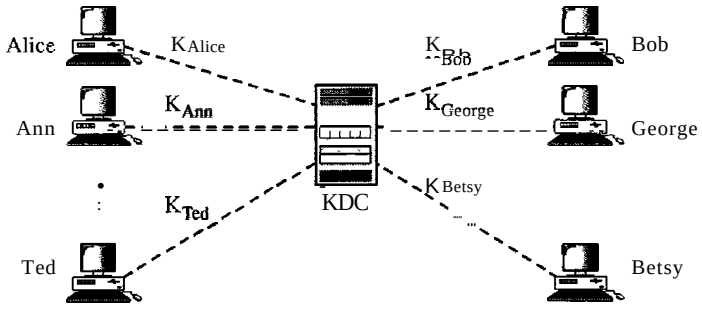
The number of keys is not the only problem; the distribution of keys is another. If Alice and Bob want to communicate, they need to somehow exchange a secret key; if Alice wants to communicate with 1 million people, how can she exchange 1 million keys with 1 million people? Using the Internet is definitely not a secure method.

It is obvious that we need an efficient way of maintaining and distributing secret keys.

Key Distribution Center: KDC

A practical solution is the use of a trusted party, referred to as a key distribution center (KDC). To reduce the number of keys, each person establishes a shared secret key with the KDC as shown in Figure 31.19.

Figure 31.19 *KDC*



A secret key is established between KDC and each member. Alice has a secret key with KDC, which we refer to as K_{Alice}; Bob has a secret key with KDC, which we refer to as K_{Bob}; and so on. Now the question is, How can Alice send a confidential message to Bob? The process is as follows:

1. Alice sends a request to KDC, stating that she needs a session (temporary) secret key between herself and Bob.
2. KDC informs Bob of Alice's request.
3. If Bob agrees, a session key is created between the two.

The secret key between Alice and Bob that is established with the KDC is used to authenticate Alice and Bob to the KDC and to prevent Eve from impersonating either of them. We discuss how a session key is established between Alice and Bob later in the chapter.

Session Keys

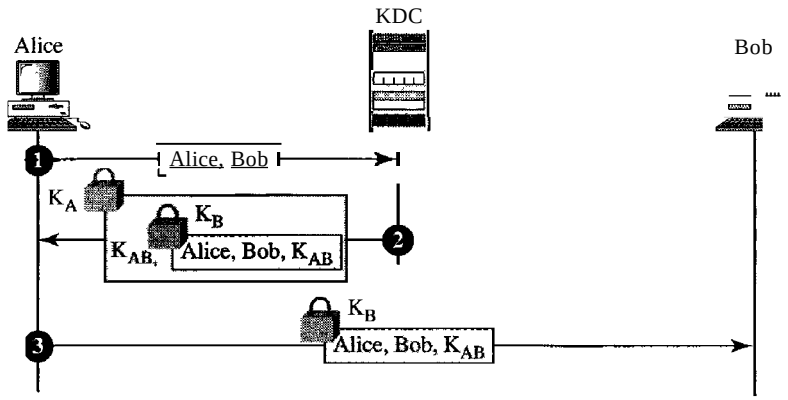
A KDC creates a secret key for each member. This secret key can be used only between the member and the KDC, not between two members. If Alice needs to communicate secretly with Bob, she needs a secret key between herself and Bob. A KDC can create a session (temporary) key between Alice and Bob using their keys with the center. The keys of Alice and Bob are used to authenticate Alice and Bob to the center and to each other before the session key is established. After communication is terminated, the session key is no longer valid.

A session symmetric key between two parties is used only once.

Several different approaches have been proposed to create the session key using ideas we previously discussed for entity authentication.

Let us discuss one approach, the simplest one, as shown in Figure 31.20. Although this system has some flaws, it shows the idea. More sophisticated approaches can be found in security books.

Figure 31.20 *Creating a session key between Alice and Bob using KDC*



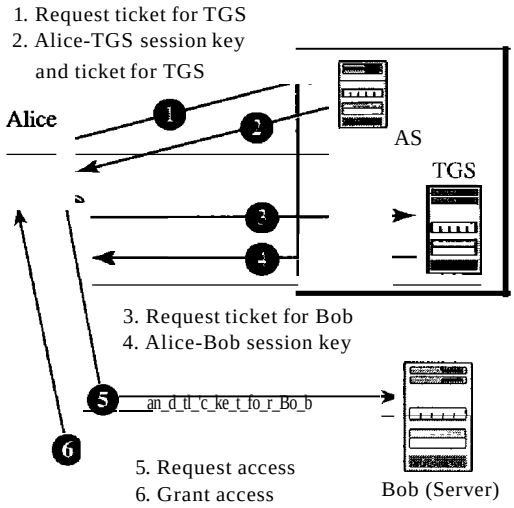
- Step 1 Alice sends a plaintext message to the KDC to obtain a symmetric session key between Bob and herself. The message contains her registered identity (the word *Alice* in the figure) and the identity of Bob (the word *Bob* in the figure). This message is not encrypted, it is public. KDC does not care.
- Step 2 KDC receives the message and creates what is called a ticket. The ticket is encrypted using Bob's key (K_B). The ticket contains the identities of Alice and Bob and the session key (K_{AB}). The ticket with a copy of the session key is sent to Alice. Alice receives the message, decrypts it, and extracts the session key. She cannot decrypt Bob's ticket; the ticket is for Bob, not for Alice. Note that we have a double encryption in this message; the ticket is encrypted and the entire message is also encrypted. In the second message, Alice is actually authenticated to the KDC, because only Alice can open the whole message using her secret key with KDC.
- Step 3 Alice sends the ticket to Bob. Bob opens the ticket and knows that Alice needs to send messages to him using K_{AB} as the session key. Note that in this message, Bob is authenticated to the KDC because only Bob can open the ticket. Since Bob is authenticated to the KDC, he is also authenticated to Alice who trusts the KDC. In the same way, Alice is also authenticated to Bob, because Bob trusts the KDC and the KDC has sent the ticket to Bob which includes the identity of Alice.

Kerberos

Kerberos is an authentication protocol and at the same time a KDC that has become very popular. Several systems including Windows 2000 use Kerberos. It is named after the three-headed dog in Greek mythology that guards the Gates of Hades. Originally designed at M.LT., it has gone through several versions. We discuss only version 4, the most popular.

Servers Three servers are involved in the Kerberos protocol: an authentication server (AS), a ticket-granting server (TGS), and a real (data) server that provides services to others. In our examples and figures *Bob* is the real server and *Alice* is the user requesting service. Figure 31.21 shows the relationship between these three servers.

Figure 31.21 *Kerberos servers*

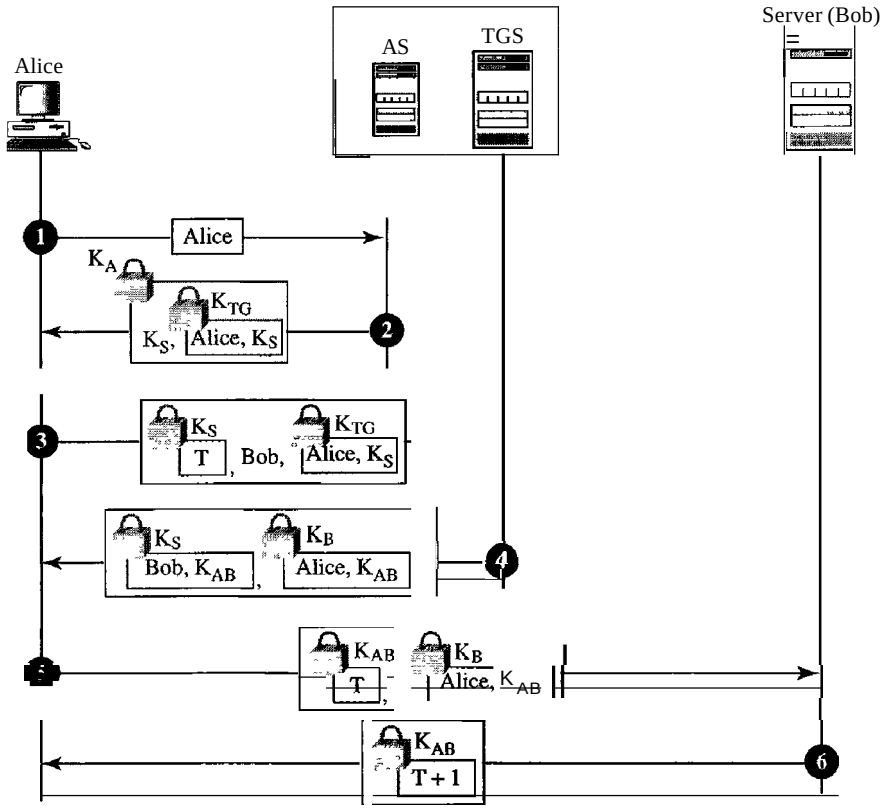


- Authentication Server (AS). AS is the KDC in Kerberos protocol. Each user registers with AS and is granted a user identity and a password. AS has a database with these identities and the corresponding passwords. AS verifies the user, issues a session key to be used between Alice and TGS, and sends a ticket for TGS.
- Ticket-Granting Server (TGS). TGS issues a ticket for the real server (Bob). It also provides the session key (K_{AB}) between Alice and Bob. Kerberos has separated the user verification from ticket issuing. In this way, although Alice verifies her ID just once with AS, she can contact TGS multiple times to obtain tickets for different real servers.
- Real Server. The real server (Bob) provides services for the user (Alice). Kerberos is designed for a client/server program such as FTP, in which a user uses the client process to access the server process. Kerberos is not used for person-to-person authentication.

Operation A client process (Alice) can access a process running on the real server (Bob) in six steps as shown in Figure 31.22.

- Step 1. Alice sends her request to AS in plaintext, using her registered identity.
- Step 2. AS sends a message encrypted with Alice's symmetric key K_A. The message contains two items: a session key K_S that is used by Alice to contact TGS and a ticket for TGS that is encrypted with the TGS symmetric key K_{TG}. Alice does not know K_A, but when the message arrives, she types her symmetric password. The password and the appropriate algorithm together create K_A if the password is correct. The password is then immediately destroyed; it is not sent to the network, and it does

Figure 31.22 *Kerberos example*



not stay in the terminal. It is only used for a moment to create K_A. The process now uses K_A to decrypt the message sent. Both K_S and the ticket are extracted.

- Step 3. Alice now sends three items to TGS. The first is the ticket received from AS. The second is the name of the real server (Bob), the third is a timestamp which is encrypted by K_S. The timestamp prevents a replay by Eve.
- Step 4. Now, TGS sends two tickets, each containing the session key between Alice and Bob K_{AB}. The ticket for Alice is encrypted with K_S; the ticket for Bob is encrypted with Bob's key K_B. Note that Eve cannot extract K_{AB} because she does not know K_S or K_B. She cannot replay step 3 because she cannot replace the timestamp with a new one (she does not know K_S)' Even if she is very quick and sends the step 3 message before the timestamp has expired, she still receives the same two tickets that she cannot decipher.
- Step 5. Alice sends Bob's ticket with the timestamp encrypted by K_{AB}.
- Step 6. Bob confirms the receipt by adding 1 to the timestamp. The message is encrypted with K_{AB} and sent to Alice.

Using Different Servers Note that if Alice needs to receive services from different servers, she need repeat only steps 3 to 6. The first two steps have verified Alice's identity and need not be repeated. Alice can ask TGS to issue tickets for multiple servers by repeating steps 3 to 6.

Realms Kerberos allows the global distribution of ASs and TGSs, with each system called a realm. A user may get a ticket for a local server or a remote server. In the second case, for example, Alice may ask her local TGS to issue a ticket that is accepted by a remote TGS. The local TGS can issue this ticket if the remote TGS is registered with the local one. Then Alice can use the remote TGS to access the remote real server.

Public-Key Distribution

In asymmetric-key cryptography, people do not need to know a symmetric shared key. If Alice wants to send a message to Bob, she only needs to know Bob's public key, which is open to the public and available to everyone. If Bob needs to send a message to Alice, he only needs to know Alice's public key, which is also known to everyone. In public-key cryptography, everyone shields a private key and advertises a public key.

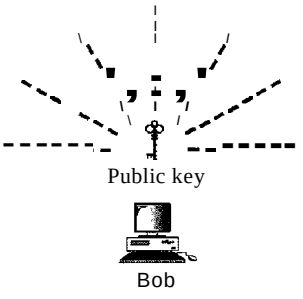
In public-key cryptography, everyone has access to everyone's public key; public keys are available to the public.

Public keys, like secret keys, need to be distributed to be useful. Let us briefly discuss the way public keys can be distributed.

Public Announcement

The naive approach is to announce public keys publicly. Bob can put his public key on his website or announce it in a local or national newspaper. When Alice needs to send a confidential message to Bob, she can obtain Bob's public key from his site or from the newspaper, or she can even send a message to ask for it. Figure 31.23 shows the situation.

Figure 31.23 *Announcing a public key*

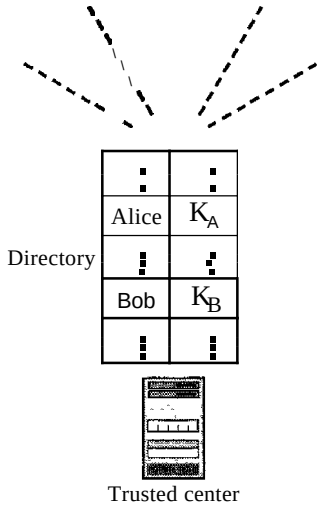


This approach, however, is not secure; it is subject to forgery. For example, Eve could make such a public announcement. Before Bob can react, damage could be done. Eve can fool Alice into sending her a message that is intended for Bob. Eve could also sign a document with a corresponding forged private key and make everyone believe it was signed by Bob. The approach is also vulnerable if Alice directly requests Bob's public key. Eve can intercept Bob's response and substitute her own forged public key for Bob's public key.

Trusted Center

A more secure approach is to have a trusted center retain a directory of public keys. The directory, like the one used in a telephone system, is dynamically updated. Each user can select a private/public key, keep the private key, and deliver the public key for insertion into the directory. The center requires that each user register in the center and prove his or her identity. The directory can be publicly advertised by the trusted center. The center can also respond to any inquiry about a public key. Figure 31.24 shows the concept.

Figure 31.24 *Trusted center*



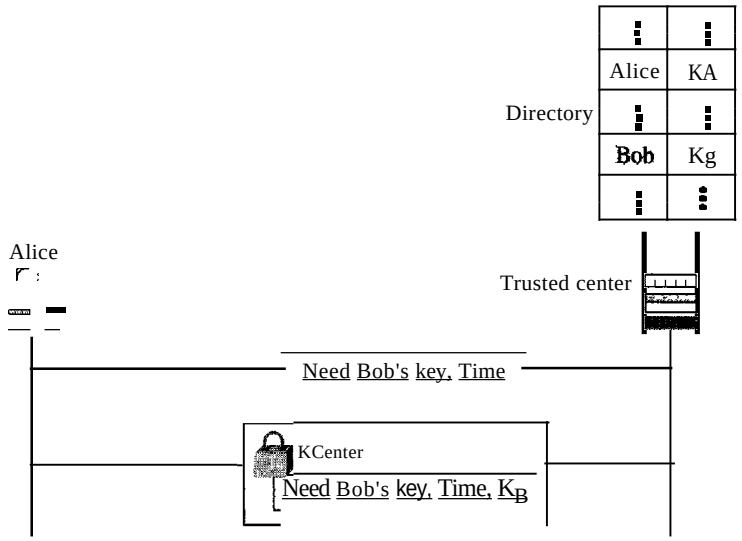
Controlled Trusted Center

A higher level of security can be achieved if there are added controls on the distribution of the public key. The public-key announcements can include a timestamp and be signed by an authority to prevent interception and modification of the response. If Alice needs to know Bob's public key, she can send a request to the center including Bob's name and a timestamp. The center responds with Bob's public key, the original request, and the timestamp signed with the private key of the center. Alice uses the public key of the center, known by all, to decrypt the message and extract Bob's public key. Figure 31.25 shows one scenario.

Certification Authority

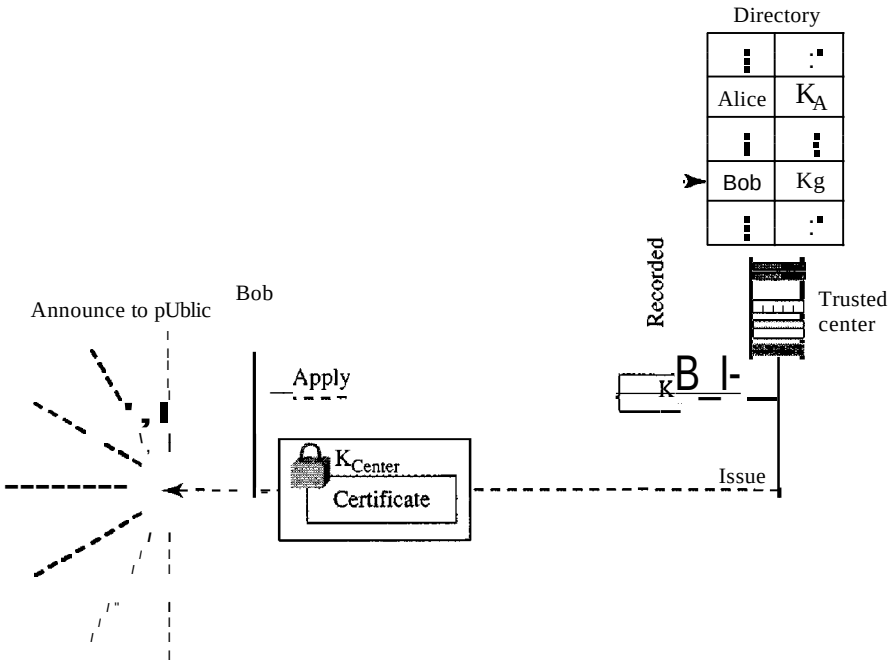
The previous approach can create a heavy load on the center if the number of requests is large. The alternative is to create public-key certificates. Bob wants two things: he wants people to know his public key, and he wants no one to accept a public key forged as his. Bob can go to a certification authority (CA)-a federal or state organization that binds a public key to an entity and issues a certificate. The CA has a well-known public key itself that cannot be forged. The CA checks Bob's identification (using a

Figure 31.25 *Controlled trusted center*



picture ID along with other proof). It then asks for Bob's public key and writes it on the certificate. To prevent the certificate itself from being forged, the CA signs the certificate with its private key. Now Bob can upload the signed certificate. Anyone who wants Bob's public key downloads the signed certificate and uses the public key of the center to extract Bob's public key. Figure 31.26 shows the concept.

Figure 31.26 *Certification authority*



X.509 Although the use of a CA has solved the problem of public-key fraud, it has created a side effect. Each certificate may have a different format. If Alice wants to use a program to automatically download different certificates and digests belonging to different people, the program may not be able to do so. One certificate may have the public key in one format and another in another format. The public key may be on the first line in one certificate and on the third line in another. Anything that needs to be used universally must have a universal format.

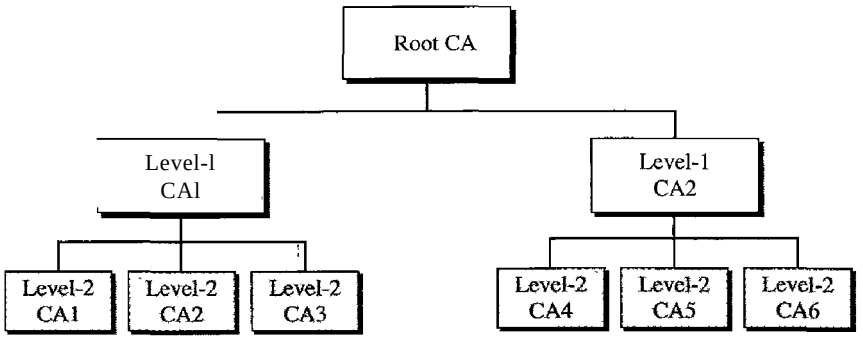
To remove this side effect, lTD has designed a protocol called X.509, which has been accepted by the Internet with some changes. X.509 is a way to describe the certificate in a structured way. It uses a well-known protocol called ASN.1 (Abstract Syntax Notation 1) that defines fields familiar to C programmers. The following lists the fields in a certificate.

- Version This field defines the version of X.509 of the certificate. The version number started at 0; the current version is 2 (the third version).
- Serial number This field defines a number assigned to each certificate. The value is unique for each certificate issued.
- Signature This field, for which the name is inappropriate, identifies the algorithm used to sign the certificate. Any parameter that is needed for the signature is also defined in this field.
- Issuer This field identifies the certification authority that issued the certificate. The name is normally a hierarchy of strings that defines a country, state, organization, department, and so on.
- Period of validity This field defines the earliest and the latest times the certificate is valid.
- Subject This field defines the entity to which the public key belongs. It is also a hierarchy of strings. Part of the field defines what is called the *common name,* which is the actual name of the beholder of the key.
- Subject's public key This field defines the subject's public key, the heart of the certificate. The field also defines the corresponding algorithm (RSA, for example) and its parameters.
- Issuer unique identifier This optional field allows two issuers to have the same *issuer* field value, if the *issuer unique identifiers* are different.
- Subject unique identifier This optional field allows two different subjects to have the same *subject* field value, if the *subject unique identifiers* are different.
- Extension This field allows issuers to add more private information to the certificate.
- Encrypted This field contains the algorithm identifier, a secure hash of the other fields, and a digital signature of that hash.

Public-Key Infrastructures (PKI)

When we want to use public keys universally, we have a problem similar to secret-key distribution. We found that we cannot have only one KDC to answer the queries. We need many servers. In addition, we found that the best solution is to put the servers in a hierarchical relationship with one another. Likewise, a solution to public-key queries is a hierarchical structure called a public-key infrastructure (PKI). Figure 31.27 shows an example of this hierarchy.

Figure 31.27 *PKI hierarchy*



At the first level, we can have a root CA that can certify the performance of CAs in the second level; these level-1 CAs may operate in a large geographic or logical area. The level-2 CAs may operate in smaller geographic areas.

In this hierarchy, everybody trusts the root. But people may or may not trust intermediate CAs. If Alice needs to get Bob's certificate, she may find a CA somewhere to issue the certificate. But Alice may not trust that CA. In a hierarchy Alice can ask the next-higher CA to certify the original CA. The inquiry may go all the way to the root.

31.8 RECOMMENDED READING

For more details about the subjects discussed in this chapter, we recommend the following books and sites. The items in brackets [...] refer to the reference list at the end of the text.

Books

Several books are dedicated to network security, such as [PHS02], [Bis03], and [SalO3].

31.9 KEY TERMS

authentication server (AS)
certification authority (CA)
challenge-response authentication
claimant
dictionary attack
digital signature
eavesdropping
entity authentication
fingerprint
fixed password
hash function
hashed message authentication code (HMAC)
identification
integrity
Kerberos
key distribution center (KDC)
message authentication
message authentication code (MAC)
message confidentiality or privacy
message digest
message integrity

message nonrepudiation
modification detection code (MDC)
nonce
nonrepudiation
one-time password
one-wayness
password
privacy
public-key infrastructure (PKI)
salting
session key
SHA-I
signature scheme
signing algorithm
strong collision
ticket
ticket-granting server (TGS)
verifier
verifying algorithm
weak collision
X.509

31.10 SUMMARY

- Cryptography can provide five services. Four of these are related to the message exchange between Alice and Bob. The fifth is related to the entity trying to access a system for using its resources.
- Message confidentiality means that the sender and the receiver expect privacy.
- Message integrity means that the data must arrive at the receiver exactly as sent.
- Message authentication means that the receiver is ensured that the message is coming from the intended sender, not an imposter.
- Nonrepudiation means that a sender must not be able to deny sending a message that he sent.
- Entity authentication means to prove the identity of the entity that tries to access the system's resources.
- A message digest can be used to preserve the integrity of a document or a message. A hash function creates a message digest out of a message.
- A hash function must meet three criteria: one-wayness, resistance to weak collision, and resistance to strong collision.
- A keyless message digest is used as a modification detection code (MDC). It guarantees the integrity of the message. To authenticate the data origin, one needs a message authentication code (MAC).
- MACs are keyed hash functions that create a compressed digest from the message added with the key. The method has the same basis as encryption algorithms.
- A digital signature scheme can provide the same services provided by a conventional signature. A conventional signature is included in the document; a digital signature is a separate entity.
- Digital signature provides message integrity, authentication, and nonrepudiation. Digital signature cannot provide confidentiality for the message. If confidentiality is needed, a cryptosystem must be applied over the scheme.

- A digital signature needs an asymmetric-key system.
- In entity authentication, a claimant proves her identity to the verifier by using one of the three kinds of witnesses: something known, something possessed, or something inherent.
- In password-based authentication, the claimant uses a string of characters as something she knows.
- Password-based authentication can be divided into two broad categories: fixed and one-time.
- In Challenge-response authentication, the claimant proves that she knows a secret without actually sending it.
- Challenge-response authentication can be divided into four categories: symmetric-key ciphers, keyed-hash functions, asymmetric-key ciphers, and digital signature.
- A key distribution center (KDC) is a trusted third party that assigns a symmetric key to two parties.
- KDC creates a secret key only between a member and the center. The secret key between members needs to be created as a session key when two members contact KDC.
- Kerberos is a popular session key creator protocol that requires an authentication server and a ticket-granting server.
- A certification authority (CA) is a federal or state organization that binds a public key to an entity and issues a certificate.
- A public-key infrastructure (PKI) is a hierarchical system to answer queries about key certification.

31.11 PRACTICE SET

Review Questions

1. What is a nonce?
2. What is the N^2 problem?
3. Name a protocol that uses a KDC for user authentication.
4. What is the purpose of the Kerberos authentication server?
5. What is the purpose of the Kerberos ticket-granting server?
6. What is the purpose of *X,S09?*
7. What is a certification authority?
8. What are some advantages and disadvantages of using long passwords?
9. We discussed fixed and one-time passwords as two extremes. What about frequently changed passwords? How do you think this scheme can be implemented? What are the advantages and disadvantages?
10. How can a system prevent a guessing attack on a password? How can a bank prevent PIN guessing if someone has found or stolen a bank card and tried to use it?

Exercises

11. A message is made of 10 numbers between 00 and 99. A hash algorithm creates a digest out of this message by adding all numbers modulo 100. The resulting digest is a number between 00 and 99. Does this algorithm meet the first criterion of a hash algorithm? Does it meet the second criterion? Does it meet the third criterion?
12. A message is made of 100 characters. A hash algorithm creates a digest out of this message by choosing characters 1, 11,21, ..., and 91. The resulting digest has 10 characters. Does this algorithm meet the first criterion of a hash algorithm? Does it meet the second criterion? Does it meet the third criterion?
13. A hash algorithm creates a digest of N bits. How many different digests can be created from this algorithm?
14. At a party, which is more probable, a person with a birthday on a particular day or two (or more) persons having the same birthday?
15. How is the solution to Exercise 14 related to the second and third criteria of a hashing function?
16. Which one is more feasible, a fixed-size digest or a variable-size digest? Explain your answer.
17. A message is 20,000 characters. We are using a digest of this message using SHA-l. After creating the digest, we decided to change the last 10 characters. Can we say how many bits in the digest will be changed?
18. Are the processes of creating a MAC and of signing a hash the same? What are the differences?
19. When a person uses a money machine to get cash, is this a message authentication, an entity authentication, or both?
20. Change Figure 31.14 to provide two-way authentication (Alice for Bob and Bob for Alice).
21. Change Figure 31.16 to provide two-way authentication (Alice for Bob and Bob for Alice).
22. Change Figure 31.17 to provide two-way authentication (Alice for Bob and Bob for Alice).
23. Change Figure 31.18 to provide two-way authentication (Alice for Bob and Bob for Alice).
24. In a university, a student needs to encrypt her password (with a unique symmetric key) before sending it when she logs in. Does encryption protect the university or the student? Explain your answer.
25. In Exercise 24, does it help if the student appends a timestamp to the password before encryption? Explain your answer.
26. In Exercise 24, does it help if a student has a list of passwords and uses a different one each time?
27. In Figure 31.20, what happens if KDC is down?
28. In Figure 31.21, what happens if the AS is down? What happens if the TGS is down? What happens if the main server is down?
29. In Figure 31.26, what happens if the trusted center is down?

30. Add a symmetric-key encryption/decryption layer to Figure 31.11 to provide privacy.
31. Add an asymmetric-key encryption/decryption layer to Figure 31.11 to provide privacy.

Research Activities

32. There is a hashing algorithm called MD5. Find the difference between this algorithm and SHA-l.
33. There is a hashing algorithm called RIPEMD-160. Find the difference between this algorithm and SHA-l.
34. Compare MD5, SHA-1, and RIPEMD-160.
35. Find some infonnation about RSA digital signature.
36. Find some information about DSS digital signature.