Lesson Proper for Week 9

LESSON 9: CRYPTOGRAPHY

Components of Cryptographic Protocols

Cryptography is the process of converting readable text, programs, and graphics into data that cannot be easily read or executed by unauthorized users. In other words, cryptography is the process of converting **plaintext** into **ciphertext** by using an encoding function such as an encryption algorithm or a secured list of substitution characters. The encrypted file can then be transmitted in many forms, including written messages and electronic data. Of course, the cryptographic process also includes providing a way for authorized users to access the plaintext. Cryptography helps achieve four critical goals of information security:

- Confidentiality of information—Only authorized users can access data.
- Integrity of data—Data has not been modified.
- Authentication—Users are who they claim to be.
- Nonrepudiation—Neither party can plausibly deny its participation in message exchanges. This is an important quality for business and legal transactions.

Cryptographic systems used in computer security contain common components, called cryptographic primitives that are combined into cryptographic protocols and standards. **Cryptographic primitives** are modular mathematical functions that include encryption algorithms, hashing functions, pseudorandom number generators, and basic logical functions.

The more detailed mathematical aspects of cryptographic design are beyond the scope of this book. Learning the basics of how these protocols are structured can help you determine what types of protocols best serve your needs and understand their vulnerabilities and deployment requirements.

Cryptographic Primitives

Used alone, cryptographic primitives cannot provide data integrity, confidentiality, nonrepudiation, and authentication. A primitive can accomplish only one of these goals. Each primitive is designed to perform a specific task reliably, such as generating a digital signature for a set of data. To provide adequate security, primitives must be used with other primitives.

In some ways, a cryptographic primitive is like a programming language. Software engineers do not create a new programming language for every new program. They use existing, proven languages in a modular fashion to create a software application that performs a defined set of functions. The same principle applies to cryptographic primitives. A cryptographic system designer does not create new primitives because they are complicated and prone to errors, even for experts.

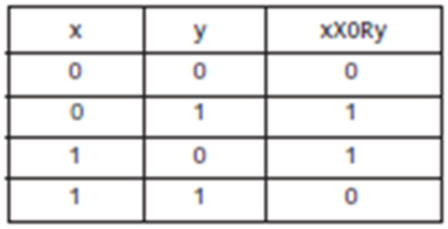
Primitives are not usually the source of security failures in cryptographic protocols. Security flaws result from mistakes in designing the protocol, such as an overall poor design, poorly chosen primitive combinations, or bugs introduced during the design process.

Each primitive in a cryptographic system handles one aspect of securing data. For example, the encryption algorithm performs encoding, thus providing message confidentiality but not message integrity or authentication. By combining the encryption algorithm with a hashing function, however, you can meet the requirement of message integrity. You can also add a digital signature for message authentication.

In the following sections, you examine cryptographic primitives that are common to modern cryptographic systems. From the most basic logical functions to more intricate encryption algorithms and key generation routines, you learn the fundamentals of how primitives work on their own and when combined with others.

Exclusive OR Function

The exclusive OR (XOR) function is used in cryptography as a linear mixing function to combine values. For example, the output of other primitive ciphers can be combined with an XOR function to produce a pseudorandom value on which another cipher performs additional operations. An XOR function is based on binary bit logic and results in a logical value of true if only one of the operands has a value of true. So, for example, if x and y are the same (both true or both false), the XOR output is 0 (false). If x and y are different, the XOR output is 1 (true). The truth table in Figure 5-1 shows inputs of x and y and the result of the XOR function in the right column.



x	y	xXORy
0	0	0
0	1	1
1	0	1
1	1	0

Figure 5-1 An XOR truth table

The XOR function is useful as a cryptographic primitive because of its reversible property, as shown in the following example:

p XOR k = c

c XOR k = p

In this equation, p represents plaintext, k is the key, and c is ciphertext. The c resulting from p XOR k is reversible in the c XOR k statement.

Permutation Functions

Bit-shuffling permutation functions, often used in symmetric algorithms, reorder sets of objects randomly—for example, by rearranging input bits, such as the binary input 010 into 001. A good analogy for a random permutation is shuffling a deck of cards, which ideally causes the deck to be dealt in a completely random order. Some variations of permutation functions are suitable for cryptographic use, but many are not. One cryptographically notable variation is an expansion permutation, in which certain bits are used more than once. For example, the input 010 is rearranged and expanded into 0101.

Substitution Box Functions

A substitution box (S-box) function transforms a number of input bits into a number of output bits and produces a lookup table that can be fixed or dynamic, depending on the cipher. It is a basic component of symmetric key algorithms. The purpose of this function, as with permutation functions, is disguising the relationship of ciphertext to cleartext. An S-box function is usually described as n input bits × m output bits, so a 6×4 S-box means that 6 input bits are transformed into 4 output bits.

Feistel Networks

Created in 1973, a Feistel network (shown in Figure 5-2) is a symmetric block cipher that is the basis of several symmetric encryption algorithms. A Feistel network's purpose is to obscure the relationship between ciphertext and keys (a shortcoming of symmetric algorithms). It does this by combining multiple rounds of repeated operations, such as processing cleartext input with XOR functions. A key schedule is used to produce different keys for each round. The advantage of a Feistel network is that its encryption and decryption operations are similar or even identical, which reduces the size of its code and the resources needed to use it. Feistel encryption works because the key schedule can be reversed, using keys in exact reverse order.

Pseudorandom Number Generators

Pseudorandom number generators (PRNGs) are essential components of a cryptographic algorithm. A PRNG is an algorithm for generating sequences of numbers that approximate random values. To be considered a cryptographically secure pseudorandom number generator (CSPRNG), a function must meet certain design principles and be resistant to known attacks. Many cryptographic functions require random values that serve as seeds for further computation:

· Nonces—A nonce is a number or bit string (usually random) that prevents generation of the same ciphertext during subsequent encryptions of a message. Using nonces strengthens encryption and makes it more resistant to being broken.

· One-way functions—A one-way function is easy to compute but difficult and time consuming to reverse. One-way functions include integer factorization, discrete logarithms, and the Rabin function. They are considered one-way because no efficient inverting algorithm has been discovered, but new methods could prove these functions reversible.

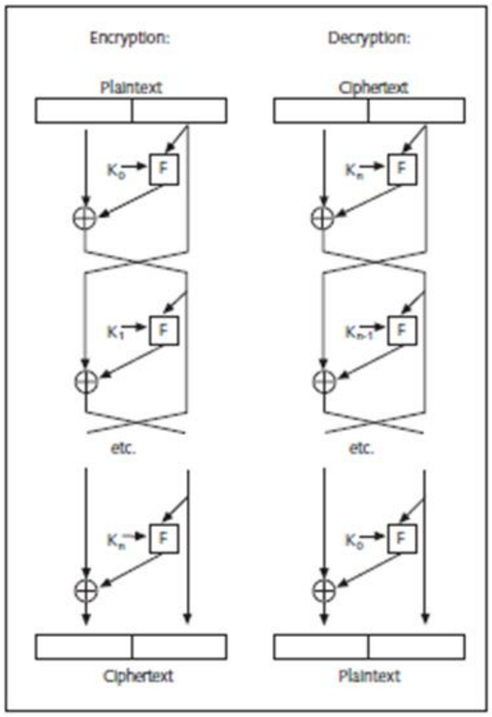


Figure 5-2 A Feistel network

· Salts—A salt consists of random bits used as input for key derivation functions and to pad values (hide the true contents). Initialization vectors, passwords, and passphrases are sometimes used as salts.

· Key derivation (generation)—A key derivation function generates secret keys from a secret value (usually a randomly generated value) and another piece of information such as a password.

PRNGs can be hardware or software based. In general, hardware PRNGs offer sequences that are closer to being truly random, but the hardware can be expensive and cumbersome and is vulnerable to the same threats as any hardware security device. Several algorithms can produce random number sequences of sufficient quality for cryptographic purposes, making hardware PRNGs unjustifiably expensive in most cases.

PRNG values are not completely random because they are generated by a structured method based on fairly small initialization values called the PRNG's state. The state is generally measured in bits, and the initial state's size determines the maximum sequence length before it repeats; this is known as the PRNG's period. In modern computing, it is possible to provide a long enough state to ensure that the PRNG's period is longer than any computer could compute in a reasonable amount of time—say, 100 million years. If one bit is produced every picosecond, the PRNG's period with a 64-bit state is more than 140 million years.

PRNGs still have some problems, even if they are considered cryptographically secure. One problem is that by providing a starting point for random number generation, the resulting value is not truly random. Another problem is that the PRNG always produces the same values when initialized with a particular state. To be secure, mechanisms are necessary to prevent a PRNG from reusing an initializing state.

CSPRNGs produce values that are random enough to suit the intended use. In practice, there is no certain way to determine the output of a CSPRNG from truly random numbers without knowing the algorithm used and its initialization state. Because distinguishing between truly random and generated pseudorandom numbers is not considered feasible, most algorithms and protocols that use CSPRNGs are considered secure.

Hashing Functions

One method of verifying message integrity is by using **hashing functions**, which generate a hash value or message digest from input. (Hashing algorithms, as you will learn later in this chapter, define the instructions for running hashing functions.) A hash value is a fixed-size string representing the original input's contents. If the input changes in any way, even by adding a period at the end of a sentence, the resulting output has a different hash value.

For messages sent over the Internet, verifying message integrity and authenticating the source are critical. Source authentication verifies the sender's identity and prevents messages from fraudulent or spoofed sources from being accepted. Using a hashing function to verify message integrity is as simple as comparing the message digest the sender calculates with the message digest the receiver calculates. If the values are the same, the sender's message has not been altered during transmission.

Hashing functions are also used for error detection, as with Cyclic Redundancy Check (CRC), a commonly used method of verifying that a message was not altered by a transmission error, such as interference on the transmission medium.

Encryption Algorithms

To see how cryptographic primitives are combined to provide confidentiality, integrity, authentication, and nonrepudiation, understanding the basic mathematical concepts is helpful. An algorithm is a precisely defined set of instructions for carrying out a task. Computer algorithms, for example, provide exact instructions for which operations to carry out, which criteria change operations, how many times to perform an operation (called looping), and when to stop.

In algorithms, a strict order of operations is essential; in computer programs, this strict order of operations is called control flow. If a program processes the same input for the same purpose, but the instructions vary, the results would never match, nor would they be repeatable, reversible, and predictable. Output values change, of course, if the input varies, but the instructions for processing input must stay the same. For example, the instruction “Find the largest item in the list” generates different values depending on the list’s contents, but must always process the list exactly the same way to achieve consistency.

An **encryption algorithm** is a set of precise instructions that provides an encoding function for a cryptographic system or generates output for use in additional operations. (Remember that some algorithms also double as CSPRNGs, such as stream and block ciphers.) In the simplest applications, an encryption algorithm is a mathematical formula that works with a key to generate ciphertext from cleartext input. Encryption algorithms also combine with other primitives that perform integrity checking or authentication. A hashing function, for example, can be used to check data integrity or to generate pseudorandom numbers that the encryption algorithm can use in encoding iterations. This process is repeatable and reversible.

Key Size in Encryption Algorithms

An encryption algorithm’s strength is often tied to its key length. The longer the key, the harder it is to break the encryption. Longer keys offer more protection against brute-force attacks, in which every possible key is tried to decrypt a message. As computers have increased in processing power, however, attackers have been able to carry out brute-force attacks more quickly and break encryption keys. Therefore, to keep up with advances in processors, encryption key sizes had to increase. As an illustration of the power of extending key size, consider a five-character password created from only lowercase letters. Such a password would offer almost 12 million possible combinations (265). However, extending the password to 10 characters would increase the number of possible combinations to more than 141 trillion (2610).

Types of Encryption Algorithms

The two major types of encryption algorithms are block ciphers and stream ciphers. A **block cipher** encrypts groups of text at a time. For example, a block cipher encrypts the whole word cat instead of encrypting each letter. A **stream cipher** encrypts cleartext one bit at a time to produce a stream of encrypted ciphertext, so the letters c, a, and t in cat are encrypted separately.

Block ciphers and stream ciphers use keys differently, so they fall into two categories: symmetric and asymmetric. **Symmetric algorithms** use the same key to encrypt and decrypt a message. They are considered the workhorses of the encryption world and are a faster, more efficient method of encrypting data because they require fewer computing resources. Most encryption protocols in daily use are based on symmetric algorithms, and the majority of symmetric algorithms are block ciphers.

By contrast, **asymmetric algorithms** use a specially generated key pair. One key encrypts cleartext into ciphertext, and the other key decrypts ciphertext into cleartext. Either of the generated pair can be used to encrypt, but the other key must be used to decrypt. Asymmetric encryption and decryption are about 10,000 times slower than symmetric encryption.

Blowfish

Blowfish is a 64-bit block cipher composed of a 16-round Feistel network and key-dependent S-box functions. This unpatented cipher used worldwide has a variable key size from 32 to 448 bits. The default key size is 128 bits. Blowfish is fast in encryption and decryption operations, but its 64-bit block size is now considered too short and makes Blowfish vulnerable to some attacks.

Notable for its public license status and excellent performance, Blowfish is still a widely used cipher, and is very fast except when changing keys. Its high memory requirements (4 kilobytes) and slow key derivation functions are not a problem for most desktop and laptop computers (even older ones), but these characteristics make it unsuitable for smart cards or similar limited-resource systems.

Twofish

Twofish, the successor to Blowfish, is a 128-bit symmetric block cipher composed of a 16-round Feistel network and key-dependent S-box functions. Twofish also has a complicated key schedule and a variable key size of 128, 192, or 256 bits. Like Blowfish, it is publicly licensed, but it has not been used as much as its predecessor. Although some theoretical work on cryptanalysis methods against Twofish has been published, it has not been broken.

Rivest Cipher Family

Rivest Cipher 4 (RC4) is a popular stream cipher in Web browsers that use Secure Sockets Layer (SSL), Wired Equivalent Privacy (WEP), Wi-Fi Protected Access (WPA), and Transport Layer Security (TLS). RC4 uses an XOR function to combine a pseudorandomly generated stream of bits (the keystream) with the plaintext and produce ciphertext. To generate the keystream, the cipher uses a secret internal state composed of two parts: a permutation of all 256 possible bytes and two 8-bit index pointers. The permutation is initialized with a variable-length key, and then a pseudorandom stream of bits is generated with a PRNG. RC4 is not recommended for new applications because of its weak use of keys and lack of nonces, but it is still used because of its speed and simplicity.

In addition to RC4, Rivest created other ciphers in the RC family. Notably, RC6, a block cipher, was developed for the Advanced Encryption Standard competition and selected as a finalist. It was not the winning algorithm, but it is still available for applications that require encoding functions. It is not free, however, because it is a patented algorithm.

Rijndael

Rijndael (pronounced like raindoll) is the encryption algorithm incorporated into the Advanced Encryption Standard (AES). It is a block cipher composed of 10 to 14 rounds of S-box and XOR functions, but it does not use a Feistel network. This symmetric algorithm specifies how to use 128-bit, 192-bit, or 256-bit keys on 128-bit, 192-bit, or 256-bit blocks. It applies 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. The standard AES implementation of Rijndael has a fixed block size of 128.

The name Rijndael comes from the names of its creators, the prominent cryptographers Vincent Rijmen and Joan Daemen. The names Rijndael and AES are used interchangeably, but they are not the same. Rijndael is the algorithm and AES is the implementation.

Rivest, Shamir, Adelman

Rivest, Shamir, and Adelman developed RSA (the first letter of each creator’s surname) for public key encryption. RSA uses a public key that is freely shared and a private key that is kept secret. If RSA is used with long enough keys and is kept updated, it is believed to be secure. RSA is widely used in e-commerce protocols and is the default encryption and signing scheme for X.509 certificates.

RSA keys are usually 1024 to 2048 bits, although larger and smaller keys are supported. Keys that are smaller than 512 bits can be broken quickly with modern computing resources. Even the security of 1024-bit keys has been called into question since 2003. Therefore, experts recommend keys that are 2048 bits or longer. As with any asymmetrical encryption system, key exchange is a challenge. Key distribution must be protected against man-in-the-middle attacks, typically by using Public-key Infrastructure (PKI) components, such as digital certificates, which are signed and can be authenticated with hashing functions.

RSA is also vulnerable to timing attacks, adaptive chosen ciphertext attacks, and branch prediction analysis. Timing attacks are a type of side channel attack, which is discussed later in this chapter. Adaptive chosen ciphertext attacks exploit flaws in the public key cryptography standard (PKCS) scheme to recover session keys; using newer PKCS versions with secure padding schemes is recommended. (You learn about PKCS later in this chapter.) Branch prediction analysis takes advantage of processors that use a predictor to determine whether a conditional branch is likely to be taken in a program’s control flow. The attack uses a spy process to determine the private key statistically.

Hashing Algorithms

Hashing algorithms are sets of instructions applied to variable-length input (the message) that generate a fixed-length message digest representing the input. The message digest is used for comparison to ensure message integrity. Hashing algorithms do not provide confidentiality because they do not encrypt the message contents in a way that can be decrypted, but they do provide verification that a message has not been altered. Remember that hashing algorithms are mathematical formulas, and hashing functions are the process the computer uses to generate a hash value.

When a message with a message digest is received, the hashing algorithm is run against the contents again. If the values match, the message is considered unaltered. If the values do not match, the message might have been tampered with or corrupted during transmission. Several hashing algorithms are available, but the most common are Message Digest 5 and Secure Hash Algorithm. Both are discussed in the following sections.

Message Digest 5

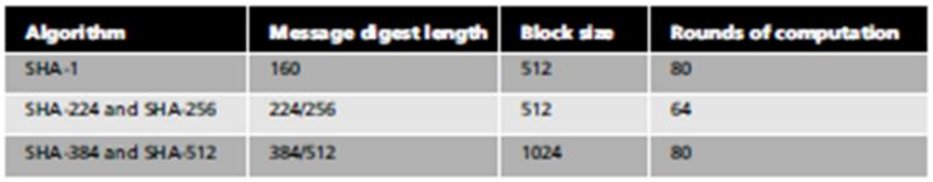
Ronald Rivest devised Message Digest 5 (MD5) in 1991 as a replacement for MD4, which was not secure. It is used in many cryptographic applications, such as digital signatures and virtual private networks (VPNs). MD5 makes only one pass on data and generates a 128-bit hash value that is displayed as a 32-character hexadecimal number.

Methods have emerged that make it easy to generate collisions in MD5. A collision occurs when computing the MD5 algorithm with two different initialization vectors produces the same hash value. For a hashing algorithm to be considered secure, three conditions must be true: No hash should be usable to determine the original input, no hashing algorithm should be run on the same input and produce different hashes, and a hashing algorithm should not be run on two different inputs and produce the same hash (collision). Collisions can be used to determine the plaintext, and because MD5 is commonly used for password storage, the possibility of deciphering plaintext is a serious security risk. Another risk with MD5 involves the use of rainbow tables: precompiled lookup tables of possible hash-plaintext combinations that are posted on the Internet and that make successful brute-force attacks more likely. They can also be used to generate collisions and make password cracking easier. If one plaintext string generates the same hash value as another, the attacker can use the known information to determine the unknown values. MD5's reputation suffered in 2008 when security researchers demonstrated that, by exploiting MD5 collisions, they could create rogue digital certificates— the electronic documents upon which secure e-commerce is based.

Secure Hash Algorithm

The National Security Agency designed Secure Hash Algorithm (SHA) as a successor to MD5, and it is approved for federal government use. SHA version 1 (SHA-1) is used in many cryptographic applications that require checking message integrity, including SSL, SSH, and IPsec.

The SHA standards include five algorithms: SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. SHA-1 is the most commonly used algorithm, and the other four are sometimes referred to collectively as SHA-2. SHA-1 generates a message digest of 160 bits, and the number added to the names of the other four algorithms denotes the message digest's length. Table 5-1 summarizes SHA message digest lengths and block sizes, and how many rounds of computation are performed.



Algorithm	Message digest length	Block size	Rounds of computation
SHA-1	160	512	80
SHA-224 and SHA-256	224/256	512	64
SHA-384 and SHA-512	384/512	1024	80

Table 5-1 Summary of SHA algorithms

Methods have been devised to decipher a message’s original text based on hash values. The

U.S. government became so concerned about SHA-1’s vulnerabilities that it was phased out in favor of SHA-2 in 2010. As this book was going to press, a competition was being held to choose the new algorithm that would become SHA-3. However, although the effects of SHA-1 attacks in a lab environment are severe, no significant attack methods outside the lab environment have proved successful in compromising SHA-1.

Until SHA-3 is selected, most cryptographic applications continue to use SHA-1 and sometimes MD5, but storing passwords is still a major concern. In some systems, the password is not saved in the file system; instead, a hashed value of the password is stored. When the system carries out password validation for authentication, it takes user input and generates a hash value to compare with the stored value.

One way to reduce the vulnerabilities of hashing algorithms is to add a salt to the plaintext before hashing. Another method called key strengthening involves applying the hashing function more than once. No method of reinforcement is completely secure, but these methods can help improve the security of MD5 and SHA-1.

Message Authentication Code

Message Authentication Code (MAC), also known as Message Integrity Check (MIC), uses a shared secret key that is agreed on by the sender and receiver in the verification process to generate a MAC tag for a message. A MAC tag is like an enhanced message digest. The shared secret key adds a measure of security to the hashing algorithm.

Do not confuse this abbreviation with Media Access Control (MAC), a common abbreviation in networking documentation.

The message and MAC tag are sent to the receiver. The key is also sent to the receiver securely; this key is usually sent separately from the message. The receiver goes through the same process of using the transmitted message and key to generate a MAC tag, and compares this tag with the one received in the message to confirm the message’s integrity and authenticity. The verification process is protected by secure communication of the key, which ensures that the sender and receiver generate the same MAC tag from the message. Figure 5-3 illustrates this process.

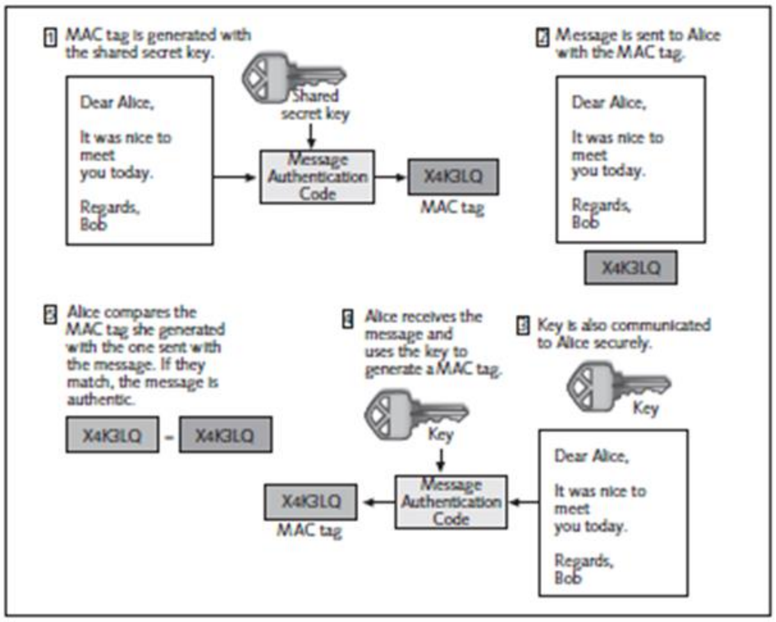

1 MAC tag is generated with the shared secret key.
Shared secret key
Dear Alice, It was nice to meet you today. Regards, Bob
Message Authentication Code
X4K3LQ
MAC tag
2 Message is sent to Alice with the MAC tag.
Dear Alice, It was nice to meet you today. Regards, Bob
X4K3LQ
3 Key is also communicated to Alice securely.
Key
4 Alice receives the message and uses the key to generate a MAC tag.
Key
Message Authentication Code
X4K3LQ
MAC tag
5 Alice compares the MAC tag she generated with the one sent with the message. If they match, the message is authentic.
X4K3LQ = X4K3LQ

Figure 5-3 The MAC process

As with symmetric cryptography, MAC uses a single key to verify message integrity, so the challenge is key management—how to communicate the secret key that the sender and receiver use securely. If communication of the secret key is compromised, an attacker could forge a message between the sender and receiver without MAC detecting it. MAC can still be used, as long as care is taken to secure communication of the secret key with encryption.

Digital Signatures

Digital signatures use hashing algorithms with asymmetric encryption to produce a method for verifying message integrity and nonrepudiation. Nonrepudiation means ensuring that participants in a message exchange cannot deny their roles in the process. This process is explained in the following example and illustrated in Figure 5-4.

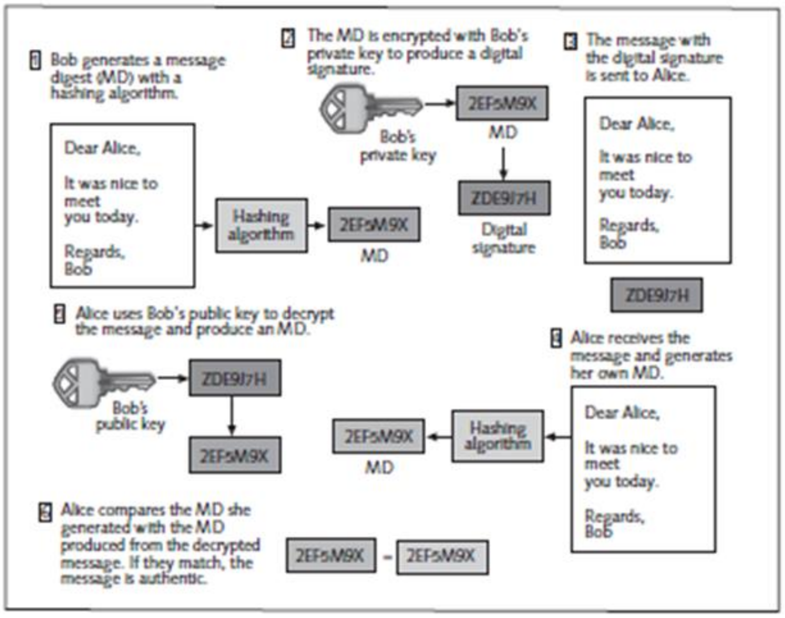

1 Bob generates a message digest (MD) with a hashing algorithm.
Dear Alice, It was nice to meet you today. Regards, Bob
Hashing algorithm
2EF5M9X
MD
2 The MD is encrypted with Bob's private key to produce a digital signature.
Bob's private key
2EF5M9X
MD
ZDE9I7H
Digital signature
3 The message with the digital signature is sent to Alice.
Dear Alice, It was nice to meet you today. Regards, Bob
ZDE9I7H
4 Alice receives the message and generates her own MD.
Dear Alice, It was nice to meet you today. Regards, Bob
Hashing algorithm
2EF5M9X
MD
5 Alice uses Bob's public key to decrypt the message and produce an MD.
Bob's public key
ZDE9I7H
2EF5M9X
6 Alice compares the MD she generated with the MD produced from the decrypted message. If they match, the message is authentic.
2EF5M9X = 2EF5M9X

Figure 5-4 The digital signature process

Bob wants to send Alice a digitally signed message. He goes to the organization's directory to find Alice's public key certificate. The process works as follows:

1. Using a hashing algorithm, a message digest of Bob's message to Alice is calculated.

2. The message digest is encrypted by Bob's private key. The resulting ciphertext is the digital signature of the message.

3. The digital signature with the message is sent to Alice.

4. Alice runs the message she receives through the same hashing algorithm Bob used to get a message digest.

5. Alice then decrypts the message's digital signature with Bob's public key, which produces the message digest calculated in Step 1.

6. Alice compares the message digest she calculated with the message digest Bob calculated to verify the message's integrity. If the message digests are the same, the message is authentic.

Or is it? Alice had to obtain Bob's public key; how can she be sure that the key actually came from Bob? Both Alice and Bob have to trust the source of the public keys.

Digital signature security vulnerabilities are mostly associated with the IT infrastructure required to support interoperability. When a sender digitally signs a message, the receiver must have access to and trust the same certification authority (CA) that issued the sender's credentials. Therefore, Internet users cannot rely on digital signatures because not all users have the configuration to trust issuing CAs. Interoperability and political issues with recognized CAs among organizations and countries are a barrier to universal acceptance of digitally signed documents and messages. Cases are being tried worldwide to determine whether digital signatures can be recognized as a legally valid method for signing documents.

However, if Bob and Alice both trust the CA, nonrepudiation is achieved in this process because the signature is encrypted by Bob's private key. Bob is the only owner of this key and the only person with access to it. The successful decryption by Bob's public key confirms his identity.

Key Management

The major problem with cryptographic algorithms is secure key exchange. To prevent the compromise of encrypted traffic, cryptographic systems change keys frequently and distribute them to all authorized parties. This process of changing and distributing keys is called key management, and is difficult to carry out reliably.

Private Key Exchange

Private key exchange uses a symmetric cryptographic algorithm in the encryption process, in which the same key (also called a "shared key" or a "shared secret") is used to encrypt and decrypt a message. Therefore, the message is only as secure as the shared key. The following is an example of a private key exchange, as shown in Figure 5-5.

1. Bob has a cleartext message for Alice. He uses a shared key to encrypt the message into ciphertext to protect it from unintended readers.

2. Bob sends the encrypted message to Alice.

3. For Alice to be able to read the message, she needs to use the shared key to decrypt the ciphertext. This key must be sent to Alice securely and separately to ensure the message's confidentiality. Sending the key with the message or even on the same unsecure medium (in-band) defeats the purpose of encryption.

4. Alice receives the message and uses the shared key she received from Bob to decrypt it.

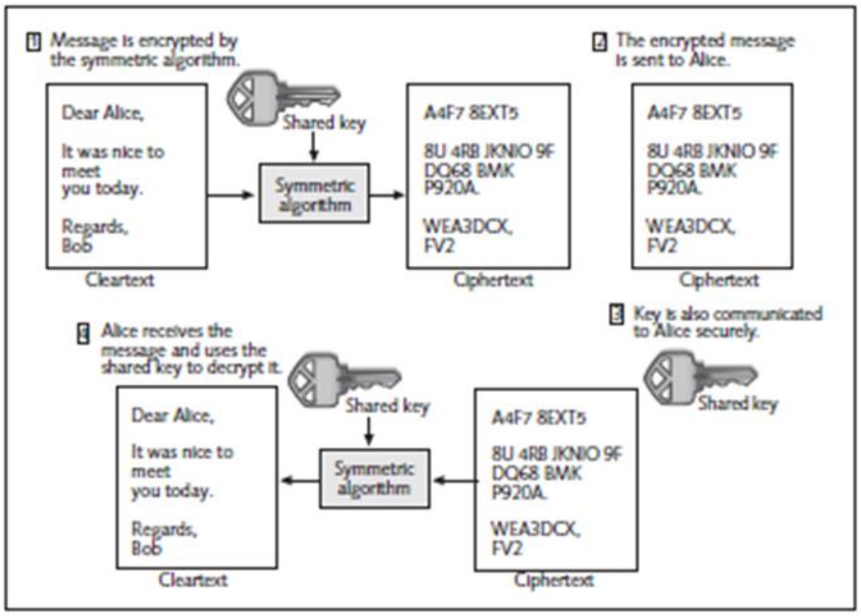

1 Message is encrypted by the symmetric algorithm.
Dear Alice, It was nice to meet you today. Regards, Bob
Cleartext
Shared key
Symmetric algorithm
A4F7 8EXT5 8U 4RB JKNIO 9F DQ68 BMK P920A. WEA3DCX, FV2
Ciphertext
2 The encrypted message is sent to Alice.
A4F7 8EXT5 8U 4RB JKNIO 9F DQ68 BMK P920A. WEA3DCX, FV2
Ciphertext
3 Key is also communicated to Alice securely.
Shared key
4 Alice receives the message and uses the shared key to decrypt it.
Shared key
Symmetric algorithm
Dear Alice, It was nice to meet you today. Regards, Bob
Cleartext
A4F7 8EXT5 8U 4RB JKNIO 9F DQ68 BMK P920A. WEA3DCX, FV2
Ciphertext

Figure 5-5 The private key exchange process

Public Key Exchange

Public key exchange uses asymmetric cryptography in the encryption process and generates a key pair: Anything encrypted by one key can be decrypted only by the other member of the pair, and vice versa. To achieve key management when a key pair is generated, one is labeled as the public key and the other as the private key. The public key is freely shared—anyone who intercepts it would be able to encrypt messages that the holder of the private key could decrypt. The private key is never shared and is kept secure.

When an encrypted message is sent to the private key owner, the sender encrypts the message with the recipient's public key. The recipient then uses the private key to decrypt the message. Confidentiality is ensured because the private key owner is the only person who can decrypt what the public key has encrypted. Encrypting and decrypting the message with the public key is not possible.

Many computer users use public key exchange in the form of SSL / TLS Web browser encryption and often do not realize it. Typically, asymmetric encryption is not used to provide data confidentiality for the substance of the messages. Because symmetric encryption is so much faster, the two sides use asymmetric encryption to negotiate a shared key securely and then switch to symmetric encryption using the shared key.

The IT infrastructure to support asymmetric cryptography and public key exchange is more complicated than symmetric cryptography and private key exchange. The following list describes components of an asymmetric cryptography system:

· Certificates—A certificate is a file that contains information about the user, service, or business entity and the assigned public key. It also contains information about the certification authority and its digital signatures so that users can trust the public key's authenticity.

· Certification authorities (CAs)—These organizations issue public and private key pairs to people, services, and organizations. A CA keeps track of issued credentials and manages the revocation of certificates, if needed. A CA also verifies that the public and private keys have been issued legitimately and are trustworthy.

· Registration authorities (RAs)—Also called registrars, RAs serve as a front end to users for registering, issuing, and revoking certificates. For security reasons, users rarely contact CAs. Instead, they interact with an RA, which acts as an intermediary in the certificate-issuing process. Before an RA issues public and private keys to a user, identity verification is required to ensure that the certificate reflects the user's identity accurately.

· Certificate revocation lists (CRLs)—CAs track and publish listings of invalid certificates. CRLs should be checked to make sure that certificates are legitimate; usually applications perform the check automatically unless users have disabled this feature. If a certificate is listed on a CRL, the cryptography system gives users a warning that the certificate is not valid. Certificates can be considered invalid for many reasons. For example, a user might have been issued a new certificate or taken a new job, so the older certificate needs to be taken out of service. Also, users might no longer be affiliated with the issuing CA because they have been fired.

· Message digests—The recipient's message software compares the received message's hash value with the transmitted message's hash value to verify that the message is unchanged.

Putting all these components together, the following process takes place when issuing a certificate for use in public key exchange (see Figure 5-6).

1. Alice applies to an RA for a certificate. The RA works with the CA to issue certificate credentials that contain the public and private keys. In this process, identity verification is carried out via the organization's standard operating procedure for issuing certificates. A copy of Alice's public key certificate is kept in the organization's directory along with issued certificates.

2. Bob wants to send Alice a message. He goes to the organization's directory to get a copy of her public key certificate. Each time the certificate is used, the CA is consulted to make sure the certificate is not listed on the CRL.

3. Bob uses Alice's public key certificate to encrypt the message and then sends the encrypted text to Alice.

4. Alice receives the message and then uses her private key to decrypt it.

Although asymmetric cryptography is commonly used for communicating symmetric keys safely and for security on the Internet, using asymmetric systems for everyday encryption of e-mail and digital signatures has met with mixed success. Asymmetric encryption systems, also known as Public-key Infrastructure (PKI), have been used in the U.S. Department of Defense and large corporations, but the complexity and cost of PKI systems have discouraged many organizations from adopting this technology.

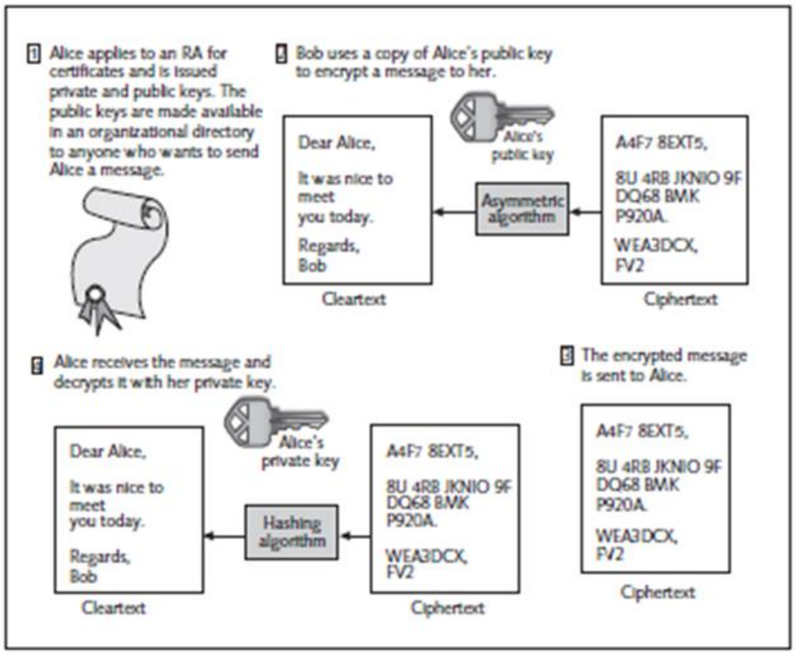

1 Alice applies to an RA for certificates and is issued private and public keys. The public keys are made available in an organizational directory to anyone who wants to send Alice a message.
2 Bob uses a copy of Alice's public key to encrypt a message to her.
Alice's public key
Dear Alice, It was nice to meet you today. Regards, Bob
Cleartext
Asymmetric algorithm
A4F7 8EXT5, 8U 4RB JKNIO 9F DQ68 BMK P920A. WEA3DCX, FV2
Ciphertext
3 The encrypted message is sent to Alice.
4 Alice receives the message and decrypts it with her private key.
Alice's private key
Hashing algorithm

Figure 5-6 The public key exchange process

Public Key Cryptography Standards

Created by RSA Labs to improve interoperability in public key cryptography, public key cryptography standards (PKCSs) are not actual industry standards, but they have helped move modern information security cryptography— PKI in particular—toward standardization. Several PKCS designations have even moved into the standards track through the IETF (Internet Engineering Task Force), and others are used as de facto standards. Some PKCS designations have been withdrawn or made obsolete; for example, PKCS #2 and #4 were withdrawn, and PKCS #6 was made obsolete by a newer version of X.509. PKCS #13, Elliptic Curve Cryptography Standard, and #14, Pseudorandom Number Generation, are under development.

X.509

X.509 is an International Telecommunication Union standard for PKI developed in 1998 by the IETF's Public-Key Infrastructure Working Group. X.509 specifies standard formats for public key certificates, a strict hierarchical system for CAs issuing certificates, and standards for CRLs. X.509 certificates use RSA for key generation and encryption and, more recently, SHA-1 hashes to verify the certificate's integrity.