2 Digital Cryptography Basics

This chapter introduces the basic building blocks of modern digital cryptography, describes what they can do for you, and explains how to compose them to produce useful and practical secure services.

If you are familiar with these basics, you can probably skip this chapter. Good general references in this area are [NetSec] and [Schneier]. This chapter—and indeed this book—do not generally get down to the details of bit-level formats or algorithms. That is, they do not provide enough information for you to implement the cryptographic algorithms mentioned. Nevertheless, this chapter should provide a foundation of knowledge about modern digital cryptography that will make later chapters more comprehensible.

Sections 2.1, 2.2, and 2.3 generally discuss "symmetric" functions where the originator and receiver compute the same quantity or use the same secret key. Sections 2.4, 2.5, and 2.6 focus on "asymmetric" functions involving a public-private key pair. The remaining sections (2.7–2.11) discuss a variety of other important aspects of basic digital cryptography.

2.1 Message Digests

Message digest functions convert sequences of bits, possibly quite long, called messages, into fixed-length binary "fingerprints" or message digests of the original sequences. See Figure 2-1. A message digest function has two goals:

- It should be computationally infeasible to find another message whose digest is the same as the digest of a given message.
- It should be computationally infeasible to find two arbitrary messages whose digests are the same.

In the common case where an authentication method takes a large amount of computational effort and that effort is proportional to the number of bits

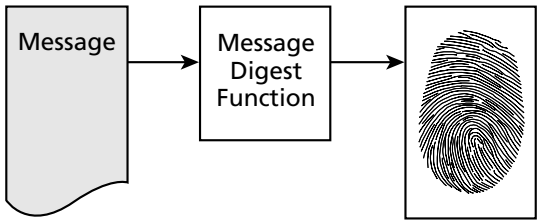


Figure 2-1 | Message digest function

being authenticated, you can secure a large document by authenticating its much smaller fixed-size message digest.

A message digest function is not identical to a checksum. A checksum is usually quite simple and is designed to detect transmission errors or accidental changes. An adversary can deliberately circumvent the testing of a checksum by adjusting the message to leave the checksum unchanged. By comparison, a message digest is complex and is designed to defeat attempts by an adversary to change the message.

First, consider a checksum calculated by simply adding all octets in a message and discarding the bits of the sum above the least significant eight bits. In most cases, an adversary could easily modify the message to become a different message with the same checksum. For example, inserting an octet with value V into the message will have no effect on the checksum if you can also insert a second octet with value $(256 - V)$ anywhere in the message.

Next, consider a more complex function in which you take the product of the octets in the message, adding to each octet its position in the message, and the check octet is the middle eight bits of this product. For example, if the message consists of bytes with values

$$V_1, V_2, V_3, \ldots$$

then the check octet is the middle eight bits of

$$(V_1 + 1) * (V_2 + 2) * (V_3 + 3) \ldots$$

With this level of complexity, it is no longer quite so trivial for an adversary to figure out how to change the message without changing the check byte, although it can still be done. Cryptographically or computationally secure message digest functions are substantially more complex than this example, producing check quantities of at least 128 bits or 16 eight-bit bytes. They largely meet the following goals.

Expressing the goals of message digests more formally, if the message digest is N bits, then

1. To find a second message with the same message digest as a given message, no method should require an expected effort significantly less than trying 2^{N-1} possible other messages;
2. To find two arbitrary messages with the same message digest, no method should require an expected effort significantly less than trying $2^{N/2}$ messages, remembering all their digests, and looking for a match.

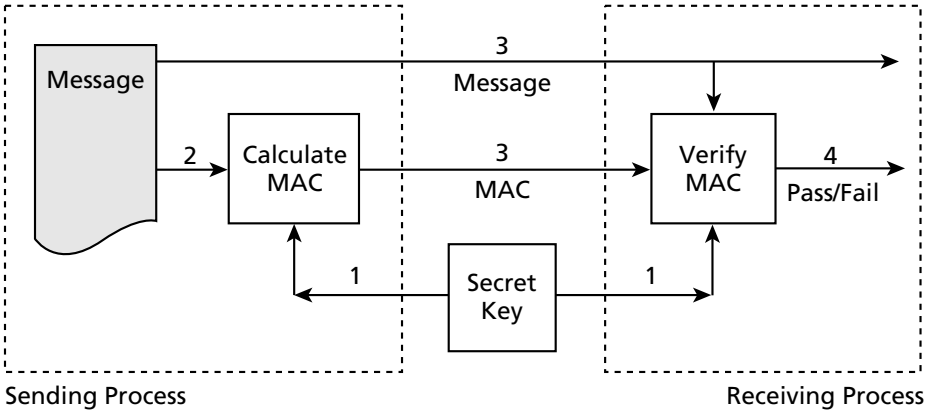
Like other modern digital cryptographic functions, message digest functions are typically used and sometimes defined only for integer numbers of octets, rather than arbitrary numbers of bits.

2.2 Message Authentication Codes

A message authentication code (MAC) function computes a MAC from a message and a secret key. If the originator and the receiver share knowledge of that secret key, the receiver can calculate the same function of the message and secret key and see if it matches the MAC accompanying the message. See Figure 2-2. If the MAC matches, then you know, within the strength of the MAC function and key, that some program with possession of the secret produced the MAC. Of course, every program that can verify the MAC needs to know this secret. Thus all of them can create valid MACs even if they should only receive and verify these codes.

A simple MAC function might append the secret to the message, then calculate a message digest of the result and use it as the MAC. The message (without the secret) and MAC could then be sent to the recipient. The recipient would also append the secret (which the receiver needs to know as well) to the message and calculate the same message digest function. If the resulting digest matches the MAC, it validates the message.

MACs are normally based on a message digest code; however, the simple technique suggested previously has subtle weaknesses that are beyond the scope of this book. Better, stronger MAC functions exist. All of the MAC functions discussed in this book (see Chapter 18) are based on the provable "security" HMAC [RFC 2104] method of combining a message, secret quantity, and message digest function to produce a MAC. [Schneier] documents other strong MAC functions as well.



1. Distribute the secret key. Done in advance by secure channels.
2. Sender calculates the MAC of the message and the key.
3. Transmit the message and MAC to the receiver.
4. Receiver calculates the MAC using the message and key, compares it with the transmitted MAC, and judges the integrity of the transmission.

Figure 2-2 | Message authentication codes

A difficulty with MAC authentication in a system with multiple originators and receivers is that you must choose between two strategies, both of which have problems:

1. Have a different secret for every pair of entities. This method is logistically difficult because the number of keys increases with the square of the number of entities and the keys must be securely distributed. If the system includes E number of entities, you have E*(E – 1)/2 pairs. For example, for 100 entities, you have 4950 pairs. For 1000 entities, you have 499,500 pairs.

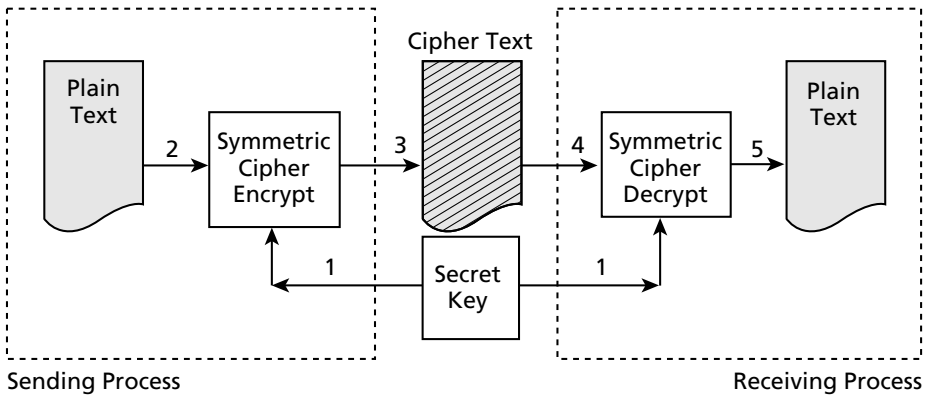
2. Share one secret among all the entities. This technique is relatively insecure. The more entities that have a secret, the more likely the secret is to be compromised due to loss, subversion, or betrayal. This technique also means the same secret will be used many times; the more exposures of the uses of a secret, the easier an adversary may find it to break that secret analytically. In addition, with this strategy any of the entities can forge messages from any of the other entities and a recipient will be unable to detect this fraud based on the MAC.

Digital signatures, as described in Section 2.6, are an operationally superior method of authentication in many circumstances.

As with message digest functions, if a strong MAC is N bits long, the difficulty of finding two messages with the same MAC is proportional to $2^{N/2}$. You should pick N large enough for your application and then, to avoid the secret quantity being the weak point, use a secret quantity that is random (see Section 2.10) and at least N/2 bits long. For example, if you need a 160-bit MAC, the secret key should be at least 80 random bits.

2.3 Secret or Symmetric Key Ciphers

Secret key ciphers have been known for centuries, although the oldest were weak and had no key. Today's modern ciphers involve a reversible data scrambling system such that it is computationally infeasible to recover the unscrambled data without knowledge of the secret key used to scramble that data. See Figure 2-3. The original version is called the plain text; the scrambled version is called the cipher text.



1. Distribute the secret key. Done in advance by secure channels.
2. Feed plain text into the symmetric cipher to encrypt it under the key.
3. Store or transmit the cipher text.
4. Retrieve or receive the cipher text, and feed it into the symmetric cipher algorithm to decrypt it under the key.
5. The result should be the original plain text.

Figure 2-3 | Secret key encryption

Perhaps the best-known symmetric cipher is the Data Encryption Standard (DES [FIPS 186-2]). With 56 bits of actual key and a number of weaknesses, DES is not considered strong by current standards. In addition, it was designed for hardware efficiency, not software efficiency. Triple DES is a popular stronger symmetric cipher consisting of three applications of DES with different keys. Of course, it is also three times slower than DES.

Recently, NIST (see Appendix D) ran an extensive public process to select a successor to DES. They eventually chose the submission called Rijndael, now known as the Advanced Encryption Standard (AES [FIPS 197]). This encryption method has three variations with different key sizes and strengths. AES is generally believed, even in its weakest variation, to be stronger than Triple DES and faster in software than DES.

For an ideal symmetric cipher with an N-bit key, no method should be able to find the plain text with an expected effort significantly less than trying 2^{N-1} key values. In reality, symmetric ciphers often fall short of this ideal. That is, although they have an N-bit key, due to some systematic weakness, there is some technique to crack the code with less expected effort than trying 2^{N-1} key values. As long as the weaknesses are known and the cipher remains strong enough, this shortcoming is acceptable.

Symmetric ciphers among multiple entities have the same security problems as message authentication codes. It may be logistically impractical to have a separate secret key pair between all pairs of entities and insecure for all of them to share the same secret key. "Enveloped encryption," as discussed in Section 2.8, may represent a superior alternative. If the system includes a centralized trusted server, it is also possible to use a secret key distribution system such as [Kerberos]. Kerberos is not covered in this book.

In real systems, further complexities arise. Many symmetric ciphers are "block ciphers" that work on blocks of data significantly larger than an eight-bit byte or octet. Thus they commonly require a "padding" method to match the data to an exact multiple of the block size before encryption and a corresponding unpadding after decryption. In addition, algorithms can be used in a variety of "modes," such as "electronic codebook mode," "cipher feedback chaining mode," and "output feedback chaining mode." See [Schneier], [FIPS 46-3, FIPS 81], or similar references for information on these topics. For the particular block cipher algorithms described in Chapter 18, the chaining and padding methods are described or referenced there.

2.4 Public or Asymmetric Key Ciphers

"Public" key ciphers rely on the recent (a few decades old) discovery of encryption functions with the following characteristics:

1. A different key is needed to decrypt information than the key used to encrypt it. See Figure 2-4.
2. It is computationally infeasible to determine the decryption key from the encryption key.
3. Knowing the encryption key offers no help in decryption. Knowing the decryption key offers no help in encryption.

This set of characteristics dramatically changes the security model in comparison with secret key ciphers. Some application that wants to receive confidential messages could advertise a "public" encryption key to the world,

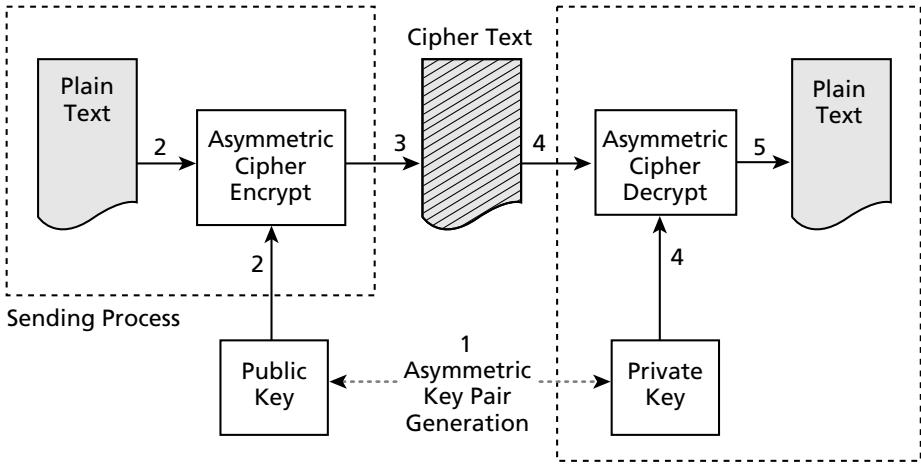


Figure 2-4 Public key encryption

while retaining and keeping secret the private decryption key. Senders would encrypt the data under this public encryption key. Only the intended recipient, who has the private decryption key, can then read it.

This cipher eliminates the problems associated with securely distributing symmetric keys among many entities in a large system. Only one public-private key pair is required for each entity and purpose because everyone can safely know all of the public keys. Unfortunately, two new problems arise with this technique:

1. Asymmetric encryption algorithms tend to be very slow. Thus, if the user sends a large confidential message, it may be prohibitive to encrypt all of the data with a public key algorithm. This problem is solved with "enveloped encryption" (see Section 2.8).
2. It becomes critical to determine whether you are using the right public key. If an adversary can convince you to use the adversary's public encryption key, instead of the public key for the intended recipient, then the adversary can decrypt your message. Remember this individual has the private decryption key corresponding to his or her own public key. After reading your secret message, the adversary may even be able to re-encrypt it with the intended recipient's public encryption key and forward it to the intended recipient. You and the intended recipient would never realize that the message had been compromised. Certificates address this problem of trust in public keys (see Section 2.7).

Real-world use of asymmetric ciphers also requires padding methods because asymmetric ciphers usually operate on a block of data whose size may depend on the key size. For the specific asymmetric algorithms described in Chapter 18, the padding method is described or referenced there.

2.5 Asymmetric Keys and Authentication

Asymmetric authentication algorithms also change the security model for signatures compared with message authentication codes. A program originating data that it wants to authenticate can send, along with that data, the same data transformed under a private key and make known the corresponding public key. (Note: Which key is public and which is private is the reverse of the confidentiality case mentioned earlier.) Then, anyone with access to

the sender's public key can verify the message using the plain text and transformed text, and determine that it comes from the sender—only the sender should have the necessary private key. This technique solves the two problems mentioned in Section 2.2 for MAC symmetric key distribution, but brings the same two new problems listed in Section 2.4, efficiency and public key trust, for public key confidentiality.

Section 2.6 on digital signatures discusses ways to handle the issue of asymmetric cipher algorithm efficiency. Section 2.7 describes the use of certificates to address the critical problem of determining which public key to use.

N-bit asymmetric keys for asymmetric algorithms are usually much weaker than the corresponding-size keys for symmetric algorithms. For example, a 2400-bit asymmetric RSA key is generally considered to only be as strong as a 112-bit triple DES symmetric cipher key [Orman] while a 112-bit asymmetric key would, for many asymmetric algorithms, be quite weak. This issue does not create a problem as long as you use large enough asymmetric keys to compensate.

2.6 Digital Signatures

You can use public key authentication to produce "digital signatures." These signatures have a very desirable characteristic—namely, it is computationally infeasible for anyone without the private key to produce a signature that will verify for a given message. Modern digital signatures consist of (1) a message and (2) a message digest of that message asymmetrically transformed under a private key of the signer. See Figure 2-5.

Because message digests are short, fixed-length quantities, the slowness of public key algorithms has minimal effects on processing. The critical need to be sure you are using the right public key still exists, however, and is usually addressed by certificates.

Real digital signature systems have many more complexities than this brief description suggests. The actual quantity being secured by asymmetric transformation under a private key typically includes not just the critical message digest value, but also two other items:

- Identification of the message digest function
- Possibly other information such as date signed or key identifier

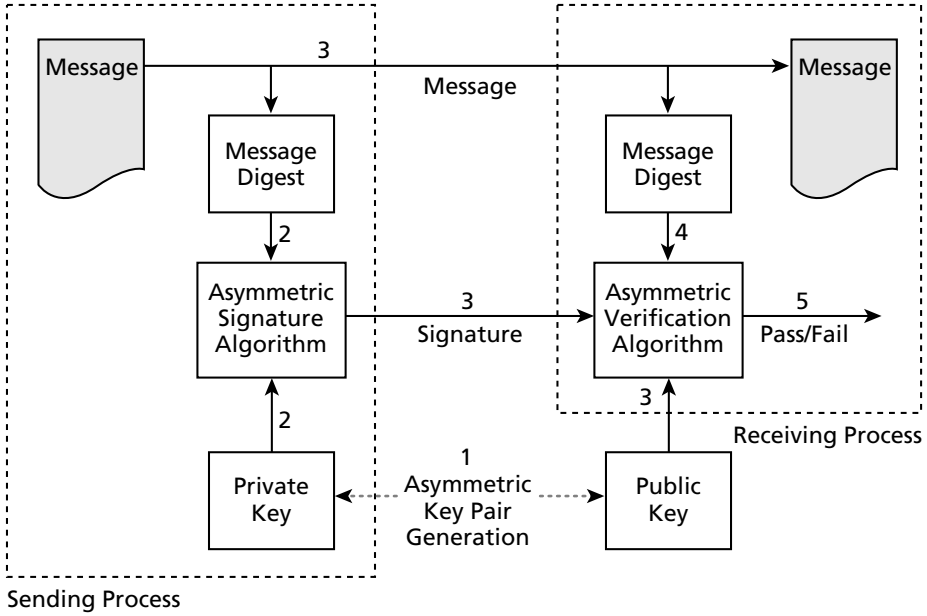


Figure 2-5 | Digital signatures

This information is formatted so that a signature verifier can parse it. The verifier uses the asymmetric algorithm, public key, and material transformed under the private key (including the message digest value, which it computes over the plain text) to verify that the corresponding private key signed it. In addition, the overall signed message must use some known format so that the system can separate the signature and possibly key identification information from the signed message information.

2.7 Certificates

Certificates offer a way of providing assurance about a public key. In general, they consist of the following components:

- The public key
- Some associated information such as an identity or access authorization
- A date of issuance and expiration
- An authenticating digital signature by a "certification authority" over this information

Anyone knowing and trusting the public key of this "authority" and having the certificate can have confidence that the public key inside the certificate is associated with the identity or should have the access indicated. See Figure 2-6.

This verification can be continued through a "chain" of certificates. Cert-1 signed by entity-1 can provide trust in the public key of entity-2, whose private key can then be used to sign Cert-2, which provides trust in the public key of entity-3, whose private key can be used to sign Cert-3, and so on. If one or only a few globally known and trusted public keys exist, a hierarchical model results, as shown in Figure 2-7. Top-level certification authorities control the private keys corresponding to these top-level or "root" public keys.

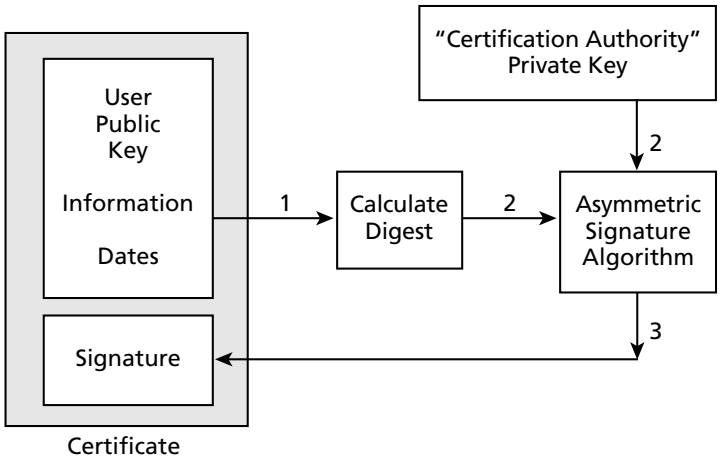


Figure 2-6 Certificate

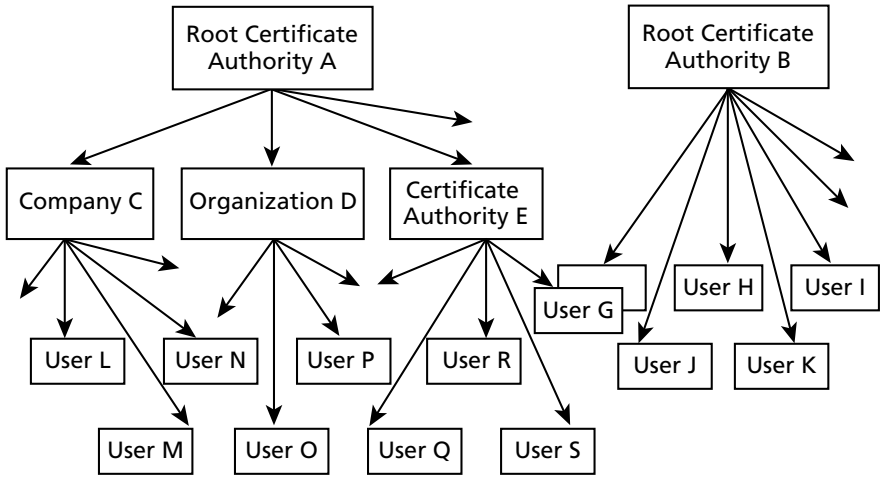


Figure 2-7 | Certificate hierarchies

These authorities sign certificates for lower-level certification authorities, possibly through several levels, ending with bottom-level "user" certificates.

The alternative to this hierarchical model is a mesh model, as shown in Figure 2-8. With a mesh, various entities sign one another's certificates. A user can then start from a few trusted public keys and, subject to any criterion desired, chain through available certificates to gain trust in other public keys.

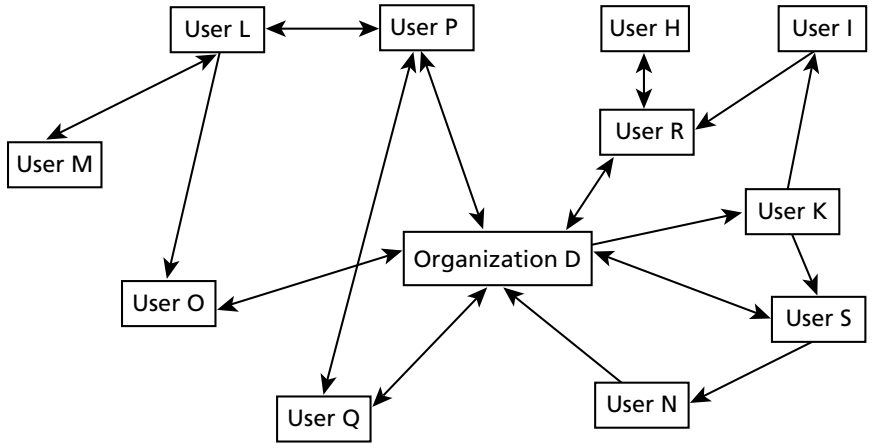


Figure 2-8 | A certificate mesh

The most common type of certificates are X.509v3 [ISO 9594] certificates. Derived from the OSI X.500 Directory standard, they were originally intended to authenticate update authority and the like on the hypothetical global X.500 directory. The original X.509 certificates bound a public key to an X.500 identity. X.509v3 (version 3) generalizes X.509 certificates in various ways to allow for other types of more reasonable identities (such as e-mail and IP addresses) and extensions.

X.500 identities are baroque hierarchical names, in which each level of the hierarchy consists of an arbitrary, unordered set of attribute–value pairs. They are just one of the complexities and false assumptions (such as the assumption that everyone would allow themselves to be listed in one global public directory, including companies listing all their employees) that doomed the X.500 Directory as originally conceived.

In theory, you can have an X.509v3 mesh. In fact, such certificates are normally used in a hierarchical mode. A common root is a commercial certification authority. An intermediate-level certification authority might be a company that obtained a certificate from a root and then issued certificates to its employees. All X.509 certificates and X.500 names appear in the binary ASN.1 BER or DER encoding. While it might seem odd to use such binary non-XML complexities to support XML Security, most of the existing public key infrastructures and commercial certification authorities work with X.509v3 so it is convenient to be able to use such certificates in XML Security.

Other types of certificates exist as well. Pretty Good Privacy [RFC 2440] has its own certificates that are almost always used in a mesh, although they can be employed hierarchically. A recent attempt to develop simpler standard certificates in the Internet Engineering Task Force (IETF) produced the Simplified Public Key Infrastructure (SPKI) certification system [RFC 2693]. Many people had hoped that SPKI (pronounced "spooky") would provide a greatly simplified and rationalized direct replacement for X.509v3. Instead, SPKI, while simple, made some radical changes. As a result, it has seen only limited deployment.

Certificate Revocation Lists

Compromised private keys really do happen. For example, a private key owner may accidentally release the key or someone may steal, guess, or compute a

copy of the private key and erroneously issue certificates. In this case, a certification authority issues a certificate for the wrong entity. In theory, the problem of compromised private keys or erroneously issued certificates is handled by a "revocation" mechanism. Whoever signed the original certificate signs and distributes a revocation of that certificate. In reality, such revocation systems require communication from the revoking entity to the entity trusting the certificate, and an adversary can block that communication.

To provide revocation, X.509 defines a certificate revocation list (CRL) structure. This structure is dated and carries a version number. It lists revoked certificates as of that date and time from a particular certificate issuer (except for certificates that have already expired). It also gives a date before or at which the next CRL version will be issued. Thus, for example, if you receive a signature that you trust based on a chain of certificates, you should get CRLs for all issuers in the chain. Furthermore, for any issuers for which you have only a CRL issued before the signature date, you should wait until the next CRL to see whether the certificate becomes revoked. This system is not terribly convenient, so most implementations, including the most popular browsers, ignore certificate revocation and CRLs entirely.


History

In January 2001, Verisign, Inc., the largest commercial certification authority, issued two of its highest-assurance-level X.509v3 certificates to some person or persons unknown who impersonated Microsoft Corporation. Thus, unless they took special precautions or obtained and followed Verisign's relevant CRL, anyone who trusted Verisign would trust these imposters to speak for Microsoft.

Online Certificate Status Protocols

Certificates were designed in a time when offline verification was virtually the norm. With modern, always-connected technology, it should be possible to dispense with the entire certificate structure and ask a trusted online server to provide or verify trust in a key. (See Chapter 14 for a proposed XML way of achieving this goal.)

The Online Certificate Status Protocol (OCSP) represents a halfway step toward this end. It eliminates CRLs by utilizing a trusted online server to indicate whether a given certificate submitted to it remains valid [RFC 2560]. A client of such a server would have to know and trust the public key of that

server. But, because such servers shield the client from complexity, the client probably needs to know only the key for one server. That server can then contact others on behalf of the client, if necessary, and forward any responses, providing its own signature to authenticate them.

For some systems (see Chapter 12), a signature may be more easily enforceable if you have checked and saved "proof" that all certificates you used to verify the signature were valid at the time. The trusted server signs and dates OCSP responses, which can be saved like CRLs.

2.8 Enveloped Encryption

Modern public key encryption systems that encrypt arbitrary-size messages use a combination of secret key and public key ciphers. Stated more precisely, they generate a random symmetric key to encrypt each message and then encrypt that key with a public encryption key of the intended recipient. The symmetrically encrypted message is then sent along with the asymmetrically encrypted random key. See Figure 2-9.

This type of encryption takes advantage of the more efficient symmetric cipher, avoiding the problem of the slowness of public key systems for large messages, while still gaining the more convenient key distribution model of public key cryptography. Of course, you must still ensure that you are really using the public key of the intended recipient of the encrypted messages, an issue that is commonly addressed through certificates.

If a message is sent confidentially to more than one recipient, the sender can transmit separate enveloped encryptions to each one. More commonly, senders create one enveloped encryption employing only one cipher text and symmetric key. That symmetric key then appears several times encrypted under a public encryption key of each intended recipient. These public keys need not be the same size or even use the same public key algorithm. Intended recipients use their own private keys to decrypt the appropriate public key encrypted copy of the symmetric key. They can then use the symmetric key to decrypt the confidential information.

Enveloped encryption does not authenticate the message's originator. If that service is desired, it is usually combined with a digital signature (Section 2.6). Such a digital signature can appear either inside or outside the encryption (or you can use two signatures, one in each place).

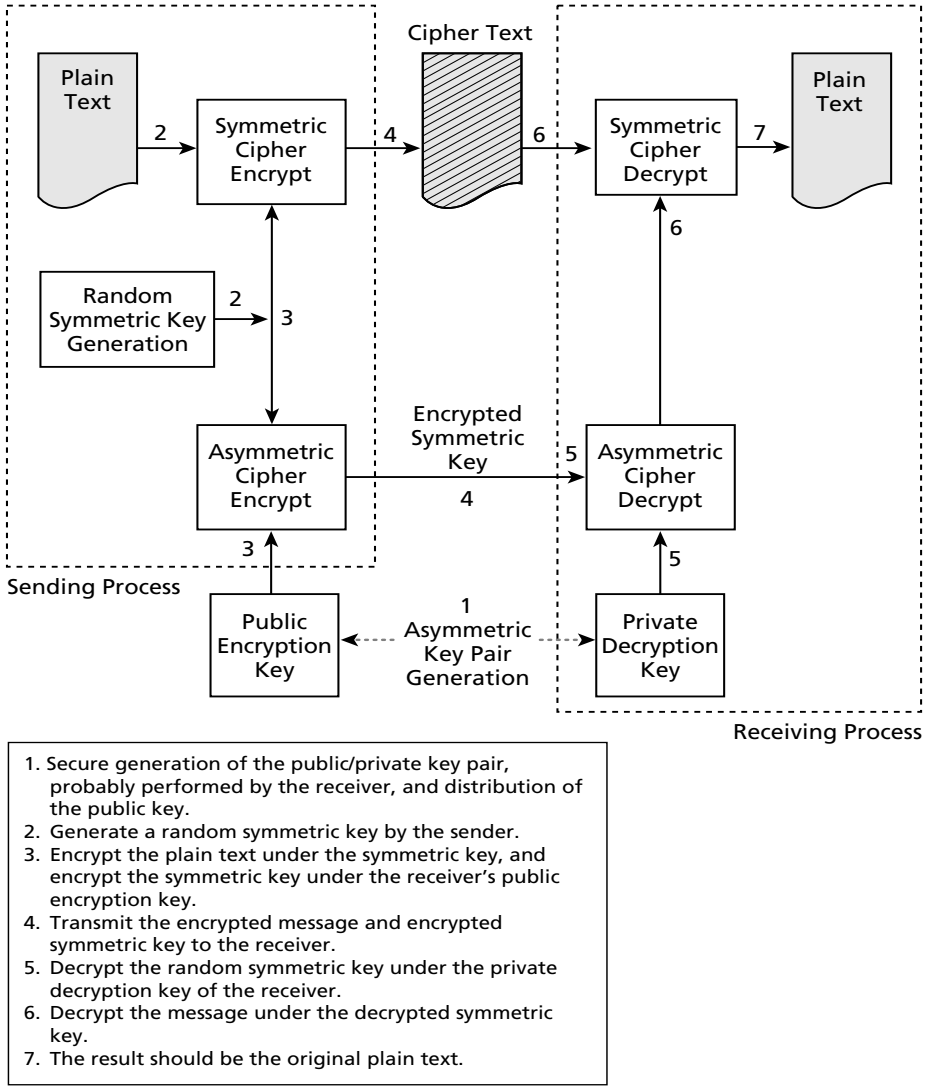


Figure 2-9 | Enveloped encryption

"Outside the encryption" means that the signature appears over the entire enveloped encryption structure and the signature takes the form of plain text. As a consequence, the recipient can first test the signature before decrypting the message. Depending on the exact presentation of the keying information for the signature, this process can also reveal the identity of the originator. Furthermore, the signature will not be useful without the cipher

text. The recipient cannot demonstrate the validity of the signature in connection with the plain text to a third party without giving the cipher text and revealing the recipient's private decryption key, which may compromise many other messages.

Placing the signature "inside the encryption," so the plain text of the confidential information together with the signature is encrypted, is generally preferable. This strategy assures that the signature is not visible in the cipher text and so cannot reveal the originator's identity. Furthermore, the signature is valid after decryption and so can be demonstrated to a third party without compromise of any recipient private decryption key. For further discussion of these issues, see Chapter 16.

2.9 Canonicalization

Canonicalization is a critical aspect of digital signatures and verification. It also has limited applicability to encryption.

To be useful, signatures (and message authentication codes, if appropriate) must be secure and robust. For the signature to be secure, any "significant" change in the signed data or the signature must cause the signature to fail. For the signature to be robust, any "insignificant" change in the signed data, or the signature itself, must not cause the signature to fail. Any change allowed by normal receipt, storage, and/or transmission of the message should be considered insignificant and should not be covered by the signature. Figuring out exactly what is significant for signature purposes can prove tricky. Message digest algorithms, which are used in message authentication codes and digital signatures, reflect **any** change in their input, so you must manage their input carefully. In particular, that input should normally consist of a canonicalization of the data being secured, discarding insignificant aspects of that data.

Chapter 9 is entirely devoted to canonicalization, particularly as it pertains to XML.

2.10 Randomness

The keys used in digital security must be generated "randomly." For our purposes, "random" is defined as hard to guess, so this makes it more difficult to guess the key. This goal turns out to be surprisingly challenging to achieve on

a computer. One strategy is to use true physical randomness such as thermal noise or radioactive decay, but it requires special hardware and usually produces random bits fairly slowly. More commonly, systems use algorithmic "pseudo-random" number generators. Unfortunately, to be unguessable, they initially require some sort of strong random seed value. Frequently such a seed can be derived from some hardware source of randomness.

Many real-world systems that did almost everything else right have been broken due to weak random numbers. Perhaps they based their random number generation on a seed that uses only the time and date. As a result, anyone with a general idea of when the seed was generated will have an embarrassingly small space to search through to find the key—possibly only a few dozen values—even if the key is 128 bits and should have 2^{128} equally probable values.

For a deeper and more detailed discussion of these issues, see [RFC 1750] and [Schneier].


History

Early versions of Netscape Navigator, although they did most everything else correctly in terms of security, depended (as have other security critical software products) on a library "random" number generator for SSL keys. On some platforms, this generator used the time of day and process ID as a seed. Its output was relatively easy to guess. This problem has, of course, been fixed. Because it drew a bit of attention, people have become more cautious about this issue. But give it a few years and likely someone will make the same mistake again. It's very easy to put in a "temporary" weak random number generator while developing a system and never get around to upgrading it.

2.11 Other Facets of Security

Next, we look briefly at a few other important facets of a complete security system, albeit issues that are somewhat outside the scope of digital cryptography. An overall security system is only as secure as the weakest facet.

Key Rollover

No key should be used forever. The longer a key has been in use and the more often its uses are exposed, the greater the probability of it being compromised due to accident, subversion, or cryptanalysis. Most systems require

regular key updates and a plan for nonscheduled rollover in case of known compromise. While the timing for such updates depends on the particular circumstances, most public keys should not be used for more than a year. In fact, sometimes it is reasonable to use a key—for example, an enveloped encryption symmetric key—once only.

Physical Security

Devices and areas where keys are exposed, cryptographic computations are performed, and the plain text version of cipher text appears must be physically secure. If an adversary can obtain keys or passwords by getting them from computer memory, observing user keystrokes, or similar activities, you are sunk. Cryptographic security relies on the security of the keys. If the actual cryptographic computations can be observed, changed, or bypassed, security is lost.

Personnel Security

In security systems of any complexity, there are always people whom you must trust. They include, but are not limited to, people with physical access to the keying material, people who implemented the software and/or hardware involved with critical operations, and people who designed the system. If operation of the system is critical or protects valuable secrets, how do you assure that these people are trustworthy?

Procedural Security

Even with good cryptography, physical, and personnel security, what sort of administrative procedures do you have? If a security violation or compromise occurs, who reports it and what action is taken? Does anyone actually check that what is supposed to be done is being done, that encrypted data are actually secure?


History

One implementation of secure Telnet used a 64-bit DES key, which includes eight nominal "parity" bits. The keys consisted of 64 randomly generated bits, but the actual encryption/decryption routine ignored the supplied key and used a key of all zero bits if the parity was wrong! Consequently, more than 99.5% of the time, the same zero key was used [JIS]. This system scrambled the bits so they looked secure to a human. This code interoperated well with other copies of itself, as both ends made the same mistake, but in a way that

was very insecure to anyone trying a few keys. A zero key is particularly obvious to try when attempting to decode unknown cipher text because it could result from a software or hardware failure and is one of the four "weak" keys for DES. The lesson: Constant vigilance and oversight of security systems are needed.

2.12 Cryptography: A Subtle Art

This chapter has presented just a brief overview of the basics of digital cryptography at a high level. Beyond the security of individual algorithms lies the question of overall system security. A secure system can be made insecure by the addition of one more operation/feature even if that operation is secure in isolation.

Technical progress in this area occurs constantly. Unless you want to make the study of this area into your life's work, we suggest you follow these guidelines:

- Do not depend on security by obscurity (i.e., the secrecy of your algorithms). Depend only on the secrecy of symmetric and private keys.
- Do not design your own basic cryptographic algorithms or formats. Use recent—but not too recent—strong algorithms and formats that have been subject to public scrutiny.
- Pay attention to such often neglected areas as randomness generation and proper canonicalization for your application.
- Keep in mind that physical security of cryptographic processing and key storage; covert channels and electromagnetic emanations that can compromise keying material or plain text; traffic analysis; operational procedures; and personnel security, all of which are beyond the scope of this book, can be crucial, depending on your threat model.
- Cross your fingers and consider actively monitoring for intrusion or compromise: No matter how careful you are, security is never perfect.