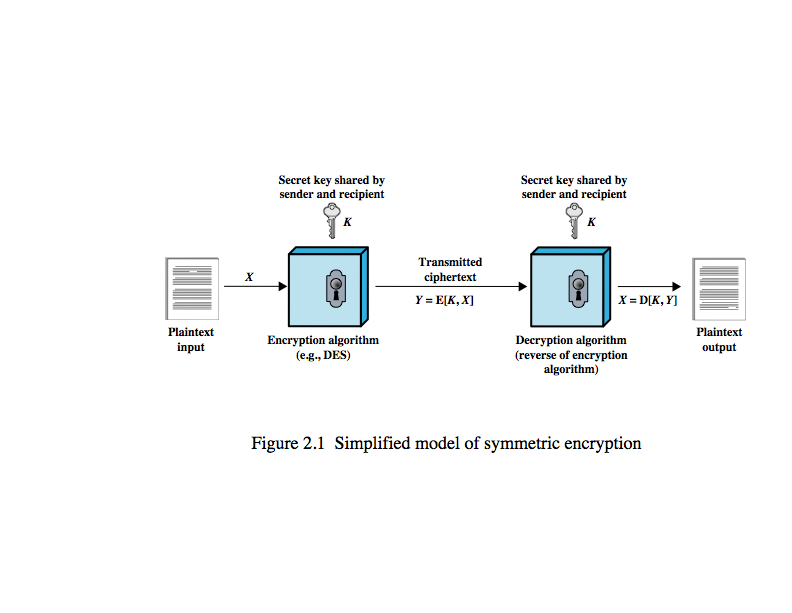
**Cryptographic Tools**

An important element in many computer security services and applications is the use of cryptographic algorithms. This chapter provides an overview of the various types of algorithms, together with a discussion of their applicability. For each type of algorithm, we introduce the most important standardized algorithms in common use. We begin with symmetric encryption, which is used in the widest variety of contexts, primarily to provide confidentiality. Next, we examine secure hash functions and discuss their use in message authentication. The next section examines public-key encryption, also known as asymmetric encryption. We then look in at the two most important applications of public-key encryption, namely digital signatures and key management. In the case of digital signatures, asymmetric encryption and secure hash functions are combined to produce an extremely useful tool. Finally, in this chapter we provide an example of an application area for cryptographic algorithms by looking at the encryption of stored data.

**Symmetric Encryption**

The universal technique for providing confidentiality for transmitted data is symmetric encryption. Symmetric encryption, also referred to as conventional encryption or single-key encryption, was the only type of encryption in use prior to the introduction of public-key encryption in the late 1970s. Countless individuals and groups, from Julius Caesar to the German U-boat force to present-day diplomatic, military, and commercial users, use symmetric encryption for secret communication. It remains by far the more widely used of the two types of encryption. A symmetric encryption scheme has five ingredients, as shown here in Figure 2.1 from the text.

• Plaintext: This is the original message or data that is fed into the algorithm as input.

• Encryption algorithm: The encryption algorithm performs various substitutions and transformations on the plaintext.

• Secret key: The secret key is also input to the encryption algorithm. The exact substitutions and transformations performed by the algorithm depend on the key.

• Ciphertext: This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts.

• Decryption algorithm: This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

There are two requirements for secure use of symmetric encryption:

1. We need a strong encryption algorithm.
2. Sender and receiver must have secure obtained, & keep secure, the secret key.

**Attacking Symmetric Encryption**

There are two general approaches to attacking a symmetric encryption scheme. The first attack is known as **cryptanalysis**. Cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some sample plaintext-ciphertext pairs. This type of attack exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or to deduce the key being used. If the attack succeeds in deducing the key, the effect is catastrophic: All future and past messages encrypted with that key are compromised. The second method, known as the **brute-force** **attack**, is to try every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained. On average, half of all possible keys must be tried to achieve success.

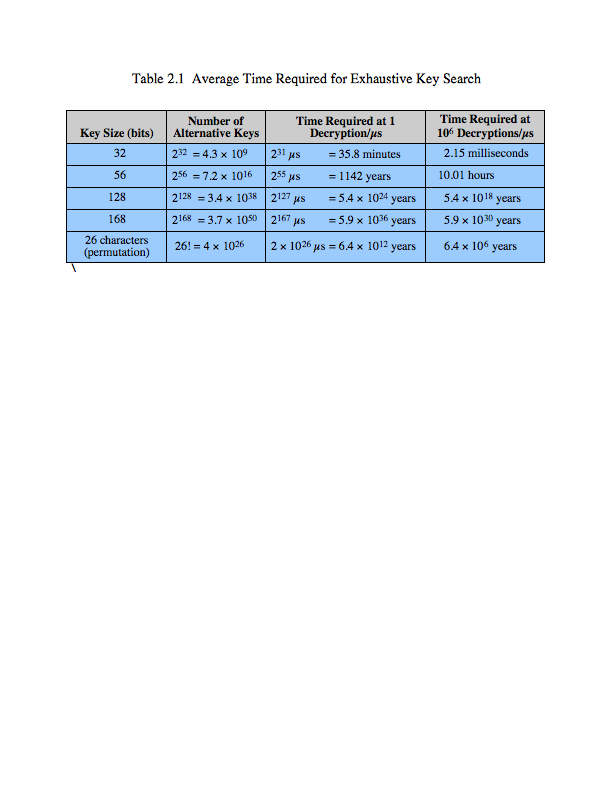
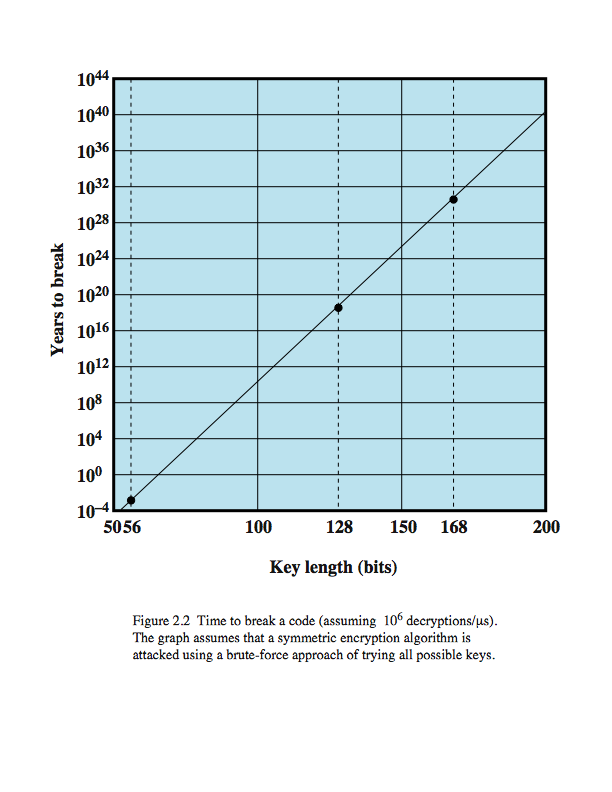
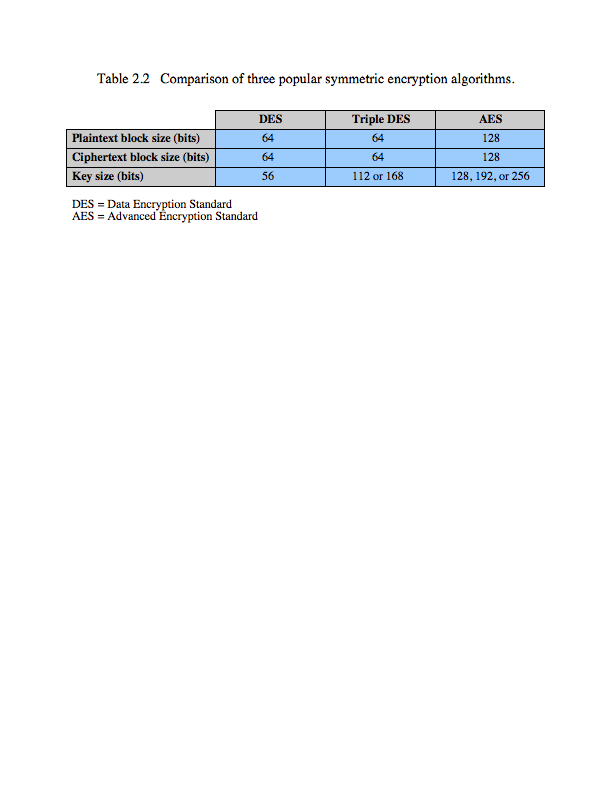
**Exhaustive Key Search**

Table 2.1 from the text shows how much time is involved for various key sizes. The table shows results for each key size, assuming that it takes 1 µs to perform a single decryption, a reasonable order of magnitude for today's computers. With the use of massively parallel organizations of microprocessors, it may be possible to achieve processing rates many orders of magnitude greater. The final column of the table considers the results for a system that can process 1 million keys per microsecond. At this performance level, a 56-bit key is no longer computationally secure.

The assumption of one encryption per microsecond is overly conservative. The widely used encryption scheme, the Data Encryption Standard (DES) was finally and definitively proved insecure in July 1998, when the Electronic Frontier Foundation (EFF) announced that it had broken a DES encryption using a special-purpose "DES cracker" machine that was built for less than $250,000. The attack took less than three days. The EFF has published a detailed description of the machine, enabling others to build their own cracker [EFF98]. It is important to note that there is more to a key-search attack than simply running through all possible keys. Unless known plaintext is provided, the analyst must be able to recognize plaintext as plaintext. If the message is just plain text in English, then the result pops out easily. If the message is some more general type of data, such as a numerical file, and this has been compressed, the problem becomes even more difficult to automate. Thus, to supplement the brute-force approach, some degree of knowledge about the expected plaintext is needed, and some means of automatically distinguishing plaintext from garble is also needed. The EFF approach addresses this issue as well and introduces some automated techniques that would be effective in many contexts. Figure 2.2 shows how long it would take to crack a DES-style algorithm as a function of key size. .

**Symmetric Encryption Algorithms**

The most commonly used symmetric encryption algorithms are block ciphers. A block cipher processes the plaintext input in fixed-size blocks and produces a block of ciphertext of equal size for each plaintext block. The algorithm processes longer plaintext amounts as a series of fixed-size blocks. The most important symmetric algorithms, all of which are block ciphers, are the Data Encryption Standard (DES), triple DES, and the Advanced Encryption Standard (AES); as summarized here in Table 2.2 from the text.Symmetric Encryption Algorithms

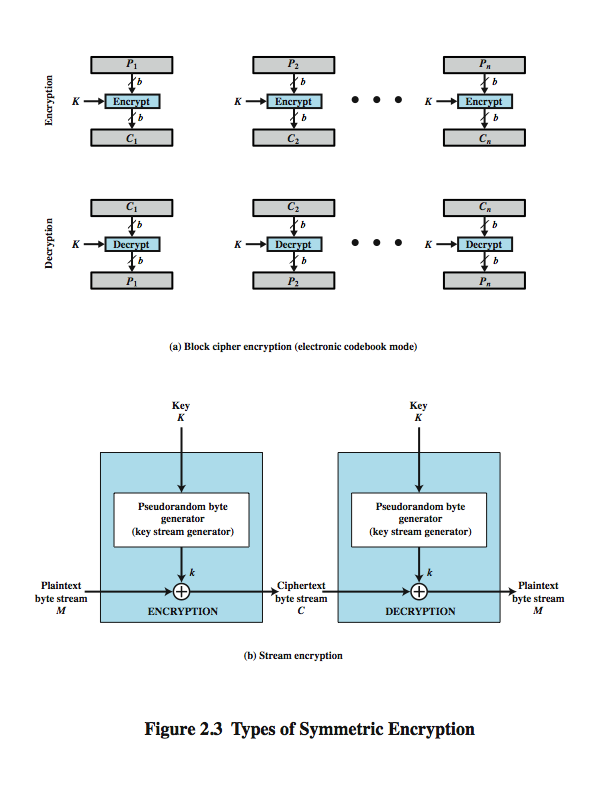
**DES and Triple-DES**

The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the NIST, as FIPS PUB 46. The algorithm itself is referred to as the Data Encryption Algorithm (DEA). DES takes a plaintext block of 64 bits and a key of 56 bits, to produce a ciphertext block of 64 bits. Concerns about the strength of DES fall into two categories: concerns about the algorithm itself and concerns about the use of a 56-bit key. The first concern refers to the possibility that cryptanalysis is possible by exploiting the characteristics of the DES algorithm. Over the years, there have been numerous attempts to find and exploit weaknesses in the algorithm, making DES the most-studied encryption algorithm in existence. A more serious concern is key length. With a key length of 56 bits, there are 256 possible keys, which is approximately 7.2 × 1016 keys. As noted on the previous slide, this can now be broken relatively easily.

The life of DES was extended by the use of triple DES (3DES), which involves repeating the basic DES algorithm three times, using either two or three unique keys, for a key size of 112 or 168 bits. Triple DES (3DES) was first standardized for use in financial applications in ANSI standard X9.17 in 1985. 3DES was incorporated as part of the Data Encryption Standard in 1999, with the publication of FIPS PUB 46-3. 3DES has two attractions that assure its widespread use over the next few years. First, with its 168-bit key length, it overcomes the vulnerability to brute-force attack of DEA. Second, the underlying encryption algorithm in 3DES is the same as in DEA. The principal drawback of 3DES is that the algorithm is relatively sluggish in software.

**Advanced Encryption Standard (AES)**

Because of its drawbacks, 3DES is not a reasonable candidate for long-term use. As a replacement, NIST in 1997 issued a call for proposals for a new Advanced Encryption Standard (AES), which should have security strength equal to or better than 3DES and significantly improved efficiency. In addition to these general requirements, NIST specified that AES must be a symmetric block cipher with a block length of 128 bits and support for key lengths of 128, 192, and 256 bits. Evaluation criteria included security, computational efficiency, memory requirements, hardware and software suitability, and flexibility. In 2001, AES was issued as a federal information processing standard (FIPS 197). In a first round of evaluation, 15 proposed algorithms were accepted. A second round narrowed the field to 5 algorithms. NIST completed its evaluation process and published a final standard (FIPS PUB 197) in November of 2001. NIST selected Rijndael as the proposed AES algorithm. AES is now widely available in commercial products.

**Block verses Stream Ciphers**

A block cipher processes the plaintext input in fixed-size blocks and produces a block of ciphertext of equal size for each plaintext block. The algorithm processes longer plaintext amounts as a series of fixed-size blocks. Typically, symmetric encryption is applied to a unit of data larger than a single 64-bit or 128-bit block. Plaintext sources must be broken up into a series of fixed-length block for encryption by a symmetric block cipher. The simplest approach to multiple-block encryption is known as electronic codebook (ECB) mode, in which plaintext is handled b bits at a time and each block of plaintext is encrypted using the same key. Typically b=64 or b=128. Figure 2.3a here shows the ECB mode. A plaintext of length nb is divided into n b-bit blocks. Each block is encrypted using the same algorithm and the same encryption key, to produce a sequence of n b-bit blocks of ciphertext. To increase the security of symmetric block encryption for large sequences of data, a number of alternative techniques have been developed, called modes of operation (see chapter 19).

A stream cipher processes the input elements continuously, producing output one element at a time. Although block ciphers are far more common, there are certain applications in which a stream cipher is more appropriate. A typical stream cipher encrypts plaintext one byte at a time, as shown in Figure 2.3b. The output of a pseudorandom number generator (the keystream), is combined one byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation. With a properly designed pseudorandom number generator, a stream cipher can be as secure as block cipher of comparable key length. The primary advantage of a stream cipher is that stream ciphers are almost always faster and use far less code than do block ciphers. The advantage of a block cipher is that you can reuse keys.

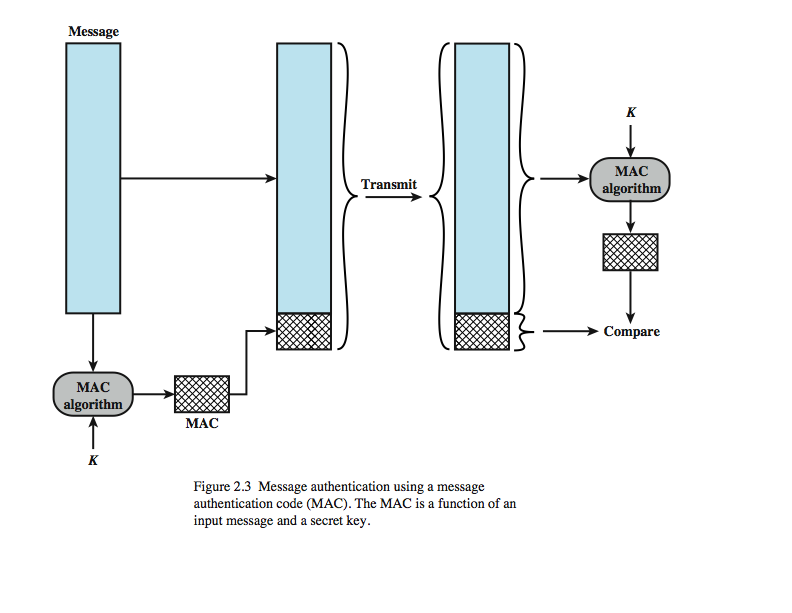
**Message Authentication**

Encryption protects against passive attack (eavesdropping). Message authentication protects against active attacks (falsification of data and transactions), by verifying that received messages are authentic, that is that the contents of the message have not been altered and that the source is authentic. We may also wish to verify a message's timeliness and sequence relative to other messages flowing between two parties.

It is possible to perform authentication simply by the use of conventional encryption. If we assume that only the sender and receiver share a key (which is as it should be), then only the genuine sender would be able to encrypt a message successfully for the other participant. Furthermore, if the message includes an error-detection code and a sequence number, the receiver is assured that no alterations have been made and that sequencing is proper. If the message also includes a timestamp, the receiver is assured that the message has not been delayed beyond that normally expected for network transit.

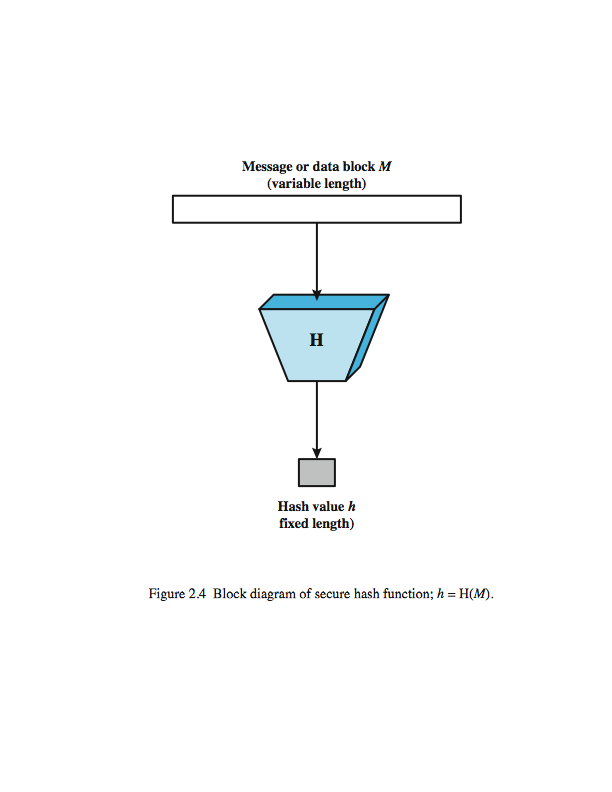
Alternatively there are several approaches to message authentication that do not relay on encryption. In all of these approaches, an authentication tag is generated and appended to each message for transmission. The message itself is not encrypted and can be read at the destination independent of the authentication function at the destination.

**Message Authentication Codes**

One authentication technique involves the use of a secret key to generate a small block of data, known as a message authentication code, that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key *KAB*. When A has a message to send to B, it calculates the message authentication code as a function of the message and the key: MAC*M* = F(*KAB*, *M*). The message plus code are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new message authentication code. The received code is compared to the calculated code, as shown here in Figure 2.4 from the text. If we assume that only the receiver and the sender know the identity of the secret key, and if the received code matches the calculated code, then:

1. The receiver is assured that the message has not been altered.
2. The receiver is assured that the message is from the alleged sender.
3. If the message includes a sequence number, then the receiver can be assured of the proper sequence.

A number of algorithms could be used to generate the code. The NIST specification, FIPS PUB 113, recommends the use of DES. DES is used to generate an encrypted version of the message, and the last number of bits of ciphertext are used as the code. A 16- or 32-bit code is typical.

**Secure Hash Functions**

An alternative to the message authentication code is the one-way hash function. As with the message authentication code, a hash function accepts a variable-size message *M* as input and produces a fixed-size message digest H(*M*) as output (Figure 2.5). Unlike the MAC, a hash function does not also take a secret key as input. To authenticate a message, the message digest is sent with the message in such a way that the message digest is authentic.

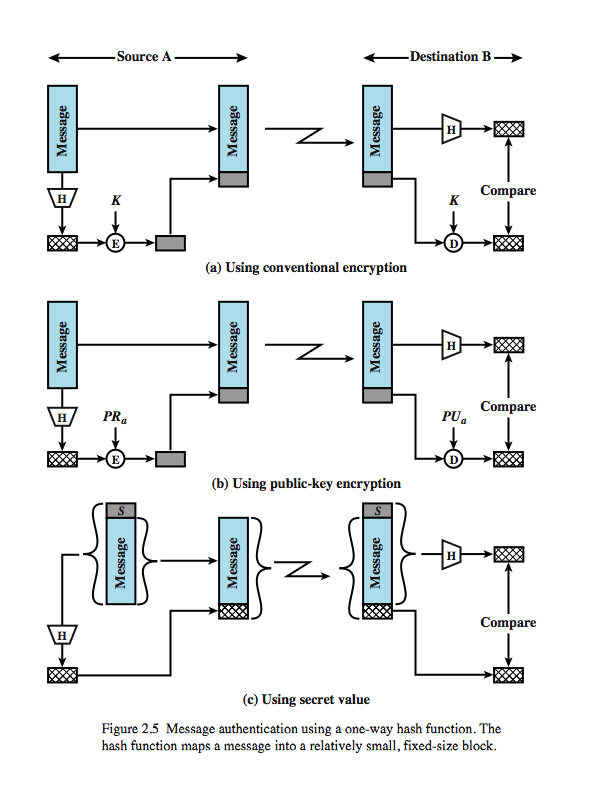
**Message Authentication**

Figure 2.6 illustrates three ways in which the message can be authenticated. The message digest can be encrypted using conventional encryption (part a); if it is assumed that only the sender and receiver share the encryption key, then authenticity is assured. The message can also be encrypted using public-key encryption (part b); this is explained later. The public-key approach has two advantages: it provides a digital signature as well as message authentication; and it does not require the distribution of keys to communicating parties. These two approaches have an advantage over approaches that encrypt the entire message in that less computation is required. Nevertheless, there has been interest in developing a technique that avoids encryption altogether. Part c shows a technique that uses a hash function but no encryption for message authentication. This technique assumes that two communicating parties, say A and B, share a common secret value *SAB*. When A has a message to send to B, it calculates the hash function over the concatenation of the secret value and the message: *MDM* = H(*SAB*||*M*). It then sends [*M*||*MDM*] to B. Because B possesses *SAB*, it can recompute H(*SAB*||*M*) and verify *MDM*. Because the secret value itself is not sent, it is not possible for an attacker to modify an intercepted message. As long as the secret value remains secret, it is also not possible for an attacker to generate a false message.

**Hash Function Requirements**

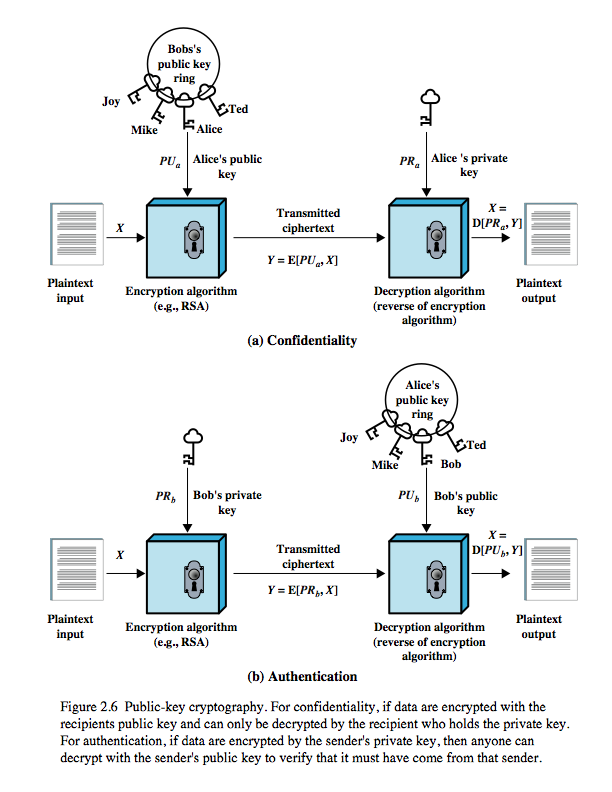
1. applied to any size data
2. H produces a fixed-length output.
3. H(*x*) is relatively easy to compute for any given *x*
4. one-way property computationally infeasible to find *x* such that H(*x*) = *h*
5. weak collision resistance computationally infeasible to find *y* ≠ *x* such tha H(*y*) = H(*x*)
6. strong collision resistance Computationally infeasible to find any pair (*x*, *y*) such that H(*x*) = H(*y*)

The purpose of a hash function is to produce a "fingerprint" of a file, message, or other block of data. To be useful for message authentication, a hash function H must have the properties listed here. The first three properties are requirements for the practical application of a hash function to message authentication. The fourth property is the one-way property: it is easy to generate a code given a message, but virtually impossible to generate a message given a code. This property is important if the authentication technique involves the use of a secret value (such as shown in Figure 2.5c). The fifth property guarantees that it is impossible to find an alternative message with the same hash value as a given message. This prevents forgery when an encrypted hash code is used (as in Figures 2.5a and b). A hash function that satisfies the first five properties in the preceding list is referred to as a weak hash function. If the sixth property is also satisfied, then it is referred to as a strong hash function. The sixth property protects against a sophisticated class of attack known as the birthday attack. In addition to providing authentication, a message digest also provides data integrity. It performs the same function as a frame check sequence: if any bits in the message are accidentally altered in transit, the message digest will be in error.

**Hash Functions**

As with symmetric encryption, there are two approaches to attacking a secure hash function: cryptanalysis and brute-force attack. As with symmetric encryption algorithms, cryptanalysis of a hash function involves exploiting logical weaknesses in the algorithm. The strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm.

If strong collision resistance is required (and this is desirable for a general-purpose secure hash code), then the value 2*n*/2 determines the strength of the hash code against brute-force attacks. Oorschot and Wiener presented a design for a $10 million collision search machine for MD5, which has a 128-bit hash length, that could find a collision in 24 days. Thus a 128-bit code may be viewed as inadequate. With a hash length of 160 bits, the same search machine would require over four thousand years to find a collision. With today's technology, the time would be much shorter, so that 160 bits now appears suspect.

In recent years, the most widely used hash function has been the Secure Hash Algorithm (SHA). SHA was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993. When weaknesses were discovered in SHA, a revised version was issued as FIPS 180-1 in 1995 and is generally referred to as SHA-1. SHA-1 produces a hash value of 160 bits. In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512. These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1. In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on the other SHA versions by 2010.

**Public Key Encryption**

Public-key encryption, first publicly proposed by Diffie and Hellman in 1976 is the first truly revolutionary advance in encryption in literally thousands of years. Public-key algorithms are based on mathematical functions rather than on simple operations on bit patterns. More important, public-key cryptography is **asymmetric**, involving the use of two separate keys, in contrast to the symmetric conventional encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication. A public-key encryption scheme has six ingredients, as shown here in Figure 2.6a:

• **Plaintext:** the readable message or data that is fed into the algorithm as input.

• **Encryption algorithm:** performs various transformations on the plaintext.

• **Public and private key:** a pair of keys selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the encryption algorithm depend on the public or private key that is provided as input.

• **Ciphertext:** the scrambled message produced as output that depends on the plaintext and key. For a given message, two different keys produce two different ciphertexts.

• **Decryption algorithm:** takes ciphertext and key to produces the original plaintext.

As the names suggest, the public key of the pair is made public for others to use, while the private key is known only to its owner. A public-key cryptographic algorithm relies on one key for encryption and a different but related key for decryption. All participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed. As long as a user protects his or her private key, incoming communication is secure.

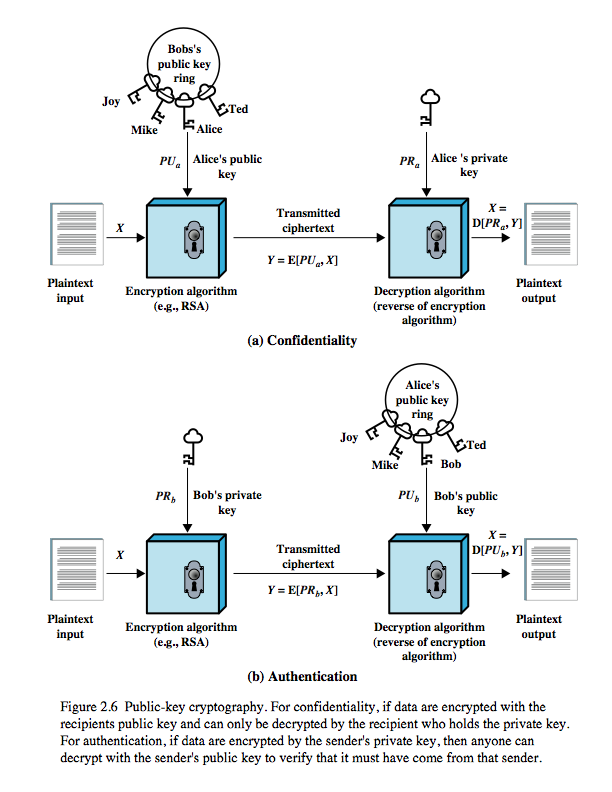
**Public Key Authentication**

Figure 2.7b shown here illustrates another mode of operation of public-key cryptography, where a user encrypts data using his or her own private key. Anyone who knows the corresponding public key will then be able to decrypt the message. This is directed toward providing authentication and/or data integrity. If a user is able to successfully recover the plaintext from Bob’s ciphertext using Bob’s public key, this indicates that only Bob could have encrypted the plaintext, thus providing authentication. Further, no one but Bob would be able to modify the plaintext because only Bob could encrypt the plaintext with Bob’s private key. This can be adapted to provide authentication or data integrity. Suppose that Bob wants to send a message to Alice and, although it is not important that the message be kept secret, he wants Alice to be certain that the message is indeed from him. In this case Bob could use his own private key to encrypt the message. Here the entire message is encrypted, which, although validating both author and contents, requires a great deal of storage and additional processing cost. A more efficient way of achieving the same results is to encrypt a small block of bits that is a function of the document. Such a block, called an authenticator, must have the property that it is infeasible to change the document without changing the authenticator. If the authenticator is encrypted with the sender's private key, it serves as a signature that verifies origin, content, and sequencing. A secure hash code such as SHA-1 can serve this function. It is important to emphasize that the digital signature does not provide confidentiality. That is, the message being sent is safe from alteration but not safe from eavesdropping.

**Public Key Requirements**

1. computationally easy to create key pairs
2. computationally easy for sender knowing public key to encrypt messages
3. computationally easy for receiver knowing private key to decrypt ciphertext
4. computationally infeasible for opponent to determine private key from public key
5. computationally infeasible for opponent to otherwise recover original message
6. useful if either key can be used for each role

**Public Key Algorithms**

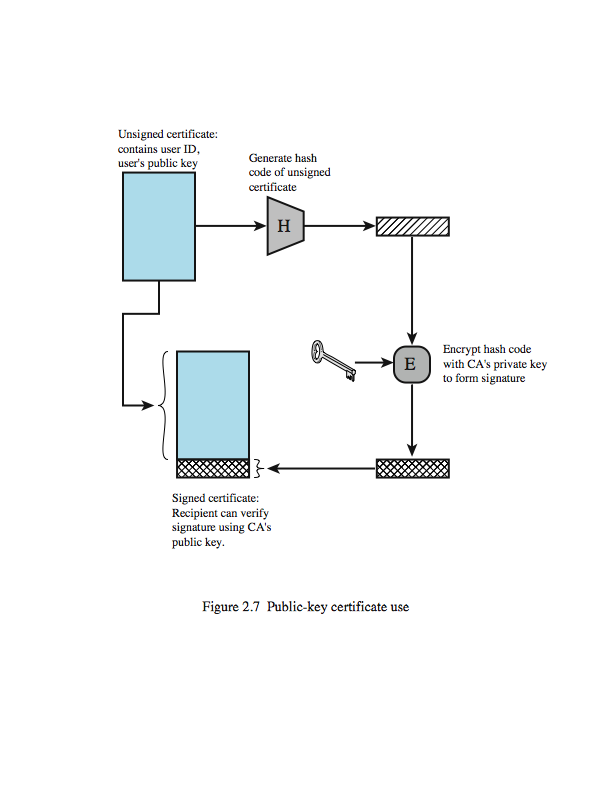
Now briefly mention the most widely-used asymmetric encryption algorithms.

One of the first public-key schemes was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT. The RSA scheme has since reigned supreme as the only widely accepted and implemented approach to public-key encryption. RSA is a block cipher in which the plaintext and ciphertext are integers between 0 and *n* – 1 for some *n*. The successful solution of the Scientific American RSA challenge, which used a public-key size (length of *n*) of 129 decimal digit, or around 428 bits, does not invalidate the use of RSA; but means that larger key sizes must be used. Currently, a 1024-bit key size (about 300 decimal digits) is considered strong enough.

The the Diffie-Hellman key exchange algorithm appeared in their seminal 1976 paper. A number of commercial products employ it. Its purpose is to enable two users to exchange a secret key securely that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of the keys.

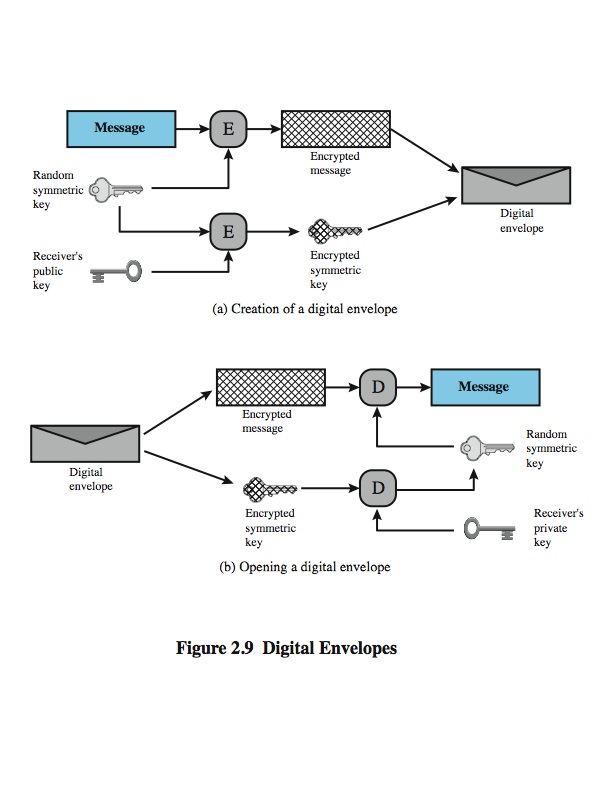
NIST published FIPS PUB 186, the Digital Signature Standard (DSS), in 1991, with revisions in 1993 & 96. The DSS makes use of the SHA-1. The DSS uses an algorithm that is designed to provide only the digital signature function.

A new alternative to RSA is elliptic curve cryptography (ECC). Its principal attraction compared to RSA is that it offers equal security for a far smaller bit size, thereby reducing processing overhead. Whilst the theory of ECC has been around for some time, only recently have products appeared and that there has been sustained cryptanalytic interest in probing for weaknesses. Thus, the confidence level in ECC is not yet as high as that in RSA. However it is appearing in new standards and products.

**Public Key Certificates**

One of the major roles of public-key encryption is to address the problem of key distribution. There are actually two distinct aspects to the use of public-key encryption in this regard: the distribution of public keys, and the use of public-key encryption to distribute secret keys. Whilst the public key is public, the problem is knowing if you actually have the public key of a specified user, and not a forgery. The solution to this problem is the public-key certificate, which consists of a public key plus a User ID of the key owner, with the whole block signed by a trusted third party. Typically, the third party is a certificate authority (CA) that is trusted by the user community, such as a government agency or a financial institution. A user can present his or her public key to the authority in a secure manner and obtain a certificate. The user can then publish the certificate. Anyone needing this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature. Figure 2.7from the text illustrates the process. One scheme has become universally accepted for formatting public-key certificates: the X.509 standard. X.509 certificates are used in most network security applications, including IP security, secure sockets layer (SSL), secure electronic transactions (SET), and S/MIME.

**Digital Envelopes**

Another application in which public-key encryption is used to protect a symmetric key is the digital envelope, which can be used to protect a message without needing to first arrange for sender and receiver to have the same secret key. The technique is referred to as a digital envelope, which is the equivalent of a sealed envelope containing an unsigned letter. The general approach is shown here from Figure 2.9 in the text. Suppose Bob wishes to send a confidential message to Alice, but they do not share a symmetric secret key. Bob does the following:

**1.** Prepare a message

**2.** Encrypt that message using conventional encryption with a one-time conventional session key.

**3.** Encrypt the session key using public-key encryption with Alice's public key.

**4.** Attach the encrypted session key to the message and send it to Alice.

Only Alice is capable of decrypting the session key and therefore of recovering the original message. If Bob obtained Alice's public key by means of Alice's public-key certificate, then Bob is assured that it is a valid key.

**Random Numbers**

Random numbers play an important role in the use of encryption for various network security applications, such as in the generation of: keys for public-key algorithms, stream keys in a stream cipher, for temporary session keys, and in key distribution scenarios. These applications give rise to two distinct and not necessarily compatible requirements for a sequence of random numbers: randomness and unpredictability. Traditionally, the concern in the generation of a sequence of allegedly random numbers has been that the sequence of numbers be random in some well-defined statistical sense (such as uniform distribution and independence). In applications such as reciprocal authentication and session key generation, the requirement is not so much that the sequence of numbers be statistically random but that the successive members of the sequence are unpredictable. With “true” random sequences, each number is statistically independent of other numbers in the sequence and therefore unpredictable. However, as is discussed shortly, true random numbers are not always used; rather, sequences of numbers that appear to be random are generated by some algorithm. In this latter case,care must be taken that an opponent not be able to predict future elements of the sequence on the basis of earlier elements.

**Pseudorandom verses Random Numbers**

One of the principal security requirements of a computer system is the protection of stored data. Although it is now routine for businesses to provide a variety of protections, including encryption, for information that is transmitted across networks, the Internet, or via wireless devices, once data is stored locally (referred to as *data at rest*), there is often little protection beyond domain authentication and operating system access controls. Data at rest is often routinely backed up to secondary storage such as CDROM or tape, archived for indefinite periods. Further, even when data are erased from a hard disk, until the relevant disk sectors are reused, the data is recoverable. Thus it become attractive, and indeed should be mandatory, to encrypt data at rest and combine this with an effective encryption key management scheme. Some more recent approaches for doing this include:

• **Back-end appliance:** a hardware device that sits between servers and storage systems and encrypts all data going from the server to the storage system and decrypts data going in the opposite direction.

• **Library-based tape encryption:** provided by means of a co-processor board embedded in the tape drive and tape library hardware. The co-processor encrypts data using a nonreadable key configured into the board. The tapes can then be sent off-site to a facility that has the same tape drive hardware.

• **Background laptop and PC data encryption:** software products that provide encryption that is transparent to the application and the user. Some encrypt all or designated files and folder,. others create a virtual disk locally on the hard drive or on a network storage device, with all data on the virtual disk encrypted. Various key management solutions are offered to restrict access to the owner of the data.

**Summary**