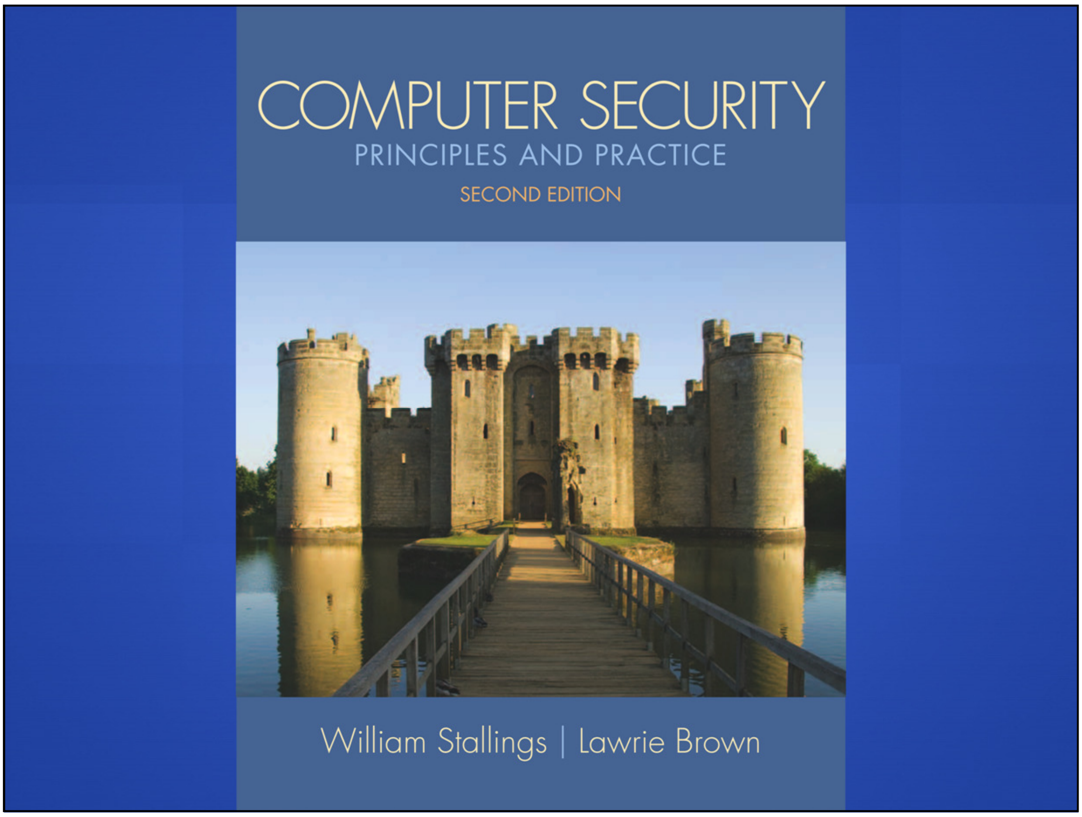
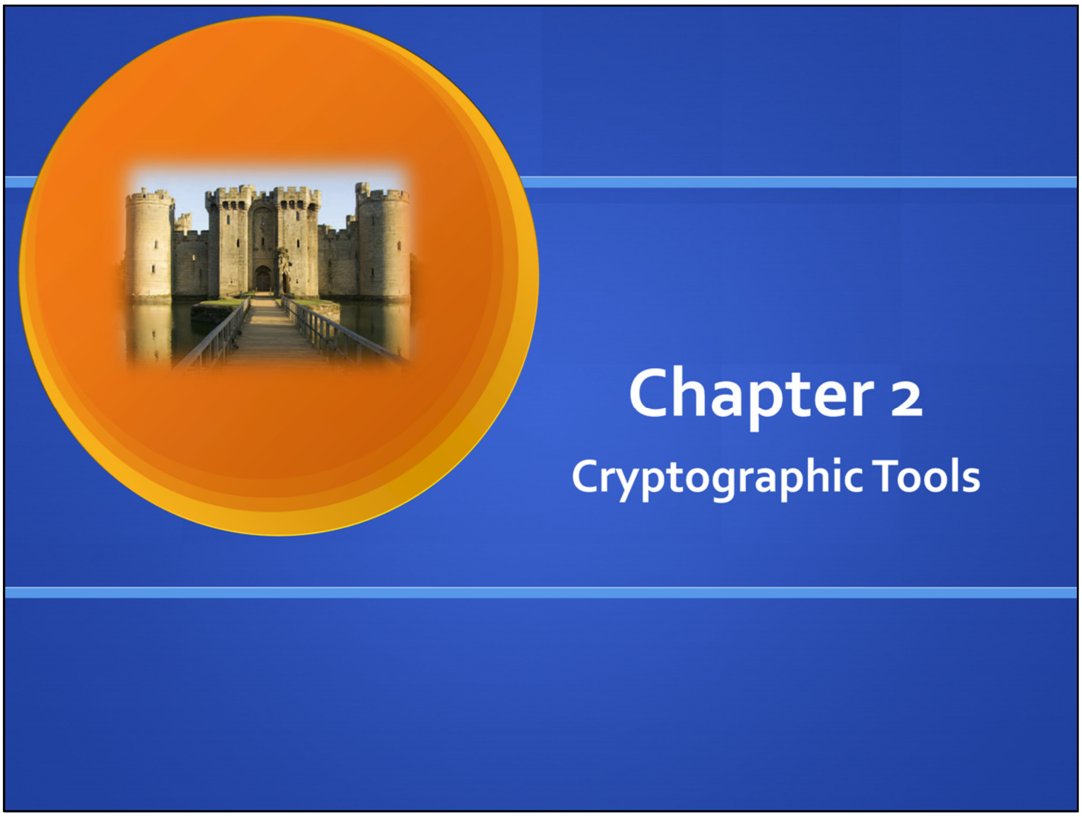


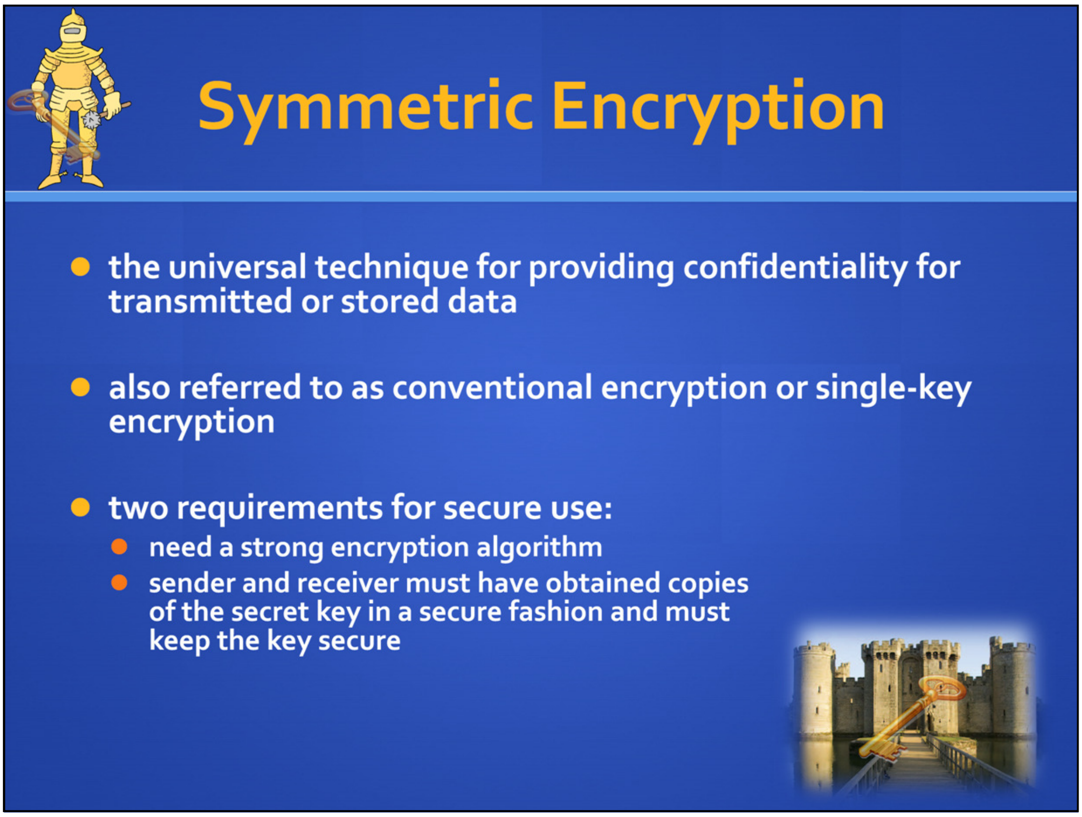
Lecture slides prepared for “Computer Security: Principles and Practice”, 2/e, by William Stallings and Lawrie Brown, Chapter 2 “Cryptographic Tools”.



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. For the technical details of the algorithms themselves, see Part Four.

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 discuss 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.



The universal technique for providing confidentiality for transmitted or stored data is symmetric encryption.

This section introduces the basic concept of symmetric encryption. This is followed by an overview of the two most important symmetric encryption algorithms: the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES), which are block encryption algorithms. Finally, this section introduces the concept of symmetric stream encryption algorithms.

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, have used symmetric encryption for secret communication. It remains the more widely used of the two types of encryption.

There are two requirements for secure use of symmetric encryption:

1. We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt ciphertext or discover the key even if he or she is in possession of a number of ciphertexts together with the plaintext that produced each ciphertext.

2. Sender and receiver must have obtained copies of the secret key in a secure
fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

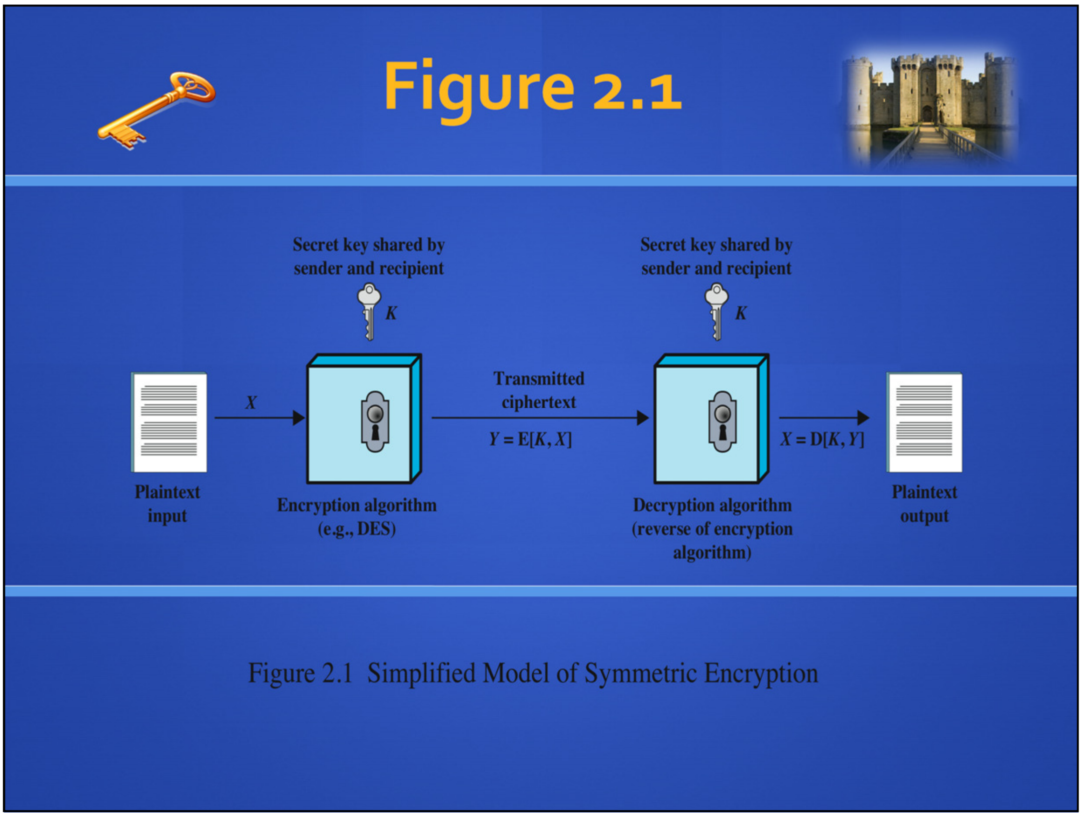


Figure 2.1 Simplified Model of Symmetric Encryption

A symmetric encryption scheme has five ingredients (Figure 2.1):

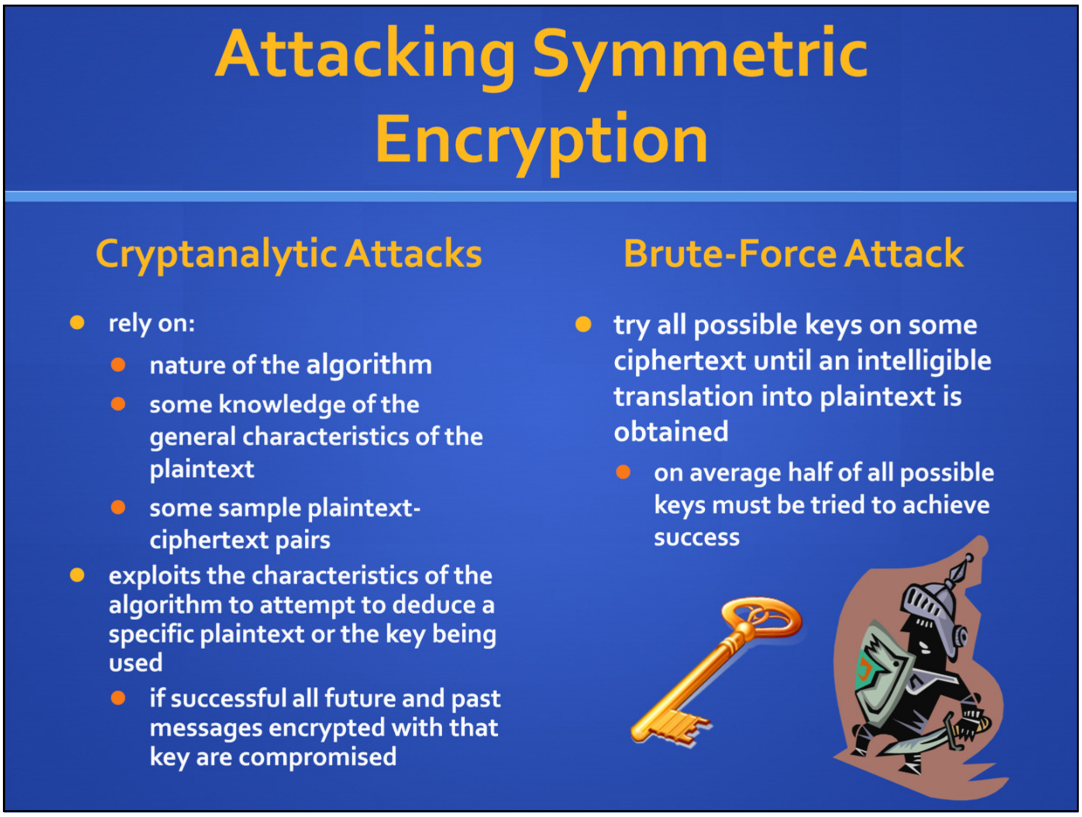
• **Plaintext: This is the original message or data that is fed into the algorithm as**
input.

• **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 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.

Table 2.1

Key Size (bits)	Number of Alternative Keys	Time Required at 1 Decryption/μs	Time Required at 10^6 Decryptions/μs
32	$2^{32} = 4.3 \times 10^9$	2^{31} μs = 35.8 minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	2^{55} μs = 1142 years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	2^{127} μs = 5.4×10^{24} years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	2^{167} μs = 5.9×10^{36} years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	2×10^{26} μs = 6.4×10^{12} years	6.4×10^6 years

Average Time Required for 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.

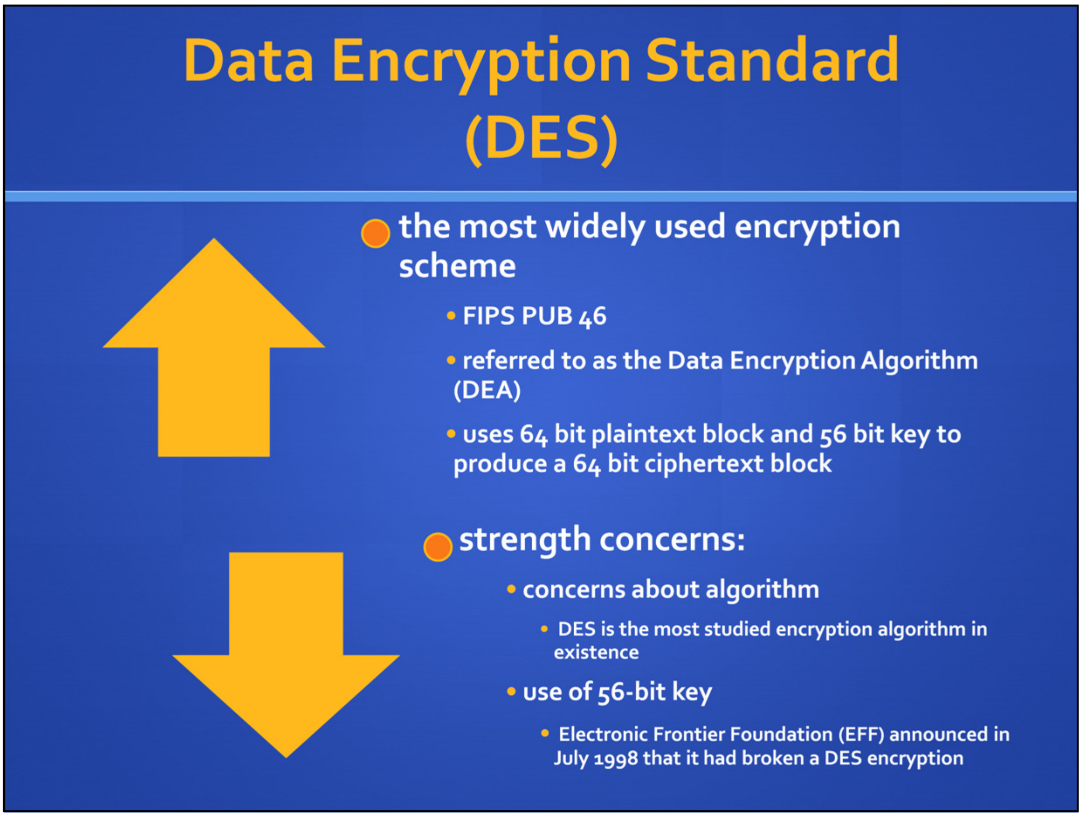
Table 2.2

	DES	Triple DES	AES
Plaintext block size (bits)	64	64	128
Ciphertext block size (bits)	64	64	128
Key size (bits)	56	112 or 168	128, 192, or 256

DES = Data Encryption Standard
AES = Advanced Encryption Standard

Comparison of Three Popular 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); see Table 2.2. This subsection provides an overview of these algorithms. Chapter 20 presents the technical details.



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 National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46).1 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. Despite numerous approaches, no one has so far reported a fatal weakness in DES.

A more serious concern is key length. With a key length of 56 bits, there are 2^{56}

possible keys, which is approximately 7.2×10^{16} keys. Thus, on the face of it, a brute-force attack appears impractical. Assuming that, on average, half the key space has to be searched, a single machine performing one DES encryption per micro second would take more than a thousand years (see Table 2.1) to break the cipher.

However, the assumption of one encryption per microsecond is overly conservative. DES 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]. And, of course, hardware prices will continue to drop as speeds increase, making DES virtually worthless.

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, although the task of recognizing English would have to be automated. If the text message has been compressed before encryption, then recognition is more difficult. And 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.

A final point: If the only form of attack that could be made on an encryption algorithm is brute force, then the way to counter such attacks is obvious: Use longer keys. To get some idea of the size of key required, let us use the EFF cracker as a basis for our estimates. The EFF cracker was a prototype and we can assume that with today's technology, a faster machine is cost effective. If we assume that a cracker can perform 1 million decryptions per µs, which is the rate used in Table 2.1, then a DES code would take about 10 hours to crack. This is a speed-up of approximately a factor of 7 compared to the EFF result

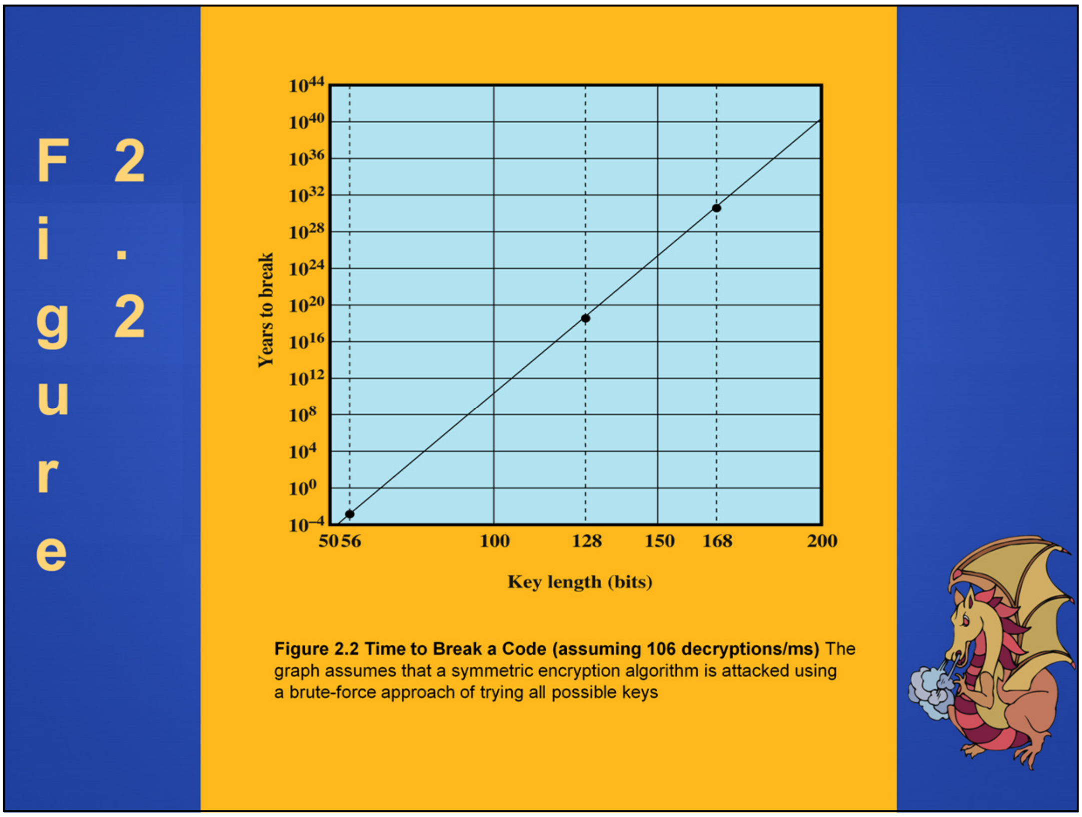


Figure 2.2 shows how long

it would take to crack a DES-style algorithm as a function of key size.2 For example,

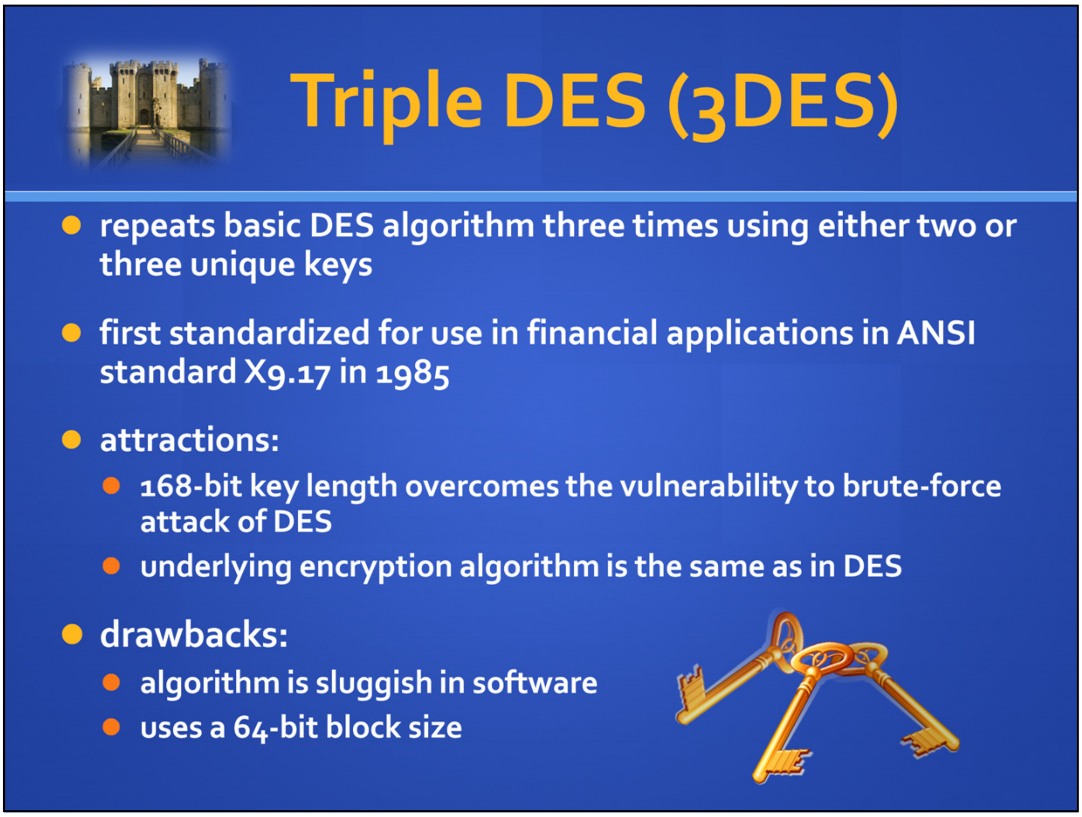
for a 128-bit key, which is common among contemporary algorithms, it would take

over 1018 years to break the code using the EFF cracker. Even if we managed to speed

up the cracker by a factor of 1 trillion (10^{12}), it would still take over 1 million years

to break the code. So a 128-bit key is guaranteed to result in an algorithm that is

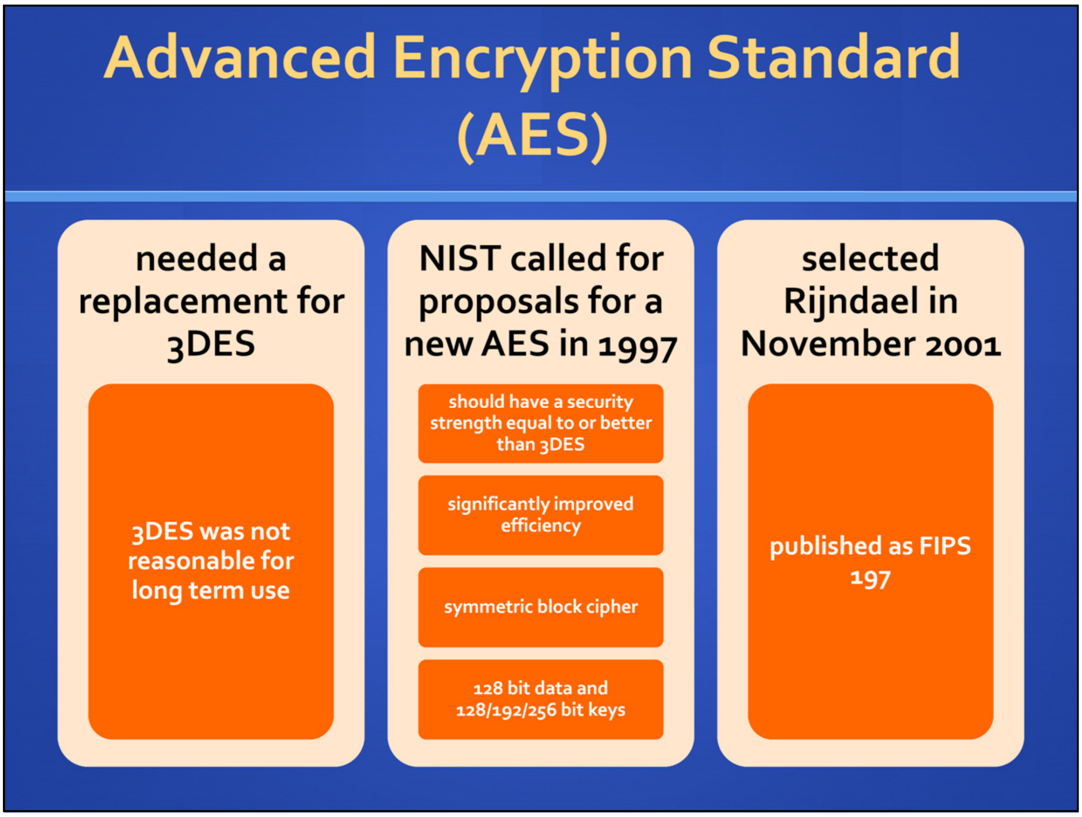
unbreakable by brute force.



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.

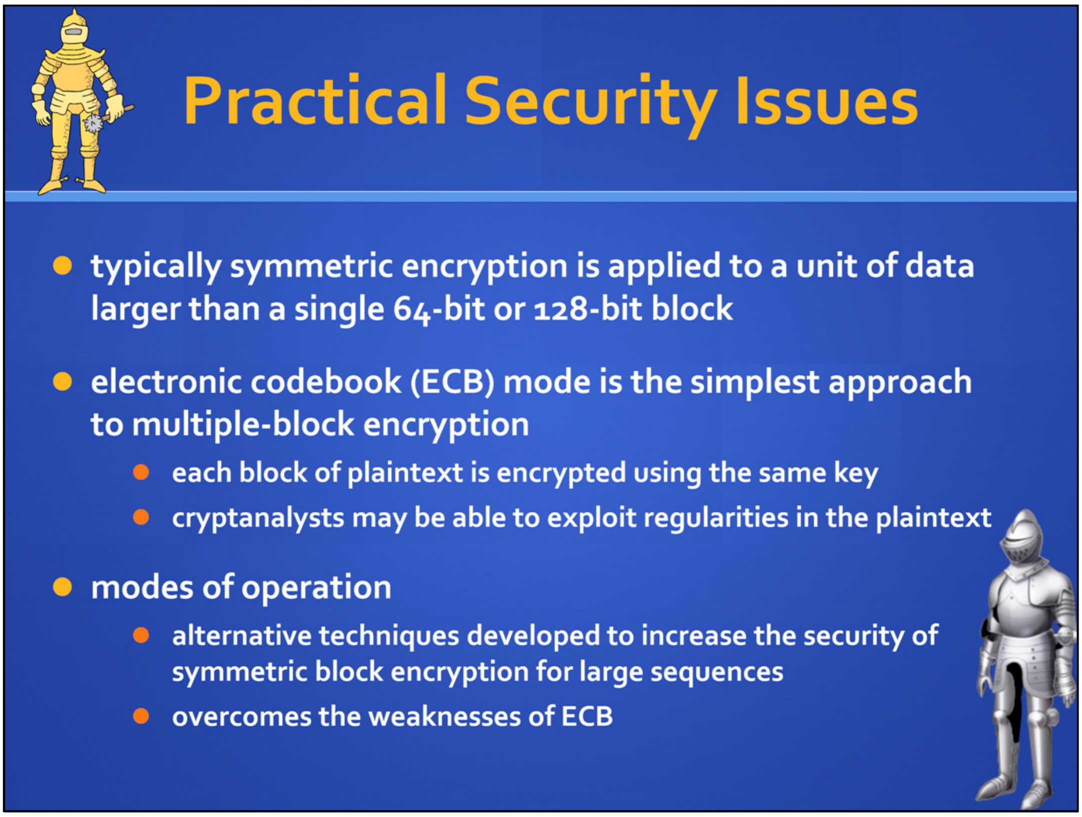
3DES has two attractions that assure its widespread use over the next few years. First, with its 168-bit key length, it overcomes the vulnerability to brute-force attack of DES. Second, the underlying encryption algorithm in 3DES is the same as in DES. This algorithm has been subjected to more scrutiny than any other encryption algorithm over a longer period of time, and no effective cryptanalytic attack based on the algorithm rather than brute force has been found. Accordingly, there is a high level of confidence that 3DES is very resistant to cryptanalysis. If security were the only consideration, then 3DES would be an appropriate choice for a standardized encryption algorithm for decades to come.

The principal drawback of 3DES is that the algorithm is relatively sluggish in software. The original DES was designed for mid-1970s hardware implementation and does not produce efficient software code. 3DES, which requires three times as many calculations as DES, is correspondingly slower. A secondary drawback is that both DES and 3DES use a 64-bit block size. For reasons of both efficiency and security, a larger block size is desirable.



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 a 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 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. AES is described in detail in Chapter 20.



Typically, symmetric encryption is applied to a unit of data larger than a single 64-bit or 128-bit block. E-mail messages, network packets, database records, and other 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*

For lengthy messages, the ECB mode may not be secure. A cryptanalyst may be able to exploit regularities in the plaintext to ease the task of decryption. For example, if it is known that the message always starts out with certain predefined fields, then the cryptanalyst may have a number of known plaintext-ciphertext pairs to work with.

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.** These modes overcome the weaknesses of ECB; each mode has its own particular advantages. This topic is explored in Chapter 20.

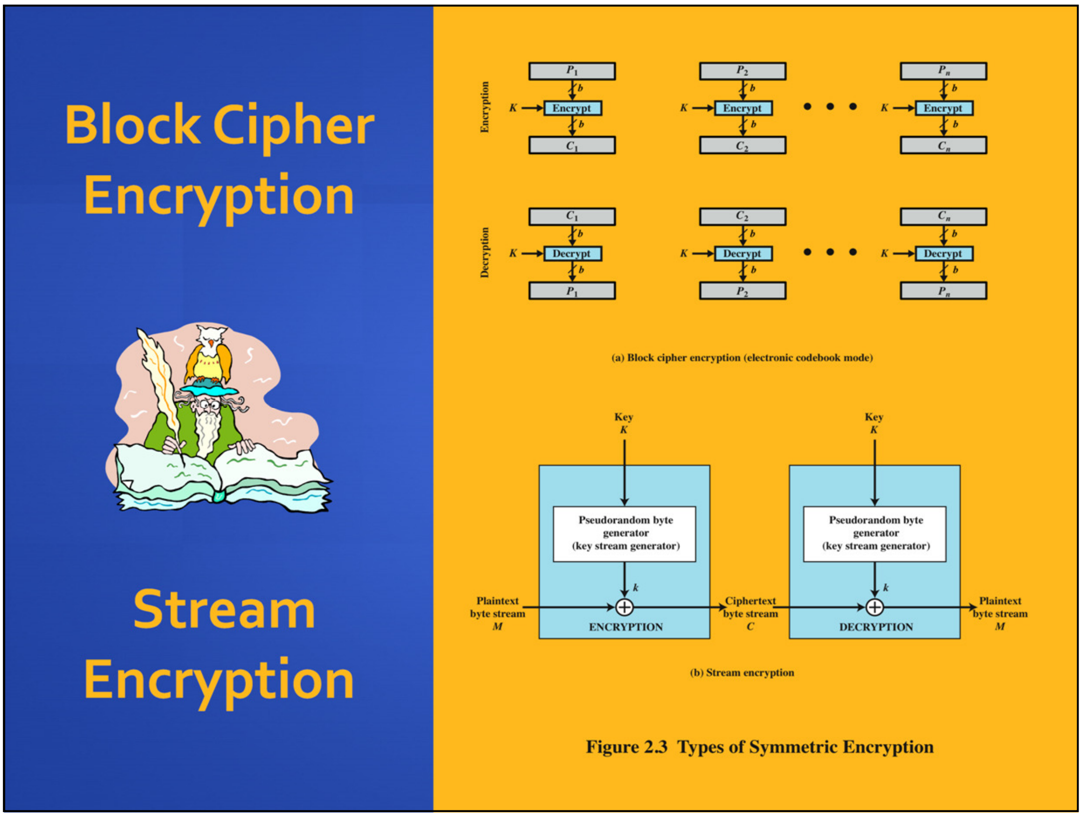


Figure 2.3a shows the

ECB mode. A plaintext of length *nb is divided into n b-bit blocks (P1, P2,c,Pn).*

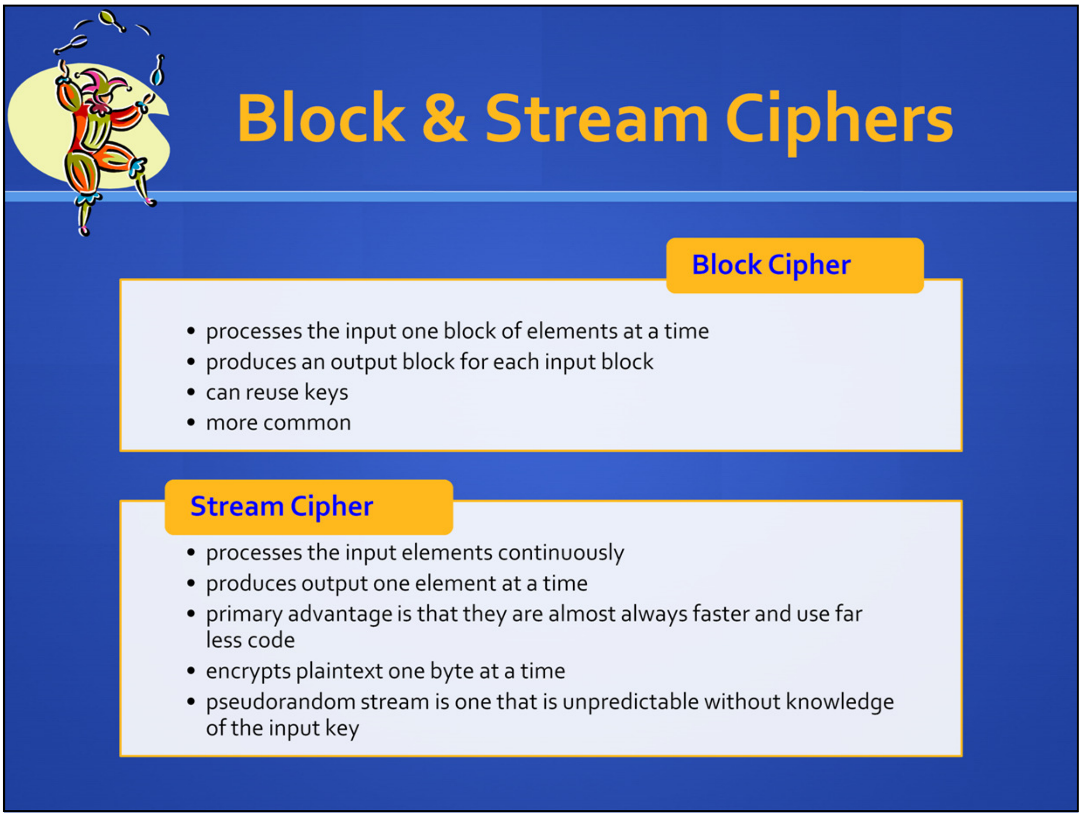
Each block is encrypted using the same algorithm and the same encryption key, to

produce a sequence of *n b-bit blocks of ciphertext (C1, C2,c,Cn).*

Figure 2.3b is a representative diagram of stream cipher structure. In this structure a key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random. A pseudorandom stream is one that is unpredictable without knowledge of the input key and which has an apparently random character (see Section 2.5). The output of the generator, called a **keystream,**

is combined one byte at a time with the plaintext stream using the bitwise exclusive-

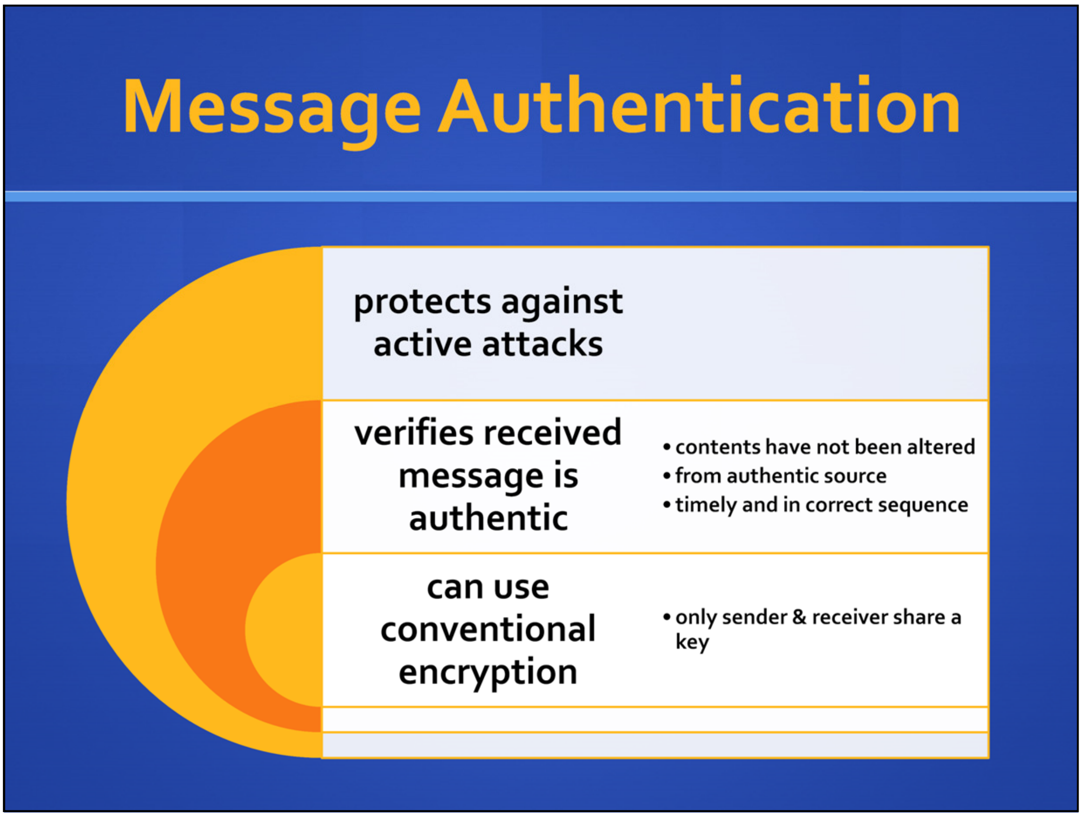
OR (XOR) operation.



A *block cipher processes the input one block of elements at a time, producing an* output block for each input block. A *stream cipher processes the input elements* continuously, producing output one element at a time, as it goes along. Although block ciphers are far more common, there are certain applications in which a stream cipher is more appropriate. Examples are given subsequently in this book.

A typical stream cipher encrypts plaintext one byte at a time, although a stream cipher may be designed to operate on one bit at a time or on units larger than a byte at a time. Figure 2.3b is a representative diagram of stream cipher structure. In this structure a key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random. A pseudorandom stream is one that is unpredictable without knowledge of the input key and which has an apparently random character (see Section 2.5). The output of the generator, called a **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. For applications that require encryption/decryption of a stream of data, such as over a data communications channel or a browser/Web link, a stream cipher might be the better alternative. For applications that deal with blocks of data, such as file transfer, e-mail, and database, block ciphers may be more appropriate. However, either type of cipher can be used in virtually any application.

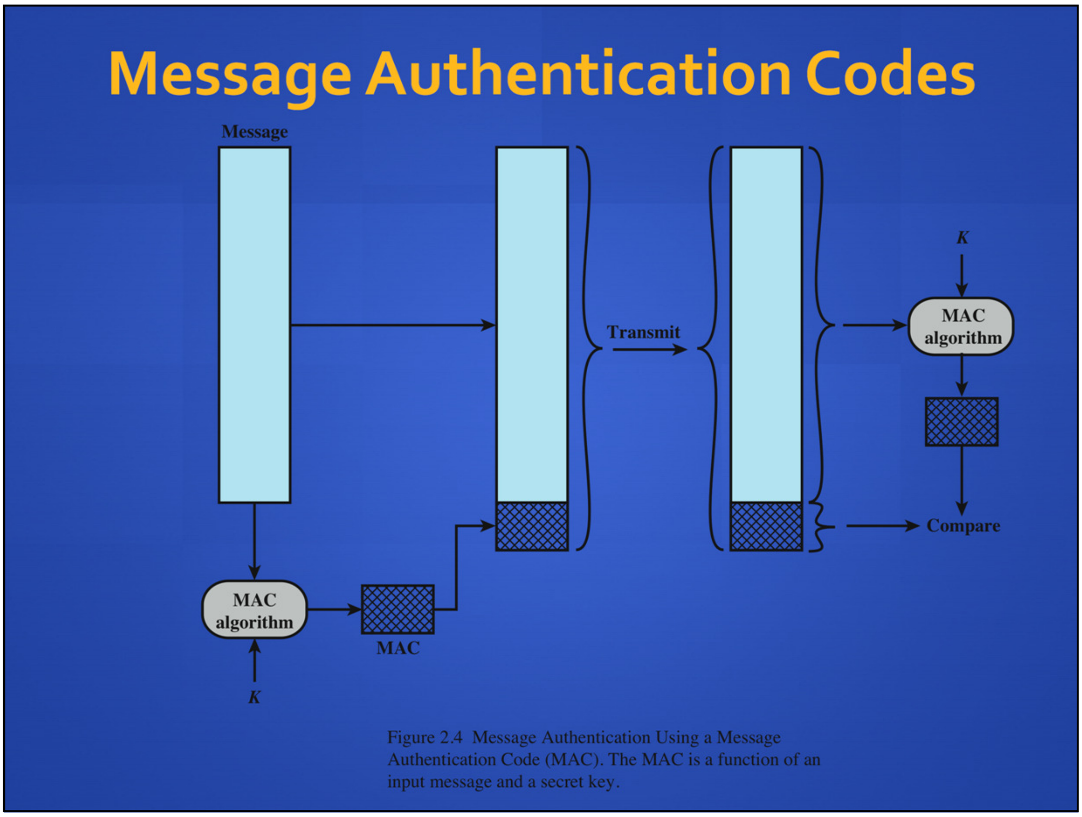


Encryption protects against passive attack (eavesdropping). A different requirement
is to protect against active attack (falsification of data and transactions). Protection
against such attacks is known as message or data authentication.

A message, file, document, or other collection of data is said to be authentic
when it is genuine and came from its alleged source. Message or data authentication
is a procedure that allows communicating parties to verify that received or stored
messages are authentic. The two important aspects are to verify 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 (it has not been artificially delayed and replayed)
and sequence relative to other messages flowing between two parties. All of these
concerns come under the category of data integrity as described in Chapter 1.

It would seem possible to perform authentication simply by the use of symmetric 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, provided the receiver can recognize a valid message. 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.

In fact, symmetric encryption alone is not a suitable tool for data authentication. To give one simple example, in the ECB mode of encryption, if an attacker reorders the blocks of ciphertext, then each block will still decrypt successfully. However, the reordering may alter the meaning of the overall data sequence. Although sequence numbers may be used at some level (e.g., each IP packet), it is typically not the case that a separate sequence number will be associated with each *b-bit block of plaintext. Thus, block reordering is a threat.*



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

complex 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

(Figure 2.4). 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. If an attacker

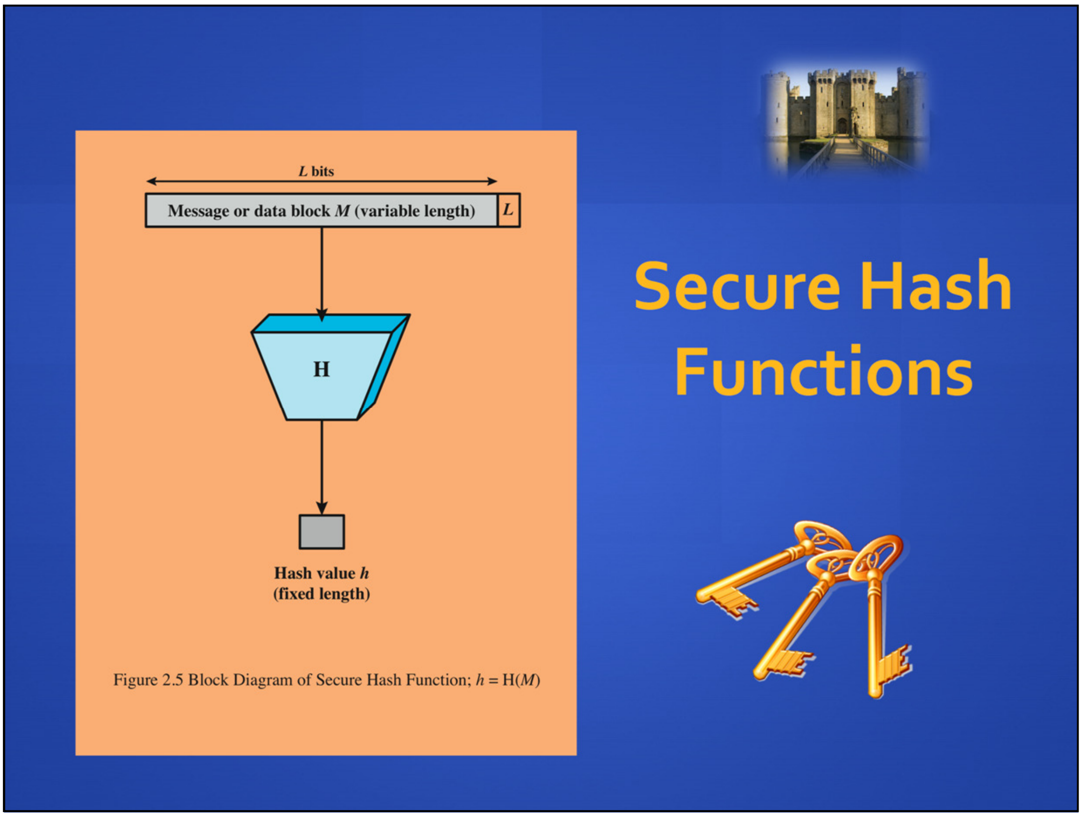
alters the message but does not alter the code, then the receiver's calculation of the code will differ from the received code. Because the attacker is assumed not to know the secret key, the attacker cannot alter the code to correspond to the alterations in the message.

2. The receiver is assured that the message is from the alleged sender. Because
no one else knows the secret key, no one else could prepare a message with a proper code.

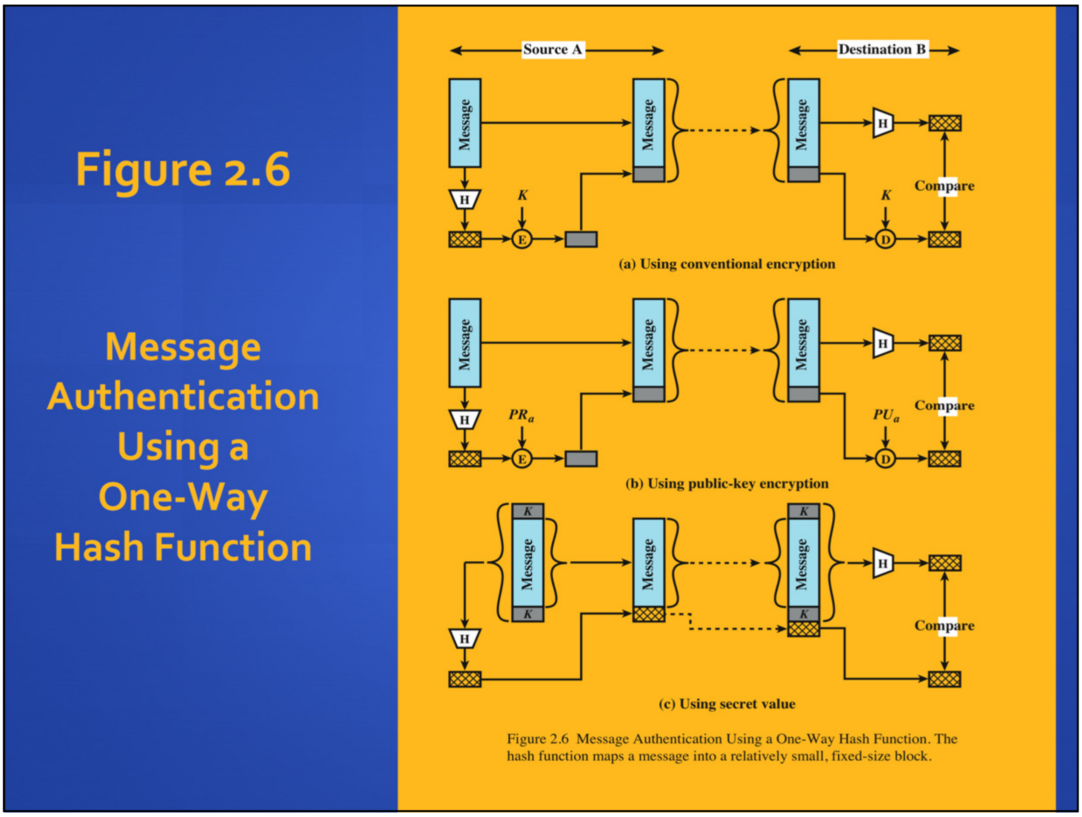
3. If the message includes a sequence number (such as is used with X.25, HDLC,
and TCP), then the receiver can be assured of the proper sequence, because an attacker cannot successfully alter the sequence number.

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.

The process just described is similar to encryption. One difference is that the authentication algorithm need not be reversible, as it must for decryption. It turns out that because of the mathematical properties of the authentication function, it is less vulnerable to being broken than encryption.



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). Typically, the message is padded out to an integer multiple*
of some fixed length (e.g., 1024 bits) and the padding includes the value of the length
of the original message in bits. The length field is a security measure to increase the
difficulty for an attacker to produce an alternative message with the same hash value.



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. Figure 2.6 illustrates three ways in which the message can be authenticated using a hash code. The message digest can be encrypted using symmetric encryption (part a); if it is assumed that only the sender and receiver share the encryption key, then authenticity is assured. The
message digest can also be encrypted using public-key encryption (part b); this is explained in Section 2.3. 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. But an even more common approach is
the use of a technique that avoids encryption altogether. Several reasons for this interest are pointed out in [TSUD92]:

• Encryption software is quite slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system.

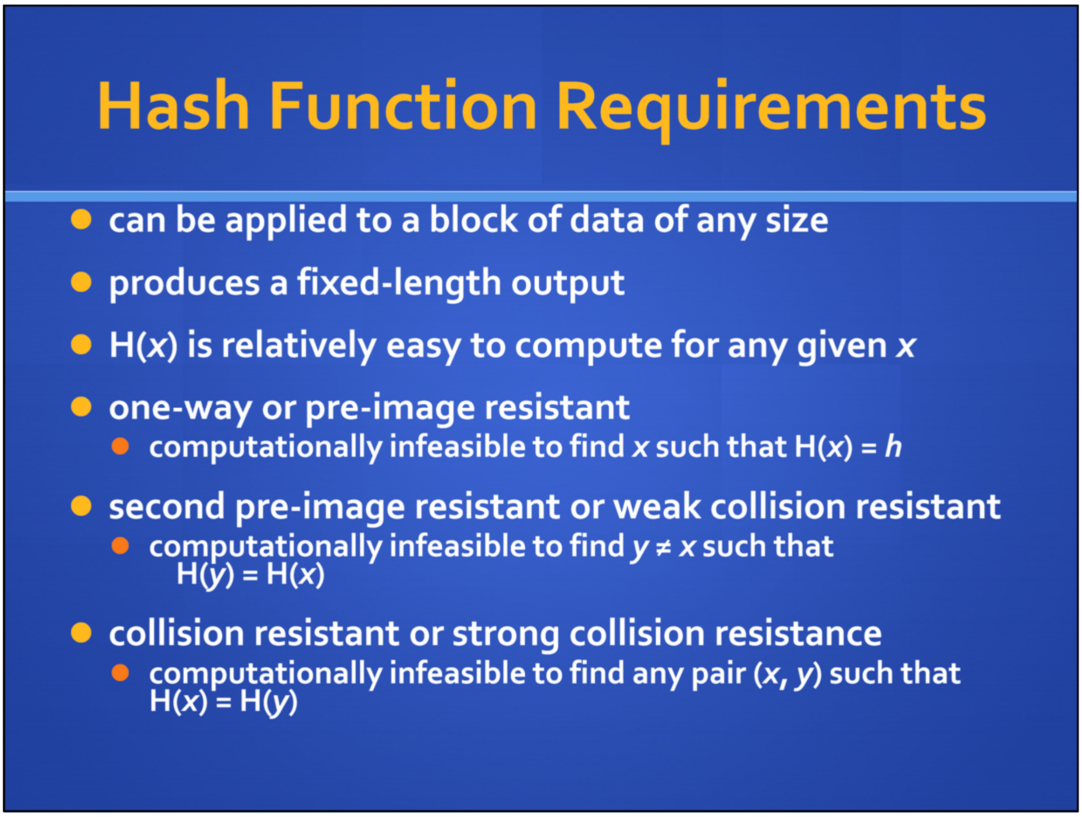
• Encryption hardware costs are non-negligible. Low-cost chip implementations of DES are available, but the cost adds up if all nodes in a network must have this capability.

• Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead.

• An encryption algorithm may be protected by a patent.

Figure 2.6c shows a technique that uses a hash function but no encryption for message authentication. This technique, known as a keyed hash MAC, assumes that two communicating parties, say A and B, share a common secret key *K.* This secret key is incorporated into the process of generating a hash code. In the approach illustrated in Figure 2.6c, when A has a message to send to B, it calculates the hash function over the concatenation of the secret key and the message: *MDM = H(KMK).6 It then sends [MMDM] to B. Because B possesses K, it can* recompute H(*K7M7K) and verify MDM. Because the secret key itself is not sent, it* should not be possible for an attacker to modify an intercepted message. As long as the secret key remains secret, it should not be possible for an attacker to generate a false message.

Note that the secret key is used as both a prefix and a suffix to the message. If the secret key is used as either only a prefix or only a suffix, the scheme is less secure. This topic is discussed in Chapter 21. Chapter 21 also describes a scheme known as HMAC, which is somewhat more complex than the approach of Figure 2.6c and which has become the standard approach for a keyed hash MAC.



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 following properties:

1. H can be applied to a block of data of any size.

2. H produces a fixed-length output.

3. H(*x) is relatively easy to compute for any given x, making both hardware and*

software implementations practical.

4. For any given code *h, it is computationally infeasible to find x such that*
H(*x) h. A hash function with this property is referred to as* **one-way or preimage resistant.**

5. For any given block *x, it is computationally infeasible to find y x with*
H(*y) H(x). A hash function with this property is referred to as **second preimage*** **resistant. This is sometimes referred to as weak collision resistant.**

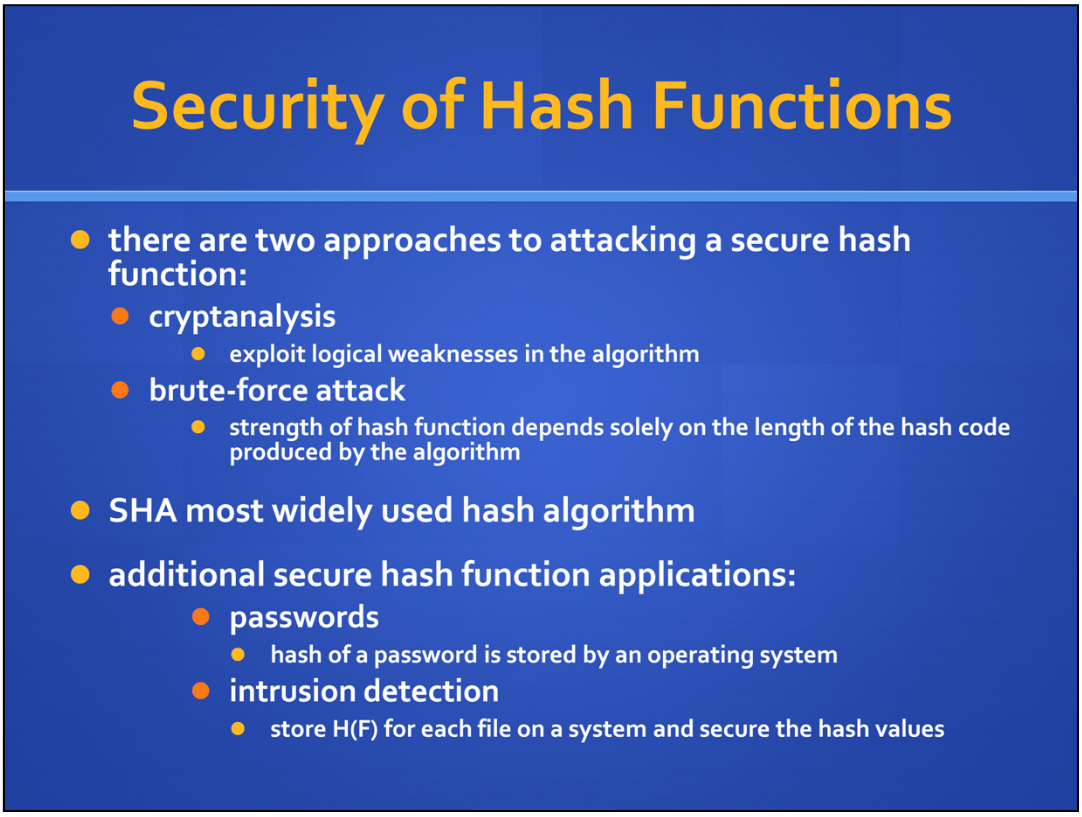
6. It is computationally infeasible to find any pair (*x, y) such that H(x) H(y).*
A hash function with this property is referred to as **collision resistant. This is** sometimes referred to as **strong collision resistant.**

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 (Figure 2.6c). The secret value itself is not sent; however, if the hash function is not one way, an attacker can easily discover the secret value: If the attacker can observe or intercept a transmission, the attacker obtains the message *M and the hash code MDM H(SAB || M). The attacker* then inverts the hash function to obtain S*AB || M H-1(MDM). Because the attacker now* has both *M and SAB || M, it is a trivial matter to recover SAB.*

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 (Figures 2.6a and b). If this property were not true, an attacker would be capable of the following sequence: First, observe or intercept a message plus its encrypted hash code; second, generate an unencrypted hash code from the message; third, generate an alternate message with the same hash code.

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. A strong hash function protects against an attack in which one party generates a message for another party to sign. For example, suppose Bob gets to write an IOU message, send it to Alice, and she signs it. Bob finds two messages with the same hash, one of which requires Alice to pay a small amount and one that requires a large payment. Alice signs the first message and Bob is then able to claim that the second message is authentic.



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.

For a hash code of length *n,*

the level of effort required is proportional to the following:

Preimage resistant 2*n*

Second preimage resistant 2*n*

Collision resistant 2*n/2*

If 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. Van Oorschot and Wiener [VANO94] pre sented 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.
The next step up, if a hash code is treated as a sequence of 32 bits, is a 160-bit hash
length. 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.

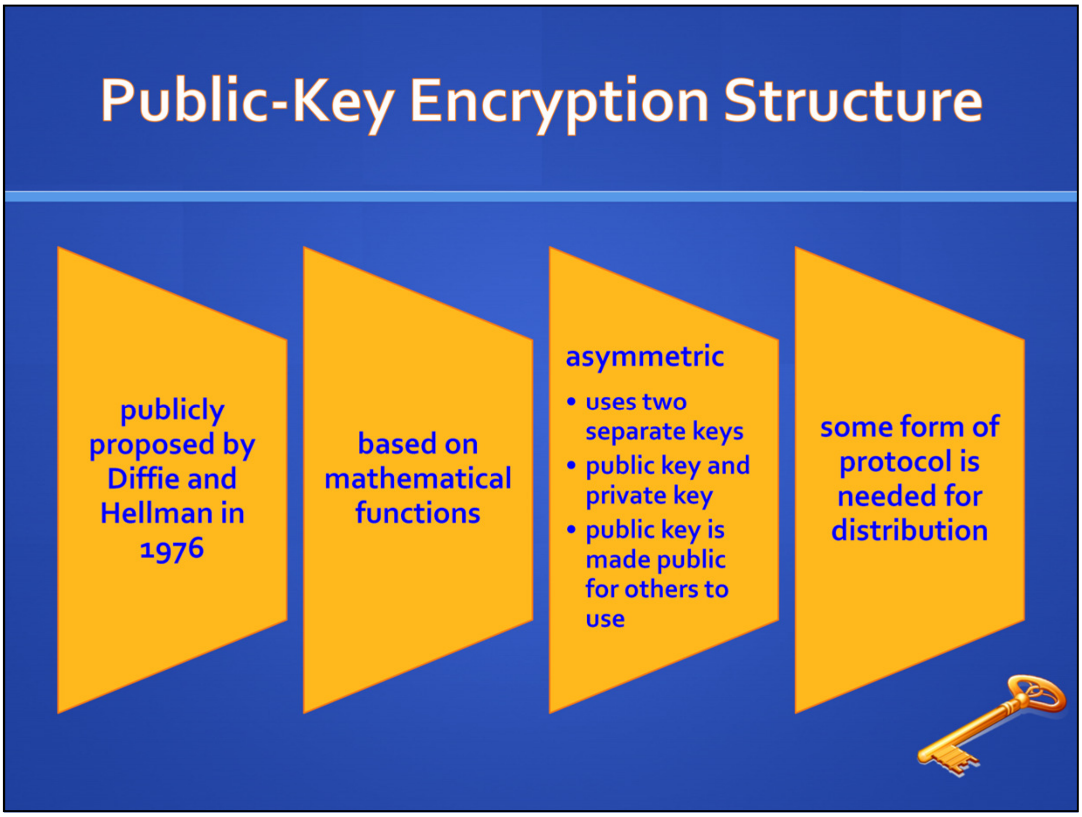
In recent years, the most widely used
hash function has been the Secure Hash Algorithm (SHA). SHA was developed
by the National Institute of Standards and Technology (NIST) and published as
a federal information processing standard (FIPS 180) in 1993. When weaknesses
were discovered in SHA, a revised 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. As discussed in Chapter
21, researchers have demonstrated that SHA-1 is far weaker than its 160-bit hash
length suggests, necessitating the move to the newer versions of SHA.

We have discussed the use of hash functions for message authentication and for the
creation of digital signatures (the latter is discussed in more detail later in this chapter).
Here are two other examples of secure hash function applications:

• **Passwords: Chapter 3 explains a scheme in which a hash of a password is**
stored by an operating system rather than the password itself. Thus, the actual
password is not retrievable by a hacker who gains access to the password file.
In simple terms, when a user enters a password, the hash of that password is

compared to the stored hash value for verification. This application requires preimage resistance and perhaps second preimage resistance.

• **Intrusion detection: Store H(F) for each file on a system and secure the hash**
values (e.g., on a CD-R that is kept secure). One can later determine if a file has been modified by recomputing H(F). An intruder would need to change F without changing H(F). This application requires weak second preimage resistance



Public-key encryption, first publicly proposed by Diffie and Hellman in 1976 [DIFF76], 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, such as are used in symmetric encryption algorithms. More important, public-key cryptography is **asymmetric, involving the use** of two separate keys, in contrast to symmetric encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication.

Before proceeding, we should first mention several common misconceptions concerning public-key encryption. One is that public-key encryption is more secure from cryptanalysis than symmetric encryption. In fact, the security of any encryption scheme depends on (1) the length of the key and (2) the computational work involved in breaking a cipher. There is nothing in principle about either symmetric or public-key encryption that makes one superior to another from the point of view of resisting cryptanalysis. A second misconception is that public-key encryption is a general- purpose technique that has made symmetric encryption obsolete. On the contrary, because of the computational overhead of current public-key encryption schemes, there seems no foreseeable likelihood that symmetric encryption will be abandoned. Finally, there is a feeling that key distribution is trivial when using public-key encryption, compared to the rather cumbersome handshaking involved with key distribution centers for symmetric encryption. For public-key key distribution, some form of protocol is needed, often involving a central agent, and the procedures involved are no simpler or any more efficient than those required for symmetric encryption.

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 general-purpose public-key cryptographic algorithm relies on one key for encryption and a different but related key for decryption.

The essential steps are the following:

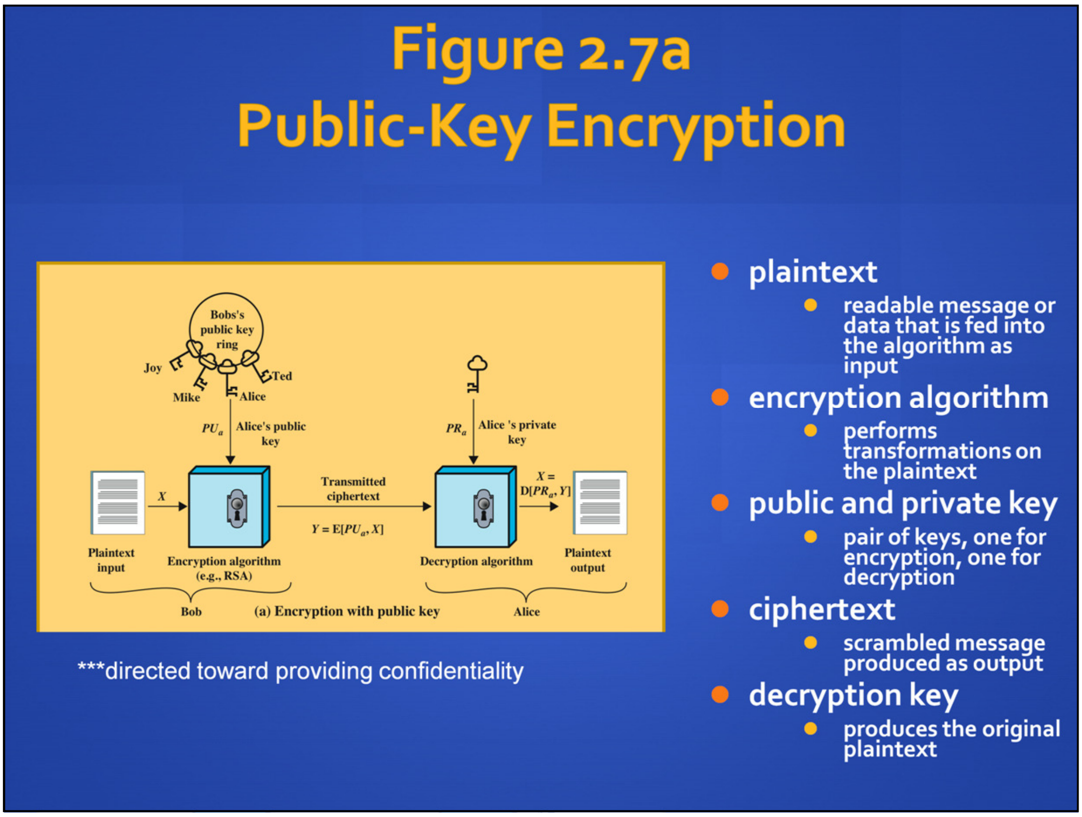
1. Each user generates a pair of keys to be used for the encryption and decryption
of messages.

2. Each user places one of the two keys in a public register or other accessible
file. This is the public key. The companion key is kept private. As Figure 2.7a suggests, each user maintains a collection of public keys obtained from others.

3. If Bob wishes to send a private message to Alice, Bob encrypts the message
using Alice's public key.

4. When Alice receives the message, she decrypts it using her private key. No
other recipient can decrypt the message because only Alice knows Alice's private key.

With this approach, 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.
At any time, a user can change the private key and publish the companion public key to replace the old public key.



A public-key encryption scheme has six ingredients (Figure 2.7a):

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

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

• **Public and private key: This is a pair of keys that have been 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: This is the scrambled message produced as output. It depends on
the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.

• **Decryption algorithm: This algorithm accepts the ciphertext and the matching**
key and 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 general-purpose public-key cryptographic algorithm relies on one key for encryption and a different but related key for decryption.

The essential steps are the following:

1. Each user generates a pair of keys to be used for the encryption and decryption
of messages.

2. Each user places one of the two keys in a public register or other accessible
file. This is the public key. The companion key is kept private. As Figure 2.7a suggests, each user maintains a collection of public keys obtained from others.

3. If Bob wishes to send a private message to Alice, Bob encrypts the message
using Alice's public key.

4. When Alice receives the message, she decrypts it using her private key. No
other recipient can decrypt the message because only Alice knows Alice's private key.

With this approach, 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.
At any time, a user can change the private key and publish the companion public key to replace the old public key.

Note that the scheme of Figure 2.7a is directed toward providing **confidentiality:** Only the intended recipient should be able to decrypt the ciphertext because only the intended recipient is in possession of the required private key. Whether in fact confidentiality is provided depends on a number of factors, including the security of the algorithm, whether the private key is kept secure, and the security of any protocol of which the encryption function is a part.

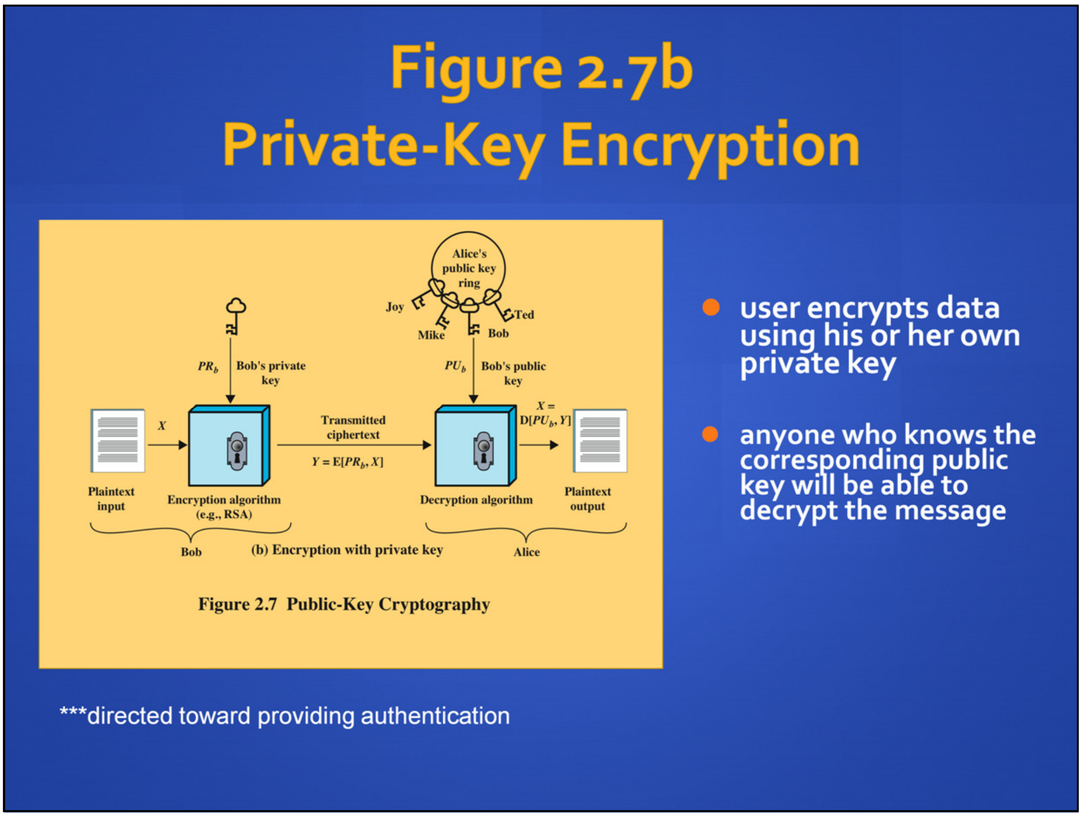


Figure 2.7b illustrates another mode of operation of public-key cryptography.
In this scheme, a user encrypts data using his or her own private key.
Anyone who knows the corresponding public key will then be able to decrypt the message.

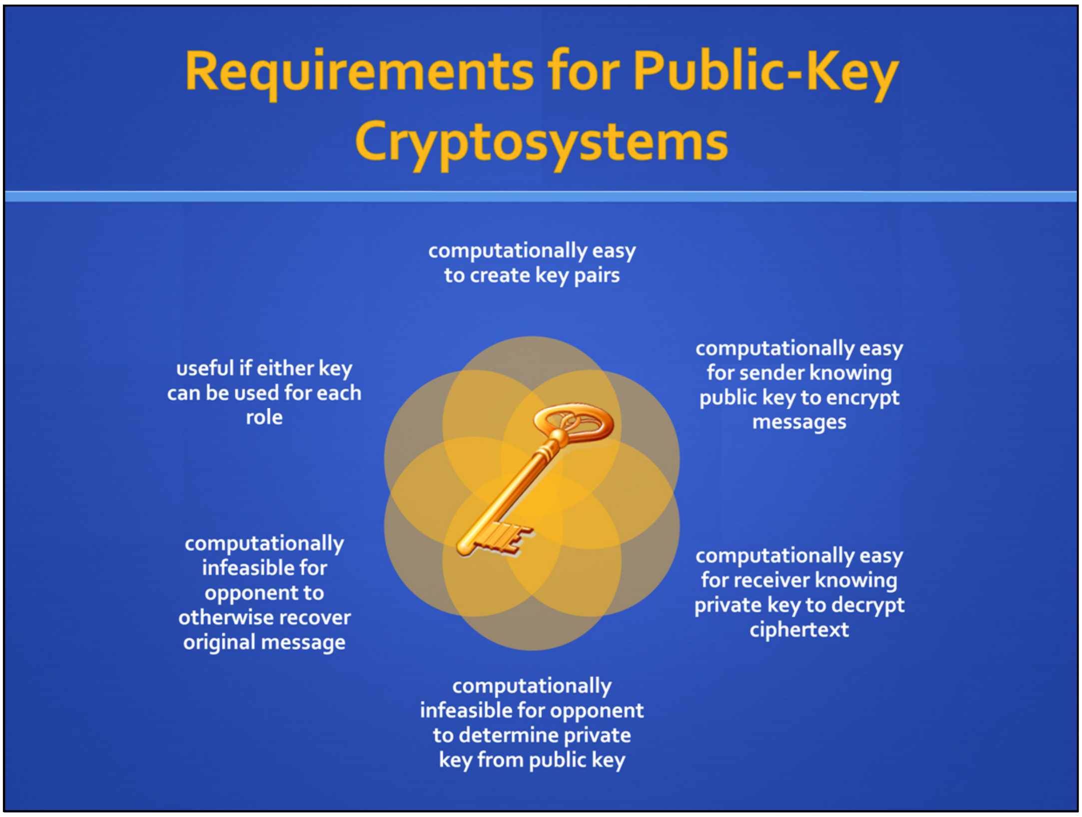
The scheme of Figure 2.7b is directed toward providing **authentication** and/or **data integrity. If a user is able to successfully recover the plaintext from**
Bob's ciphertext using Bob's public key, this indicates that only Bob could have encrypted the plaintext, thus providing authentication. Further, no one but Bob would be able to modify the plaintext because only Bob could encrypt the plaintext with Bob's private key. Once again, the actual provision of authentication or data integrity depends on a variety of factors. This issue is addressed primarily in Chapter 21, but other references are made to it where appropriate in this text.

Table 2.3

Algorithm	Digital Signature	Symmetric Key Distribution	Encryption of Secret Keys
RSA	Yes	Yes	Yes
Diffie-Hellman	No	Yes	No
DSS	Yes	No	No
Elliptic Curve	Yes	Yes	Yes

Applications for Public-Key Cryptosystems

Table 2.3 indicates the applications supported by the algorithms discussed in this section.



The cryptosystem illustrated in Figure 2.7 depends on a cryptographic algorithm based on two related keys. Diffie and Hellman postulated this system without demonstrating that such algorithms exist. However, they did lay out the conditions that such algorithms must fulfill [DIFF76]:

1. It is computationally easy for a party B to generate a pair (public key *PUb,* private key *PRb).*

2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted, *M, to generate the corresponding ciphertext:*

C = E(PUb, M)

3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message:

M = D(PRb,C) = D[PRb, E(PUb, M)]

4. It is computationally infeasible for an opponent, knowing the public key, *PUb,*

to determine the private key, *PRb.*

5. It is computationally infeasible for an opponent, knowing the public key, *PUb,*

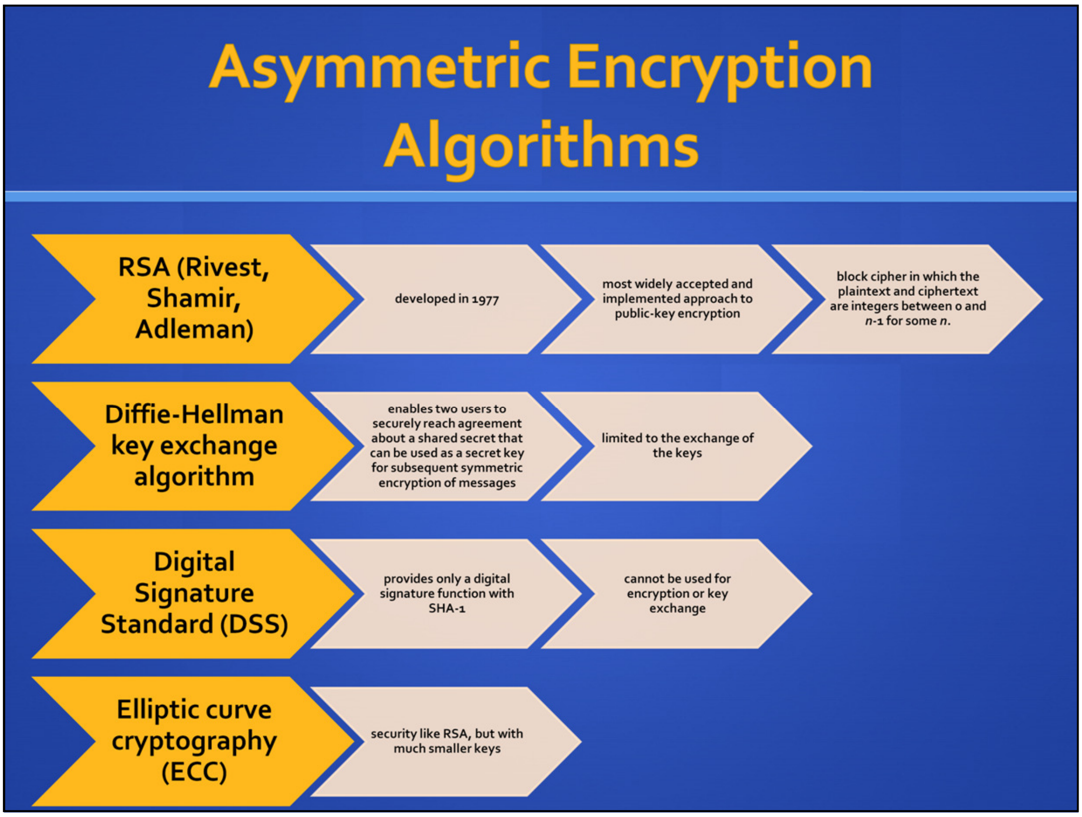
and a ciphertext, *C, to recover the original message, M.*

We can add a sixth requirement that, although useful, is not necessary for all public-key applications:

6. Either of the two related keys can be used for encryption, with the other used

for decryption.

M = D[PUb, E(PRb, M)] = D[PRb, E(PUb, M)]



RSA One of the first public-key schemes was developed in 1977 by Ron Rivest, Adi

Shamir, and Len Adleman at MIT and first published in 1978 [RIVE78]. The RSA scheme has since reigned supreme as the most 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.*

In 1977, the three inventors of RSA dared *Scientific American readers to decode* a cipher they printed in Martin Gardner's "Mathematical Games" column. They offered a $100 reward for the return of a plaintext sentence, an event they predicted might not occur for some 40 quadrillion years. In April of 1994, a group working over the Internet and using over 1600 computers claimed the prize after only eight months of work [LEUT94]. This challenge used a public-key size (length of *n) of 129 decimal* digits, or around 428 bits. This result does not invalidate the use of RSA; it simply

means that larger key sizes must be used. Cur rently, a 1024-bit key size (about 300
decimal digits) is considered strong enough for virtually all applications.

DIFFIE-HELLMAN KEY AGREEMENT The first published public-key algo rithm
appeared in the seminal paper by Diffie and Hellman that defined public-key cryptography [DIFF76] and is generally referred to as Diffie-Hellman key exchange,
or key agreement. A number of commercial products employ this key exchange technique.

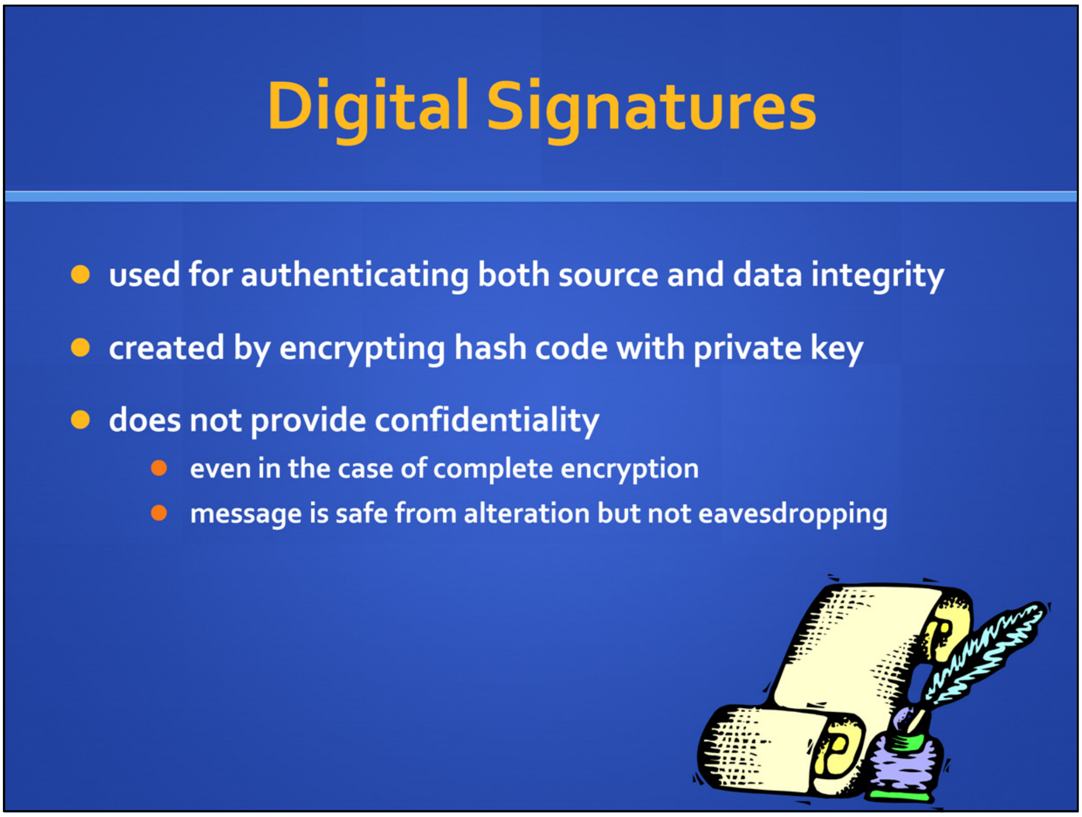
The purpose of the algorithm is to enable two users to securely reach agreement about a shared secret that can be used as a secret key for subsequent symmetric encryption of messages. The algorithm itself is limited to the exchange of the keys.

DIGITAL SIGNATURE STANDARD The National Institute of Standards and Technology
(NIST) has published Federal Information Processing Standard FIPS PUB 186, known as the Digital Signature Standard (DSS). The DSS makes use of SHA-1 and presents a new digital signature technique, the Digital Signature Algorithm (DSA). The DSS was originally proposed in 1991 and revised in 1993 in response to
public feedback concerning the security of the scheme. There was a further minor revision in 1996. The DSS uses an algorithm that is designed to provide only the digital
signature function. Unlike RSA, it cannot be used for encryption or key exchange.

ELLIPTIC CURVE CRYPTOGRAPHY The vast majority of the products and standards
that use public-key cryptography for encryption and digital signatures use RSA. The bit length for secure RSA use has increased over recent years, and this has put
a heavier processing load on applications using RSA. This burden has ramifications,
especially for electronic commerce sites that conduct large numbers of secure transactions. Recently, a competing system has begun to challenge RSA: elliptic curve cryptography (ECC). Already, ECC is showing up in standardization efforts,

including the IEEE (Institute of Electrical and Electronics Engineers) P1363 Standard for Public-Key Cryptography.

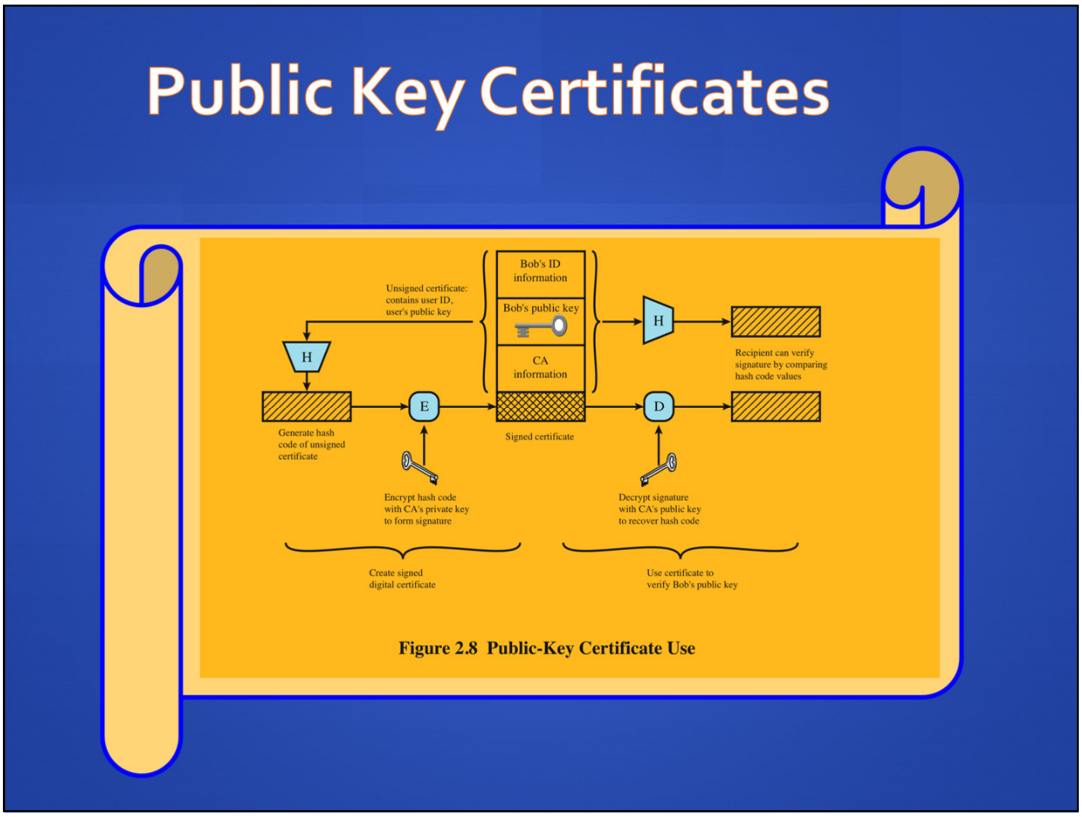
The principal attraction of ECC compared to RSA is that it appears to offer equal security for a far smaller bit size, thereby reducing processing overhead. On the other hand, although the theory of ECC has been around for some time, it is only recently that products have begun to appear and that there has been sustained cryptanalytic interest in probing for weaknesses. Thus, the confidence level in ECC is not yet as high as that in RSA.



Public-key encryption can be used for authentication, as suggested by Figure 2.6b. Suppose that Bob wants to send a message to Alice. Although it is not important that the message be kept secret, he wants Alice to be certain that the message is indeed from him. For this purpose, Bob uses a secure hash function, such as SHA-512, to generate a hash value for the message and then encrypts the hash code with his private key, creating a **digital signature. Bob sends the message with the signature attached.**

When Alice receives the message plus signature, she (1) calculates a hash value for the message; (2) decrypts the signature using Bob's public key; and (3) compares the calculated hash value to the decrypted hash value. If the two hash values match, Alice is assured that the message must have been signed by Bob. No one else has Bob's private key and therefore no one else could have created a ciphertext that could be decrypted with Bob's public key. In addition, it is impossible to alter the message without access to Bob's private key, so the message is authenticated both in terms of source and in terms of data integrity.

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. This is obvious in the case of a signature based on a portion of the message, because the rest of the message is transmitted in the clear. Even in the case of complete encryption, there is no protection of confidentiality because any observer can decrypt the message by using the sender's public key.

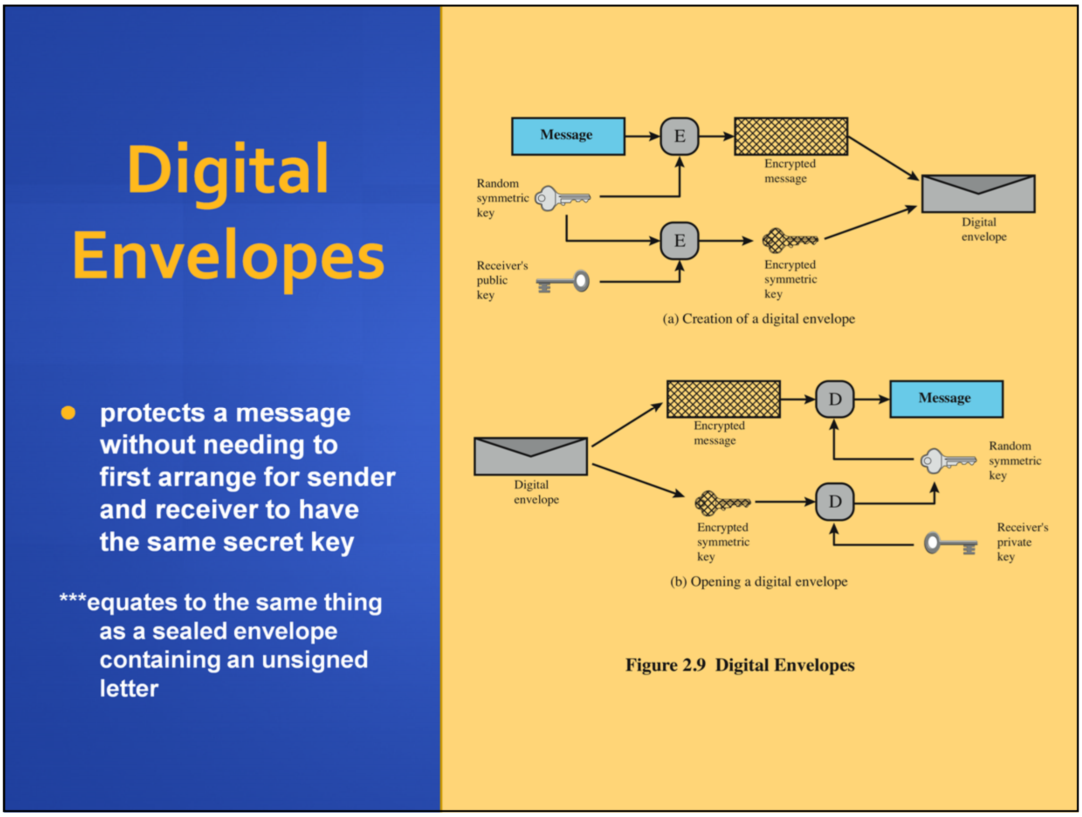


On the face of it, the point of public-key encryption is that the public key is public. Thus, if there is some broadly accepted public-key algorithm, such as RSA, any participant can send his or her public key to any other participant or broadcast the key to the community at large. Although this approach is convenient, it has a major weakness. Anyone can forge such a public announcement. That is, some user could pretend to be Bob and send a public key to another participant or broadcast such a public key. Until such time as Bob discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication.

The solution to this problem is the public-key certificate. In essence, a certificate consists of a public key plus a user ID of the key owner, with the whole block signed by a trusted third party. The certificate also includes some information about the third party plus an indication of the period of validity of the certificate.

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 signed 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 means of the attached trusted signature. Figure 2.8 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 (IPsec), Transport Layer Security (TLS), Secure Shell (SSH), and Secure/Multipurpose Internet Mail Extension (S/MIME). We examine most of these applications in Part Five.



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 in Figure 2.9. 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. Generate a random symmetric key that will be used this one time only.

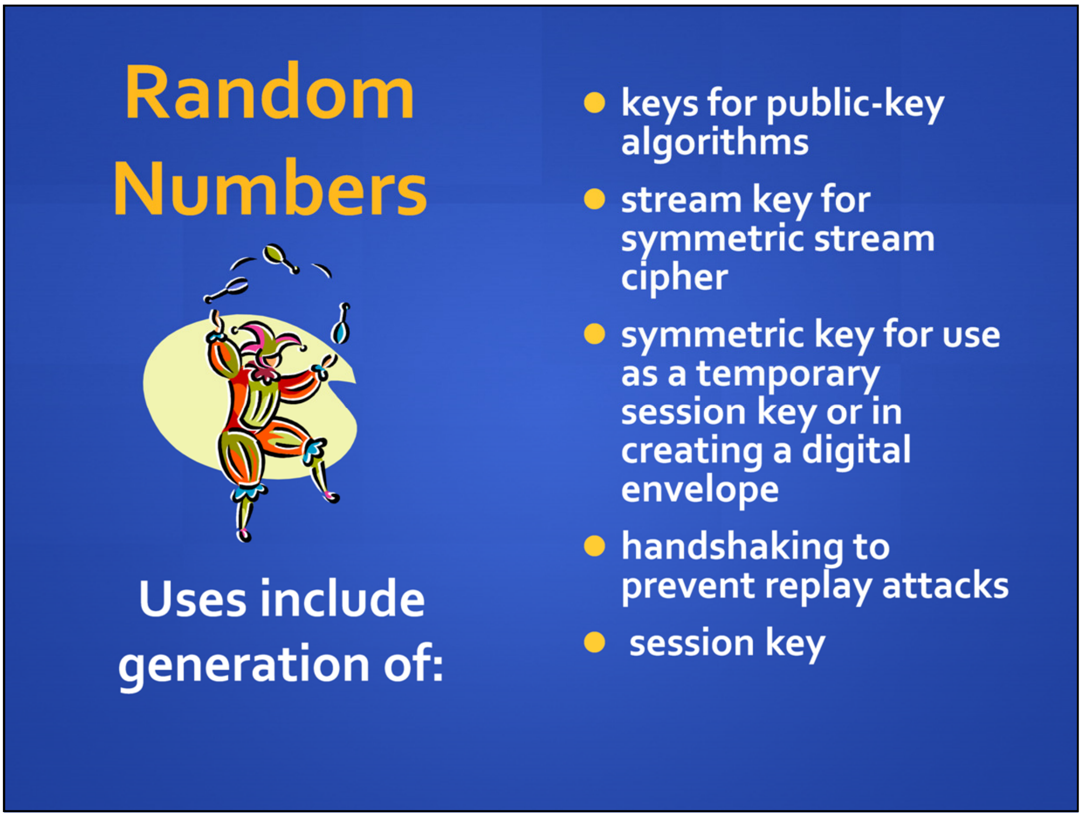
3. Encrypt that message using symmetric encryption the one-time key.

4. Encrypt the one-time key using public-key encryption with Alice's public key.

5. Attach the encrypted one-time key to the encrypted message and send it to

Alice.

Only Alice is capable of decrypting the one-time 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.



A number of network security algorithms based on cryptography make use of random numbers. For example,

• Generation of keys for the RSA public-key encryption algorithm (described in Chapter 21) and other public-key algorithms.

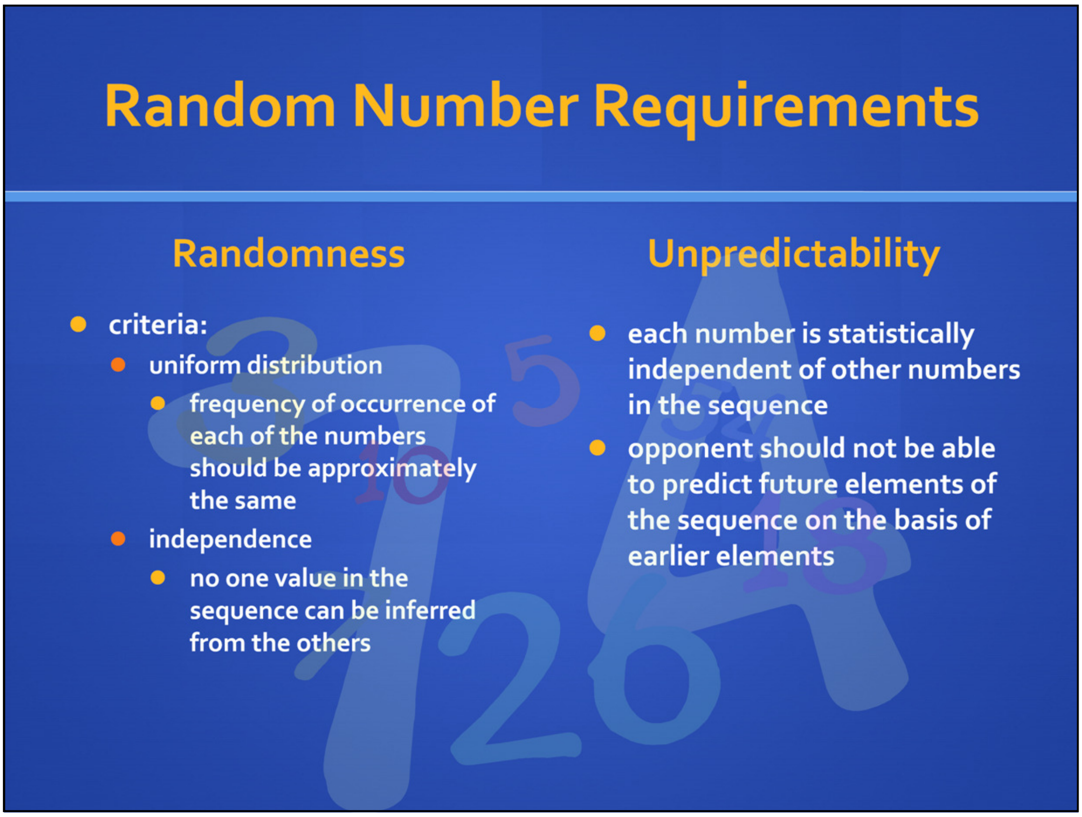
• Generation of a stream key for symmetric stream cipher.

• Generation of a symmetric key for use as a temporary session key or in creating
a digital envelope.

• In a number of key distribution scenarios, such as Kerberos (described in Chapter 23), random numbers are used for handshaking to prevent replay attacks.

• Session key generation, whether done by a key distribution center or by one of the principals.

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. The following two criteria are used to validate that a sequence of numbers is random:

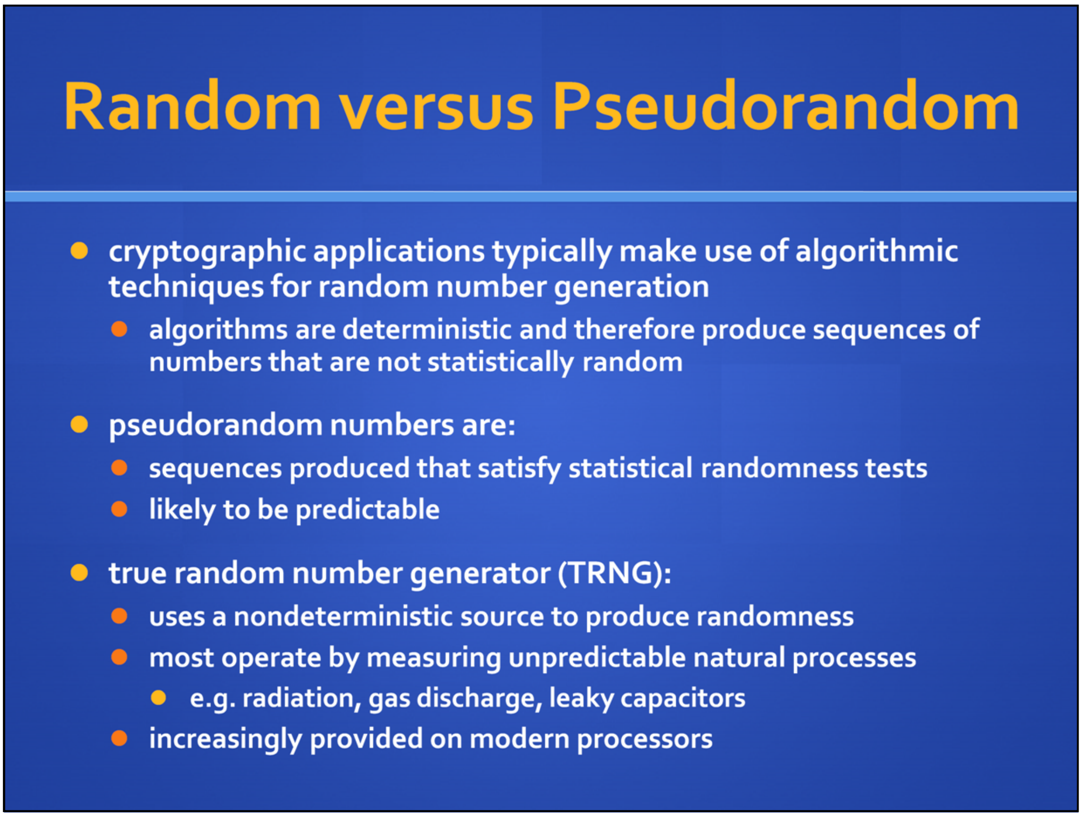
• **Uniform distribution: The distribution of numbers in the sequence should be** uniform; that is, the frequency of occurrence of each of the numbers should be approximately the same.

• **Independence: No one value in the sequence can be inferred from the others.**

Although there are well-defined tests for determining that a sequence of numbers matches a particular distribution, such as the uniform distribution, there is no such test to "prove" independence. Rather, a number of tests can be applied to demonstrate if a sequence does not exhibit independence. The general strategy is to apply a number of such tests until the confidence that independence exists is sufficiently strong.

In the context of our discussion, the use of a sequence of numbers that appear statistically random often occurs in the design of algorithms related to cryptography. For example, a fundamental requirement of the RSA public-key encryption scheme is the ability to generate prime numbers. In general, it is difficult to determine if a given large number *N is prime. A brute-force approach* would be to divide *N by every odd integer less than 1N. If N is on the order, say,* of 10150, a not uncommon occurrence in public-key cryptography, such a brute-force approach is beyond the reach of human analysts and their computers. However, a number of effective algorithms exist that test the primality of a number by using a sequence of randomly chosen integers as input to relatively simple computations. If the sequence is sufficiently long (but far, far less than 110150), the primality of a number can be determined with near certainty. This type of approach, known as randomization, crops up frequently in the design of algorithms. In essence, if a problem is too hard or time-consuming to solve exactly, a simpler, shorter approach based on randomization is used to provide an answer with any desired level of confidence.

UNPREDICTABILITY
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.

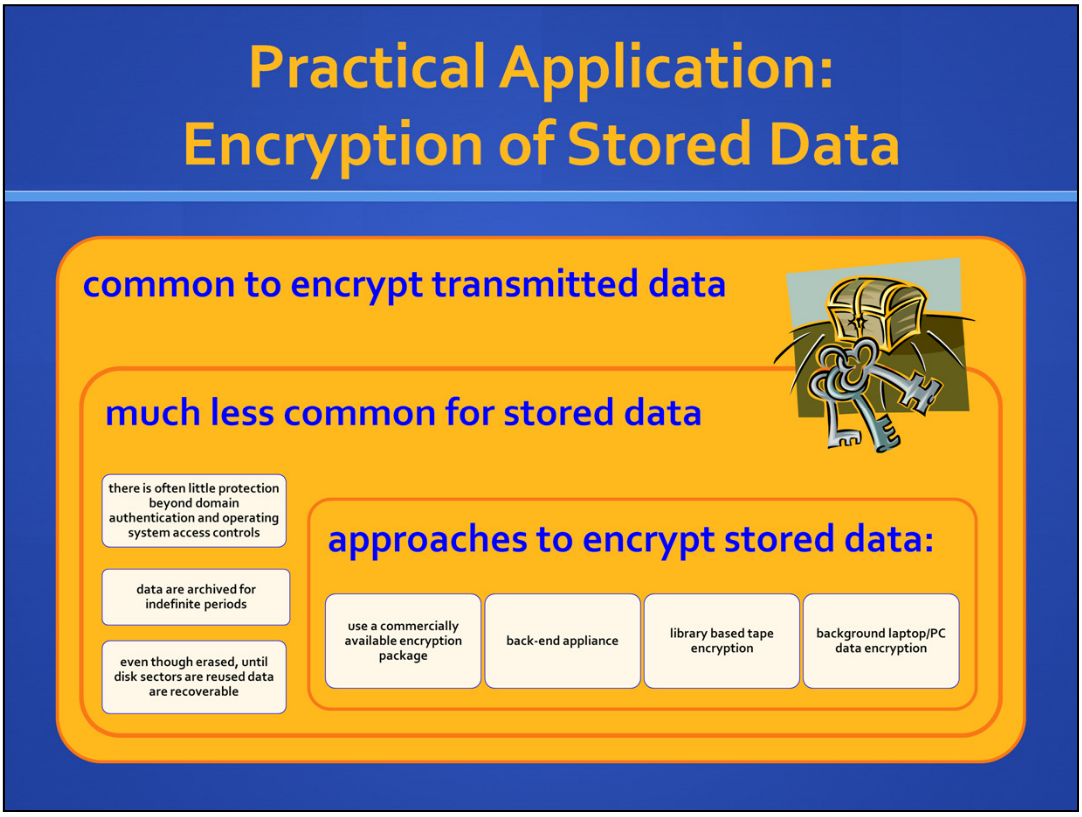


Cryptographic applications typically make use of algorithmic techniques for random number generation. These algorithms are deterministic and therefore produce sequences of numbers that are not statistically random. However, if the algorithm is good, the resulting sequences will pass many reasonable tests of randomness. Such numbers are referred to as **pseudorandom numbers.**

You may be somewhat uneasy about the concept of using numbers generated by a deterministic algorithm as if they were random numbers. Despite what might be called philosophical objections to such a practice, it generally works. As one expert on probability theory puts it [HAMM91],

For practical purposes we are forced to accept the awkward concept of “relatively random” meaning that with regard to the proposed use we can see no reason why they will not perform as if they were random (as the theory usually requires). This is highly subjective and is not very palatable to purists, but it is what statisticians regularly appeal to when they take “a random sample”—they hope that any results they use will have approximately the same properties as a complete counting of the whole sample space that occurs in their theory.

A true random number generator (TRNG) uses a nondeterministic source to produce randomness. Most operate by measuring unpredictable natural processes, such as pulse detectors of ionizing radiation events, gas discharge tubes, and leaky capac itors. Intel has developed a commercially available chip that samples thermal noise by amplifying the voltage measured across undriven resistors [JUN99]. A group at Bell Labs has developed a technique that uses the variations in the response time of raw read requests for one disk sector of a hard disk [JAKO98]. LavaRnd is an open source project for creating truly random numbers using inexpensive cameras, open source code, and inexpensive hardware. The system uses a saturated charge- coupled device (CCD) in a light-tight can as a chaotic source to produce the seed. Software processes the result into truly random numbers in a variety of formats.



One of the principal security requirements of a computer system is the protection of stored data. Security mechanisms to provide such protection include access control,
intrusion detection, and intrusion prevention schemes, all of which are discussed in this book. The book also describes a number of technical means by which these various security mechanisms can be made vulnerable. But beyond technical
approaches, these approaches can become vulnerable because of human factors.
We list a few examples here, based on [ROTH05].

• In December of 2004, Bank of America employees backed up and sent to its backup data center tapes containing the names, addresses, bank account numbers,
and Social Security numbers of 1.2 million government workers enrolled in a charge-card account. None of the data were encrypted. The tapes never arrived and indeed have never been found. Sadly, this method of backing up and shipping data is all too common. As an another example, in April of 2005, Ameritrade blamed its shipping vendor for losing a backup tape containing unencrypted information on 200,000 clients.

• In April of 2005, San Jose Medical group announced that someone had physically stolen one of its computers and potentially gained access to 185,000 unencrypted patient records.

• There have been countless examples of laptops lost at airports, stolen from a parked car, or taken while the user is away from his or her desk. If the data on the laptop's hard drive are unencrypted, all of the data are available to the thief.

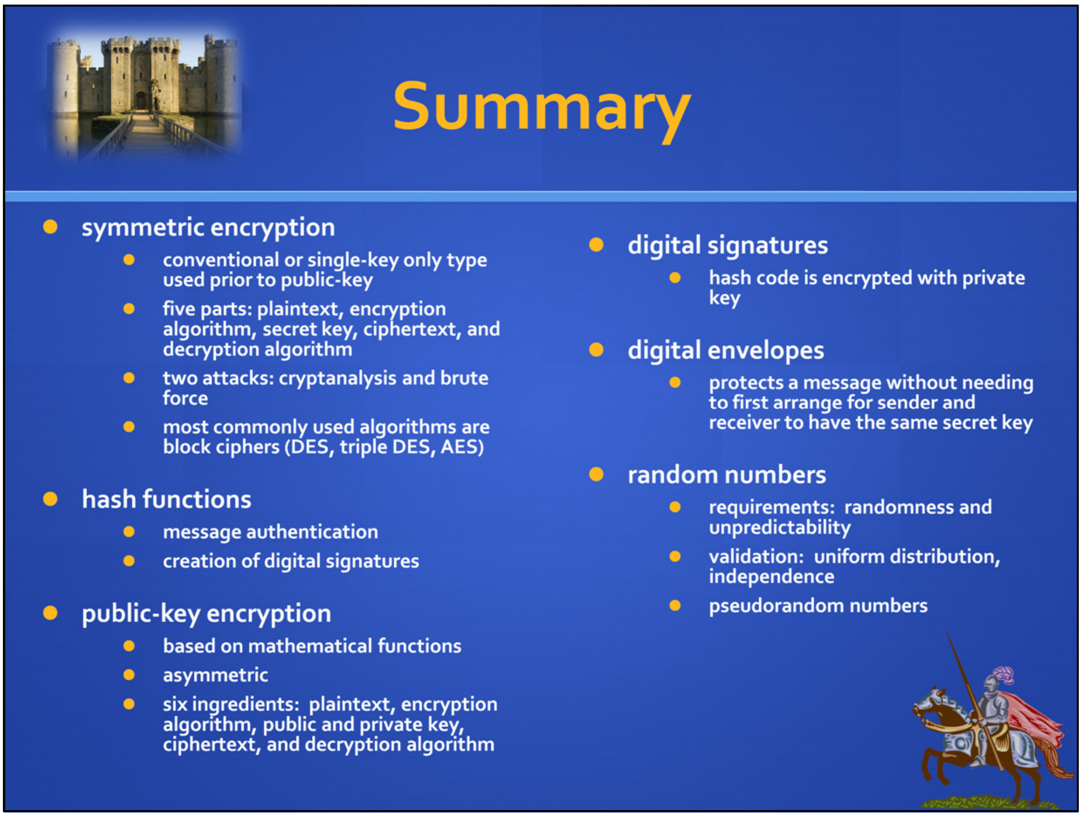
Although it is now routine for businesses to provide a variety of protections, including encryption, for information that is transmitted across networks, via the Internet, or via wireless devices, once data are stored locally (referred to as *data at rest), there is often little protection beyond domain authentication and operating* system access controls. Data at rest are 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 are recoverable. Thus it becomes attractive, and indeed should be mandatory, to encrypt data at rest and combine this with an effective encryption key management scheme.

There are a variety of ways to provide encryption services. A simple approach available for use on a laptop is to use a commercially available encryption package such as Pretty Good Privacy (PGP). PGP enables a user to generate a key from a password and then use that key to encrypt selected files on the hard disk. The PGP package does not store the password. To recover a file, the user enters the password, PGP generates the password, and PGP decrypts the file. So long as the user protects his or her password and does not use an easily guessable password, the files are fully protected while at rest. Some more recent approaches are listed in [COLL06]:

• **Back-end appliance: This is 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. These devices encrypt data at close to wire speed, with very little latency. In contrast, encryption software on servers and storage systems slows backups. A system man ager configures the appliance to accept requests from specified clients, for which unencrypted data are supplied.

• **Library-based tape encryption: This is 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. The key can be exported via secure e-mail or a small flash drive that is transported securely. If the matching tape drive hardware co-processor is not available at the other site, the target facility can use the key in a software decryption package to recover the data.

• **Background laptop and PC data encryption: A number of vendors offer software**
products that provide encryption that is transparent to the application and the user. Some products encrypt all or designated files and folders. Other products create a virtual disk, which can be maintained locally on the user's hard drive or maintained 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.



Chapter 2 summary.