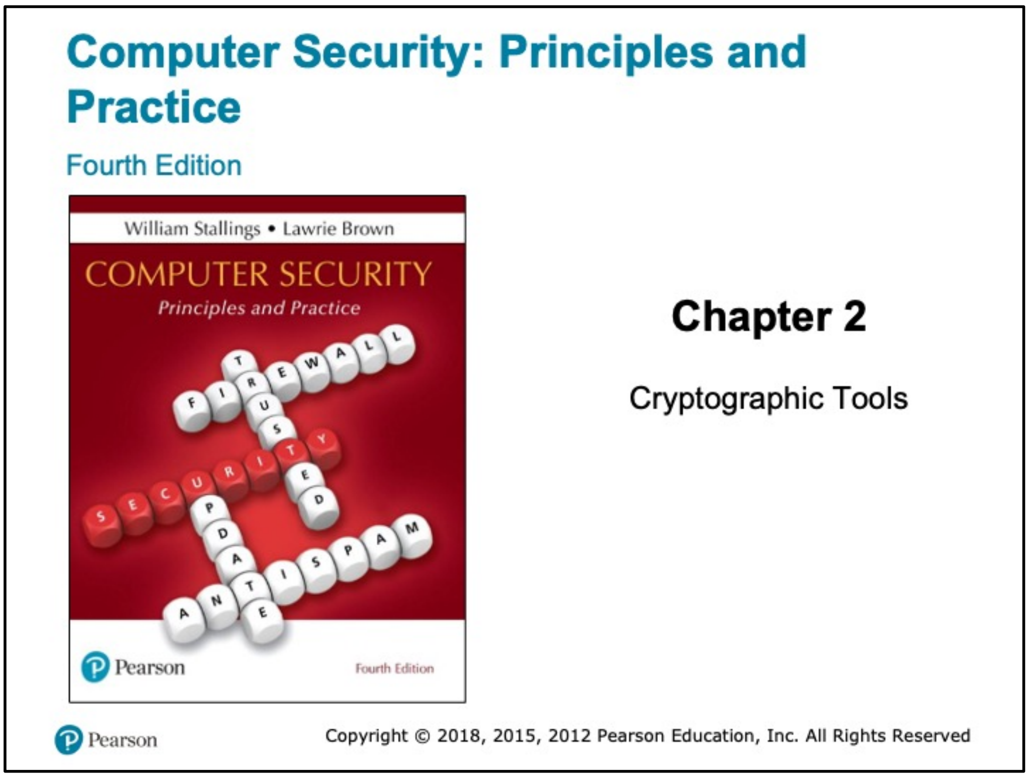
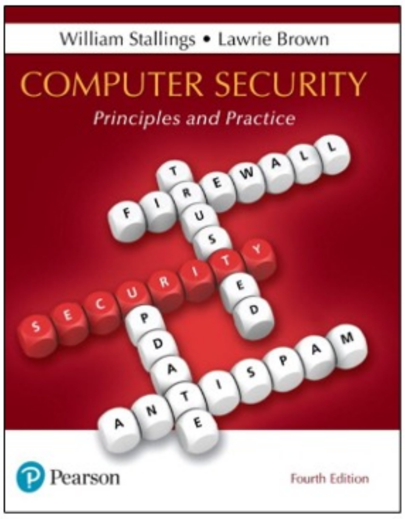


Computer Security: Principles and Practice

Fourth Edition



Chapter 2

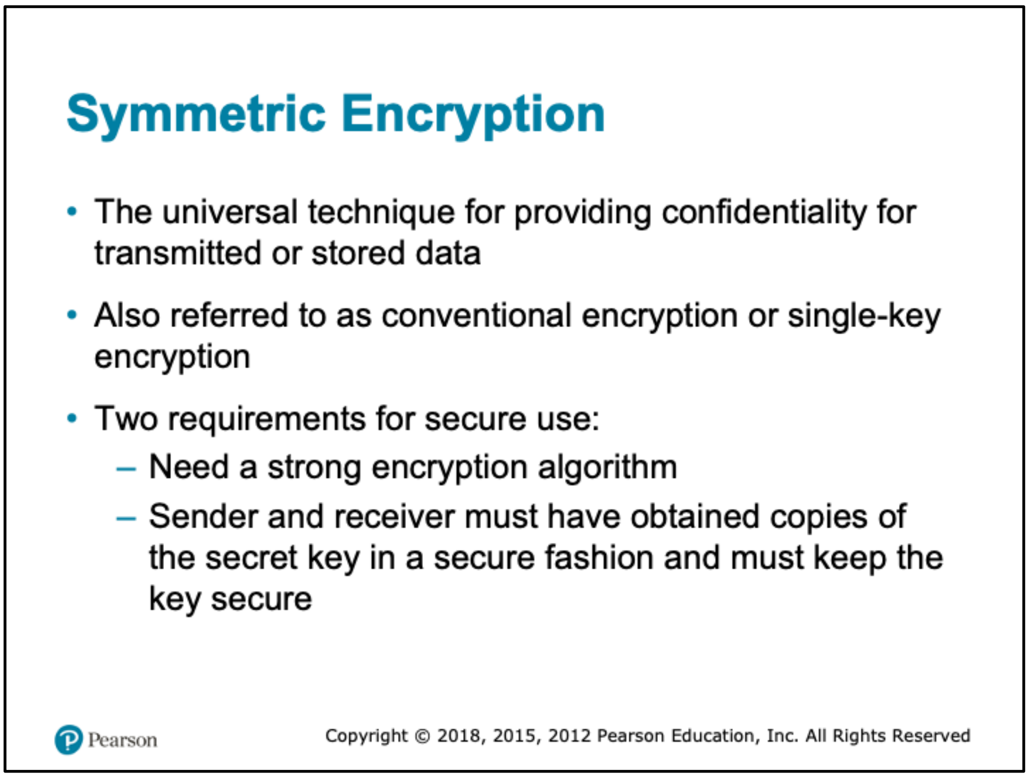
Cryptographic Tools

If this PowerPoint presentation contains mathematical equations, you may need to check that your computer has the following installed:
1) MathType Plugin
2) Math Player (free versions available)
3) NVDA Reader (free versions available)

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 will 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.



Symmetric Encryption

- The universal technique for providing confidentiality for transmitted or stored data
- Also referred to as conventional encryption or single-key encryption
- Two requirements for secure use:
  - Need a strong encryption algorithm
  - Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure

Pearson

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.

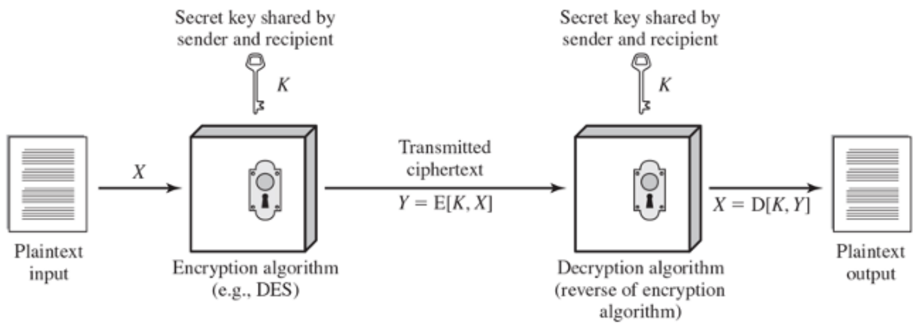
There are two requirements for secure use of symmetric encryption:

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


Secret key shared by sender and recipient
K
Secret key shared by sender and recipient
K
X
Transmitted ciphertext
Y = E[K, X]
X = D[K, Y]
Plaintext input
Encryption algorithm (e.g., DES)
Decryption algorithm (reverse of encryption algorithm)
Plaintext output

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.

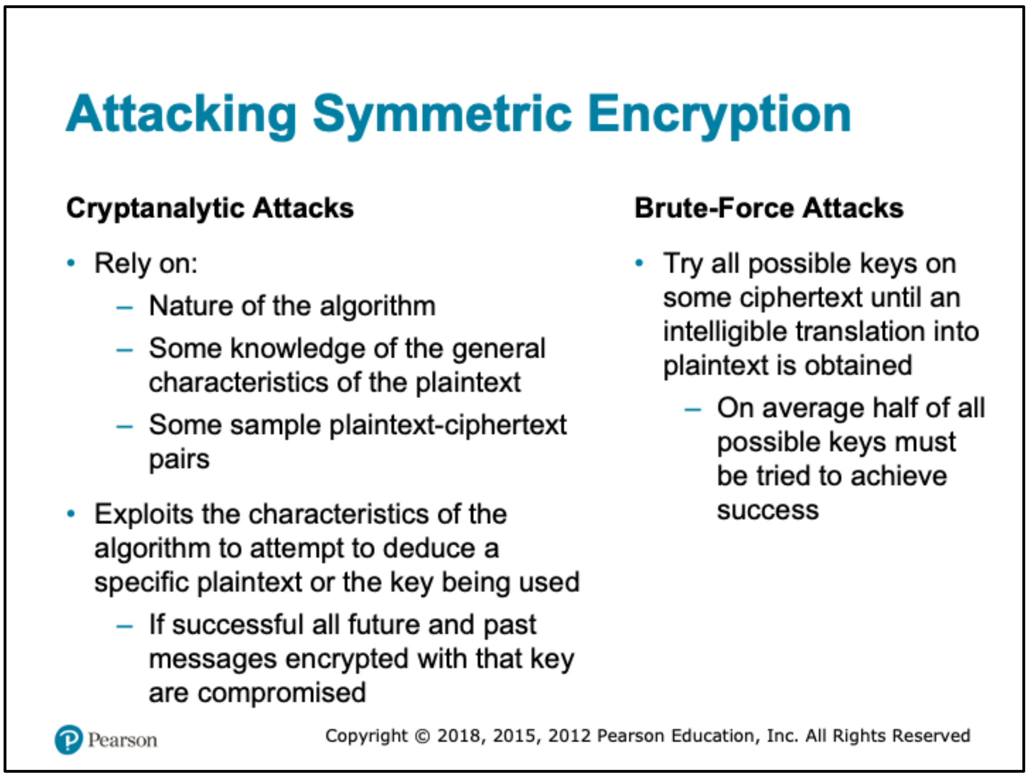
• Encryption algorithm: The encryption algorithm performs various substitutions and transformations on the plaintext.

• 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.

The plain text is fed in to the algorithm as input. Encryption Algorithm, example, A E S. The input secret key K is shared by sender and recipient to the encryption algorithm. The encryption algorithm produces the output as ciphertext, Y = E (K, X). Decryption algorithm, reverse of encryption algorithm. It takes the ciphertext and the secret key K, shared by sender and recipient as the input and produces the original plain text.



Attacking Symmetric Encryption

Cryptanalytic Attacks

- Rely on:
  - Nature of the algorithm
  - Some knowledge of the general characteristics of the plaintext
  - Some sample plaintext-ciphertext pairs
- Exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or the key being used
  - If successful all future and past messages encrypted with that key are compromised

Brute-Force Attacks

- Try all possible keys on some ciphertext until an intelligible translation into plaintext is obtained
  - On average half of all possible keys must be tried to achieve success

Pearson

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.

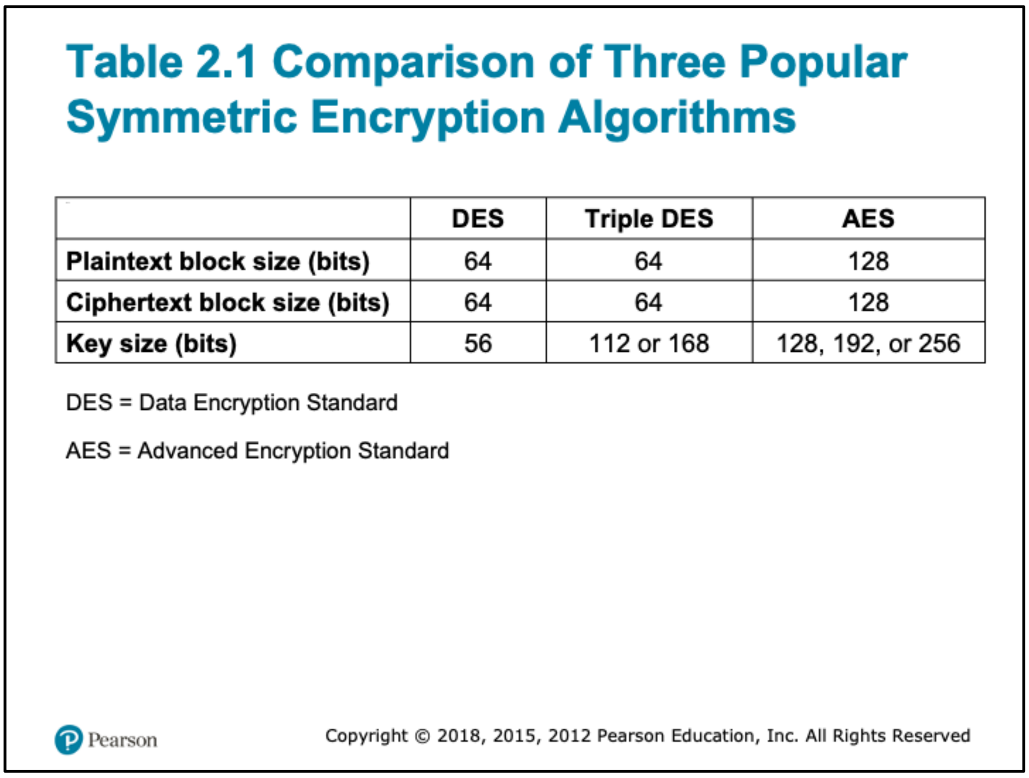


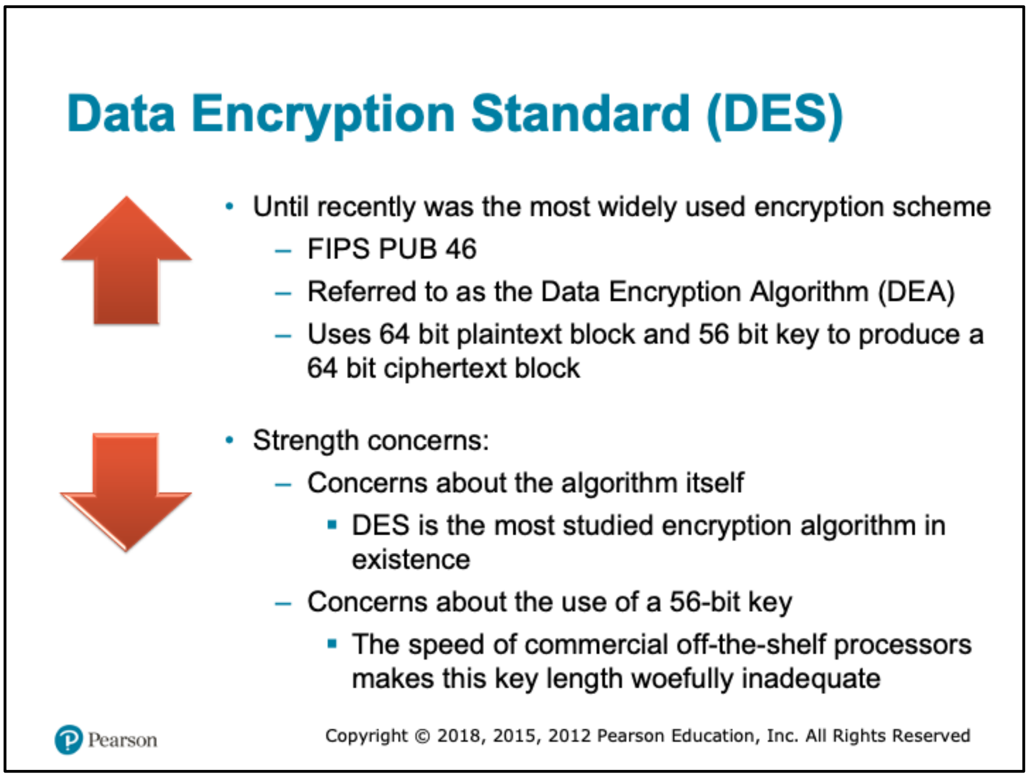
Table 2.1 Comparison of Three Popular Symmetric Encryption Algorithms

	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

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 2.1. This subsection provides an overview of these algorithms. Chapter 20 presents the technical details.



Data Encryption Standard (DES)

- Until recently was the most widely used encryption scheme
  - FIPS PUB 46
  - Referred to as the Data Encryption Algorithm (DEA)
  - Uses 64 bit plaintext block and 56 bit key to produce a 64 bit ciphertext block
- Strength concerns:
  - Concerns about the algorithm itself
    - DES is the most studied encryption algorithm in existence
  - Concerns about the use of a 56-bit key
    - The speed of commercial off-the-shelf processors makes this key length woefully inadequate

Until recently, the most widely used encryption scheme was 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 FIPS PUB 46 (Data Encryption Standard , January 1977). 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. Given the speed of commercial off-the-shelf processors, this key length is woefully inadequate. A paper from Seagate Technology [SEAG08] suggests that a rate of one billion (10^9) key combinations per second is reasonable for today's multicore computers. Recent offerings confirm this. Both Intel and AMD now offer hardware-based instructions to accelerate the use of AES. Tests run on a contemporary multicore Intel machine resulted in an encryption rate of about half a billion encryptions per second [BASU12]. Another recent analysis suggests that with contemporary supercomputer technology, a rate of 10^{13} encryptions/s is reasonable [AROR12].

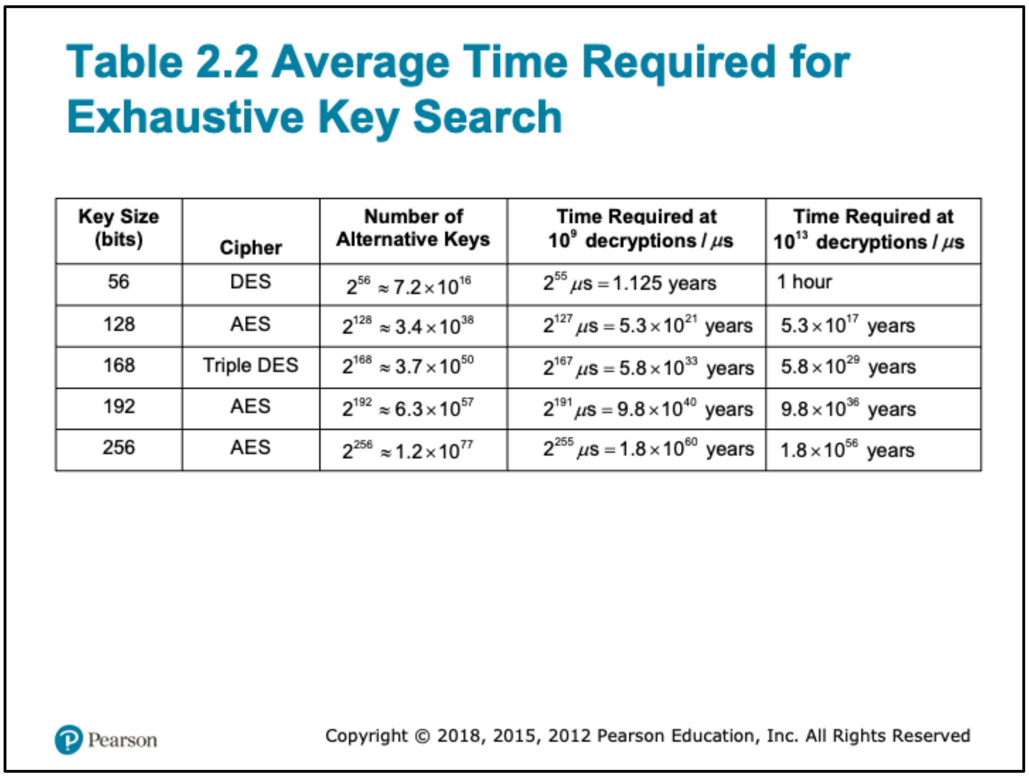
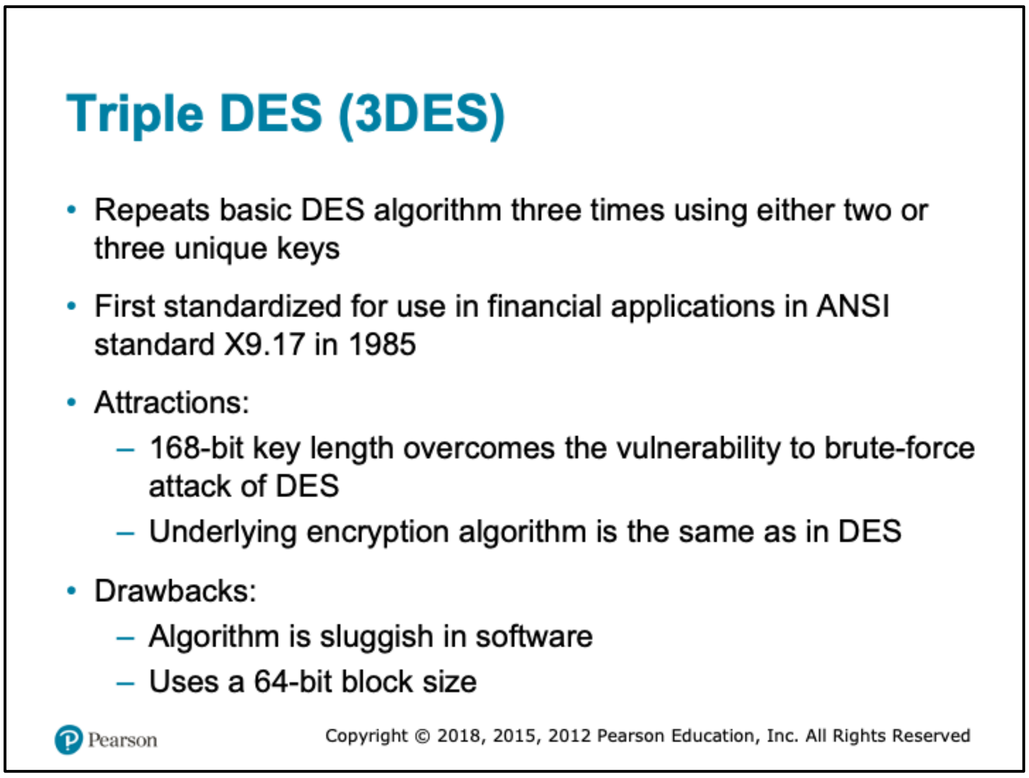


Table 2.2 Average Time Required for Exhaustive Key Search

Key Size (bits)	Cipher	Number of Alternative Keys	Time Required at 10^9 decryptions / μs	Time Required at 10^{13} decryptions / μs
56	DES	$2^{56} \approx 7.2 \times 10^{16}$	$2^{55}\ \mu s = 1.125$ years	1 hour
128	AES	$2^{128} \approx 3.4 \times 10^{38}$	$2^{127}\ \mu s = 5.3 \times 10^{21}$ years	5.3×10^{17} years
168	Triple DES	$2^{168} \approx 3.7 \times 10^{50}$	$2^{167}\ \mu s = 5.8 \times 10^{33}$ years	5.8×10^{29} years
192	AES	$2^{192} \approx 6.3 \times 10^{57}$	$2^{191}\ \mu s = 9.8 \times 10^{40}$ years	9.8×10^{36} years
256	AES	$2^{256} \approx 1.2 \times 10^{77}$	$2^{255}\ \mu s = 1.8 \times 10^{60}$ years	1.8×10^{56} years

Pearson

Table 2.2 shows how much time is required for a brute-force attack for various key sizes. As can be seen, a single PC can break DES in about a year; if multiple PCs work in parallel, the time is drastically shortened. And today's supercomputers should be able to find a key in about an hour. Key sizes of 128 bits or greater are effectively unbreakable using simply a brute-force approach. Even if we managed to speed up the attacking system by a factor of 1 trillion (10^{12}), it would still take over 100,000 years to break a code using a 128-bit key.


Triple DES (3DES)

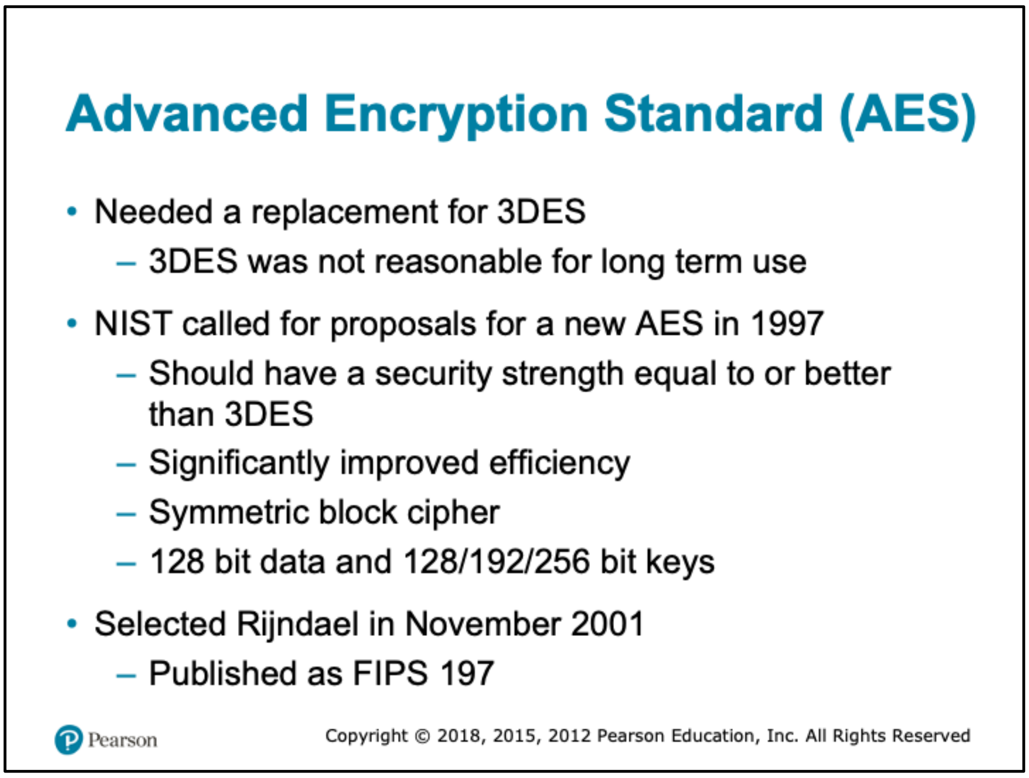
- Repeats basic DES algorithm three times using either two or three unique keys
- First standardized for use in financial applications in ANSI standard X9.17 in 1985
- Attractions:
  - 168-bit key length overcomes the vulnerability to brute-force attack of DES
  - Underlying encryption algorithm is the same as in DES
- Drawbacks:
  - Algorithm is sluggish in software
  - Uses a 64-bit block size

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.

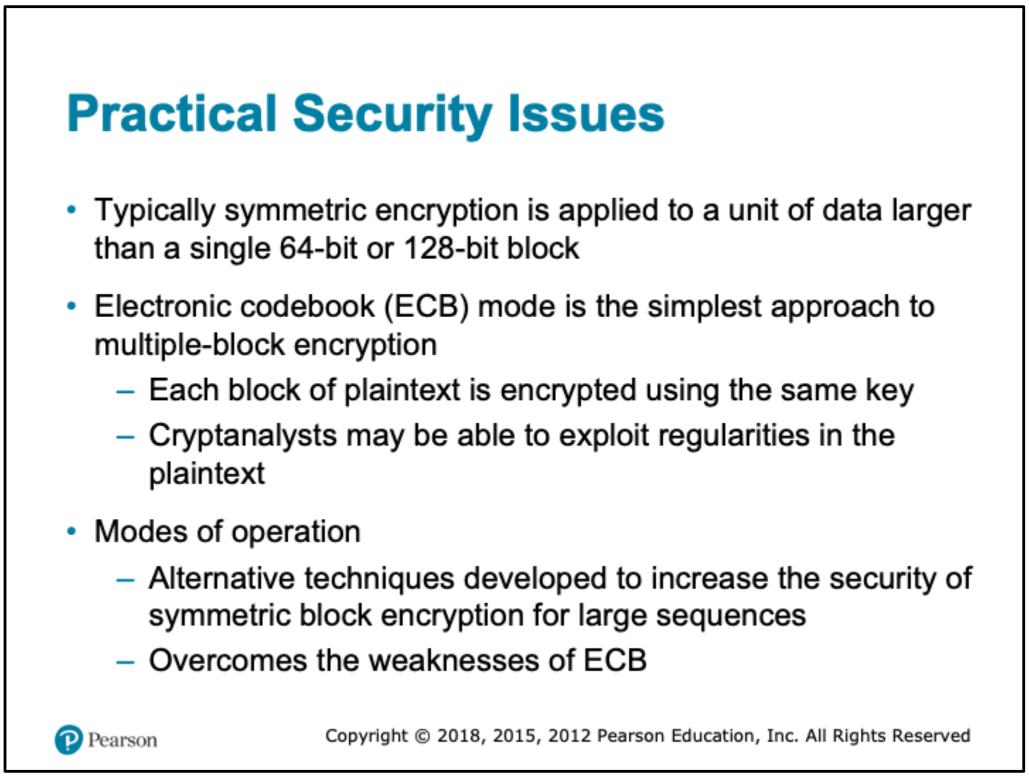


Advanced Encryption Standard (AES)

- Needed a replacement for 3DES
  - 3DES was not reasonable for long term use
- NIST called for proposals for a new AES in 1997
  - Should have a security strength equal to or better than 3DES
  - Significantly improved efficiency
  - Symmetric block cipher
  - 128 bit data and 128/192/256 bit keys
- Selected Rijndael in November 2001
  - Published as FIPS 197

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.



Practical Security Issues

- Typically symmetric encryption is applied to a unit of data larger than a single 64-bit or 128-bit block
- Electronic codebook (ECB) mode is the simplest approach to multiple-block encryption
  - Each block of plaintext is encrypted using the same key
  - Cryptanalysts may be able to exploit regularities in the plaintext
- Modes of operation
  - Alternative techniques developed to increase the security of symmetric block encryption for large sequences
  - Overcomes the weaknesses of ECB

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 2.2 Types of Symmetric Encryption

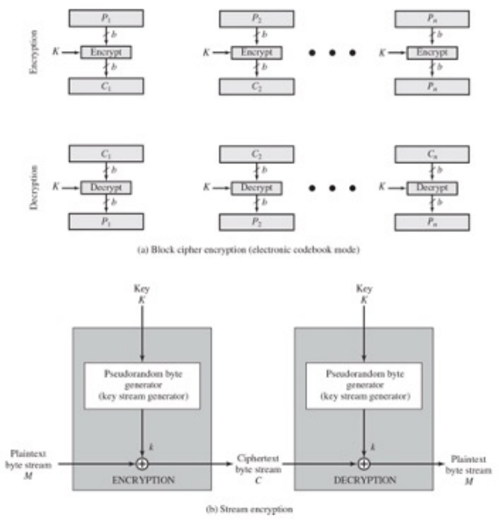


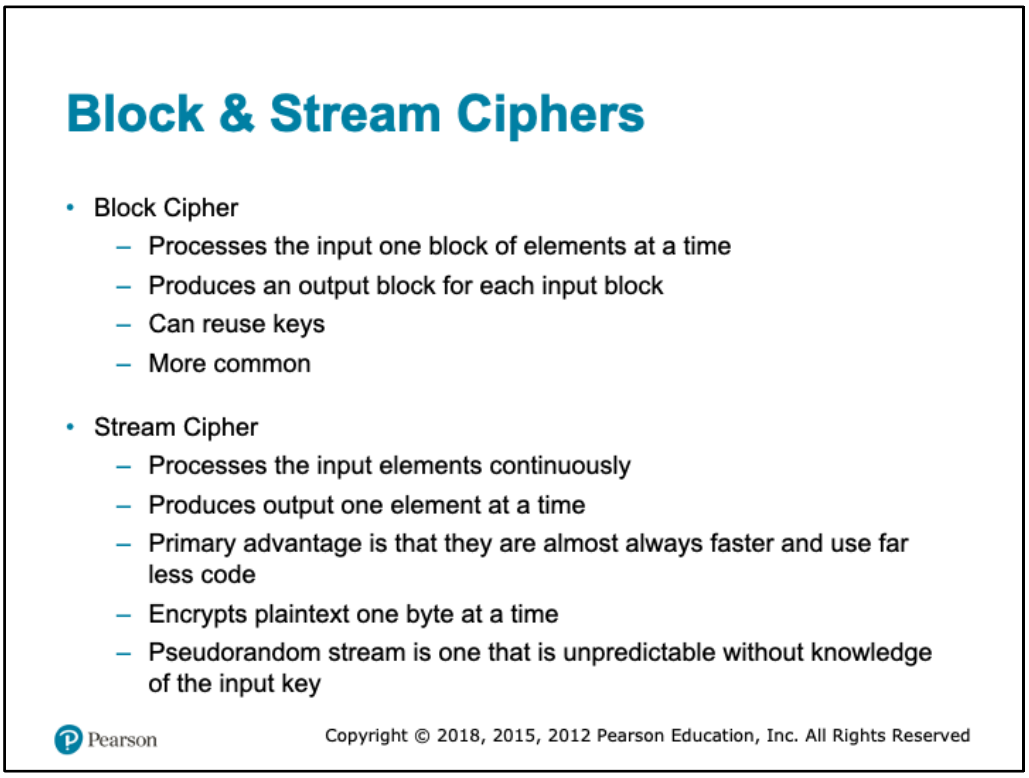
Figure 2.2a shows the ECB mode. A plaintext of length nb is divided into n b-bit blocks (P1, P2,….,Pn). Each block is encrypted using the same algorithm and the same encryption key, to produce a sequence of n b-bit blocks of ciphertext (C1, C2,….,Cn).

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 with which to work.

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.

Figure 2.2b 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.

Diagram a, block cipher encryption and decryption, electronic codebook mode. The plain text is divided into blocks from P sub 1 to P sub n. Each block is encrypted one at a time to produce the cipher block . The same key is used to encrypt each block. The block P sub n is encrypted to produce the cipher block c sub n. Diagram b, stream encryption. The encryption and decryption is depicted in two blocks. The plain text is fed to the encryption. An input is shared by a key K, to pseudorandom byte generator, key stream generator and the pseudorandom byte k is fed to the encryption algorithm. The encryption algorithm produces to the output as ciphertext byte stream C. The ciphertext is fed to the decryption algorithm. An input is shared by a key K, to pseudorandom byte generator, key stream generator and the pseudorandom byte k is fed to the decryption and it produces the final output as plain text byte stream M.



Block & Stream Ciphers

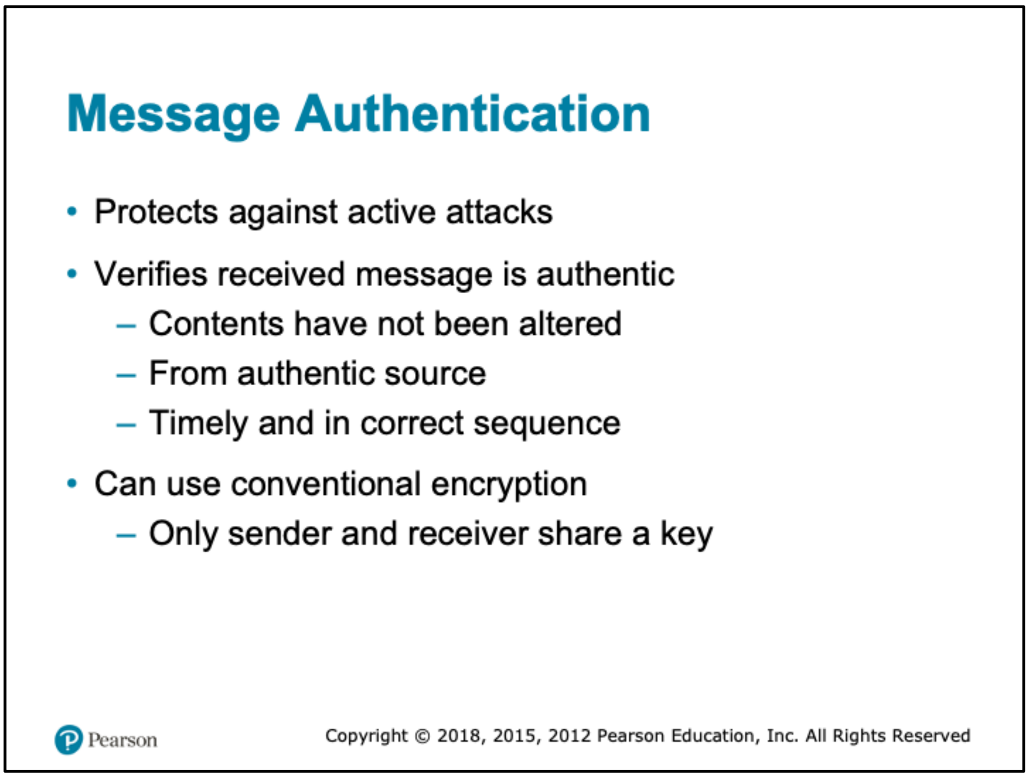
- Block Cipher
  - Processes the input one block of elements at a time
  - Produces an output block for each input block
  - Can reuse keys
  - More common
- Stream Cipher
  - Processes the input elements continuously
  - Produces output one element at a time
  - Primary advantage is that they are almost always faster and use far less code
  - Encrypts plaintext one byte at a time
  - Pseudorandom stream is one that is unpredictable without knowledge of the input key

Pearson

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.

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.



Message Authentication

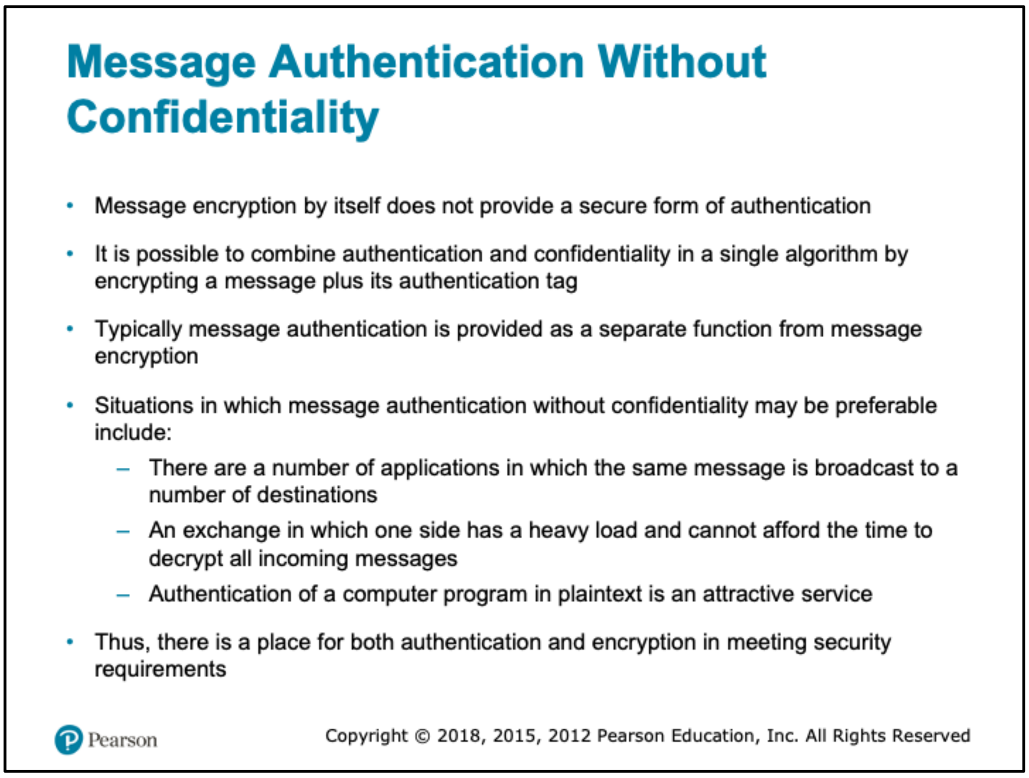
- Protects against active attacks
- Verifies received message is authentic
  - Contents have not been altered
  - From authentic source
  - Timely and in correct sequence
- Can use conventional encryption
  - Only sender and receiver share a key

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.*



Message Authentication Without Confidentiality

- Message encryption by itself does not provide a secure form of authentication
- It is possible to combine authentication and confidentiality in a single algorithm by encrypting a message plus its authentication tag
- Typically message authentication is provided as a separate function from message encryption
- Situations in which message authentication without confidentiality may be preferable include:
  - There are a number of applications in which the same message is broadcast to a number of destinations
  - An exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming messages
  - Authentication of a computer program in plaintext is an attractive service
- Thus, there is a place for both authentication and encryption in meeting security requirements

Because the approaches discussed in this section do not encrypt the message, message confidentiality is not provided. As was mentioned, message encryption by itself does not provide a secure form of authentication. However, it is possible to combine authentication and confidentiality in a single algorithm by encrypting a message plus its authentication tag. Typically, however, message authentication is provided as a separate function from message encryption. [DAVI89] suggests three situations in which message authentication without confidentiality is preferable:

1. There are a number of applications in which the same message is broadcast to a number of destinations. Two examples are notification to users that the network is now unavailable, and an alarm signal in a control center. It is cheaper and more reliable to have only one destination responsible for monitoring authenticity. Thus, the message must be broadcast in plaintext with an associated message authentication tag. The responsible system performs authentication. If a violation occurs, the other destination systems are alerted by a general alarm.

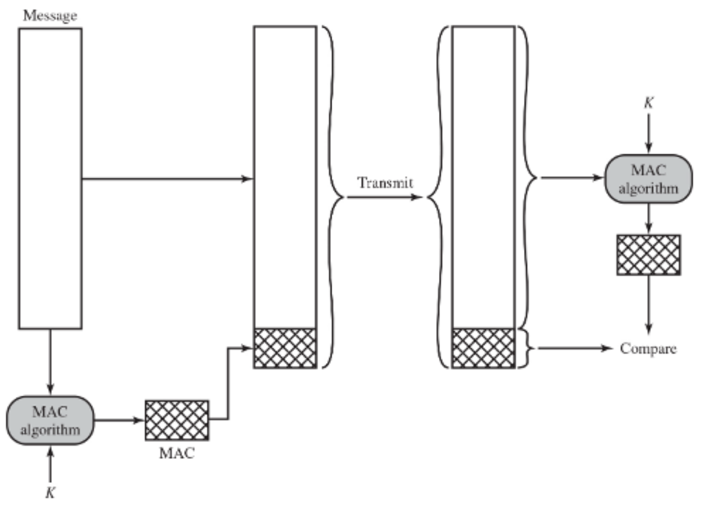
2. Another possible scenario is an exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming messages. Authentication is carried out on a selective basis, with messages being chosen

at random for checking.

3. Authentication of a computer program in plaintext is an attractive service. The computer program can be executed without having to decrypt it every time, which would be wasteful of processor resources. However, if a message authentication tag were attached to the program, it could be checked whenever assurance is required of the integrity of the program.

Thus, there is a place for both authentication and encryption in meeting security requirements.

Figure 2.3 Message Authentication Using a Message Authentication Code (MAC)



Pearson

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: MACM 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 2.3). 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.

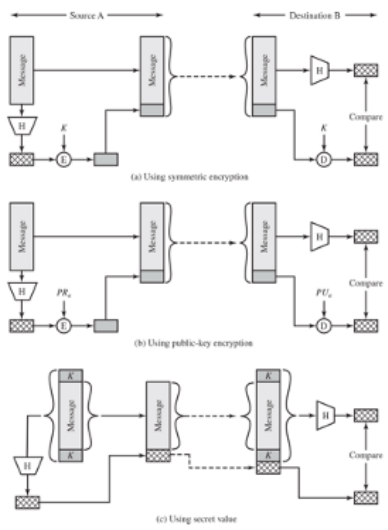
3. If the message includes a sequence number (such as is used with X.25, HDLC, and TCP), then the receiver can be assured of the proper sequence, because an attacker cannot successfully alter the sequence number.

A number of algorithms could be used to generate the code. The now withdrawn NIST publication FIPS PUB 113 (Computer Data Authentication , May 1985), recommended the use of DES. However AES would now be a more suitable choice. DES or AES is used to generate an encrypted version of the message, and some of the bits of ciphertext are used as the code. A 16- or 32-bit code used to be typical, but would now be much too small to provide sufficient collision resistance, as we will discuss shortly.

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.

The steps are as follows. The first block labeled message is shared to M A C algorithm and a key K is shared to M A C algorithm. Step 2. The first block transmits the message to the second block with M A C and M A C is shared to the M A C in the second block. Step 3. The second block transmits the message to the third block with M A C. Step 4. The message in the third block is transmitted to M A C algorithm. A key is shared to the algorithm. The output from the algorithm is compared by M A C from the third block.

Figure 2.5 Message Authentication Using a One-Way Hash Function



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 2.5 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.

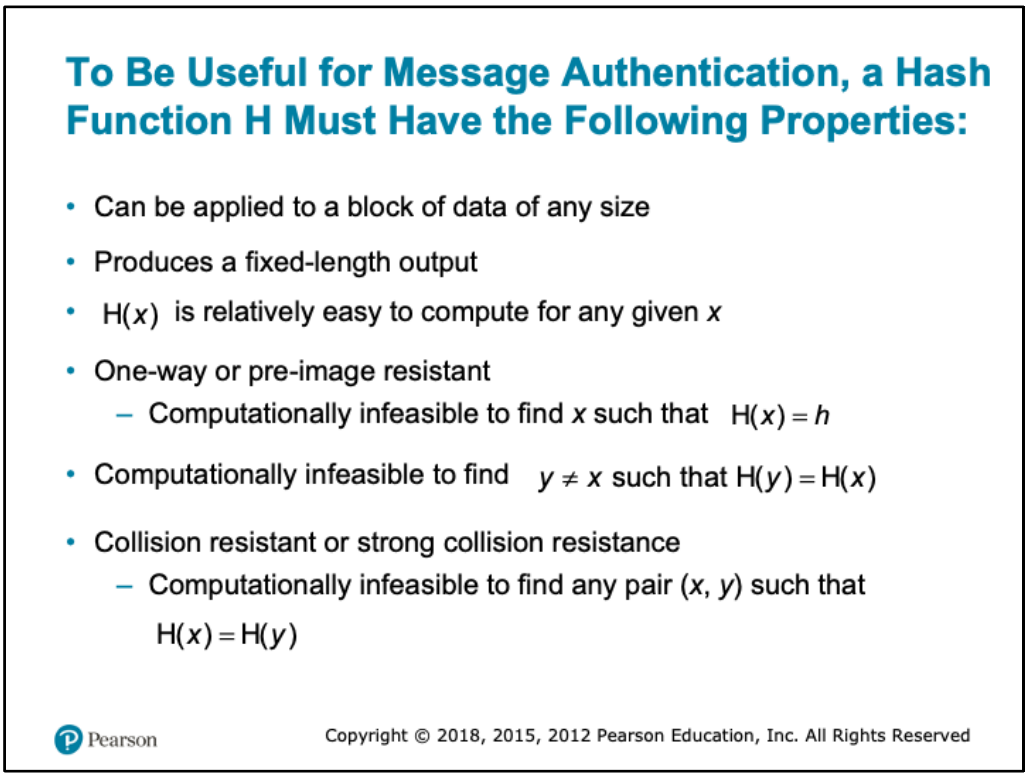
• Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead.

• An encryption algorithm may be protected by a patent.

Figure 2.5c 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 2.5c, 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(K II M II K). It then sends [M ii MDM] to B. Because B possesses K, it can recompute H(K II M II K) 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 2.5c and which has become the standard approach for a keyed hash MAC.

Diagram a, using symmetric encryption. The first block labeled message is shared to Hash function H algorithm M A C algorithm. A key K, is shared to the encryption algorithm. The first block transmits the message to the second block and the encrypted message is shared to the encrypted message in the second block. The process is represented as source A. The second block with encrypted message transmits the message to the third block with encrypted message. The message in the third block is transmitted to hash function and M A C. The encrypted message is fed to the decryption algorithm. A key is shared to the decryption algorithm. The output from the decryption algorithm is compared by M A C from the third block. This process is labeled, Destination B. Diagram b, using public key encryption. The first block labeled message is shared to Hash function H algorithm M A C algorithm. A public key P R sub a, is shared to the encryption algorithm. The first block transmits the message to the second block and the encrypted message is shared to the encrypted message in the second block. The process is represented as source A. The second block with encrypted message transmits the message to the third block with encrypted message. The message in the third block is transmitted to hash function and M A C. The encrypted message is fed to the decryption algorithm. A public key P U sub a is shared to the decryption algorithm. The output from the decryption algorithm is compared by M A C from the third block. This process is labeled, Destination B. Diagram c, using secret value. The first block labeled, message with keys at the top and bottom is transmitted to hash function, H. The hash function is fed to M A C and the message from the first block is transmitted to second block with M A C. The M A C is fed to the M A C in the second block. The message from the second block is transmitted to message in the third block with keys at the top and bottom. The M A C from the second block is fed to M A C in the third block. The message is transmitted to the hash function H. The output from H is fed to M A C. The M A C from the third block and the M A C from the hash function are compared.



To Be Useful for Message Authentication, a Hash Function H Must Have the Following Properties:

- Can be applied to a block of data of any size
- Produces a fixed-length output
- $H(x)$ is relatively easy to compute for any given x
- One-way or pre-image resistant
  - Computationally infeasible to find x such that $H(x) = h$
- Computationally infeasible to find $y \neq x$ such that $H(y) = H(x)$
- Collision resistant or 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 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

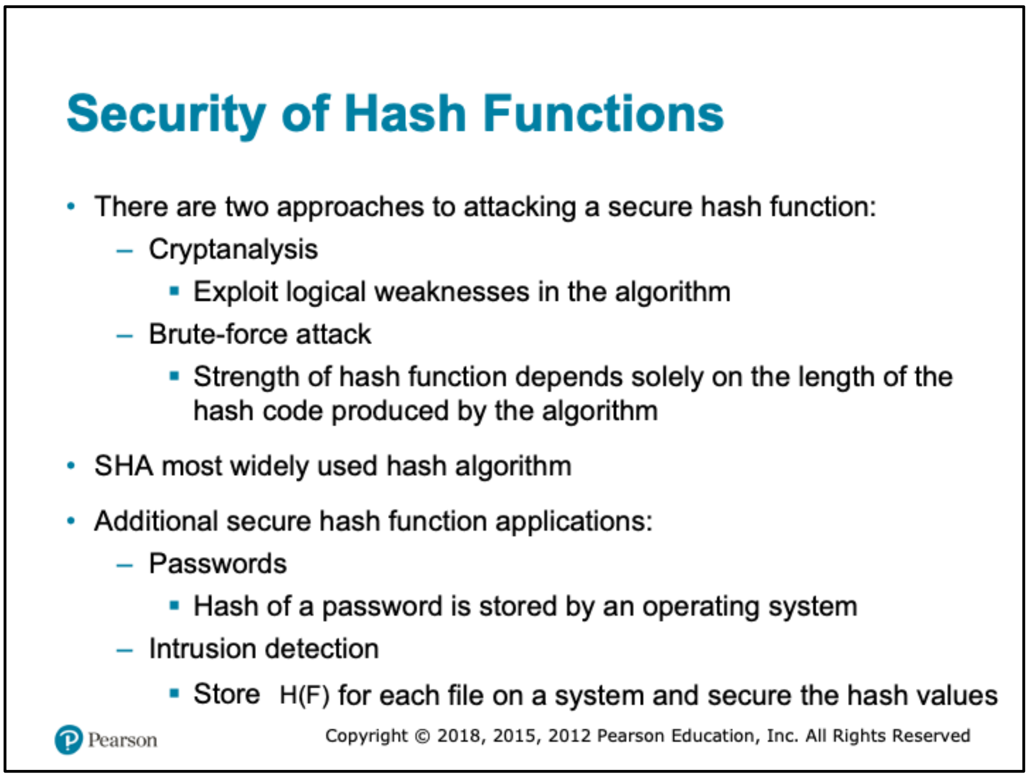
sometimes referred to as strong collision resistant.

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 2.5c). 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* $MD_M = H(K \| M \| K)$*. The attacker* then inverts the hash function to obtain $K \| M \| K = H^{-1}(MD_M)$*. Because the attacker now* has both *M and* $K \| M \| K$*, it is a trivial matter to recover K.*

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 2.5a 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.



Security of Hash Functions

- There are two approaches to attacking a secure hash function:
  - Cryptanalysis
    - Exploit logical weaknesses in the algorithm
  - Brute-force attack
    - Strength of hash function depends solely on the length of the hash code produced by the algorithm
- SHA most widely used hash algorithm
- Additional secure hash function applications:
  - Passwords
    - Hash of a password is stored by an operating system
  - Intrusion detection
    - Store H(F) for each file on a system and secure the hash values

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] 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. 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

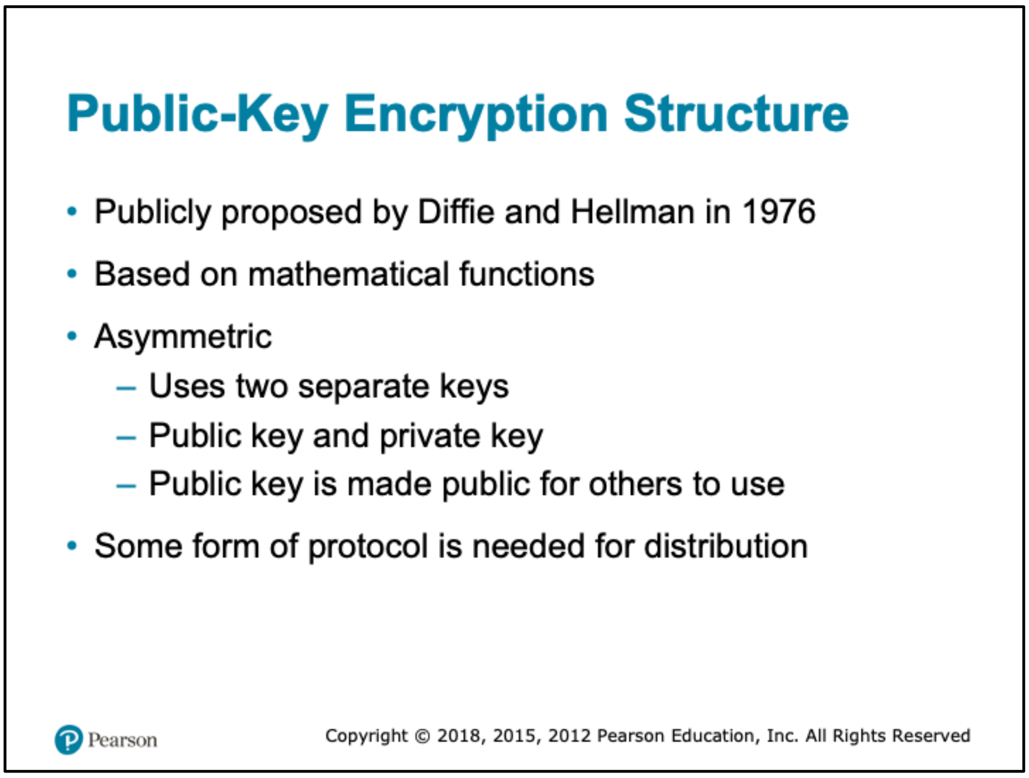
160 bits now appears suspect.

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 Structure

- Publicly proposed by Diffie and Hellman in 1976
- Based on mathematical functions
- Asymmetric
  - Uses two separate keys
  - Public key and private key
  - Public key is made public for others to use
- Some form of protocol is needed for distribution

Pearson

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.

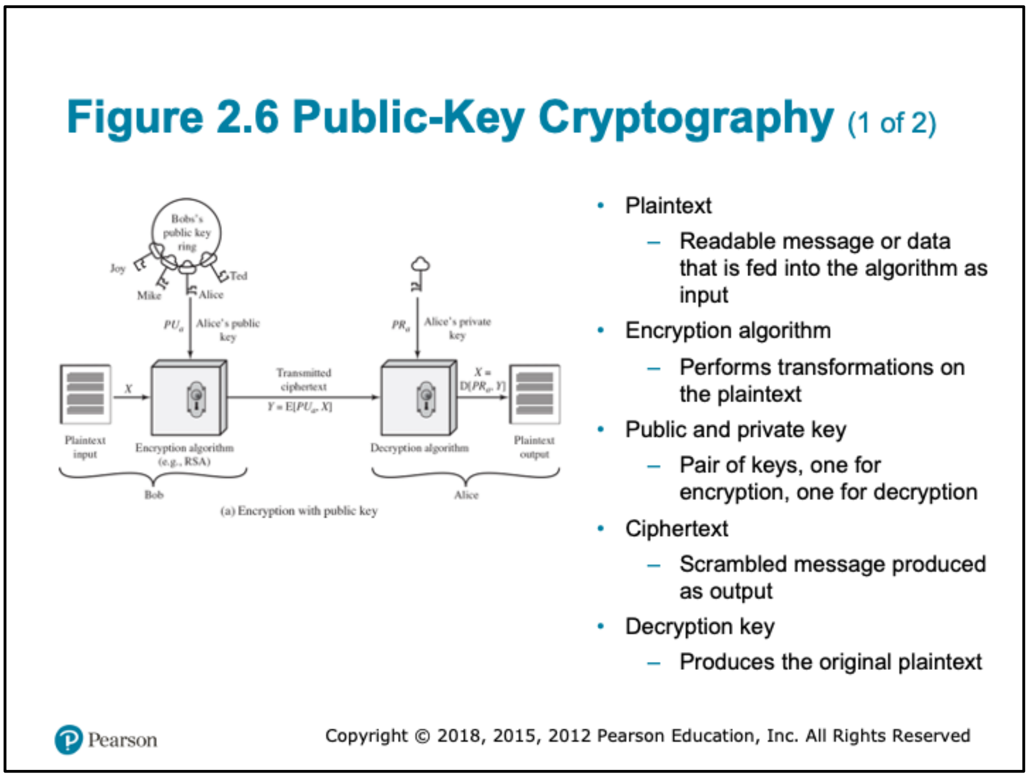
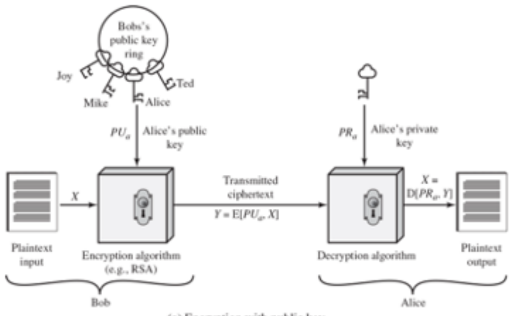


Figure 2.6 Public-Key Cryptography (1 of 2)



(a) Encryption with public key

- Plaintext
  - Readable message or data that is fed into the algorithm as input
- Encryption algorithm
  - Performs transformations on the plaintext
- Public and private key
  - Pair of keys, one for encryption, one for decryption
- Ciphertext
  - Scrambled message produced as output
- Decryption key
  - Produces the original plaintext

A public-key encryption scheme has six ingredients (Figure 2.6a):

• 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 2.6a 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 2.6a 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.

Diagram a, encryption with public key. The plain text is fed in to the algorithm as input. The plain text produces the output X to encryption Algorithm, example, R S A. The input Bob's public key ring has pair of keys labeled, Joy, Mike, Alice, and Ted. The public key from Alice, P U sub a is shared to the encryption algorithm. The encryption algorithm transmits the output as ciphertext, Y = E left bracket P U sub a, X right bracket. Decryption algorithm, reverse of encryption algorithm. It takes the ciphertext and the private key P R sub a, shared by Alice as the input and produces the original plain text by a description key X = D left bracket P R Sub a, Y right bracket.

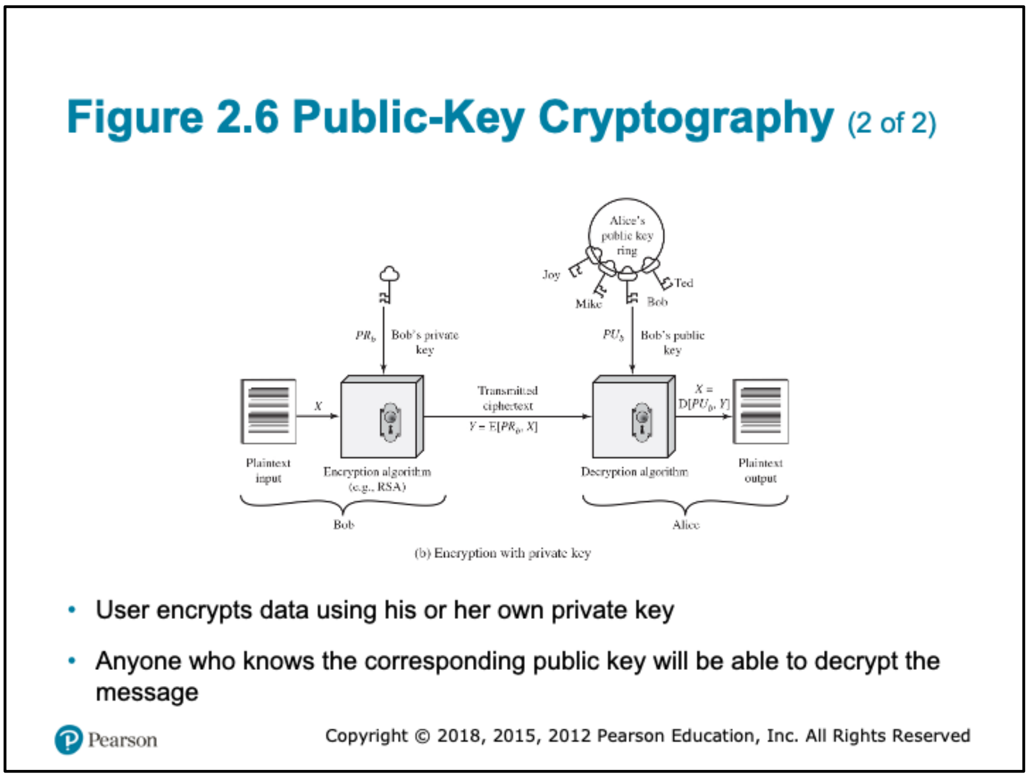
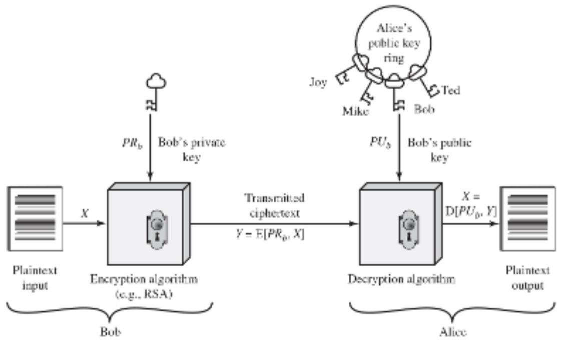


Figure 2.6 Public-Key Cryptography (2 of 2)



(b) Encryption with private key

- User encrypts data using his or her own private key
- Anyone who knows the corresponding public key will be able to decrypt the message

Pearson

Figure 2.6b 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 2.6b 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.

Diagram b, encryption with private key. The plain text is fed in to the algorithm as input. The plain text produces the output X to encryption Algorithm, example, R S A.. The private key from Bob, P R sub b is shared to the encryption algorithm. The encryption algorithm transmits the output as ciphertext, Y = E left bracket P R sub b, X right bracket. Decryption algorithm, reverse of encryption algorithm. It takes the ciphertext and the Alice public key ring with pair of keys labeled, Joy, Mike, Alice, and Ted. From the ring Bob's public key is shared as the input and produces the original plain text by a description key X = D left bracket P U Sub b, Y right bracket.

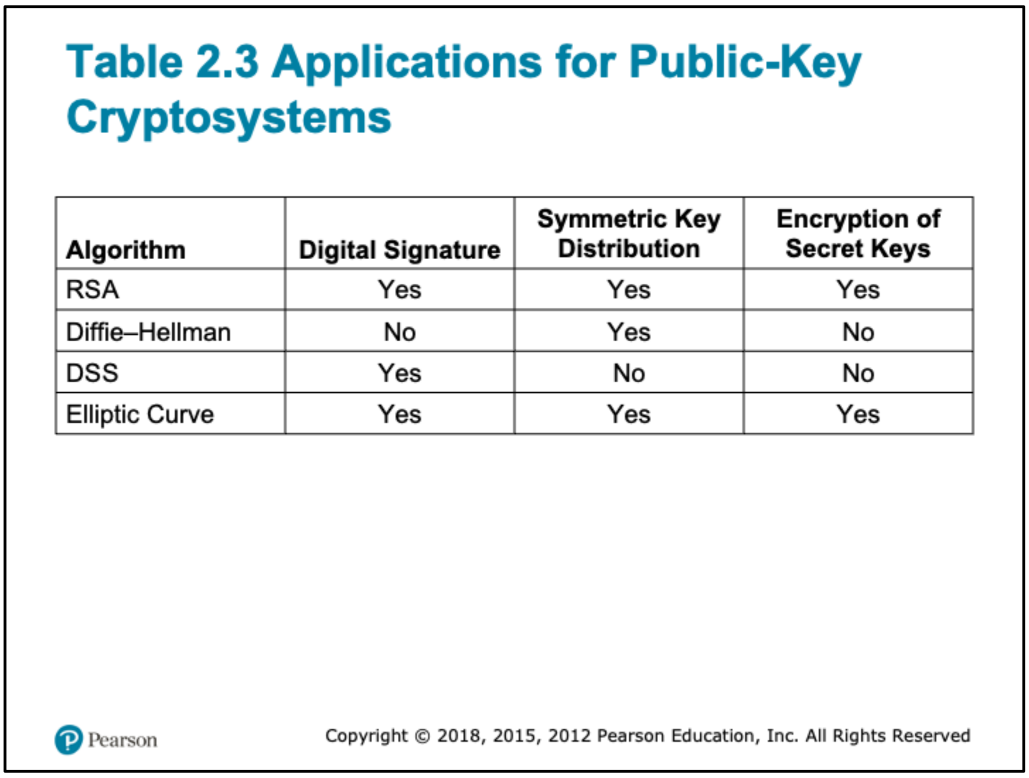


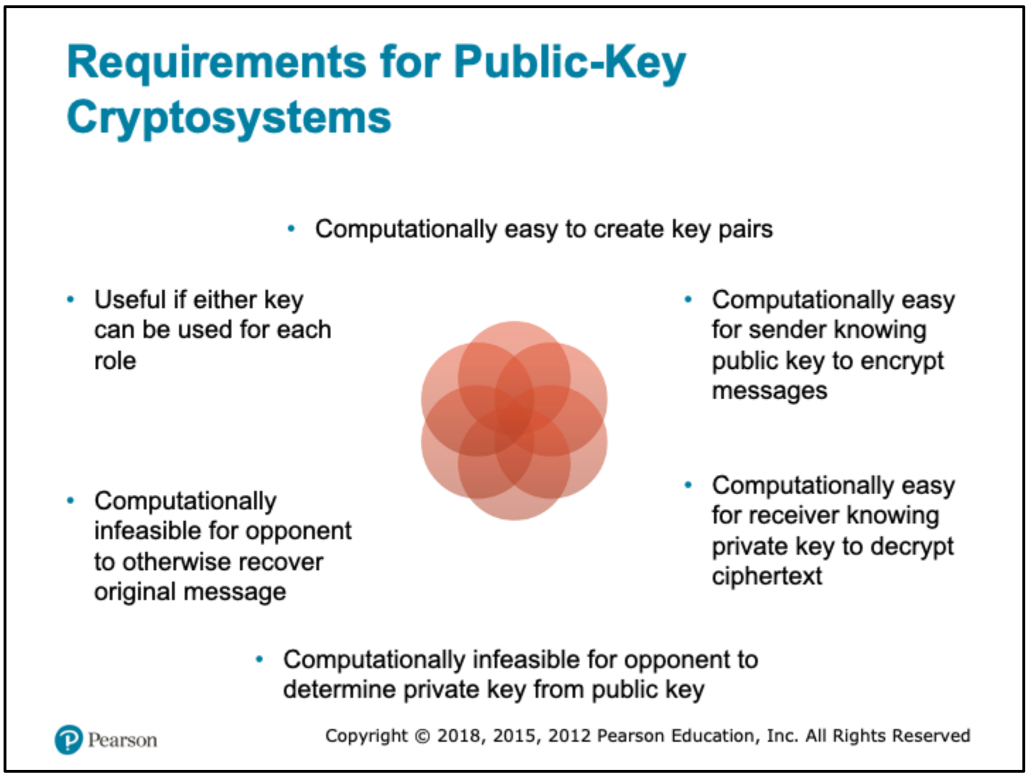
Table 2.3 Applications for Public-Key Cryptosystems

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

Pearson

Public-key systems are characterized by the use of a cryptographic type of algorithm with two keys, one held private and one available publicly. Depending on the application, the sender uses either the sender's private key or the receiver's public key, or both, to perform some type of cryptographic function. In broad terms, we can classify the use of public-key cryptosystems into three categories: digital signature, symmetric key distribution, and encryption of secret keys.

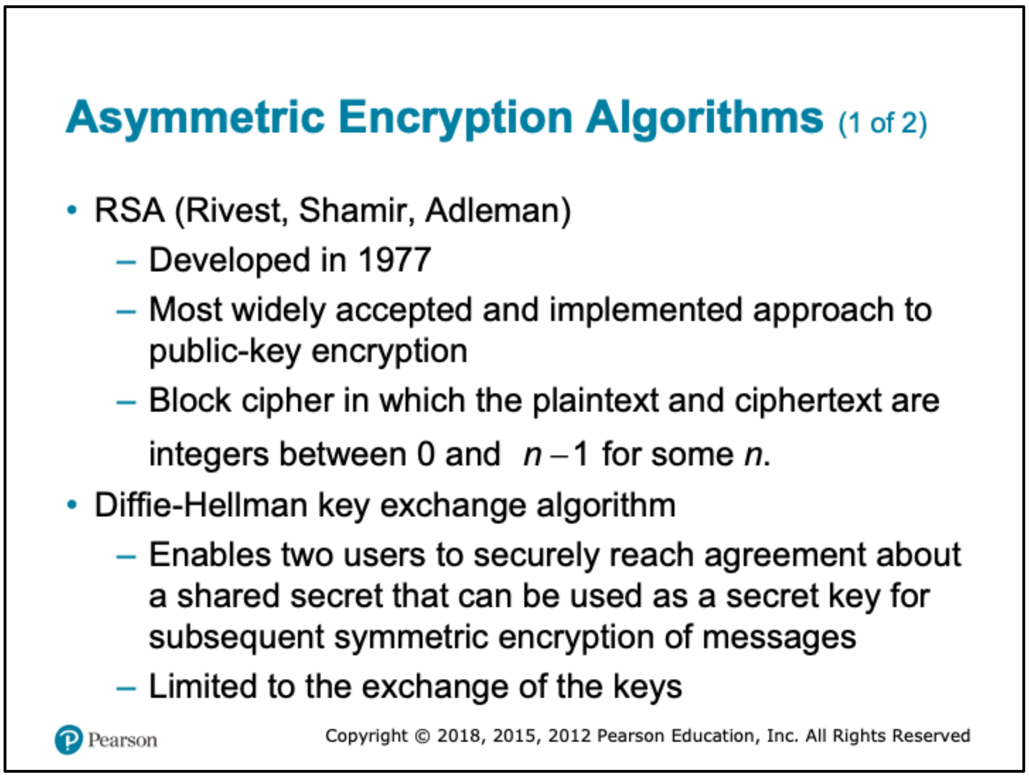
These applications will be discussed in Section 2.4. Some algorithms are suitable for all three applications, whereas others can be used only for one or two of these applications. Table 2.3 indicates the applications supported by the algorithms discussed in this section.



Requirements for Public-Key Cryptosystems

- Computationally easy to create key pairs
- Useful if either key can be used for each role
- Computationally easy for sender knowing public key to encrypt messages
- Computationally infeasible for opponent to otherwise recover original message
- Computationally easy for receiver knowing private key to decrypt ciphertext
- Computationally infeasible for opponent to determine private key from public key

The cryptosystem illustrated in Figure 2.6 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]:


Asymmetric Encryption Algorithms (1 of 2)

- RSA (Rivest, Shamir, Adleman)
  - Developed in 1977
  - Most widely accepted and implemented approach to public-key encryption
  - Block cipher in which the plaintext and ciphertext are integers between 0 and $n-1$ for some n.
- Diffie-Hellman key exchange algorithm
  - Enables two users to securely reach agreement about a shared secret that can be used as a secret key for subsequent symmetric encryption of messages
  - Limited to the exchange of the keys

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. Currently, a 1024-bit key size (about 300 decimal digits) is considered strong enough for virtually all applications.

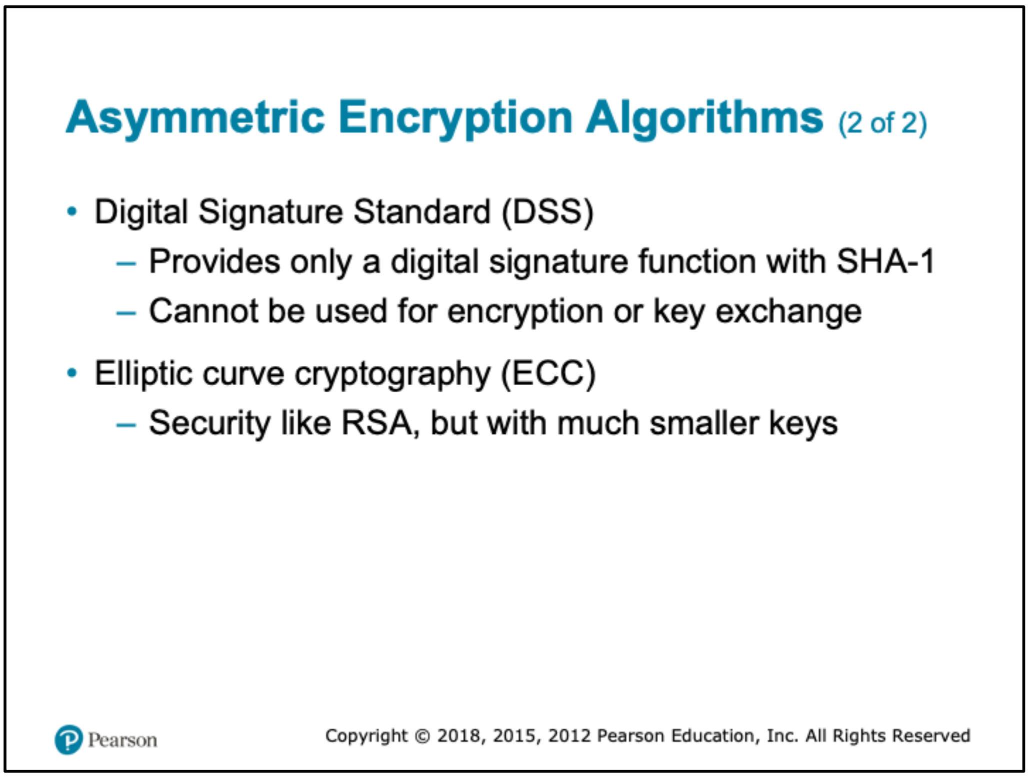
DIFFIE-HELLMAN KEY AGREEMENT The first published public-key algorithm appeared in the seminal paper by Diffie and Hellman that defined public-key cryptography [DIFF76] and is generally referred to as Diffie-Hellman key exchange, or key agreement. A number of commercial products employ this key exchange 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 were further revisions in 1998, 2000, 2009, and most recently in 2013 as FIPS PUB 186–4. 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.

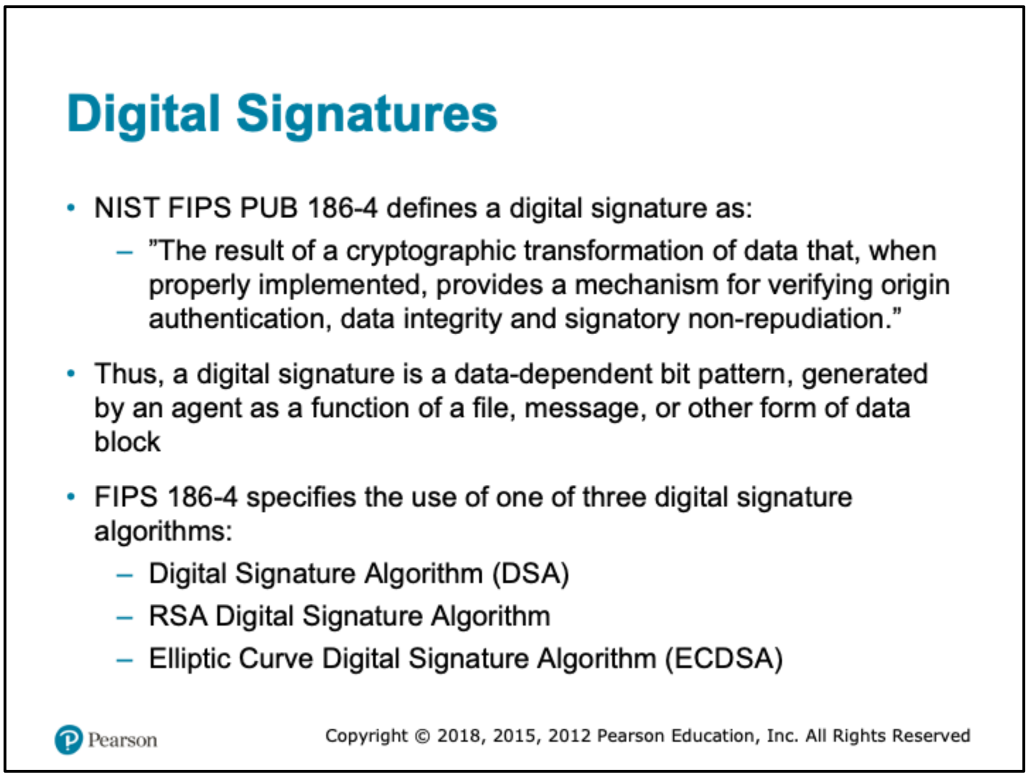
The principal attraction of ECC compared to RSA is that it appears to offer equal security for a far smaller bit size, thereby reducing processing overhead. On the other hand, although the theory of ECC has been around for some time, it is only recently that products have begun to appear and that there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



Asymmetric Encryption Algorithms (2 of 2)

- Digital Signature Standard (DSS)
  - Provides only a digital signature function with SHA-1
  - Cannot be used for encryption or key exchange
- Elliptic curve cryptography (ECC)
  - Security like RSA, but with much smaller keys

Pearson



Digital Signatures

- NIST FIPS PUB 186-4 defines a digital signature as:
  - ”The result of a cryptographic transformation of data that, when properly implemented, provides a mechanism for verifying origin authentication, data integrity and signatory non-repudiation.”
- Thus, a digital signature is a data-dependent bit pattern, generated by an agent as a function of a file, message, or other form of data block
- FIPS 186-4 specifies the use of one of three digital signature algorithms:
  - Digital Signature Algorithm (DSA)
  - RSA Digital Signature Algorithm
  - Elliptic Curve Digital Signature Algorithm (ECDSA)

Public-key encryption can be used for authentication with a technique known as the digital signature. NIST FIPS PUB 186-4 [*Digital Signature Standard (DSS*) , July 2013] defines a digital signature as follows: The result of a cryptographic transformation of data that, when properly implemented, provides a mechanism for verifying origin authentication, data integrity and signatory non-repudiation.

Thus, a digital signature is a data-dependent bit pattern, generated by an agent as a function of a file, message, or other form of data block. Another agent can access the data block and its associated signature and verify (1) the data block has been signed by the alleged signer, and (2) the data block has not been altered since the signing. Further, the signer cannot repudiate the signature.

FIPS 186-4 specifies the use of one of three digital signature algorithms:

• Digital Signature Algorithm (DSA): The original NIST-approved algorithm, which is based on the difficulty of computing discrete logarithms.

• RSA Digital Signature Algorithm: Based on the RSA public-key algorithm.

• Elliptic Curve Digital Signature Algorithm (ECDSA): Based on elliptic-curve cryptography.
Public-key encryption can be used for authentication with a technique known as the digital signature. NIST FIPS PUB 186-4 [*Digital Signature Standard (DSS*) , July 2013] defines a digital signature as follows: The result of a cryptographic transformation of data that, when properly implemented, provides a mechanism for verifying origin authentication, data integrity and signatory non-repudiation.

Thus, a digital signature is a data-dependent bit pattern, generated by an agent as a function of a file, message, or other form of data block. Another agent can access the data block and its associated signature and verify (1) the data block has been signed by the alleged signer, and (2) the data block has not been altered since the signing. Further, the signer cannot repudiate the signature.

FIPS 186-4 specifies the use of one of three digital signature algorithms:

• Digital Signature Algorithm (DSA): The original NIST-approved algorithm, which is based on the difficulty of computing discrete logarithms.

• RSA Digital Signature Algorithm: Based on the RSA public-key algorithm.

• Elliptic Curve Digital Signature Algorithm (ECDSA): Based on elliptic-curve cryptography.

Figure 2.7 Simplified Depiction of Essential Elements of Digital Signature Process

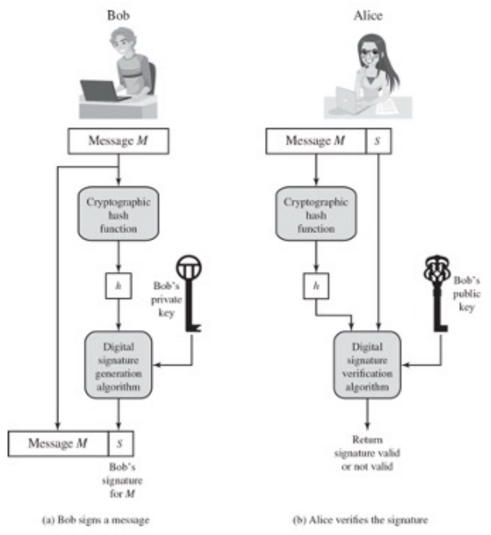
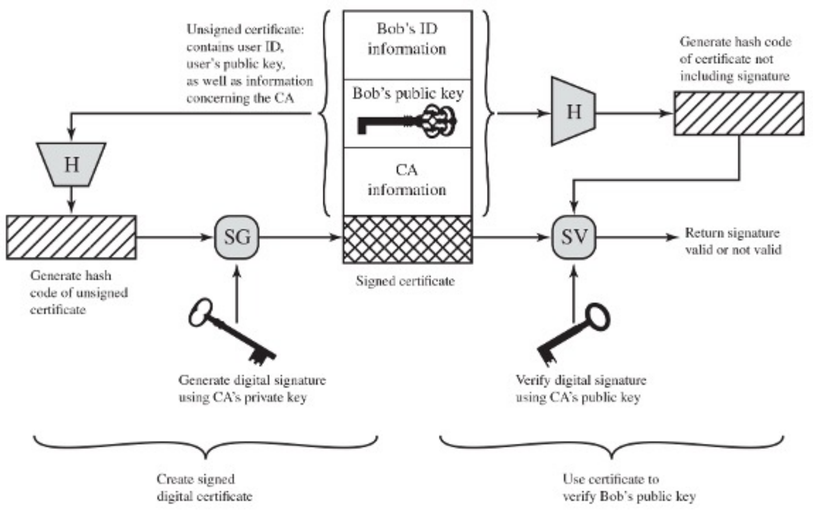

Bob
Alice
Message M
Message M
s
Cryptographic hash function
Cryptographic hash function
h
h
Bob's private key
Bob's public key
Digital signature generation algorithm
Digital signature verification algorithm
Message M
s
Bob's signature for M
Return signature valid or not valid
(a) Bob signs a message
(b) Alice verifies the signature

Figure 2.7 is a generic model of the process of making and using digital signatures. All of the digital signature schemes in FIPS 186-4 have this structure. Suppose 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. That hash value, together with Bob's private key, serve as input to a digital signature generation algorithm that produces a short block that functions as 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) provides the hash value and Bob's public key as inputs to a digital signature verification algorithm. If the algorithm returns the result that the signature is valid, 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 signature that could be verified for this message 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.

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.

Flow diagram a, Bob signs a message. Step 1. Message M is fed to the final step of message M with attached signature. Step 2. Cryptographic has function. Step 3. Hash value, h. Step 4. Digital signature generation algorithm, Bob’s private key is shared to the digital signature. Step 5. Bob’s signature for M with attached message M. Flow diagram b, Alice verifies the signature. Step 1. Message M with attached signature is fed to the final step, Cryptographic has function. Step 2. Cryptographic has function. Step 3. Hash value, h. Step 4. Digital signature verification algorithm, Bob’s public key is shared to the digital signature. Step 4 produces the output to check, return signature valid or not valid.

Figure 2.8 Public-Key Certificate Use


Unsigned certificate: contains user ID, user's public key, as well as information concerning the CA
Bob's ID information
Bob's public key
CA information
Signed certificate
H
Generate hash code of unsigned certificate
SG
Generate digital signature using CA's private key
H
Generate hash code of certificate not including signature
SV
Return signature valid or not valid
Verify digital signature using CA's public key
Create signed digital certificate
Use certificate to verify Bob'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 Bob 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.

The key steps can be summarized as follows:

1. User software (client) creates a pair of keys: one public and one private.

2. Client prepares an unsigned certificate that includes the user ID and user's public key.

3. User provides the unsigned certificate to a CA in some secure manner. This might require a face-to-face meeting, the use of registered e-mail, or happen via a Web form with e-mail verification.

4. CA creates a signature as follows:
    a. CA uses a hash function to calculate the hash code of the unsigned certificate.
    A hash function is one that maps a variable-length data block or message into a fixed-length value called a hash code, such as SHA family that we will discuss in Sections 2.2 and 21.1.

    b. CA generates digital signature using the CA's private key and a signature generation algorithm.

5. CA attaches the signature to the unsigned certificate to create a signed certificate.

6. CA returns the signed certificate to client.

7. Client may provide the signed certificate to any other user.

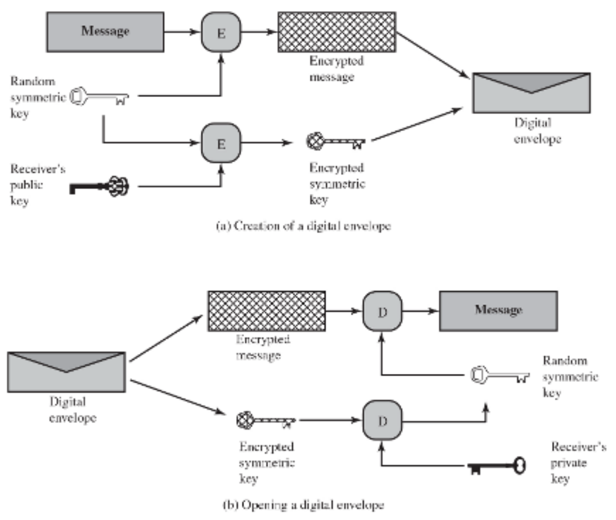
8. Any user may verify that the certificate is valid as follows:
    a. User calculates the hash code of certificate (not including signature).

    b. User verifies digital signature using CA's public key and the signature verification algorithm. The algorithm returns a result of either signature valid or invalid.

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 will examine most of these applications in Part Five.

The diagram is as follows. User software creates a pair of keys, one public and one private. User prepares an unsigned certificate that includes the Bob's I D and Bob's public key as well as information concerning the C A. C A uses a hash function to calculate the hash code of the unsigned certificate. C A generates digital signature using the CA's private key and a signature generation algorithm. C A attaches the signature to the unsigned certificate to create a signed certificate. C A returns the signed certificate to Bob. User calculates the hash code of certificate, not including signature. User verifies digital signature using C A's public key and the signature verification algorithm. The algorithm returns a result of either signature valid or invalid.

Figure 2.9 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 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.

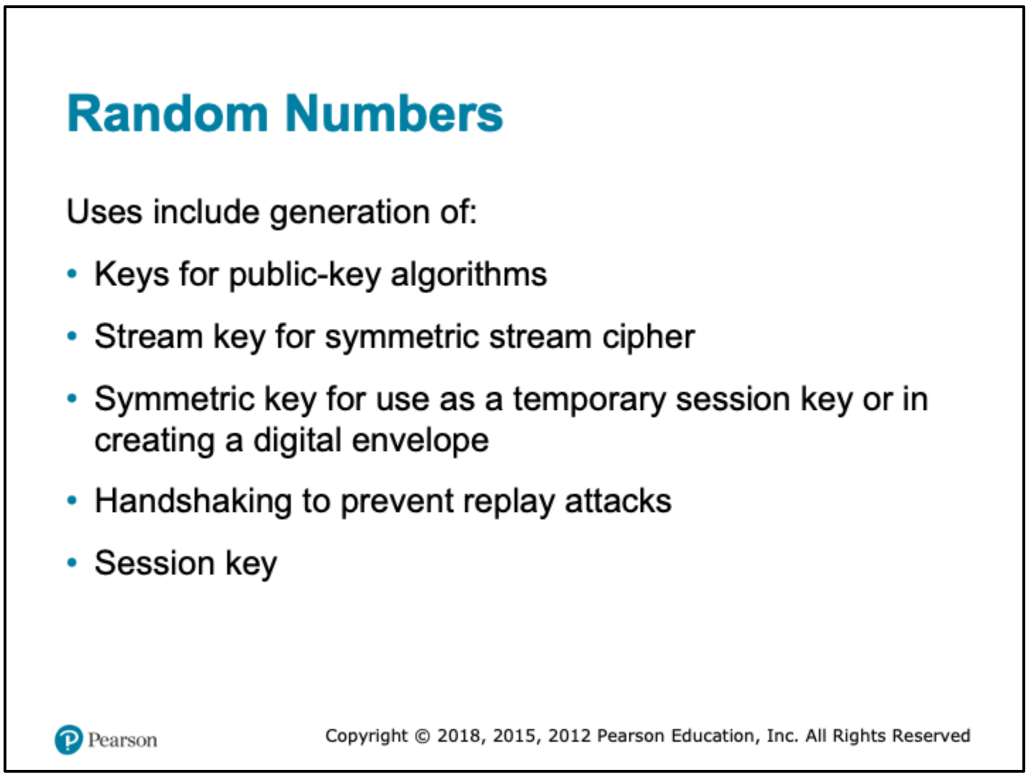
3. Encrypt that message using symmetric encryption the one-time key.

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.

Diagram a, creation of digital envelope. Diagram a, creation of digital envelope. Step 1. Prepare a message. Step 2. Generate a random symmetric key. Step 3. Encrypt the message using symmetric encryption key. Step 4. Encrypt the key using public-key encryption with receiver’s public key. Step 5. The encrypted key to the encrypted message are attached and provides the output as digit envelope. Diagram b, opening a digital envelope. Step 1. Digital envelope is generated as encrypted message and encrypted symmetric key. Step 2. The encrypted message and encrypted key is fed into decryption algorithm. The decryption algorithm gives the output as message. Step 4. The random symmetric key and receiver’s private key are decrypted.



Random Numbers

Uses include generation of:

- Keys for public-key algorithms
- Stream key for symmetric stream cipher
- Symmetric key for use as a temporary session key or in creating a digital envelope
- Handshaking to prevent replay attacks
- Session 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.

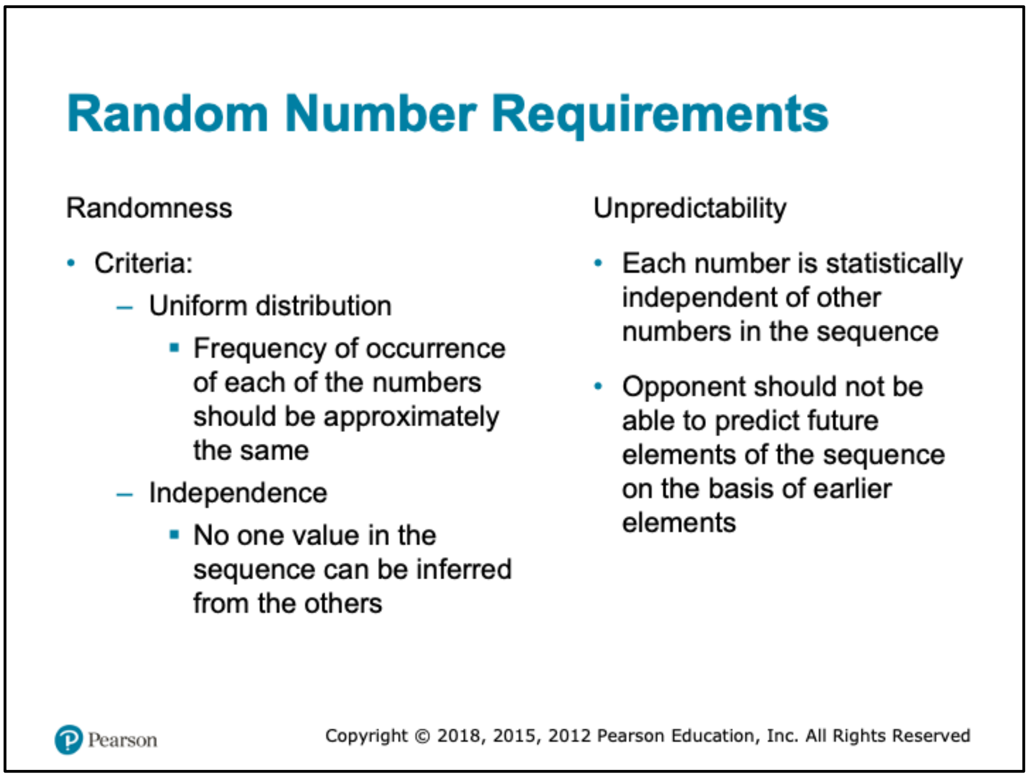
• Generation of a stream key for symmetric stream cipher.

• 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.



Random Number Requirements

Randomness

- Criteria:
  - Uniform distribution
    - 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

Unpredictability

- Each number is statistically independent of other numbers in the sequence
- Opponent should not be able to predict future elements of the sequence on the basis of earlier elements

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.

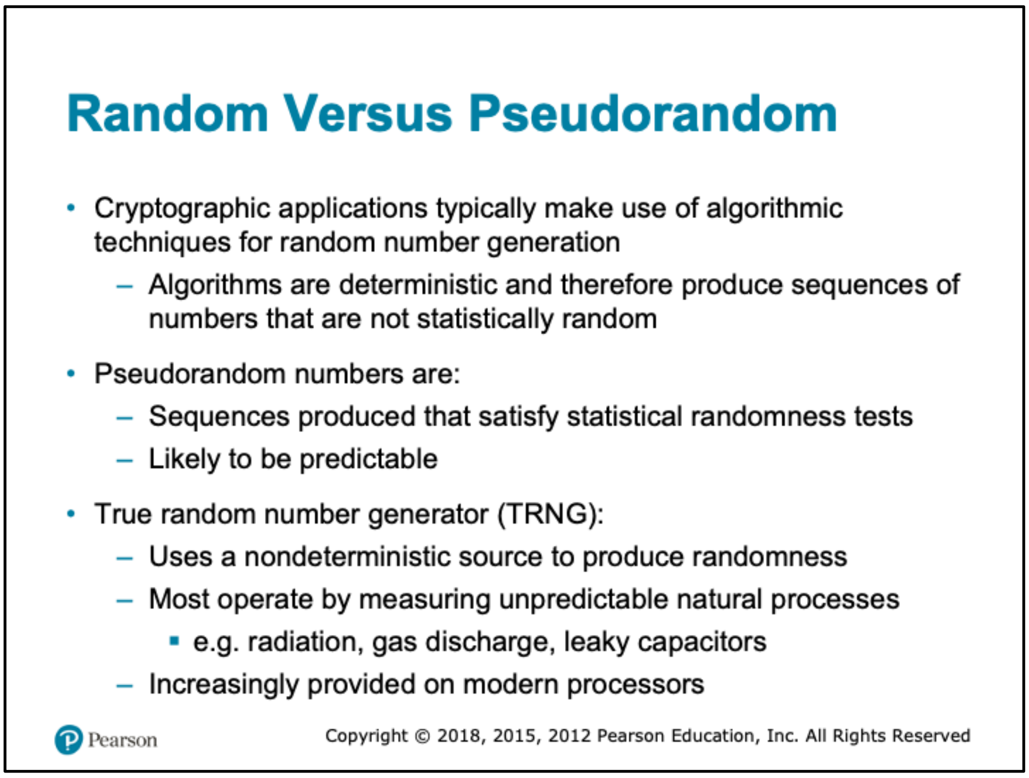
• Independence: No one value in the sequence can be inferred from the others.

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.
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.



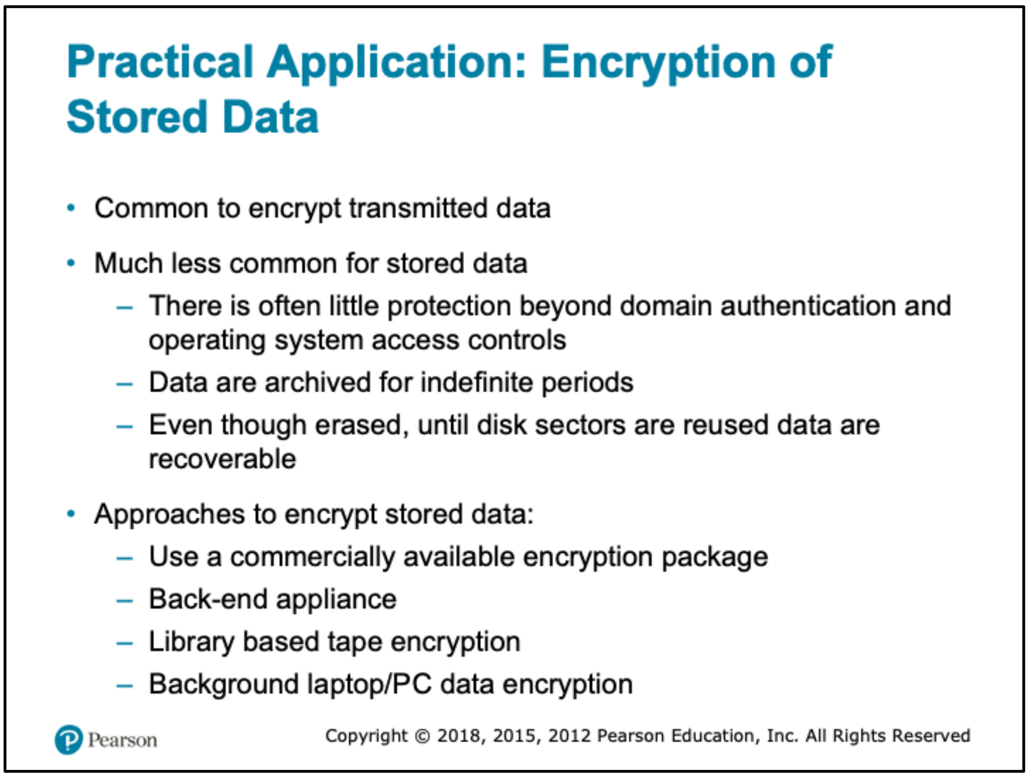
Random Versus Pseudorandom

- Cryptographic applications typically make use of algorithmic techniques for random number generation
  - Algorithms are deterministic and therefore produce sequences of numbers that are not statistically random
- Pseudorandom numbers are:
  - Sequences produced that satisfy statistical randomness tests
  - Likely to be predictable
- True random number generator (TRNG):
  - Uses a nondeterministic source to produce randomness
  - Most operate by measuring unpredictable natural processes
    - e.g. radiation, gas discharge, leaky capacitors
  - Increasingly provided on modern processors

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. That is, under most circumstances, pseudorandom numbers will perform as well as if they were random for a given use. The phrase “as well as” is unfortunately subjective, but the use of pseudorandom numbers is widely accepted. The same principle applies in statistical applications, in which a statistician takes a sample of a population and assumes the results will be approximately the same as if the whole population were measured.

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 capacitors. 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. The first commercially available TRNG that achieves bit production rates comparable with that of PRNGs is the Intel digital random number generator (DRNG) [TAYL11], offered on new multicore chips since May 2012.



Practical Application: Encryption of Stored Data

- Common to encrypt transmitted data
- Much less common for stored data
  - There is often little protection beyond domain authentication and operating system access controls
  - Data are archived for indefinite periods
  - Even though erased, until disk sectors are reused data are recoverable
- Approaches to encrypt stored data:
  - Use a commercially available encryption package
  - Back-end appliance
  - Library based tape encryption
  - Background laptop/PC data encryption

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.

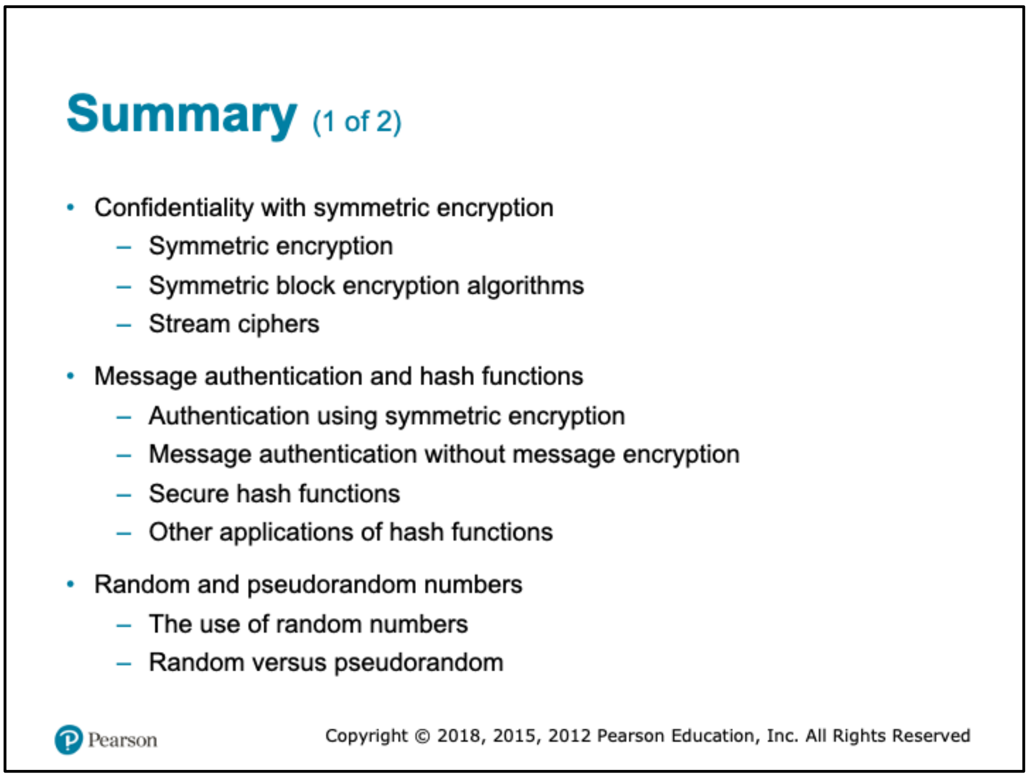
Although it is now routine for businesses to provide a variety of protections, including encryption, for information that is transmitted across networks, via the Internet, or via wireless devices, once data are stored locally (referred to as *data at rest),* there is often little protection beyond domain 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.

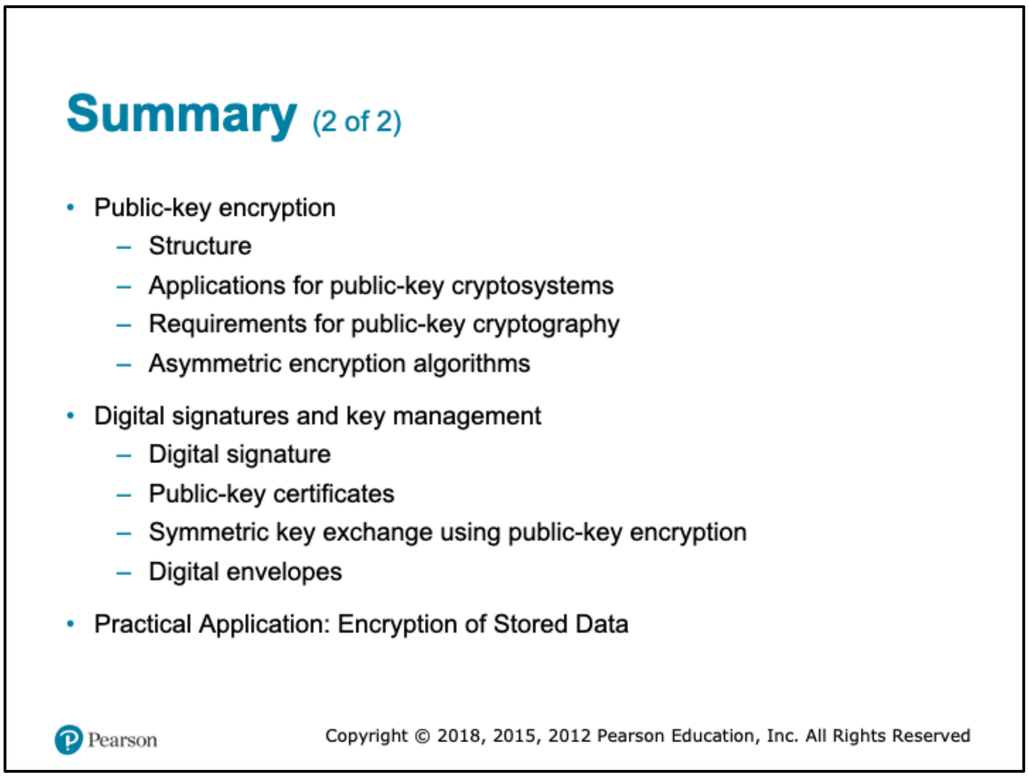
• **Background laptop and PC data encryption**: A number of vendors offer software products that provide encryption that is transparent to the application and the user. Some products encrypt all or designated files and folders. Other products, such as Windows BitLocker and MacOS FileVault, encrypt an entire disk or disk image located on either 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.



Summary (1 of 2)

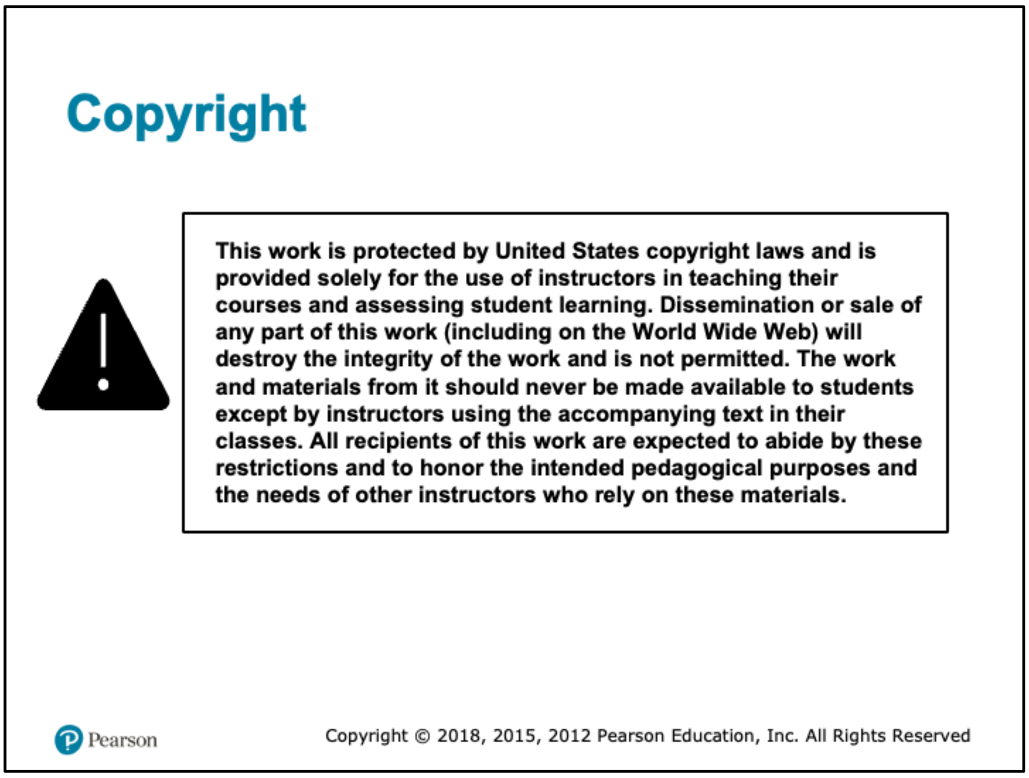
- Confidentiality with symmetric encryption
  - Symmetric encryption
  - Symmetric block encryption algorithms
  - Stream ciphers
- Message authentication and hash functions
  - Authentication using symmetric encryption
  - Message authentication without message encryption
  - Secure hash functions
  - Other applications of hash functions
- Random and pseudorandom numbers
  - The use of random numbers
  - Random versus pseudorandom

Chapter 2 summary.



Summary (2 of 2)

- Public-key encryption
  - Structure
  - Applications for public-key cryptosystems
  - Requirements for public-key cryptography
  - Asymmetric encryption algorithms
- Digital signatures and key management
  - Digital signature
  - Public-key certificates
  - Symmetric key exchange using public-key encryption
  - Digital envelopes
- Practical Application: Encryption of Stored Data



Copyright

Pearson