


ECE 4309

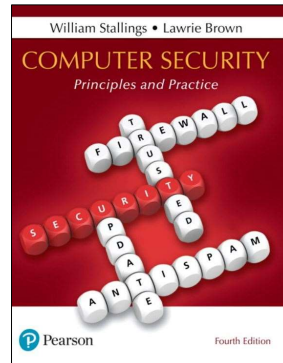
Basics of cryptography – part 1

Dr. Valerio Formicola



Computer Security: Principles and Practice

Fourth Edition



Chapter 2 and 20

Cryptographic Tools



Copyright © 2018, 2015, 2012 Pearson Education, Inc. All Rights Reserved

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.

Cryptography


It's a technic that can be used in various ways to provide all CIA properties, Confidentiality, Integrity and Availability.

Techniques used in cryptography with different purposes:

- *Symmetric encryption*
- *Secure message hashing (hash functions)*
- *Asymmetric encryption*



Confidentiality with symmetric encryption



Confidentiality

- It can be achieved if we avoid anybody able to observe a message, is still not able to understand the content of the message
 - e.g., the attacker might capture it on a transmission medium, like cable or air (type of attack called as *eavesdropping* attack, a type of passive attack)



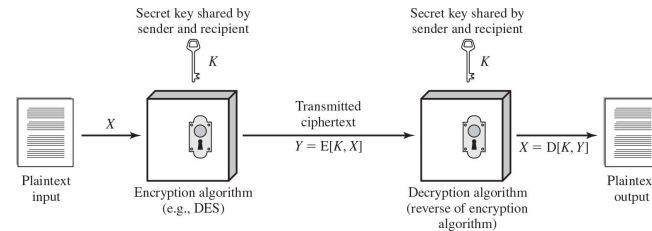
Symmetric Encryption

- Also referred to as:
 - Conventional encryption
 - Secret-key or single-key encryption
- Only alternative before public-key encryption in 1970's
 - Still most widely used alternative
- Has five ingredients:
 - Plaintext
 - Encryption algorithm
 - Secret key
 - Ciphertext
 - Decryption algorithm

At this point the reader should review Section 2.1 . Recall that 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 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 same secret key and produces the original plaintext.

Figure 2.1 Simplified Model of Symmetric Encryption



Note: a plaintext message is not necessarily text. It is anything that you can use with proper decoding; for example, a text with letters or a multimedia form, or simply commands or instructions

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.

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

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.

What can be attacked in such a schema?

- The attacker and anybody else usually knows how the encryption/decryption algorithm works, so that's not a secret
 - Algorithms are mostly standard and public
- The attacker might very well intercept and study the encrypted messages
 - that's why we need a mechanism that it doesn't allow to reverse the process

Attacking Symmetric Encryption

Cryptanalytic Attacks

- Rely on:
 - Mathematical “nature” of the algorithm
 - Some knowledge of the general characteristics of the plaintext
 - e.g., all emails start with Hello X ...
 - Ideal for the crypto-analyst: Some sample plaintext-ciphertext pairs (aka, *cleartext* analysis)
- 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
 - An attacking tool shows results of decryption with a wrong key

A cryptographic algorithm is said to be **breakable** if a crypto-analyst can systematically recover the original message without knowing the key

- *Example of attack in cryptanalysis: frequency analysis* (also known as **counting letters**) is the study of the frequency of letters or groups of letters in a ciphertext.
- *Example of attack in brute force attacks: Dictionary attack* to use names and common words with small substitutions; **credential stuffing** to use databases of users and passwords on different accounts.

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

Block ciphers: The most commonly used symmetric encryption algorithms.

A block cipher processes the plaintext input in fixed-size blocks and produces a block of ciphertext of equal size for each plaintext block

	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.
 - Quite stable in practice but very old (i.e., very studied)
 - 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	Time Required at
			10 ⁹ decryptions/s	10 ¹² decryptions/s
56	DES	$2^{56} \approx 7.2 \times 10^{16}$	$2^{56} \mu s = 1.125$ 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

- Note: book as some error. The number of decryptions are per seconds (not per micro-seconds)

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 (slow when implemented)
 - Uses a 64-bit block size
 - Too many fragmentations for larger data

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
- AES is now widely available in commercial products.

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.

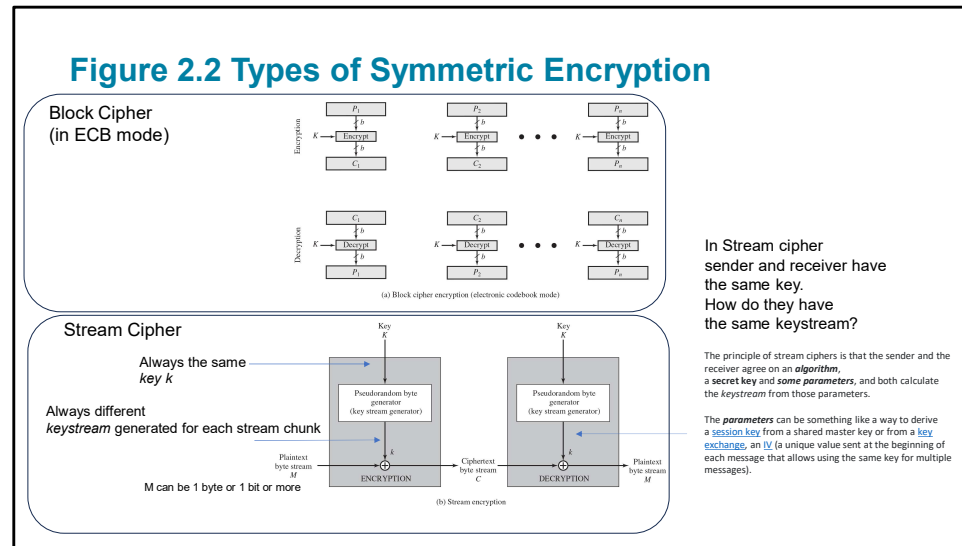


Figure 2.2a shows the ECB mode. A plaintext of length nb is divided into n b -bit blocks (P_1, P_2, \dots, P_n). Each block is encrypted using the same algorithm and the same encryption key, to produce a sequence of n b -bit blocks of ciphertext (C_1, C_2, \dots, C_n).

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 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 .

Practical Security Issues of Block ciphers

- 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 same size) of plaintext is encrypted using the same key
 - Cryptanalysts may be able to exploit regularities in the plaintext
 - E.g., the beginning of a message might follow some patterns like a headline in an email, etc.
- Modes of operation
 - Alternative techniques developed to increase the security of symmetric block encryption for large sequences:
 - Cipher Block Chaining (CBC)
 - Cipher Feedback Mode (CFB)
 - Output Feedback Mode (OFB)
 - Counter Mode (CM)
 - Overcomes the weaknesses of ECB

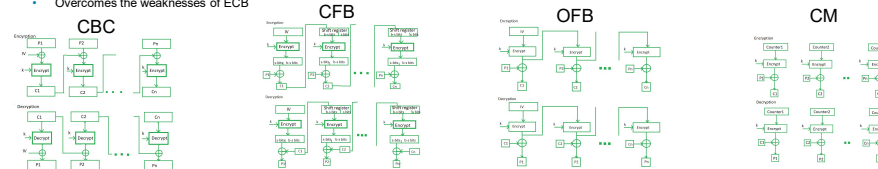
Advantages of using ECB –

- Faster way of encryption in parallel mode.

- Simple way of the block cipher.

Disadvantages of using ECB –

- Prone to cryptanalysis since there is a direct relationship between plaintext and ciphertext.



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.

Applications for Block ciphers

1. **Data Encryption (independently of support media):** Block Ciphers are widely used for the encryption of private and sensitive data such as passwords, credit card details and other information that is transmitted or stored for a communication. This encryption process converts a plain data into non-readable and complex form. Encrypted data can be decrypted only by the authorised person with the private keys.
2. **File and Disk Encryption (encryption of support media, as hard drives):** Block Ciphers are used for encryption of entire files and disks in order to protect their contents and restrict from unauthorised users. The disk encryption softwares such as BitLocker, TrueCrypt also uses block cipher to encrypt data and make it secure.
3. **Virtual Private Networks (VPN):** Virtual Private Networks (VPN) use block cipher for the encryption of data that is being transmitted between the two communicating devices over the internet. This process makes sure that data is not accessed by unauthorised person when it is being transmitted to another user.
4. **Secure Sockets Layer (SSL) and Transport Layer Security (TLS):** SSL and TLS protocols use block ciphers for encryption of data that is transmitted between web browsers and servers over the internet. This encryption process provides security to confidential data such as login credentials, card information etc. (initially, stream ciphers were used, but now they have been replaced)
5. **Digital Signatures:** Block ciphers are used in the digital signature algorithms, to provide authenticity and integrity to the digital documents. This encryption process generates the unique signature for each document that is used for verifying the authenticity and detecting if any malicious activity is detected.

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 and/or initialization parameters


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.

Applications for Stream ciphers

- Stream ciphers are often used for their speed and simplicity of implementation in hardware, and in applications where plaintext comes in quantities of unknowable length like a secure wireless connection (WEP, WPA) or wired (TLS/SSL, deprecated). If a block cipher (not operating in a stream cipher mode) were to be used in this type of application, the designer would need to choose either transmission efficiency or implementation complexity, since block ciphers cannot directly work on blocks shorter than their block size.
- Mostly used for Real Time applications, e.g., media stream cryptography in DVD/BD players, etc.



**More details about
algorithms in
symmetric key**



Cryptography in symmetric keys algorithms

- Classified along three independent dimensions:
 - The type of operations used for transforming plaintext to ciphertext
 - **Substitution** – each element in the plaintext is mapped into another element
 - **Transposition** – elements in plaintext are rearranged
 - The number of keys used
 - Sender and receiver use same key – symmetric
 - Sender and receiver each use a different key - asymmetric
 - The way in which the plaintext is processed
 - Block cipher – processes input one block of elements at a time
 - Stream cipher – processes the input elements continuously

Cryptographic systems are generically classified along three independent dimensions:

1. The type of operations used for transforming plaintext to ciphertext. All encryption algorithms are based on two general principles: substitution, in which each element in the plaintext (bit, letter, group of bits or letters) is mapped into another element, and transposition, in which elements in the plaintext are rearranged. The fundamental requirement is that no information be lost (i.e., that all operations be reversible). Most systems, referred to as product systems, involve multiple stages of substitutions and transpositions.

2. The number of keys used. If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver each use a different key, the system is referred to as asymmetric, two-key, or public-key encryption.

3. The way in which the plaintext is processed. 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.

Table 20.1 Types of Attacks on Encrypted Messages

Difficulty level ↑	Type of Attack	Known to Cryptanalyst
	Ciphertext only (unrealistic under attack)	<ul style="list-style-type: none"> • Encryption algorithm • Ciphertext to be decoded
	Known plaintext (most common target of defense)	<ul style="list-style-type: none"> • Encryption algorithm • Ciphertext to be decoded • One or more plaintext-ciphertext pairs formed with the secret key (includes when the plaintext is a partial message that follows a pattern, e.g., a common text written in English during communications in headers or banners)
	Chosen plaintext (less common, but possible)	<ul style="list-style-type: none"> • Encryption algorithm • Ciphertext to be decoded • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key (the analyst may deliberately pick patterns that can be expected to reveal the structure of the key, e.g., a hacker writes in the commented source code header some information in a software company)
	Chosen ciphertext (rare)	<ul style="list-style-type: none"> • Encryption algorithm • Ciphertext to be decoded • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key (E.g., analyst submit encrypted queries (not by him/her but with the unknown key) and retrieves the answer in plaintext, aka "lunchtime attacks".)
	Chosen text	<ul style="list-style-type: none"> • Encryption algorithm • Ciphertext to be decoded • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key

The process of attempting to discover the plaintext or key is known as **cryptanalysis**. The strategy used by the cryptanalyst depends on the nature of the encryption scheme and the information available to the cryptanalyst.

Table 20.1 summarizes the various types of cryptanalytic attacks, based on the amount of information known to the cryptanalyst. The most difficult problem is presented when all that is available is the *ciphertext only*. In some cases, not even the encryption algorithm is known, but in general we can assume that the opponent does know the algorithm used for encryption. One possible attack under these circumstances is the brute-force approach of trying all possible keys. If the key space is very large, this becomes impractical. Thus, the opponent must rely on an analysis of the ciphertext itself, generally applying various statistical tests to it. To use this approach, the opponent must have some general idea of the type of plaintext that is concealed, such as English or French text, an EXE file, a Java source listing, an accounting file, and so on.

The ciphertext-only attack is the easiest to defend against because the opponent has the least amount of information to work with. In many cases, however, the analyst has more information. The analyst may be able to capture one or more plaintext messages as well as their encryptions. Or the analyst may know that certain plaintext patterns will appear in a

message. For example, a file that is encoded in the Postscript format always begins with the same pattern, or there may be a standardized header or banner to an electronic funds transfer message, and so on. All these are examples of *known plaintext*. With this knowledge, the analyst may be able to deduce the key on the basis of the way in which the known plaintext is transformed.

Closely related to the known-plaintext attack is what might be referred to as a probable-word attack. If the opponent is working with the encryption of some general prose message, he or she may have little knowledge of what is in the message. However, if the opponent is after some very specific information, then parts of the message may be known. For example, if an entire accounting file is being transmitted, the opponent may know the placement of certain key words in the header of the file. As another example, the source code for a program developed by a corporation might include a copyright statement in some standardized position. If the analyst is able somehow to get the source system to insert into the system a message chosen by the analyst, then a *chosen-plaintext* attack is possible. In general, if the analyst is able to choose the messages to encrypt, the analyst may deliberately pick patterns that can be expected to reveal the structure of the key.

Table 20.1 lists two other types of attack: chosen ciphertext and chosen text. These are less commonly employed as cryptanalytic techniques but are nevertheless possible avenues of attack.

Only relatively weak algorithms fail to withstand a ciphertext-only attack. Generally, an encryption algorithm is designed to withstand a known-plaintext attack.

Computationally Secure Encryption Schemes

- Encryption is **computationally secure** if:
 - **Cost of breaking** cipher exceeds value of information
 - **Time required to break** cipher exceeds the useful lifetime of the information
 - E.g., time to break a key is 1 hour. Time to change the key is 10 seconds
 - E.g., even worst, the information encrypted is not useful after 1 hour
- Usually very difficult to estimate the amount of effort required to break
 - **Can estimate time/cost in a brute-force attack**
 - **Time to try half of the keys possible**

An encryption scheme is **computationally secure** if the ciphertext generated by the scheme meets one or both of the following criteria:

- The cost of breaking the cipher exceeds the value of the encrypted information.
- The time required to break the cipher exceeds the useful lifetime of the information.

Unfortunately, it is very difficult to estimate the amount of effort required to cryptanalyze ciphertext successfully. However, assuming there are no inherent mathematical weaknesses in the algorithm, then a brute-force approach is indicated, and here we can make some reasonable estimates about costs and time.

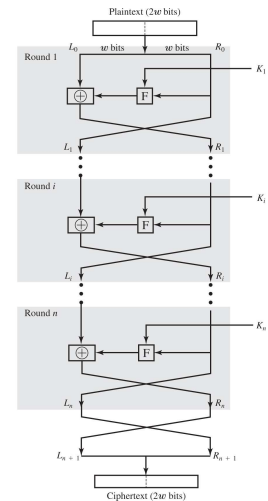
A brute-force approach involves trying every possible key until an intelligible translation of the ciphertext into plaintext is obtained. On average, half of all possible keys must be tried to achieve success. This type of attack is discussed in Section 2.1 .

Figure 20.1 Classical Feistel Network

**Common schema for many algorithms,
most notably DES**

Encryption in DES:
 $2w = 64$ bits (1 word w is 4 bytes)
 Primary K size = 56 bit
 # Rounds = 16
 # Subkeys = 16
 # F (round functions) = 1 (same for each round)

Decryption in DES:
 Subkeys as before but in reverse order
 Round as before but in reverse order



Many symmetric block encryption algorithms, including DES, have a structure first described by Horst Feistel of IBM in 1973 [FEIS73] and shown in Figure 20.1. The inputs to the encryption algorithm are a plaintext block of length $2w$ bits and a key K . The plaintext block is divided into two halves, L_0 and R_0 . The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block. Each round i has as inputs L_{i-1} and R_{i-1} , derived from the previous round, as well as a subkey K_i , derived from the overall K . In general, the subkeys K_i are different from K and from each other and are generated from the key by a subkey generation algorithm.

All rounds have the same structure. A substitution is performed on the left half of the data. This is done by applying a round function F to the right half of the data and then taking the exclusive-OR (XOR) of the output of that function and the left half of the data. The round function has the same general structure for each round but is parameterized by the round subkey K_i . Following this substitution, a permutation is performed that consists of the interchange of the two halves of the data.

Round 1. The plaintext, $2w$ bits, is divided into 2 parts, $L_{sub\ 0}$ and $R_{sub\ 0}$, each of w bits. The transformed $R_{sub\ 0}$ into

F, with input $K_{sub\ 1}$, is added to $L_{sub\ 0}$. The output is $L_{sub\ 1}$ and $R_{sub\ 1}$. Ellipsis. Round i . the input is transformed and added the partial output F with input $K_{sub\ i}$, to get output $L_{sub\ i}$ and $R_{sub\ i}$. Round n . The input is transformed and added the partial output F with input $K_{sub\ n}$, to get output $L_{sub\ n}$ and $R_{sub\ n}$. The output is crossed again to get $L_{sub\ start\ expression\ n + 1\ end\ expression}$ and $R_{sub\ start\ expression\ n + 1\ end\ expression}$. The output is combined to get ciphertext of 2 bits.

Block Cipher Structure

- Symmetric block cipher consists of:
 - A sequence of rounds
 - With substitutions and permutations controlled by key
- Parameters and design features:
 - Block size: larger is better, 128 bits usually
 - Key size: larger better but slower
 - Number of rounds: more is better, usually 16 rounds
 - Subkey generation algorithm: more complex is better
 - Round function: more complex is better
 - Fast software encryption/decryption: software versions are cheaper but slower
 - Ease of analysis: tradeoff more analysis possible for clearness vs more difficult for strength

The Feistel structure is a particular example of the more general structure used by all symmetric block ciphers. In general, a symmetric block cipher consists of a sequence of rounds, with each round performing substitutions and permutations conditioned by a secret key value. The exact realization of a symmetric block cipher depends on the choice of the following parameters and design features:

- **Block size:** Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed. A block size of 128 bits is a reasonable tradeoff and is nearly universal among recent block cipher designs.
- **Key size:** Larger key size means greater security but may decrease encryption/ decryption speed. The most common key length in modern algorithms is 128 bits.
- **Number of rounds:** The essence of a symmetric block cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.
- **Subkey generation algorithm :** Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.

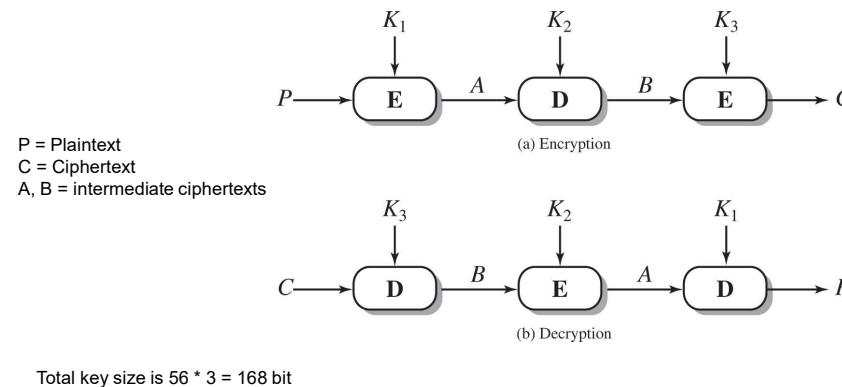
- **Round function:** Again, greater complexity generally means greater resistance to cryptanalysis.

There are two other considerations in the design of a symmetric block cipher:

- **Fast software encryption/decryption:** In many cases, encryption is embedded in applications or utility functions in such a way as to preclude a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.
- **Ease of analysis:** Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength. DES, for example, does not have an easily analyzed functionality.

Decryption with a symmetric block cipher is essentially the same as the encryption process. The rule is as follows: Use the ciphertext as input to the algorithm, but use the subkeys K_i in reverse order. *That is, use K_n in the first round, K_{n-1} in the second round, and so on until K_1 is used in the last round.* This is a nice feature because it means we need not implement two different algorithms, one for encryption and one for decryption.

Figure 20.2 Triple DES



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 uses three keys and three executions of the DES algorithm. The function follows an encrypt-decrypt-encrypt (EDE) sequence (see Figure 20.2a):

$$C = E(K_3, D(K_2, E(K_1, P)))$$

where: C = ciphertext; P = plaintext; $E[K, X]$ = encryption of X using key K , and $D[K, Y]$ = decryption of Y using key K . Decryption is simply the same operation with the keys reversed (Figure 20.2b):

$$P = D(K_1, E(K_2, D(K_3, C)))$$

There is no cryptographic significance to the use of decryption for the second stage of 3DES encryption. Its only advantage is that it allows users of 3DES to decrypt data encrypted by users of the older single DES:

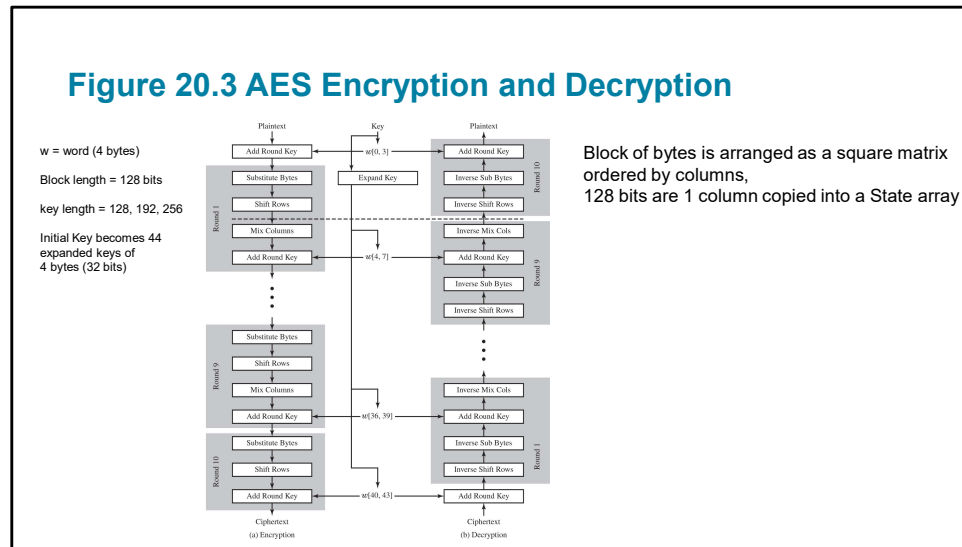
$$C = E(K1, D(K1, E(K1, P))) = E[K, P]$$

With three distinct keys, 3DES has an effective key length of 168 bits. FIPS 46-3 also allows for the use of two keys, with $K1 = K3$; this provides for a key length of 112 bits. FIPS 46-3 includes the following guidelines for 3DES:

- 3DES is the FIPS approved symmetric encryption algorithm of choice.
- The original DES, which uses a single 56-bit key, is permitted under the standard for legacy systems only. New procurements should support 3DES.
- Government organizations with legacy DES systems are encouraged to transition to 3DES.
- It is anticipated that 3DES and the Advanced Encryption Standard (AES) will coexist as FIPS-approved algorithms, allowing for a gradual transition to AES. It is easy to see that 3DES is a formidable algorithm. Because the underlying cryptographic algorithm is DEA, 3DES can claim the same resistance to cryptanalysis based on the algorithm as is claimed for DEA. Further, with a 168-bit key length, brute-force attacks are effectively impossible.

Ultimately, AES is intended to replace 3DES, but this process will take a number of years. NIST anticipates that 3DES will remain an approved algorithm (for U.S. government use) for the foreseeable future.

The encryption and decryption process involves running encryption and decryption process three times. During encryption, the keys k_1 to k_3 is used to encrypt a message P to cipher text, C . The encryption, decryption, and encryption processes are run in series with key K_1 to k_3 as inputs to get outputs, A , B , and C , respectively. During decryption, the keys k_3 to k_1 is used to encrypt a cipher C to message text, P . The decryption, encryption, and decryption processes are run in series with key K_3 to k_1 as inputs to get outputs, B , A , and P , respectively.



The Advanced Encryption Standard (AES) was issued as a federal information processing standard FIPS 197 (Advanced Encryption Standard, November 2001). It is intended to replace DES and triple DES with an algorithm that is more secure and efficient.

AES uses a block length of 128 bits and a key length that can be 128, 192, or 256 bits. In the description of this section, we assume a key length of 128 bits, which is likely to be the one most commonly implemented.

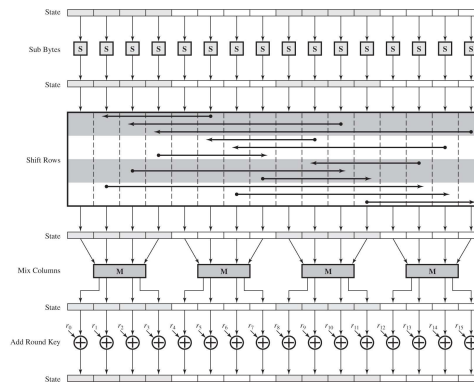
Figure 20.3 shows the overall structure of AES. The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a square matrix of bytes. This block is copied into the State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix. Similarly, the 128-bit key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words; each word is 4 bytes and the total key schedule is 44 words for the 128-bit key. The ordering of bytes within a matrix is by column. So, for example, the first 4 bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the in matrix, the second 4 bytes occupy the second column, and so on. Similarly, the first 4 bytes of the expanded key, which form a word, occupy the first column of the w matrix.

The following comments give some insight into AES:

1. One noteworthy feature of this structure is that it is not a Feistel structure. Recall that in the classic Feistel structure, half of the data block is used to modify the other half of the data block, and then the halves are swapped. AES does not use a Feistel structure but processes the entire data block in parallel during each round using substitutions and permutation.
2. The key that is provided as input is expanded into an array of forty-four 32-bit words, $w[i]$. Four distinct words (128 bits) serve as a round key for each round.
3. Four different stages are used, one of permutation and three of substitution:
 - Substitute Bytes: Uses a table, referred to as an S-box, to perform a byte-by-byte substitution of the block
 - Shift Rows: A simple permutation that is performed row by row
 - Mix Columns: A substitution that alters each byte in a column as a function of all of the bytes in the column
 - Add Round key: A simple bitwise XOR of the current block with a portion of the expanded key

The encryption process is as follows. Plaintext, add round key with key left bracket 0, 3 right bracket. Round 1. Substitute bytes, shift rows, mix columns, add round key w left bracket 4, 7 right bracket. ellipsis. Round 9. Substitute bytes, shift rows, mix columns, add round key w left bracket 36, 39 right bracket. Round 10. Substitute bytes, shift rows, add round key w left bracket 40, 43 right bracket. output, cipher text. The decryption process is as follows. Cipher text. Add round key w left bracket 40, 43 right bracket. Round 1. Inverse shift, inverse sub bytes, add round key, and inverse mix Columns. Ellipsis. Round 9. Inverse shift rows, inverse sub bytes, add round key w left bracket 4, 7 right bracket, inverse mix Columns. Round 10. Inverse shift rows, inverse sub bytes, and add round key w left bracket 0, 3 right bracket. The output is plain text. The key expanded to w left parenthesis 4, 7 right bracket, w left bracket 36, 39 right bracket, and w left bracket 40, 43 right bracket.

Figure 20.4 AES Encryption Round



4. The structure is quite simple. For both encryption and decryption, the cipher begins with an Add Round Key stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. Figure 20.4 depicts the structure of a full encryption round.

5. Only the Add Round Key stage makes use of the key. For this reason, the cipher begins and ends with an Add Round Key stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.

6. The Add Round Key stage by itself would not be formidable. The other three stages together scramble the bits, but by themselves would provide no security because they do not use the key. We can view the cipher as alternating operations of XOR encryption (Add Round Key) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure.

7. Each stage is easily reversible. For the Substitute Byte, Shift Row, and Mix Columns stages, an inverse function is used in the decryption algorithm. For the Add Round Key stage, the inverse is achieved by XORing the same round key to the

block, using the result that $A \oplus A \oplus B = B$.

8. As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.

9. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. Figure 20.3 lays out encryption and decryption going in opposite vertical directions. At each horizontal point (e.g., the dashed line in the figure), State is the same for both encryption and decryption.

10. The final round of both encryption and decryption consists of only three stages. Again, this is a consequence of the particular structure of AES and is required to make the cipher reversible.

The encryption process has the following steps. Data is represented as State. The first operation is sub bytes where s boxes are used to perform pi pi pi substitution. The resultant data is arranged in state array, and the permutations are performed row by row on the data. The data output is arranged in state array. Mix columns is performed on the resultant data, which is arranged in state. The data is substituted where each byte is altered as a function of all the bytes in the column. The result is arranged in state array. Add round keys where keys from r sub 0 to r sub 15, are added to the message, and the resultant data is arranged in state array.

Stream Ciphers

- Processes input elements continuously
- Key input to a pseudorandom bit generator
 - Produces stream of random like numbers
 - Unpredictable without knowing input key
 - XOR keystream output with plaintext bytes

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.

In this section, we look at perhaps the most popular symmetric stream cipher, RC4. We begin with an overview of stream cipher structure and then examine RC4.

A typical stream cipher encrypts plaintext 1 byte at a time, although a stream cipher may be designed to operate on 1 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 that has an apparently random character. The output of the generator, called a **keystream**, is combined 1 byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation.

Stream Cipher operations

A typical stream cipher encrypts plaintext 1 byte at a time, although a stream cipher may be designed to operate on 1 bit at a time or on units larger than a byte at a time. 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 that has an apparently random character. The output of the generator, called a **keystream**, is combined 1 byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation.

For example, if the next byte generated by the generator is 01101100 and the next plaintext byte is 11001100, then the resulting ciphertext byte is:

11001100	plaintext
01101100	key stream
10100000	ciphertext

Decryption requires the use of the same pseudorandom sequence:

10100000	ciphertext
01101100	key stream
11001100	plaintext



Security of stream ciphers

- The advantage of a block cipher is that you can reuse keys.
- However, if two plaintexts are encrypted with the same key using a stream cipher, then cryptanalysis is often quite simple [DAWS96]. If the two ciphertext streams are XORed together, the result is the XOR of the original plaintexts. If the plaintexts are text strings, credit card numbers, or other byte streams with known properties, then cryptanalysis may be successful.

Most common standards

- RC4 (proprietary but leaked):
 - Variable key size
 - 1 byte stream
 - Very simple software implementation, so very fast (8 to 16 instructions)
 - Used in SSL/TLS and in WEP, WPA protocols of Wi-Fi
- ChaCha20:
 - Future replacement for RC4

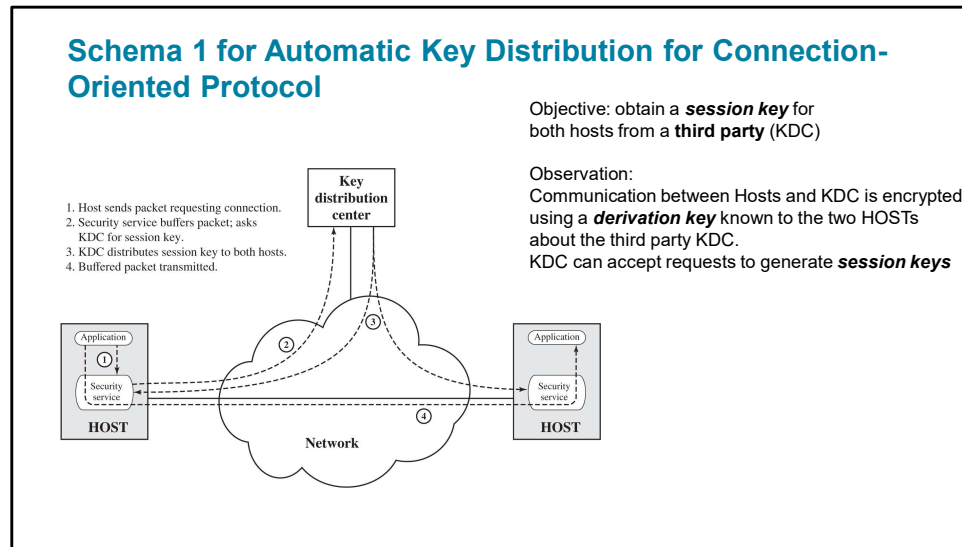


Figure 20.10 illustrates an implementation that satisfies option 4 for end-to-end encryption. In the figure, link encryption is ignored. This can be added, or not, as required. For this scheme, two kinds of keys are identified:

- **Session key:** When two end systems (hosts, terminals, etc.) wish to communicate, they establish a logical connection (e.g., virtual circuit). For the duration of that logical connection, all user data are encrypted with a one-time session key. At the conclusion of the session, or connection, the session key is destroyed.
- **Permanent key:** A permanent key is a key used between entities for the purpose of distributing session keys. The configuration consists of the following elements:
 - **Key distribution center:** The key distribution center (KDC) determines which systems are allowed to communicate with each other. When permission is granted for two systems to establish a connection, the KDC provides a one-time session key for that connection.
 - **Security service module (SSM):** This module, which may consist of functionality at one protocol layer, performs end-to-

end encryption and obtains session keys on behalf of users.

The steps involved in establishing a connection are shown in Figure 20.10 . When one host wishes to set up a connection to another host, it transmits a connection request packet (step 1). The SSM saves that packet and applies to the KDC for permission to establish the connection (step 2). The communication between the SSM and the KDC is encrypted using a master key shared only by this SSM and the KDC. If the KDC approves the connection request, it generates the session key and delivers it to the two appropriate SSMs, using a unique permanent key for each SSM (step 3). The requesting SSM can now release the connection request packet, and a connection is set up between the two end systems (step 4). All user data exchanged between the two end systems are encrypted by their respective SSMs using the one-time session key.

The automated key distribution approach provides the flexibility and dynamic characteristics needed to allow a number of terminal users to access a number of hosts and for the hosts to exchange data with each other.

Another approach to key distribution uses public-key encryption, which is discussed in Chapter 21 .

The steps are as follows. 1. Application through security service in the host sends packet requesting connection. 2. Security service buffers packet, asks K D C for session key. 3. K D C distributes session key to both hosts. 4. Buffered packet transmitted.

Schema 2: Diffie-Hellman

- It is an asymmetric algorithm
- We'll see soon...