Chapter 1:

Introduction

4th Year- Course, CCSIT, UoA

Lecture Goals

- To introduce the student to some basic definitions of information security.
- To describe the conventional cryptography model.
- To supply the student with the some relevant historical background emphasizing the role of Muslim scholars in the field.

Definitions

- **Computer Security** - generic name for the collection of tools designed to protect data and to thwart hackers
- **Network Security** - measures to protect data during their transmission
- **Internet Security** - measures to protect data during their transmission over a collection of interconnected networks

❖ OSI Security Architecture: **ITU-T X.800** "Security Architecture for OSI" defines a systematic way of defining and providing security requirements

Aspects of Security

1. **Security attack:** Any action that compromises information security
2. **Security mechanism:** Process designed to detect, prevent or recover from security attack
3. **Security service:** Service that enhances system's security by utilizing one or more security mechanisms

Security Attack

- Any action that compromises the security of information owned by an organization is a security attack.
- Information security is about how to prevent attacks, or failing that, to detect attacks on information-based systems
- Threat & attack used to mean same thing
- There are two generic types of attacks:
  1. Passive attacks
  2. Active attacks

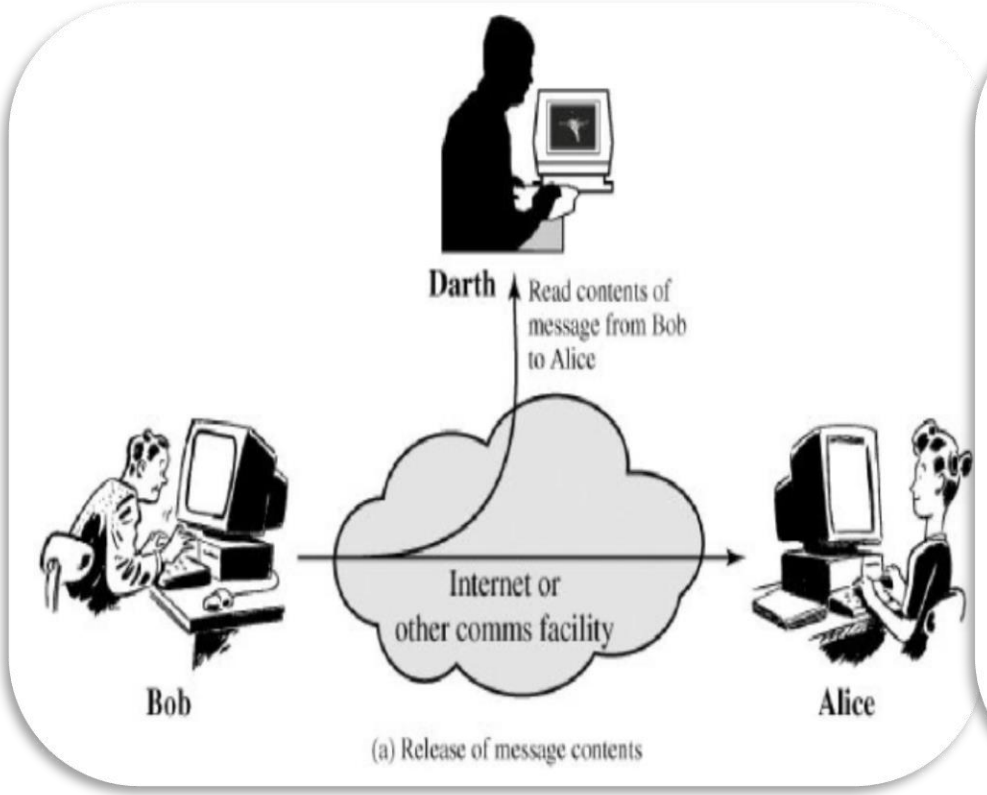
Passive Attacks (1)

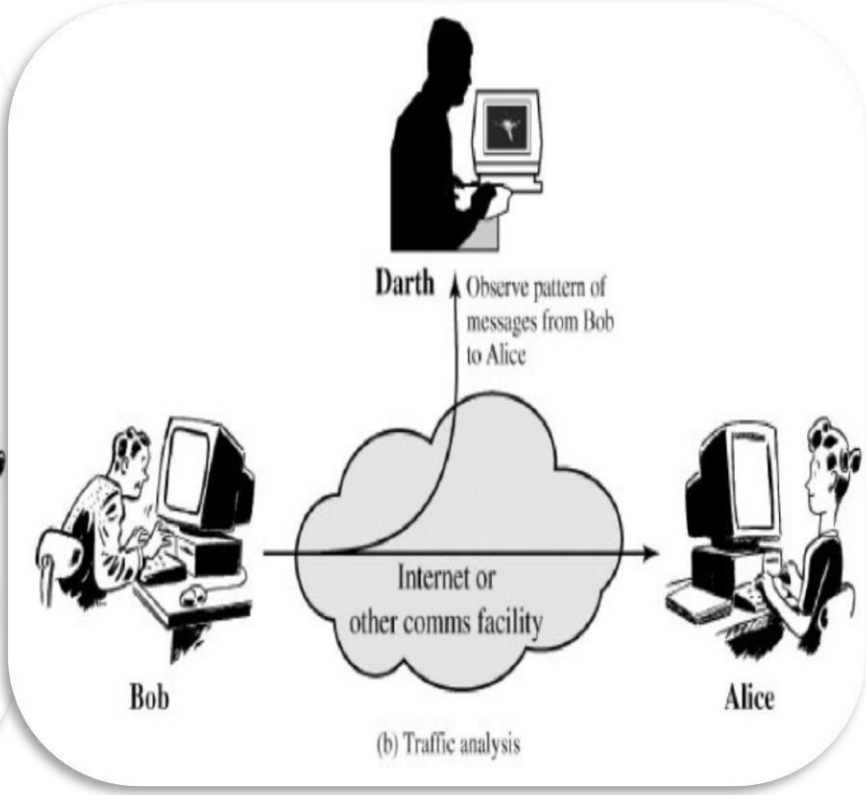
- The strategy for such attack is to eavesdrop, and monitor transmission, in order to obtain information being transmitted

1. Type 1: **Release of message contents**: like tapping on phone line to hear conversation, or getting unauthorized copy of email message
2. Type 2: **Traffic analysis**: This is done by observing message pattern, even if encrypted, and then determining location and identity of parties

- Passive attacks are very difficult to detect; because they make no alteration of data

Passive Attacks (2)



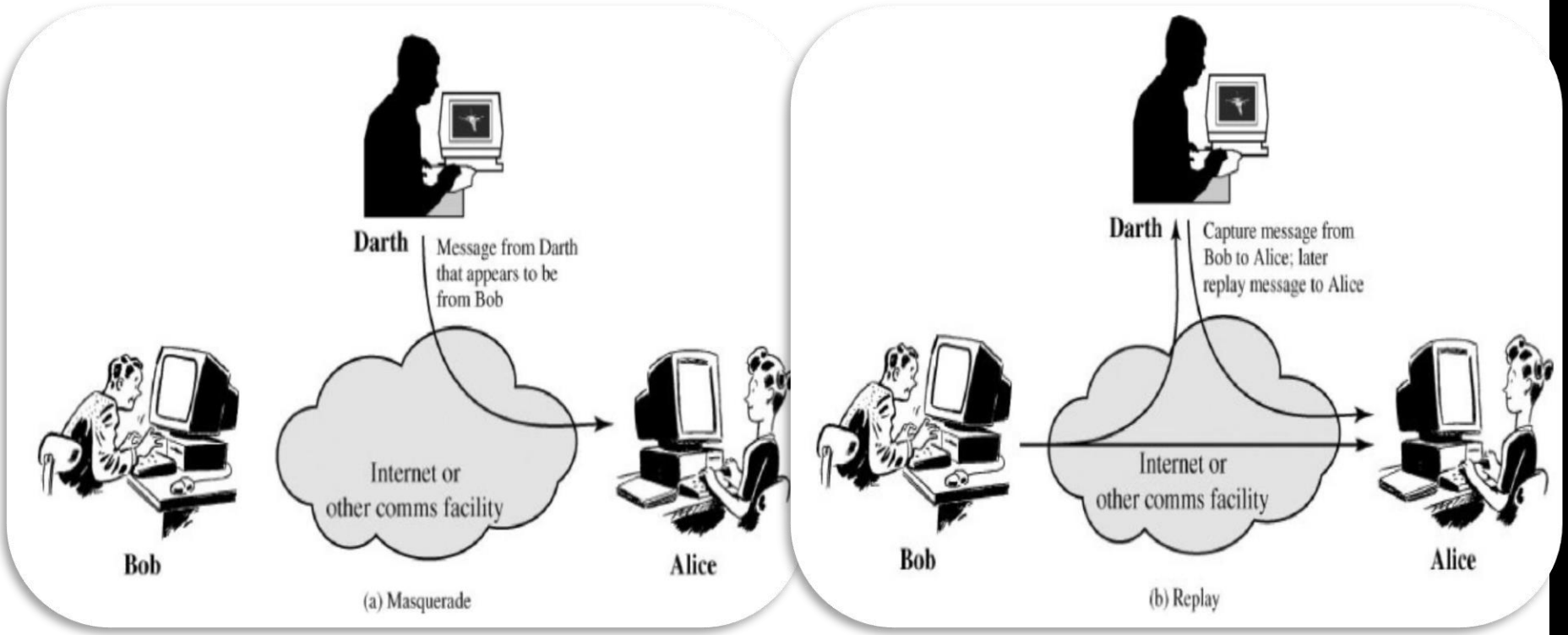


Active Attacks (1)

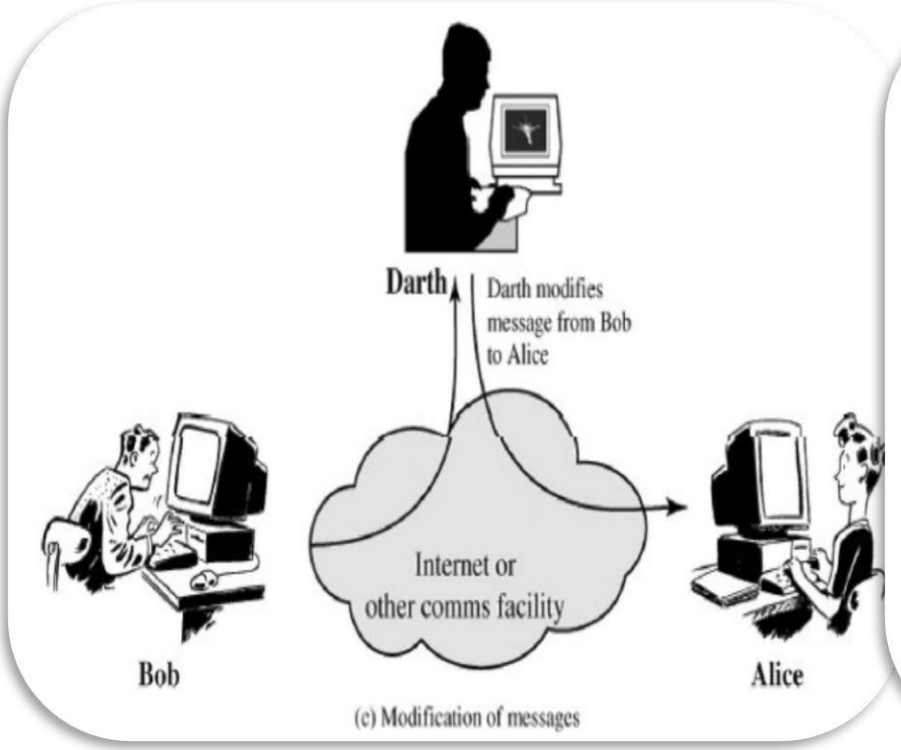
- Modification of transmitted data or creating false data represent an active attack.

1. Type 1- **Masquerade**: Pretending to be a different entity
2. Type 2- **Replay**: Capturing data for subsequent retransmission
3. Type 3- **Modification of message**: Some portion of legitimate message is altered
4. Type 4- **Denial of service**: Disruption of network or system resources by disabling or overloading

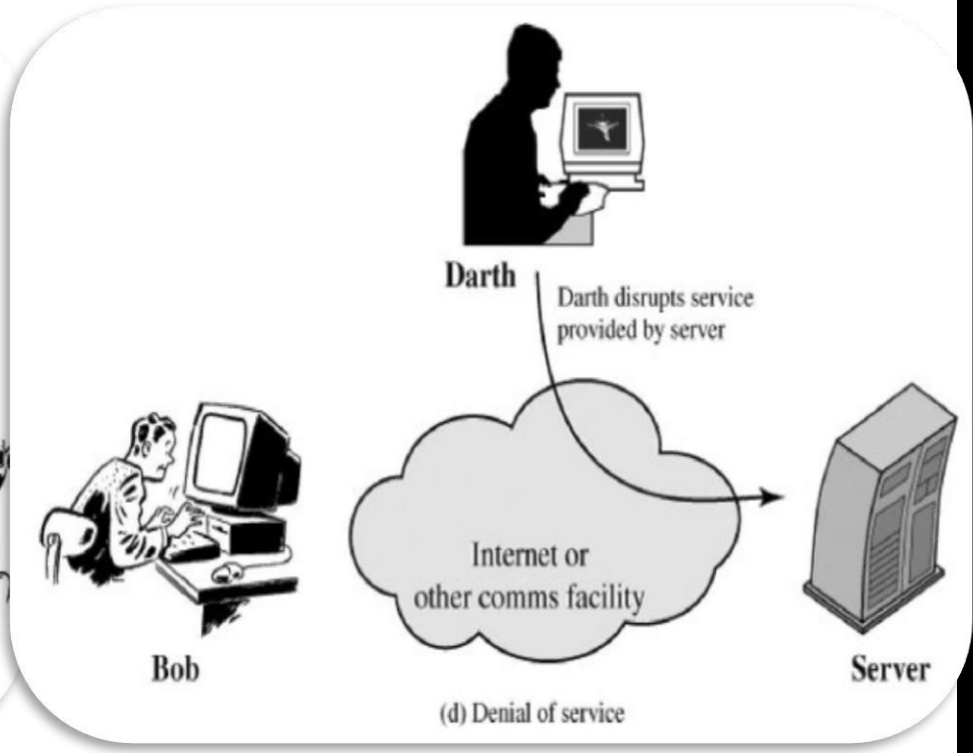
Active Attacks (2)



Active Attacks (3)



(c) Modification of messages



(d) Denial of service

Security Service

- A security service is used to enhance security of data processing systems and information transfers of an organization
- Security services are intended to counter security attacks
- A security service may use one or more security mechanisms
- Security service often replicates functions normally associated with physical documents which, for example, have signatures, dates; need protection from disclosure, tampering, or destruction; be notarized or witnessed; be recorded or licensed

Main Security Services (X.800)

1. **Authentication** - assurance that the communicating entity is the one claimed
2. **Access Control** - prevention of the unauthorized use of a resource
3. **Data Confidentiality** -protection of data from unauthorized disclosure
4. **Data Integrity** - assurance that data received is as sent by an authorized entity
5. **Non-Repudiation** - protection against denial by one of the parties in a communication

Security Mechanism

- Security mechanism is a feature designed to detect, prevent, or recover from a security attack
- There is no single mechanism that will support all services required
- However, there is one particular element underlies many of the security mechanisms in use which is:

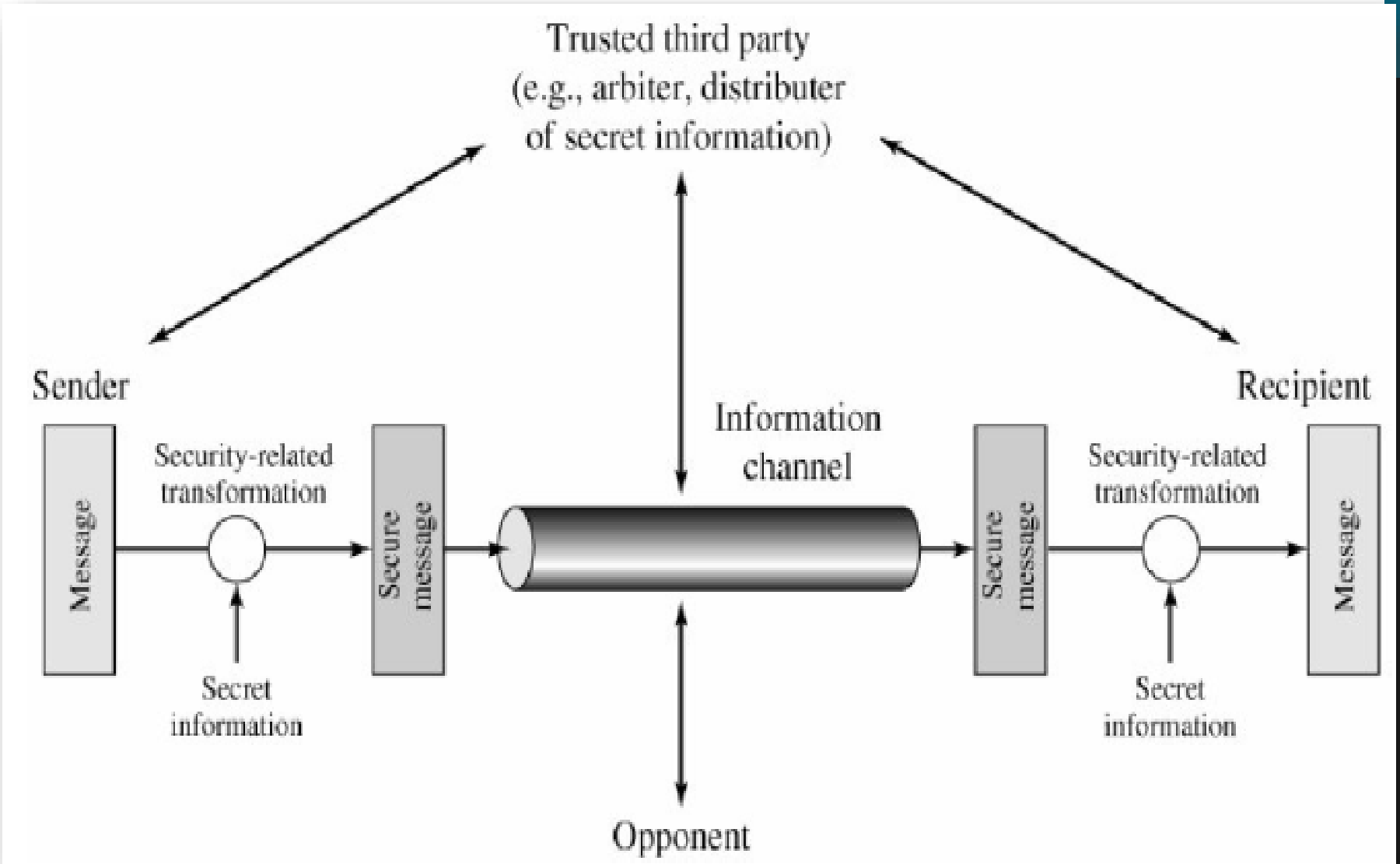
  ❖ **Cryptography**

Security Mechanisms (X.800)

Table 1.4. Relationship between Security Services and Mechanisms

	Mechanism							
Service	Encipherment	Digital Signature	Access Control	Data Integrity	Authentication Exchange	Traffic Padding	Routing Control	Notarization
Peer entity authentication	Y	Y			Y			
Data origin authentication	Y	Y						
Access control			Y					
Confidentiality	Y						Y	
Traffic flow confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Nonrepudiation		Y		Y				Y
Availability				Y	Y			

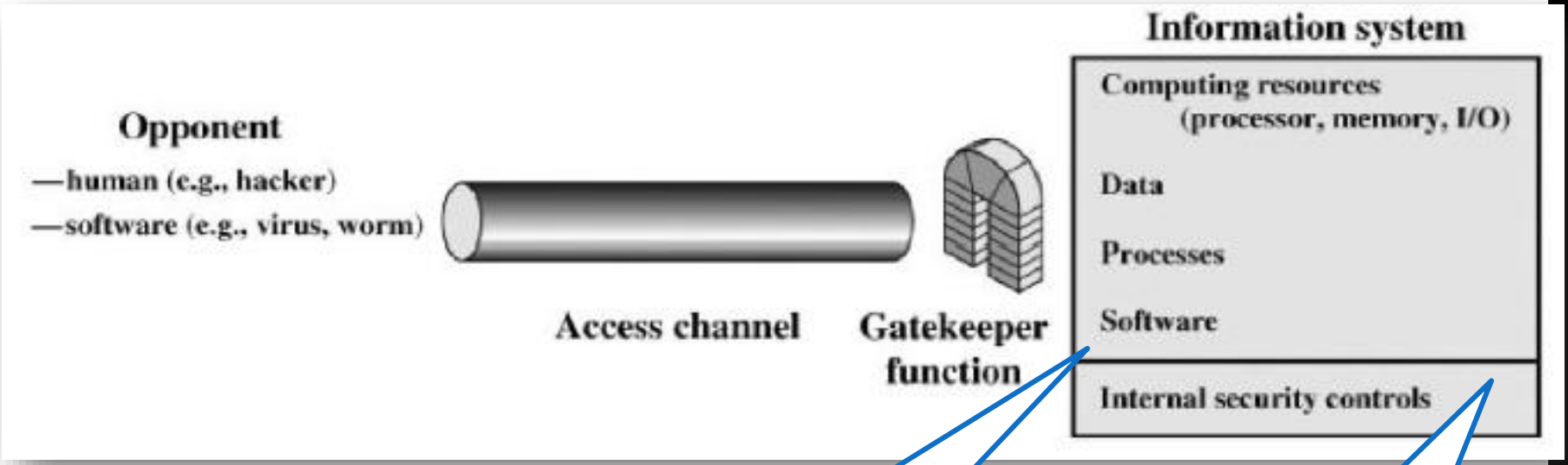
Model for Network Security



Requirements of the Model for Network Security

1. Design a suitable algorithm for the security transformation
2. Generate the secret information (keys) used by the algorithm
3. Develop methods to distribute and share the secret information (keys)
4. Specify a protocol enabling the principals to use the transformation and secret information for a security service

Model For Network (or System) Access Security



Intrusion Prevention

1. Intrusion Detection
2. System Recovery

Requirements of the Model for Network (or System) Access Security

1. Select appropriate gatekeeper functions to identify users
2. Implement security controls to ensure only authorized users access designated information or resources
3. Implement intrusion detection and recovery tools
4. Trusted computer systems may be useful to help implement this model

Cryptography

- **Cryptography** in Greek means “secret writing“. It is the practice and study of (mathematical) techniques for secure communication in the presence of third parties (called adversaries).
- Modern cryptography exists at the intersection of the disciplines of mathematics, computer science, and electrical engineering.
- Cryptographic systems can be divided into two main types:
  1. **Conventional cryptography** (also called symmetric, private-key, or single-key cryptography)
  2. **Public-key cryptography** (also called asymmetric, or two-key cryptography)

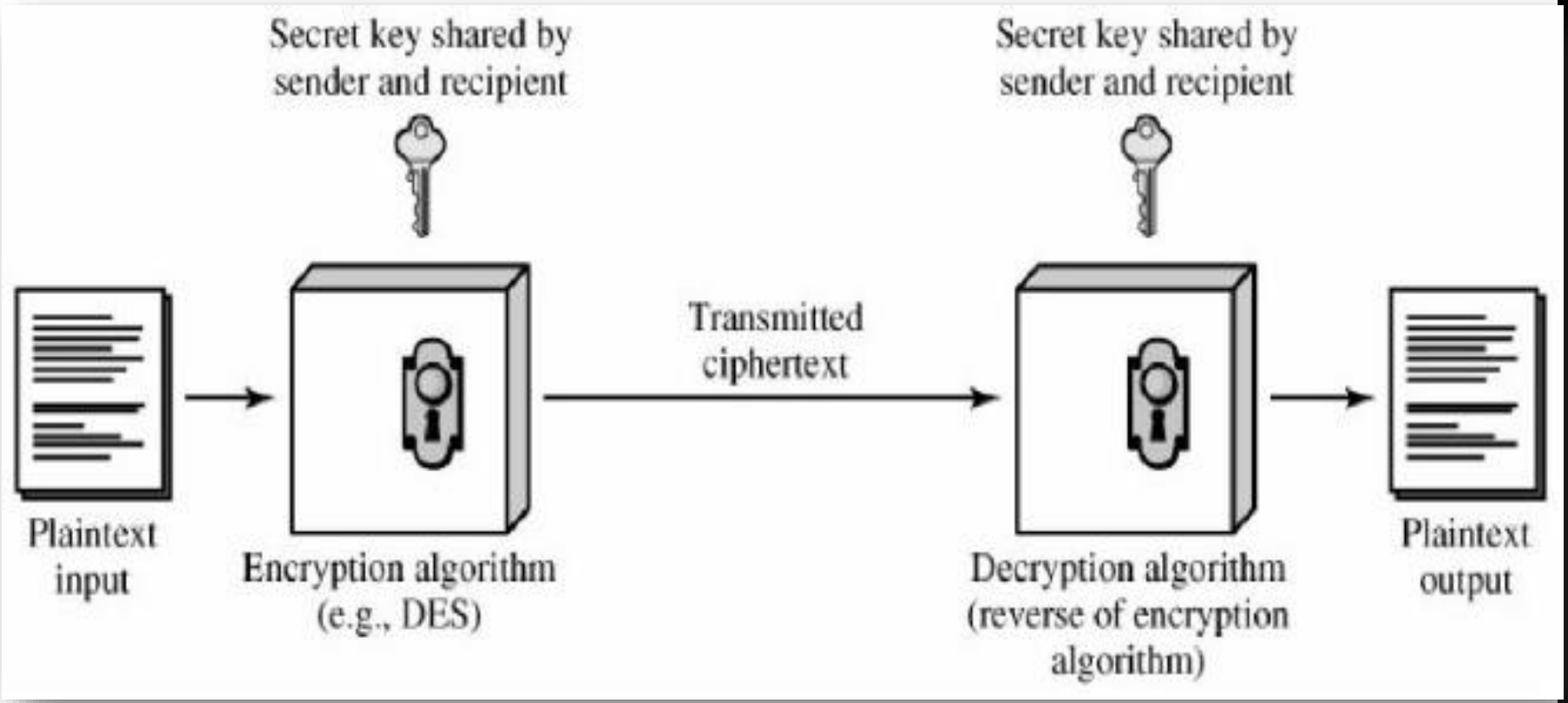
Conventional Cryptography

- **Conventional cryptography** is also called symmetric, private-key, secret key, or single-key cryptography
- In conventional cryptography, sender and recipient share a common key
- All **classical encryption** algorithms (i.e. before computer age) are from this type.
- Conventional cryptography was the only type prior to invention of **public-key cryptography** in 1970's and by far most widely used

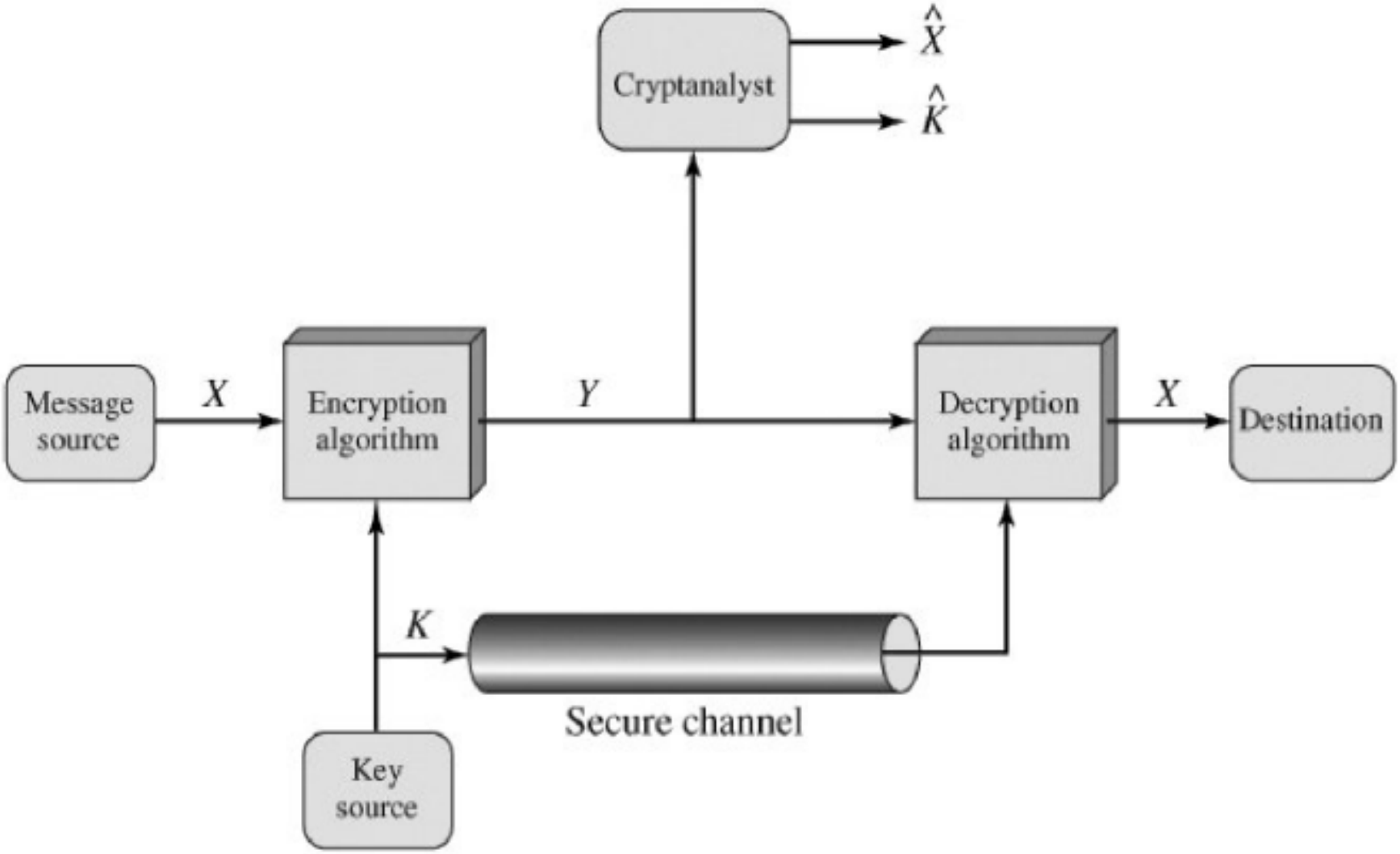
Ingredients of Conventional Cryptography

1. **Plaintext:** Original intelligible message
2. **Encryption algorithm:** Performs substitutions and/or transformations
    - Input: plaintext, key
    - Output: ciphertext
3. **Secret Key** (Different keys produce different outputs)
4. **Cipher text:** Unintelligible scrambled message that depends on plaintext and key
5. **Decryption algorithm:** It is the Encryption algorithm run in reverse
    - Input: ciphertext, key
    - Output: plaintext

Simplified Model for Conventional Cryptography



Conventional Cryptography Formal Model



Requirements of Conventional Cryptography

1. Strong encryption algorithm
2. Secret key known only to sender / receiver
3. $Y = E_K(X)$
4. $X = D_K(Y)$
5. Assume encryption algorithm is known (Kerckhoffs' Principle)
6. Implies a secure channel to distribute key

Characterization of Conventional Cryptography

1. **According to the type of operation:**
   A. **Substitution:** each element of plaintext (bit, character) mapped to another element
   B. **Transposition:** plaintext elements rearranged
2. **According to the processing method:**
   A. **Stream cipher:** element by element (bit, byte)
   B. **Block cipher:** block (of bytes) transformed as a whole

Historical Notes (Not required in Exam)

Two absolutely fascinating books:

1. The Codebreakers, by David Kahn, 1996, Scribner.

2. The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography, by Simon Singh, 1999, Anchor Books

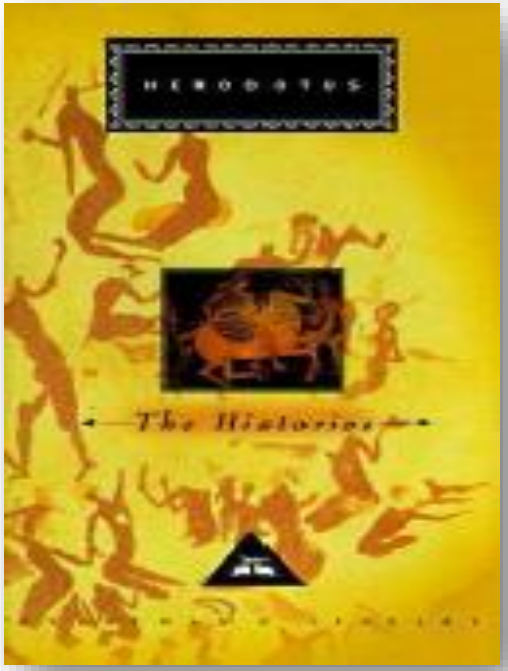
Ancient Mesopotamia

The oldest Mesopotamian encipherment:

- A 3" x 2" cuneiform tablet, dating from ~1500 B.C.
- Earliest known formula for pottery glazes.
- Uses cuneiform signs in their least common syllabic values to attempt to hide the secrets of the formulae

Ancient Greece

Herodotus, in The Histories, chronicled the conflicts between Greece and Persia in the 5th century B.C. Herodotus also described another incident:



- Histaiaeus wanted to encourage Aristagoras of Miletus to revolt against the Persian king.
- To send the message securely, he shaved the head of his messenger, wrote on his scalp, and then waited for the hair to regrow.

The Evolution of Secret Writing

- Strategy was: Hiding the message (**steganography**)
- Cryptography: Hiding the meaning (encryption).
- Cryptography was developed in parallel with steganography. It had the obvious advantage that without knowing the scrambling protocol, the enemy could not easily determine the message.
- Two branches of cryptography: Transposition & Substitution.
- The Caesar (shift) cipher is based on a cipher alphabet that is shifted a certain number of places (in Caesar's case three) relative to the plain alphabet.

Muslim Cryptanalysts (1)

- The year 750 A.D. heralded the golden age of Islamic civilization.
- The social order relied on an effective system of administration, which in turn relied on secure communication achieved through the use of encryption.
- The Arab scholars invented cryptanalysis, the science of unscrambling a message without knowledge of the key. They cracked the monoalphabetic substitution cipher after several centuries of its successful use.
- Starting around the 8th century A.D. Al-Khalil ibn Ahmad al Farahidi:
  1. Solved a cryptogram in Greek for the Byzantine emperor
  2. Was the first to discover and write down the methods of cryptanalysis

Muslim Cryptanalysts (2)

- The innocuous observation that some letters are more common than others in written documents would lead to the first great breakthrough in cryptanalysis. The method, called **frequency analysis** is described in a treatise by Abu Yusuf Ya'qub al-Kindi in the ninth century.
- Greatest treatise, rediscovered in 1987 in the Sulaimaniyyah Ottoman Archive in Instanbul, is entitled: "A Manuscript on Deciphering Cryptographic Messages". It describes a revolutionary system of cryptanalysis which is still in use today.
- In 1412 A.D., the Arabic knowledge of cryptology was fully described in the "Subh al-a'sha" a huge 14- volume encyclopedia, written by Shihab al-Din abu'l-Abbas Ahmad al-Qalqashandi

Renaissance in the West

- 13th Century—Cryptography was introduced into Western Civilization by European monks: Epistle on the Secret Works of Art and the Nullity of Magic Roger Bacon (English Franciscan monk)
- 15th Century—The revival in the arts, sciences and scholarship during the Renaissance nurtured the capacity for cryptography, while an explosion in political machinations offered ample motivation for secret communication. Each state had a cipher office, and each ambassador had a cipher secretary.
- This was a period of transition, with cryptographers still relying on the monoalphabetic substitution cipher, while cryptanalysts were beginning to use frequency analysis to break it.

Finally . . .

- **Acknowledgment:** These lecture notes are based on the textbook by William Stallings and notes prepared by Avinash Kak, Purdue University. My sincere thanks are devoted to them and to all other people who offered the material on the web.

- Students are advised to study and solve the problems and answer the questions in **Assignment-1**.

Chapter 2:
Classical Encryption Techniques I

4th Year- Course, CCSIT, UoA

Lecture Goals

1. To introduce the rudiments of encryption vocabulary.
2. To trace the history of some early approaches to cryptography.
3. To emphasize the basic principles of *substitution* and *transposition*, which are still valid till now (despite of the change of technology).

Vocabulary of Classical Encryption (1)

- **plaintext:** This is the original message that we want to encrypt
- **ciphertext**: The encrypted (message) output
- **enciphering or encryption**: The process by which plaintext is converted into ciphertext
- **encryption algorithm**: The sequence of data processing steps that go into transforming plaintext into ciphertext.
  - Various parameters used by an encryption algorithm are derived from a *secret key*.
  - In classical cryptography for commercial and other civilian applications, the encryption algorithm is made *public*.

Vocabulary of Classical Encryption (2)

- **secret key**: A secret key is used to set some or all of the various parameters used by the encryption algorithm. The important thing to note is that the same secret key is used for encryption and decryption in classical cryptography.
- **deciphering or decryption**: Recovering plaintext from ciphertext
- **decryption algorithm**: The sequence of data processing steps that go into transforming ciphertext back into plaintext.
  - Various parameters used by a decryption algorithm are derived from the same secret key that was used in the encryption algorithm.
  - In classical cryptography for commercial and other civilian applications, the decryption algorithm is made public.

Vocabulary of Classical Encryption (3)

- **cryptography**: The many schemes available today for encryption and decryption
- **cryptographic system**: Any single scheme for encryption
- **cipher:** A cipher means the same thing as a "cryptographic system"
- **block cipher**: A block cipher processes a block of input data at a time and produces a ciphertext block of the same size.
- **stream cipher**: A stream cipher encrypts data on the fly, usually one byte (or bit) at time.

Vocabulary of Classical Encryption (4)

- **cryptanalysis**: Means "breaking the code". Cryptanalysis relies on a knowledge of the encryption algorithm and some knowledge of the possible structure of the plaintext (such as the structure of a typical inter-bank financial transaction) for a partial or full reconstruction of the plaintext from ciphertext. Additionally, the goal is to also infer the key for decryption of future messages. The precise methods used for cryptanalysis depend on whether:

    1. the "attacker" has just a piece of ciphertext,
    2. or pairs of plaintext and ciphertext,
    3. how much structure is possessed by the plaintext,
    4. and how much of that structure is known to the attacker.

Vocabulary of Classical Encryption (5)

- **brute-force attack**: When encryption and decryption algorithms are publicly available, a brute-force attack means trying every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained.
- **key space**: The total number of all possible keys that can be used in a cryptographic system. For example, DES uses a 56-bit key. So the key space is of size 2^{56}, which is approximately the same as 7.2×10^{16}.
- **cryptology**: Cryptography and cryptanalysis together constitute the area of cryptology

Building Blocks of Classical Encryption Techniques

Two building blocks of all classical encryption techniques are substitution and transposition.

1. **Substitution** means replacing an element of the plaintext with an element of ciphertext.
2. **Transposition** means rearranging the order of appearance of the elements of the plaintext. Transposition is also referred to as *permutation*.

Caesar Cipher (1)

- This is the earliest known example of a substitution cipher.
- Each character of a message is replaced by a character ***three*** position down in the alphabet.

plaintext: a r e y o u r e a d y

ciphertext: DUH BRX UHDGB

Note that the alphabet is wrapped around, so that the letter following Z is A. We can define the transformation by listing all possibilities, as follows:

plain: a b c d e f g h i j k l m n o p q r s t u v w x y z

cipher: d e f g h i j k l m n o p q r s T u v w x y z a b c

Caesar Cipher (2)

- Let us assign a numerical equivalent to each letter:

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12

n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

- If we represent each letter of the alphabet by an integer that corresponds to its position in the alphabet, the formula for replacing each character 'p' of the plaintext with a character 'C' of the ciphertext can be expressed as

$$C = E(3, p) = (p + 3) \bmod 26$$

Caesar Cipher (3)

- A more general version of this cipher that allows for any degree of shift would be expressed by

$$C = E(k, p) = (p + k) \bmod 26$$

- The formula for decryption would be

$$p = D(k, C) = (C - k) \bmod 26$$

- In these formulas, *'k'* would be the secret key [1,…, 25]. The symbols *'E'* and *'D'* represent encryption and decryption.

❖ *We define* ***a mod n*** *to be the remainder when* ***a*** *is divided by* ***n****. For example, 11 mod 7 = 4.*

Caesar Cipher (4)

- If it is known that a given ciphertext is a Caesar cipher, then a brute-force cryptanalysis is easily performed: simply try all the **25** possible keys.
- Three important characteristics of this problem enabled us to use a **brute-force** cryptanalysis:

1. The encryption and decryption algorithms are known.
2. There are only 25 keys to try.
3. The language of the plaintext is known and easily recognizable.

Monoalphabetic Ciphers (1)

- With only 25 possible keys, the Caesar cipher is far from secure. A dramatic *increase in the key space* can be achieved by allowing an arbitrary substitution.
- *A **permutation*** of a finite set of elements S is an ordered sequence of all the elements of S , with each element appearing exactly once.
- For example, if S = {a, b, c}, there are six permutations of S: abc, acb, bac, bca, cab, cba
- In general, there are ***n!*** permutations of a set of ***n*** elements, because the first element can be chosen in one of n ways, the second in n - 1 ways, the third in n – 2 ways, and so on.

Monoalphabetic Ciphers (2)

- In a monoalphabetic cipher, our substitution characters are a random permutation of the 26 letters of the alphabet.
- The *key now is the sequence of substitution letters*. In other words, the key in this case is the actual random permutation of the alphabet used.
- Note that there are 26! permutations of the alphabet. That is a number larger than 4×10^{26}.
- Such an approach is referred to as a ***monoalphabetic*** substitution cipher, because a single cipher alphabet (mapping from plain alphabet to cipher alphabet) is used per message

A Very Large Key Space But ...

- That gives us a ***huge key space*** (meaning the total number of all possible keys that would need to be guessed in a brute-force attack). This key space is 10 orders of magnitude larger than the size of the key space for DES, the now somewhat outdated (but still widely used) NIST standard.
- Obviously, this would rule out a brute-force attack. (Even if each key took only a nanosecond to try, it would still take zillions of years to try out even half the keys.)
- However, this system is ***still weak!*** Why? Read on.

The All-Fearsome Statistical Attack

- If you know the nature of plaintext, any substitution cipher, regardless of the size of the key space, can be broken easily with a ***statistical attack***.
- When the plaintext is plain English, a simple form of statistical attack consists measuring the frequency distribution for single characters, for pairs of characters, for triples of characters, etc., and comparing those with similar statistics for English.
- **Figure 1** shows the relative frequency of the letters in a sample of English text. Obviously, by ***comparing*** this distribution with a histogram for the characters in a piece of ciphertext, you may be able to establish the true identities of the ciphertext characters.

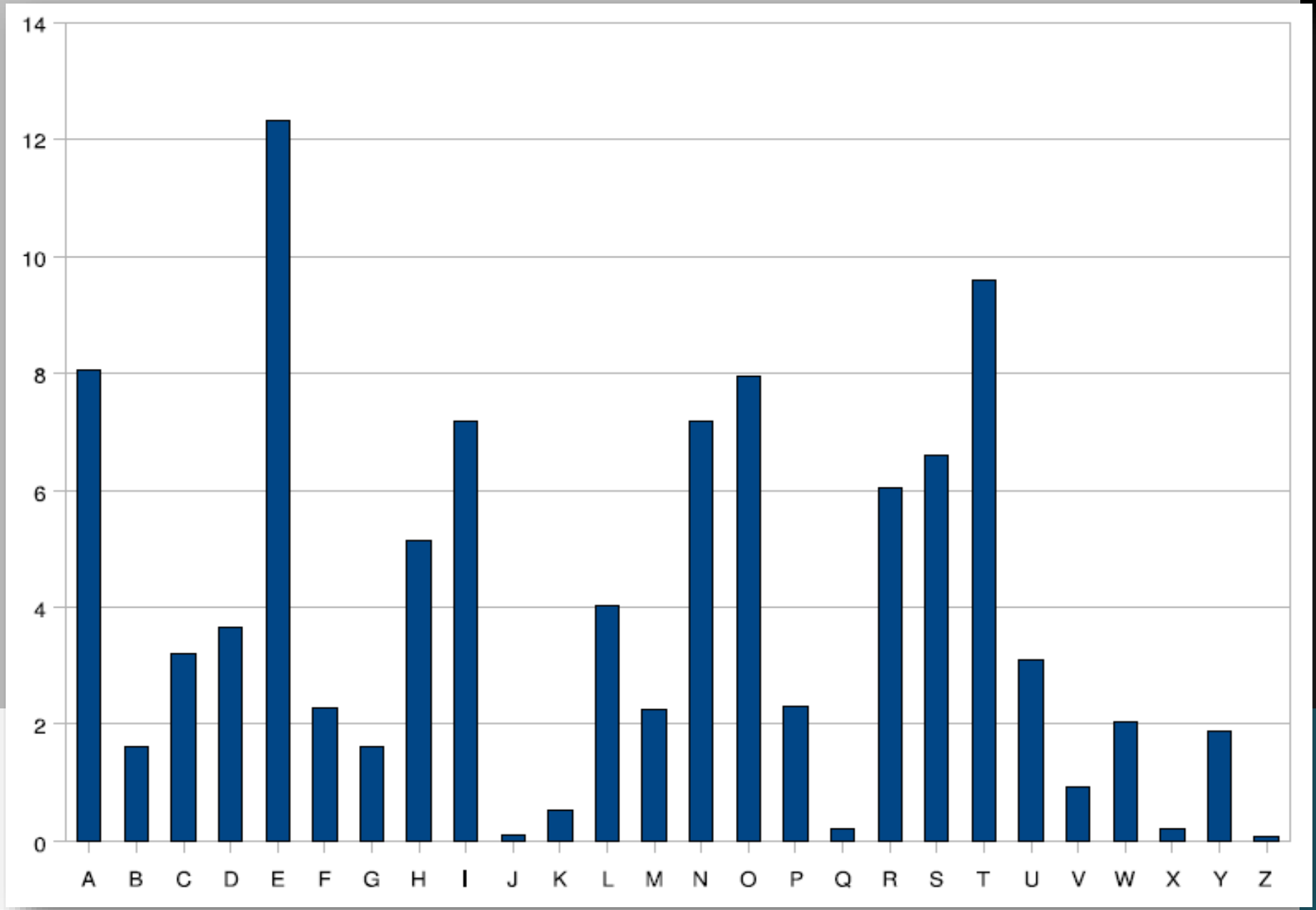


Figure 1.

Statistics for Digrams and Trigrams

- Equally powerful statistical inferences can be made by comparing the relative frequencies for pairs and triples of characters in the ciphertext and the language believed to be used for the plaintext.
- Pairs of adjacent characters are referred to as ***digrams***, and triples of characters as ***trigrams***.
- Shown in Table 1 are some of the digram frequencies.
- The most frequently occurring trigrams ordered by decreasing frequency are:

the and ent ion tio for nde ...

digram	*frequency*	*digram*	*frequency*	*digram*	*frequency*	*digram*	*frequency*
th	3.15	to	1.11	sa	0.75	ma	0.56
he	2.51	nt	1.10	hi	0.72	ta	0.56
an	1.72	ed	1.07	le	0.72	ce	0.55
in	1.69	is	1.06	so	0.71	ic	0.55
er	1.54	ar	1.01	as	0.67	ll	0.55
re	1.48	ou	0.96	no	0.65	na	0.54
es	1.45	te	0.94	ne	0.64	ro	0.54
on	1.45	of	0.94	ec	0.64	ot	0.53
ea	1.31	it	0.88	io	0.63	tt	0.53
ti	1.28	ha	0.84	rt	0.63	ve	0.53
at	1.24	se	0.84	co	0.59	ns	0.51
st	1.21	et	0.80	be	0.58	ur	0.49
en	1.20	al	0.77	di	0.57	me	0.48
nd	1.18	ri	0.77	li	0.57	wh	0.48
or	1.13	ng	0.75	ra	0.57	ly	0.47

Table 1.

How to Mask Plaintext Structure?

- One character at a time substitution obviously leaves too much of the plaintext structure in ciphertext.
- Two principal methods are used in substitution ciphers to lessen the extent to which the structure of the plaintext survives in the ciphertext:
  1. One approach is to encrypt (substitute) multiple letters of plaintext
  2. The other is to use multiple cipher alphabets.
- The best known approach that carries out multiple-character substitution is known as ***Playfair cipher***.

Playfair Cipher

- In Playfair cipher, you first choose an encryption key.
- You then enter the letters of the key in the cells of a 5 × 5 matrix in a left to right fashion starting with the first cell at the top-left corner.
- You fill the rest of the cells of the matrix with the remaining letters in alphabetic order.
- The letters I and J are assigned the same cell.
- In the following example, the key is "smythework":

S	M	Y	T	H
E	W	O	R	K
A	*B*	*C*	*D*	*F*
G	*I/J*	*L*	*N*	*P*
Q	*U*	*V*	*X*	*Z*

Substitution Rules in Playfair Cipher (1)

1. Two plaintext letters that fall in the ***same row*** of the 5 × 5 matrix are replaced by letters to the ***right*** of each in the row.

   - The *"rightness"* property is to be interpreted circularly in each row, meaning that the first entry in each row is to the right of the last entry. Therefore, the pair of letters "bf" in plaintext will get replaced by "CA" in ciphertext.

2. Two plaintext letters that fall in the ***same column*** are replaced by the letters just ***below*** them in the column.

   - The *"belowness"* property is to be considered circular, in the sense that the topmost entry in a column is below the bottom-most entry. Therefore, the pair "ol" of plaintext will get replaced by "CV" in ciphertext.

Substitution Rules in Playfair Cipher (2)

3. Otherwise, for each plaintext letter in a pair, replace it with the letter that is ***in the same row but in the column of the other letter***.

- Consider the pair "gf" of the plaintext. We have 'g' in the fourth row and the first column; and 'f' in the third row and the fifth column.
- So we replace 'g‘ by the letter in the same row as 'g' but in the column that contains 'f' . This given us 'P' as a replacement for 'g'. And we replace 'f' by the letter in the same row as 'f' but in the column that contains 'g'. That gives us 'A' as replacement for 'f'.
- Therefore, 'gf' gets replaced by 'PA‘.

Dealing With Duplicates Letters in Key and Repeating Letters in Plaintext

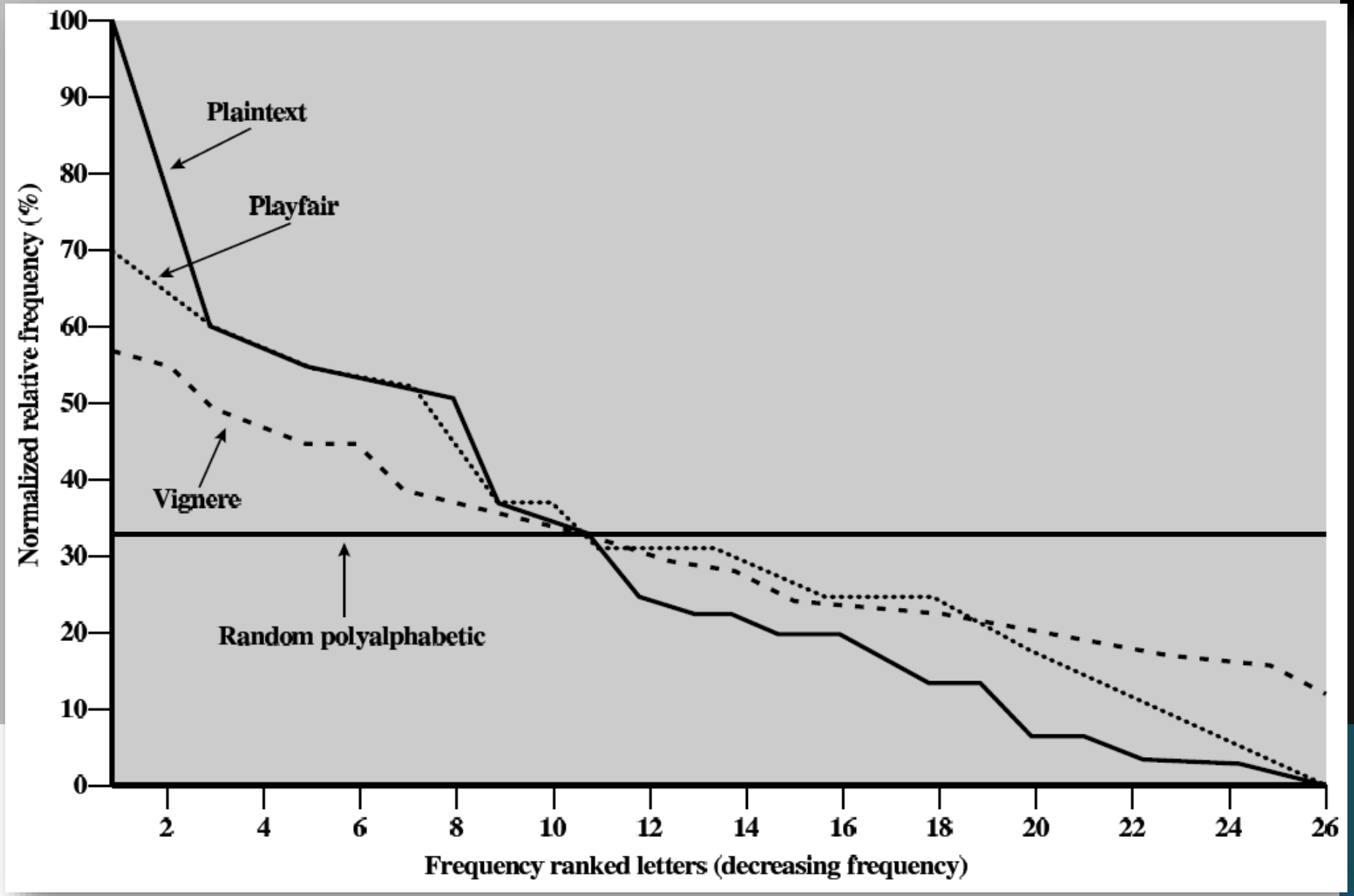
1. You must drop any duplicates in a key.
2. Before the substitution rules are applied, you must insert a chosen "filler" letter (let's say it is 'x') between any repeating letters in the plaintext. So a plaintext word such as "hurray" becomes "hurxray“
3. In case you have gotten a single final letter, append an ‘x’ with it and use the substitution rules normally.

How Secure is the Playfair Cipher? (1)

- Playfair was thought to be unbreakable for many decades.
- It was used as the encryption system by the British Army in World War 1. It was also used by the U.S. Army and other Allied forces in World War 2.
- But, as it turned out, Playfair was extremely easy to break.
- As expected, the cipher does alter the relative frequencies associated with the individual letters and with digrams and with trigrams, but not sufficiently.

How Secure is the Playfair Cipher? (2)

- The figure on the next page shows the single-letter relative frequencies in descending order (and normalized to the relative frequency of the letter 'e') for different ciphers. There is still considerable information left in the distribution for good guesses.
- The cryptanalysis of the Playfair cipher is also aided by the fact that a digram and its reverse will encrypt in a similar fashion. That is, if AB encrypts to XY, then BA will encrypt to YX. So by looking for words that begin and end in reversed digrams, one can try to compare then with plaintext words that are similar. Example of words that begin and end in reversed digrams: receiver, departed, repairer, redder, denuded, etc.



RELATIVE FREQUENCY OF OCCURRENCE OF LETTERS

Finally . . .

- **Acknowledgment:** These lecture notes are based on the textbook by William Stallings and notes prepared by Avinash Kak, Purdue University. My sincere thanks are devoted to them and to all other people who offered the material on the web.

- Students are advised to study and solve the problems and answer the questions in **Assignment-2**.

Chapter 3:
Classical Encryption Techniques II

4th Year- Course, CCSIT, UoA

Lecture Goals

1. To trace the history of some early approaches to cryptography.
2. To emphasize the basic principles of *substitution* and *transposition*, which are still valid till now (despite of the change of technology).

The Hill Cipher (1)

The Hill cipher is another multi-letter cipher that takes a very different (more mathematical) approach to multi-letter substitution:

- You assign an integer to each letter of the alphabet. For the sake of discussion, let's say that you have assigned the integers 0 through 25 to the letters 'a' through 'z' of the plaintext.
- The encryption key, call it K, consists of a 3 ×3 matrix of integers:

$$\mathbf{K} = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}$$

The Hill Cipher (2)

- Now we can transform ***three letters*** at a time from plaintext, the letters being represented by the numbers p_1, p_2, and p_3, into three ciphertext letters c_1, c_2, and c_3 in their numerical representations by

$$
\begin{aligned}
c_1 &= (k_{11}p_1 + k_{12}p_2 + k_{13}p_3) \; mod \; 26 \\
c_2 &= (k_{21}p_1 + k_{22}p_2 + k_{23}p_3) \; mod \; 26 \\
c_3 &= (k_{31}p_1 + k_{32}p_2 + k_{33}p_3) \; mod \; 26
\end{aligned}
$$

- The above set of linear equations can be written more compactly in the following vector-matrix form:

$$
\vec{\mathbf{C}} = [\mathbf{K}] \, \vec{\mathbf{P}} \; mod \; 26
$$

The Hill Cipher (3)

- Obviously, the decryption would require the ***inverse*** of K matrix.

$$\vec{\mathbf{P}} = \left[\mathbf{K}^{-1}\right] \vec{\mathbf{C}} \; mod \; 26$$

This works because

$$\vec{\mathbf{P}} = \left[\mathbf{K}^{-1}\right] [\mathbf{K}] \vec{\mathbf{P}} \; mod \; 26 = \vec{\mathbf{P}}$$

- This can be illustrated in the following example.

Example 1

- Consider the plaintext "***paymoremoney***" and use the encryption Key

$$\mathbf{K} = \begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix}$$

- The first three letters of the plaintext (p a y) are represented by the (row) vector (15 0 24).
- Then, (15 0 24) **K** = (303 303 531) *mod 26*
  = (17 17 11) = R R L.
- Continuing in this fashion, the ciphertext for the entire plaintext is: *RRLMWBKASPDH* .

Example 1 (cont.)

- Decryption requires using the inverse of the matrix K which is:

$$\mathbf{K}^{-1} = \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix}$$

- Note that:

$$\begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix} \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix} = \begin{pmatrix} 443 & 442 & 442 \\ 858 & 495 & 780 \\ 494 & 52 & 365 \end{pmatrix} \bmod 26 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- It is easily seen that if the matrix $\mathbf{K}^{-1}$ is applied to the ciphertext, then the plaintext is recovered.

- **H.W.:** Do the complete encryption and decryption process for this example.

Example 2

Encrypt the word "design" using Hill cipher key $\begin{bmatrix} 9 & 4 \\ 5 & 7 \end{bmatrix}$.

- Since the matrix is 2*2, we encrypt the message in pairs. The pairs "de", "si", and "gn" are represented numerically as (column) vectors

- See the accompanied pdf file for a detailed solution of this example

How Secure is the Hill Cipher?

- As with Playfair, the strength of the Hill cipher is that it completely hides single-letter frequencies.
- It is extremely secure against ***ciphertext only attacks***. That is because the key space can be made extremely large by choosing the matrix elements from a large set of integers.
- The key space can be made even large by generalizing the technique to larger-sized matrices.
- But it has zero security when the plaintext-ciphertext pairs are known. The key matrix can be calculated easily from a set of known P, C pairs.

Polyalphabetic Ciphers

- Another way to improve on the simple monoalphabetic technique is to use different monoalphabetic substitutions as one proceeds through the plaintext message. The general name for this approach is ***polyalphabetic*** substitution cipher.
- All these techniques have the following features in common:
  1. A set of related monoalphabetic substitution rules is used.
  2. A key determines which particular rule is chosen for a given transformation.
- The best known, and one of the simplest, polyalphabetic ciphers is the ***Vigenère cipher***.

The Vigenere Cipher

- Let each letter of the encryption key denote a ***shifted Caesar cipher***, the shift corresponding to the key.
- Since, in general, the encryption key will be shorter than the message to be encrypted, for the Vigenere cipher the key is repeated as required. For example, the key here is the string "***abracadabra***".
- Now a plaintext message may be encrypted as follows

key:	a b r a c a d a b r a a b r a c a d a b r a
plaintext:	c a n y o u m e e t m e a t m i d n i g h t
ciphertext:	C B E Y Q U P E F K M E B K . . . . .

The Vigenere Cipher: Example

If the keyword is "***deceptive***", the message "*we are discovered save yourself*" is encrypted as:

key: d ec e p t i v e de c e p t i v e de c e p t i ve
plaintext: wea r e d i s c ov e r e d s a ve y o u r s e l f
ciphertext: ZICVTWQNGRZGVTWAVZHCQYGLMGJ

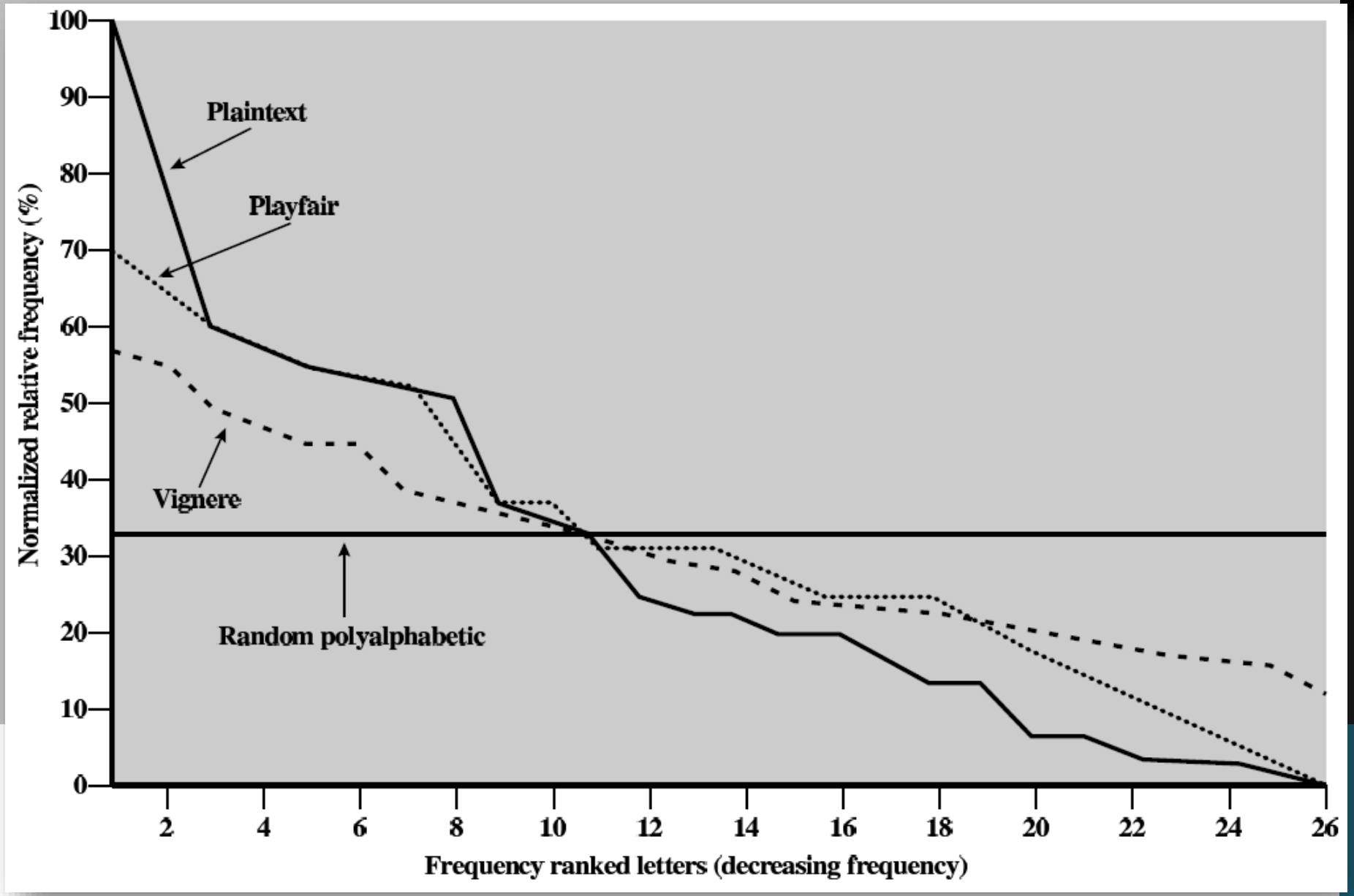
Expressed numerically, we have the following result.

key	3	4	2	4	15	19	8	21	4	3	4	2	4	15
plaintext	22	4	0	17	4	3	8	18	2	14	21	4	17	4
ciphertext	25	8	2	21	19	22	16	13	6	17	25	6	21	19

key	19	8	21	4	3	4	2	4	15	19	8	21	4
plaintext	3	18	0	21	4	24	14	20	17	18	4	11	5
ciphertext	22	0	21	25	7	2	16	24	6	11	12	6	9

How Secure is the Vigenere Cipher? (1)

- Since there exist in the output multiple ciphertext letters for each plaintext letter, you would expect that the relative frequency distribution would be effectively destroyed. But as can be seen in the plots on next page, a great deal of the input statistical distribution still shows up in the output.
  - The plot shown for Vigenere cipher is for an encryption key that is 9 letters long.
- Obviously, the longer the encryption key, the greater the masking of the structure of the plaintext.
- The best possible key is as long as the plaintext message and consists of a purely random permutation of the 26 letters of the alphabet. This would yield the ideal plot shown in the figure. The ideal plot is labeled "Random polyalphabetic" in the figure.



RELATIVE FREQUENCY OF OCCURRENCE OF LETTERS

How Secure is the Vigenere Cipher? (2)

- In general, to break the Vigenere cipher, you first try to estimate the ***length of the encryption key***. This length can be estimated by using the logic that plaintext words separated by multiples of the length of the key will get encoded in the same way.
- If the estimated length of the key is ***N***, then the cipher consists of ***N*** *monoalphabetic substitution* ciphers and the plaintext letters at positions 1, ***N***, 2***N***, 3***N***, etc., will be encoded by the same monoalphabetic cipher.
- This insight can be useful in the decoding of the monoalphabetic ciphers involved.

Vernam Cipher (1)

- The ultimate defense against cryptanalysis is to choose a keyword that is as long as the plaintext and has no statistical relationship to it. Such a system was introduced by Gilbert Vernam in 1918. His system works on binary data (bits) rather than letters. The system can be expressed as follows.

$$c_i = p_i \oplus k_i$$

where

p_i = ith binary digit of plaintext

k_i = ith binary digit of key

c_i = ith binary digit of ciphertext

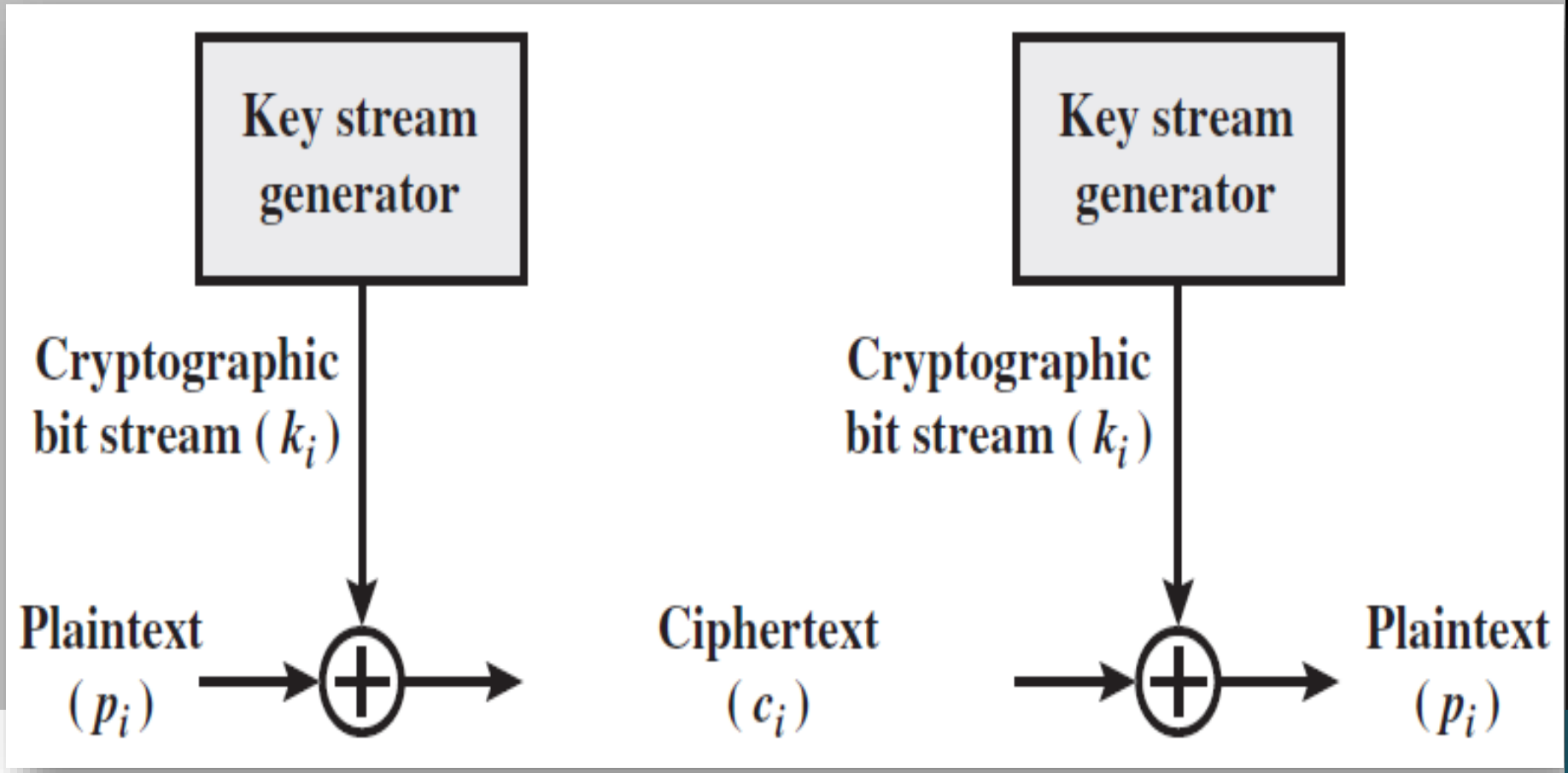
$\oplus$ = exclusive-or (XOR) operation

Vernam Cipher (2)

- Thus, the ciphertext is generated by performing the bitwise XOR of the plaintext and the key. Because of the properties of the XOR, decryption simply involves the same bitwise operation:

$$p_i = c_i \oplus k_i$$

- The essence of this technique is the means of ***construction of the key***.
  - Vernam proposed the use of a running loop of tape that eventually repeated the key, so that in fact the system worked with a very long but repeating keyword.
  - Although such a scheme, with a long key, presents formidable cryptanalytic difficulties, it can be broken with sufficient ciphertext, the use of known or probable plaintext sequences, or both.



Vernam Cipher Encryption and Decryption

One-Time Pad (1)

- Joseph Mauborgne proposed an improvement to Vernam cipher that yields the ultimate in security. He suggested:
  1. using of a random key
  2. key that is truly as long as the message
  3. and with no repetitions.
- Such a scheme, known as ***one-time pad***, is ***unbreakable***.
- It produces random output that bears no statistical relationship to the plaintext. Because the ciphertext contains no information whatsoever about the plaintext, there is no way to break the code.

One-Time Pad (2)

- In theory, we need look no further for a cipher. The one-time pad offers complete security but, in practice, has ***two fundamental difficulties*** that make the one-time pad is of limited utility. These are:

  1. There is the practical problem of making large quantities of random keys.
  2. Even more daunting is the problem of key distribution and protection. For every message to be sent, a key of equal length is needed by both sender and receiver.

Transposition Techniques

- All of our discussion so far has dealt with substitution ciphers. We have talked about monoalphabetic substitutions, polyalphabetic substitutions, etc.
- We will now talk about a different notion in classical cryptography: ***permuting the plaintext***.
- This is how a pure permutation cipher could work:
  1. You write your plaintext message along the ***rows*** of a matrix of some size.
  2. You generate ciphertext by reading along the ***columns***.
  3. The *order in which you read* the columns is determined by the ***encryption Key***.
- The cipher can be made more secure by performing ***multiple rounds of such permutations.***

Columnar Transposition Cipher Example

Encrypt the message: "*meet me at midnight for the godies*" with the following key:

```
key:                4 1 3 6 2 5

plaintext:          m e e t m e
                    a t m i d n
                    i g h t f o
                    r t h e g o
                    d i e s x y

ciphertext:         ETGTIMDFGXEMHHEMAIRDENOOYTITES
```

Steganography

- A plaintext message may be hidden in one of two ways.
  1. The methods of ***steganography*** conceal the existence of the message,
  2. whereas the methods of ***cryptography*** render the message unintelligible to outsiders by various transformations of the text.
- A simple form of steganography is one in which an arrangement of words or letters within an apparently innocuous text spells out the real message. For example, the sequence of first letters of each word of the overall message spells out the hidden message.
- As a contemporary example consider hiding a message by using the least significant bits of frames on a CD. The least significant bit of each 24-bit pixel can be changed without greatly affecting the quality of the image.

Finally . . .

- **Acknowledgment:** These lecture notes are based on the textbook by William Stallings and notes prepared by Avinash Kak, Purdue University. My sincere thanks are devoted to them and to all other people who offered the material on the web.

- Students are advised to study and solve the problems and answer the questions in **Assignment-3**.

Chapter 4:

Block Ciphers and DES

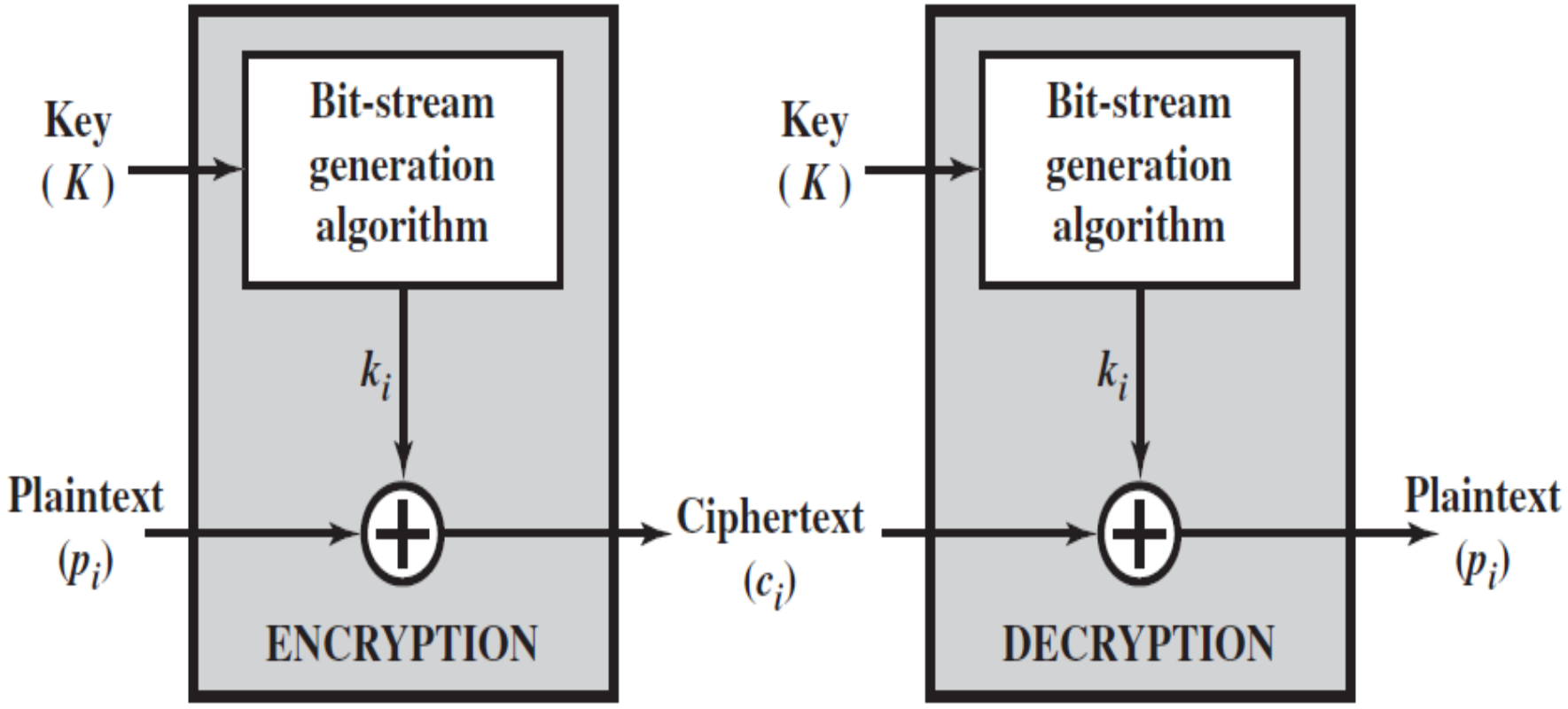
4th Year- Course, CCSIT, UoA

Lecture Goals

1. To introduce the notion of a block cipher in the modern context.
2. To introduce the notion of the Feistel Cipher Structure
3. To go over DES, the Data Encryption Standard

Stream Ciphers

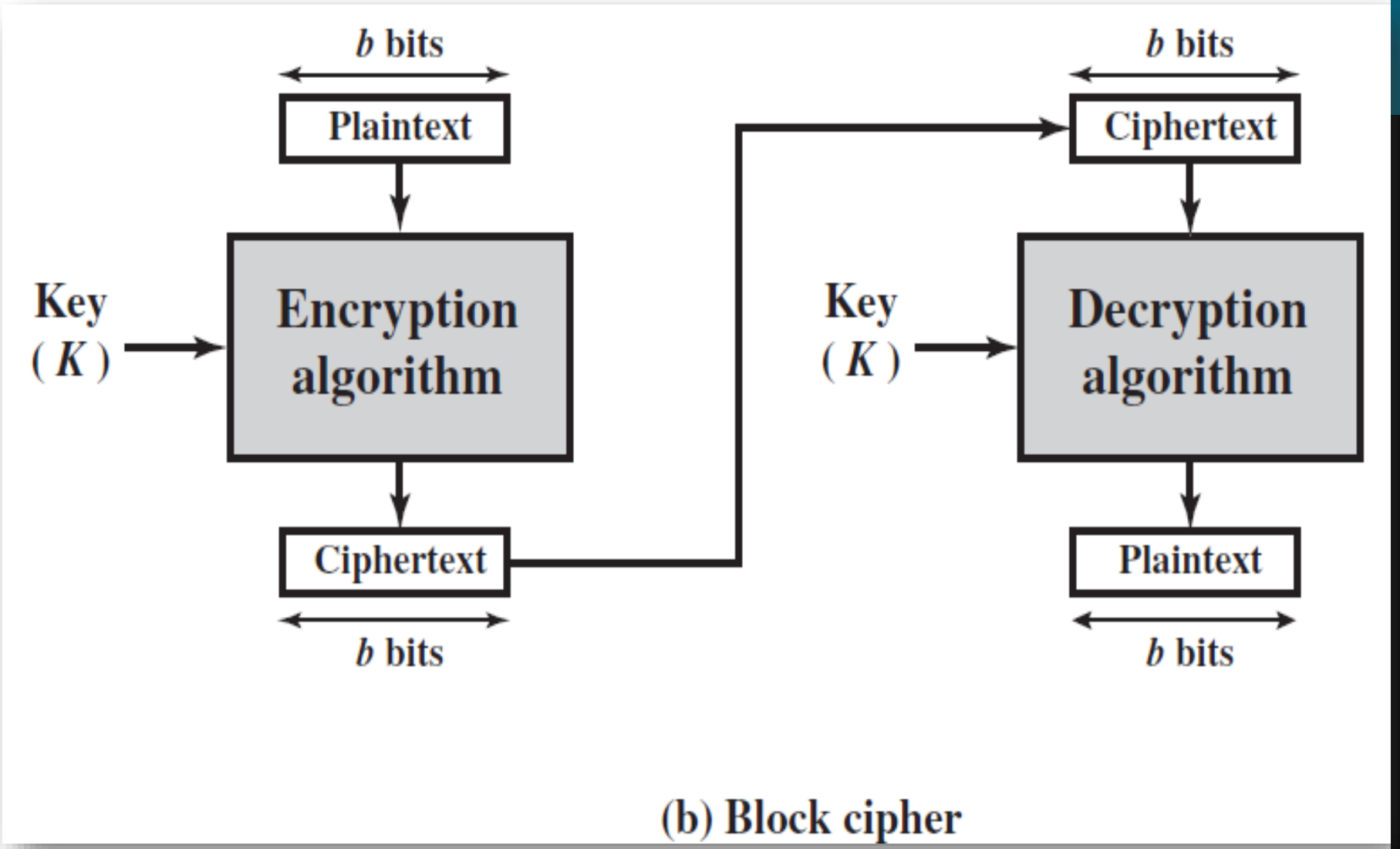
- A ***stream cipher*** is one that encrypts a digital data stream one bit or one byte at a time. Examples of classical stream ciphers are the autokeyed Vigenère cipher and the Vernam cipher.
- For practical reasons, the bit-stream generator must be implemented as an algorithmic procedure, so that the cryptographic bit stream can be produced by both users. In this approach (see next Figure), the bit-stream generator is a key-controlled algorithm and must produce a bit stream that is cryptographically strong.
- The two users need only share the generating key, and each can produce the keystream.



(a) Stream cipher using algorithmic bit-stream generator

Block Ciphers

- A ***block cipher*** is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a block size of 64 or 128 bits is used. As with a stream cipher, the two users share a symmetric encryption key (see next Figure).
- Using some of the modes of operation (explained later), a block cipher can be used to achieve the same effect as a stream cipher.
- Far more effort has gone into analyzing block ciphers. In general, they seem applicable to a broader range of applications than stream ciphers.



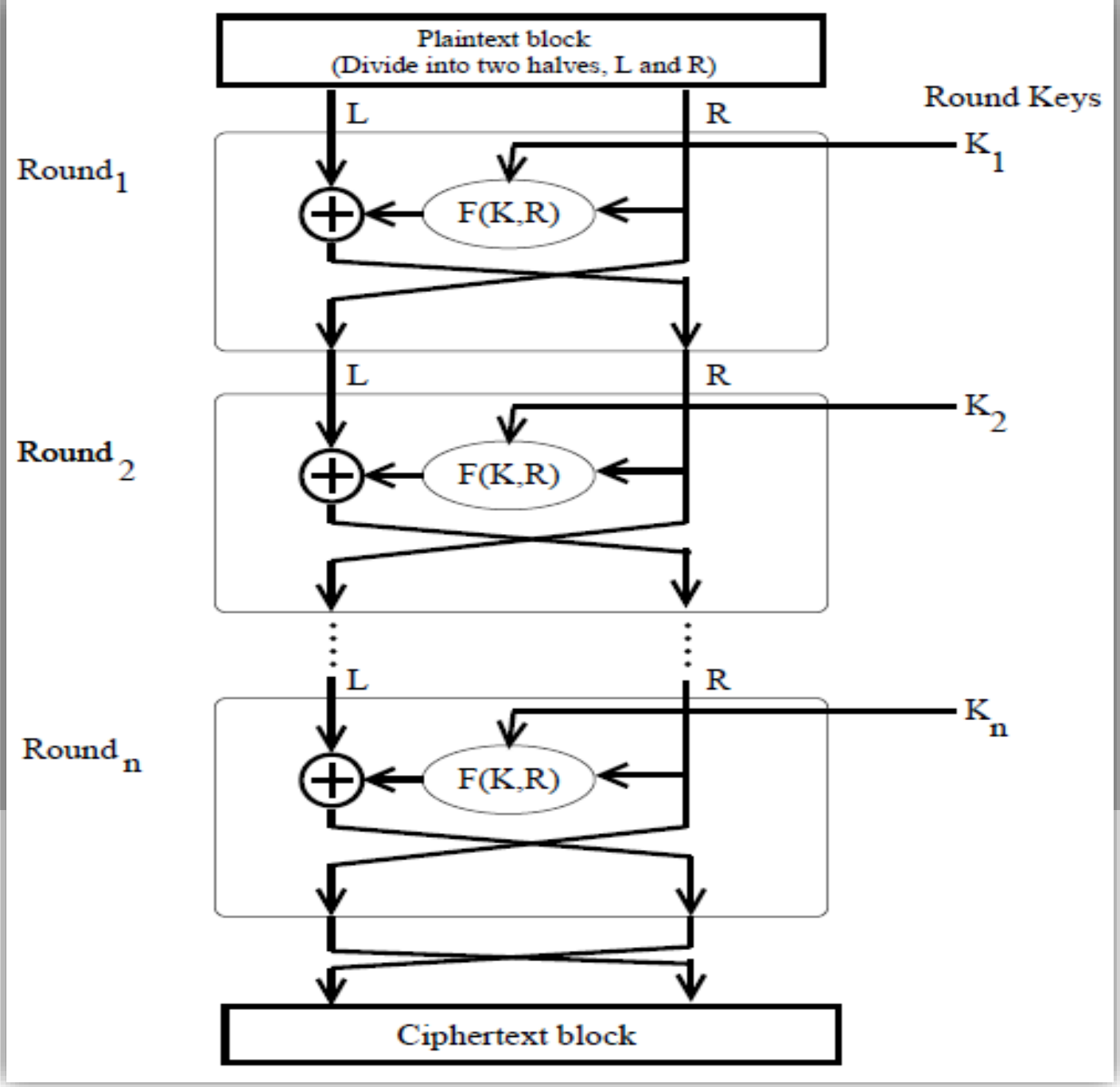
(b) Block cipher

The Feistel Structure for Block Ciphers (1)

- Named after the IBM cryptographer Horst Feistel and first implemented in the Lucifer cipher by Horst Feistel and Don Coppersmith.
- A cryptographic system based on Feistel structure uses the ***same basic algorithm*** for both encryption and decryption.
- As shown in the next Figure, the Feistel structure consists of ***multiple rounds*** of processing of the plaintext, with each round consisting of a ***substitution*** step followed by a ***permutation*** step.

The Feistel Structure for Block Ciphers (2)

- The input block to each round is divided into ***two halves*** that we can denote L and R for the left half and the right half.
- In each round, the right half of the block, R, goes through ***unchanged***. But the left half, L, goes through an ***operation*** that depends on R and the encryption key.
- The permutation step at the end of each round consists of ***swapping*** the modified L and R. Therefore, the L for the next round would be R of the current round. And R for the next round be the output L of the current round.


Plaintext block
(Divide into two halves, L and R)
Round Keys
L
R
Round 1
K 1
F(K,R)
L
R
Round 2
K 2
F(K,R)
L
R
Round n
K n
F(K,R)
Ciphertext block

Mathematical Description of Each Round in the Feistel Structure (1)

- Let LE_i and RE_i denote the output half-blocks at the end of the i^{th} round of processing. The letter 'E' denotes encryption.
- We obviously have

$$LE_i = RE_{i-1}$$
$$RE_i = LE_{i-1} \oplus F(RE_{i-1}, K_i)$$

- where $\oplus$ denotes the bitwise EXCLUSIVE OR operation. The symbol F denotes the operation that "scrambles" RE_{i-1} of the previous round with the current round key K_i.

Mathematical Description of Each Round in the Feistel Structure (2)

- Note that the ***round key*** K_i is derived from the ***main encryption key*** as we will explain later.
- F is referred to as the ***Feistel function***, after Horst Feistel, naturally.
- Assuming 16 rounds of processing (which is typical), the output of the last round of processing is given by

$$LE_{16} = RE_{15}$$
$$RE_{16} = LE_{15} \oplus F(RE_{15}, K_{16})$$

Decryption in Ciphers Based on the Feistel Structure (1)

- As shown in the next Figure, the decryption algorithm is exactly the ***same*** as the encryption algorithm with the ***only difference*** that the round keys are used in the ***reverse order***.
- The output of each round during decryption is the input to the corresponding round during encryption. This property holds true ***regardless of the choice of the Feistel function F*** .
- To prove the above claim, let LD_i and RD_i denote the left half and the right half of the output of the i^{th} round.

Decryption in Ciphers Based on the Feistel Structure (2)

- That means that the output of the first decryption round consists of LD_1 and RD_1. So we can denote the input to the first decryption round by LD_0 and RD_0.
- The relationship between the two halves that are input to the first decryption round and what is output by the encryption algorithm is

$$LD_0 = RE_{16}$$
$$RD_0 = LE_{16}$$

Decryption in Ciphers Based on the Feistel Structure (3)

- We can write the following equations for the output of the first decryption round

$$
\begin{aligned}
LD_1 &= RD_0 \\
&= LE_{16} \\
&= RE_{15} \\
RD_1 &= LD_0 \oplus F(RD_0, K_{16}) \\
&= RE_{16} \oplus F(LE_{16}, K_{16}) \\
&= [LE_{15} \oplus F(RE_{15}, K_{16})] \oplus F(RE_{15}, K_{16}) \\
&= LE_{15}
\end{aligned}
$$

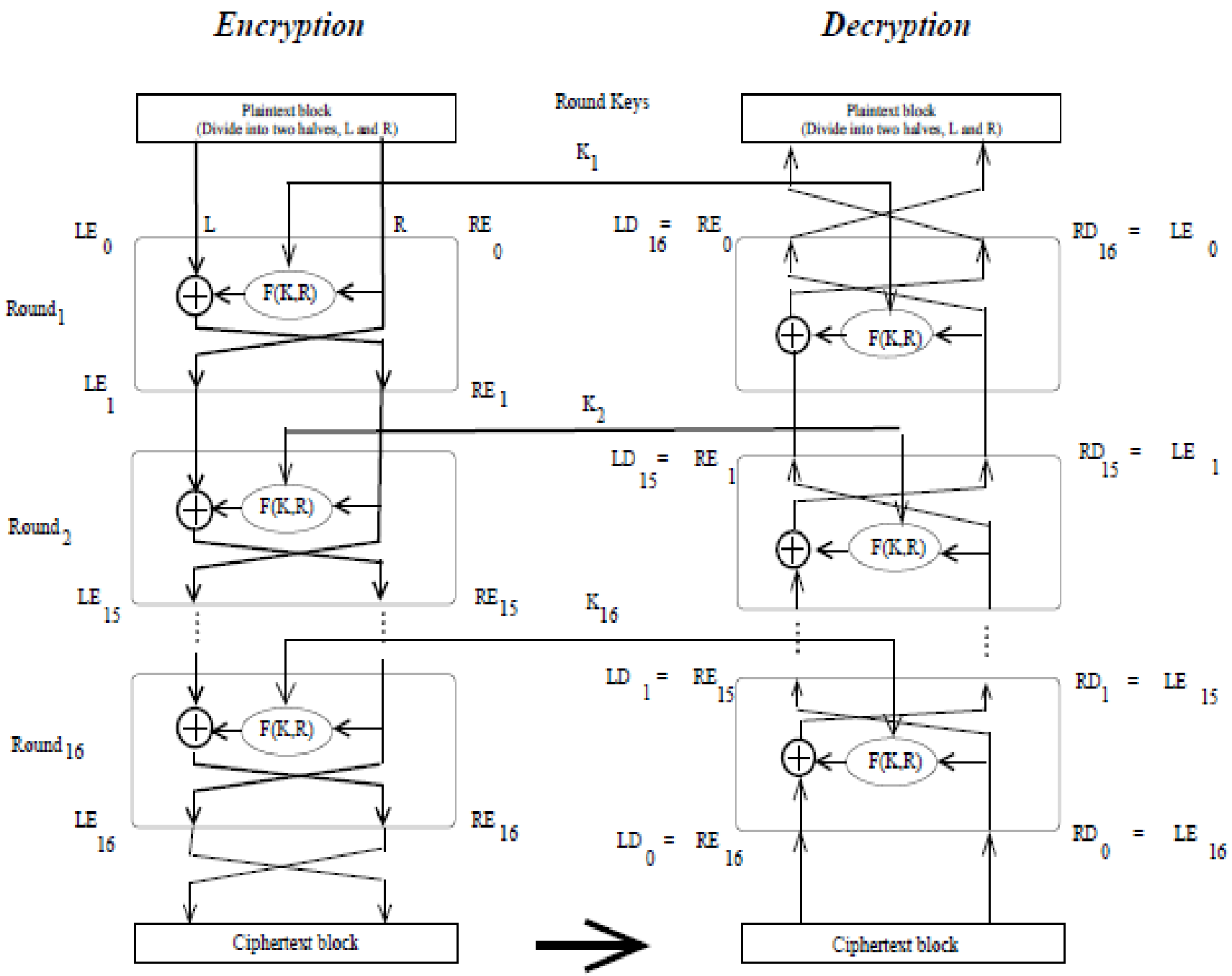
- This shows that the output of the first round of decryption is the same as the input to the last stage of the encryption round since we have $LD_1 = RE_{15}$ and $RD_1 = LE_{15}$

Decryption in Ciphers Based on the Feistel Structure (4)

- The following equalities are used in the above derivation. Assume that A, B, and C are bit arrays.

$$[A \oplus B] \oplus C = A \oplus [B \oplus C]$$
$$A \oplus A = 0$$
$$A \oplus 0 = A$$

- ***The above result is independent of the precise nature of F .***


Encryption
Decryption
Plaintext block
(Divide into two halves, L and R)
Round Keys
K1
LE0
L
R
RE0
LD16 = RE0
RD16 = LE0
Round1
F(K,R)
LE1
RE1
K2
LD15 = RE1
RD15 = LE1
Round2
LE15
RE15
K16
LD1 = RE15
RD1 = LE15
Round16
LE16
RE16
LD0 = RE16
RD0 = LE16
Ciphertext block
Ciphertext block

Feistel Network Parameters

1. ***Block size:*** Larger block sizes mean greater security but reduced encryption/decryption speed for a given algorithm.
2. ***Key size:*** Larger key size means greater security but may decrease encryption/ decryption speed.
3. ***Number of rounds:*** The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security.
4. ***Subkey generation algorithm:*** Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.
5. ***Round function F :*** Again, greater complexity generally means greater resistance to cryptanalysis.

The F Function

1. The heart of a Feistel block cipher is the function F, which provides the element of ***confusion*** (will be explained latter) in a Feistel cipher.
2. Thus, it must be difficult to "unscramble" the substitution performed by F.
3. One obvious criterion is that F be ***nonlinear***. The more nonlinear F, the more difficult any type of cryptanalysis will be.

DES: The Data Encryption Standard (1)

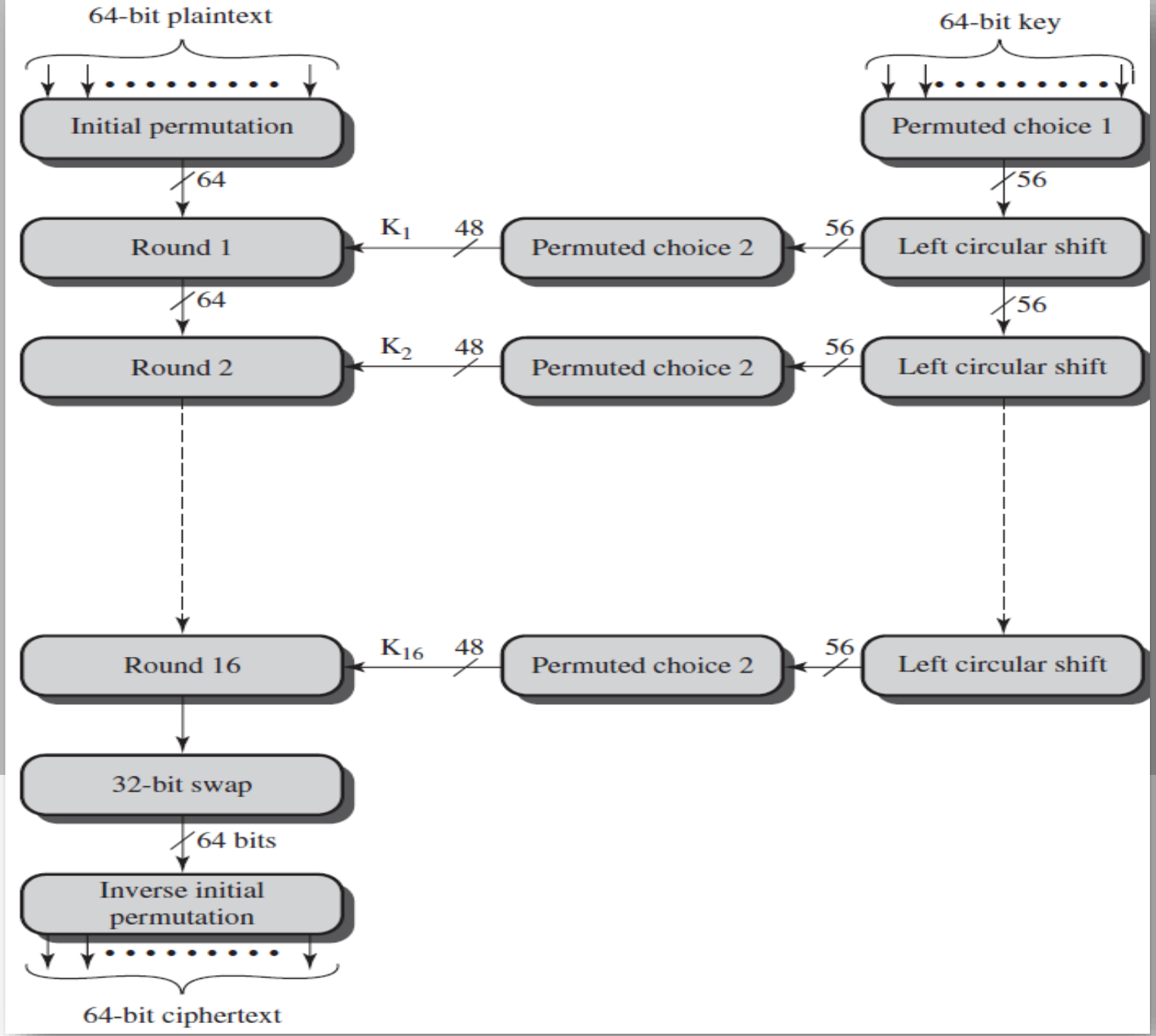
- Adopted by NIST in 1977.
- Based on a cipher (Lucifer) developed earlier by IBM.
- DES uses the Feistel cipher structure with 16 rounds of processing.
- DES uses a 56-bit encryption key. (The key size was apparently dictated by the memory and processing constraints imposed by a single-chip implementation of the algorithm for DES.)
- The key itself is specified with 8 bytes, but one bit of each byte is used as a parity check.

DES (2)

- DES encryption was ***broken*** in 1999 by Electronics Frontiers Organization. This resulted in NIST issuing a new directive that year that required organizations to use ***Triple DES***, that is three consecutive applications of DES.
- That DES was found to be not as strong as originally believed also prompted NIST to initiate the development of new standards for data encryption in 2001. The result is AES (Advanced Encryption Standard).
- Triple DES continues to enjoy wide usage in commercial applications. To understand Triple DES, you must first understand the basic DES encryption.

DES (3)

- As mentioned, DES uses the Feistel structure with 16 rounds.
- What is specific to DES is:
  1. the implementation of ***the F function*** in the algorithm
  2. and how ***the round keys are derived*** from the main encryption key.
- The round keys are generated from the main key by a sequence of permutations. Each round key is 48 bits in length.
- The figure in the next page shows the general description of DES encryption process.


64-bit plaintext
64-bit key
Initial permutation
Permuted choice 1
64
56
Round 1
K_1
48
Permuted choice 2
56
Left circular shift
64
56
Round 2
K_2
48
Permuted choice 2
56
Left circular shift
Round 16
K_{16}
48
Permuted choice 2
56
Left circular shift
32-bit swap
64 bits
Inverse initial permutation
64-bit ciphertext

DES Decryption

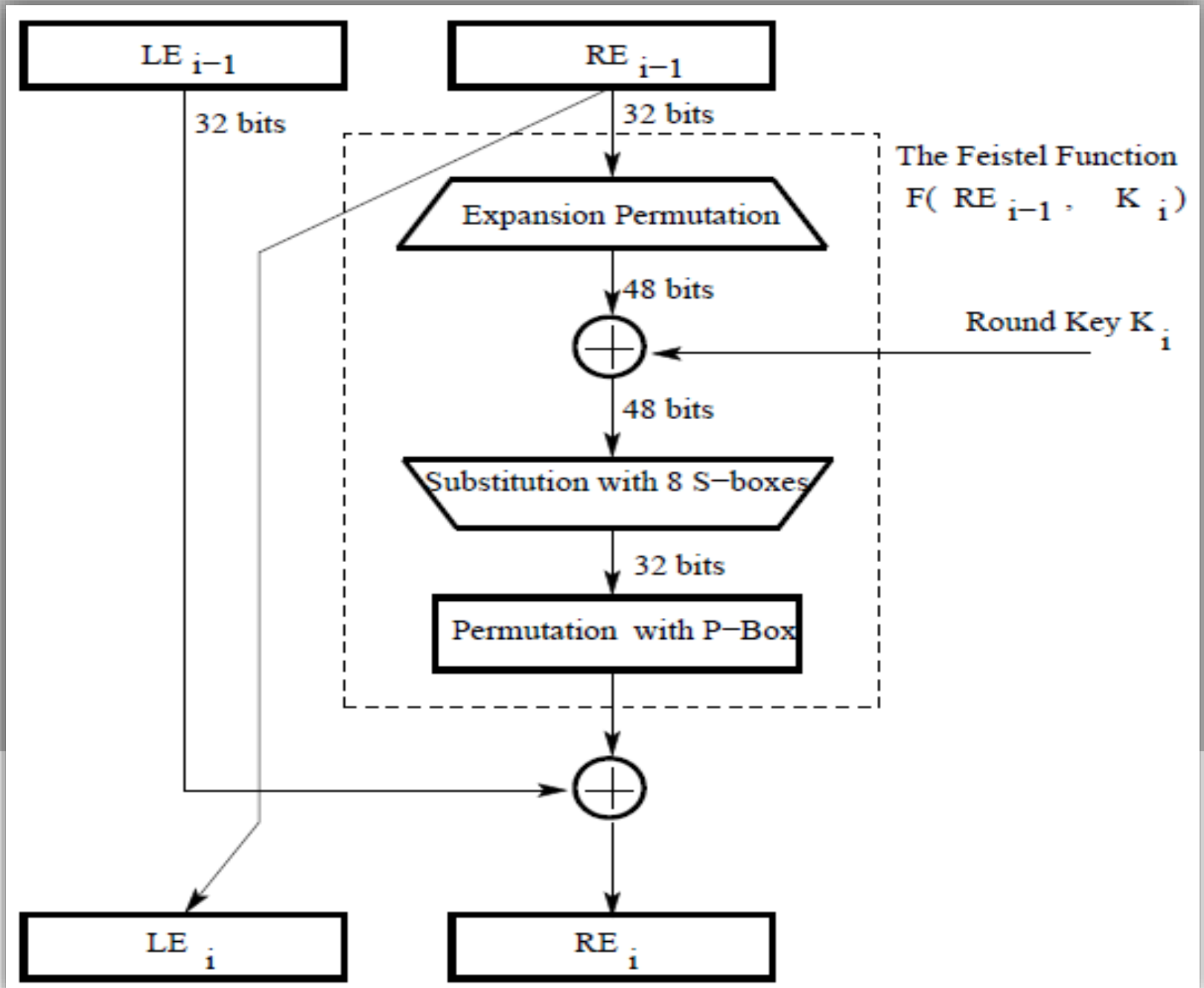
1. As with any Feistel cipher, decryption uses the same algorithm as encryption, except that the application of the subkeys is ***reversed***.

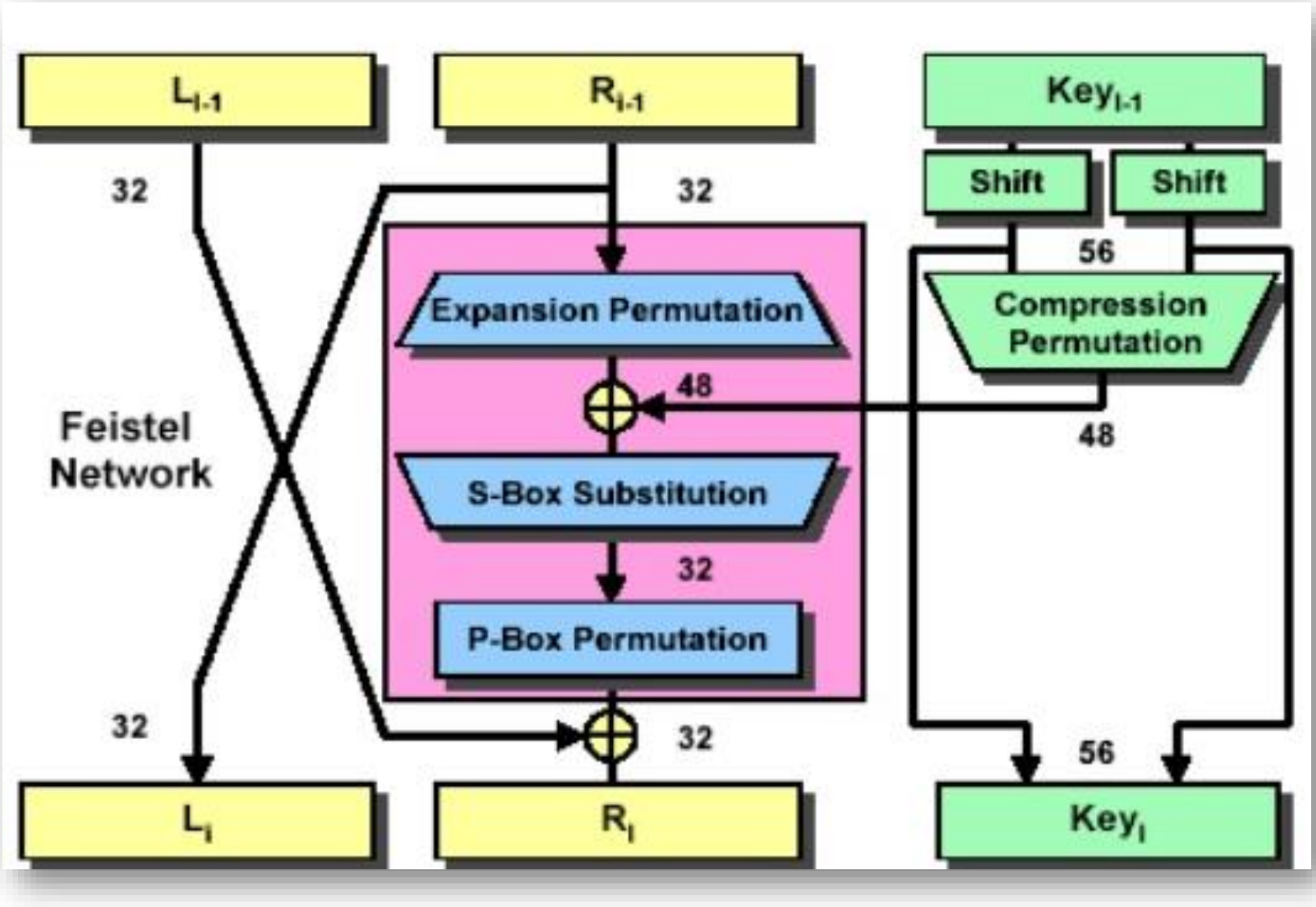
2. Additionally for DES, the initial and final permutations are ***reversed*** (Note that these two permutations can be omitted from DES without security degradation).

One Round of Processing in DES (1)

- The next Figure a single round of processing in DES. The dotted rectangle constitutes the F function.
- The 32-bit right half of the 64-bit input data block is expanded by into a 48-bit block. This is referred to as the ***expansion permutation step*** (E-step).
- The above-mentioned E-step entails the following:

1. first divide the 32-bit block into eight 4-bit words
2. attach an additional bit on the left to each 4-bit word that is the last bit of the previous 4-bit word
3. attach an additional bit to the right of each 4-bit word that is the beginning bit of the next 4-bit word.

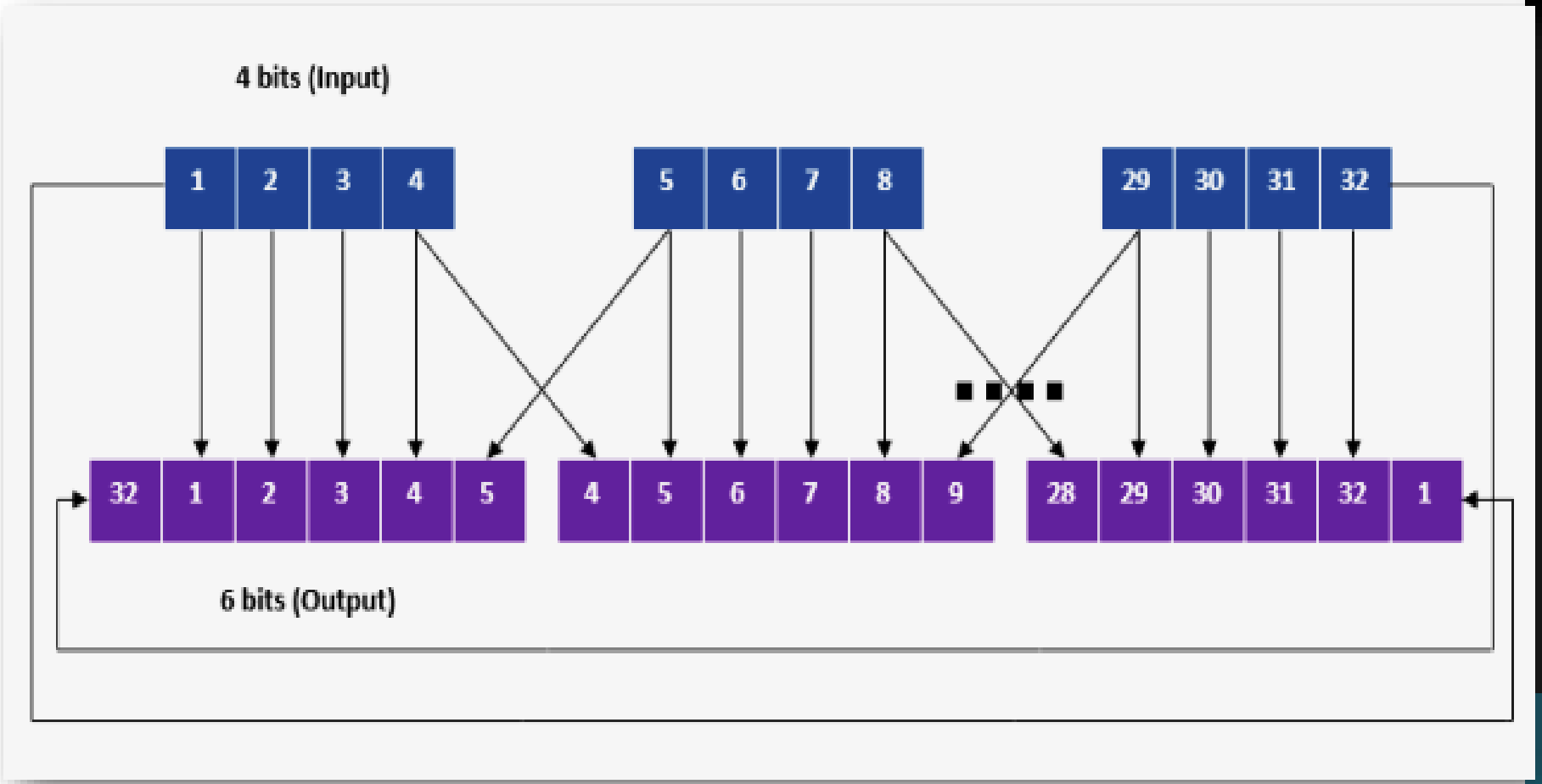

LE i–1
RE i–1
32 bits
32 bits
The Feistel Function
F(RE i–1 , K i)
Expansion Permutation
48 bits
Round Key K i
48 bits
Substitution with 8 S–boxes
32 bits
Permutation with P–Box
LE i
RE i



One Round of DES (with round key generation)

One Round of Processing in DES (2)

- Note that what gets prefixed to the first 4-bit block is the last bit of the last 4-bit block. By the same token, what gets appended to the last 4-bit block is the first bit of the first 4-bit block.
- The 56-bit key is divided into two halves, each half shifted separately, and the combined 56-bit key permuted/contracted to yield a 48-bit round key.
- The 48 bits of the expanded output produced by the E-step are XORed with the round key. This is referred to as ***key mixing***.



Expansion permutation step (E-step)

One Round of Processing in DES (3)

- The output produced by the previous step is broken into eight six-bit words. Each six-bit word goes through a ***substitution step***; its replacement is a 4-bit word. The substitution is carried out with an **S-box**.
- So after all the substitutions, we again end up with a 32-bit word.
- The 32-bits of the previous step then go through a ***P-box*** based permutation, to be shown later.
- What comes out of the P-box is then XORed with the left half of the 64-bit block that we started out with. The output of this XORing operation gives us the ***right half*** block for the next round.

One Round of Processing in DES (4)

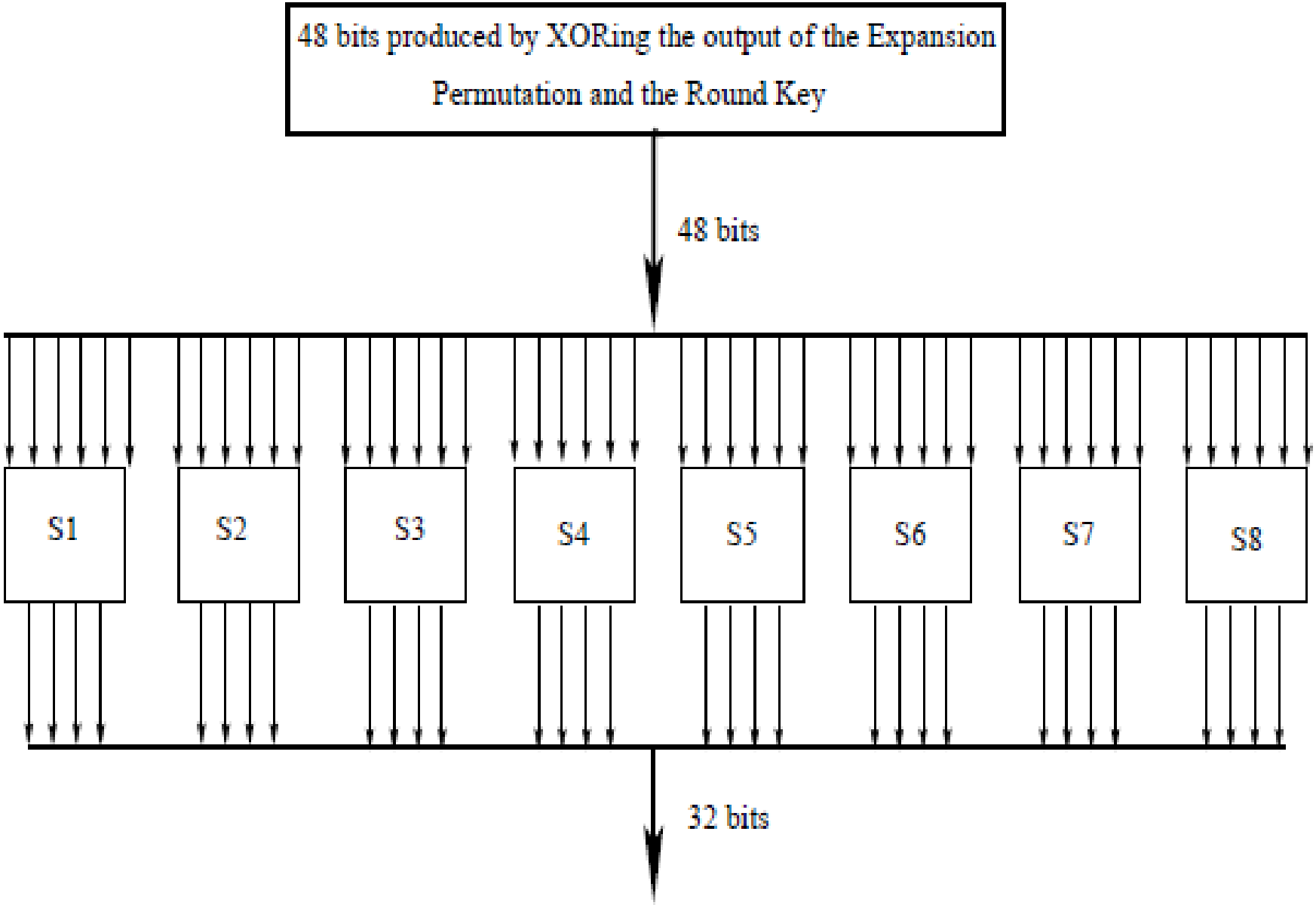
- Note that the goal of the substitution step implemented by the **S-box** is to introduce diffusion in the generation of the output from the input. ***Diffusion means that each plaintext bit must affect as many ciphertext bits as possible***.
- The strategy used for creating the **different round keys** from the main key is meant to introduce confusion into the encryption process. ***Confusion in this context means that the relationship between the encryption key and the ciphertext must be as complex as possible***.
- ***Diffusion*** and ***confusion*** are the two cornerstones of block cipher design.

The S-boxs Step in Each Round (1)

- As shown in the next Figure, the 48-bit input word is divided into eight 6-bit words and each 6-bit word fed into a separate S-box. Each S-box produces a 4-bit output. Therefore, the ***8 S-boxes*** together generate a 32-bit output. As you can see, the overall substitution step takes the 48-bit input back to a 32-bit output.
- Each of the eight S-boxes consists of a 4×16 table lookup for an output 4-bit word. The first and the last bit of the 6-bit input word are decoded into one of ***4 rows*** and the middle 4 bits into one of ***16 columns*** for the table lookup.

The S-boxs Step in Each Round (2)

- The goal of the substitution carried out by an S-box is to ***enhance diffusion***.
- As mentioned previously, the expansion-permutation step (the E-step) expands a 32- bit block into a 48-bit block by attaching a bit at the beginning and a bit at the end of each 4-bit sub-block, the two bits needed for these attachments belong to the adjacent blocks.
- Thus, the row lookup for each of the eight S-boxes becomes a function of the input bits for the previous S-box and the next S-box.


48 bits produced by XORing the output of the Expansion Permutation and the Round Key
48 bits
S1
S2
S3
S4
S5
S6
S7
S8
32 bits

The Substitution Tables

The 4×16 substitution table for S_1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

The 4×16 substitution table for S_2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

- One can similarly specify tables for the other six substitution boxes (S_3, ... , S_8).

S1 EXAMPLE

Row #	S_1	1	2	3	...	7	...	15	Column #
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S(i, j) <16, can be represented with 4 bits

Example: B =101111

b_1b_6 = 11 = row 3

$b_2b_3b_4b_5$= 0111 = column 7

C=7=0111

Another example: B=011011, C=?

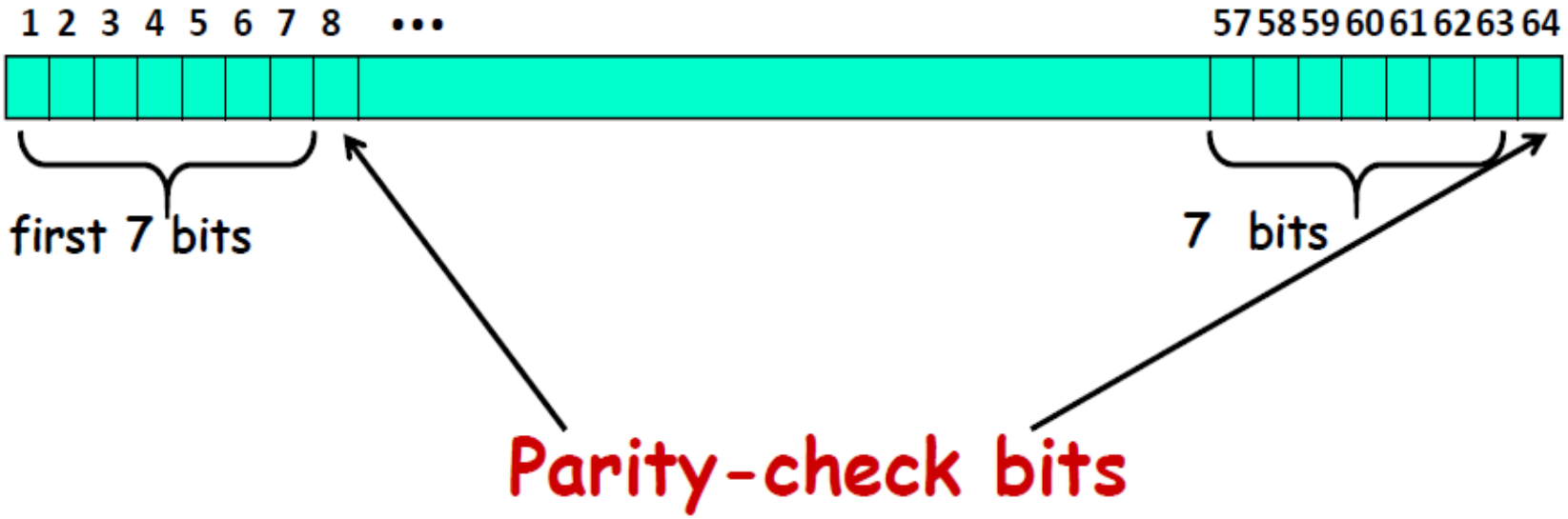
The P-box Permutation in the F Function

P-Box Permutation							
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

- As with ***all permutation tables***, this permutation table simply means that the first output bit will be the 16th bit of the input, the second output bit the 7th bit of the input, and so on, for all of the 32 bits of the output that are obtained from the 32 bits of the input. *Note that bit indexing starts with 1 and not with 0.*

Round Key Generation

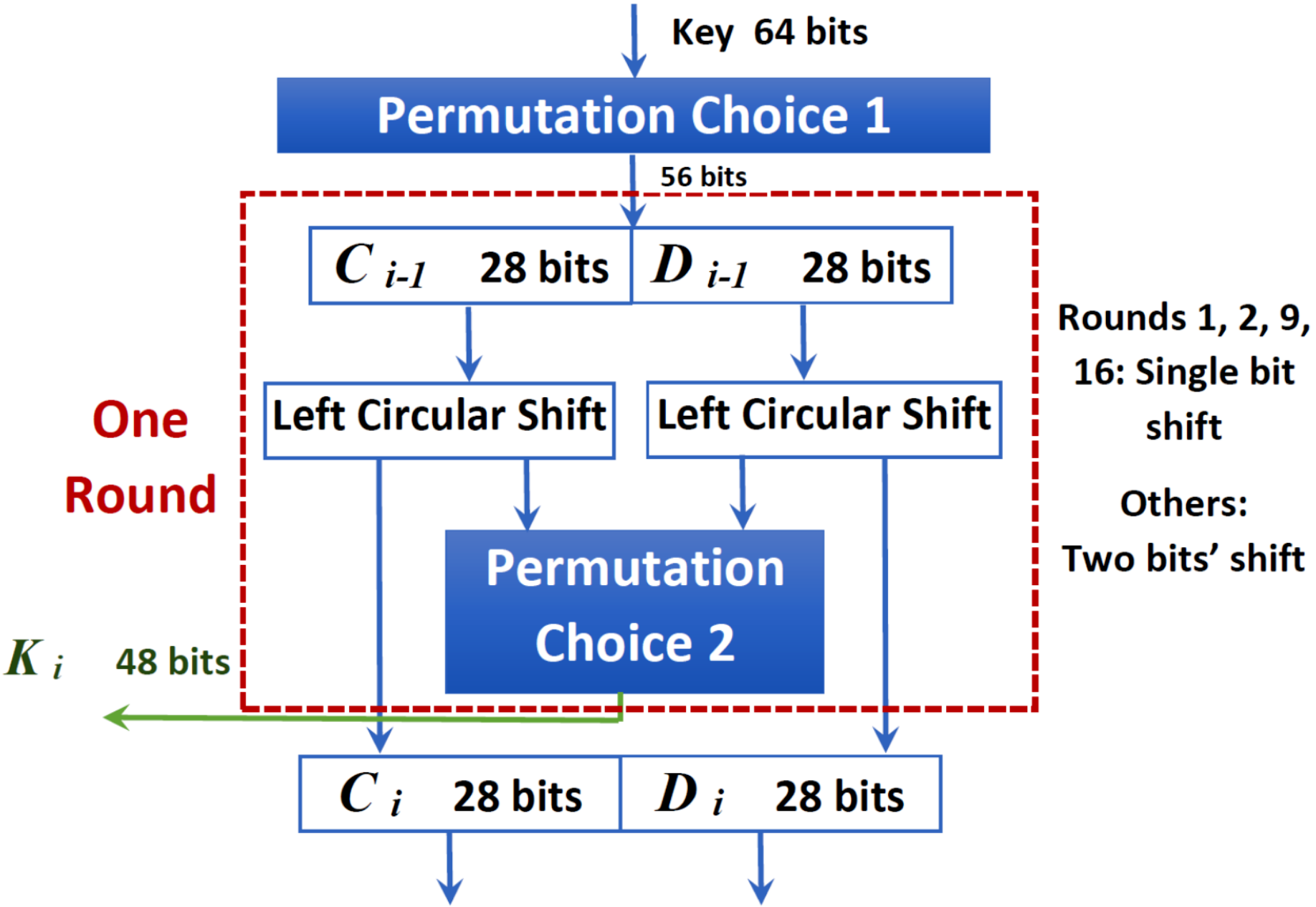
1. The initial 56-bit key may be represented as 8 bytes, with the last bit of each byte used as ***a parity bit***.



2. The relevant 56 bits are subject to a permutation at the beginning before any round keys are generated. This is our ***Permutation Choice 1.***

Round Key Generation (cont.)

3. At the beginning of each round, we divide the 56 relevant key bits into two 28 bit halves and ***circularly shift*** each half by one or two bits.
4. For generating round key, we join together the two halves and apply a 56 bit to 48 bit contracting permutation (***Permutation Choice 2)*** to the joined bit pattern. The resulting 48 bits constitute our round key.
5. The two halves generated in each round ***are fed*** as the two halves going into the next round.



DES Round Key Generation Algorithm

Initial Permutation of the Encryption Key

Permutation Choice 1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

- Note that the bit positions assume that the key bits are addressed 1 through 64 in an 8-byte bit pattern. But note that the *last bit of each byte is used as a parity bit*. Also note that the permutation shown is not a table, in the sense that the rows and the columns do not carry any special and separate meanings. The permutation order for the bits is given by reading the entries shown *from the upper left corner to the lower right corner*.

Contraction-permutation that Generates the 48-bit Round Key from the 56 Key

Permutation Choice 2							
14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

- Also note that the permutation shown is not a table, in the sense that the rows and the columns do not carry any special and separate meanings. The permutation order for the bits is given by reading the entries shown from the upper left corner to the lower right corner.

Security of DES (1)

- ✓ The substitution step is very effective as far as ***diffusion*** is concerned. It has been shown that if you change just one bit of the 64-bit input data block, on the average that alters 34 bits of the ciphertext block.
- ✓ The manner in which the round keys are generated from the encryption key is also very effective as far as ***confusion*** is concerned. It has been shown that if you change just one bit of the encryption key, on the average that changes 35 bits of the ciphertext.
- ✓ Both effects mentioned above are referred to as the ***avalanche effect***.

Security of DES (2)

- And, of course, the 56-bit encryption key means a ***key space*** of size $2^{56} \approx 7.2 \times 10^{16}$.
- Assuming that, on the average, you'd need to try half the keys in a *brute-force attack*, a machine trying one key per ***microsecond*** would take 1142 years to break the code.
- However, a parallel-processing machine trying ***1 million keys simultaneously*** would need only about 10 hours.

Security of DES (3)

- In the design of the DES, the S-boxes were tuned to enhance the resistance of DES to what is known as the ***differential cryptanalysis attack***. Even a slight modification of the S-boxes can weaken the DES to differential cryptanalysis attack.

- For more details on this issue, the student is requested to refer to the textbook.

Finally . . .

- **Acknowledgment:** These lecture notes are based on the textbook by William Stallings and notes prepared by Avinash Kak, Purdue University. My sincere thanks are devoted to them and to all other people who offered the material on the web.
- Students are advised to study and solve the problems and answer the questions in **Assignment-4**.
- Students are also advised to read the following:

1. The material on the Simplified DES (S-DES)
2. Tutorial by Howard Heys on differential cryptanalysis

Chapter 5:

Groups, Rings, and Fields

4th Year- Course, CCSIT, UoA

Lecture Goals

- To review groups, rings, and fields as the fundamental elements of modern algebra, which is an important area of the mathematical foundation of modern cryptography.

What Does it Take for a Set of Objects to form a Group (1)

We need ***four*** properties to be satisfied by a set of objects (along with an operation on the objects) to form a group:

1. ***Closure*** with respect to the operation. Closure means that if a and b are in the set, then the element $a \circ b = c$ is also in the set. The symbol $\circ$ denotes the operation.
2. ***Associativity*** with respect to the operation. Associativity means that $(a \circ b) \circ c = a \circ (b \circ c)$.

What Does it Take for a Set of Objects to form a Group (2)

3. Guaranteed existence of ***a unique identity element*** with regard to the operation. An element i would be called an identity element if for every a in the set, we have $a \circ i = a$.

4. The existence of an ***inverse element*** for each element with regard to the operation. That is, for every a in the set, the set must also contain an element b such that $a \circ b = i$ assuming that i is the identity element.

In general, a group is denoted by $\{G, \circ\}$ where G is the set of objects and $\circ$ is the operation.

Example of a Group

- Let $L_n = \{1, 2, ..., n\}$ denote a set of labels for n objects. [Note that this is NOT the set that we will turn into a group. The set that we will turn into a group is the set of permutations of the labels in L_n, as explained below.]
- Let's now consider the set of all permutations of the labels in the set L_n. Denote this set by S_n. Each element of the set S_n stands for a permutation $(p_1, p_2, p_3, ..., p_n)$ where each $p_i \in L_n$ and $p_i \neq p_j$ whenever $i \neq j$. [The size of the set S_n is $n!$]
- Now consider the following binary operation on any two elements ρ and π of the set S_n: $\pi \circ \rho$ means that we want to permute the elements of ρ according to the elements of π.

An Example that Explains the Operation ' ∘ ' on the Elements of the Set S_n (1)

- Let's say we have $L_3 = \{1, 2, 3\}$ as the set of labels for some three objects.
- We will now construct a set S_3 whose each element will be a distinct permutation of the set of three labels in L_3. That is,

$$S_3 = \{ (p_1, p_2, p_3) \mid p_1, p_2, p_3 \in L_3 \ with \ p_1 \neq p_2 \neq p_3 \}$$

- Now consider the following two elements π and ρ in the set S_3 of permutations:

$$\pi = (3, 2, 1)$$

$$\rho = (1, 3, 2)$$

An Example that Explains the Operation ' ∘ ' on the Elements of the Set S_n (2)

- Let's now consider the following operation between the elements π and ρ:

$$\pi \circ \rho = (3, 2, 1) \circ (1, 3, 2)$$

- To permute ρ according to the elements of π means that we first choose the third element of ρ, followed by the second element of ρ, and followed by the first element of ρ. The result is, of course, the permutation $\{2, 3, 1\}$. So we say

$$\pi \circ \rho = (3, 2, 1) \circ (1, 3, 2) = (2, 3, 1)$$

- Clearly, $\pi \circ \rho \in S_3$.
- This shows that S_3 ***closed*** with respect to the operation '∘'.

Going Back to Our Group Example ... (1)

- Since it is a small enough set, we can also easily demonstrate that S_3 obeys the ***associativity property*** with respect to the '$\circ$' operator. This we can do by showing that for any three elements ρ_1, ρ_2, and ρ_3 of the set S_3, the following will always be true

$$\rho_1 \circ (\rho_2 \circ \rho_3) = (\rho_1 \circ \rho_2) \circ \rho_3$$

- The set S_3 obviously contains a special element (1, 2, 3) that can serve as the ***identity element*** with respect to the operation '$\circ$'. It is definitely the case that for any $\rho \in S_3$ we have

$$(1, 2, 3) \circ \rho = \rho \circ (1, 2, 3) = \rho$$

Going Back to Our Group Example ... (2)

- Again, because S_3 is a small sized set, we can easily demonstrate that for every $\rho \in S_3$ there exists another unique element $\pi \in S_3$ such that

$$\rho \circ \pi = \pi \circ \rho = \textit{the identity element}$$

For each ρ, we may refer to such a π as ρ's ***inverse***. For the sake of convenience, we may use the notation $-\rho$ for such a π.

❖ Obviously, then, S_3 along with the operation '$\circ$' is a ***group***.

More about Groups (1)

- Note that the set S_n of all permutations of the labels in the set L_n can only be finite. As a result, S_n along with the operation $'\circ'$ forms a ***finite*** group.

- However, a group can also be ***infinite***. The set of *all integers*, positive, negative and zero, along with the operation of arithmetic addition is an infinite group.

- If the operation on the set elements is *commutative*, the group is called an ***abelian*** group. An operation $\circ$ is commutative if $a \circ b = b \circ a$.

More about Groups (2)

- Is $\{S_n, \circ\}$ an abelian group? If not for n in general, is $\{S_n, \circ\}$ an abelian group for any particular value of n? [S_n is abelian for only $n = 2$.]
- Is the set of all integers, positive, negative, and zero, along with the operation of arithmetic addition an abelian group? [The answer is yes.]
- A group is also denoted **{*G*, +}**, where *G* denotes the set and '+' the operation.
- Note that the name of the operation, ***addition***, used in the context of defining a group may have nothing to do with your usual arithmetic definition of addition.

If the group operation is referred to as addition, then a group also allows for subtraction

- A group is guaranteed to have a special element, the ***identify element***. The identity element of a group is frequently denoted by the symbol **0** .
- For every element ρ_1, the group must contain its inverse element ρ_2 such that $\rho_1 + \rho_2 = 0$ where the operator '+' is the group operator.
- So if we maintain the illusion that we want to refer to this operation as addition, we can think of ρ_2 is the ***additive inverse*** of ρ_1 and even denote it by $-\rho_1$. We can therefore write $\rho_1 + (-\rho_1) = 0$ or more compactly as $\rho_1 - \rho_1 = 0$.
- In general $\rho_1 - \rho_2 = \rho_1 + (-\rho_2)$ where $-\rho_2$ is the additive inverse of ρ_2 with respect to the group operator + . We may now refer to an expression of the sort $\rho_1 - \rho_2$ as representing ***subtraction***.

Rings

- If we can define ***one more operation*** on an ***abelian group***, we have a ***ring***, provided the elements of the set satisfy some properties with respect to this new operation also.
- Just to set it apart from the operation defined for the abelian group, we will refer to the new operation as ***multiplication***. Note that the use of the name 'multiplication' for the new operation is merely a notational convenience.
- A ring is typically denoted **{*R*, +, ×}** where R denotes the set of objects, '+' the operator with respect to which *R* is an abelian group, the '×' the additional operator needed for *R* to form a ring.

Rings: Properties of the elements with respect to the other operator

1. R must be ***closed*** with respect to the additional operator '×'.
2. R must exhibit ***associativity*** with respect to the additional operator '×'.
3. The additional operator (that is, the "multiplication operator") must ***distribute*** over the group addition operator. That is

$$a \times (b + c) = a \times b + a \times c$$
$$(a + b) \times c = a \times c + b \times c$$

✓ The "multiplication" operation is frequently shown by just concatenation in such equations:

$$a\,(b + c) = a\,b + a\,c$$
$$(a + b)\,c = a\,c + b\,c$$

Examples of a Ring

- ✓ For a given value of N , the set of all $N \times N$ square matrices over the real numbers under the operations of *matrix addition* and *matrix multiplication* constitutes a ***ring***.
- ✓ The set of *all even integers*, positive, negative, and zero, under the operations arithmetic addition and multiplication is a ***ring***.
- ✓ The set of *all integers* under the operations of arithmetic addition and multiplication is a ***ring***.
- ✓ The set of *all real numbers* under the operations of arithmetic addition and multiplication is a ***ring***.

Commutative Rings

- A ring is commutative if the ***multiplication operation is commutative*** for all elements in the ring. That is, if all a and b in R satisfy the property

$$a\,b = b\,a$$

- Examples of a commutative ring:

1. The set of ***all even integers***, positive, negative, and zero, under the operations arithmetic addition and multiplication.
2. The set of ***all integers*** under the operations of arithmetic addition and multiplication.
3. The set of ***all real numbers*** under the operations of arithmetic addition and multiplication.

Integral Domain

An ***integral domain*** $\{R, +, \times\}$ is a commutative ring that obeys the following two additional properties:

1. ***Additional Property 1***: The set R must include an *identity element* for the *multiplicative operation*. That is, it should be possible to symbolically designate an element of the set R as '1' so that for every element a of the set we can say $a \; 1 = 1 \; a = a$
2. ***Additional Property 2***: Let 0 denote the identity element for the *addition operation*. If a multiplication of any two elements a and b of R results in 0, that is if $a \, b = 0$ then either a or b ***must be*** 0.

Examples of Integral Domain

1. The set of ***all integers*** under the operations of arithmetic addition and multiplication.

2. The set of ***all real numbers*** under the operations of arithmetic addition and multiplication.

Fields

A ***field***, denoted $\{F, +, \times\}$, is an *integral domain* whose elements satisfy the following *additional property*:

- For *every element* a in F, except the element designated 0 (the identity element for the '+' operator), there must also exist in F its ***multiplicative inverse***. That is, if $a \in F$ and $a \neq 0$, then there must exist an element $b \in F$ such that

$$a\,b = b\,a = 1$$

where '1' symbolically denotes the element which serves as the identity element for the multiplication operation. For a given a, such a b is often designated a^{-1}.

Positive and Negative Examples of Fields

- ✓ The set of *all real numbers* under the operations of arithmetic addition and multiplication is a ***field***.
- ✓ The set of *all rational numbers* under the operations of arithmetic addition and multiplication is a ***field***.
- ✓ The set of *all complex numbers* under the operations of complex arithmetic addition and multiplication is a ***field***.
- ❑ The set of *all even integers*, positive, negative, and zero, under the operations arithmetic addition and multiplication is ***NOT a field*** .
- ❑ The set of *all integers* under the operations of arithmetic addition and multiplication is ***NOT a field***.

Finally . . .

- **Acknowledgment:** These lecture notes are based on the textbook by William Stallings and notes prepared by Avinash Kak, Purdue University. My sincere thanks are devoted to them and to all other people who offered the material on the web.

- Students are advised to study and solve the problems and answer the questions in **Assignment-5**.

Chapter 6:

Modular Arithmetic

4th Year- Course, CCSIT, UoA

Lecture Goals

1. To review modular arithmetic
2. To present Euclid's gcd algorithms
3. To present the prime finite field Z_p
4. To show how Euclid's gcd algorithm can be extended to find multiplicative inverses

Modular Arithmetic Notation (1)

- Given any integer a and a positive integer n, and given a division of a by n that leaves the remainder between 0 and $n - 1$, both inclusive, we define $a \bmod n$ to be the ***remainder***. Note that the remainder must be between 0 and $n - 1$, both ends inclusive, even if that means that we must use a negative quotient when dividing a by n.
- We will call two integers a and b to be ***congruent*** modulo n if

$$(a \bmod n) = (b \bmod n)$$

- Symbolically, we will express such a congruence by

$$a \equiv b \pmod{n}$$

Modular Arithmetic Notation (2)

- We say a non-zero integer a is a ***divisor*** of another integer b provided there is no remainder when we divide b by a. That is, when $b = ma$ for some integer m.

- When a is a divisor of b, we express this fact by $a \mid b$.

Examples of Congruences

- Here are some congruences modulo 3:

$$7 \equiv 1 \ (mod\ 3) \qquad -8 \equiv 1 \ (mod\ 3)$$
$$-2 \equiv 1 \ (mod\ 3) \qquad 7 \equiv -8 \ (mod\ 3)$$
$$-2 \equiv 7 \ (mod\ 3)$$

- One way of seeing the above congruences (for mod 3 arithmetic):

...	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	...
...-	9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	...

  where the top line is the output of modulo 3 arithmetic and the bottom line the set of all integers.

- Obviously, then, modulo n arithmetic ***maps all integers into*** the set $\{0, 1, 2, 3, ..., n-1\}$.

Modular Arithmetic Operations

- The following equalities are easily shown to be true (with the ordinary meaning to be ascribed to the arithmetic operators):

$$[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$$
$$[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$$
$$[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$$

- For arithmetic modulo n, let Z_n denote the set

$$Z_n = \{0, 1, 2, 3, ..., n - 1\}$$

- $\boldsymbol{Z_n}$ is obviously the set of remainders in arithmetic modulo n. It is officially called the ***set of residues***.

Properties of the Set Z_n (1)

1. ***Commutativity:***

$$(w + x) \bmod n = (x + w) \bmod n$$

$$(w \times x) \bmod n = (x \times w) \bmod n$$

2. ***Associativity:***

$$[(w + x) + y] \bmod n = [w + (x + y)] \bmod n$$

$$[(w \times x) \times y] \bmod n = [w \times (x \times y)] \bmod n$$

3. ***Distributivity of multiplication over addition:***

$$[w \times (x + y)] \bmod n = [(w \times x) + (w \times y)] \bmod n$$

Properties of the Set Z_n (2)

4. ***Existence of Identity Elements:***

$$(0 + w) \bmod n = (w + 0) \bmod n$$

$$(1 \times w) \bmod n = (w \times 1) \bmod n$$

5. ***Existence of Additive Inverses:*** *For each* $w \in Z_n$ *there exists a* z *such that* $w + z = 0 \bmod n$

- **Z_n is a commutative ring.** Why? [See the previous lecture]

- ***Actually, Z_n is more than a commutative ring, but not quite an integral domain.*** What do we mean by that? [Because it contains a multiplicative identity element. Commutative rings are not required to possess multiplicative identities.]

More about Z_n

- ***Why is Z_n not an integral domain?*** [Even though it possesses a multiplicative identity, it does NOT satisfy the other condition of integral domains which says that if $a \times b = 0$ then either a or b must be zero. Consider modulo 8 arithmetic. We have $2 \times 4 = 0$, which is a clear violation of the second rule for integral domains.]
- ***Why is Z_n not a field?***
- ***Is Z_n a group? If so, what is the group operator?*** [The group operator is the modulo n addition.]
- ***Is Z_n an abelian group?***
- ***Is Z_n a ring?***

Asymmetries between modulo addition and modulo multiplication over Z_n (1)

- For every element of Z_n, there exists an ***additive inverse*** in Z_n. But there *does not* exist a ***multiplicative inverse*** for every element of Z_n.
- Shown below are the additive and the multiplicative inverses for ***modulo 8*** arithmetic:

Z	=	0	1	2	3	4	5	6	7
8									
addit. inv.	=	0	7	6	5	4	3	2	1
multi. inv	=	-	1	-	3	-	5	-	7

Asymmetries between modulo addition and modulo multiplication over Z_n (2)

- Note that the ***multiplicative inverses*** exist for only those elements of Z_n that are ***relatively prime*** to n. Two integers are relatively prime to each other if the integer 1 is their only one common positive divisor. More formally, two integers a and b are relative prime to each other if $gcd\,(a, b) = 1$ where ***gcd*** denotes the ***Greatest Common Divisor.***

- The following property of ***modulo n addition*** is the same as for ordinary addition: $(a + b) \equiv (a + c)\ (mod\ n)$ implies $b \equiv c\ (mod\ n)$

- But a similar property is **NOT** obeyed by ***modulo n multiplication***. That is $(a \times b) \equiv (a \times c)\ (mod\ n)$ does not imply $b \equiv c\ (mod\ n)$ unless a and n are relatively prime to each other.

Asymmetries between modulo addition and modulo multiplication over Z_n (3)

- That the modulo n addition property stated above should hold true for all elements of Z_n follows from the fact that the ***additive inverse*** $-a$ exists for every $a \in Z_n$. So we can add $-a$ to both sides of the equation to prove the result.
- To prove the same result for modulo n multiplication, we will need to multiply both sides of the second equation above by the ***multiplicative inverse*** a^{-1}. But, not all elements of Z_n possess multiplicative inverses.
- Since the answer to the question whether two integers are *relatively prime* to each other depends on their *greatest common divisor (gcd)*, let's explore next the world's most famous algorithm for finding the gcd of two integers.

Euclid's Method for Finding the Greatest Common Divisor of Two Integers

Euclid's GCD is based on the following observations:

1. $gcd\ (a, a) = a$
2. *if* $b \mid a$ *then* $gcd\ (a, b) = b$
3. $gcd\ (a, 0) = a$ *since it is always true that* $a \mid 0$
4. Assuming without loss of generality that a is larger than b, it can be shown that

$$gcd\ (a, b) = gcd\ (b, a \bmod b)$$

❖This is the heart of Euclid's algorithm (now over 2000 years old).

Steps in a Recursive Invocation of Euclid's Algorithm

gcd(b_1, b_2):

= gcd (b_2, b_1 mod b_2) = gcd (b_2, b_3)

= gcd (b_3, b_2 mod b_3) = gcd (b_3, b_4)

= gcd (b_4, b_3 mod b_4)

....

until b_{m-1} mod b_m = 0

then gcd (b_1, b_2) = b_m

- Note that the algorithm works for any two non-negative integers b_1 and b_2 *regardless of which is the larger integer*. If the first integer is smaller compared to the second integer, the first iteration will swap the two.

Examples of Euclid's Algorithm in Action

```
gcd( 70, 38 )
     =   gcd( 38, 32 )
     =   gcd( 32, 6 )
     =   gcd( 6, 2 )
     =   gcd( 2, 0 )

Therefore,  gcd( 70, 38 )  = 2
```

```
gcd( 8, 17 ):
    =  gcd( 17, 8 )
    =  gcd( 8, 1 )
    =  gcd( 1, 0 )

Therefore,  gcd( 8, 17 )  = 1
```

- When the smaller of the two numbers is 1 (which happens when the two starting numbers are ***relatively prime***), there is no need to go to the last step in which the smaller of the two numbers is 0.

An Example of Euclid's Algorithm for Moderately Large Numbers

```
gcd( 40902, 24140 )
     =    gcd( 24140, 16762 )
     =    gcd( 16762, 7378 )
     =    gcd( 7378, 2006 )
     =    gcd( 2006, 1360 )
     =    gcd( 1360, 646 )
     =    gcd( 646, 68 )
     =    gcd( 68, 34 )
     =    gcd( 34, 0 )

Therefore,  gcd( 40902, 24140 )  = 34
```

Prime Finite Fields (1)

- Earlier we showed that, in general, Z_n is a ***commutative ring***.
- The main reason for why, in general, Z_n is only a commutative ring and not a finite field is because ***not*** every element in Z_n is guaranteed to have a multiplicative inverse.
- In particular, as shown before, an element a of Z_n does not have a multiplicative inverse if a is ***not relatively prime*** to the modulus n.

Prime Finite Fields (2)

- What if we choose the modulus n to be a ***prime number***? (A prime number has only two divisors, one and itself.)

- For ***prime*** n, every element $a \in Z_n$ will be relatively prime to n. That implies that there will exist a multiplicative inverse for every $a \in Z_n$ for prime n.

- Therefore, $\boldsymbol{Z_p}$ is a finite field if we assume p denotes a prime number. Z_p is sometimes referred to as a ***prime finite field***. Such a field is also denoted ***GF (p)***, where *GF* stands for "Galois Field".

What Happened to the Main Reason for Why Z_n Could not be an Integral Domain? (1)

- Earlier, when we were looking at how to characterize Z_n, we said that, although it possessed a multiplicative identity, it could not be an integral domain because Z_n allowed for the equality $a \times b = 0$ even for non-zero a and b. (Recall, 0 means the additive identity element.)
- If we have now decided that Z_p is a *finite field* for prime p because every element in Z_p has a unique multiplicative inverse, are we sure that we can now also guarantee that if $a \times b = 0$ then either a or b must be 0?

What Happened to the Main Reason for Why Z_n Could not be an Integral Domain? (2)

- Yes, *we have that guarantee* because $a \times b = 0$ for general Z_n occurs only when non-zero a and b are factors of the modulus n. When n is a prime, its only factors are 1 and n. So with the elements of Z_n being in the range 0 through $n - 1$, the only time we will see $a \times b = 0$ is when either a is 0 or b is 0.

Finding Multiplicative Inverses for Elements of Z_p (1)

- In general, to find the multiplicative inverse of $a \in Z_n$, we need to find the element b such that

$$a \times b = 1 \; mod \; n$$

- Based on the discussion so far, we can say that the multiplicative inverses exist for all $a \in Z_n$ for which we have

$$gcd\,(a, n) = 1$$

- Obviously, when n equals a prime p, this condition will always ***be satisfied by all elements*** of Z_p.

Finding Multiplicative Inverses for Elements of Z_p (2)

- In general, it can be shown that when a and n are any pair of positive integers, the following must always hold for some integers x and y (that may be positive or negative or zero):

$$gcd\,(a, n) = x \times a + y \times n$$

- This is known as the ***Bezout's Identity***.
- For example, when $a = 16$ and $n = 6$, we have $gcd\,(16, 6) = 2$. We can certainly write: $2 = (-1) \times 16 + 3 \times 6 = 2 \times 16 + (-5) \times 6$. This shows that x and y *do not have to be unique* in Bezout's identity for given a and n.

Finding Multiplicative Inverses Using Bezout's Identity (1)

- Given an a that is relatively prime to n, we must obviously have $gcd\,(a, n) = 1$. Such an a and n must satisfy the following constraint for some x and y:

$$x \times a + y \times n = 1$$

- Let's now consider this equation modulo n. Since y is an integer, obviously $y \times n \bmod n$ equals 0. Thus, it must be the case that, considered modulo n, x equals a^{-1}, the multiplicative inverse of a modulo n.
- The equation shown above gives us a ***strategy*** for finding the multiplicative inverse of an element a :

Finding Multiplicative Inverses Using Bezout's Identity (2)

This strategy is:

1. We use the same Euclid algorithm as before to find the $gcd\,(a, n)$,
2. but now at each step we write the expression in the form $a \times x + n \times y$ for the remainder
3. eventually, before we get to the remainder becoming 0, when the remainder becomes 1 (which will happen only when a and n are relatively prime), x will automatically be the multiplicative inverse we are looking for.

The Extended Euclid's Algorithm for Calculating the Multiplicative Inverse (1)

- So our quest for finding the ***multiplicative inverse (MI)*** of a number *num* modulo *mod* boils down to expressing the residues at each step of Euclid's recursion as a linear sum of *num* and *mod*, and, when the recursion terminates, taking for ***MI*** the coefficient of *num* in the final linear summation.
- As we step through the recursion called for by Euclid's algorithm, the originally supplied values for *num* and *mod* become modified as shown earlier. So let's use *NUM* to refer to the originally supplied value for *num* and *MOD* to refer to the originally supplied value for *mod*.

The Extended Euclid's Algorithm for Calculating the Multiplicative Inverse (2)

- Let x represent the coefficient of NUM and y the coefficient of MOD in our linear summation expressions for the residue at each step in the recursion. So our goal is to express the residue at each step in the form

$$residue = x * NUM + y * MOD$$

- And then, when the *residue* is 1, to take the value of x as the multiplicative inverse of NUM modulo MOD, assuming, the MI exists.

The Extended Euclid's Algorithm for Calculating the Multiplicative Inverse (3)

What is interesting is that as the Euclid's recursion proceeds, the new values of x and y can be computed directly from their current values and their previous values (which we will denote x_{old} and y_{old}) by the formulas:

$$x <= x_{old} + x * q$$

$$y <= y_{old} + y * q$$

where q is the integer quotient obtained by dividing *num* by *mod*. To establish this fact, the following table in the next page illustrates an example for calculating *gcd* (17, 32) where we are interested in finding the ***MI*** of 17 modulo 32:

Example: Calculating the Multiplicative Inverse of 17 mod 32

Row	q = num//mod	num	mod	x	y
A. Initialization				1	0
B.		17	32	0	1
C. gcd(17, 32)					
D. residue = 17	17//32 = 0	32	17	1	0
E. gcd(32, 17)					
F. residue = 15	32//17 = 1	17	15	-1	1
G. gcd(17, 15)					
H. residue = 2	17//15 = 1	15	2	2	-1
I. gcd(15, 2)					
J. residue = 1	15//2 = 7	2	1	-15	8

Rules for Table Construction in the Extended Euclid's Algorithm (1)

1. Rows A and B of the table are for ***initialization***. We set x_{old} and y_{old} to 1 and 0, respectively, and their current values to 0 and 1. At this point, *num* is 17 and *mod* 32.

2. Note that the ***first thing*** we do in each new row is to calculate the ***quotient*** obtained by dividing the current *num* by the current *mod*.

3. ***Only after that*** we ***update the values*** of *num* and *mod* in that row according to Euclid's recursion.

Rules for Table Construction in the Extended Euclid's Algorithm (2)

- For example, when we calculate *q* in row F, the current *num* is 32 and the current *mod* 17. Since the integer quotient obtained when you divide 32 by 17 is 1, the value of *q* in this row is 1.

- Having obtained the ***residue***, we now invoke ***Euclid's recursion***, which causes *num* to become 17 and *mod* to become 15 in row F.

4. We *update* the ***values of x*** on the basis of its current value and its previous value and the current value of the quotient *q*

Rules for Table Construction in the Extended Euclid's Algorithm (3)

- For example, when we calculate the value of x in row J, the current value for x at that point is the one shown in row H, which is 2, and the previous value for x is shown in row F, which is -1. Since the current value for the quotient q is 7, we obtain the new value of x in row J by $-1 - 7 * 2 = -15$. This is according to the update formula for the x coefficients:

$$x_{\text{new}} = x_{old} - q \times x_{\text{current}}$$

5. The same goes for the ***variable y*** . It is ***updated*** in the same manner through the formula

$$y_{\text{new}} = y_{old} - q \times y_{\text{current}}$$

Final remarks on calculating the multiplicative inverse

- You should stop the iteration when the residue = 1, and pick the corresponding x value as the required MI.
- If the x value is negative, you have to change it to positive by adding the required multiple of the mod.
- For the above example, when residue =1 , x = - 15 which is a negative value. So, you add the value of the mod (32): - 15 + 32 = 17. This means that 17 is the MI of itself mod 32.

Checking solution correctness for calculating the multiplicative inverse

- You can check solution correctness by:

1. Multiplying the number by its MI to obtain 1, or
2. Checking Bezout identity when residue = 1

- For the above example,

  $17 * 17 \bmod 32 = 289 - (32 * 9) = 1$ (O.K.)

Euler's Totient Function (1)

- Euler's totient function, written $\varphi(n)$, is defined as the number of positive integers less than n and relatively prime to n.
- By convention, $\varphi(1) = 1$.
- Example: Determine $\varphi(35)$.
  - To determine $\varphi(35)$, we list all of the positive integers less than 35 that are relatively prime to it:
  - 1, 2, 3, 4, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 26, 27, 29, 31, 32, 33, 34
  - There are 24 numbers on the list, so $\varphi(35) = 24$.

Euler's Totient Function (2)

- Example: Determine φ(37).
  - Because 37 is prime, all of the positive integers from 1 through 36 are relatively prime to 37.
  - Thus φ(37) = 36.

- It should be clear that, for a prime number p,

$$\phi(p) = p - 1$$

- Now suppose that we have two prime numbers p and q with $p \neq q$. Then we can show that, for $n = p * q$,

$$\phi(n) = \phi(pq) = \phi(p) \times \phi(q) = (p - 1) \times (q - 1)$$

Euler's Totient Function (3)

- Example: Determine $\varphi(21)$.

  - $\varphi(21) = \varphi(3) * \varphi(7) = (3 - 1) * (7 - 1) = 2 * 6 = 12$

  - where the 12 integers are
    {1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20}.

Finally . . .

- **Acknowledgment:** These lecture notes are based on the textbook by William Stallings and notes prepared by Avinash Kak, Purdue University. My sincere thanks are devoted to them and to all other people who offered the material on the web.
- Students are advised to study and solve the problems and answer the questions in **Assignment-6**.
- Students are encouraged to read more about the details of the ***Hill cipher*** in the supplied ***tutorial***.

Chapter 7:
Using Block Ciphers in Real-World Systems

4th Year- Course, CCSIT, UoA

Lecture Goals

1. To present Double DES (2DES) and its vulnerability to the meet-in-the-middle attack
2. To present Triple DES (3DES)
3. To present the five different modes in which a block cipher can be used in practical systems for secure communications

Double DES (2DES)

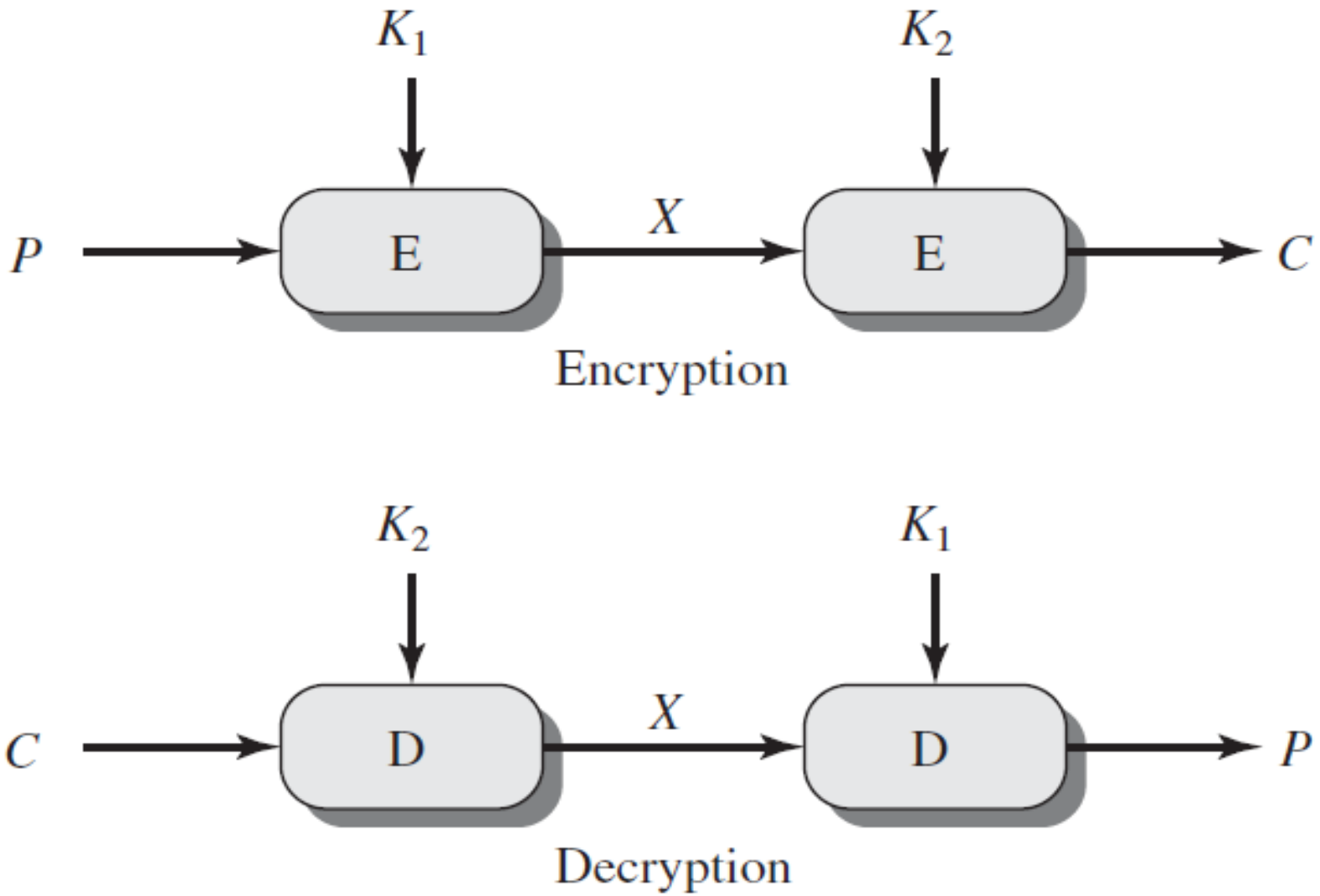
- The simplest form of multiple encryption with DES is ***double DES*** that has two DES-based encryption stages using two different keys (See next Figure).
- Let's say that P represents a 64-byte block of plaintext. Let E represent the process of encryption that transforms a plaintext block into a ciphertext block. Let's use two 56-bit encryption keys K_1 and K_2 for a double application of DES to the plaintext. Let C represent the resulting block of ciphertext. We have

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

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

where D represents the process of decryption.

- With two keys, each of length 56 bits, double DES in effect uses a ***112 bit key***.



Double DES Encryption and Decryption

Vulnerability of double DES to the meet-in-the-middle attack

- Would this result in a dramatic increase in the cryptographic strength of the cipher — at least against the ***brute-force attacks*** to which the regular DES is so vulnerable- ?

[The answer is NO!]

- Any double block cipher, that is a cipher that carries out double encryption of the plaintext using two different keys in order to increase the cryptographic strength of the cipher, is open to what is known as ***the meet-in-the-middle attack***.

- It can be shown that the effort required to do such attack is 2^{56}, which is ***comparable*** to the effort required to break the regular DES.

Triple DES With Two Keys (1)

- An obvious defense against the meet-in-the-middle attack is to use ***triple DES (3DES)***.
- The most straightforward way to use triple DES is to employ *three stages of encryption*, each with its own key:

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

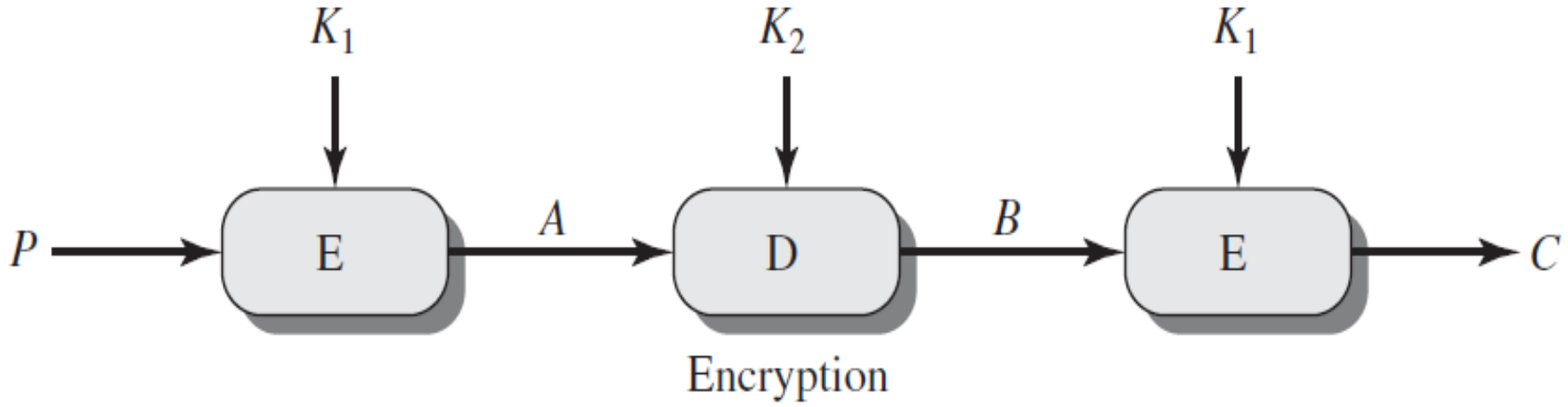
But this calls for ***168-bit keys***, which is considered to be unwieldy for many applications.

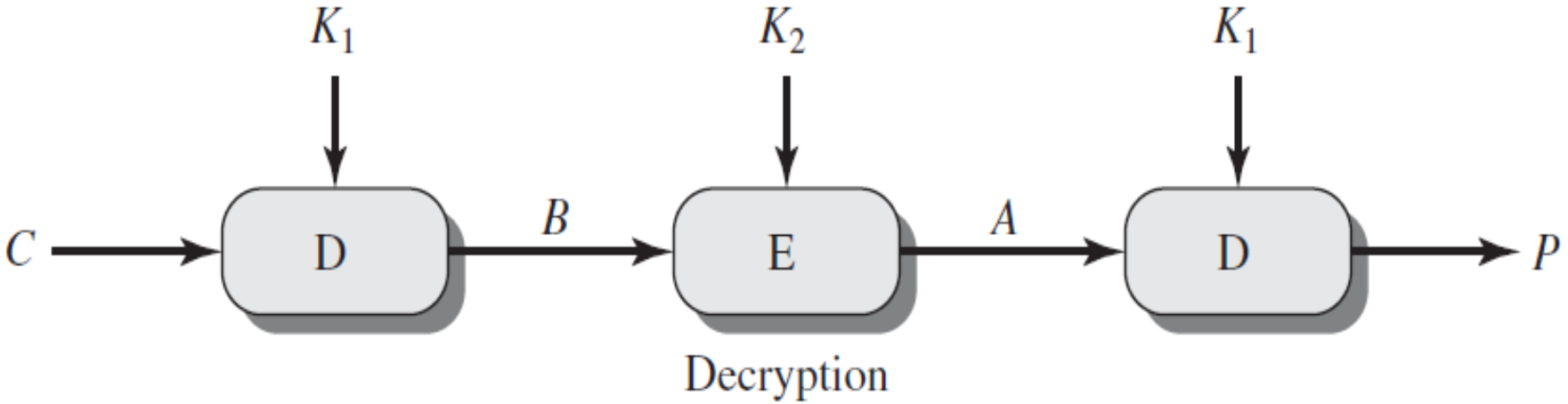
- One way to use triple DES is with just ***two keys*** as follows

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

Triple DES With Two Keys (2)

- In this latter case, one stage of encryption is followed by one stage of decryption, followed by another stage of encryption. This is also referred to as ***EDE encryption*** (See next Figure).
- There is an important reason for juxtaposing a stage of decryption between two stages of encryption: it makes the triple DES system ***backward compatible*** with regular DES by setting $K_1 = K_2$ in triple DES.
- It is important to realize that juxtaposing a decryption stage between two encryption stages ***does not weaken*** the resulting cryptographic system in any way.





EDE and DED

Triple DES with two keys (3)

- Recall, decryption in DES works in exactly the same manner as encryption. So if you encrypt data with one key and try to decrypt with a different key, the final output will be still be an encrypted version of the original input.
- Triple DES with two keys is ***a popular alternative*** to regular DES.
- It is theoretically possible to extend the meet-in-the-middle attack to the case of 3DES based on two keys. However, the running time of such attack, given n pairs of (P, C), would be of the order of $2^{120-\log n}$

Triple DES with Three Keys (1)

- Here is a 3-key version of a more secure cipher that is based on multiple encryption with DES:

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

where the decryption step in the middle is purely for the sake of ***backward compatibility*** with the regular DES, with 2DES, and with 3DES using two keys.

- When all three keys are the same, that is when $K_1 = K_2 = K_3$, 3DES with three keys become identical to regular DES.

Triple DES with Three Keys (2)

- When $K_1 = K_3$, we have 3DES with two keys.
- Note that as with 3DES using two keys, the decryption stage in the middle ***does NOT reduce*** the cryptographic strength of 3DES with three keys, especially since the encryption and decryption algorithms are the same in DES.
- A number of internet-based applications have adopted 3DES with three keys. These include PGP and S/MIME.

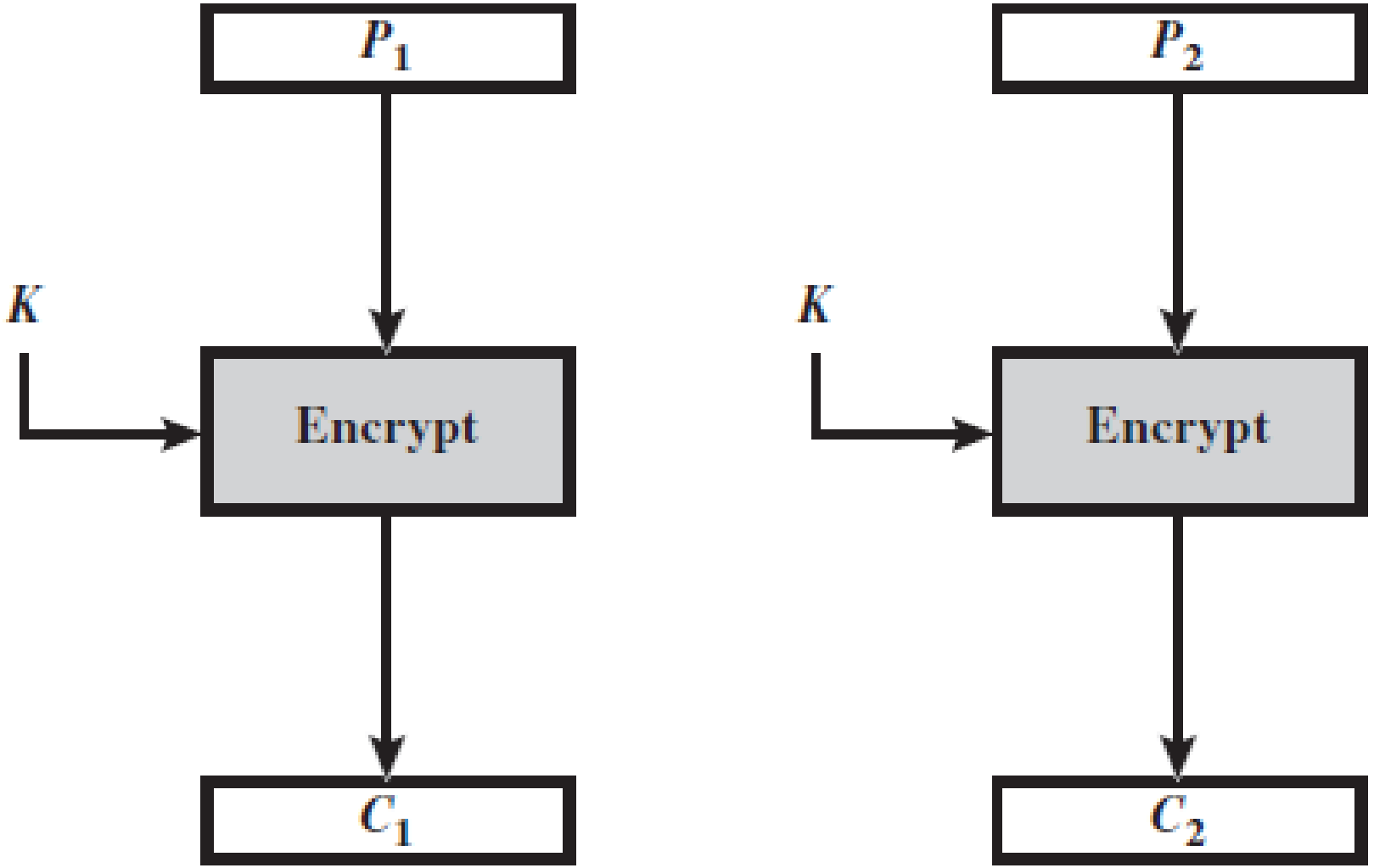
Modes of Operation for Block Ciphers

There are **five** different modes in which a block cipher, such as DES or AES, can be used:

1. Electronic Code Book (ECB)
2. Cipher Block Chaining Mode (CBC)
3. Cipher Feedback Mode (CFB)
4. Output Feedback Mode (OFB)
5. Counter Mode (CTR)

Electronic Code Book (ECB)

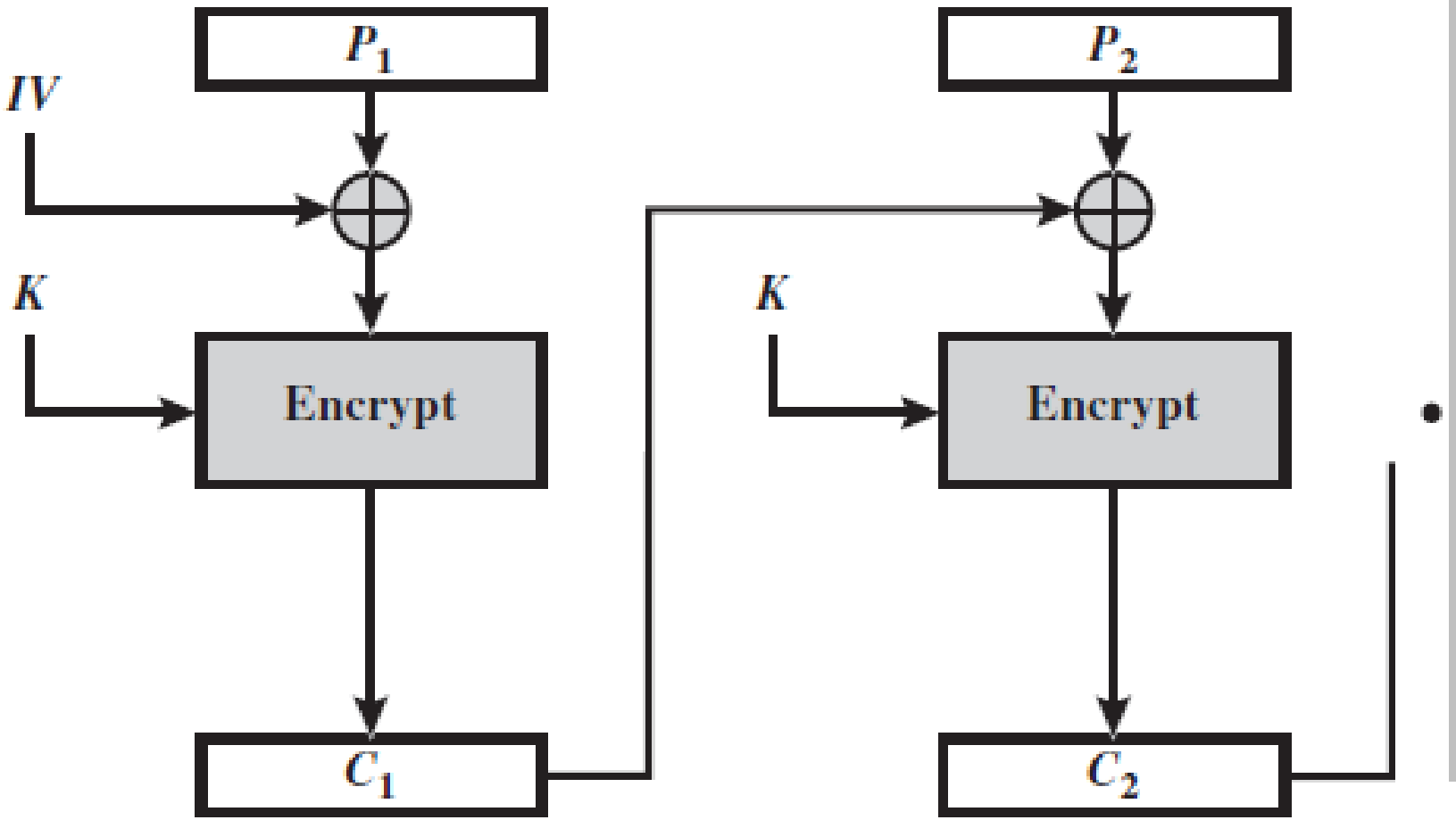
- Each block of plaintext is coded independently. Not very secure for long segments of plaintext, especially plaintext containing repetitive information.
- Used primarily for secure transmission of short pieces of information, such as an encryption key.
- Another shortcoming of ECB is that the length of the plaintext message must be integral multiple of the block size. When that condition is not met, the plaintext message must be padded appropriately.
- The rest of the modes discussed below provide enhanced security by making the ciphertext for any block a function of all the blocks seen previously.



ECB Encryption

CIPHER BLOCK CHAINING MODE (CBC)

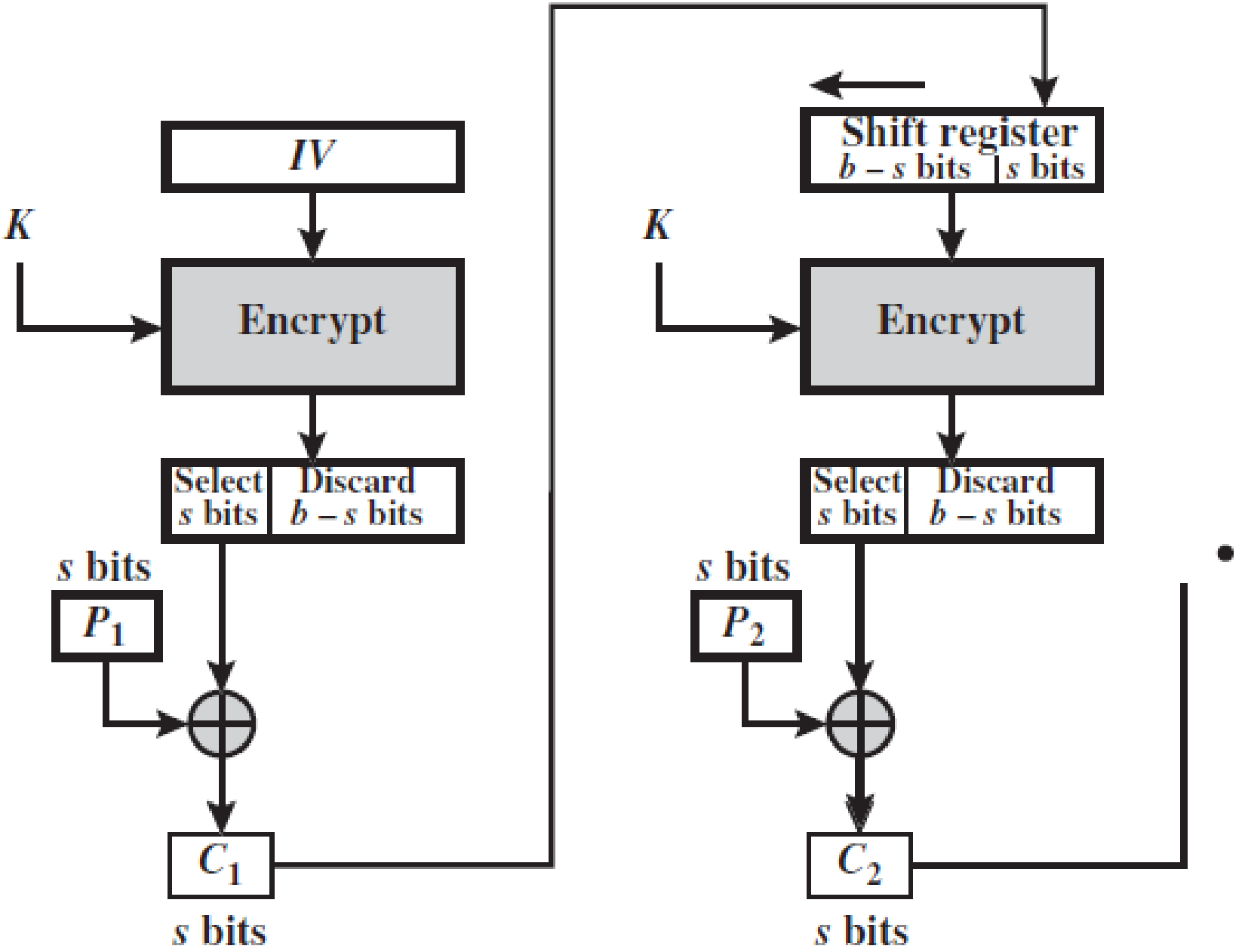
- The input to the encryption algorithm is the XOR of the next block of plaintext and the previous block of ciphertext. This is obviously ***more secure for long segments*** of plaintext.
- This mode also requires that length of the plaintext message be an integral multiple of the block size. When that condition is not satisfied, the message must be suitably padded.
- To get started, the chaining scheme obviously needs what is known as the initialization vector for the first invocation of the encryption algorithm.
- With this chaining scheme, the ciphertext block for any given plaintext block becomes a function of all the previous ciphertext blocks.



CBC Encryption

Cipher Feedback Mode (CFB)

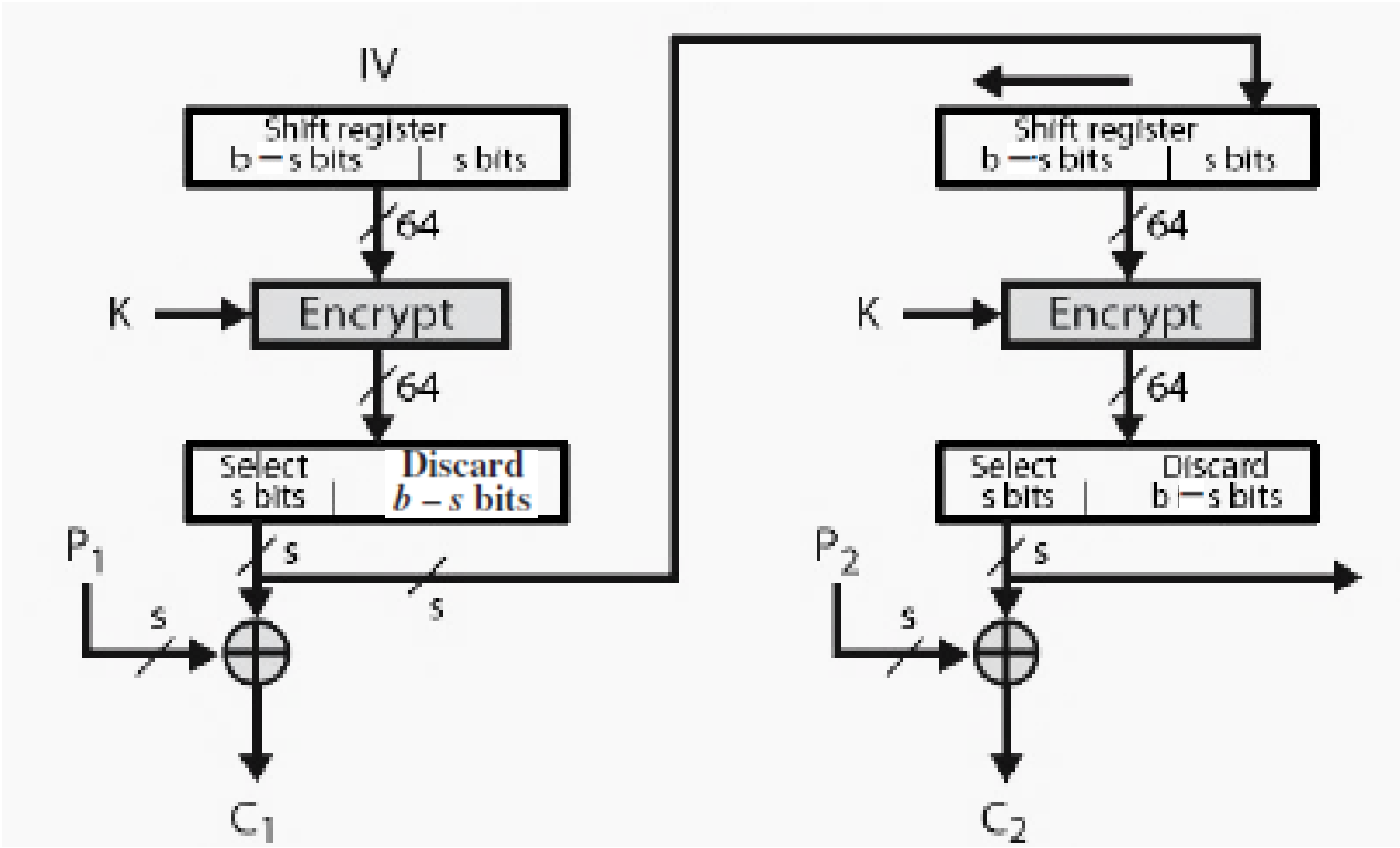
- Whereas the CBC mode uses all of the previous ciphertext block to compute the next ciphertext block, the CFB mode uses only a fraction thereof.
- Also, whereas in the CBC mode the encryption system digests b bits of plaintext at a time (where b is the block size used by the block cipher), now the encryption system digests only $s < b$ number of plaintext bits at a time even though the encryption algorithm itself carries out a b-bits to b-bits transformation. Since s can be any number, including one byte, that makes CFB suitable as a ***stream cipher***.
- CFB uses only the encryption algorithm in both encryption and decryption.



CFB Encryption

Output Feedback Mode (OFB)

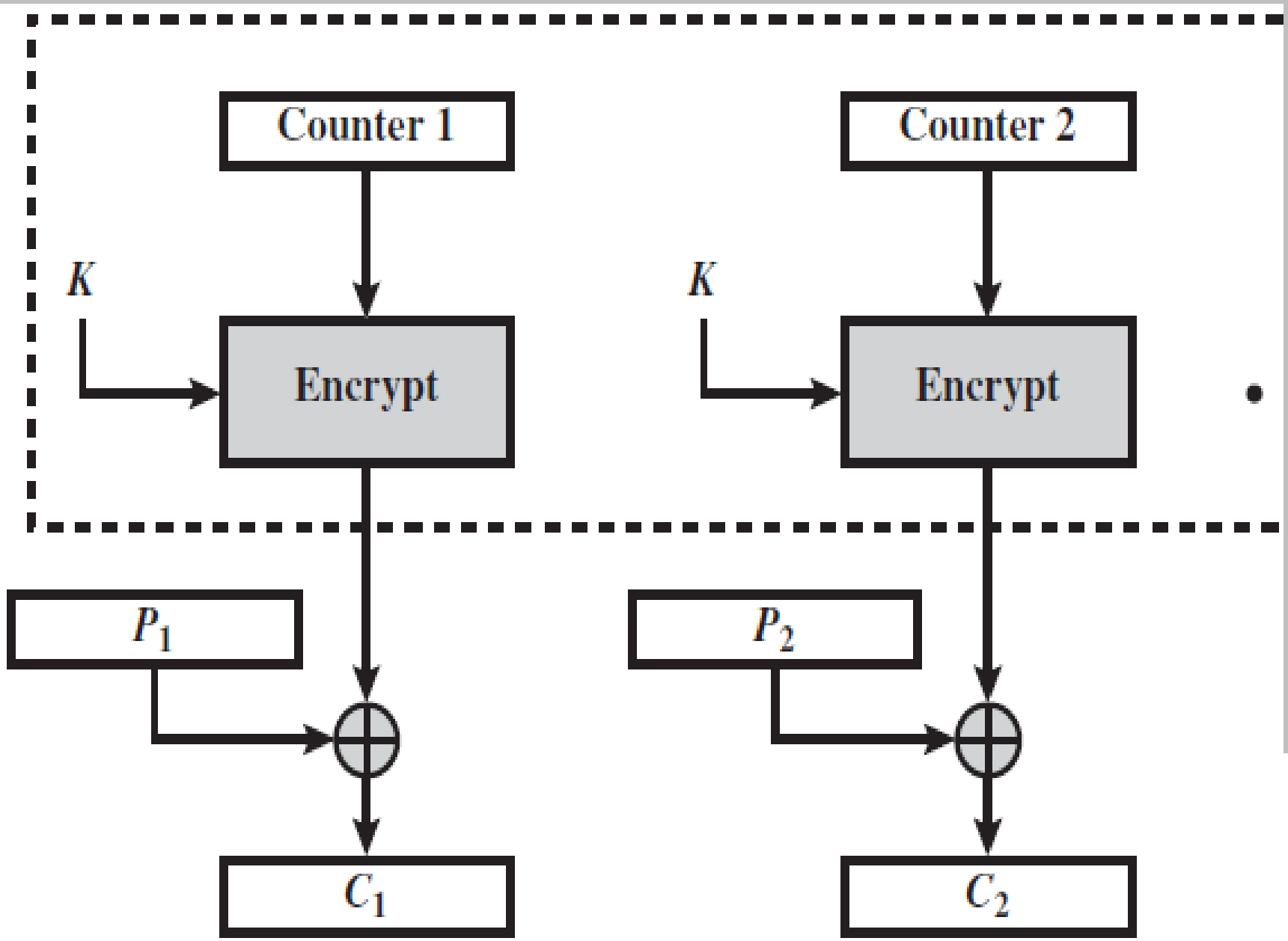
- The basic logic here is the same as in CFB, only the nature of what gets fed from stage to stage is different. In CFB, you feed $s < b$ number of ciphertext bits from the current stage into the b-bits to b-bits transformation carried out by the next-stage encryption. But in OFB, you feed s bits from the output of the transformation itself.
- This mode of operation is also suitable if you want to use a block cipher as a ***stream cipher***.
- Similarly to CFB, OFB uses only the encryption algorithm in both encryption and decryption.
- OFB is more resistant to transmission bit errors.



OFB Encryption

COUNTER MODE (CTR)

- Whereas the previous four modes for using a block cipher are intuitively plausible, this new mode at first seems strange and seemingly not secure. But it has been theoretically established that this mode is at least as secure as the other modes.
- As for CFB and OFB, an interesting property of this mode is that only the encryption algorithm is used at both the encryption end and at the decryption end.
- The basic idea consists of applying the encryption algorithm not to the plaintext directly, but to a b-bit number (and its increments modulo 2^b for successive blocks) that is chosen beforehand. The ciphertext consists of what is obtained by XORing the encryption of the number with a b-bit block of plaintext.



CTR Encryption

Advantages of CTR

1. Fast encryption and decryption. If memory is not a constraint, we can pre-compute the encryptions for as many counter values as needed. Then, at the transmit time, we only have to XOR the plaintext blocks with the pre-computed *b*-bit blocks. The same applies to fast decryption.
2. It has been shown that the CTR is at least as secure as the other four modes for using block ciphers.
3. Because there is no block-to-block feedback, the algorithm is highly amenable to implementation on parallel machines.
4. For the same reason, any block can be decrypted with random access.

Finally . . .

- **Acknowledgment:** These lecture notes are based on the textbook by William Stallings and notes prepared by Avinash Kak, Purdue University. My sincere thanks are devoted to them and to all other people who offered the material on the web.

- Students are advised to study and solve the problems and answer the questions in **Assignment-7**.

Chapter 8:
Public-Key Cryptography and RSA

4th Year- Course, CCSIT, UoA

Lecture Goals

1. To review public-key cryptography
2. To demonstrate that confidentiality and sender-authentication can be achieved simultaneously with public-key cryptography
3. To review the Rivest-Shamir-Adleman (RSA) algorithm for public-key cryptography
4. To discuss the security of RSA

Public-key Cryptography

- Public-key cryptography is also known as asymmetric-key cryptography.
- Encryption and decryption is carried out using ***two different keys.*** The two keys in such a key pair are referred to as the public key and the private key.
- This solves one of the most vexing problems associated with symmetric-key cryptography — the problem of key distribution.
- With public key cryptography, all parties interested in secure communications can publish their public keys.

Providing Confidentiality

- Public-key cryptography can be used to provide confidentiality (or secrecy) for a message.
- Party A, if wanting to communicate confidentially with party B, can encrypt a message using B's publicly available key. Such a communication would only be decipherable by B as only B would have access to the corresponding private key.
- This is illustrated in he next Figure.
- The notation here is that; A 's public and private keys are designated PU_a and PR_a. B 's public and private keys are designated PU_b and PR_b, respectively.

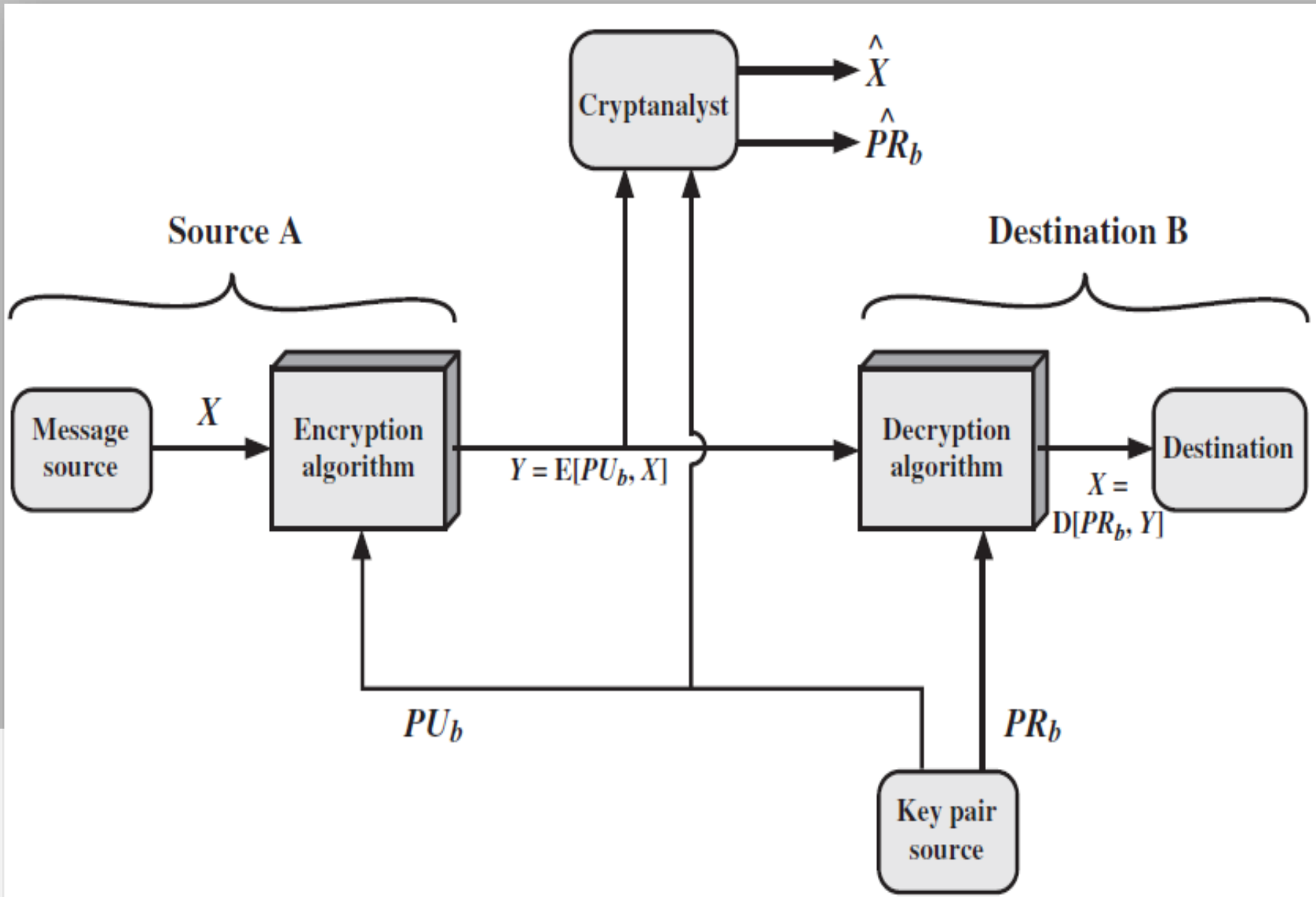


Figure 9.2 Public-Key Cryptosystem: Secrecy

Providing Authentication

- Public-key cryptography can also be used to provide authentication.
- Party A, if wanting to send an authenticated message to party B, would encrypt the message with A's own private key.
- Since this message would only be decipherable with A's public key, that would establish the authenticity of the message — meaning that A was indeed the source of the message.
- This is illustrated in the next Figure.

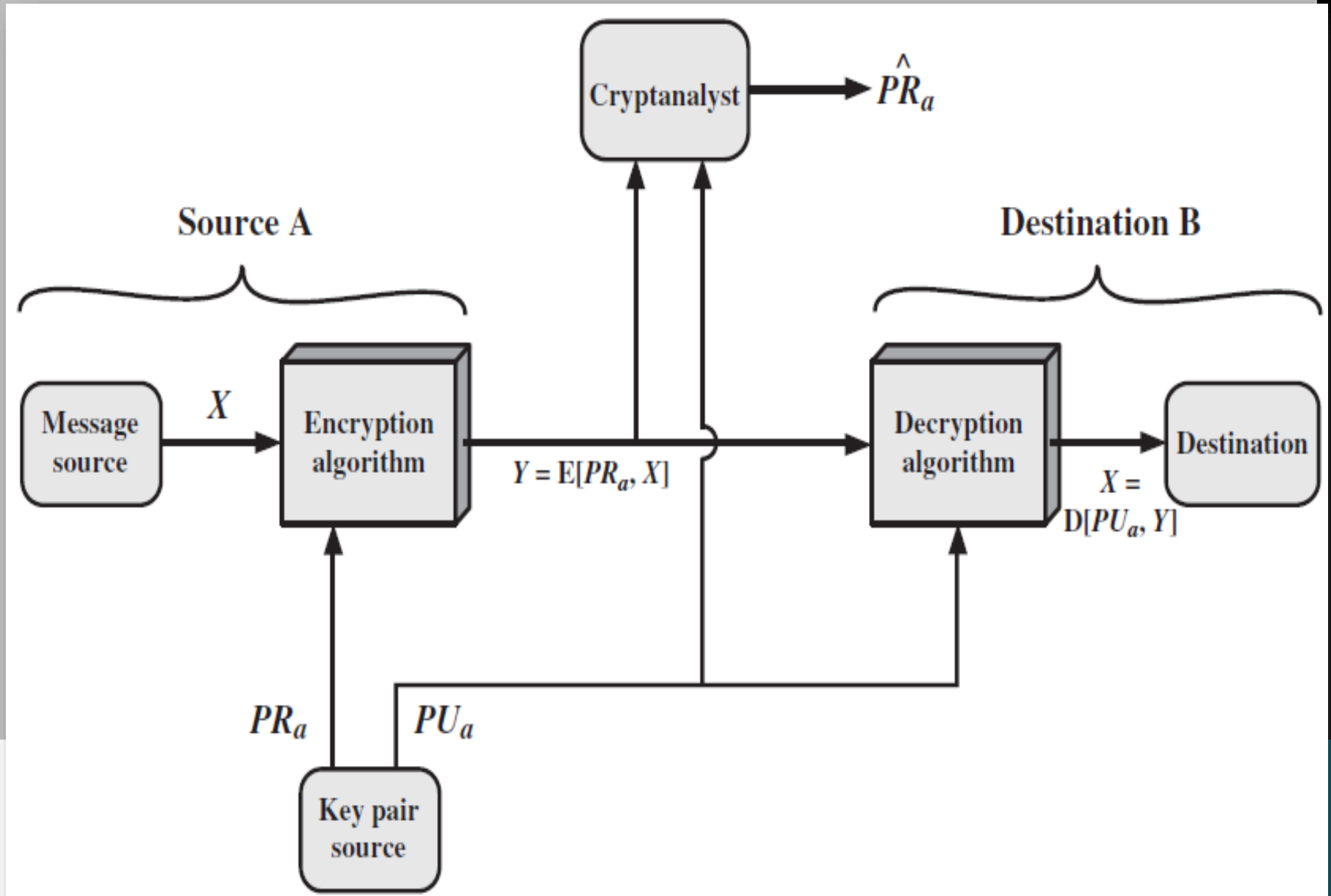


Figure 9.3 Public-Key Cryptosystem: Authentication

Both Confidentiality and Authentication (1)

- ✓ The communication link in the next Figure shows how public-key encryption can be used to provide both confidentiality and authentication at the same time.
- ✓ Note again that confidentiality means that we want to protect a message from eavesdroppers and authentication means that the recipient needs a guarantee as to the identity of the sender.
- ✓ Let's say that A wants to send a message M to B with both authentication and confidentiality. The processing steps undertaken by A to convert M into its encrypted form C that can be placed on the wire are:

$$C = E(P U_b, E(P R_a, M))$$

where $E(\ )$ stands for encryption.

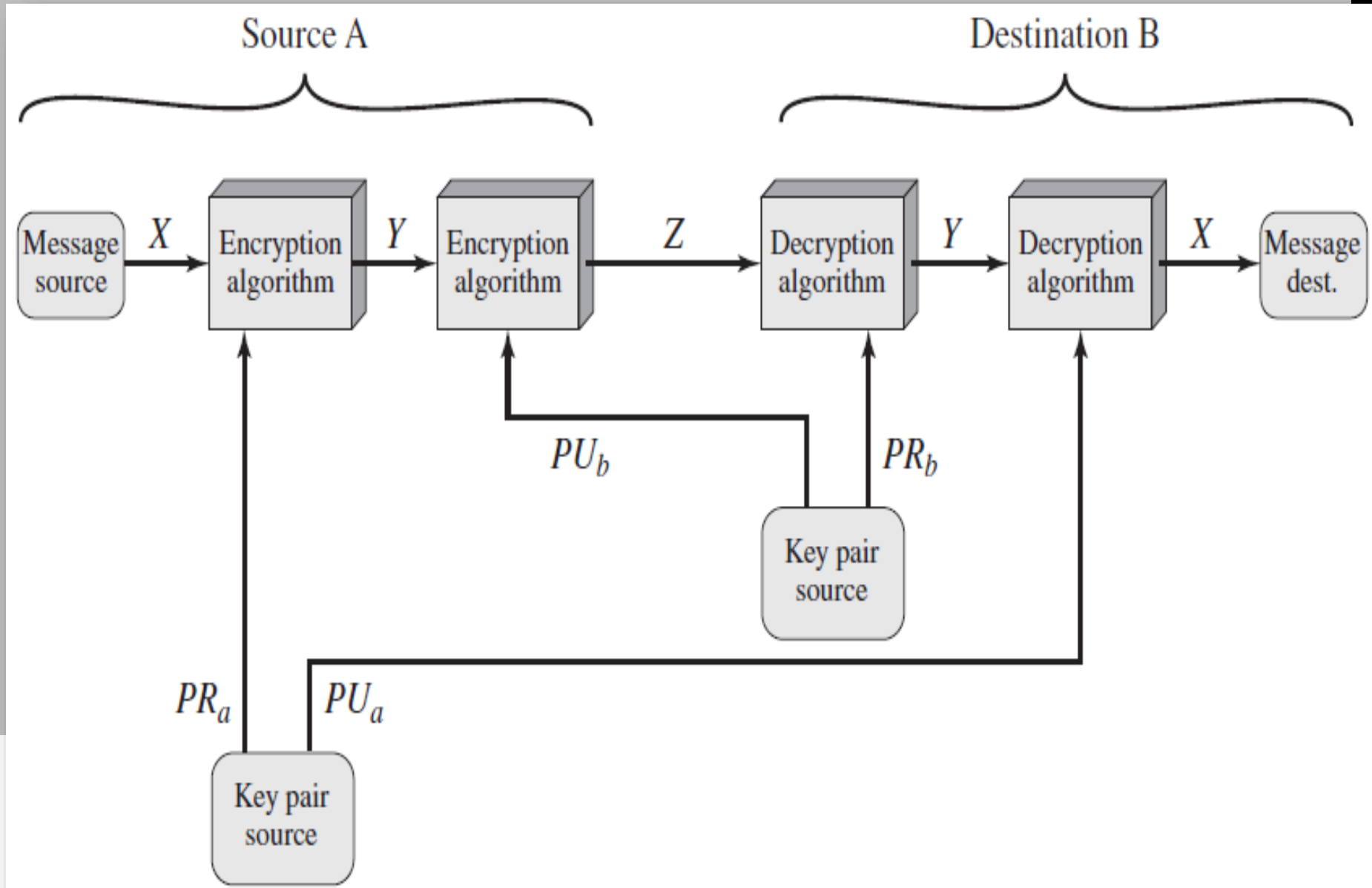


Figure 9.4 Public-Key Cryptosystem: Authentication and Secrecy

Both Confidentiality and Authentication (2)

- ✓ The processing steps undertaken by B to recover M from C are $M = D(P U_a, D(P R_b, C))$

  where $D( )$ stands for decryption.

- ✓ The sender A encrypting his/her message with its own private key $P R_a$ provides authentication. This step constitutes A putting his/her **digital signature** on the message.

- ✓ Instead of applying the private key to the entire message, a sender may also "sign" a message by applying his/her private key to just a **small block** of data that is derived from the message to be sent.

- ✓ The sender A further encrypting his/her message with the receiver's public key $P U_b$ provides **confidentiality**.

More on public-key cryptography

- The price paid for achieving confidentiality and authentication at the same time is that now the message must be processed four times in all for encryption/ decryption.
- Because of the greater computational overhead associated with public-key cryptosystems, public-key cryptography does not make obsolete the more traditional symmetric-key cryptography.
- However, it is generally agreed that public-key encryption is indispensable for key management and digital signature applications.

The RSA Cryptosystem

The RSA scheme is one of the most widely applied public-key systems. The basic computational steps for key generation in RSA are:

1. *Generate two different* ***primes*** p *and* q
2. *Calculate the modulus* $n = p \times q$
3. *Calculate the totient function* $\phi(n) = (p - 1) \times (q - 1)$
4. *Select integer* e *such that* $1 < e < \phi(n)$, $gcd(\phi(n), e) = 1$
5. *Calculate* d *such that* $d = e^{-1} \bmod \phi(n)$
6. *Public Key* $= [e, n]$
7. *Private Key* $= [d, n]$

The RSA Algorithm (1)

- Now suppose we are given an integer M, $M < n$, that represents our message, then we can transform M into another integer C that will represent our ciphertext by the following ***modular exponentiation***:

$$C = M^e \bmod n \qquad \textbf{(encryption)}$$

- As you will soon see, we can recover M back from C by the following modulo operation

$$M = C^d \bmod n \qquad \textbf{(decryption)}$$

- An individual who wishes to receive messages confidentially will use the pair of integers $\{e, n\}$ has his/her public key.
- At the same time, this individual can use the pair of integers $\{d, n\}$ as the private key.

The RSA Algorithm (2)

- RSA could be used as a block cipher for the encryption of the message. The block size would equal the number of bits required to represent the modulus n .
- If the modulus required, say, 1024 bits for its representation, message encryption would be based on 1024-bit blocks. Every plaintext block can now be thought of as an integer M of value

$$0 \leq M \leq 2^{1024} - 1.$$

- If this integer is expressed in decimal form, its value would be less than 10^{309}. In other words, the message integer M will have 309 decimal digits for each block of the plaintext.

How to choose the modulus for RSA algorithm?

- The modulus n must be selected in such a manner that the following is guaranteed:

$$(M^{e})^{d} \equiv M^{ed} \equiv M \pmod{n}$$

- It was shown by Rivest, Shamir, and Adleman that we have this guarantee when n is a product of two prime numbers: $n = p \times q$
- So that the cipher cannot be broken by an exhaustive search for the prime factors of the modulus n , it is important that both p and q be very large primes.
- Finding the prime factors of a large integer is computationally harder than determining its primality.

Choosing a value for e in RSA

- Recall that we want to raise the message integer M to the power e modulo n. This step is referred to as **modular exponentiation**.
- The mathematical requirement on e is that $gcd\,(e, \phi(n)) = 1$. Since $n = p \times q$, this requirement is equivalent to the two requirements **simultaneously**:

  1. $gcd\,(e, p - 1) = 1$
  2. $gcd\,(e, q - 1) = 1$

- Typical values for e can be 3, 17, and 65537 ($= 2^{16} + 1$). However, small values for e, such as 3, are considered **cryptographically insecure**.

Calculating d from e and n in RSA

- Once we have settled on a value for the encryption exponent e, the next step is to calculate the decryption exponent d from e and the modulus n.
- Recall that $d \times e \equiv 1 \pmod{\phi(n)}$. We can also write this as

$$d = e^{-1} \bmod \phi(n)$$

- Since d is the multiplicative inverse of e modulo $\phi(n)$, we can use the Extended Euclid's Algorithm for calculating d. Recall that we know the value for $\phi(n)$ since it is equal to $(p - 1) \times (q - 1)$.

Note that …

- This is the main source of security in the system — keeping p and q secret and therefore also keeping $\phi(n)$ secret.
- It is important to realize that knowing either will reveal the other.
- That is, if you know the factors p and q, you can calculate $\phi(n)$ by multiplying $p-1$ with $q-1$. And if you know $\phi(n)$ and n, you can calculate the factors p and q readily.

Simple Example for RSA Algorithm (1)

For this example, the keys were generated as follows.

1. Select two prime numbers, $p = 17$ *and* $q = 11$.

2. Calculate $n = pq = 17 * 11 = 187$.

3. Calculate $\phi(n) = (p - 1)(q - 1) = 16 * 10 = 160$.

4. Select *e such that e is relatively prime to* $\phi(n) = 160$ *and less than* $\phi(n)$*; we* choose $e = 7$.

5. Determine *d such that* $de \equiv 1 \pmod{160}$ *and* $d < 160$. *The correct value is* $d = 23$, *because* $23 * 7 = 161 = (1 * 160) + 1$; *d can be calculated using the* extended Euclid's algorithm.

Simple Example for RSA Algorithm (2)

The resulting keys are public key $PU = \{7, 187\}$ and private key $PR = \{23, 187\}$. The example shows the use of these keys for a plaintext input of $M = 88$. For encryption, we need to calculate $C = 88^7 \bmod 187$. Exploiting the properties of modular arithmetic, we can do this as follows.

$$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187$$

$$88^1 \bmod 187 = 88$$

$$88^2 \bmod 187 = 7744 \bmod 187 = 77$$

$$88^4 \bmod 187 = 59{,}969{,}536 \bmod 187 = 132$$

$$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894{,}432 \bmod 187 = 11$$

Simple Example for RSA Algorithm (3)

For decryption, we calculate $M = 11^{23} \bmod 187$:

$$11^{23} \bmod 187 = [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$$

$$11^1 \bmod 187 = 11$$

$$11^2 \bmod 187 = 121$$

$$11^4 \bmod 187 = 14{,}641 \bmod 187 = 55$$

$$11^8 \bmod 187 = 214{,}358{,}881 \bmod 187 = 33$$

$$11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 = 79{,}720{,}245 \bmod 187 = 88$$

A toy example for A block cipher application of RSA (1)

- For the sake of illustrating how you'd use RSA as a block cipher, consider a toy example in which the block size is 16.
- What that means is that we want to scan a disk file and convert each 16-bit plaintext block into a 16-bit ciphertext block.
- So our first job is to find a modulus n whose size is 16 bits. Recall that n must be a product of two primes p and q. Assuming that we want these two primes to be roughly the same size, let's allocate 8 bits to p and 8 bits to q.

A toy example for A block cipher application of RSA (2)

- If you carry out a Google search with a string like "first 1000 primes," you will discover that there exist many candidates for such primes. Let's select the following two: $p = 197$ and $q = 211$
- This gives us for the modulus $n = 197 \times 211 = 41567$.
- The bit pattern for the chosen p, q, and modulus n are:

bits of p : 1100 0101

bits of q : 1101 0011

bits of n : 1010 0010 0101 1111

A toy example for a block cipher application of RSA (3)

- As you can see, for a 16-bit block cipher, we have a modulus that requires 16 bits for its representation.
- Now let's try to select appropriate values for e and d.
- For e we want an integer that is relatively prime to the totient $\phi(n) = 196 \times 210 = 41160$. Such an e will also be relatively prime to 196 and 210, the totients of p and q respectively.
- It is preferable to select a small prime for e. However, $e = 3$ or $e = 5$ do not work since they are not relatively prime to 210.
- Let's choose $e = 17$.

A toy example for a block cipher application of RSA (4)

- With e set to 17, we must now choose d as the multiplicative inverse of e modulo 41160.
- Using the Bezout's identity based calculations, we can use the Extended Euclid's Algorithm to find that the multiplicative inverse of 17 modulo 41160 which is 26633. [You can verify this fact by showing $17 \times 26633 \bmod 41160 = 1$ on your calculator].
- Our (unsecure!) 16-bit block cipher based on RSA therefore has the following numbers for n, e, and d:

$$n = 41567, \quad e = 17, \quad \text{and} \quad d = 26633$$

Modular Exponentiation for Encryption and Decryption

- The message integer M is raised to the power e modulo n. That gives us the ciphertext integer C. Decryption consists of raising C to the power d modulo n.
- The exponentiation operation for encryption can be carried out efficiently by simply choosing an appropriate e. Note that the only condition on e is that it be coprime to $\phi(n)$.
- Modular exponentiation for decryption, meaning the calculation of $C^d \bmod n$, is an entirely different matter since we are not free to choose d. The value of d is determined completely by e and n.

The Security of RSA (1)

- A particular form of attack on RSA that has been a focus of considerable attention is the mathematical attack.
- The mathematical attack consists of figuring out the prime factors p and q of the modulus n .
- Obviously, knowing p and q , the attacker will be able to figure out the exponent d for decryption.
- Another way of stating the same as above would be that the attacker would try to figure out the totient $\phi(n)$ of the modulus n . But knowing $\phi(n)$ is equivalent to knowing the factors p and q.

The Security of RSA (2)

- The security of RSA encryption depends critically **on the difficulty of factoring large integers**.
- As integer factorization algorithms have become more and more powerful over the years, RSA cryptography has had to rely on increasingly larger values for the integer modulus and, therefore, increasingly longer encryption keys.
- These days you are unlikely to use keys shorter than 1024 bits for RSA. Some people recommend 2048 or even 4096 bit keys.

The Security of RSA (3)

- As you'd expect, the computational overhead of RSA encryption/decryption goes up as the size of the modulus integer increases.
- This makes RSA inappropriate for encryption/ decryption of actual message content for high data-rate communication links.
- However, RSA is ideal for the exchange of secret keys that can subsequently be used for the more traditional (and much faster) symmetric-key encryption and decryption of the message content.

Finally . . .

- **Acknowledgment:** These lecture notes are based on the textbook by William Stallings and notes prepared by Avinash Kak, Purdue University. My sincere thanks are devoted to them and to all other people who offered the material on the web.
- Students are advised to study and solve the problems and answer the questions in **Assignment-8**.
- Students are also advised to refer to the "**More Details on RSA**" attached lecture version.

Chapter 9:

Access Control I:

Authentication

4th Year Course- CCSIT, UoA

Aim of Lecture

- To show the importance of including access control mechanisms in information systems
- To study most important system authentication techniques

Access Control

We'll use the term access control as an umbrella for any security issues related to access of system resources. Thus, there are two areas of primary interest:

1. **Authentication:** Are you who you say you are?
   - Determine whether access is allowed or not
   - Authenticate human to machine (or possibly machine to machine)
2. **Authorization:** Are you allowed to do that?
   - Once you have access, what can you do?
   - Enforces limits on actions

- *Note: "access control" sometimes used as synonym for authorization*

Authentication

- We are specifically here interested in system authentication as the process of determining whether a user (or other entity) should be allowed access to a system. This can be based on:

1. **Something you know** (For example, a password)
2. **Something you have** (For example, a smartcard)
3. **Something you are** (For example, your fingerprint)

- *Another type of authentication problem arises when the authentication information must pass through a network (This has been explored previously in the course).*

Something You Know

- An ideal password is:
  1. something that you know,
  2. something that a computer can verify that you know, and
  3. something nobody else can guess—even with access to unlimited computing resources.
  - *However, in practice it's difficult to even come close to this ideal*
- Lots of things act as passwords!
  1. PIN
  2. Social security number
  3. Mother's maiden name
  4. Date of birth
  5. Name of your pet, etc.

Trouble with Passwords

- Passwords are one of the biggest practical problems facing security engineers today;
- If left to their own devices, users tend to select bad passwords, which makes password cracking surprisingly easy
- From a security perspective, a solution to the password problem would be to instead use randomly generated cryptographic keys.
  1. However, humans are incapable of securely storing high-quality cryptographic keys
  2. Humans have unacceptable speed and accuracy when performing cryptographic operations.
  3. cryptographic keys are also large, expensive to maintain, and difficult to manage

Why Passwords?

- Despite their security problems, passwords are so popular. So, why is “something you know” more popular than “something you have” and “something you are”?

  1. **Cost**: Passwords are free while smartcards and biometric devices cost money.

  2. **Convenience**: Easier for system administrator to reset password than to issue a new thumb(!!!)

Crypto Keys vs. Passwords

Crypto keys

- For example, assume using key size of 64 bits
- Then the space is 2^{64} keys
- It is easy to choose key at random…
- Then attacker must try about 2^{63} keys before reaching a solution (**brute force attack**)

Passwords

- For example, assume using passwords of 8 characters with 256 possible choices for each character.
- Then the space is $256^8 = 2^{64}$ passwords
- Users do not select passwords at random
- Attacker has far less than 2^{63} passwords to try (**dictionary attack**)

Dictionary Attack 1

- users don't select passwords at random, because users must remember their passwords.
- As a result, a user is far more likely to choose an 8-character dictionary word such as “computer” than, say, “kf&Yw!a[“
- So, for example, a carefully selected dictionary of 2^{20} (about 1,000,000 passwords) would likely give attacker a reasonable probability of cracking a given password.
- On the other hand, if attacker attempted to find a randomly generated 64-bit key by trying only 2^{20} possible keys, the chance of success would be a mere $2^{20}/2^{64} = 1/2^{44}$, or less than 1 in 17 trillion.
- The bottom line is that the non-randomness of password selection is at the root of the problems with passwords.

Good and Bad Passwords Examples

Bad passwords

- frank
- Fido
- Password
- incorrect
- Pikachu
- 102560
- AustinStamp

Good Passwords?

- jflej,43j-EmmL+y
- 09864376537263
- P0kem0N
- FSa7Yago
- 0nceuP0nAt1m8
- PokeGCTall150

Password *passphrase*

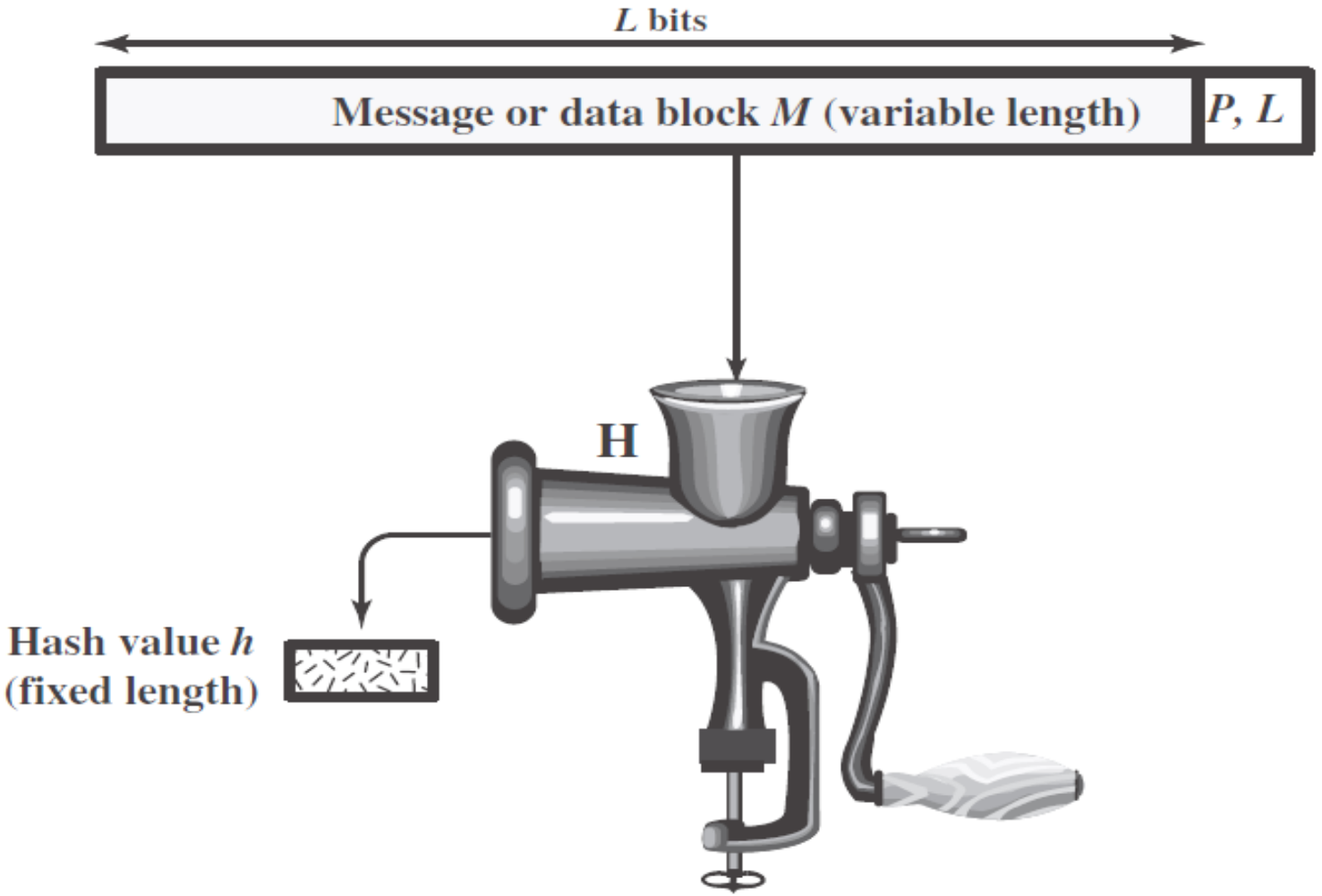
- The password *FSa7Yago* might appear to reside in the difficult to guess, but too difficult to remember category. However, there is a trick to help the user remember it—it's based on a passphrase.
- That is, *FSa7Yago* is derived from the phrase "*four score and seven years ago*."
- Consequently, this password should be relatively easy for Alice to remember, and yet relatively difficult for Trudy to guess.
- passphrases provide good option for password selection, since the resulting passwords are relatively difficult to crack yet easy to remember.

Password Retry

- Suppose system locks after 3 bad passwords. How long should it lock?
  1. 5 seconds?
  2. 5 minutes?
  3. Until administrator restores service?
- Five seconds might be insufficient to deter an automated attack. If it takes more than five seconds for attacker to make three password guesses for every user on the system, then he/she could simply cycle through all accounts, making three guesses on each
- On the other hand, five minutes might open the door to a denial of service attack, where attacker is able to lock accounts indefinitely by periodically making three password guesses on an account.
- The correct answer to this dilemma is not readily apparent.

Cryptographic Hash Function

- A **hash function *H*** accepts a variable-length block of data *M* as input and produces a fixed-size hash value *h = H(M).*
- A **"good"** hash function has the property that the results of applying the function to a large set of inputs will produce outputs that are evenly distributed and apparently random.
- The kind of hash function needed for security applications is referred to as a **cryptographic hash function.**
- **A cryptographic hash function** is an algorithm for which it is computationally infeasible to find either:
  1. a data object that maps to a pre-specified hash result (the one-way property) or
  2. two data objects that map to the same hash result (the collision-free property).



P, *L* = padding plus length field

Cryptographic Hash Function; h = H(M)

[the figure is from the textbook of Stallings]

Password Verification (1)

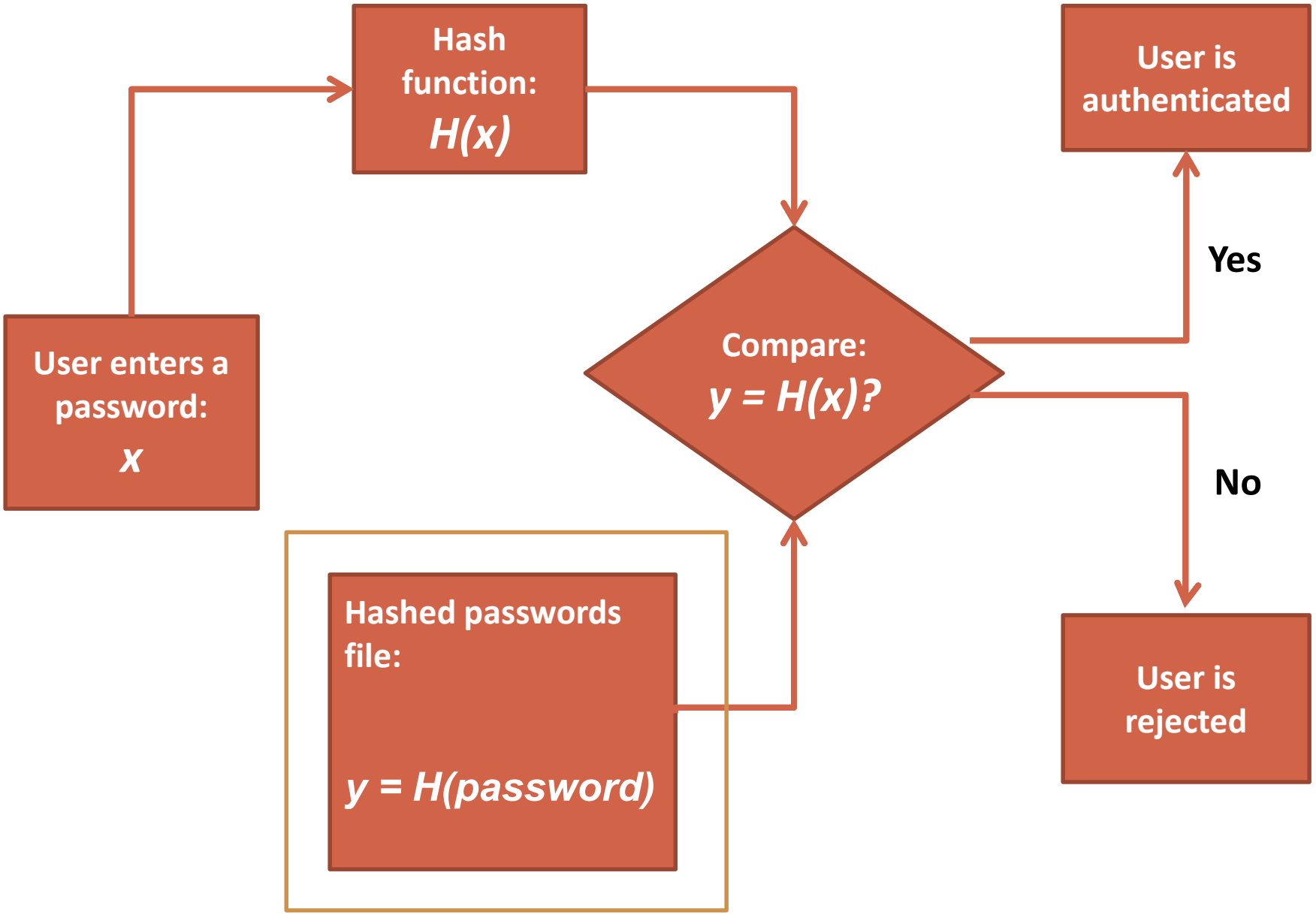
- For a computer to determine the validity of a password, it must have something to compare against.
- It is a bad idea to store **actual passwords** in a file for verification
- It might be tempting to **encrypt** the password file with a symmetric key. However, to verify passwords, the file must be decrypted, so the decryption key must be as accessible as the file itself. Consequently, if Trudy can steal the password file, she can probably steal the key as well.
- Consequently, encryption is of little value here..

Password Verification (2)

- One possible solution is **Hashing** passwords. This solution is based on the one-way property of cryptographic hash functions to protect the passwords
- So, instead of storing raw passwords in a file, it's more secure to store hashed passwords, i.e.

$$y = H(password)$$

- Then when someone claiming to be Alice enters a password x, it is hashed and compared to y, and if y = H(x) then the entered password is assumed to be correct and the user is authenticated (*See the next Figure*).


Hash function: H(x)
User is authenticated
User enters a password: x
Compare: y = H(x)?
Yes
No
Hashed passwords file:
y = H(password)
User is rejected

Password Verification (3)

- The advantage of hashing passwords is that if attacker obtains the password file, he/she does not obtain the actual passwords—instead he/she only has the hashed passwords.
- Of course, if attacker knows the hash value *y*, he/she can conduct a forward search attack by:
  1. guessing likely passwords *x*
  2. Check *y* until finding an *x* for which $y = H(x)$
  3. at which point he/she will have cracked the password.
- But at least attacker has work to do after he/she has obtained the password file.

Dictionary Attack 2

- Suppose attacker has a dictionary containing *N common passwords, say,*

$$d_0, d_1, d_2, \ldots, d_{N-1}$$

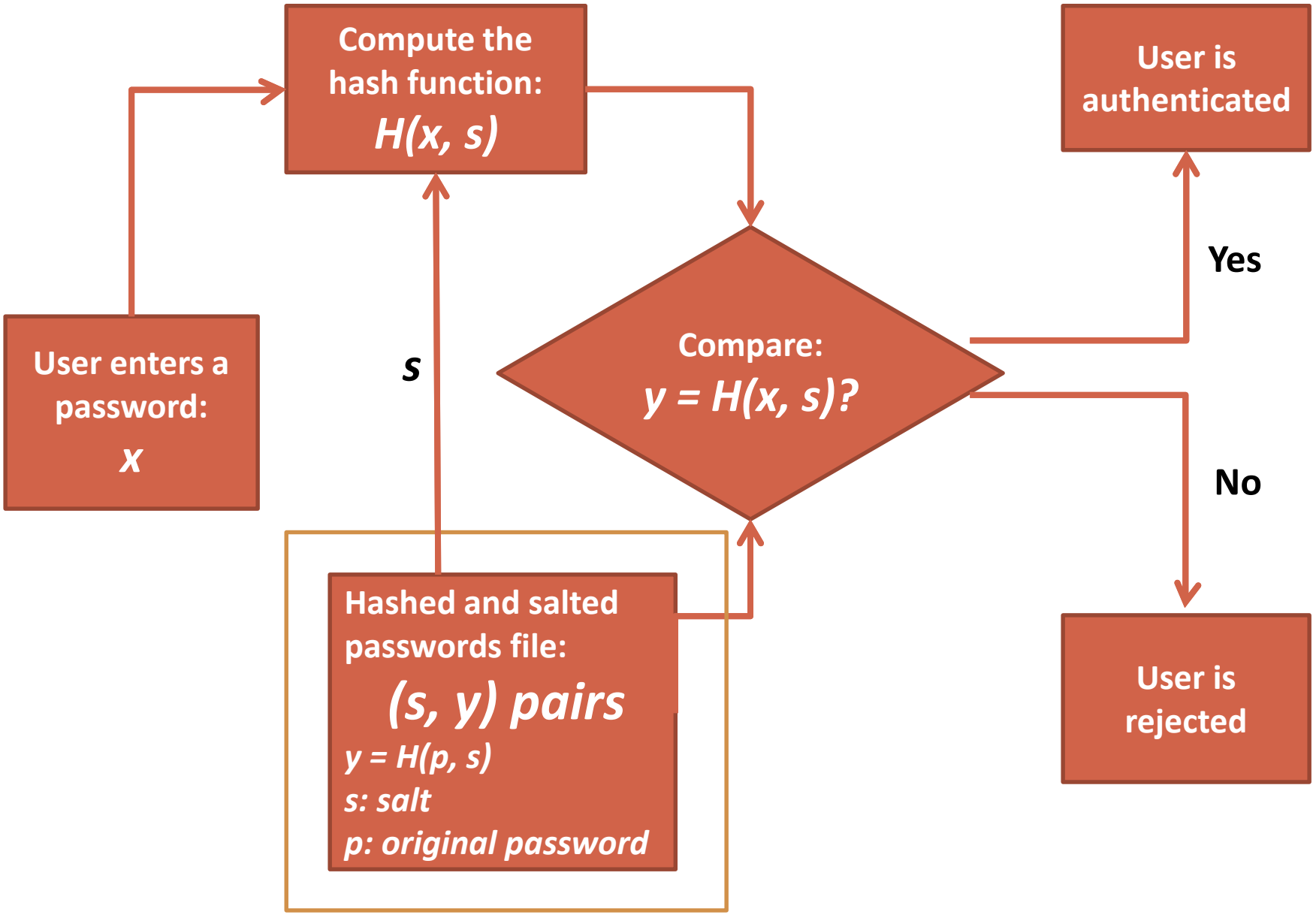
- Then he/she could pre-compute the hash of each password in the dictionary,

$$y_0 = H(d_0), \; y_1 = H(d_1), \ldots, \; y_{N-1} = H(d_{N-1})$$

- Suppose Trudy gets access to password file containing hashed passwords
  1. She only needs to compare hashes to her pre-computed dictionary
  2. After one-time work of computing hashes in dictionary, actual attack is trivial
- Can we prevent this forward search attack? Or at least make it more difficult?

Salt (1)

- We can prevent forward search attack by appending a non-secret random value, known as a **salt**, to each password before hashing.
- A password salt is analogous to the initialization vector, or IV, in, say, cipher block chaining (CBC) mode encryption. Whereas an IV is a non-secret value that causes identical plaintext blocks to encrypt to different ciphertext values, a salt is a non-secret value that causes identical password to hash to different values.
- Let p be a newly entered password. We generate a random salt value s and compute $y = H(p, s)$ and store the pair (s, y) in the password file.
- Note that the salt s is no more secret than the hash value.
- Now to verify an entered password x, we retrieve (s, y) from the password file, compute $H(x, s)$, and compare this result with the stored value y (*See the next Figure*).


Compute the hash function: H(x, s)
User is authenticated
User enters a password: x
s
Compare: y = H(x, s)?
Yes
No
Hashed and salted passwords file: (s, y) pairs
y = H(p, s)
s: salt
p: original password
User is rejected

Salt (2)

- Note that salted password verification is just as easy as it was in the unsalted case. But attacker's job has become much more difficult.
- Suppose Alice's password is hashed with salt value s_a and Bob's password is hashed with salt value s_b.
  - Then, to test Alice's password using her dictionary of common passwords, attacker must compute the hash of each word in his/her dictionary ***with salt value*** s_a,
  - but to attack Bob's password, attacker must re-compute the hashes ***using salt value*** s_b.
  - For a password file with N users, attacker's work has just increased by a factor of N. Consequently, a pre-computed file of hashed passwords is no longer useful for attacker.

Some Other Password Issues

1. Today, most users need multiple passwords, but users can't (or won't) remember a large number of passwords. This results in a significant amount of password reuse, and any password is only as secure as the least secure place it's used.
2. Social engineering is also a major concern with passwords. For example, if someone calls you, claiming to be a system administrator who needs your password to correct a problem with your account, would you give away your password?
3. Keystroke logging software and similar Spyware are also serious threats to password-based security
4. There are many available and popular password cracking tools. These tools come with preconfigured dictionaries, and it is easy to produce customized dictionaries

Something You Are: Biometrics

- There are many different types of biometrics including:
    1. fingerprints,
    2. iris scan,
    3. hand geometry,
    4. biometrics based on speech recognition,
    5. gait (walking) recognition,
    6. and digital doggie (odor recognition).

Why Biometrics?

- ✓ In the information security arena, biometrics are seen as a more secure alternative to passwords.
- ✓ For biometrics to be a practical replacement for passwords, cheap and reliable systems are needed.
- ✓ Today, usable biometric systems exist, including laptops using thumbprint authentication, palm print systems for secure entry into restricted facilities, the use of fingerprints to unlock car doors, and so on.
- ✓ But given the potential of biometrics and the well-known weaknesses of password-based authentication, biometrics are not really that popular yet.

Ideal Biometric Requirements

1. **Universal** — applies to (almost) everyone
   - *In reality, no biometric applies to everyone*
2. **Distinguishing** — distinguish with certainty
   - *In reality, cannot hope for 100% certainty*
3. **Permanent** — physical characteristic being measured never changes
   - *In reality, OK if it to remains valid for long time*
4. **Collectable** — easy to collect required data
   - *Depends on whether subjects are cooperative*
5. Also, safe, user-friendly, reliable, …

Biometric Identification vs. Authentication

- Biometrics are also applied in various identification problems
- **Identification** — Who goes there?
  - Compare **one-to-many**
  - Example: Fingerprint database
- **Authentication** — Are you who you say you are?
  - Compare **one-to-one**
  - Example: Thumbprint mouse
- Identification problem is more difficult because more "random" matches since more comparisons
- Here, we are (mostly) interested in authentication

Two phases of biometric systems

1. **Enrollment phase**; where subjects have their biometric information gathered and entered into a database:
   - Subject's biometric info put into database
   - OK if slow and repeated measurement needed
   - Must be very precise
2. **Recognition phase**; this occurs when the biometric detection system is used in practice to determine whether to authenticate the user or not:
   - Must be quick and simple
   - But must be reasonably accurate

Two Types of Biometric Errors 1

- There are two types of errors that can occur in biometric recognition:
  1. Suppose Bob poses as Alice and the system mistakenly authenticates Bob as Alice. The rate at which such misauthentication occurs is the **fraud rate**.
  2. Now suppose that Alice tries to authenticate as herself, but the system fails to authenticate her. The rate at which this type of error occurs is the **insult rate**.

Two Types of Biometric Errors 2

- For any biometric, we can decrease the **fraud** or **insult** rate at the expense of the other.
  1. For example, if we require a 99% voiceprint match, then we can obtain a low fraud rate, but the insult rate will be high, since a speaker's voice will naturally change slightly from time to time.
  2. On the other hand, if we set the threshold at a 30% voiceprint match, the fraud rate will likely be high, but the system will have a low insult rate.
- The **equal error rate** is the rate for which the fraud and insult rates are the same. This is a useful measure for comparing different biometric systems.

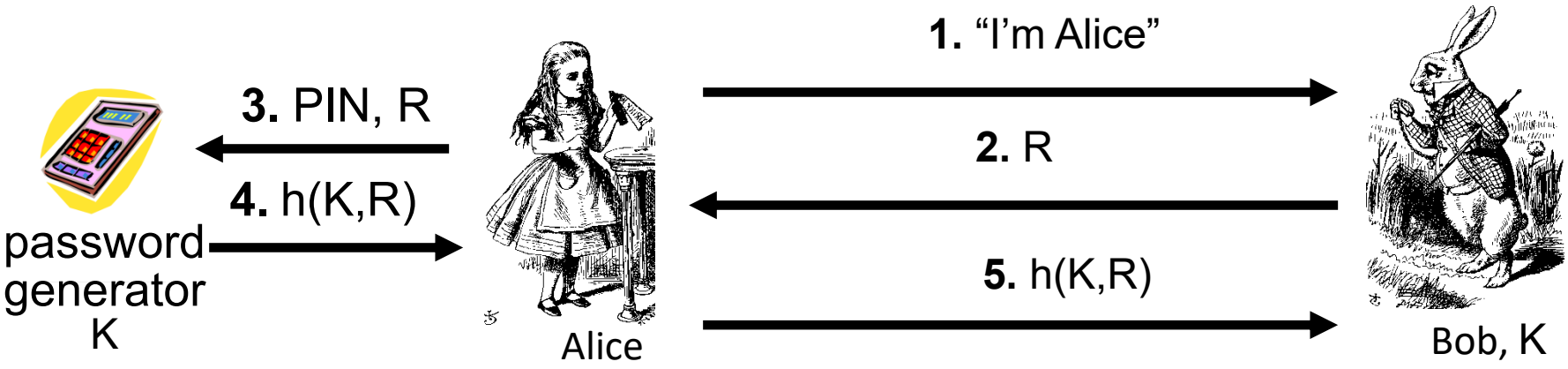
Equal Error Rate Comparison

- **Equal error rate** (**EER**): *fraud rate == insult rate*
- **Fingerprint** biometrics used in practice typically have EER ranging from about 10^{-3} to as high as 5%
- **Hand geometry** has EER of about 10^{-3}
- In theory, **iris scan** has EER of about 10^{-6} . But to achieve such spectacular results, the enrollment phase must be extremely accurate.
- Most biometrics much worse than fingerprint!
- *Biometrics are useful for authentication, but for identification, they still need more enhancement today*

Something You Have

- Something in your possession
- Examples include following…
  1. Car key
  2. Laptop computer (or MAC address)
  3. Password generator (next)
  4. ATM card, smartcard, etc.

Password Generator (1)



1. Alice introduces herself to Bob
2. Alice receives random "challenge" R from Bob
3. Alice enters PIN and R in password generator
4. Password generator hashes symmetric key K with R
5. Alice sends "response" h(K,R) back to Bob
6. Bob verifies response

➢ *Note: Alice **has** password generator and **knows** PIN*

Password Generator (2)

- For a challenge-response authentication scheme to work, Bob must be able to verify that Alice's response is correct. Thus, Bob and the password generator must both have access to the key *K, since the* password generator needs the key to compute the hash, and Bob needs the key to verify Alice's response.
- Alice accesses the key *K only indirectly*—by entering her PIN into the key generator.
- In fact, the password generator scheme above requires both "something you have" (the password generator) and "something you know" (the PIN).
- Any authentication method that requires two out of the three "somethings" is known as ***two-factor authentication.***

Finally . . .

- *Acknowledgment*: These lecture notes are based on the textbook of Mark Stamp and ppt slides offered by him. My sincere thanks are devoted to him and to all other people who offered the material on the web.

- Students are advised to study and solve the problems and answer the questions in **Assignment-9** (*the extra questions in this assignment are optional*).

Chapter 10:

Access Control II:

Authorization

4th Year Course- CCSIT, UoA

Aim of Lecture

- To study authorization as the part of access control concerned with restrictions on the actions of authenticated users.

- To explore CAPTCHAs

Authentication vs. Authorization

- **Authentication** — Are you who you say you are?
  - Restrictions on who (or what) can access system
- **Authorization** — Are you allowed to do that?
  - Restrictions on actions of authenticated users
- In its most basic form, authorization deals with the situation where we've already authenticated Alice and we want to enforce restrictions on what she is allowed to do.
- Note that while authentication is binary (either a user is authenticated or not), authorization can be a much more fine grained process

Lampson's Access Control Matrix (1)

- This matrix contains all of the relevant information needed by an operating system to make decisions about which users are allowed to do what with the various system resources.
- We'll define a ***subject*** as a user of a system (not necessarily a human user) and an ***object*** as a system resource.
- **Two fundamental constructs** in the field of authorization are access control lists, or **ACLs**, and capabilities, or **C-lists**.
- Both ACLs and C-lists are derived from Lampson's access control matrix, which has a row for every subject and a column for every object.
- Sensibly enough, the access allowed by subject *S* to object *O* is stored at the intersection of the row indexed by *S* and the column indexed by *O*.
- An example of an access control matrix appears in the next slide, where *x*, *r*, and *w* stand for execute, read, and write privileges, respectively.

Lampson's Access Control Matrix (2)

- **Subjects** (users) index the rows
- **Objects** (resources) index the columns

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	—	—
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

Performance for authorization operations (1)

- Since all subjects and all objects appear in the access control matrix, it contains all of the relevant information on which authorization decisions can be based. However, there is a practical issue in managing a large access control matrix.
- Access control matrix has all relevant information. But this could be (e.g.) 100's of users, 10,000's of resources. Then matrix has 1,000,000's of entries
- How to manage such a large matrix?
- *Note: We need to check this matrix before access to any resource by any user*
- How to make this more efficient/practical?

Performance for authorization operations (2)

- The solution is that the access control matrix can be partitioned into more manageable pieces. There are **two** obvious ways for this:

1. We could split the matrix into its columns and store each column with its corresponding object. Then, whenever an object is accessed, its column of the access control matrix would be consulted to see whether the operation is allowed. These columns are known as access control lists, or **ACL**s.
2. We could store the access control matrix by row, where each row is stored with its corresponding subject.. This approach is know as capabilities, or **C-lists**.

Access Control Lists (ACLs)

- ACL: store access control matrix by **column**
- Example: ACL for **insurance data** is in **blue**

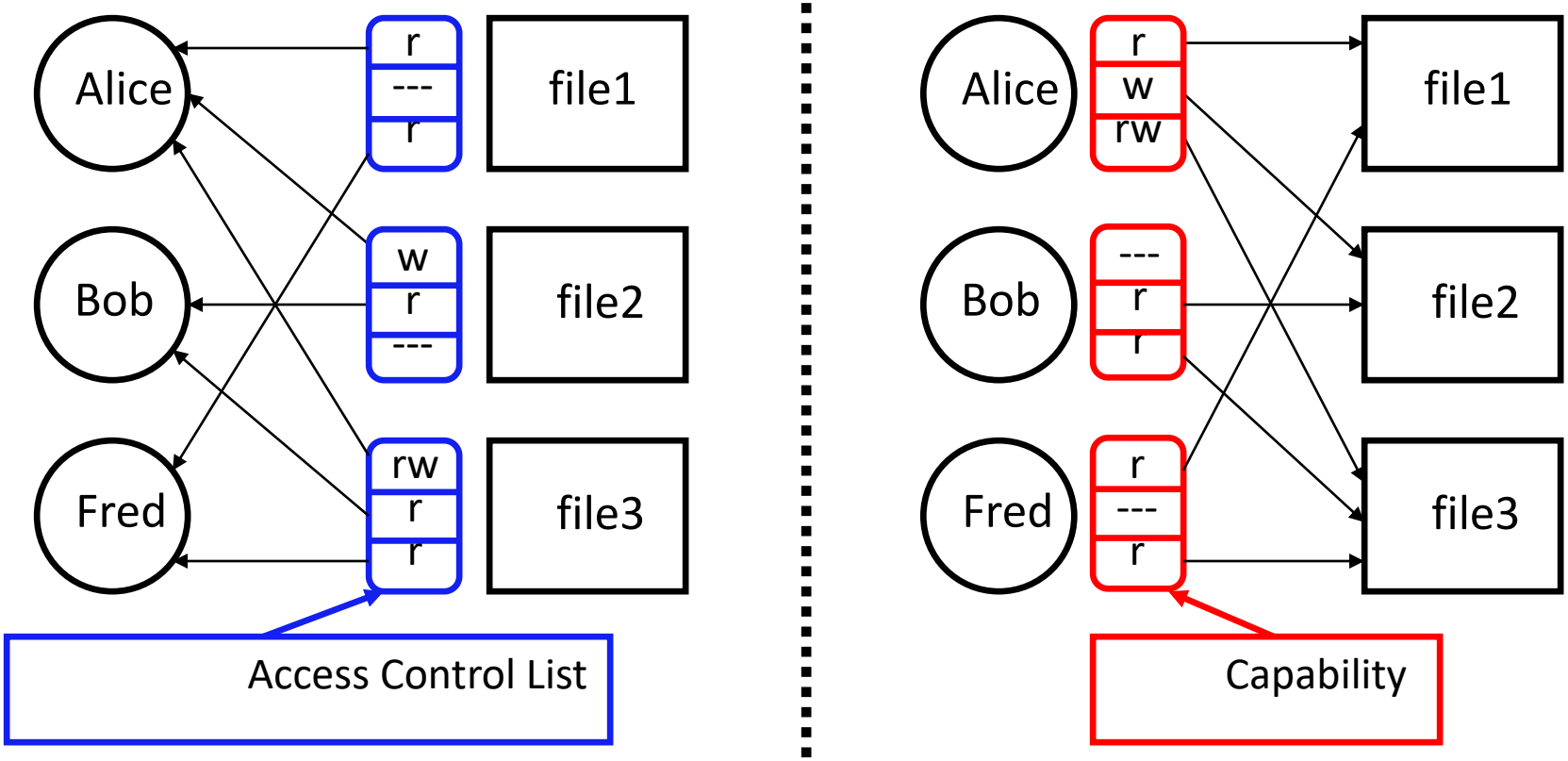
	OS	Accounting program	Accounting data	**Insurance data**	Payroll data
Bob	rx	rx	r	—	—
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

Capabilities (or C-Lists)

- Store access control matrix by **row**
- Example: Capability for **Alice** is in **red**

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	—	—
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

ACLs vs. Capabilities (1)



- Note that arrows point in opposite directions…
- with capabilities, the association between users and files is built into the system,
- while for an ACL-based system, a separate method for associating users to files is required

ACLs vs. Capabilities (2)

- Capabilities have several security advantages over ACLs and, for this reason.
- One potential security advantage of capabilities over ACLs is the so called ***confused deputy*** problem. To illustrate this problem, consider:

1. A system with two resources, a compiler and a file named BILL that contains critical billing information, and one user, Alice.
2. The compiler can write to any file.
3. Alice can invoke the compiler and she can provide a filename where debugging information will be written.
4. However, Alice is not allowed to write to the file BILL, since she might corrupt the billing information.

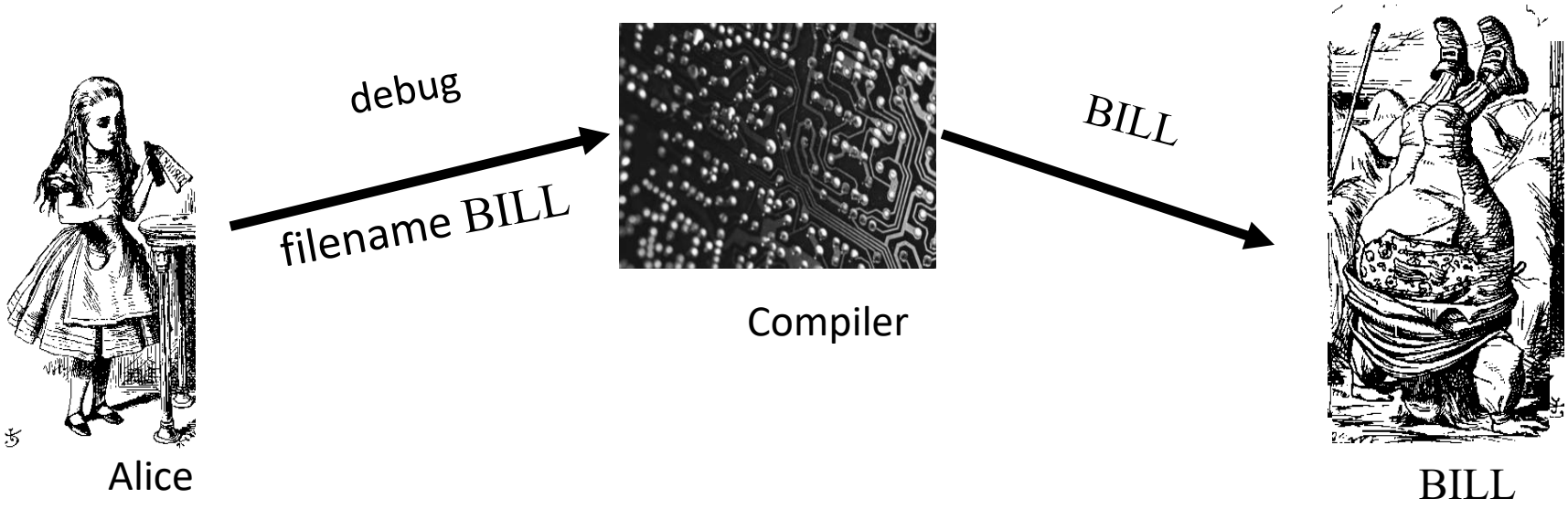
Confused Deputy

- Two resources: Compiler and BILL file (billing info)
- One user: Alica
- Compiler can write file BILL
- Alice can invoke compiler with a debug filename
- Alice not allowed to write to BILL

- Access control matrix

	Compiler	BILL
Alice	x	—
Compiler	rx	rw

ACL's and Confused Deputy



- Compiler is **deputy** acting on behalf of Alice
- Compiler is **confused:** Alice is not allowed to write BILL
- Compiler has confused its rights with Alice's

Capabilities and Confused Deputy

- Compiler acting for Alice is confused
- There has been a separation of **authority** from the **purpose** for which it is used
- With ACLs, it's more difficult to avoid the confused deputy.
- In contrast, with capabilities it's relatively easy to prevent this problem, ***since capabilities are easily delegated***, while ACLs are not.
- In a capabilities-based system, when Alice invokes the compiler, she can simply give her C-list to the compiler. The compiler then consults Alice's C-list when checking privileges before attempting to create the debug file

ACLs vs. Capabilities (3)

- **ACLs:**
  1. Good when users manage their own files
  2. Protection is data-oriented
  3. Easy to change rights to a resource
- **Capabilities:**
  1. Easy to delegate — avoid the confused deputy
  2. Easy to add/delete users
  3. More complex to implement and they have somewhat higher overhead
- *Despite their security advantage, many of the difficult issues inherent in distributed systems arise in the context of capabilities. Thus, ACLs are used in practice far more often than capabilities*

Turing Test

- This test was proposed by Alan Turing in 1950:
  1. Human asks questions to a human and a computer, without seeing either
  2. If questioner cannot distinguish human from computer, computer passes
- This is the **gold standard** in AI
- No computer can pass this today
- But some claim they are close to passing

CAPTCHA

- **CAPTCHA** (**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part) !!!
- Also known as **HIP** == **H**uman **I**nteractive **P**roof
- It is a test that a human can pass, but a computer can't pass with a probability better than guessing.
- This could be considered as a sort of inverse Turing test.
- CAPTCHA purpose is that only humans get access (not bots/computers). So, CAPTCHA is for **access control**
- The assumptions here are that the test is generated by a computer program and graded by a computer program, yet no computer can pass the test

CAPTCHA Uses?

Today, CAPTCHAs are used in a wide variety of applications. For example:

1. Free email services — spammers like to use bots to sign up for 1000s of email accounts. CAPTCHA employed so only humans get accounts
2. Sites that do not want to be automatically indexed by search engines. CAPTCHA would force human intervention

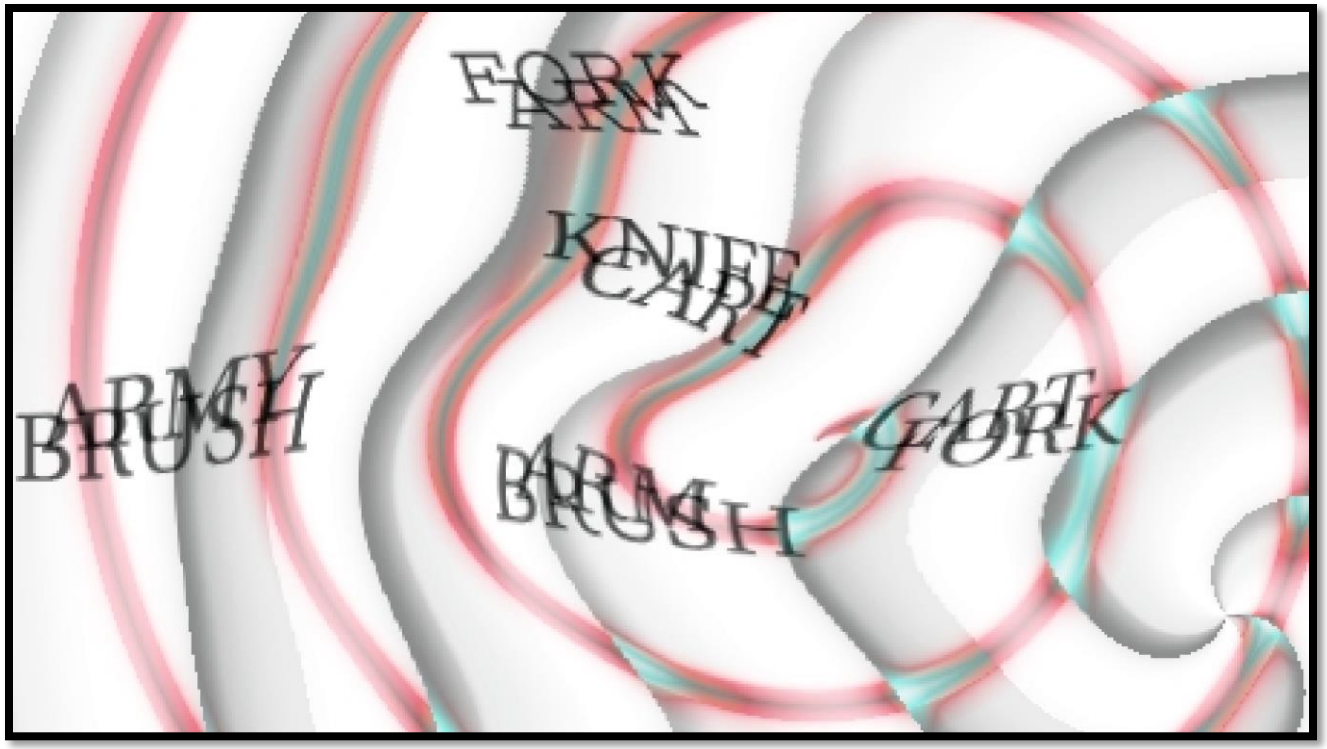
CAPTCHA: Rules of the Game

Requirements for CAPTCHA include:

1. Easy for most humans to pass
2. Difficult or impossible for machines to pass; e**ven with access to CAPTCHA software**
3. From attacker's perspective, the only unknown is a random number that is used to generate the specific CAPTCHA (Similar to Kerckhoffs' Principle)
4. Good to have different CAPTCHAs in case someone cannot pass one type. For example, many websites allow users to choose an *audio CAPTCHA* as an alternative to the usual *visual CAPTCHA*.

CAPTCHA Example 1

- In this example, a human might be asked to find three words that appear in the image.
- This is a relatively easy problem for humans and today it is also a fairly easy problem for computers to solve
- Much stronger CAPTCHAs exist



Fundamental visual CAPTCHA problems

Modern text-based CAPTCHAs are designed such that they require the simultaneous use of the following three separate abilities to correctly complete the task with any consistency:

1. ***Invariant recognition***; refers to the ability to recognize the large amount of variation in the shapes of letters. There are nearly an infinite number of versions for each character that a human brain can successfully identify. The same is not true for a computer.
2. ***Segmentation***, or the ability to separate one letter from another, is also made difficult in CAPTCHAs, as characters are crowded together with no white space in between.
3. ***Context*** is also critical. The CAPTCHA must be understood holistically to correctly identify each character. For example, in one segment of a CAPTCHA, a letter might look like an “m.” Only when the whole word is taken into context does it become clear that it is a “u” and an “n.”

CAPTCHA Example 2

- Each of these problems poses a significant challenge for a computer. The presence of all three at the same time is what makes CAPTCHAs difficult to solve
- It has been shown that computers are actually better than humans at solving all of the fundamental visual CAPTCHA problems, with one exception—the so-called ***segmentation problem***
- The segmentation problem is the problem of separating the letters from each other.
- Consequently, strong CAPTCHAs tend to look more like:

Types of CAPTCHAs

1. ***Text-based CAPTCHAs*** like previous examples. Here, we assume that attacker knows the set of possible words that could appear and he/she knows the general format of the image, as well as the types of distortions that can be applied. The only unknown for him/her is a random number that is used to select the word or words and to distort the resulting image
2. There are also ***audio (words or music) CAPTCHAs*** in which the audio is distorted in some way. The human ear is very good at removing such distortion, while automated methods are not so good
3. No text-based CAPTCHAs like ***image recognition CAPTCHAs*** which require users to identify simple objects in the images presented.

CAPTCHA's and AI

- Optical Character Recognition (OCR) is a challenging AI problem. Hardest part is the **segmentation problem.** Humans good at solving this problem
- Distorted sound also makes good CAPTCHA. Humans also good at solving this
- Hackers who break CAPTCHA have solved a hard AI problem. So, putting hacker's effort to good use!
- However, there are other ways to defeat CAPTCHAs. For example, the attackers may not play by the rules. Instead, the so-called *CAPTCHA farming* is possible, where humans are paid to solve CAPTCHAs (examples?!).

Finally . . .

- *Acknowledgment:* These lecture notes are based on the textbook by Mark Stamp and ppt slides offered by him. My sincere thanks are devoted to him and to all other people who offered the material on the web.

- Students are advised to study and solve the problems and answer the questions in Assignment-10.