SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

www.sathyabama.ac.in

SEC1316
Network Security

Dr.M.S.Godwin Premi

Professor, School of EEE

UNIT -II
Cryptography Basics

- Terminologies
- Cryptography
- Classification : based on
  - operation
  - number of keys used
  - Processing
- Crypt analysis
- Types - Classical Encryption
- Stream cipher
- Substitution Cipher

- Ceaser Cipher, Brute Force attack
- Vignere Cipher, One time pad
- Transposition Cipher
  - Simple row column Transfer
- Play Fair Cipher, 2X2 Hill cipher
- Rail fence Cipher
- Block Cipher
- Modes of operation
- DES, AES
- RSA algorithm

Course Outcome: Encrypt and decrypt using basic cryptographic algorithms

Some Basic Terminologies

- **Plaintext** : original message
- **Ciphertext** : coded message
- **Cipher** : algorithm for transforming plaintext to ciphertext
- **Key** : info used in cipher known only to sender/receiver

- **Encipher (encrypt)** : converting plaintext to ciphertext
- **Decipher (decrypt)** : recovering ciphertext from plaintext

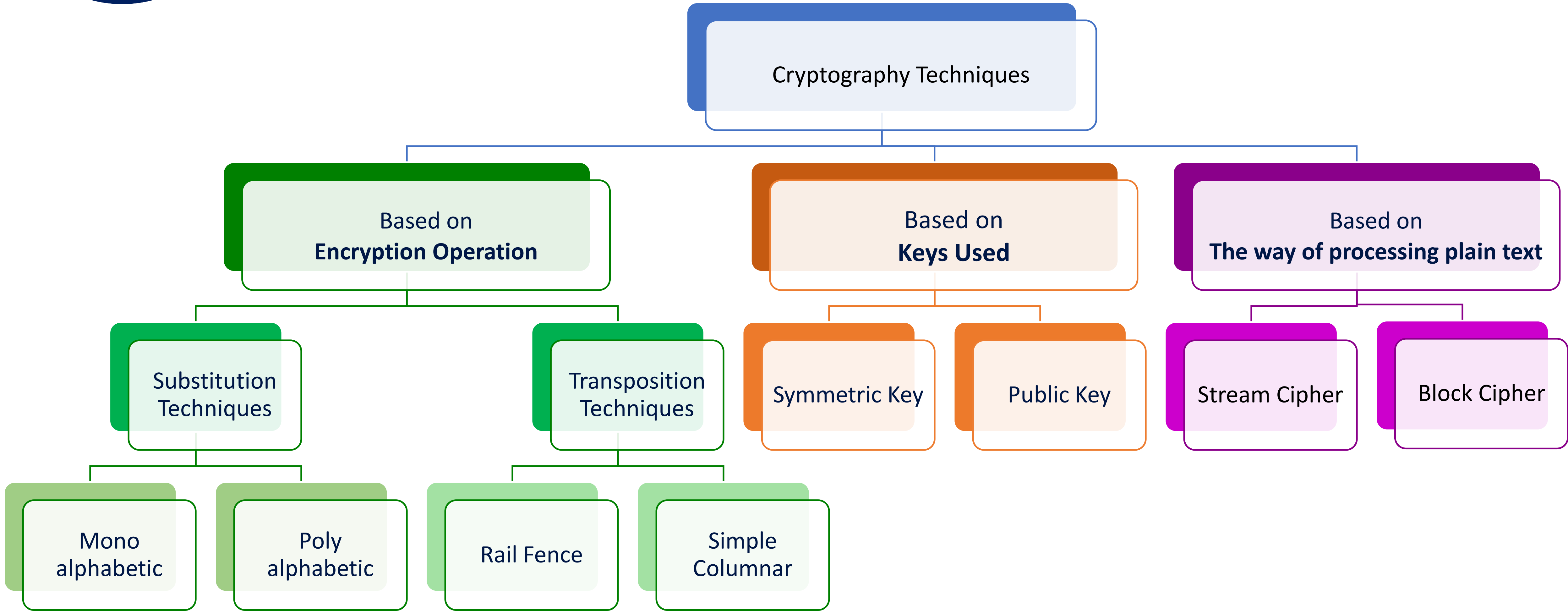
- **Cryptography** : study of encryption principles/methods
- **Cryptanalysis** : study of principles/ methods of deciphering ciphertext *without* knowing key
- **Cryptology** : field of both cryptography and cryptanalysis

Cryptography

Characterize cryptographic system by:

- **Type of encryption operations used**
  - ✓substitution / transposition / product
- **Number of keys used**
  - ✓single-key or private / two-key or public
- **Way in which plaintext is processed**
  - ✓block / stream

Cryptography - Classification



Cryptanalysis

- Cryptanalysis is the science of recovering the plaintext of a message without access to the key
- Successful cryptanalysis may recover the plaintext or the key
- The two basic categories of cryptanalysis are
  - Linear Cryptanalysis and
  - Differential cryptanalysis

Linear Cryptanalysis

- is a known plaintext attack, in which the attacker studies probabilistic linear relations known as linear approximations between parity bits of the plaintext, the Ciphertext and the secrete key

- the attacker obtains high probability approximations for the parity bit of the secrete key by analysing the parity bits of the known plaintexts and cipher texts

- By use of several techniques such as the auxiliary technique, the attacker can extend the attack to find more bits of the secret key

Differential Cryptanalysis

- Differential cryptanalysis can be described as a general form of cryptanalysis that is primarily applicable to block ciphers, cryptographic hash functions
- it entails a careful analysis of how differences in information input can affect the resulting difference at the output
- In block cipher, differential analysis can be described as a set of techniques for tracing differences through the network of transformation, discovering where the cipher

Cryptanalysis

Objective: to recover key / plain text

General Approaches

1. Cryptanalytic attack

- **ciphertext only**
  - ✓ only know algorithm & ciphertext, is statistical, know or can identify plaintext
- **known plaintext**
  - ✓ know/suspect plaintext & ciphertext
- **chosen plaintext**
  - ✓ select plaintext and obtain ciphertext
- **chosen ciphertext**
  - ✓ select ciphertext and obtain plaintext
- **chosen text**
  - ✓ select plaintext or ciphertext to en/decrypt

2. Brute force attack

- always possible to simply try every key
- most basic attack, proportional to key size
- assume either know / recognise plaintext

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

Classical Encryption Techniques

- Before computers, cryptography consisted of character-based algorithms popularly referred to as Classical encryption techniques
- Different cryptographic algorithms either substituted characters for one another or transposed characters with one another
- The better algorithms did both
- Things are more complex these days, but the philosophy remains the same. The primary change is that algorithms work on bits instead of characters
- Most good cryptographic algorithms still combine elements of substitution and transposition

Substitution Ciphers

- Where letters of plaintext are **replaced** by other letters or by numbers or symbols

  Or

  if plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with ciphertext bit patterns

- Monoalphabetic and Polyalphabetic
  - ✓ Monoalphabetic cipher is one in which each character of the plaintext is replaced with a corresponding character of ciphertext.
  - ✓ The cryptograms in newspapers are simple substitution ciphers
  - ✓ A polyalphabetic substitution cipher is made up of multiple simple substitution ciphers. For example1 there might be five different simple substitution cipher

Caesar Cipher

- ✓ **Earliest** known substitution cipher
- ✓ By **Julius Caesar**
- ✓ First used in **military** affairs
- ✓ **Replaces each letter by 3rd letter**

$$C = E(P) = (P + K) \bmod (26)$$

Example: (Encryption)

Plain Text : meet me after the toga party

```
                  m  e  e   t     m  e
P(Num)---------12  4  4  19   12  4
Key= 3 --------  3  3  3   3    3  3    +
C(Num)---------15  7  7  22   15  7
                  P  H  H   W     P  H
```

Cipher Text: PHHW PH DIWHU WKH WRJD SDUWB

P	Num	C
A	0	A
B	1	B
C	2	C
D	3	D
E	4	E
F	5	F
G	6	G
H	7	H
I	8	I
J	9	J
K	10	K
L	11	L
M	12	M
N	13	N
O	14	O
P	15	P
Q	16	Q
R	17	R
S	18	S
T	19	T
U	20	U
V	21	V
W	22	W
X	23	X
Y	24	Y
Z	25	Z

Caesar Cipher

$P = D(C) = (C - k) \bmod (26)$

Example: (Decryption)

Cipher Text: PHHW PH DIWHU WKH WRJD SDUWB

```
                P  H  H   W    P  H
C(Num)---------15  7  7  22   15  7
Key= 3 -------- 3  3  3   3    3  3   -
P(Num)---------12  4  4  19   12  4
                m  e  e   t    m  e
```

Plain Text : meet me after the toga party

P	Num	C
A	0	A
B	1	B
C	2	C
D	3	D
E	4	E
F	5	F
G	6	G
H	7	H
I	8	I
J	9	J
K	10	K
L	11	L
M	12	M
N	13	N
O	14	O
P	15	P
Q	16	Q
R	17	R
S	18	S
T	19	T
U	20	U
V	21	V
W	22	W
X	23	X
Y	24	Y
Z	25	Z

Cryptanalysis of Caesar Cipher

- ✓ **Brute Force Search**
- ✓ Key is not known and it can be any one value from 1 to 26
- ✓ Only have **26 possible ciphers**
  - ✓ A maps to A,B,..Z
- ✓ For a given ciphertext, just try all shifts of letters
- ✓ Do need to **recognize** when have plaintext

$P = D(C) = (C - k) \bmod (26)$

Example:

Break the ciphertext "GCUA VQ DTGCM“

K=1 -----> fbtz up csfbl

K=2 -----> **easy to break** ----- Plain Text

P	Num	C
A	0	A
B	1	B
C	2	C
D	3	D
E	4	E
F	5	F
G	6	G
H	7	H
I	8	I
J	9	J
K	10	K
L	11	L
M	12	M
N	13	N
O	14	O
P	15	P
Q	16	Q
R	17	R
S	18	S
T	19	T
U	20	U
V	21	V
W	22	W
X	23	X
Y	24	Y
Z	25	Z

Playfair Cipher

- Not even the large number of keys in a monoalphabetic cipher provides security
- one approach to improving security was to encrypt multiple letters
- The **Playfair Cipher** is an example
- Invented by **Charles Wheatstone in 1854**, but named after his friend **Baron Playfair**

Playfair Key Matrix

Constructing Key Matrix

Key matrix= 5X5 matrix of letters based on a keyword

- **Step 1:** Fill in matrix with letters of keyword from left to right; top to bottom (without duplicates)
- **Step 2:** Fill rest of matrix with other letters (Letters I & J count as one letter)

Example:

using the keyword "**MONARCHY**"

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

Playfair - Encryption

Rules:

1. Plaintext is encrypted **two letters** at a time
2. Repeating plain text **letters** that are in the same pair are **repeated** with a **filler** letter like ‘x’
3. Each plain text letter in a pair is **replaced** by the letter that lies **in its own row** and the **column occupied by the other** plain text letter

Example:

P = hello

he/ ll/o

he/ lx/lo

he ---- CF

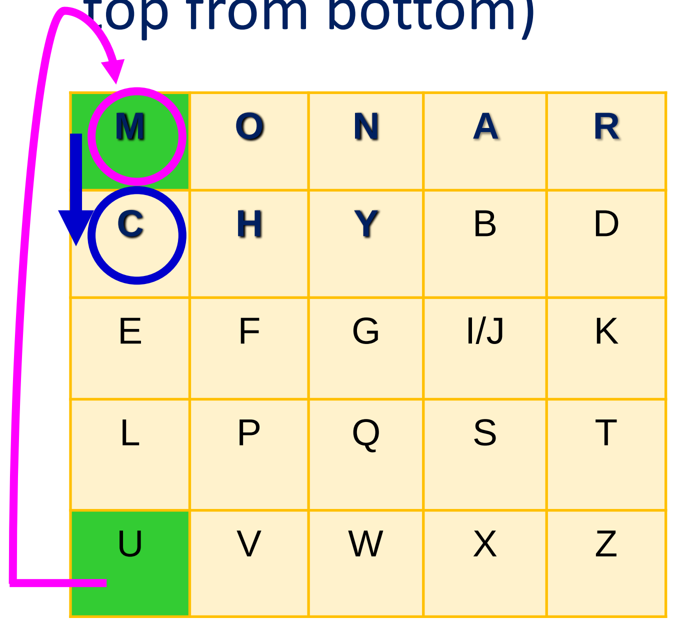
M	**O**	**N**	**A**	**R**
C	**H**	**Y**	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

Playfair - Encryption

Rules:

4. If both **letters** fall in the **same row**, replace each with letter to right (wrapping back to start from end)

5. If both **letters** fall in the **same column**, replace each with the letter below it (again wrapping to top from bottom)



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

Example:

P = mute

mu/ te

mu ---- CM

C=CM

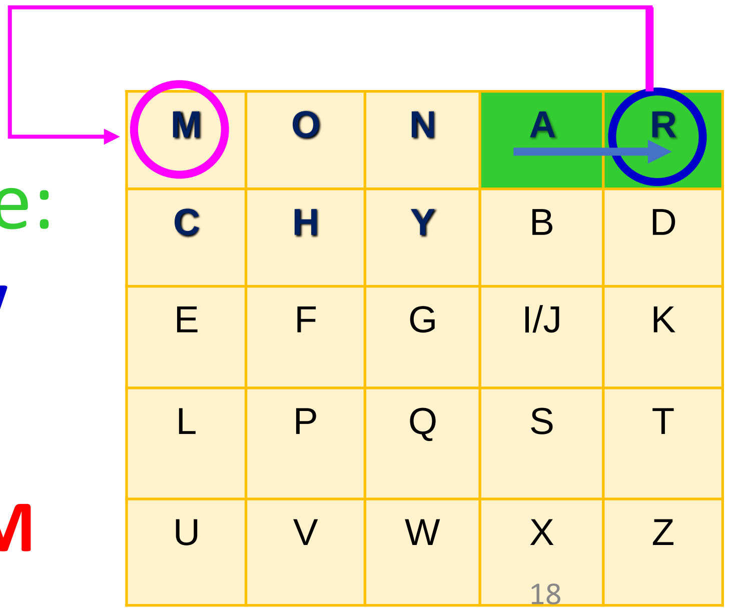
Example:

P = army

ar/ my

ar ---- RM

C=RM



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

Security of Playfair Cipher

- Security much improved over monoalphabetic
    1. since have 26 x 26 = 676 digrams
    2. would need a 676 entry frequency table to analyse (verses 26 for a monoalphabetic) and
    3. correspondingly more ciphertext
- Widely used for many years
    - eg. by US & British military in WW1
- It **can** be **broken**, given a few hundred letters
    - since still has much of plaintext structure

Hill Cipher

Encryption

$C = E(P) = (P * K) \bmod (26)$

Decryption

$P = D(C) = (C * K^{-1}) \bmod (26)$

- It takes 'm' successive plain text letters and substitutes for them in 'm' cipher text letters
- Substitution is determined by 'm' linear equations

$$C_1 = P_1K_1 + P_2K_2 + P_3K_3 \bmod 26$$

- $(C_1\ C_2\ C_3) = (P_1P_2P_3) \begin{pmatrix} K_{11} & K_{12} & K_{13} \\ K_{11} & K_{12} & K_{13} \\ K_{11} & K_{12} & K_{13} \end{pmatrix} \bmod 26$

Hill Cipher – 2x2

Encryption

$C = E(P) = (P * K) \bmod (26)$

Key size = 2

Example:

P = pay your money = pa / yy / ou / rm / on / ey

= 15 0 / 24 24 / 14 20 / 17 12 / 14 13 / 4 24

$$\text{Key} = \begin{bmatrix} 5 & 18 \\ 17 & 3 \end{bmatrix}$$

$$C = (15 \quad 0)\begin{bmatrix} 5 & 18 \\ 17 & 3 \end{bmatrix} = (15x5 + 0 \quad 15x18 + 0) = (75 \quad 270) \bmod 26 = (23 \ 10) \bmod 26$$

$$= XK$$

P	Num	C
A	0	A
B	1	B
C	2	C
D	3	D
E	4	E
F	5	F
G	6	G
H	7	H
I	8	I
J	9	J
K	10	K
L	11	L
M	12	M
N	13	N
O	14	O
P	15	P
Q	16	Q
R	17	R
S	18	S
T	19	T
U	20	U
V	21	V
W	22	W
X	23	X
Y	24	Y
Z	25	Z

Hill Cipher – 2x2

Decryption

$P = D(C) = (C * K^{-1}) \bmod (26)$

Example:

C= XK

$P = CK^{-1} \bmod 26$

$= (23\ \ 10)\begin{bmatrix} 15 & -12 \\ -7 & 25 \end{bmatrix}$

$= (23 \times 15 + 10 \times (-7)\ \ 23 \times (-12) + 10 \times 25)$

$= (345-70\ \ -276+250) = (275\ \ 0)$

$= (15\ 0)$ = p a

Key for Decryption:

$K^{-1} = \begin{bmatrix} 5 & 18 \\ 17 & 3 \end{bmatrix}^{-1} = \frac{1}{|K|}\text{adj}(K)$

$|K| = (5\times3 - 18\times17) = |-291| \bmod 26 = -5 = \mathbf{21 \bmod 26}$

$\text{Adj}(K) = \begin{bmatrix} 3 & -18 \\ -17 & 5 \end{bmatrix}$

$K^{-1} = 21^{-1}\begin{bmatrix} 3 & -18 \\ -17 & 5 \end{bmatrix} = 5\begin{bmatrix} 3 & -18 \\ -17 & 5 \end{bmatrix} = \begin{bmatrix} 15 & -90 \\ -85 & 25 \end{bmatrix}$

$= \begin{bmatrix} 15 & -12 \\ -7 & 25 \end{bmatrix}$

$21^{-1} \bmod 26 = 5 \bmod 26$

26= 1x21 + 5

21= 4x5 + 1

1= 21 – 4x5

= 21 – 4x[26-1x21]=5x21 -4x26=5

Monoalphabetic Cipher

- Rather than just shifting the alphabet could shuffle (jumble) the letters arbitrarily
- Each plaintext letter maps to a different random ciphertext letter
- Hence, key is 26 letters long

```
P: abcdefghijklmnopqrstuvwxyz
C: DKVQFIBJWPESCXHTMYAUOLRGZN

P: ifwewishtoreplaceletters
C: WIRFRWAJUHYFTSDVFSFUUFYA
```

Polyalphabetic Ciphers

Polyalphabetic substitution ciphers

- Improve security using multiple cipher alphabets
- Make cryptanalysis harder with more alphabets to guess and flatter frequency distribution
- Use a key to select which alphabet is used for each letter of the message
- Use each alphabet in turn
- Repeat from start after end of key is reached

Vigenère Cipher

- Simplest **polyalphabetic substitution cipher**
- Effectively use **multiple Caesar ciphers**
- Key is multiple letters long $K = k_1\ k_2\ \dots\ k_d$
- Decryption simply works in reverse

- The encryption of the original text is done using the **Vigenère square** or **Vigenère table**
- The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar ciphers

Vigenère Cipher

- Write the plaintext out
- Write the keyword repeated above it
- Use each key letter as a Caesar cipher key
- Encrypt the corresponding plaintext letter

Example: Encryption

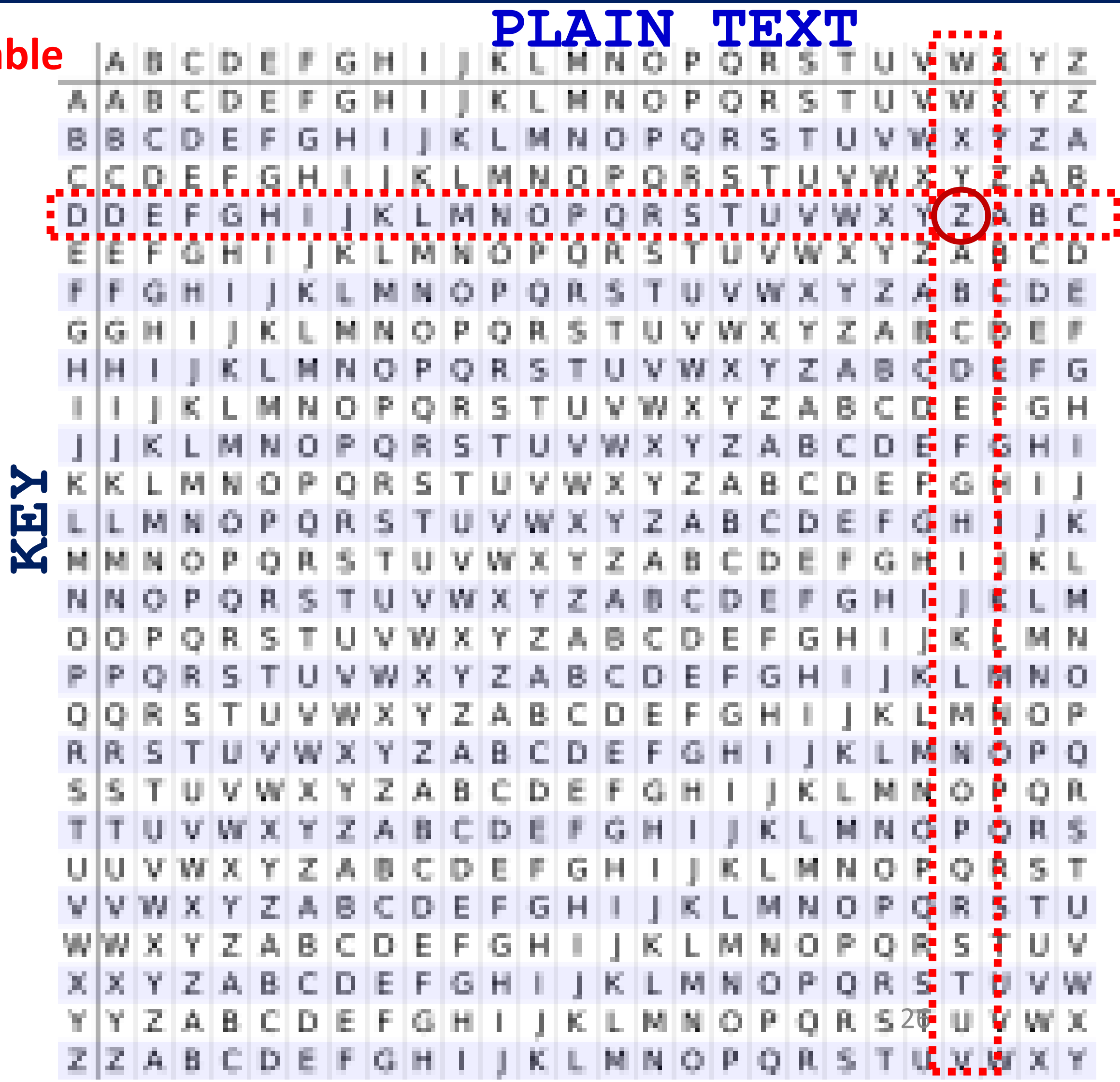
- using keyword ***deceptive***

P:We are discovered save yourself

K:deceptivedeceptivedeceptive

C=ZICVTWQNGRZGVTWAVZHCQYGLMGJ

Vigenère table


PLAIN TEXT
KEY

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

Vigenère Cipher

Example: Decryption

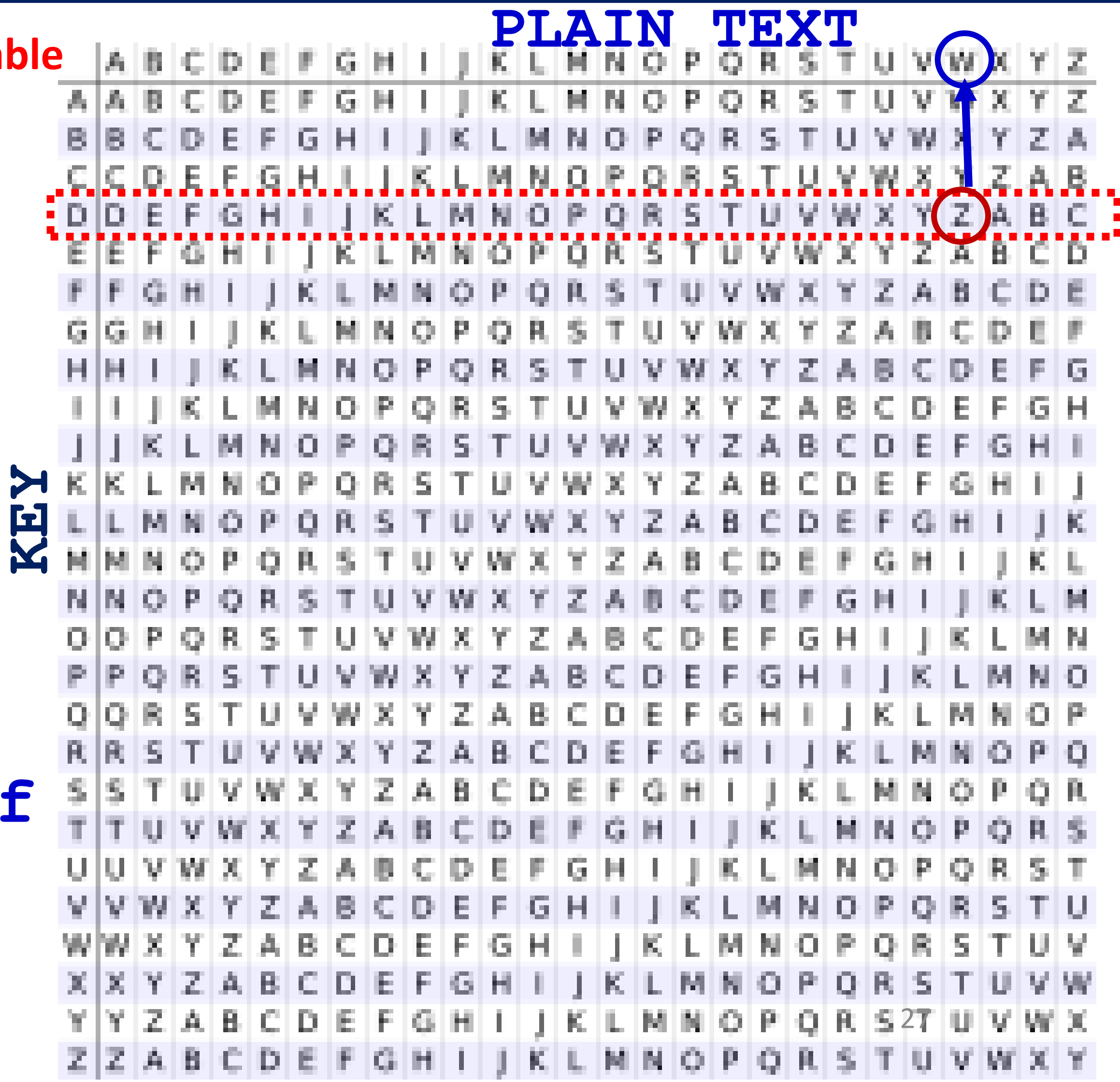
- using keyword ***deceptive***

C: ZICVTWQNGRZGVTWAVZHCQYGLMGJ

K: deceptivedeceptivedeceptive

P: *We are discovered save yourself*

Vigenère table


PLAIN TEXT
KEY

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

Security of Vigenère Ciphers

- Have multiple ciphertext letters for each plaintext letter
- Hence letter frequencies are obscured
- But not totally lost
- Start with letter frequencies
  - see if look monoalphabetic or not
- If not, then need to determine number of alphabets, since then can attach each

One -Time Pad (OTP)

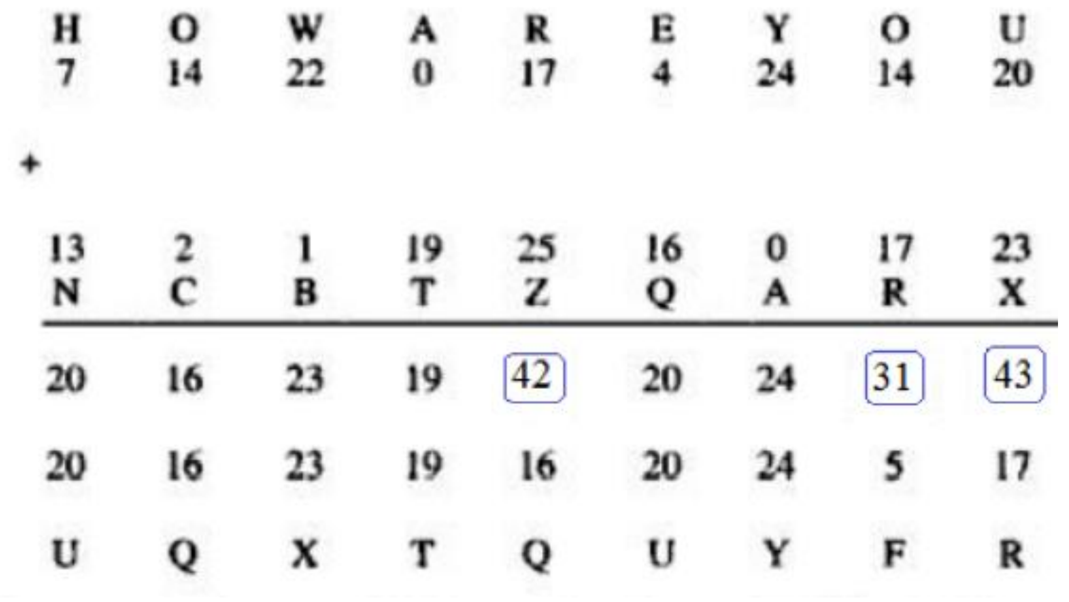
- If a truly random key as long as the message is used, the cipher will be secure
- called a One-Time pad
- is **unbreakable** since ciphertext bears no statistical relationship to the plaintext
- since for **any plaintext** & **any ciphertext** there exists a key mapping one to other
- can only use the key **once** though
- problems in generation & safe distribution of key

One -Time Pad (OTP)

- **Vernam Cipher**, also called as One-Time Pad,
- is implemented using a **random set of non-repeating characters** as the key
- The length of the key is always made equal to the length of the original plain text
- The most significant point here is that once a **key for encryption** is used, it is **never used again** for any other message (hence the name one-time)

One -Time Pad (OTP)

Example: Vernam Cipher algorithm to a plain text message **how are you** using a one-time pad **NCBTZQARX** to produce a cipher text message **UQXTQUYFR** as shown in Fig.



H	O	W	A	R	E	Y	O	U
7	14	22	0	17	4	24	14	20
+								
13	2	1	19	25	16	0	17	23
N	C	B	T	Z	Q	A	R	X
20	16	23	19	$\boxed{42}$	20	24	$\boxed{31}$	$\boxed{43}$
20	16	23	19	16	20	24	5	17
U	Q	X	T	Q	U	Y	F	R

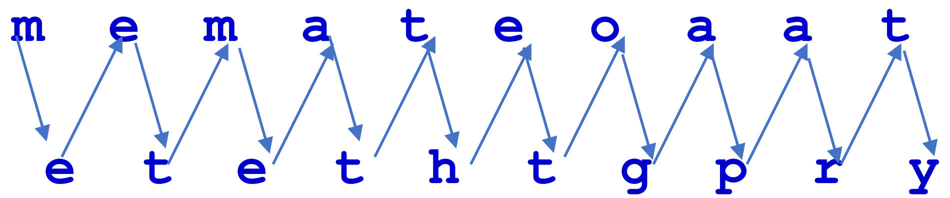
Transposition Ciphers

- **Transposition** or **Permutation** ciphers
- These hide the message by rearranging the letter order (only position of the letters are changed)
- Without altering the actual letters used
- Can recognise these since have the same frequency distribution as the original text

Rail Fence cipher

- write message letters out diagonally over a number of rows (zig-zag pattern)
- then read the cipher row by row
- Example (encryption): P= meet me at the toga party ; write it as:

P =



- Ciphertext (read row by row)

C = M E M A T E O A A T E T E T H T G P R Y

Rail Fence cipher

- Example (decryption):

C = M E M A T E O A A T E T E T H T G P R Y

Divide into two rows (Total=20 letters; 10 letters in each row)

M E M A T E O A A T / E T E T H T G P R Y

```
1st row ----   m  e  m  a  t  e  o  a  a  t
2nd row ----   e  t  e  t  h  t  g  p  r  y
```

- Plain text (read column by column)

  P= meet me at the toga party

Simple Columnar Transposition Ciphers

- a more complex transposition
- Write letters of message in rows over a specified number of columns
- then reorder the columns according to some key before reading the rows
- **Example:P : attack postponed until two am**

Key:	3	4	2	1	5	6	7
Plaintext:	a	t	t	a	c	k	p
	o	s	t	p	o	n	e
	d	u	n	t	i	l	t
	w	o	a	m	x	y	z

Filler letter

Ciphertext: APTM TTNA AODW TSUO COIX KNLY PETZ

Simple Columnar Transposition Ciphers

Decryption

Example:

Ciphertext: APTM TTNA AODW TSUO COIX KNLY PETZ

(Total=28 letters; divide by 7--> 4 letters in each column)

Key = 3421567

Column size=7

(enter column number)

1	2	3	4	5	6	7
APTM	**TTNA**	**AODW**	**TSUO**	**COIX**	**KNLY**	**PETZ**

Plain Text:

- write the 3rd column as first **(1st element of key)**
  Next 4th column; next 2nd column, and so on.
  Read row by row

A	T	T	A	C	K	P
O	S	T	P	O	N	E
D	U	N	T	I	L	T
W	O	A	M	X	Y	Z

Product Ciphers

- Ciphers using substitutions or transpositions are not secure because of language characteristics
- Hence consider using several ciphers in succession to make harder, but:
  - two substitutions make a more complex substitution
  - two transpositions make more complex transposition
  - but a substitution followed by a transposition makes a new much harder cipher
- this is a bridge **from classical to modern ciphers**

Stream and Block Ciphers

- **Stream Cipher:**
  - ✓In this the digital version of the plain text which is in binary form is processed for encryption as a stream of bits
  - ✓That means the bits are send serially one by one and processed to get the cipher text
- **Block cipher:**

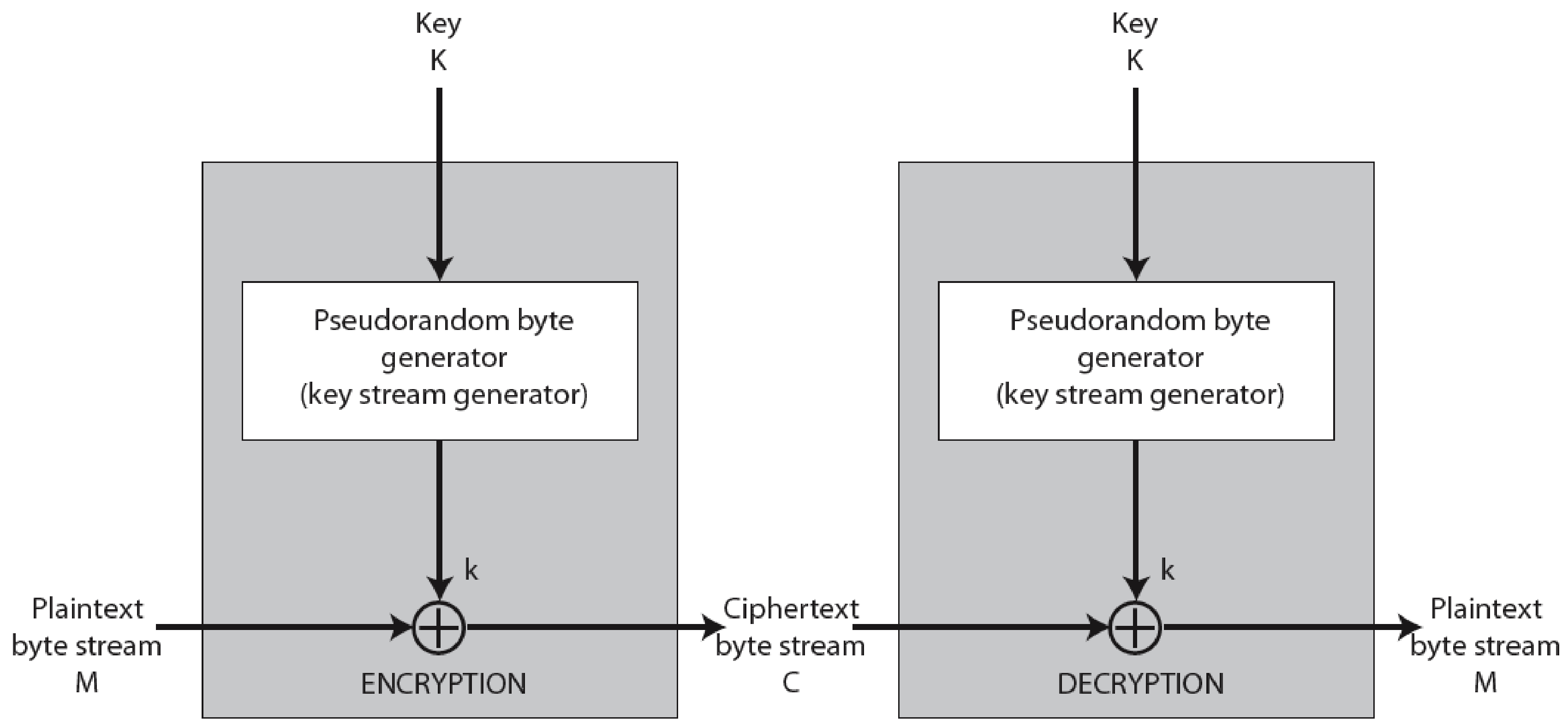
  If the processing of plain text takes place over a block of bits as a whole

  (usually the block size will be 64 bits.) then the cipher will be known as block cipher.

Stream Ciphers

- process message bit by bit (as a stream)
- have a pseudo random **keystream**
- combined (XOR) with plaintext bit by bit
- randomness of **stream key** completely destroys statistically properties in message
  - $C_i = M_i \text{ XOR } StreamKey_i$
- but must never reuse stream key
  - otherwise can recover messages (cf book cipher)

Stream Cipher Structure


Key
K
Pseudorandom byte generator (key stream generator)
k
Plaintext byte stream M
ENCRYPTION
Ciphertext byte stream C
Key
K
Pseudorandom byte generator (key stream generator)
k
DECRYPTION
Plaintext byte stream M

Stream Cipher Properties

- some design considerations are:
  - long period with no repetitions
  - statistically random
  - depends on large enough key
  - large linear complexity
- properly designed, can be as secure as a block cipher with same size key
- but usually simpler & faster

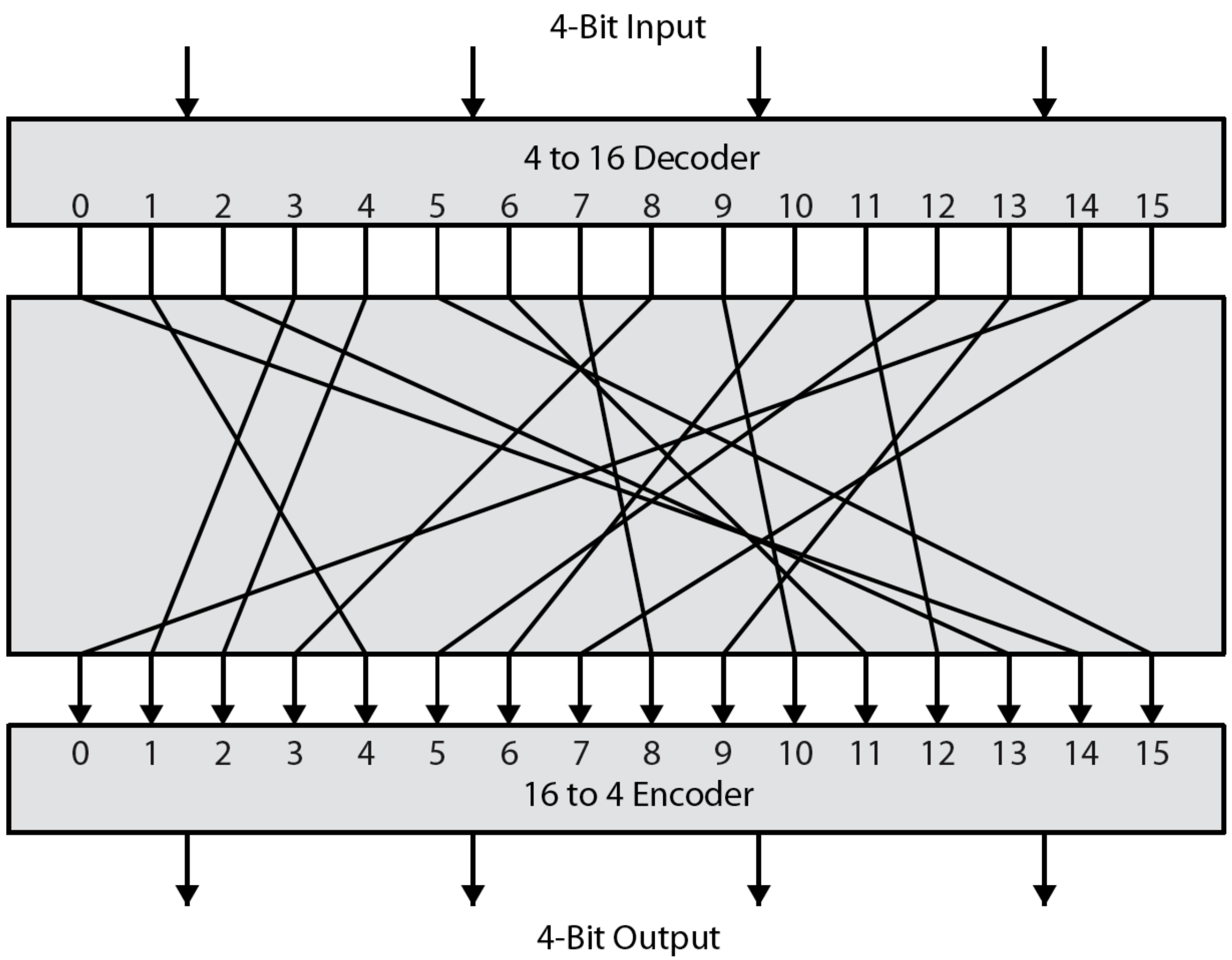
Block vs Stream Ciphers

- block ciphers process messages in blocks, each of which is then en/decrypted
- like a substitution on very big characters
  - 64-bits or more
- stream ciphers process messages a bit or byte at a time when en/decrypting
- many current ciphers are block ciphers
- broader range of applications

Block Cipher Principles

- most symmetric block ciphers are based on a **Feistel Cipher Structure**
- needed since must be able to **decrypt** ciphertext to recover messages efficiently
- block ciphers look like an extremely large substitution
- would need table of 2^{64} entries for a 64-bit block
- instead create from smaller building blocks
- using idea of a product cipher

Ideal Block Cipher



Claude Shannon and Substitution-Permutation Ciphers

- Claude Shannon introduced idea of substitution-permutation (S-P) networks in 1949 paper
- form basis of modern block ciphers
- S-P nets are based on the two primitive cryptographic operations seen before:
  - ***substitution* (S-box)**
  - ***permutation* (P-box)**
- provide *confusion* & *diffusion* of message & key

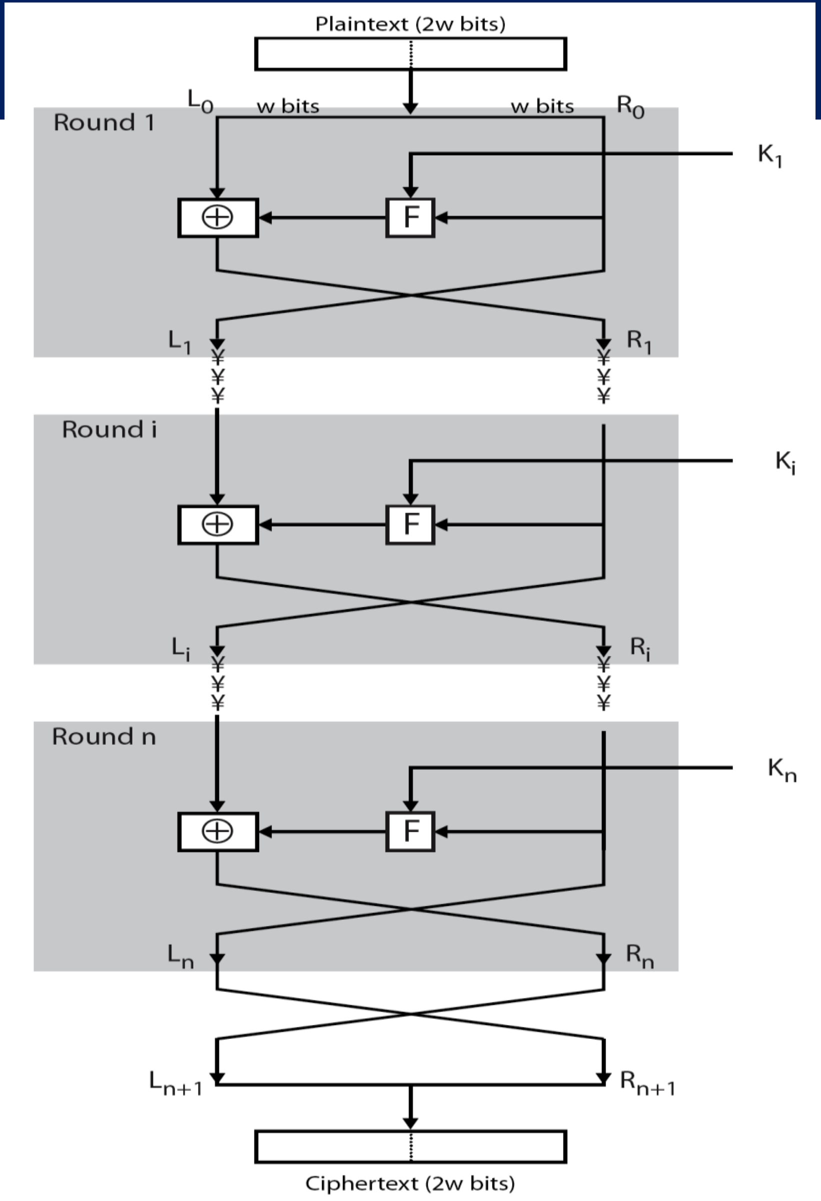
Confusion and Diffusion

- cipher needs to completely obscure statistical properties of original message
- a one-time pad does this
- more practically Shannon suggested combining S & P elements to obtain:
- **diffusion** – dissipates statistical structure of plaintext over bulk of ciphertext
- **confusion** – makes relationship between ciphertext and key as complex as possible

Feistel Cipher Structure

- Horst Feistel devised the **feistel cipher**
  - based on concept of invertible product cipher
- partitions input block into two halves
  - process through multiple rounds which
  - perform a substitution on left data half
  - based on round function of right half & subkey
  - then have permutation swapping halves
- implements Shannon's S-P net concept

Feistel Cipher Structure



Feistel Cipher Design Elements

- block size
- key size
- number of rounds
- subkey generation algorithm
- round function
- fast software en/decryption
- ease of analysis

Block Cipher - Modes of Operation

- block ciphers encrypt fixed size blocks
  - eg. DES encrypts 64-bit blocks with 56-bit key
- need some way to en/decrypt arbitrary amounts of data in practise
- **ANSI X3.106-1983 Modes of Use** (now FIPS 81) defines 4 possible modes
- subsequently 5 defined for AES & DES
- have **block** and **stream** modes

Modes:

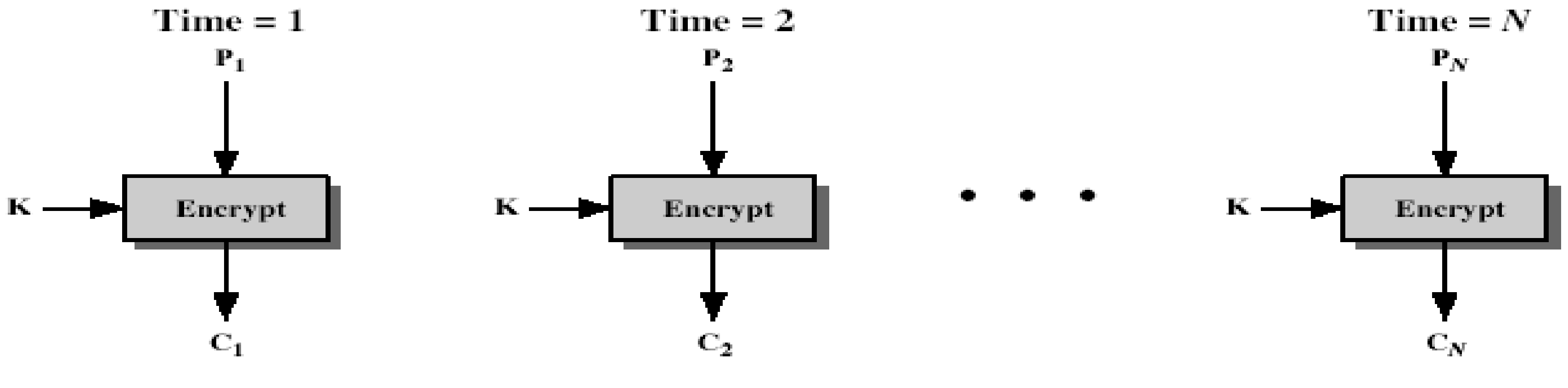
1. **Electronic Code Book (ECB)**
2. **Cipher Block Chaining (CBC)**
3. **Cipher Feedback (CFB)**
4. **Output Feedback (OFB)**
5. **Counter (CTR)**

1. Electronic Code Book (ECB)

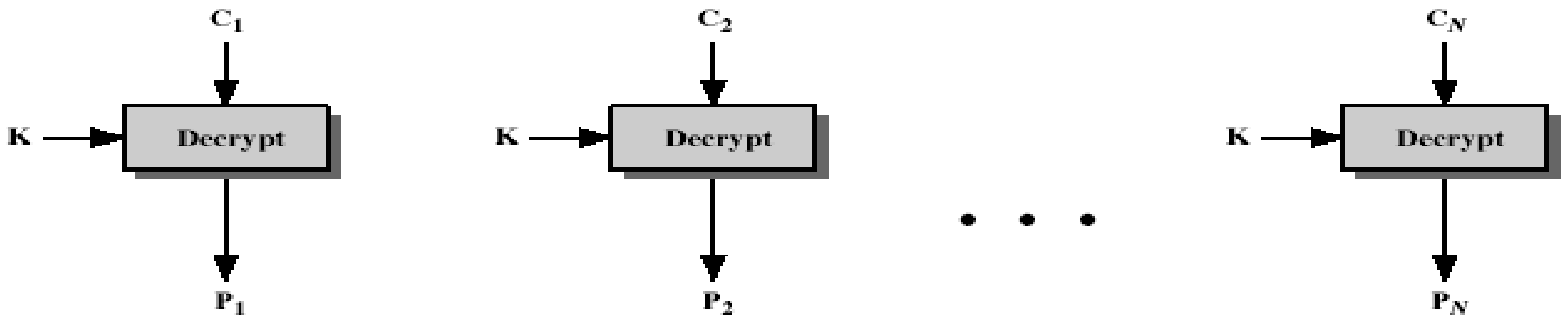
- message is broken into independent blocks which are encrypted
- each block is a value which is substituted, like a codebook, hence name
- each block is encoded independently of the other blocks

  $C_i = DES_{K1}(P_i)$
- uses: secure transmission of single values

1. Electronic Code Book (ECB)



(a) Encryption



(b) Decryption

1. Advantages and Limitations of ECB

- message repetitions may show in ciphertext
  - if aligned with message block
  - particularly with data such graphics
  - or with messages that change very little, which become a code-book analysis problem
- weakness is due to the encrypted message blocks being independent
- main use is sending a few blocks of data

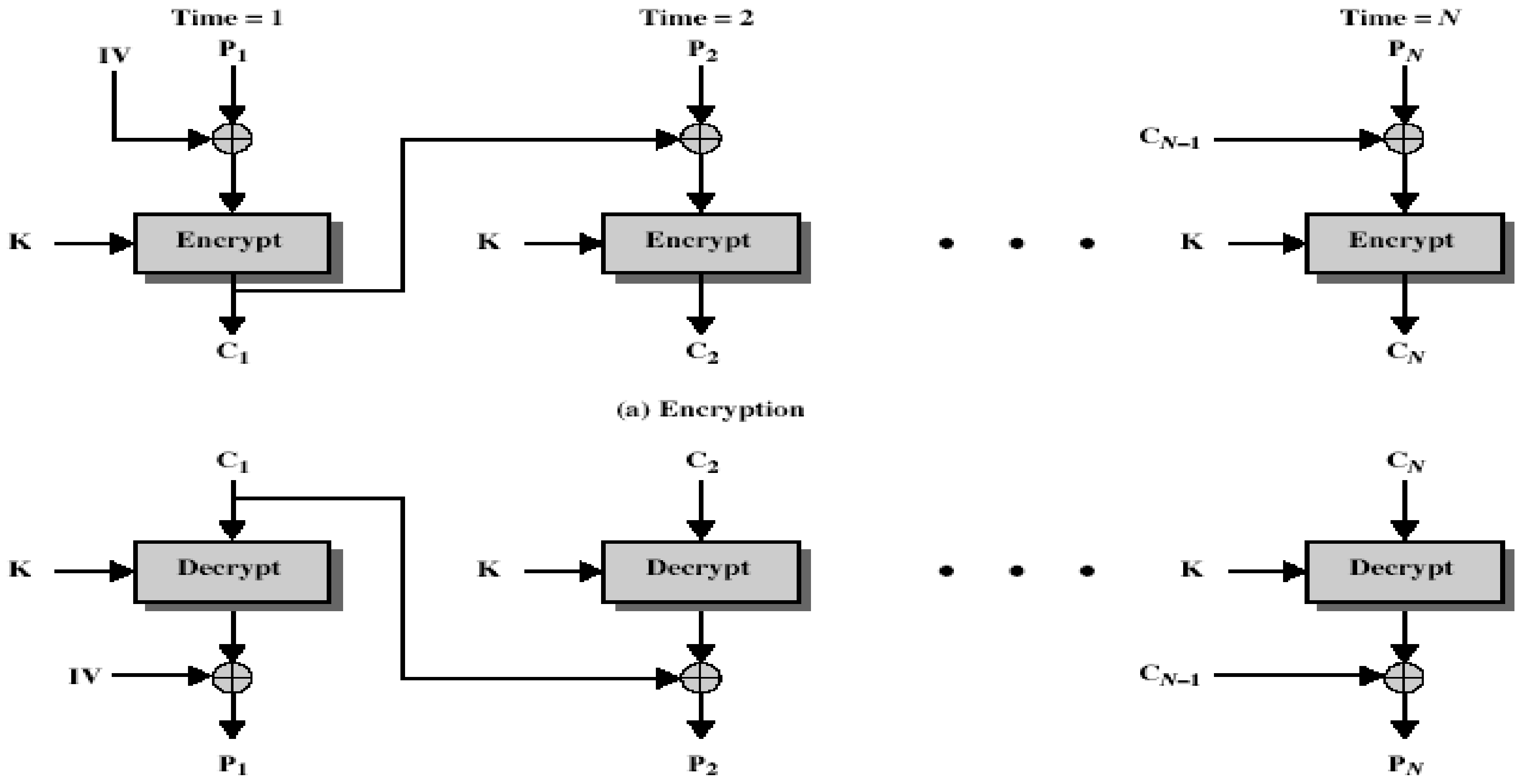
2. Cipher Block Chaining (CBC)

- message is broken into blocks
- **linked together** in encryption operation
- each previous cipher blocks is chained with current plaintext block, hence name
- use **Initial Vector** (IV) to start process

  $C_i = DES_{K1}(P_i \text{ XOR } C_{i-1})$

  $C_{-1} = IV$
- uses: bulk data encryption, authentication

2. Cipher Block Chaining (CBC)



(a) Encryption

(b) Decryption

2. Cipher Block Chaining (CBC)

Message Padding

- at end of message must handle a possible last short block
  - which is not as large as block size of cipher
  - pad either with known non-data value (eg. nulls)
  - or pad last block along with count of pad size
    - eg. [b1 b2 b3 0 0 0 0 5]
    - means have 3 data bytes, then 5 bytes pad+count
  - this may require an extra entire block over those in message
- there are other, more esoteric modes, which avoid the need for an extra block

2. Advantages and Limitations of CBC

- a ciphertext block depends on **all** blocks before it
- any change to a block affects all following ciphertext blocks
- need **Initialization Vector** (IV)
  - which must be known to sender & receiver
  - if sent in clear, attacker can change bits of first block, and change IV to compensate
  - hence IV must either be a fixed value (as in EFTPOS)
  - or must be sent encrypted in ECB mode before rest of message

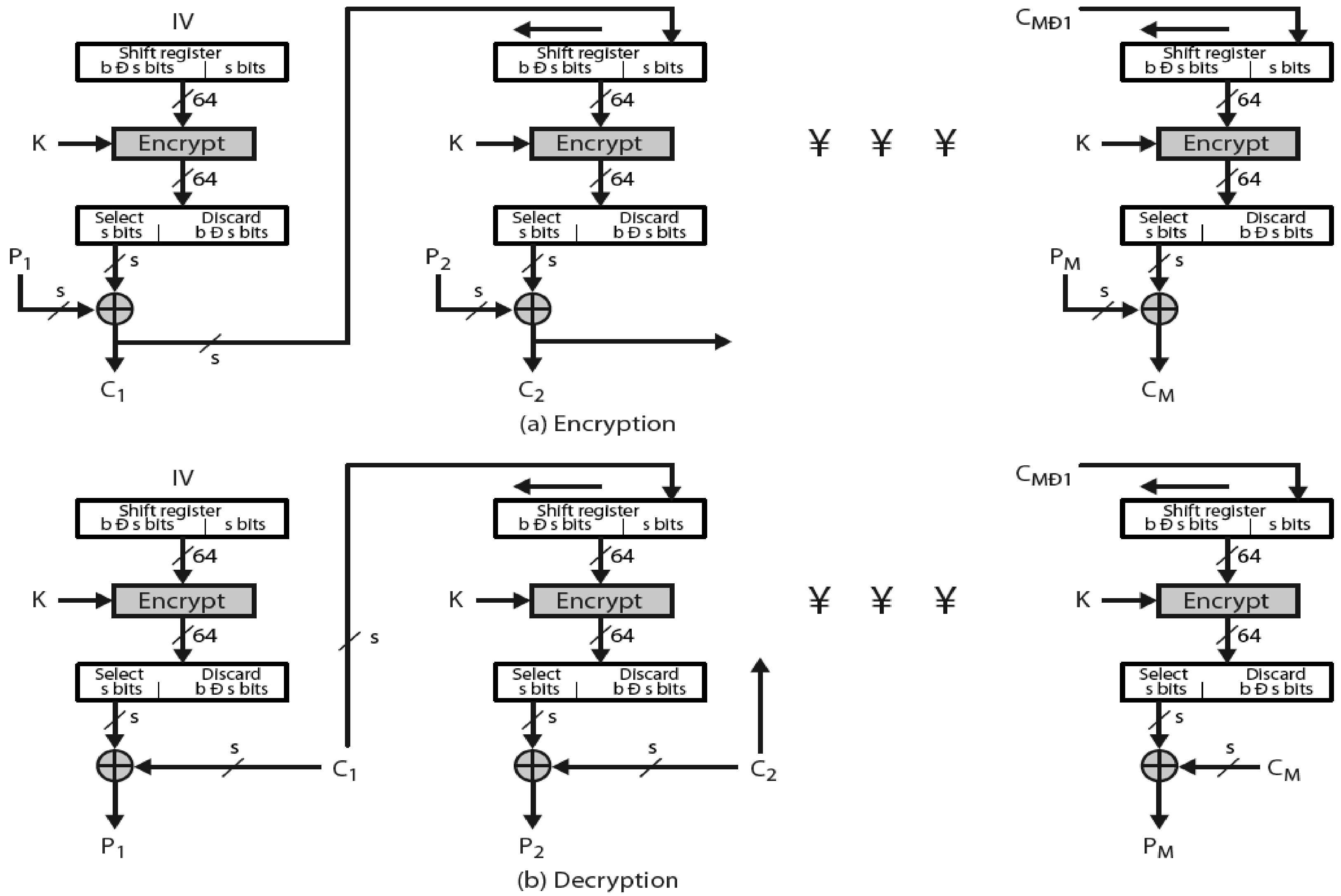
3. Cipher Feedback (CFB)

- message is treated as a stream of bits
- added to the output of the block cipher
- result is feed back for next stage (hence name)
- standard allows any number of bit (1,8, 64 or 128 etc) to be feed back
  - denoted CFB-1, CFB-8, CFB-64, CFB-128 etc
- most efficient to use all bits in block (64 or 128)

  $C_i = P_i \text{ XOR } DES_{K1}(C_{i-1})$

  $C_{-1} = IV$
- uses: stream data encryption, authentication

3. Cipher Feedback (CFB)



3. Advantages and Limitations of CFB

- appropriate when data arrives in bits/bytes
- most common stream mode
- the block cipher is used in **encryption** mode at **both** ends (Encryption & Decryption)
- errors propagate for several blocks after the error

4. Output Feedback (OFB)

- message is treated as a stream of bits
- output of cipher is added to message
- output is then feed back (hence name)
- feedback is independent of message
- can be computed in advance

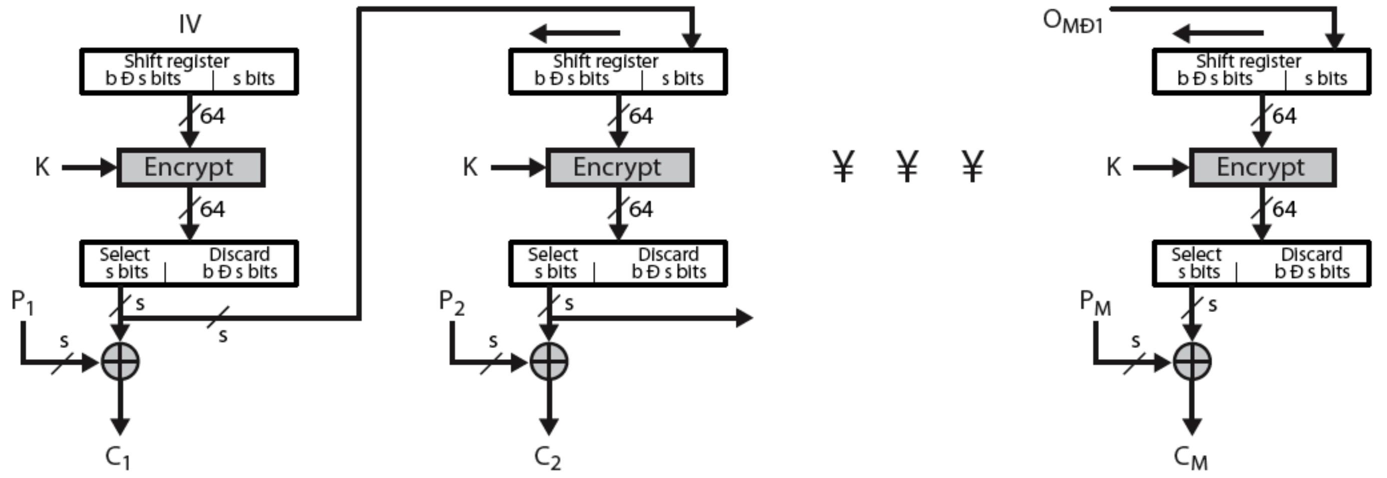
  $C_i = P_i \text{ XOR } O_i$

  $O_i = DES_{K1}(O_{i-1})$

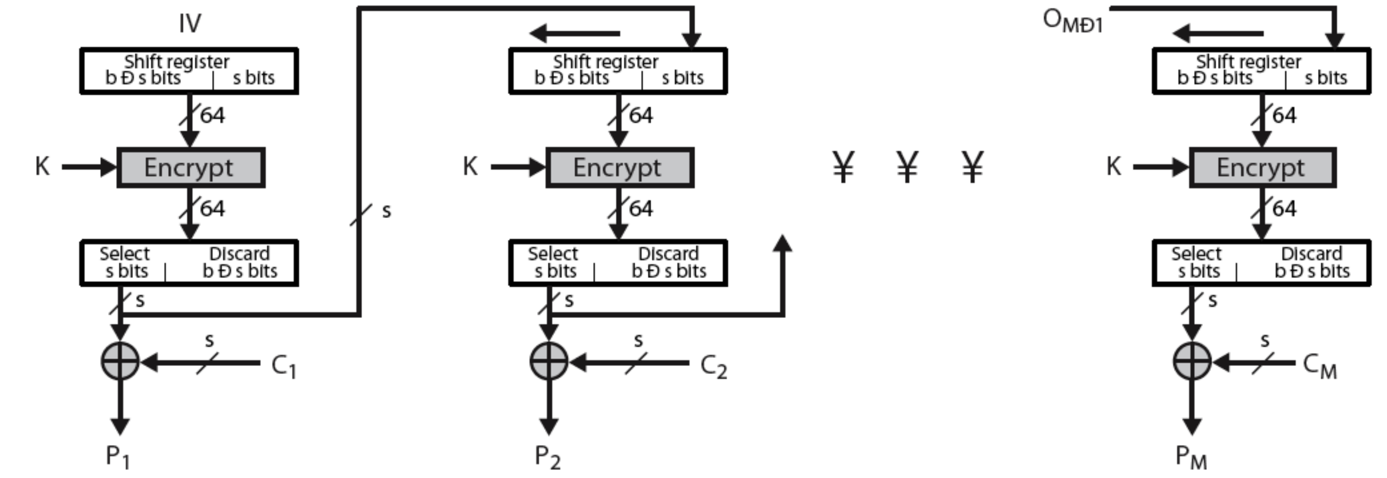
  $O_{-1} = IV$

- uses: stream encryption on noisy channels

4. Output Feedback (OFB)



(a) Encryption



(b) Decryption

4. Advantages and Limitations of OFB

- bit errors do not propagate
- more vulnerable to message stream modification
- a variation of a Vernam cipher
  - hence must **never** reuse the same sequence (key+IV)
- sender & receiver must remain in sync
- originally specified with m-bit feedback
- subsequent research has shown that only **full block feedback** (ie CFB-64 or CFB-128) should ever be used

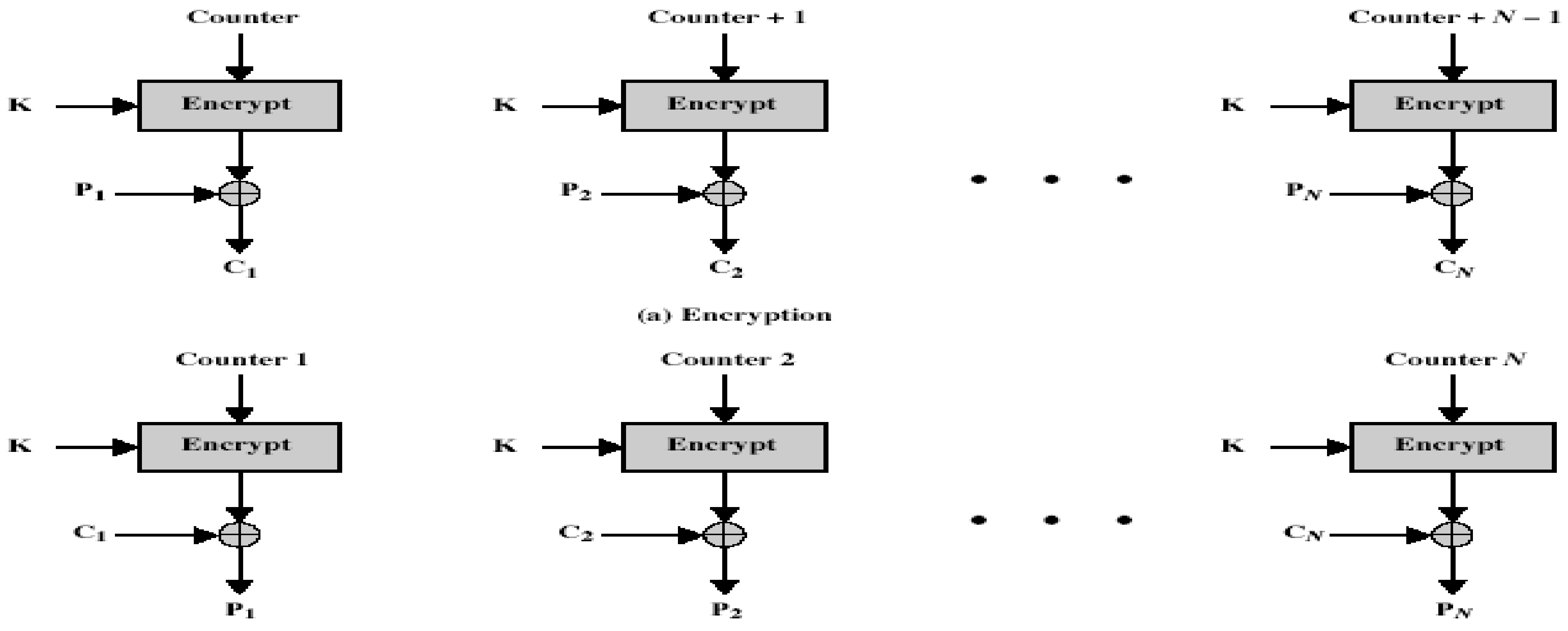
5. Counter (CTR)

- a “new” mode, though proposed early on
- similar to OFB but encrypts counter value rather than any feedback value
- must have a different key & counter value for every plaintext block (never reused)

$$C_i = P_i \text{ XOR } O_i$$
$$O_i = DES_{K1}(i)$$

- uses: high-speed network encryptions

5. Counter (CTR)



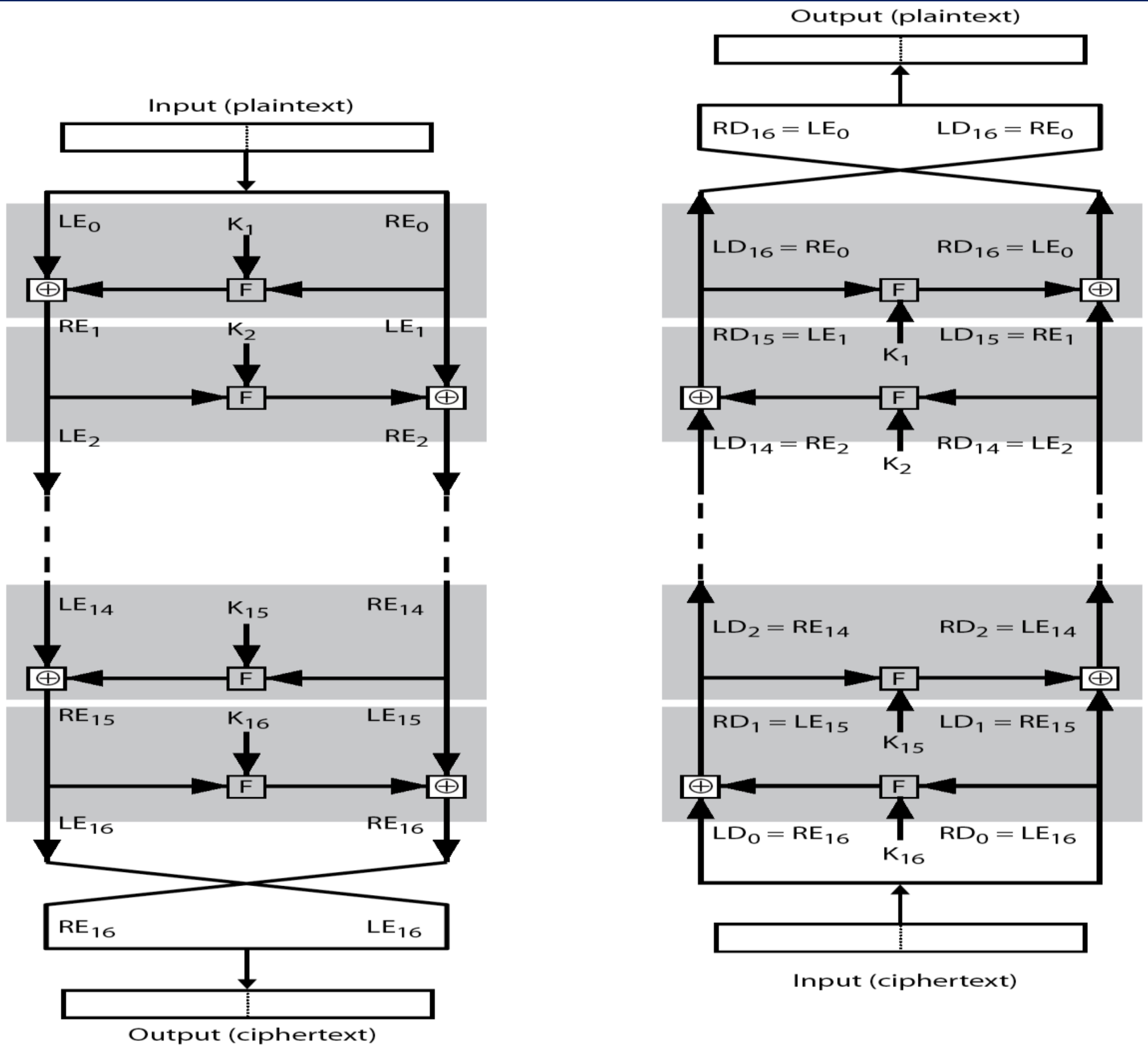
(a) Encryption

(b) Decryption

5. Advantages and Limitations of CTR

- efficiency
  - can do parallel encryptions in h/w or s/w
  - can preprocess in advance of need
  - good for bursty high speed links
- random access to encrypted data blocks
- provable security (good as other modes)
- but must ensure never reuse key/counter values, otherwise could break (cf OFB)

Feistel Cipher Encryption/Decryption



Data Encryption Standard (DES)

- most widely used block cipher in world
- adopted in 1977 by NBS (now NIST)
  - as FIPS PUB 46
- encrypts 64-bit data using 56-bit key
- has widespread use
- has been considerable controversy over its security

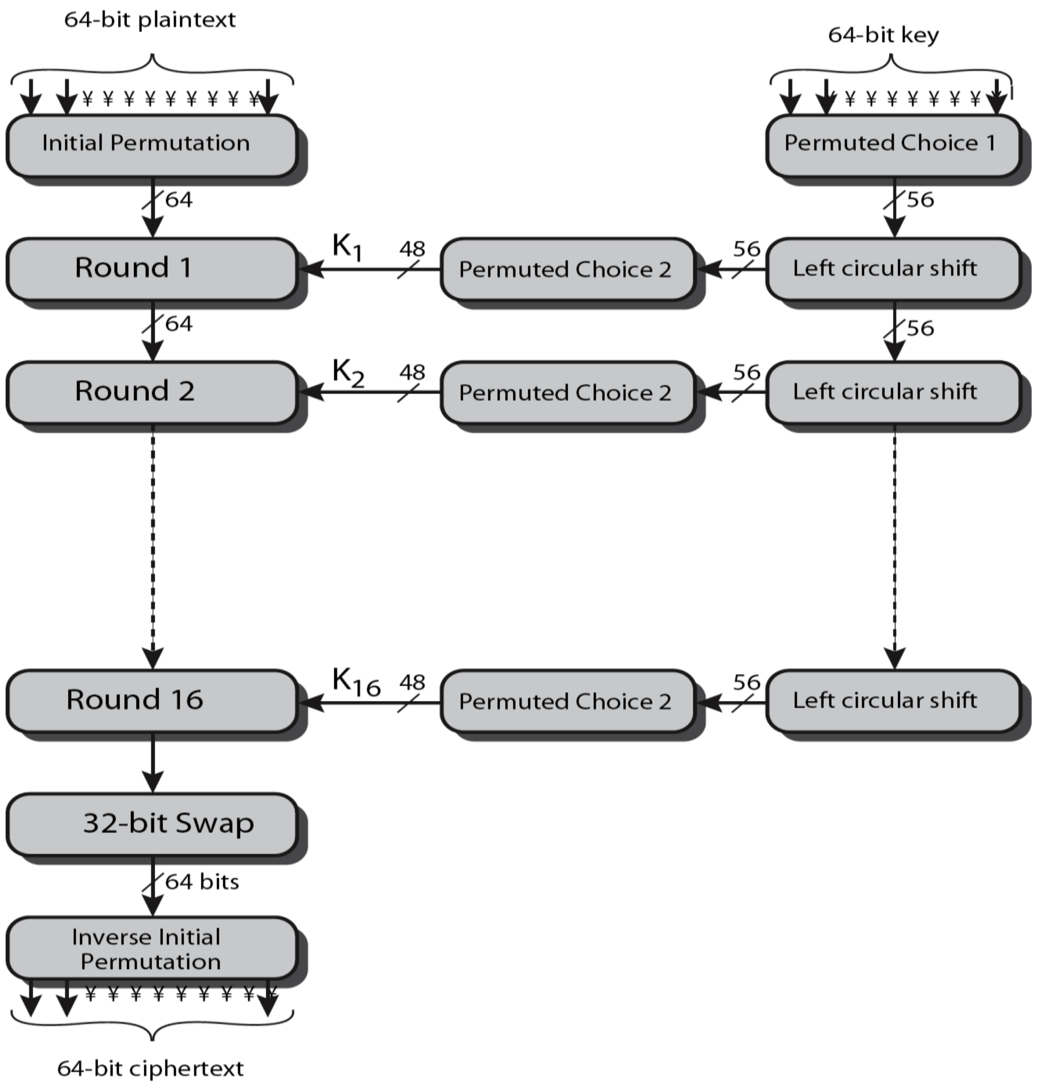
DES History

- IBM developed Lucifer cipher
  - by team led by Feistel in late 60's
  - used 64-bit data blocks with 128-bit key
- then redeveloped as a commercial cipher with input from NSA and others
- in 1973 NBS issued request for proposals for a national cipher standard
- IBM submitted their revised Lucifer which was eventually accepted as the DES

DES Design Controversy

- although DES standard is public
- was considerable controversy over design
  - in choice of 56-bit key (vs Lucifer 128-bit)
  - and because design criteria were classified
- subsequent events and public analysis show in fact design was appropriate
- use of DES has flourished
  - especially in financial applications
  - still standardised for legacy application use

DES Encryption Overview



DES : Initial Permutation IP

- first step of the data computation
- IP reorders the input data bits
- even bits to LH half, odd bits to RH half
- quite regular in structure (easy in h/w)
- example:

```
IP(675a6967 5e5a6b5a) = (ffb2194d 004df6fb)
```

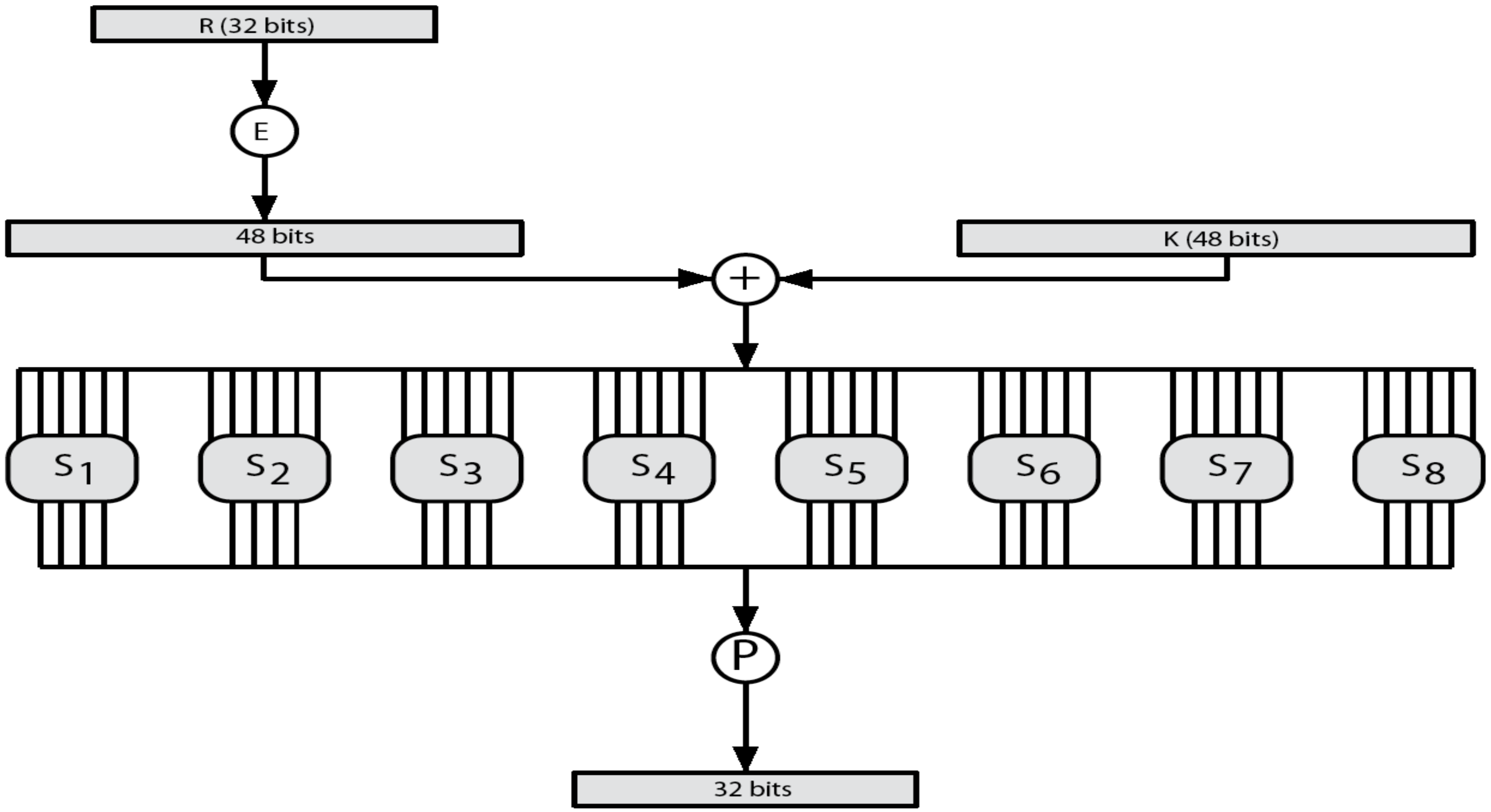
DES Round Structure

- uses two 32-bit L & R halves
- as for any Feistel cipher can describe as:

  $L_i = R_{i-1}$

  $R_i = L_{i-1} \oplus \mathrm{F}(R_{i-1}, K_i)$
- F takes 32-bit R half and 48-bit subkey:
  - expands R to 48-bits using perm E
  - adds to subkey using XOR
  - passes through 8 S-boxes to get 32-bit result
  - finally permutes using 32-bit perm P

DES Round Structure



Substitution Boxes S

- have eight S-boxes which map 6 to 4 bits
- each S-box is actually 4 little 4 bit boxes
  - outer bits 1 & 6 (**row** bits) select one row of 4
  - inner bits 2-5 (**col** bits) are substituted
  - result is 8 lots of 4 bits, or 32 bits
- row selection depends on both data & key
  - feature known as autoclaving (autokeying)
- example:
  - `S(18 09 12 3d 11 17 38 39) = 5fd25e03`

DES Key Schedule

- forms subkeys used in each round
  - initial permutation of the key (PC1) which selects 56-bits in two 28-bit halves
  - 16 stages consisting of:
    - rotating **each half** separately either 1 or 2 places depending on the **key rotation schedule** K
    - selecting 24-bits from each half & permuting them by PC2 for use in round function F
- note practical use issues in h/w vs s/w

DES Decryption

- decrypt must unwind steps of data computation
- with Feistel design, do encryption steps again using subkeys in reverse order (SK16 … SK1)
  - IP undoes final FP step of encryption
  - 1st round with SK16 undoes 16th encrypt round
  - ….
  - 16th round with SK1 undoes 1st encrypt round
  - then final FP undoes initial encryption IP
  - thus recovering original data value

DES: Avalanche Effect

- key desirable property of encryption alg
- where a change of **one** input or key bit results in changing approx **half** output bits
- making attempts to “home-in” by guessing keys impossible
- DES exhibits strong avalanche

Strength of DES – Key Size

- 56-bit keys have $2^{56} = 7.2 \times 10^{16}$ values
- brute force search looks hard
- recent advances have shown is possible
  - in 1997 on Internet in a few months
  - in 1998 on dedicated h/w (EFF) in a few days
  - in 1999 above combined in 22hrs!
- still must be able to recognize plaintext
- must now consider alternatives to DES

Strength of DES – Analytic Attacks

- now have several analytic attacks on DES
- these utilise some deep structure of the cipher
  - by gathering information about encryptions
  - can eventually recover some/all of the sub-key bits
  - if necessary then exhaustively search for the rest
- generally these are statistical attacks
- include
  - differential cryptanalysis
  - linear cryptanalysis
  - related key attacks

Strength of DES – Timing Attacks

- attacks actual implementation of cipher
- use knowledge of consequences of implementation to derive information about some/all subkey bits
- specifically use fact that calculations can take varying times depending on the value of the inputs to it
- particularly problematic on smartcards

Advanced Encryption Standard : AES Requirements

- private key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- stronger & faster than Triple-DES
- active life of 20-30 years (+ archival use)
- provide full specification & design details
- both C & Java implementations
- NIST have released all submissions & unclassified analyses

AES Evaluation Criteria

- Initial criteria:
  - security – effort for practical cryptanalysis
  - cost – in terms of computational efficiency
  - algorithm & implementation characteristics
- Final criteria
  - general security
  - ease of software & hardware implementation
  - implementation attacks
  - flexibility (in en/decrypt, keying, other factors)

AES Shortlist

- after testing and evaluation, shortlist in Aug-99:
  - MARS (IBM) - complex, fast, high security margin
  - RC6 (USA) - v. simple, v. fast, low security margin
  - Rijndael (Belgium) - clean, fast, good security margin
  - Serpent (Euro) - slow, clean, v. high security margin
  - Twofish (USA) - complex, v. fast, high security margin
- then subject to further analysis & comment
- saw contrast between algorithms with
  - few complex rounds verses many simple rounds
  - which refined existing ciphers verses new proposals

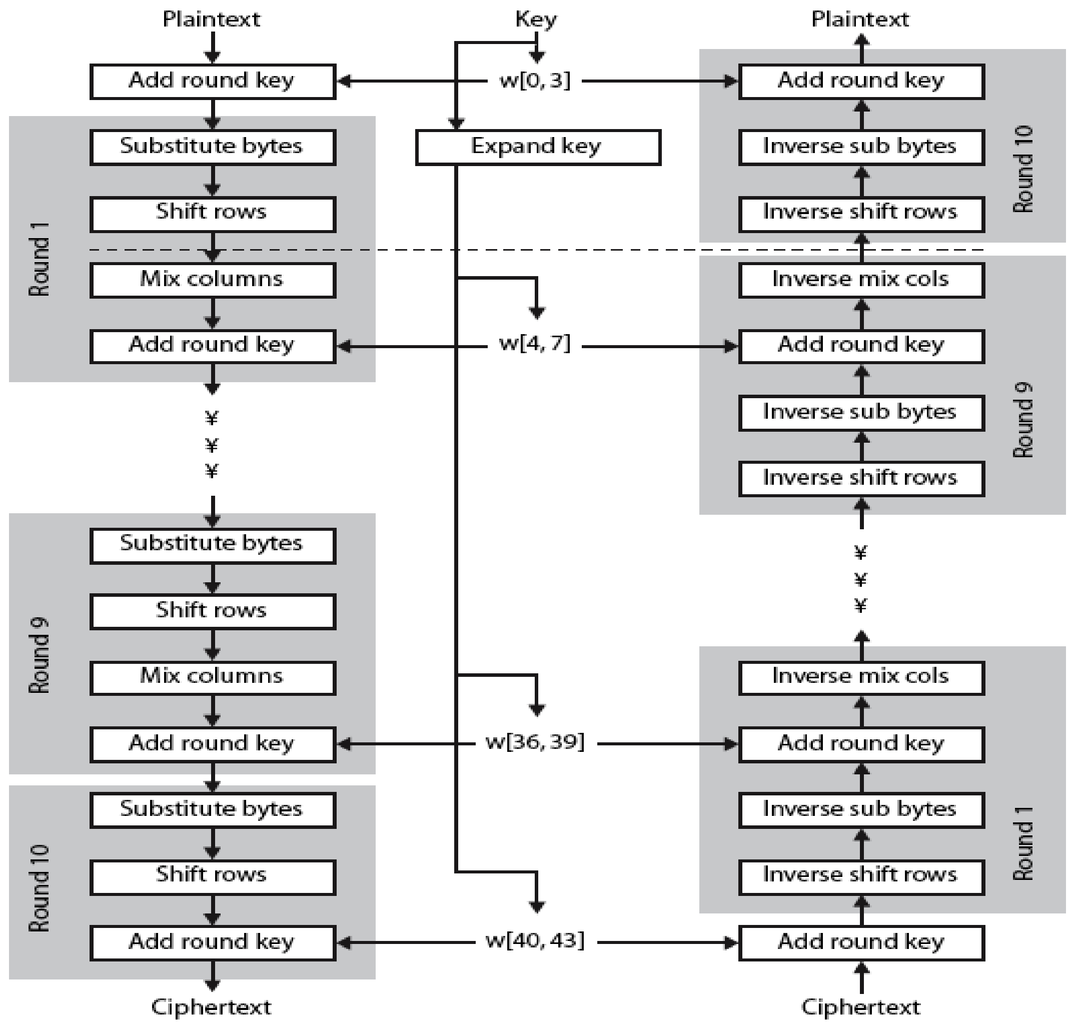
The AES Cipher - Rijndael

- designed by Rijmen-Daemen in Belgium
- has 128/192/256 bit keys, 128 bit data
- an **iterative** rather than **feistel** cipher
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round
- designed to be:
  - resistant against known attacks
  - speed and code compactness on many CPUs
  - design simplicity

AES: Rijndael

- data block of 4 columns of 4 bytes is state
- key is expanded to array of words
- has 9/11/13 rounds in which state undergoes:
  - byte substitution (1 S-box used on every byte)
  - shift rows (permute bytes between groups/columns)
  - mix columns (subs using matrix multipy of groups)
  - add round key (XOR state with key material)
  - view as alternating XOR key & scramble data bytes
- initial XOR key material & incomplete last round
- with fast XOR & table lookup implementation

AES: Rijndael

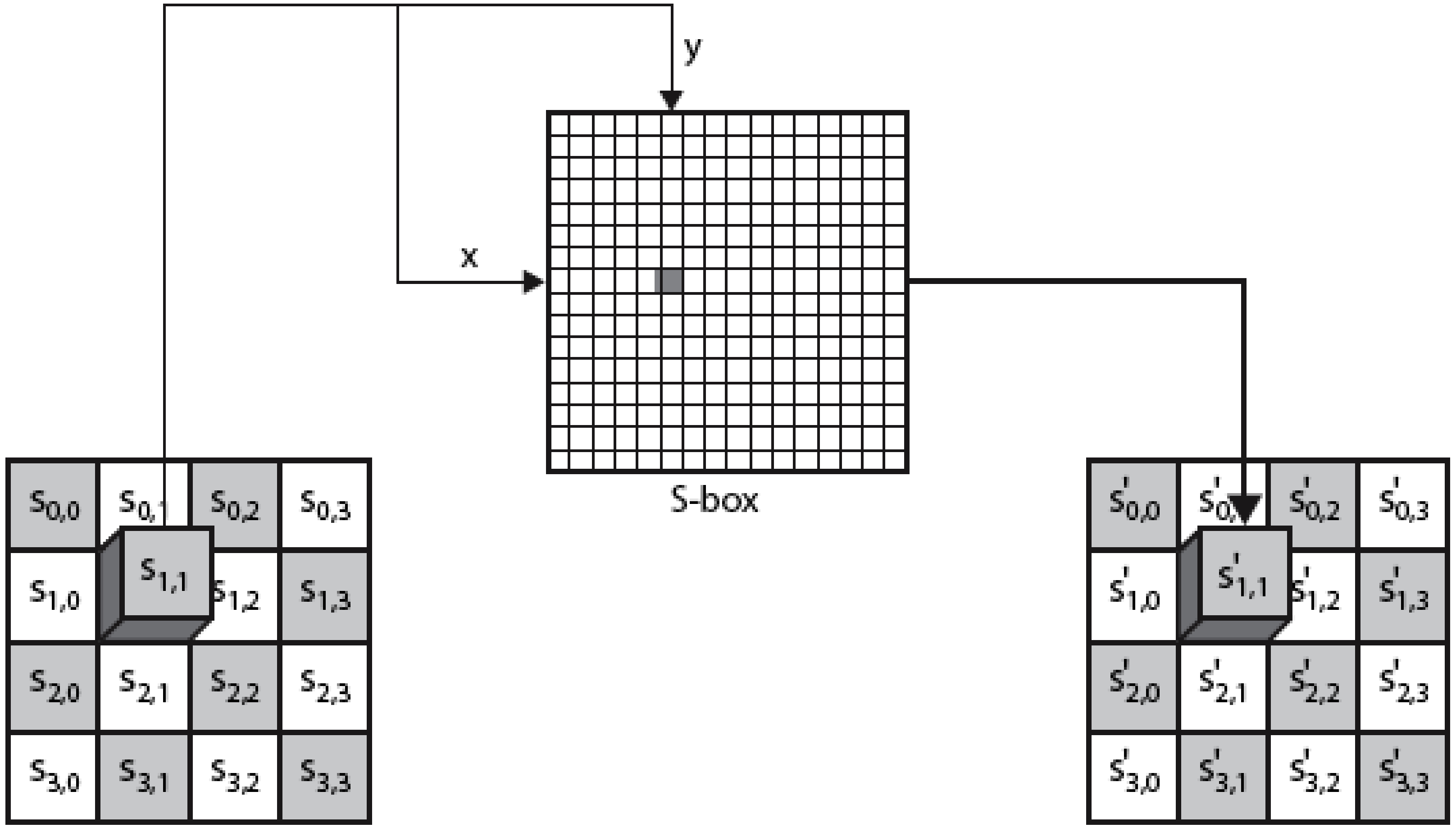


(a) Encryption (b) Decryption

AES: Byte Substitution

- a simple substitution of each byte
- uses one table of 16x16 bytes containing a permutation of all 256 8-bit values
- each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)
  - eg. byte {95} is replaced by byte in row 9 column 5
  - which has value {2A}
- S-box constructed using defined transformation of values in GF(2^8)
- designed to be resistant to all known attacks

AES: Byte Substitution

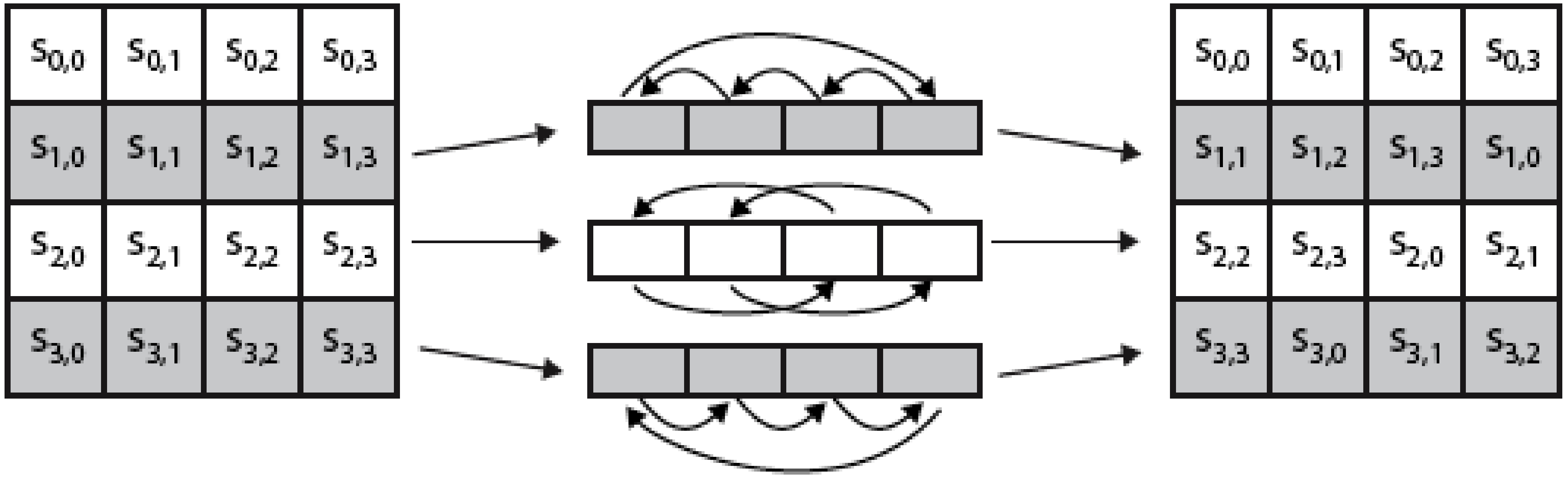


y
x
S-box
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

AES: Shift Rows

- a circular byte shift in each each
  - 1st row is unchanged
  - 2nd row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left
- decrypt inverts using shifts to right
- since state is processed by columns, this step permutes bytes between the columns

AES: Shift Rows



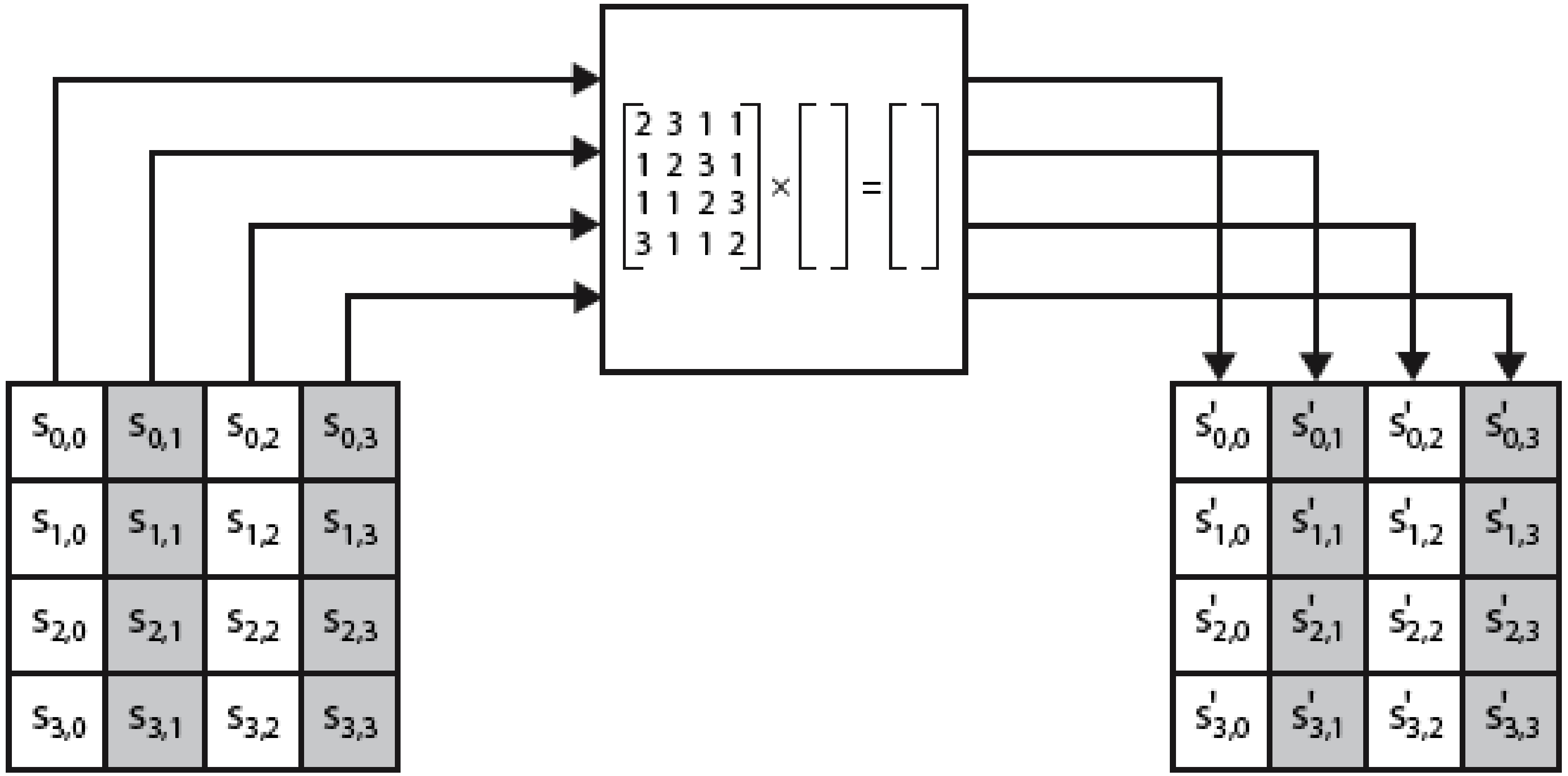
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$

AES: Mix Columns

- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column
- effectively a matrix multiplication in GF(2^8) using prime poly m(x) =$x^8+x^4+x^3+x+1$

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

AES: Mix Columns



$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \\ \end{bmatrix}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

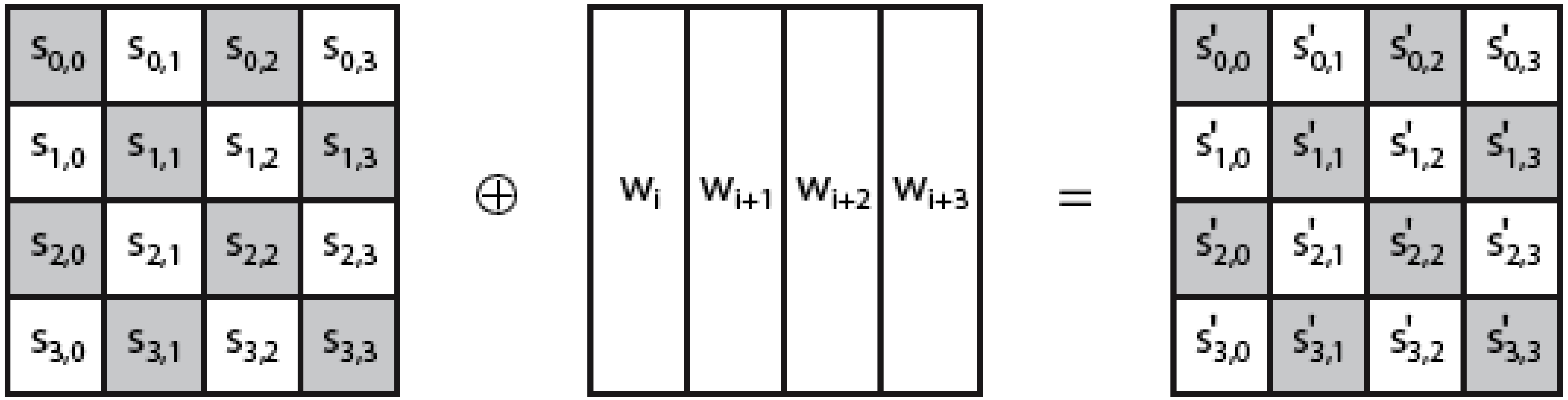
AES: Mix Columns

- can express each col as 4 equations
  - to derive each new byte in col
- decryption requires use of inverse matrix
  - with larger coefficients, hence a little harder
- have an alternate characterization
  - each column a 4-term polynomial
  - with coefficients in GF(2^8)
  - and polynomials multiplied modulo (x^4+1)

AES: Add Round Key

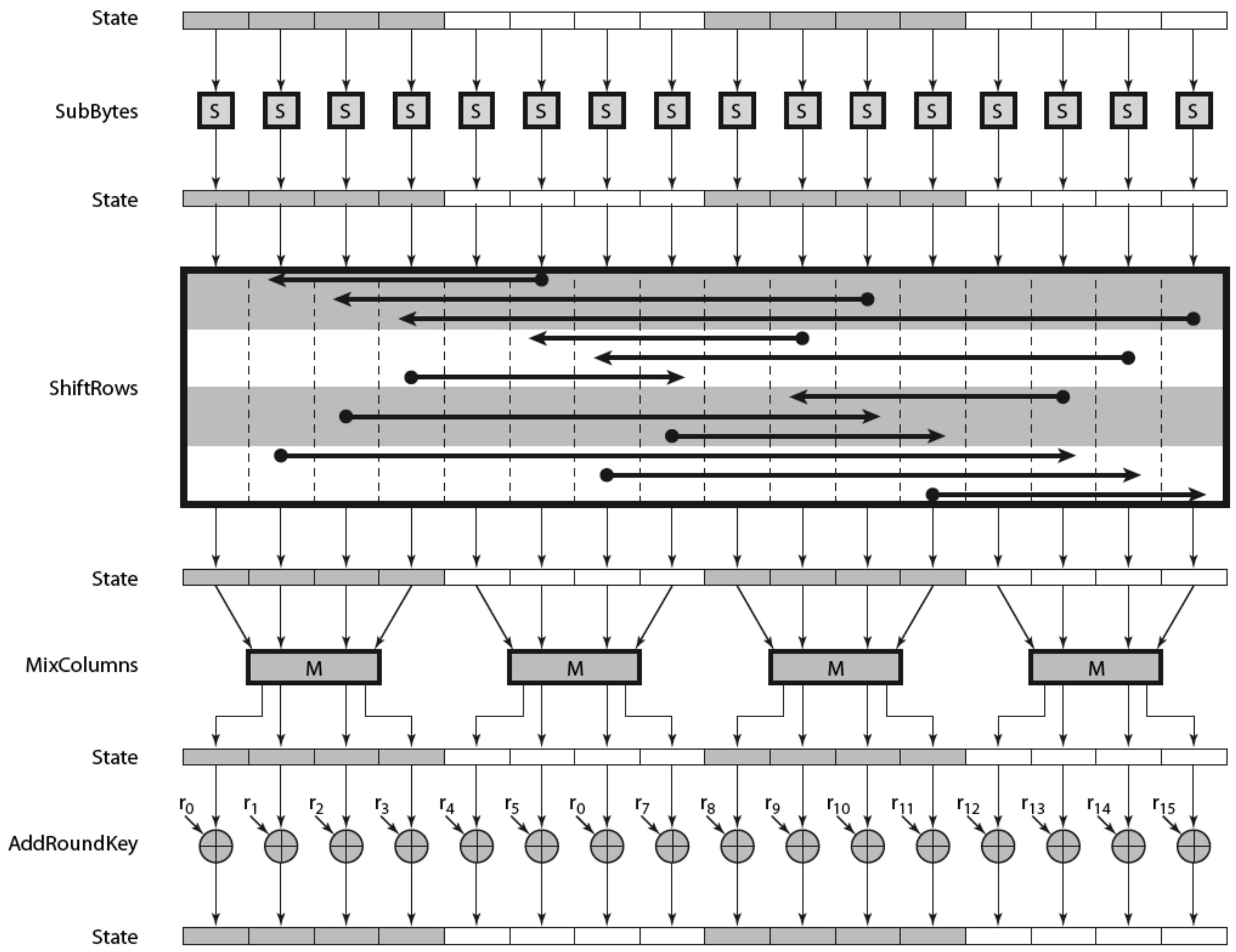
- XOR state with 128-bits of the round key
- again processed by column (though effectively a series of byte operations)
- inverse for decryption identical
  - since XOR own inverse, with reversed keys
- designed to be as simple as possible
  - a form of Vernam cipher on expanded key
  - requires other stages for complexity / security

AES: Add Round Key



$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$\oplus$
w_i w_{i+1} w_{i+2} w_{i+3}
$=$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

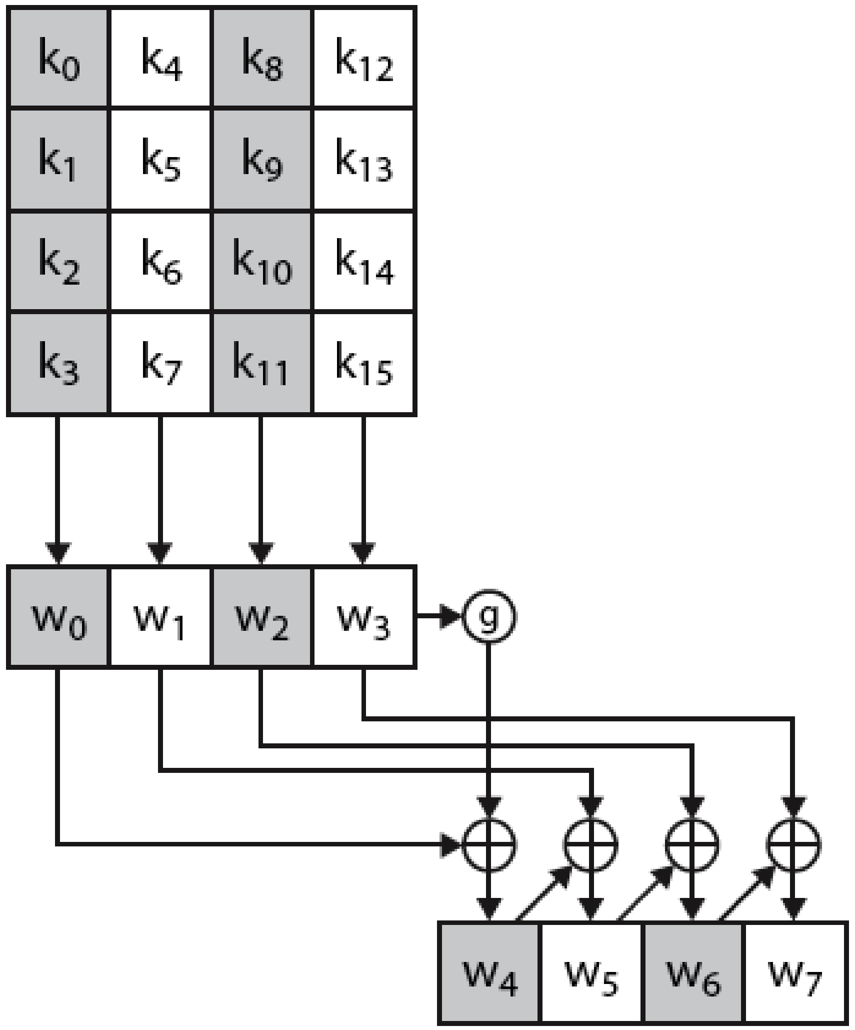
AES Round


State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r_0 r_1 r_2 r_3 r_4 r_5 r_0 r_7 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14} r_{15}
State

AES Key Expansion

- takes 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words
- start by copying key into first 4 words
- then loop creating words that depend on values in previous & 4 places back
  - in 3 of 4 cases just XOR these together
  - 1st word in 4 has rotate + S-box + XOR round constant on previous, before XOR 4th back

AES Key Expansion



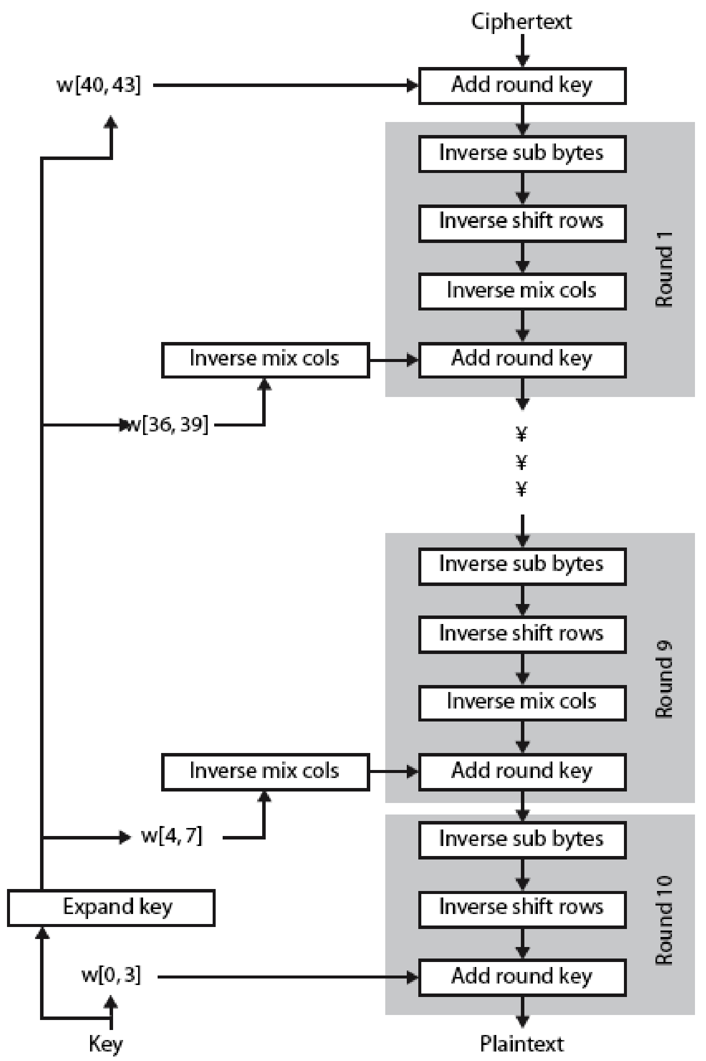
AES: Key Expansion Rationale

- designed to resist known attacks
- design criteria included
  - knowing part key insufficient to find many more
  - invertible transformation
  - fast on wide range of CPU’s
  - use round constants to break symmetry
  - diffuse key bits into round keys
  - enough non-linearity to hinder analysis
  - simplicity of description

AES Decryption

- AES decryption is not identical to encryption since steps done in reverse
- but can define an equivalent inverse cipher with steps as for encryption
  - but using inverses of each step
  - with a different key schedule
- works since result is unchanged when
  - swap byte substitution & shift rows
  - swap mix columns & add (tweaked) round key

AES Decryption



AES: Implementation Aspects

- can efficiently implement on 8-bit CPU
  - byte substitution works on bytes using a table of 256 entries
  - shift rows is simple byte shift
  - add round key works on byte XOR's
- mix columns requires matrix multiply in GF(2^8) which works on byte values, can be simplified to use table lookups & byte XOR's
- can efficiently implement on 32-bit CPU
  - redefine steps to use 32-bit words
  - can precompute 4 tables of 256-words
  - then each column in each round can be computed using 4 table lookups + 4 XORs
  - at a cost of 4Kb to store tables
- designers believe this very efficient implementation was a key factor in its selection as the AES cipher

RSA

- by **Rivest, Shamir & Adleman** of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation in a finite (Galois) field over integers modulo a prime
  - exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
  - factorization takes $O(e^{\log n \log \log n \log n})$ operations (hard)

RSA Key Setup

- each user generates a public/private key pair by:
- selecting two large primes at random : $p,\ q$
- computing their system modulus $n=p.q$
  - note $\varnothing(n)=(p-1)(q-1)$
- selecting at random the encryption key e
  - where $1<e<\varnothing(n),\ \gcd(e,\varnothing(n))=1$
- solve following equation to find decryption key d
  - $e.d=1 \bmod \varnothing(n)$ and $0\le d\le n$
- publish their public encryption key: PU={e,n}
- keep secret private decryption key: PR={d,n}

RSA Use

- to encrypt a message M the sender:
  - obtains **public key** of recipient $PU=\{e,n\}$
  - computes: $C = M^e \bmod n$, **where** $0 \le M < n$
- to decrypt the ciphertext C the owner:
  - uses their private key $PR=\{d,n\}$
  - computes: $M = C^d \bmod n$
- note that the message M must be smaller than the modulus n (block if needed)

Why RSA Works

- because of Euler's Theorem:
  - $a^{\varnothing(n)} \bmod n = 1$ where $\gcd(a,n)=1$
- in RSA have:
  - $n=p.q$
  - $\varnothing(n)=(p-1)(q-1)$
  - carefully chose e & d to be inverses $\bmod \varnothing(n)$
  - hence $e.d=1+k.\varnothing(n)$ for some k
- hence :

$$C^d = M^{e.d} = M^{1+k.\varnothing(n)} = M^1.(M^{\varnothing(n)})^k$$

$$= M^1.(1)^k = M^1 = M \bmod n$$

RSA Example - Key Setup

1. Select primes: $p=17$ & $q=11$
2. Compute $n = pq = 17 \times 11 = 187$
3. Compute $\varnothing(n) = (p-1)(q-1) = 16 \times 10 = 160$
4. Select e: $\gcd(e,160)=1$; choose $e=7$
5. Determine d: $de=1 \bmod 160$ and $d < 160$ Value is $d=23$ since $23 \times 7 = 161 = 10 \times 160 + 1$
6. Publish public key $PU=\{7,187\}$
7. Keep secret private key $PR=\{23,187\}$

RSA Example - En/Decryption

- sample RSA encryption/decryption is:
- given message M = 88
- encryption:

  $C = 88^7 \bmod 187 = 11$
- decryption:

  $M = 11^{23} \bmod 187 = 88$

RSA - Exponentiation

- can use the Square and Multiply Algorithm
- a fast, efficient algorithm for exponentiation
- concept is based on repeatedly squaring base
- and multiplying in the ones that are needed to compute the result
- look at binary representation of exponent
- only takes $O(\log_2 n)$ multiples for number n
  - eg. $7^5 = 7^4.7^1 = 3.7 = 10 \bmod 11$
  - eg. $3^{129} = 3^{128}.3^1 = 5.3 = 4 \bmod 11$

RSA -Efficient Encryption

- encryption uses exponentiation to power e
- hence if e small, this will be faster
  - often choose e=65537 (2^{16}-1)
  - also see choices of e=3 or e=17
- but if e too small (eg e=3) can attack
  - using Chinese remainder theorem & 3 messages with different modulii
- if e fixed must ensure `gcd(e,ø(n))=1`
  - ie reject any p or q not relatively prime to e

RSA - Efficient Decryption

- decryption uses exponentiation to power d
  - this is likely large, insecure if not
- can use the Chinese Remainder Theorem (CRT) to compute mod p & q separately. then combine to get desired answer
  - approx 4 times faster than doing directly
- only owner of private key who knows values of p & q can use this technique

RSA Key Generation

- users of RSA must:
  - determine two primes at random - `p, q`
  - select either `e` or `d` and compute the other
- primes `p,q` must not be easily derived from modulus `n=p.q`
  - means must be sufficiently large
  - typically guess and use probabilistic test
- exponents `e, d` are inverses, so use Inverse algorithm to compute the other

RSA Security

- possible approaches to attacking RSA are:
  - brute force key search (infeasible given size of numbers)
  - mathematical attacks (based on difficulty of computing ø(n), by factoring modulus n)
  - timing attacks (on running of decryption)
  - chosen ciphertext attacks (given properties of RSA)

RSA: Factoring Problem

- mathematical approach takes 3 forms:
  - factor `n=p.q`, hence compute `ø(n)` and then d
  - determine `ø(n)` directly and compute d
  - find d directly
- currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - as of May-05 best is 200 decimal digits (663) bit with LS
  - biggest improvement comes from improved algorithm
    - cf QS to GHFS to LS
  - currently assume 1024-2048 bit RSA is secure
    - ensure p, q of similar size and matching other constraints

RSA: Timing Attacks

- developed by Paul Kocher in mid-1990’s
- exploit timing variations in operations
  - eg. multiplying by small vs large number
  - or IF's varying which instructions executed
- infer operand size based on time taken
- RSA exploits time taken in exponentiation
- countermeasures
  - use constant exponentiation time
  - add random delays
  - blind values used in calculations

RSA: Chosen Ciphertext Attacks

- RSA is vulnerable to a Chosen Ciphertext Attack (CCA)
- attackers chooses ciphertexts & gets decrypted plaintext back
- choose ciphertext to exploit properties of RSA to provide info to help cryptanalysis
- can counter with random pad of plaintext
- or use Optimal Asymmetric Encryption Padding (OASP)