

**II4031 Kriptografi dan Koding**

# Block Cipher



Oleh: Rinaldi Munir

Program Studi Sistem dan Teknologi Informasi  
Sekolah Teknik Elektro dan Informatika

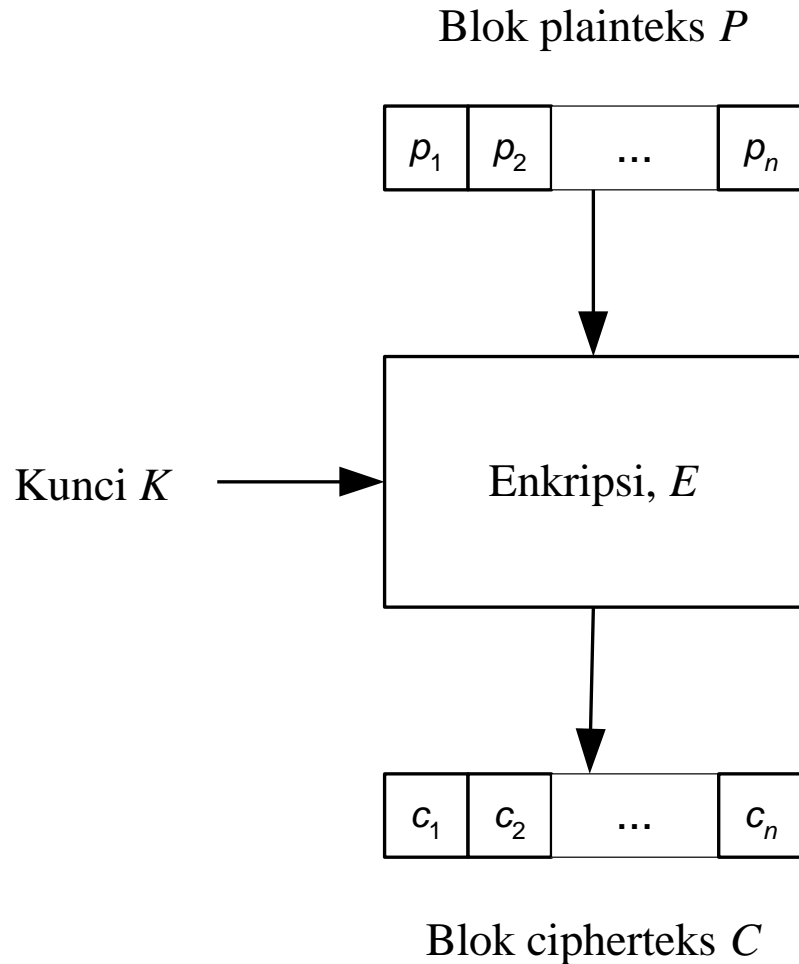
ITB

# Cipher Blok (*Block Cipher*)

- Berbeda dengan *cipher* alir, pada *cipher* blok plainteks dibagi menjadi blok-blok bit dengan panjang sama. Ukuran blok yang umum adalah 64 bit, 128 bit, 256 bit, dsb.
- Panjang blok cipherteks = panjang blok plainteks.
- Enkripsi dilakukan pada setiap blok plainteks dengan menggunakan bit-bit kunci
- Panjang kunci eksternal (yang diberikan oleh pengguna) tidak harus sama dengan panjang blok plainteks.

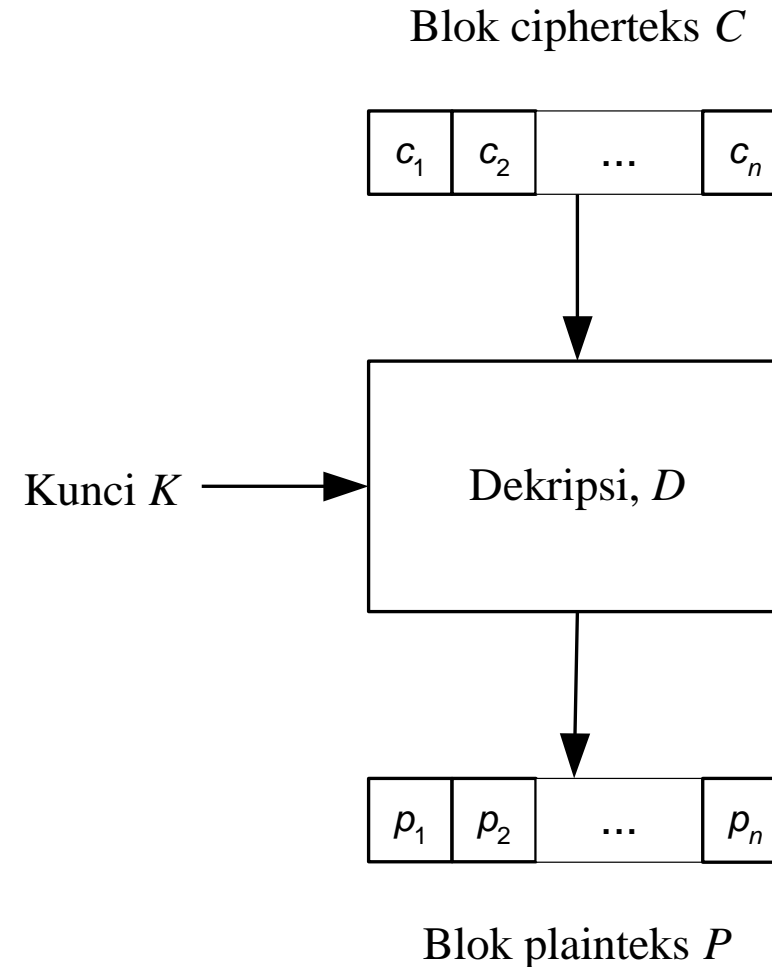
Blok plainteks berukuran  $n$  bit:

$$P = (p_1, p_2, \dots, p_n), p_i \in \{0, 1\}$$



Blok cipherteks berukuran  $n$  bit:

$$C = (c_1, c_2, \dots, c_n), c_i \in \{0, 1\}$$



- E dan D adalah *cipher* yang digunakan untuk enkripsi dan dekripsi

$$C = E(P) \quad \text{dan} \quad P = D(C)$$

- Contoh E yang sederhana misalkan algoritmanya adalah sbb:
  1. XOR-kan blok plainteks  $P$  dengan  $K$
  2. lalu geser secara *wrapping* bit-bit dari hasil langkah 1 satu posisi ke kiri

$$C = E_K(P) = (P \oplus K) \ll 1$$

- D adalah kebalikan (*invers*) dari E. Contoh D yang sederhana adalah sbb:
  1. Geser secara *wrapping* bit-bit ciphertes  $C$  satu posisi ke kanan
  2. Lalu XOR-kan blok hasil Langkah 1 dengan  $K$

$$P = E_K(C) = (C \gg 1) \oplus K$$

**Contoh:** Misalkan plainteks adalah **110111010001000101000101**, dibagi menjadi tiga buah blok masing-masing berukuran 8 bit:

**11011101**    **00010001**    **01000101**  
P1                    P2                    P3

Kunci yang digunakan adalah  $K = 01010110$

Enkripsi blok pertama, **11011101**, dengan E sebagai berikut:

1.  $P1 + K = 11011101 \oplus 01010110 = 10001011$
2.  $10001011 \ll 1 = 00010111$

Jadi,  $C1 = 00010111$ . Cara yang sama dilakukan untuk P2 dan P3.

Dekripsi blok cipherteks pertama,  $00010111$ , dengan D sebagai berikut:

1.  $00010111 \gg 1 = 10001011$
2.  $10001011 \oplus K = 10001011 \oplus 01010110 = 11011101 = P1$

- Lakukan proses yang sama untuk blok P2, P3, dst.
- Tentu saja algoritma enkripsi E dan D di atas sangat lemah karena mudah dipecahkan.
- *Cipher* blok modern membuat E dan D sangat rumit prosesnya sehingga sangat sukar dipecahkan oleh kriptanalis.
- Fungsi E dan D melibatkan sejumlah teknik seperti teknik substitusi, permutasi, pergeseran (*shifting*), kompresi, ekspansi, dsb.
- Setiap blok tidak dienkripsi satu kali, tetapi berulang kali untuk mendapatkan cipherteks yang kuat.

# Mode Operasi Cipher Blok

- Mode operasi: berkaitan dengan cara blok dioperasikan sebelum dienkripsi/dekripsi oleh fungsi E dan D.
- Ada 5 mode operasi *cipher* blok:
  1. *Electronic Code Book (ECB)*
  2. *Cipher Block Chaining (CBC)*
  3. *Cipher Feedback (CFB)*
  4. *Output Feedback (OFB)*
  5. *Counter Mode*

# 1. *Electronic Code Book (ECB)*

- Setiap blok plainteks  $P_i$  dienkripsi secara individual dan independen dari blok lainnya menjadi blok cipherteks  $C_i$ .

- Enkripsi:  $C_i = E_K(P_i)$

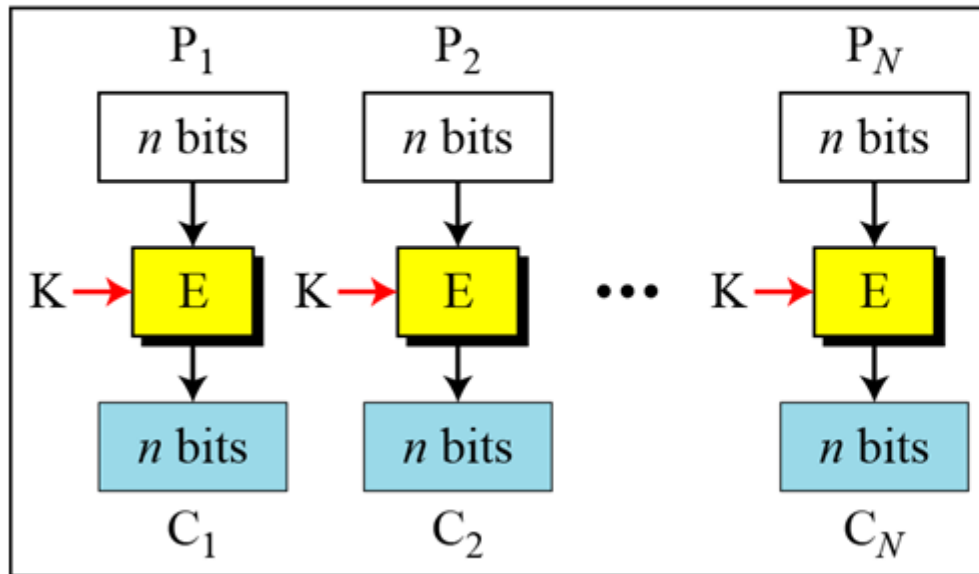
Dekripsi:  $P_i = D_K(C_i)$

yang dalam hal ini,  $P_i$  dan  $C_i$  masing-masing blok plainteks dan cipherteks ke- $i$ .

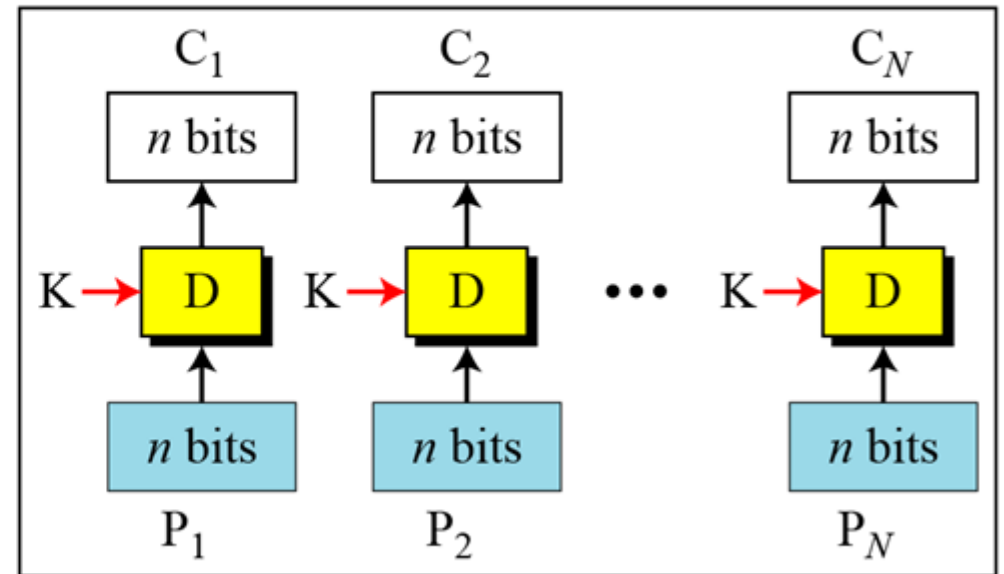


# Mode ECB

E: Encryption      D: Decryption  
 $P_i$ : Plaintext block  $i$      $C_i$ : Ciphertext block  $i$   
K: Secret key



Encryption



Decryption

- Contoh:

Plainteks: 10100010001110101001

Bagi plaintext menjadi blok-blok 4-bit:

1010 0010 0011 1010 1001

( dalam notasi HEX :A23A9)

- Kunci (juga 4-bit): 1011

- Misalkan fungsi enkripsi  $E$  yang sederhana algoritmanya sbb:

1. XOR-kan blok plaintext  $P_i$  dengan  $K$
2. geser secara *wrapping* bit-bit dari hasil langkah 1 satu posisi ke kiri

$$E_K(P) = (P \oplus K) \ll 1$$

$$E_K(P) = (P \oplus K) \lll 1$$

Enkripsi:

	1010	0010	0011	1010	1001	
	1011	1011	1011	1011	1011	$\oplus$
Hasil <i>XOR</i> :	0001	1001	1000	0001	0010	
Geser 1 bit ke kiri:	0010	0011	0001	0010	0100	
Dalam notasi HEX:	2	3	1	2	4	

Jadi, hasil enkripsi plainteks

10100010001110101001      (A23A9 dalam notasi HEX)

adalah

00100011000100100100      (23124 dalam notasi HEX)

- Pada mode ECB, blok plainteks yang sama selalu dienkripsi menjadi blok cipherteks yang sama.

	1010	0010	0011	1010	1001	
	1011	1011	1011	1011	1011	$\oplus$
Hasil <i>XOR</i> :	0001	1001	1000	0001	0010	
Geser 1 bit ke kiri:	0010	0011	0001	0010	0100	
Dalam notasi HEX:	2	3	1	2	4	

- Pada contoh di atas, blok 1010 muncul dua kali dan selalu dienkripsi menjadi 0010.

- Karena setiap blok plainteks yang sama selalu dienkripsi menjadi blok cipherteks yang sama, maka secara teoritis dimungkinkan membuat buku kode plainteks dan cipherteks yang berkoresponden (asal kata “*code book*” di dalam *ECB* ). Enkripsi/dekripsi dilakukan dengan *look up table*.

Plainteks	Cipherteks
0000	0100
0001	1001
0010	1010
...	...
1111	1010

- Untuk setiap kunci  $K$  yang berbeda, dibuat buku kode yang berbeda pula. Jika panjang kunci  $n$  bit, maka terdapat  $2^n$  buku kode.

- Jumlah entri (baris) di dalam setiap buku kode untuk adalah  $2^n$ .
- Namun, semakin besar ukuran blok, semakin besar pula ukuran buku kodenya.
- Misalkan jika blok berukuran 64 bit, maka buku kode terdiri dari  $2^{64}$  buah kode (*entry*), yang berarti terlalu besar untuk disimpan. Lagipula, setiap kunci mempunyai buku kode yang berbeda.

- Jika panjang plainteks tidak habis dibagi dengan ukuran blok, maka blok terakhir berukuran lebih pendek daripada blok-blok lainnya.
- Untuk itu, kita tambahkan bit-bit *padding* untuk menutupi kekurangan bit blok.
- Misalnya ditambahkan bit 0 semua, atau bit 1 semua, atau bit 0 dan bit 1 berselang-seling.

# Kelebihan Mode *ECB*

- 1. Karena tiap blok plainteks dienkripsi secara independen, maka kita tidak perlu mengenkripsi pesan secara sekuensial/linier.**
  - Kita dapat mengenkripsi 5 blok pertama, kemudian blok-blok di akhir, dan kembali ke blok-blok di tengah dan seterusnya.
  - Mode *ECB* cocok untuk mengenkripsi isi arsip (*file*) yang diakses secara acak, misalnya arsip-arsip basisdata.
  - Jika basisdata dienkripsi dengan mode *ECB*, maka sembarang *record* dapat dienkripsi secara independen dari *record* lainnya (dengan asumsi setiap *record* terdiri dari sejumlah blok diskrit yang sama banyaknya).



- 2. Kesalahan 1 atau lebih bit pada blok cipherteks hanya mempengaruhi cipherteks yang bersangkutan pada proses dekripsi.**

Blok-blok cipherteks lainnya bila didekripsi tidak terpengaruh oleh kesalahan bit cipherteks tersebut.

# Kelemahan Mode ECB

- 1. Karena plainteks sering mengandung bagian yang berulang (sehingga terdapat blok-blok plainteks yang sama), maka enkripsinya menghasilkan blok cipherteks yang sama pula**
  - ➔ contoh berulang: spasi panjang
  - ➔ mudah diserang secara statistik dengan analisis frekuensi

- 2. Pihak lawan dapat memanipulasi cipherteks untuk “membodohi” atau mengelabui penerima pesan.**

Contoh: Seseorang mengirim pesan

`“Uang ditransfer lima satu juta rupiah”`

Andaikan kriptanalisis mengetahui ukuran blok = 2 karakter (16 bit), spasi diabaikan.

Blok-blok cipherteks:

$C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}, C_{13}, C_{14}, C_{15}, C_{16}$

Misalkan kriptanalisis mengetahui posisi pesan yang berkaitan dengan nominal uang, yaitu blok  $C_8$  dan  $C_9$ .

Kriptanalisis membuang blok cipherteks  $C_8$  dan  $C_9$  :

$C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_{10}, C_{11}, C_{12}, C_{13}, C_{14}, C_{15}, C_{16}$

Penerima pesan mendekripsi cipherteks yang sudah dimanipulasi dengan kunci yang benar menjadi

**Uang ditransfer satu juta rupiah**

Karena dekripsi menghasilkan pesan yang bermakna, maka penerima menyimpulkan bahwa uang yang dikirim kepadanya sebesar satu juta rupiah.

- Cara mengatasi kelemahan ini: enkripsi tiap blok individual bergantung pada semua blok-blok sebelumnya.
- Prinsip ini mendasari mode *Cipher Block Chaining* (CBC).

# ***Cipher Block Chaining(CBC)***

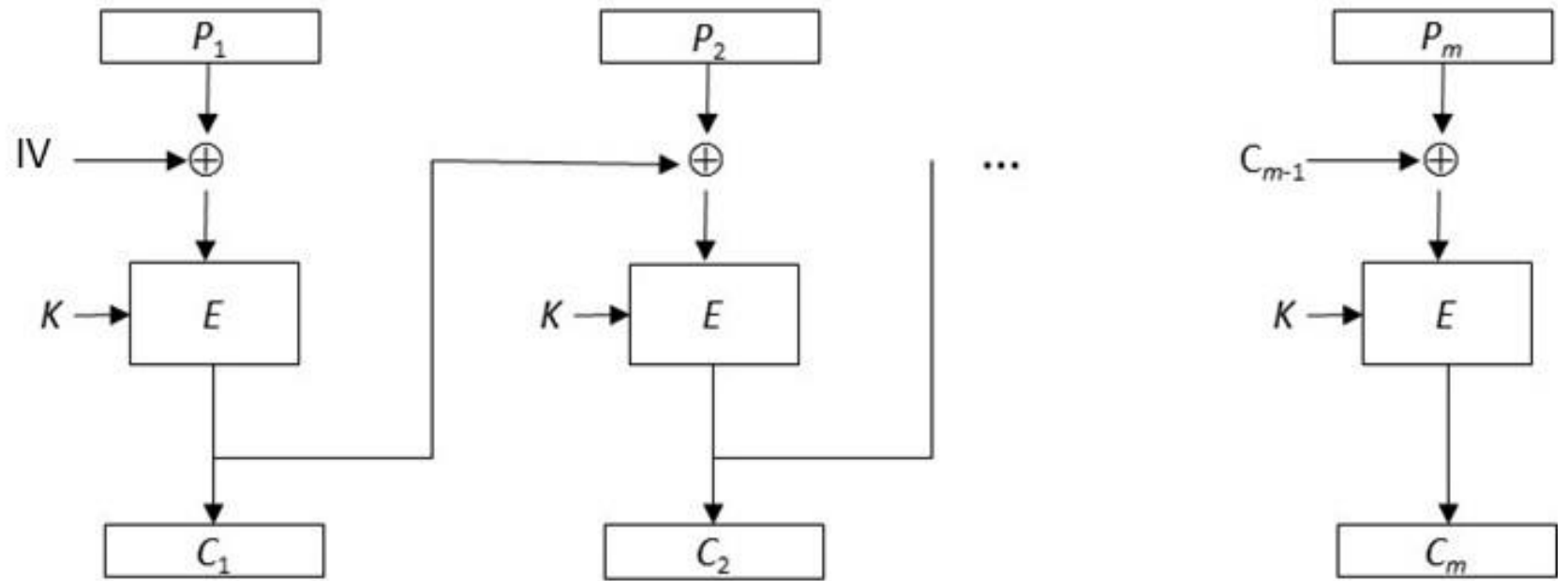
- Tujuan: membuat ketergantungan antar blok (mengatasi kelemahan mode ECB).
- Setiap blok cipherteks bergantung tidak hanya pada blok plainteksnya tetapi juga pada seluruh blok plainteks sebelumnya.
- Hasil enkripsi blok sebelumnya di-umpan-balikkan ke dalam enkripsi blok yang *current*.
- Enkripsi blok pertama memerlukan blok semu ( $C_0$ ) yang disebut *IV (initialization vector)*.
- *IV* dapat diberikan oleh pengguna atau dibangkitkan secara acak oleh program.
- Pada dekripsi, blok plainteks diperoleh dengan cara meng-*XOR*-kan *IV* dengan hasil dekripsi terhadap blok cipherteks pertama.

(a) Skema enkripsi mode CBC

**Encryption:**

$$C_0 = IV$$

$$C_i = E_K(P_i \oplus C_{i-1})$$

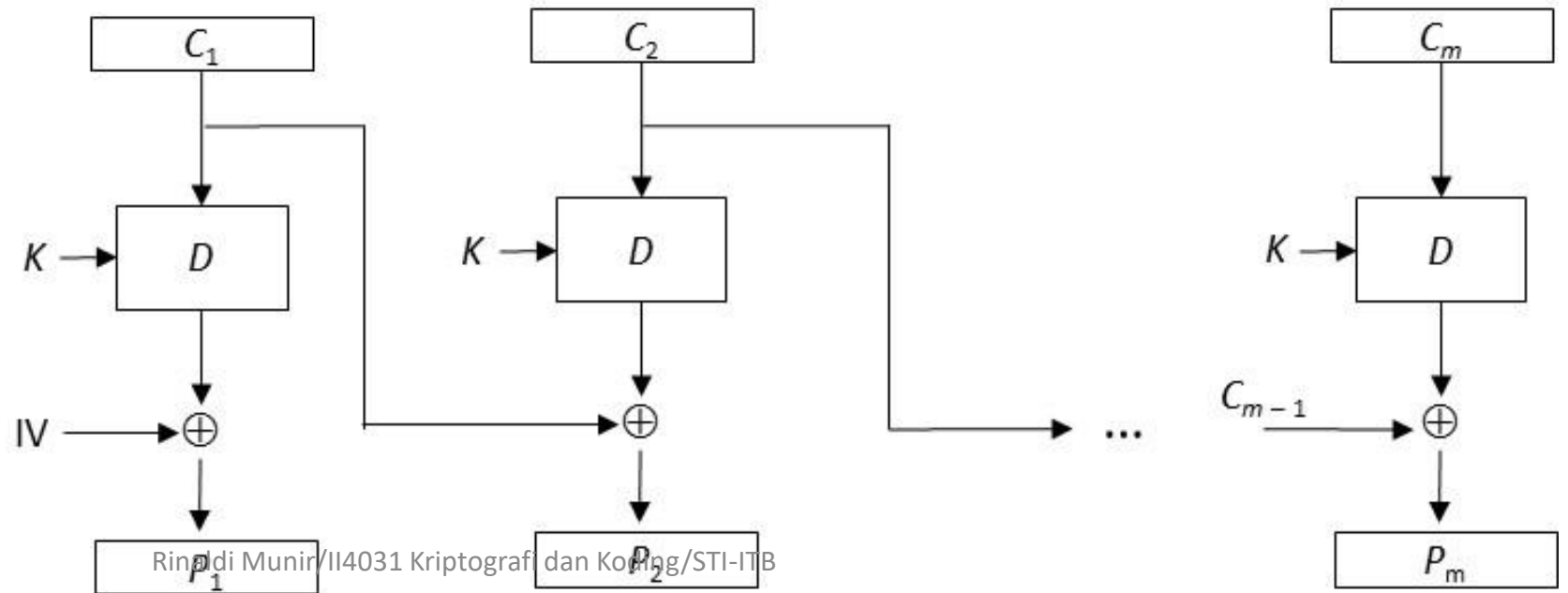


(b) Skema dekripsi mode CBC

**Decryption:**

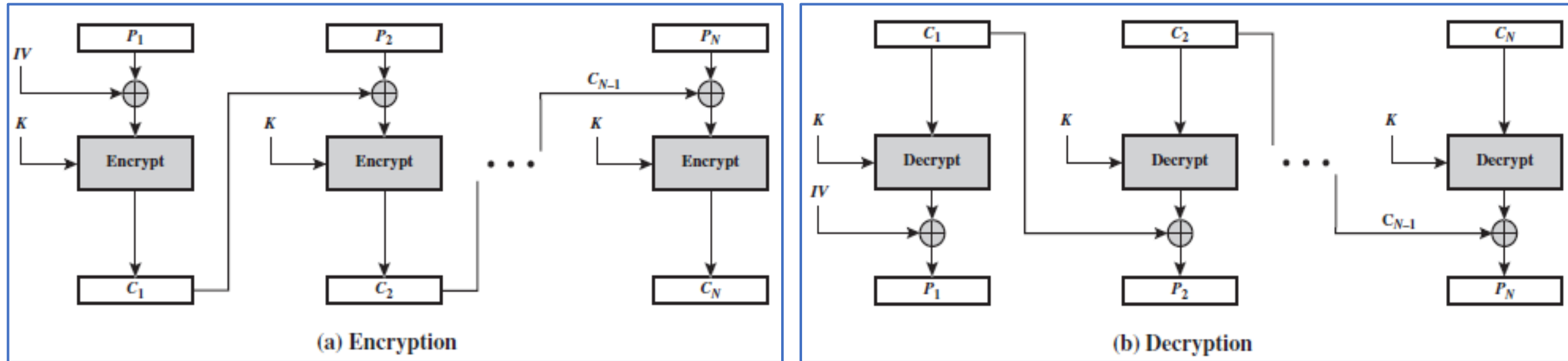
$$C_0 = IV$$

$$P_i = D_K(C_i) \oplus C_{i-1}$$





# Mode CBC



CBC	$C_1 = E(K, [P_1 \oplus IV])$	$P_1 = D(K, C_1) \oplus IV$
	$C_j = E(K, [P_j \oplus C_{j-1}]) \quad j = 2, \dots, N$	$P_j = D(K, C_j) \oplus C_{j-1} \quad j = 2, \dots, N$

**Contoh 9.8.** Tinjau kembali plainteks dari Contoh 9.6:

10100010001110101001

Bagi plainteks menjadi blok-blok yang berukuran 4 bit:

1010 0010 0011 1010 1001

atau dalam notasi HEX adalah A23A9.

Misalkan kunci ( $K$ ) yang digunakan adalah (panjangnya juga 4 bit)

1011

atau dalam notasi HEX adalah B. Sedangkan  $IV$  yang digunakan seluruhnya bit 0 (Jadi,  $C_0 = 0000$ )

Fungsi enkripsi  $E$  yang digunakan sama seperti sebelumnya: XOR-kan blok plainteks  $P_i$  dengan  $K$ , kemudian geser secara *wrapping* bit-bit dari  $P_i \oplus K$  satu posisi ke kiri.

$$E_K(P) = (P \oplus K) \ll 1$$

$C_1$  diperoleh sebagai berikut:

$$P_1 \oplus C_0 = 1010 \oplus 0000 = 1010$$

Enkripsikan hasil ini dengan fungsi  $E$  sbb:

$$1010 \oplus K = 1010 \oplus 1011 = 0001$$

Geser (*wrapping*) hasil ini satu bit ke kiri: 0010

Jadi,  $C_1 = 0010$  (atau 2 dalam HEX)

$C_2$  diperoleh sebagai berikut:

$$P_2 \oplus C_1 = 0010 \oplus 0010 = 0000$$

$$0000 \oplus K = 0000 \oplus 1011 = 1011$$

Geser (*wrapping*) hasil ini satu bit ke kiri: 0111

Jadi,  $C_2 = 0111$  (atau 7 dalam HEX)

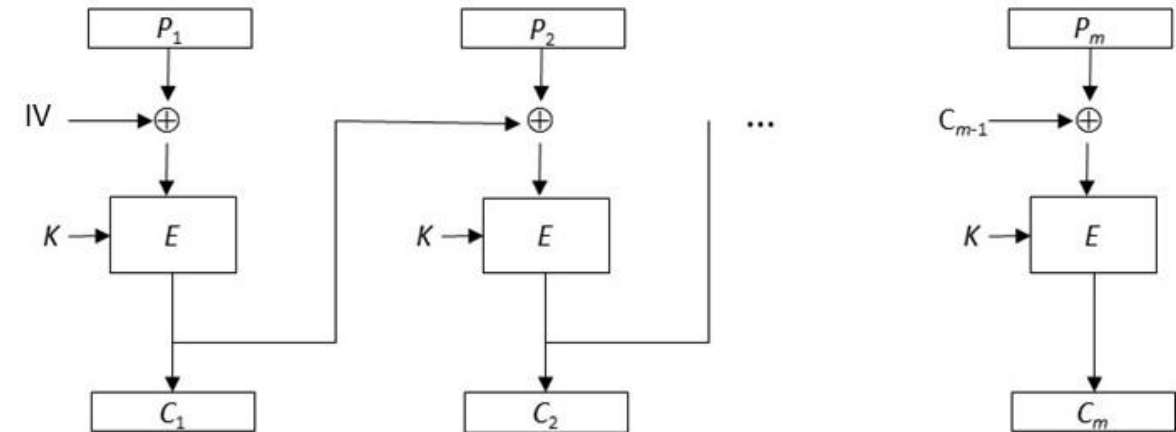
$C_3$  diperoleh sebagai berikut:

$$P_3 \oplus C_2 = 0011 \oplus 0111 = 0100$$

$$0100 \oplus K = 0100 \oplus 1011 = 1111$$

Geser (*wrapping*) hasil ini satu bit ke kiri: 1111

Jadi,  $C_3 = 1111$  (atau F dalam HEX)



Demikian seterusnya, sehingga plainteks dan cipherteks hasilnya adalah:

Pesan (plainteks):                   A23A9

Cipherteks (mode *ECB*):           23124

Cipherteks (mode *CBC*):           27FBF

## *Kelebihan Mode CBC*

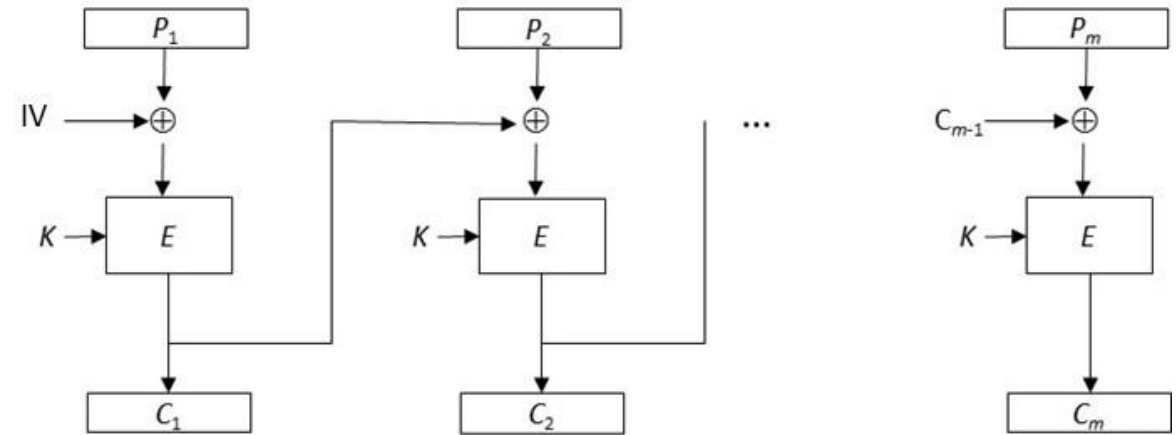
Blok-blok plainteks yang sama tidak selalu menghasilkan blok-blok cipherteks yang sama

Oleh karena blok-blok plainteks yang sama tidak menghasilkan blok-blok cipherteks yang sama, maka kriptanalisis menjadi lebih sulit.

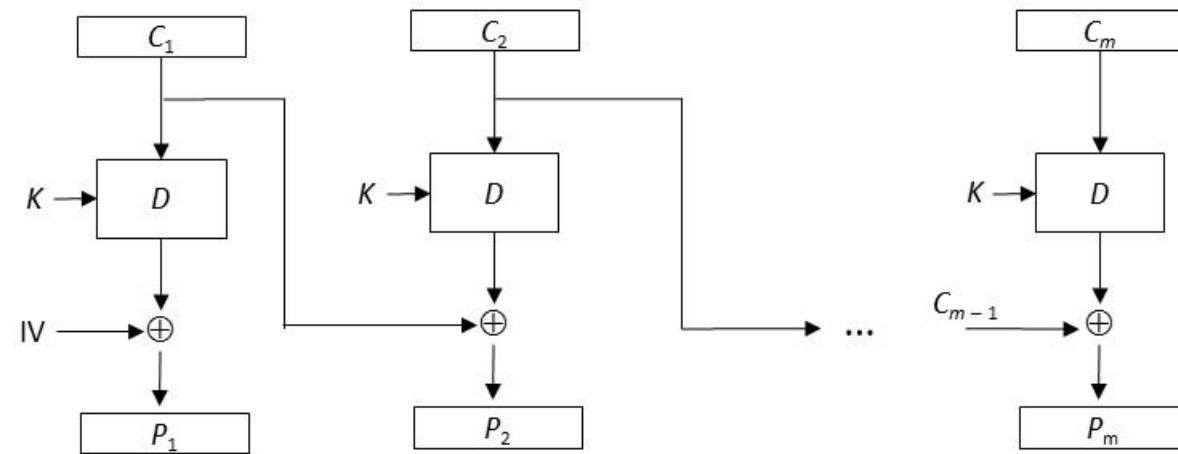
Inilah alasan utama penggunaan mode *CBC*.

# Kelemahan Mode CBC

1. Kesalahan satu bit pada sebuah blok plainteks akan menghasilkan kesalahan pada blok cipherteks yang berkoresponden dan kesalahan tersebut merambat ke semua blok cipherteks berikutnya.



2. Tetapi, hal ini berkebalikan pada proses dekripsi. Kesalahan satu bit pada blok cipherteks hanya mempengaruhi blok plainteks yang berkoresponden dan satu bit pada blok plainteks berikutnya (pada posisi bit yang berkoresponden pula).



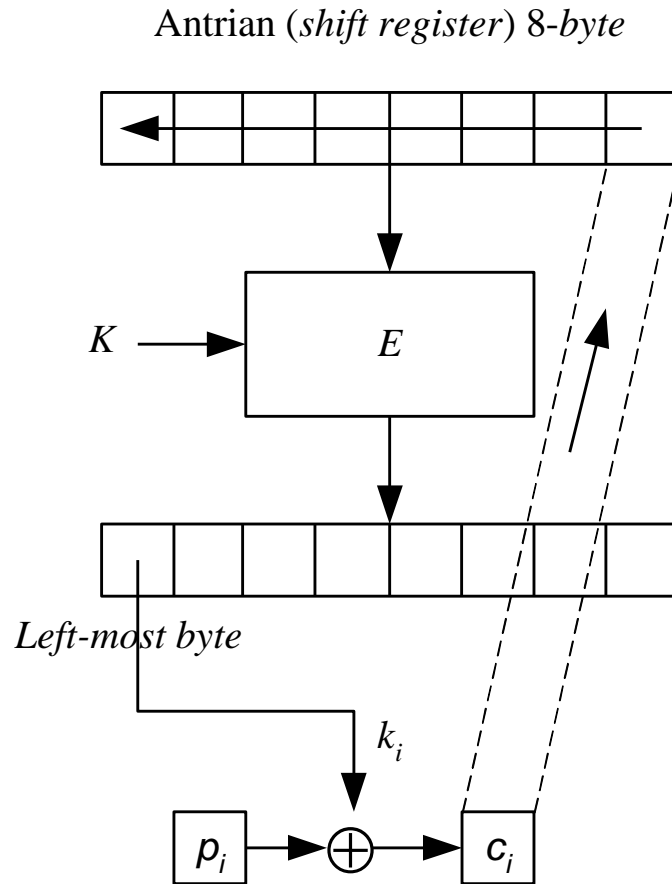
# *Cipher-Feedback (CFB)*

- Mengatasi kekurangan pada mode *CBC* apabila diterapkan pada pengiriman data yang belum mencapai ukuran satu blok
- Data dienkripsi dalam unit yang lebih kecil daripada ukuran blok.
- Unit data yang dienkripsi panjangnya bisa 1 bit, 2 bit, 4-bit, 8 bit, dan lain-lain.
- Bila unit yang dienkripsi adalah satu karakter setiap kalinya, maka mode *CFB*-nya disebut *CFB 8-bit*.

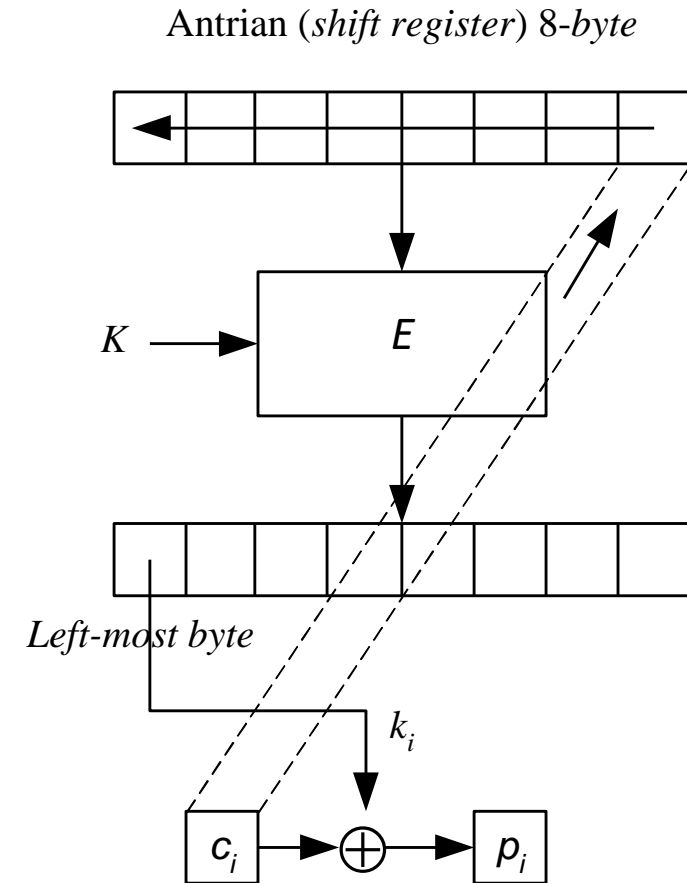
- *CFB*  $r$ -bit mengenkripsi plainteks sebanyak  $r$  bit setiap kalinya,  $r \leq n$  ( $n$  = ukuran blok).
- Dengan kata lain, *CFB*  $r$ -bit memperlakukan *cipher* blok sama seperti pada *cipher* alir.
- Mode *CFB* membutuhkan sebuah antrian (*queue*) yang berukuran sama dengan ukuran blok masukan.
- Tinjau mode *CFB* 8-bit yang bekerja pada blok berukuran 64-bit (setara dengan 8 *byte*) pada gambar berikut



# Mode CFB-8 bit untuk ukuran blok 64 bit (8 byte)



(a) Enkripsi



(b) Dekripsi

Secara matematis, mode *CFB*  $r$ -bit dapat dinyatakan sebagai:

Enkripsi:

$$C_i = P_i \oplus MSB_r(E_K(X_i))$$

$$X_{i+1} = LSB_{n-r}(X_i) || C_i$$

Dekripsi:

$$P_i = C_i \oplus MSB_r(E_K(X_i))$$

$$X_{i+1} = LSB_{n-r}(X_i) || C_i$$

yang dalam hal ini,

$$i = 1, 2, 3, \dots, m.$$

$X_i$  = isi antrian dengan  $X_1$  adalah *IV*

$E$  = fungsi enkripsi

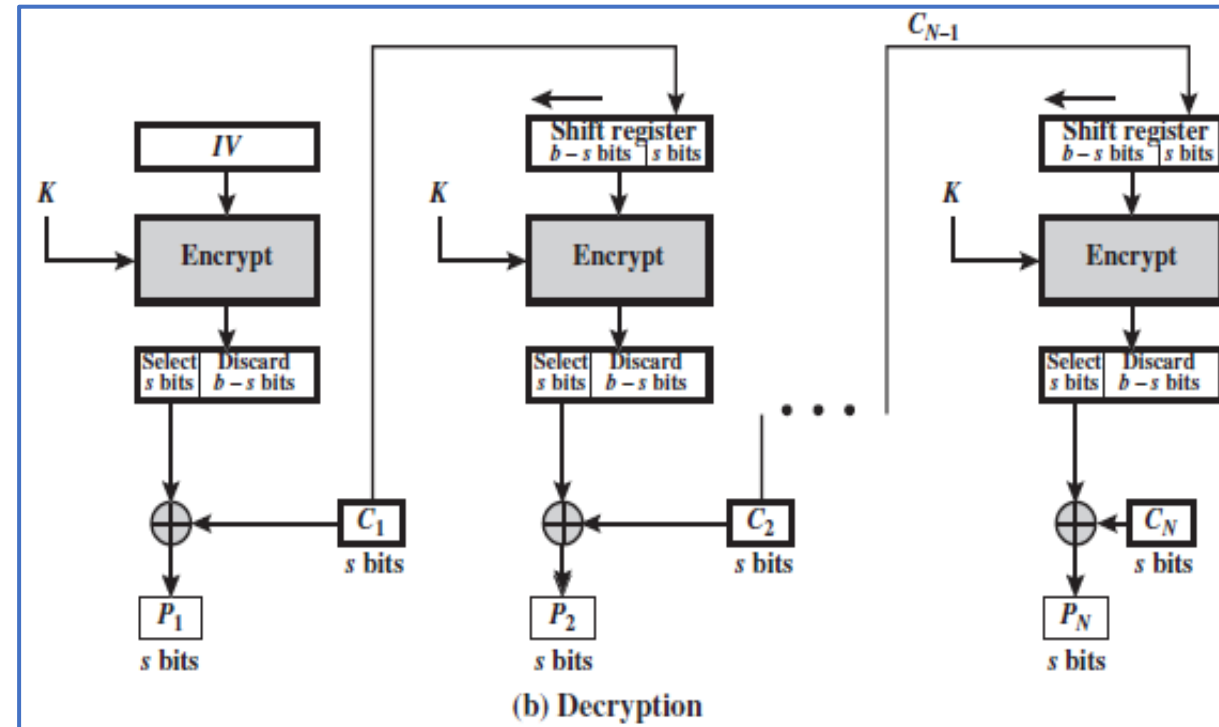
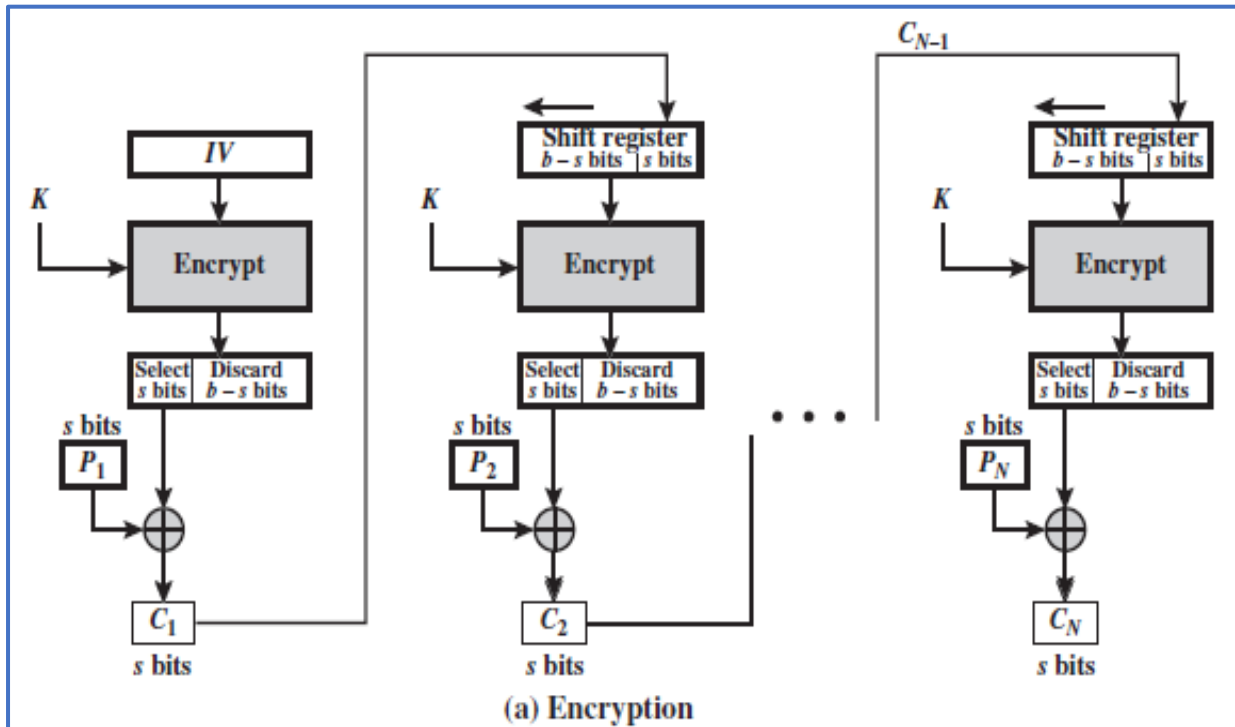
$K$  = kunci

$n$  = panjang blok enkripsi

$r$  = panjang unit enkripsi

$||$  = operator penyambungan (*concatenation*)

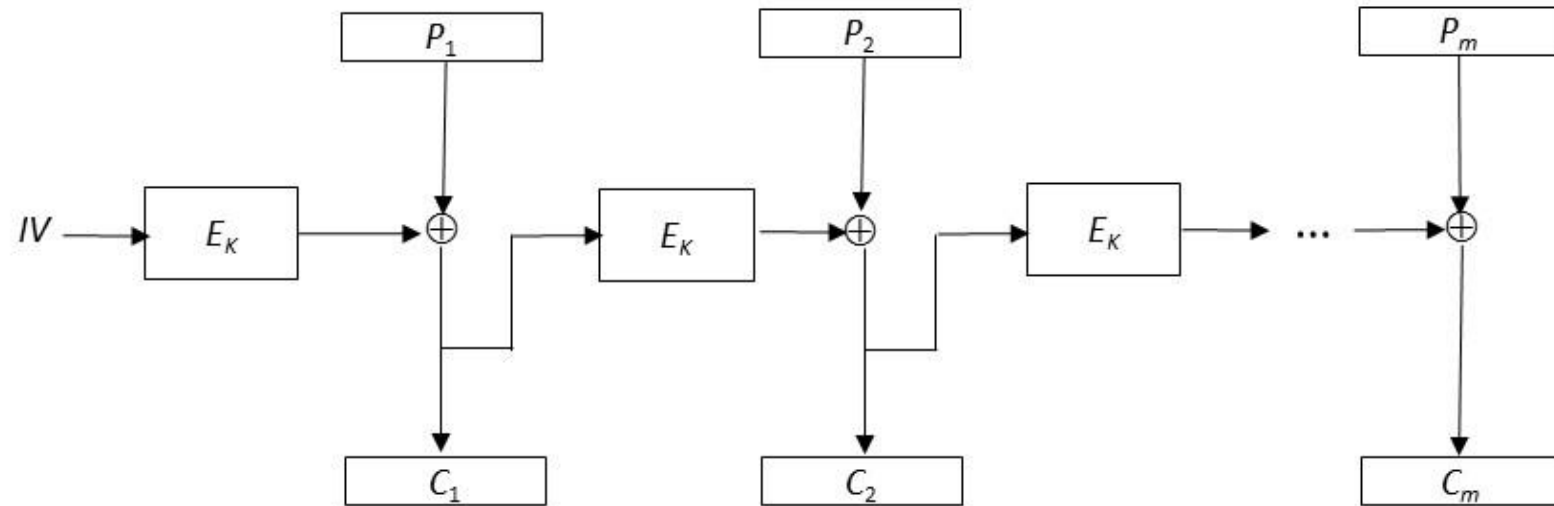
# Mode CFB



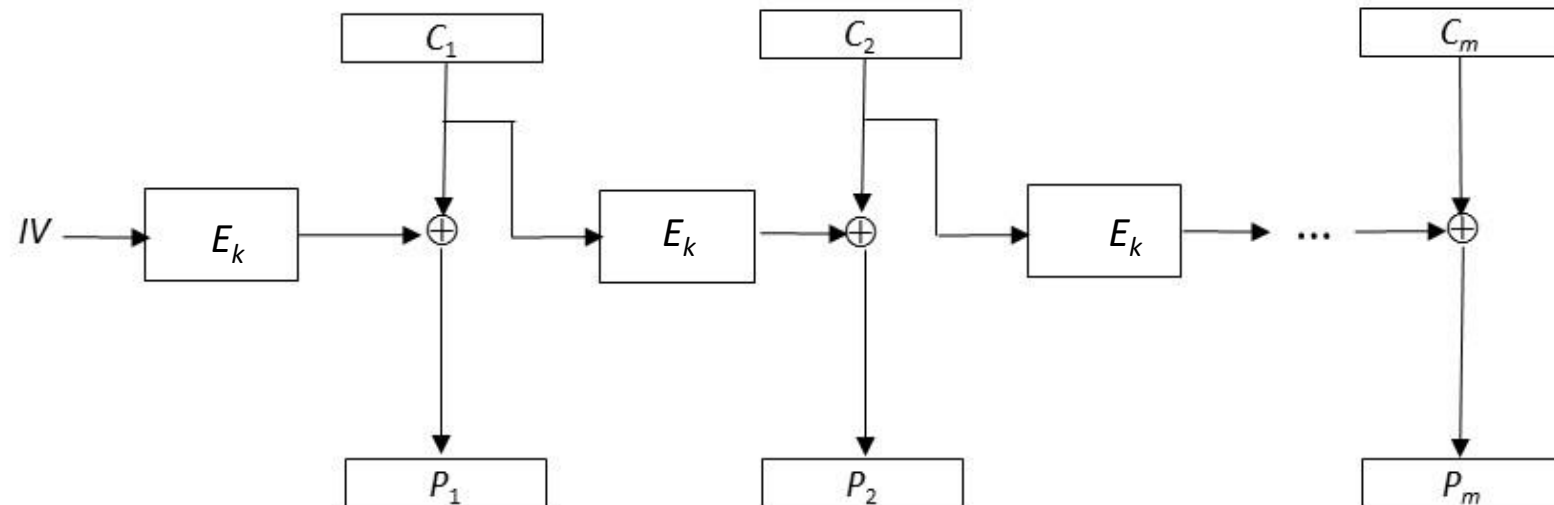
CFB	$I_1 = IV$	$I_1 = IV$
	$I_j = \text{LSB}_{b-s}(I_{j-1}) \parallel C_{j-1} \quad j = 2, \dots, N$	$I_j = \text{LSB}_{b-s}(I_{j-1}) \parallel C_{j-1} \quad j = 2, \dots, N$
	$O_j = E(K, I_j) \quad j = 1, \dots, N$	$O_j = E(K, I_j) \quad j = 1, \dots, N$
	$C_j = P_j \oplus \text{MSB}_s(O_j) \quad j = 1, \dots, N$	$P_j = C_j \oplus \text{MSB}_s(O_j) \quad j = 1, \dots, N$

- Jika  $n = m$ , maka mode *CFB*  $n$ -bit adalah sbb:

(a) Enkripsi



(b) Dekripsi

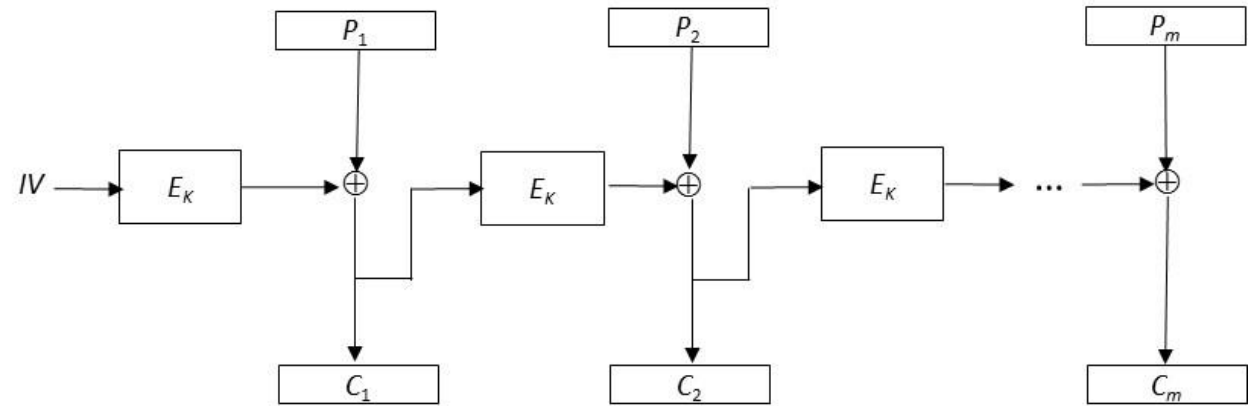


- Dari Gambar tersebut dapat dilihat bahwa:

$$C_i = P_i \oplus E_k (C_{i-1})$$

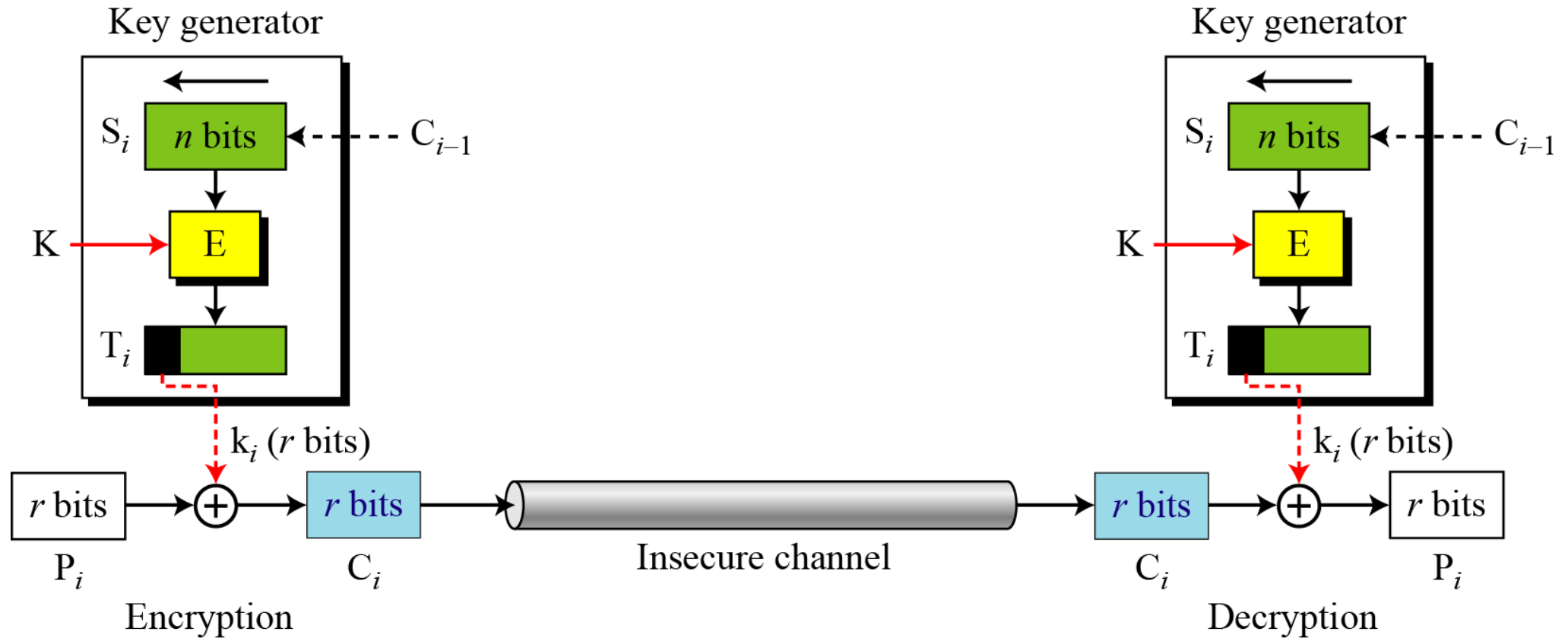
$$P_i = C_i \oplus D_k (C_{i-1})$$

yang dalam hal ini,  $C_0 = IV$ .



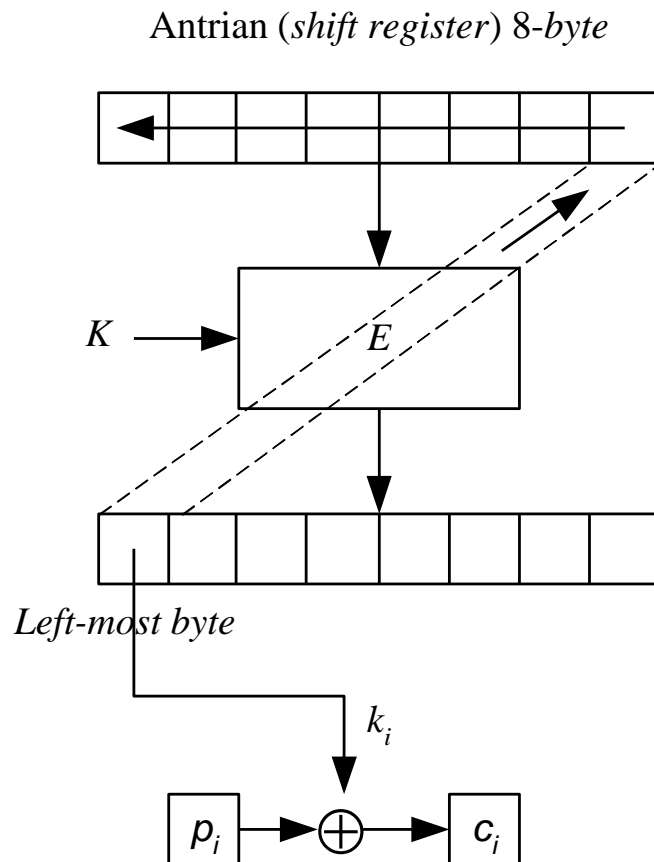
- Kesalahan 1-bit pada blok plainteks akan merambat pada blok-blok cipherteks yang berkoresponden dan blok-blok ciphereks selanjutnya pada proses enkripsi.
- Hal yang kebalikan terjadi pada proses dekripsi.

- CFB  $r$ -bit (misalnya  $r = 1, r = 8$ ) dapat dibuat menjadi *stream cipher*:

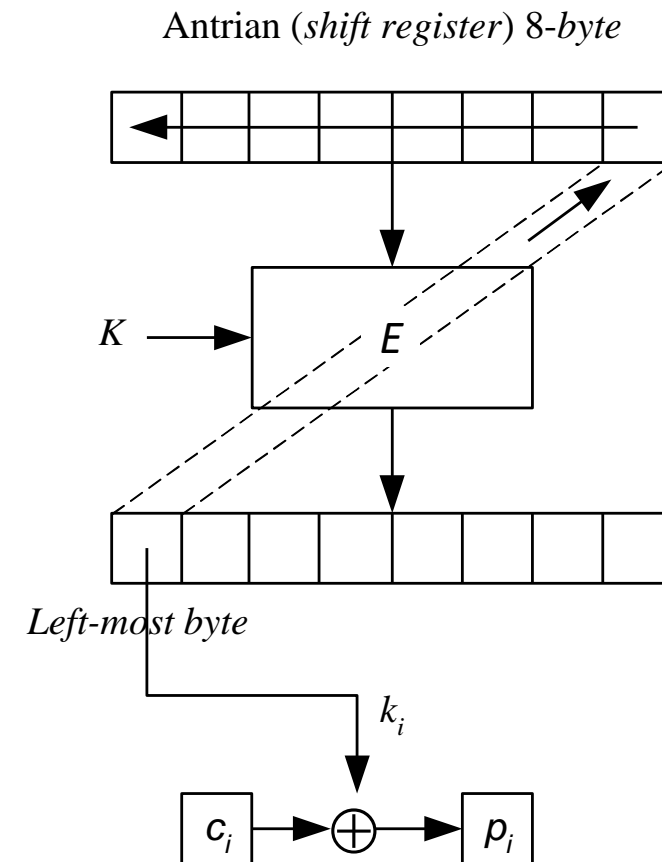


# Output-Feedback (OFB)

- Mode *OFB* mirip dengan mode *CFB*, kecuali  $r$ -bit dari hasil enkripsi antrian disalin menjadi elemen posisi paling kanan di antrian.



(a) Enkripsi



(b) Dekripsi

OFB 8-bit

# Mode OFB

E : Encryption

$P_i$ : Plaintext block  $i$

K: Secret key

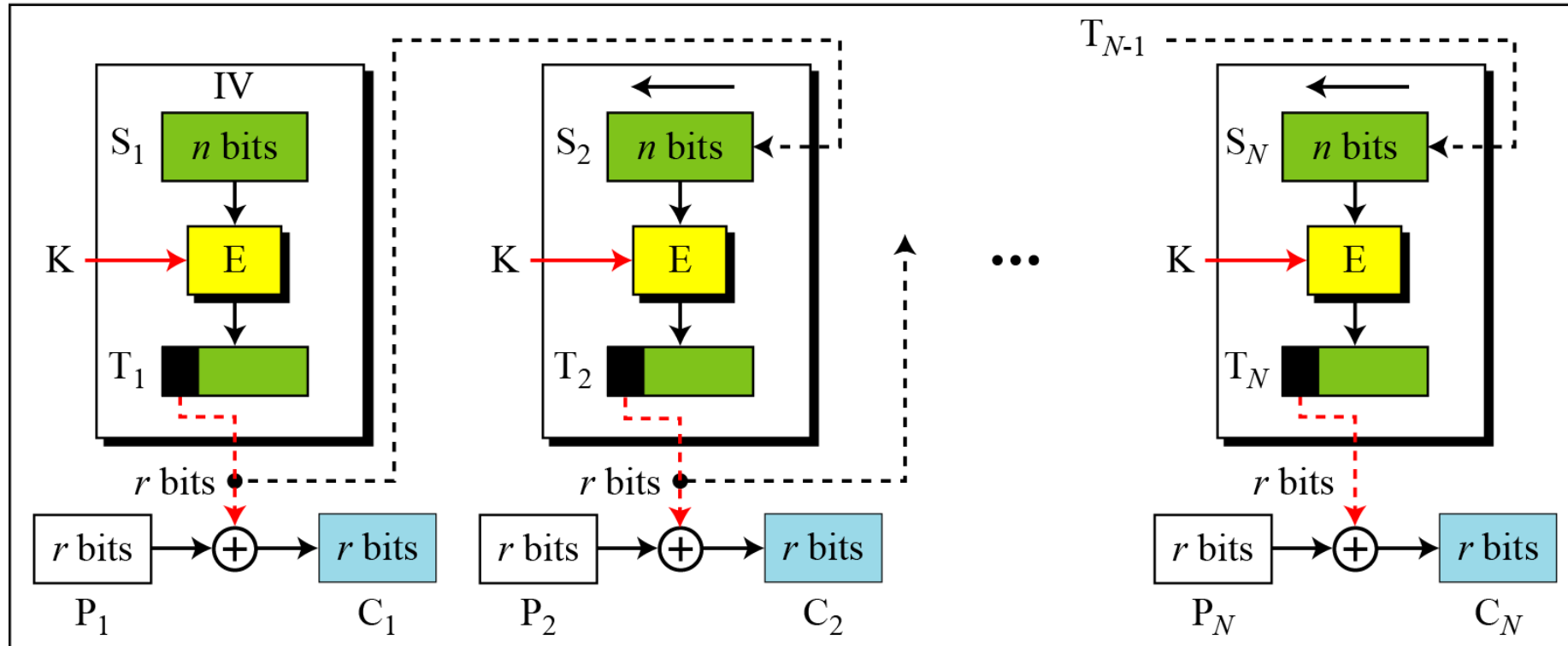
D : Decryption

$C_i$ : Ciphertext block  $i$

IV: Initial vector ( $S_1$ )

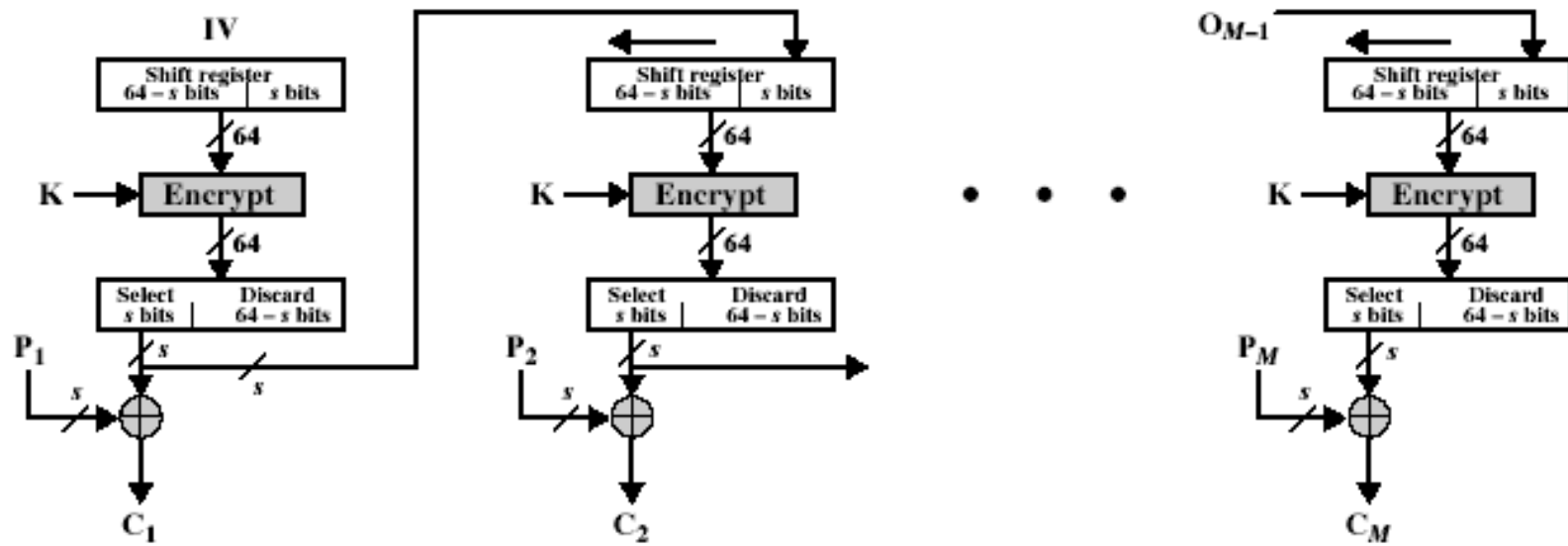
$S_i$ : Shift register

$T_i$ : Temporary register

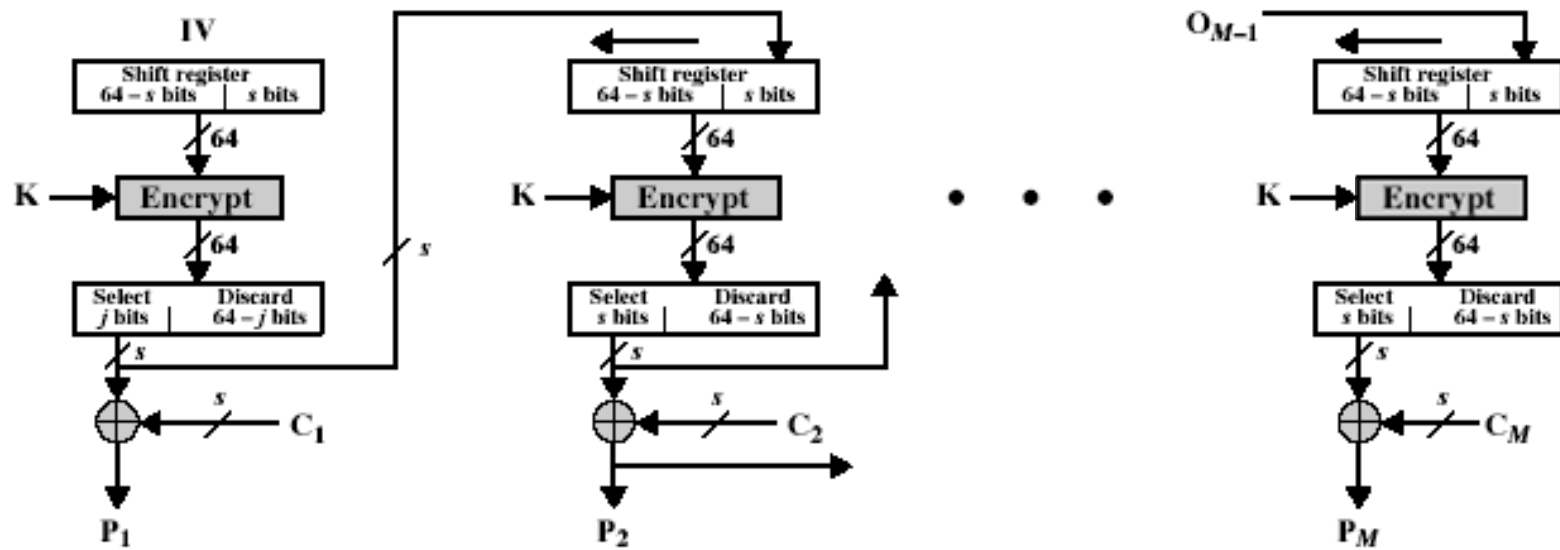


Encryption



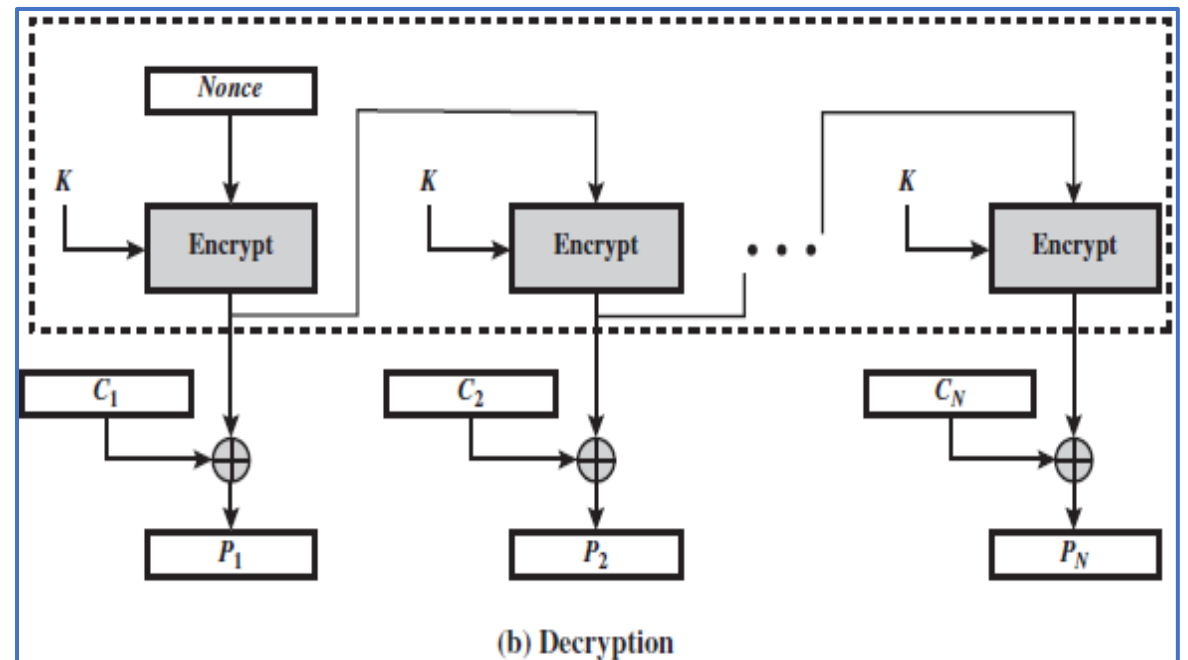
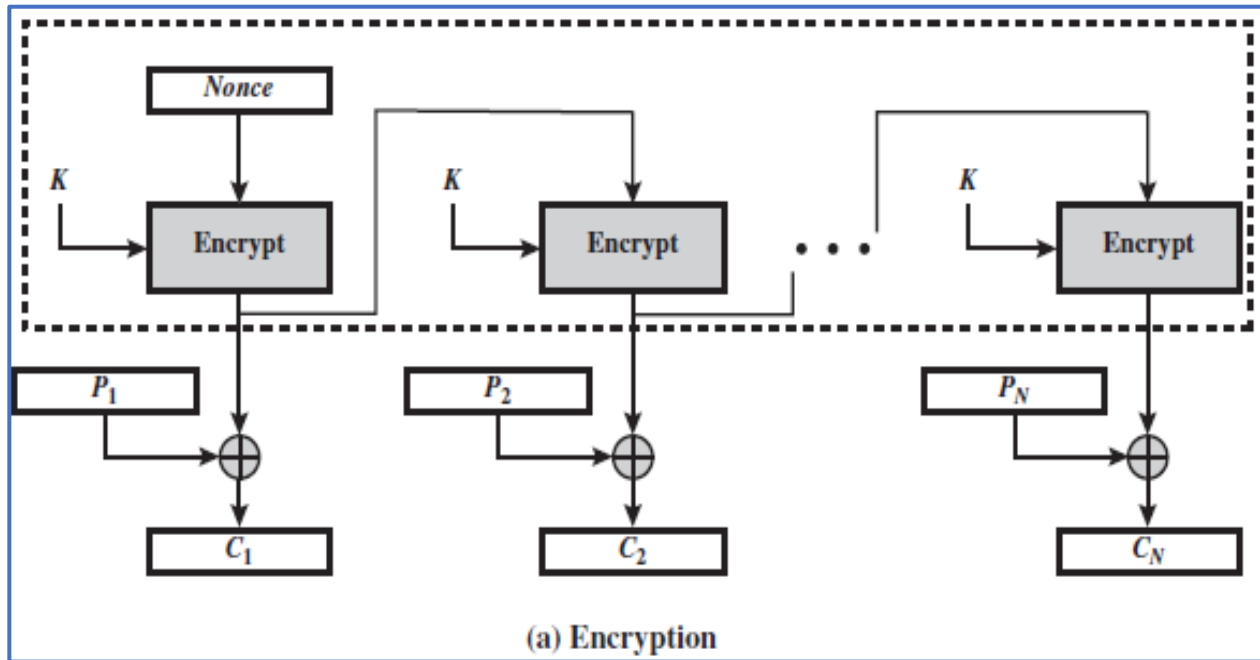


(a) Encryption

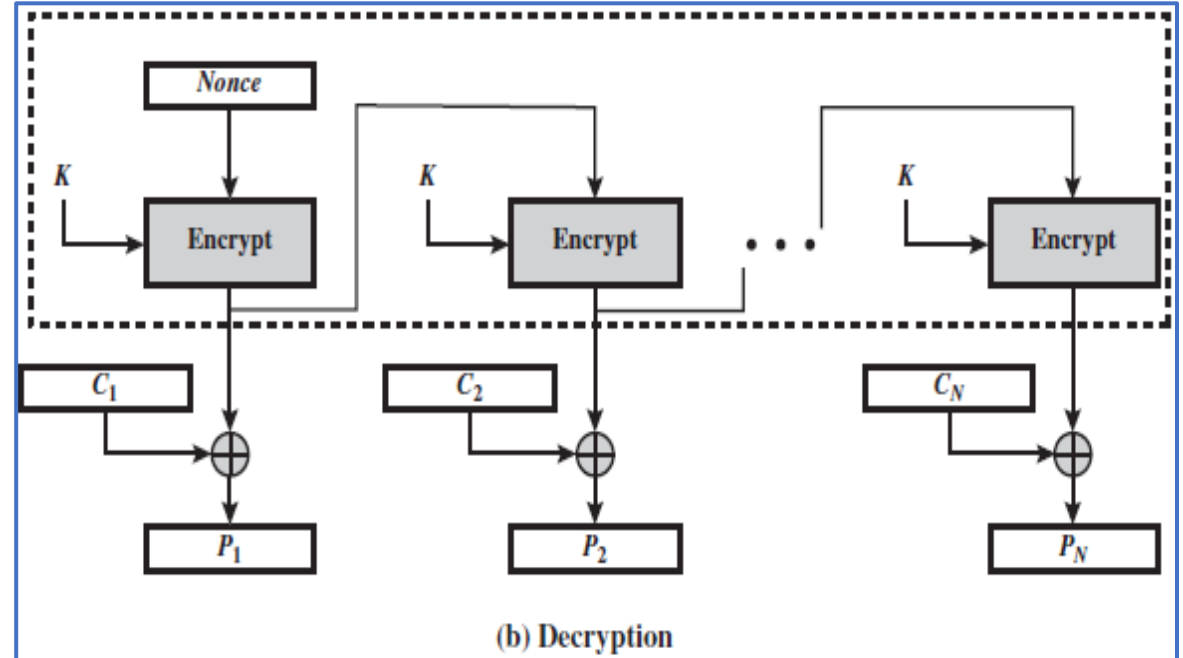
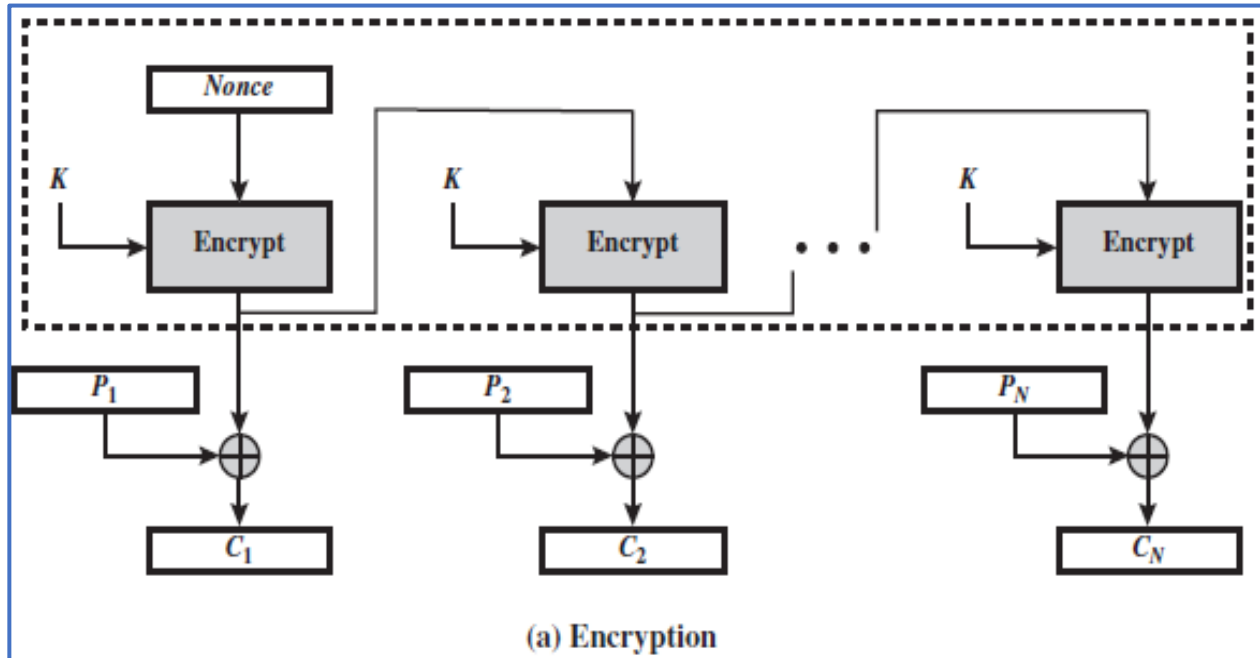


(b) Decryption

Jika  $r = n$ , maka mode *OFB*  $r$ -bit adalah seperti pada Gambar berikut

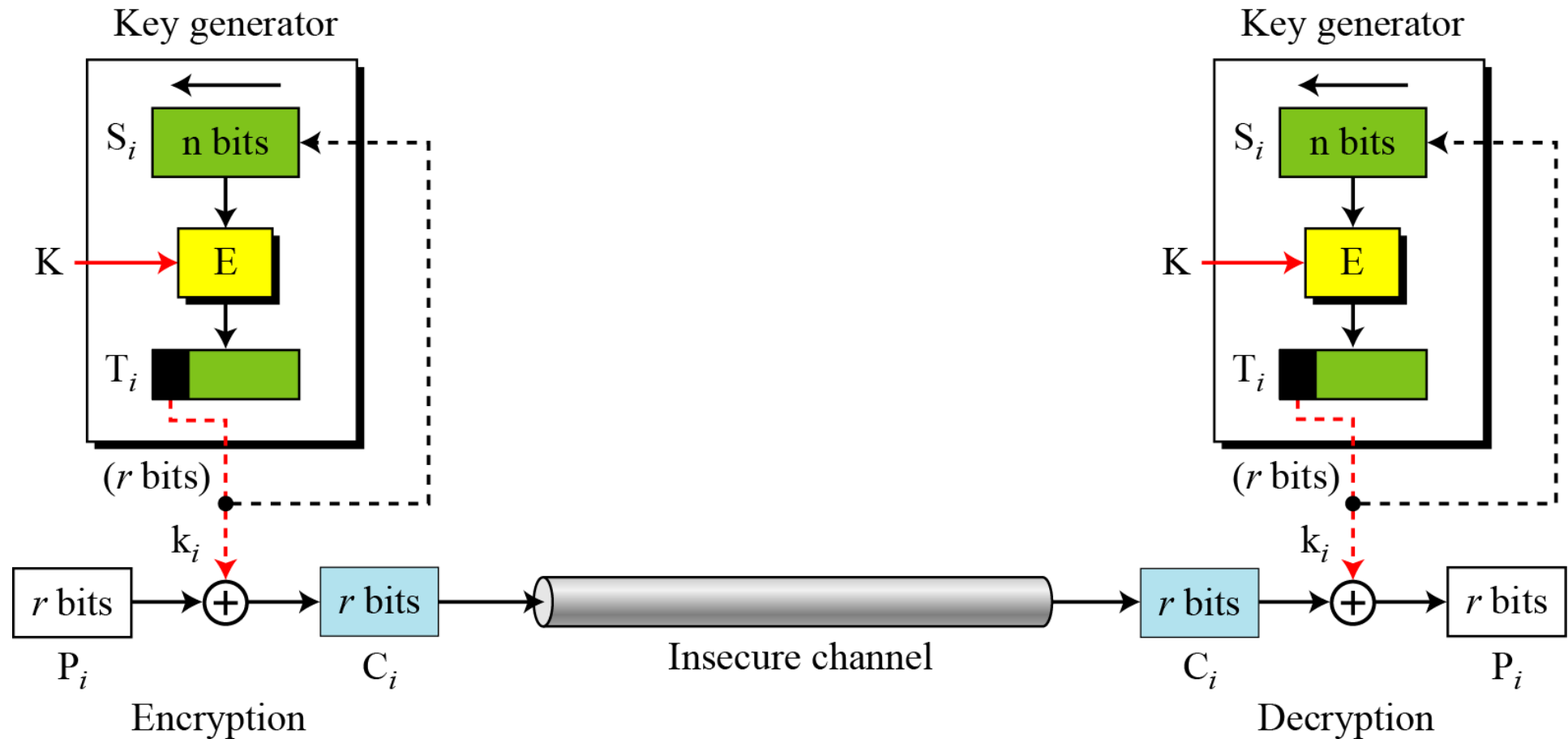


- Kesalahan 1-bit pada blok plainteks hanya mempengaruhi blok cipherteks yang berkoresponden saja; begitu pula pada proses dekripsi, kesalahan 1-bit pada blok cipherteks hanya mempengaruhi blok plainteks yang bersangkutan saja.



- Karakteristik kesalahan semacam ini cocok untuk transmisi analog yang didigitisasi, seperti suara atau video, yang dalam hal ini kesalahan 1-bit dapat ditolerir, tetapi penjarangan kesalahan tidak dibolehkan.

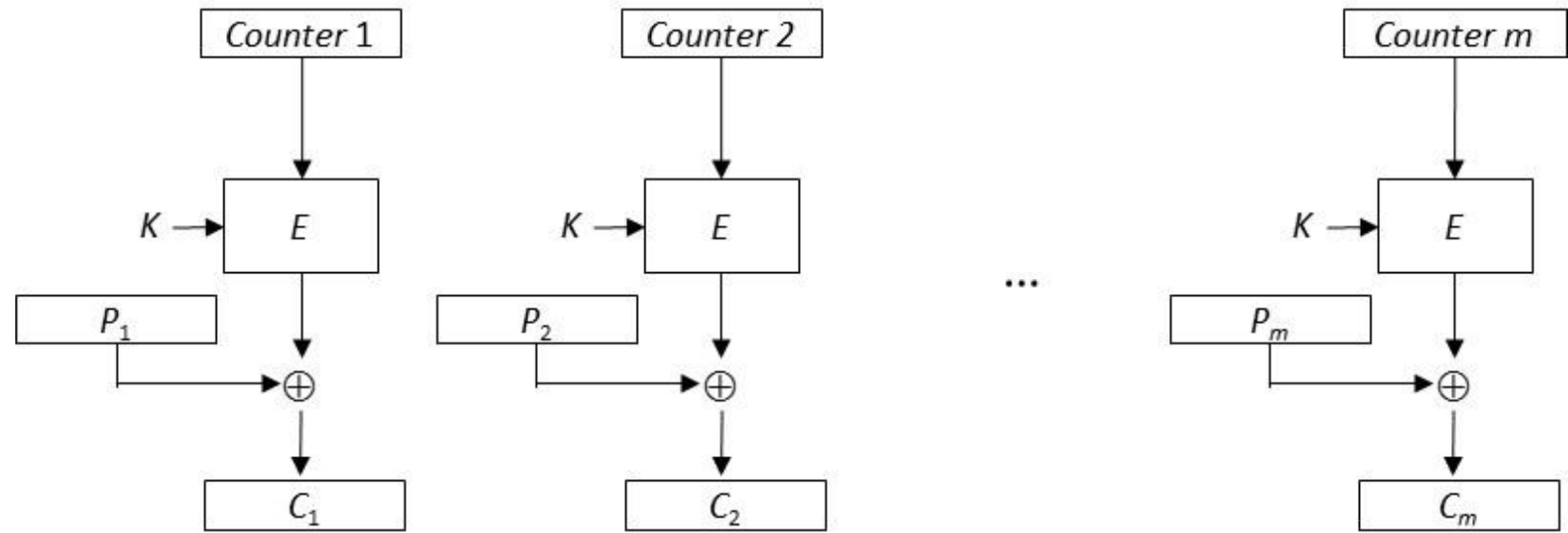
- OFB dapat dibuat menjadi *stream cipher*:



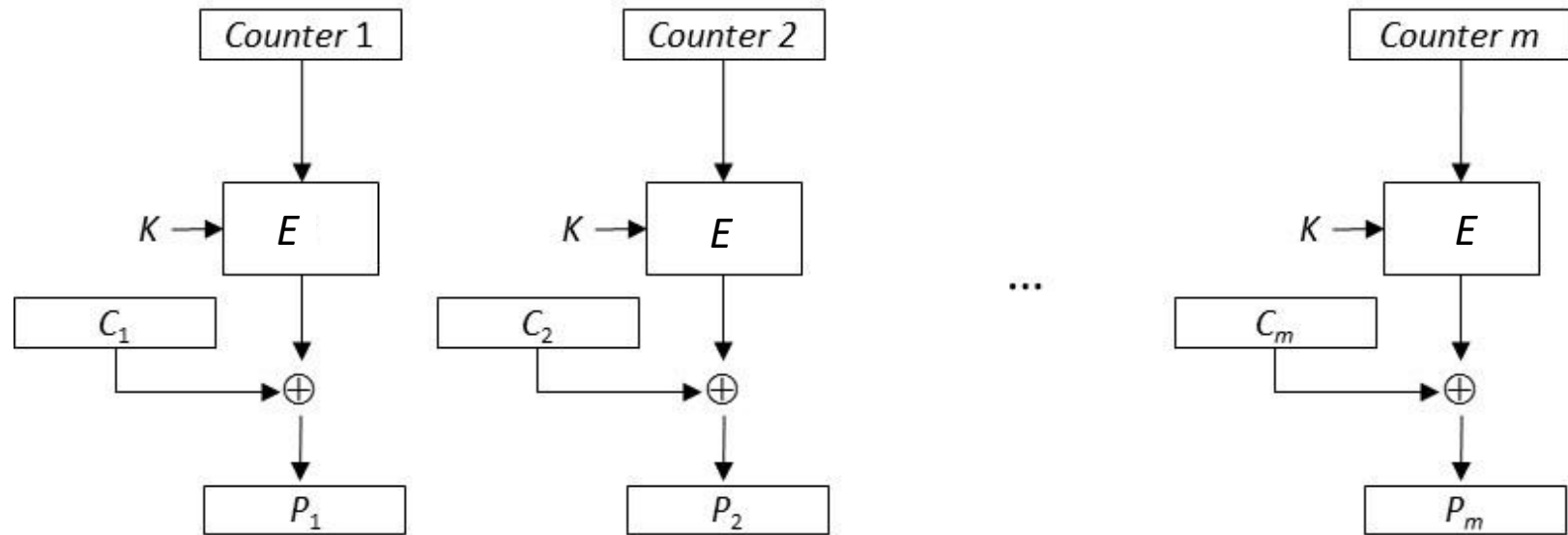
# Counter Mode

- Mode *counter* tidak melakukan perantaraan (*chaining*) seperti pada *CBC*.
- *Counter* adalah sebuah nilai berupa blok bit yang ukurannya sama dengan ukuran blok plainteks.
- Nilai *counter* harus berbeda dari setiap blok yang dienkripsi. Pada mulanya, untuk enkripsi blok pertama, *counter* diinisialisasi dengan sebuah nilai.
- Selanjutnya, untuk enkripsi blok-blok berikutnya *counter* dinaikkan (*increment*) nilainya satu ( $\text{counter} \leftarrow \text{counter} + 1$ ).

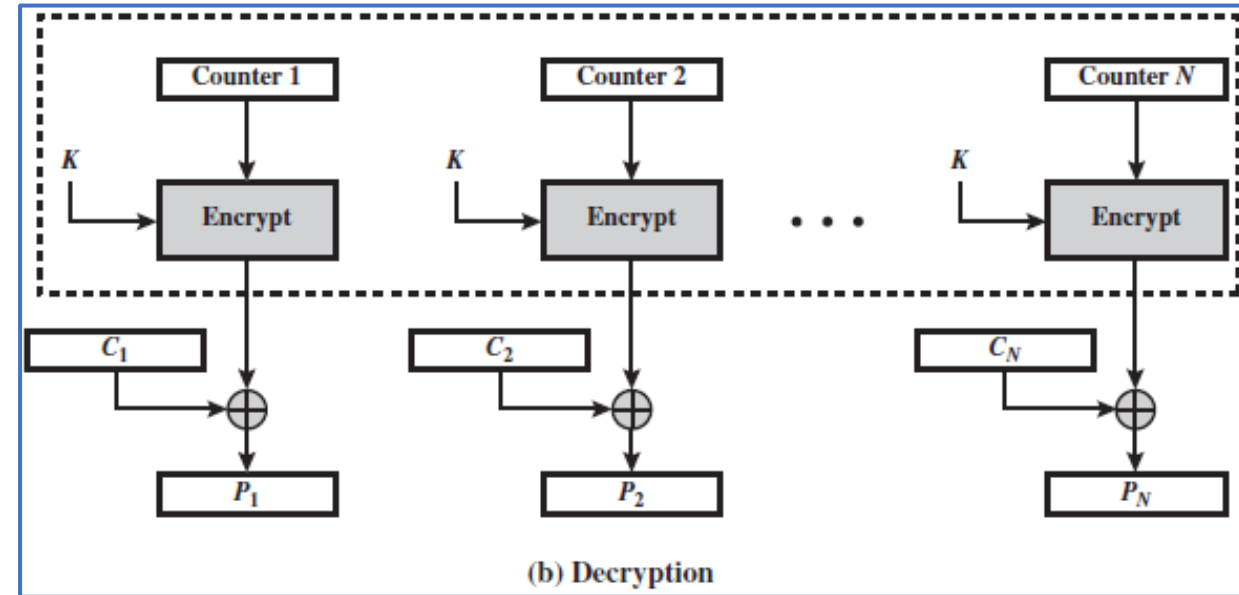
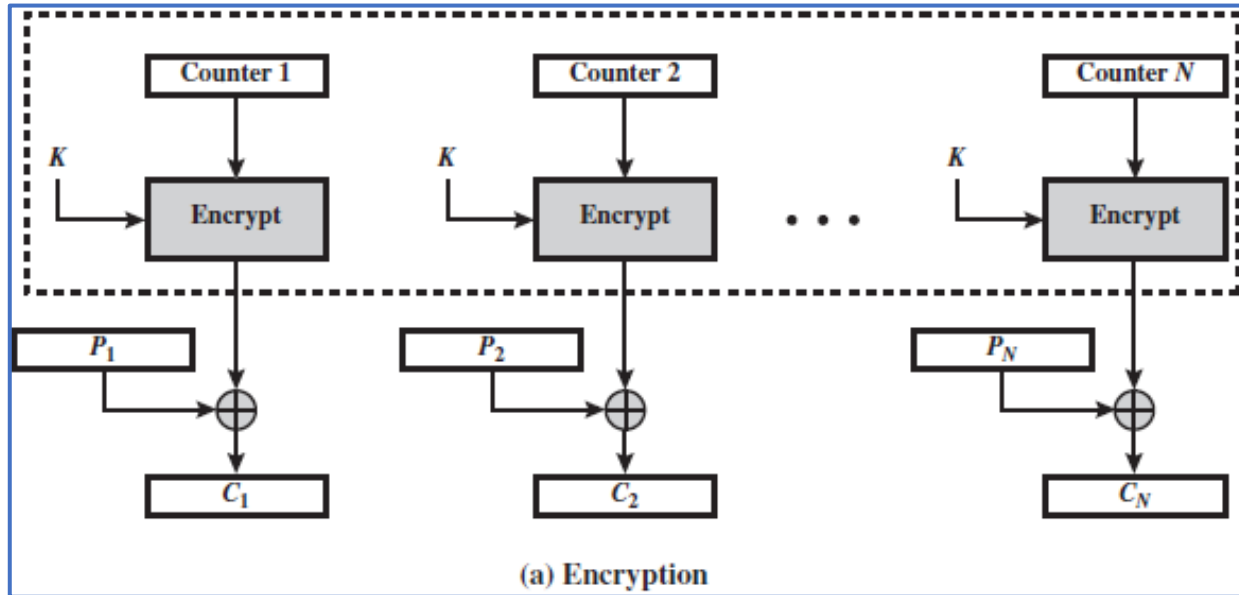
(a) Enkripsi



(b) Dekripsi



# Mode Counter



CTR	$C_j = P_j \oplus E(K, T_j) \quad j = 1, \dots, N - 1$	$P_j = C_j \oplus E(K, T_j) \quad j = 1, \dots, N - 1$
	$C_N^* = P_N^* \oplus \text{MSB}_m[E(K, T_N)]$	$P_N^* = C_N^* \oplus \text{MSB}_m[E(K, T_N)]$

$T_j$  adalah counter, nilainya bertambah satu setiap kali enkripsi blok

- Kelebihan:
  - Hanya perlu algoritma enkripsi saja
  - Dapat mengenkripsi blok data secara acak (tidak harus sekuensial)
    - Blok plainteks/cipherteks dapat diproses(enkripsi atau dekrpsi) secara paralal
  - Sederhanan, enkripsi dan dekripsi cepat
- Counter haruslah:
  - Tidak diketahui atau tidak dapat diprediksi nilainya



- Contoh: tinjau kembali contoh sebelumnya

Plainteks: 1010 0010 0011 1010 1001 (A23A9)

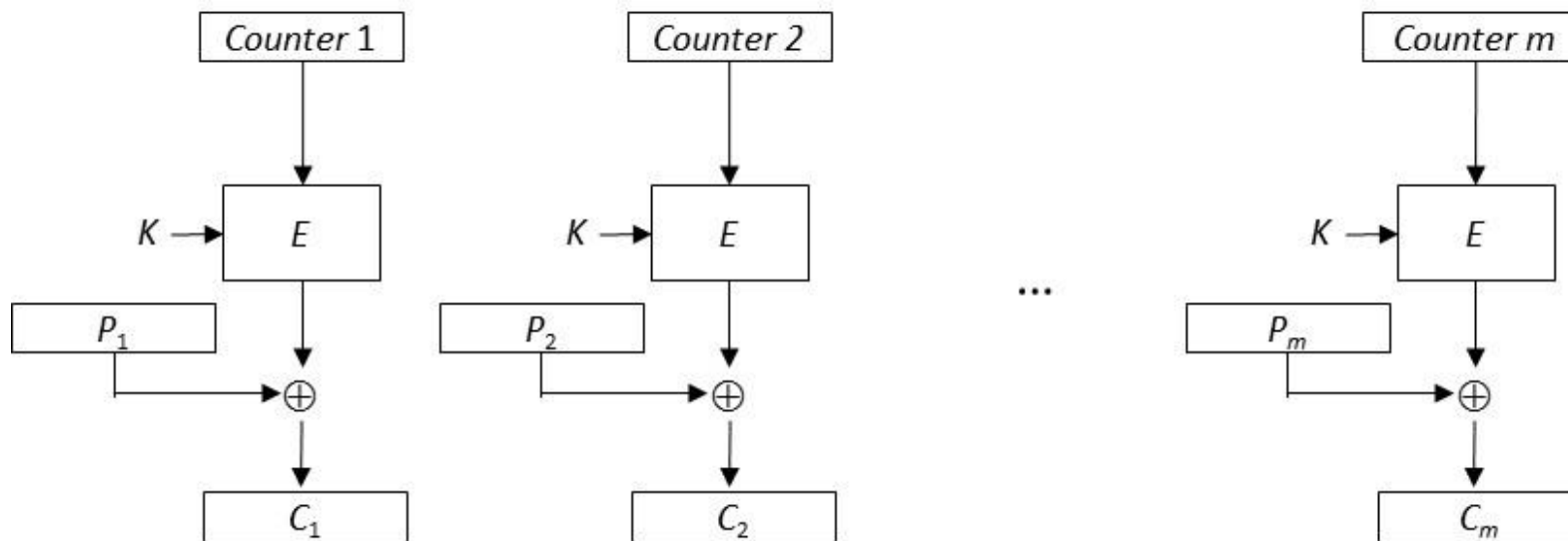
Kunci: 1011

Counter: 0000

Fungsi enkripsi  $E$  adalah sbb:

1. XOR-kan blok pesan  $T_i$  dengan  $K$
2. geser secara *wrapping* bit-bit dari hasil langkah 1 satu posisi ke kiri

$$E_K(T) = (T \oplus K) \ll 1$$



- Enkripsi:  $E_K(T) = (T \oplus K) \ll 1$

Counter:	0000	0001	0010	0011	0100	
	1011	1011	1011	1011	1011	⊕
Hasil XOR	1011	1010	1001	1000	1111	
Geser 1 bit ke kiri	0111	0101	0011	0001	1111	
XOR dengan P	1010	0010	0011	1010	1001	⊕
Cipherteks:	1101	0111	0000	1011	0111	
Cipherteks (Hex):	D	7	0	B	7	

Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of plaintext bits is encoded independently using the same key.	<ul style="list-style-type: none"> <li>Secure transmission of single values (e.g., an encryption key)</li> </ul>
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next block of plaintext and the preceding block of ciphertext.	<ul style="list-style-type: none"> <li>General-purpose block-oriented transmission</li> <li>Authentication</li> </ul>
Cipher Feedback (CFB)	Input is processed $s$ bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	<ul style="list-style-type: none"> <li>General-purpose stream-oriented transmission</li> <li>Authentication</li> </ul>
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding encryption output, and full blocks are used.	<ul style="list-style-type: none"> <li>Stream-oriented transmission over noisy channel (e.g., satellite communication)</li> </ul>
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	<ul style="list-style-type: none"> <li>General-purpose block-oriented transmission</li> <li>Useful for high-speed requirements</li> </ul>

- *Daftar block cipher:*

1. Lucifer
2. DES
3. GOST
4. 3DES (Triple-DES)
5. RC2
6. RC5
7. Blowfish
8. AES
9. IDEA
10. LOKI
11. FEAL
12. CAST-128
13. CRAB
14. SAFER
15. Twofish
16. Serpent
17. RC6
18. Camellia
19. 3-WAY
20. MMB
21. Skipjack
22. TEA
23. XTEA
24. SEED
25. Coconut
26. Cobra
27. MARS
28. BATON
29. CRYPTON
30. LEA, dll

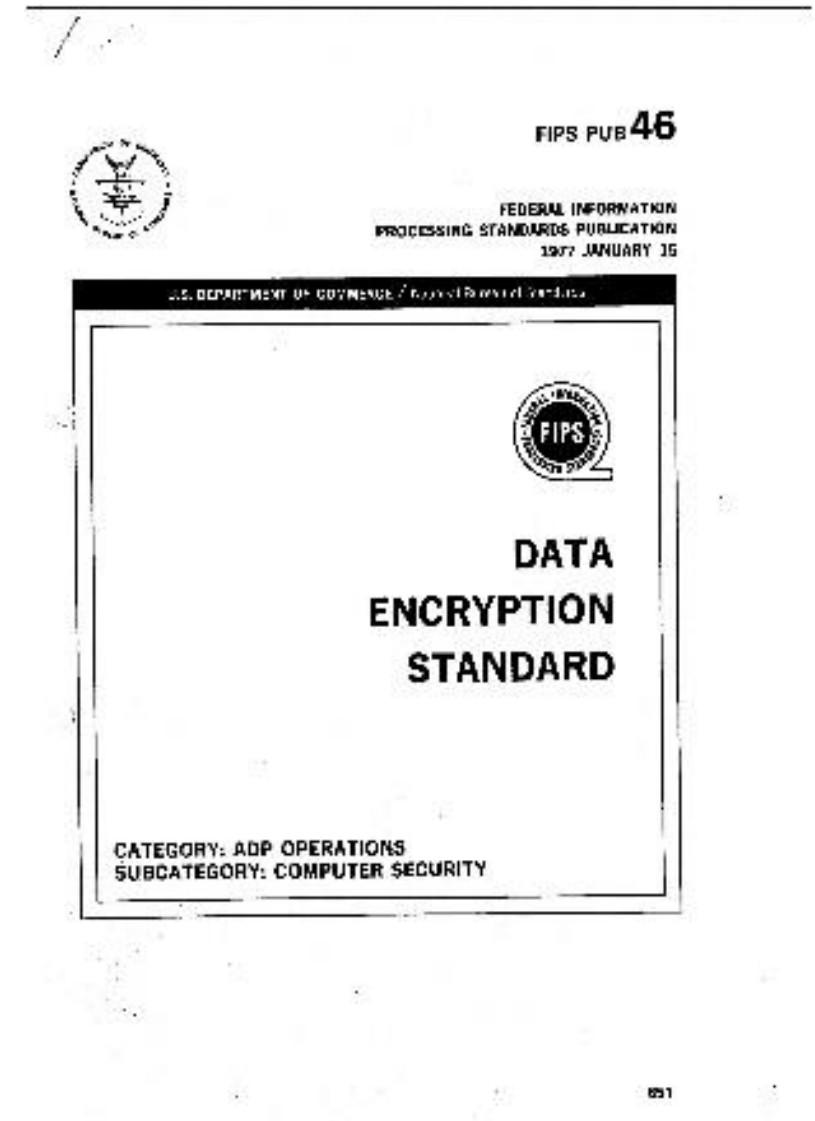
V · T · E	Block ciphers (security summary)
<b>Common algorithms</b>	AES · Blowfish · DES (internal mechanics, Triple DES) · Serpent · Twofish
<b>Less common algorithms</b>	ARIA · Camellia · CAST-128 · GOST · IDEA · LEA · RC2 · RC5 · RC6 · SEED · Skipjack · TEA · XTEA
<b>Other algorithms</b>	3-Way · Akelarre · Anubis · BaseKing · BassOmatic · BATON · BEAR and LION · CAST-256 · Chiasmus · CIKS-1 · CIPHERUNICORN-A · CIPHERUNICORN-E · CLEFIA · CMEA · Cobra · COCONUT98 · Crab · Cryptomeria/C2 · CRYPTON · CS-Cipher · DEAL · DES-X · DFC · E2 · FEAL · FEA-M · FROG · G-DES · Grand Cru · Hasty Pudding cipher · Hierocrypt · ICE · IDEA NXT · Intel Cascade Cipher · Iraqi · Kalyna · KASUMI · KeeLoq · KHAZAD · Khufu and Khafre · KN-Cipher · Kuznyechik · Ladder-DES · LOKI (97, 89/91) · Lucifer · M6 · M8 · MacGuffin · Madryga · MAGENTA · MARS · Mercy · MESH · MISTY1 · MMB · MULTI2 · MultiSwap · New Data Seal · NewDES · Nimbus · NOEKEON · NUSH · PRESENT · Prince · Q · REDOC · Red Pike · S-1 · SAFER · SAVILLE · SC2000 · SHACAL · SHARK · Simon · SM4 · Speck · Spectr-H64 · Square · SXAL/MBAL · Threefish · Treyfer · UES · xmx · XXTEA · Zodiac
<b>Design</b>	Feistel network · Key schedule · Lai–Massey scheme · Product cipher · S-box · P-box · SPN · Confusion and diffusion · Avalanche effect · Block size · Key size · Key whitening (Whitening transformation)
<b>Attack (cryptanalysis)</b>	Brute-force (EFF DES cracker) · MITM (Biclique attack · 3-subset MITM attack) · Linear (Piling-up lemma) · Differential (Impossible · Truncated · Higher-order) · Differential-linear · Distinguishing (Known-key) · Integral/Square · Boomerang · Mod <i>n</i> · Related-key · Slide · Rotational · Side-channel (Timing · Power-monitoring · Electromagnetic · Acoustic · Differential-fault) · XSL · Interpolation · Partitioning · Rubber-hose · Black-bag · Davies · Rebound · Weak key · Tau · Chi-square · Time/memory/data tradeoff
<b>Standardization</b>	AES process · CRYPTREC · NESSIE
<b>Utilization</b>	Initialization vector · Mode of operation · Padding
V · T · E	Cryptography <span style="float: right;">[show]</span>

# Data Encryption Standard (DES)

# Sejarah DES

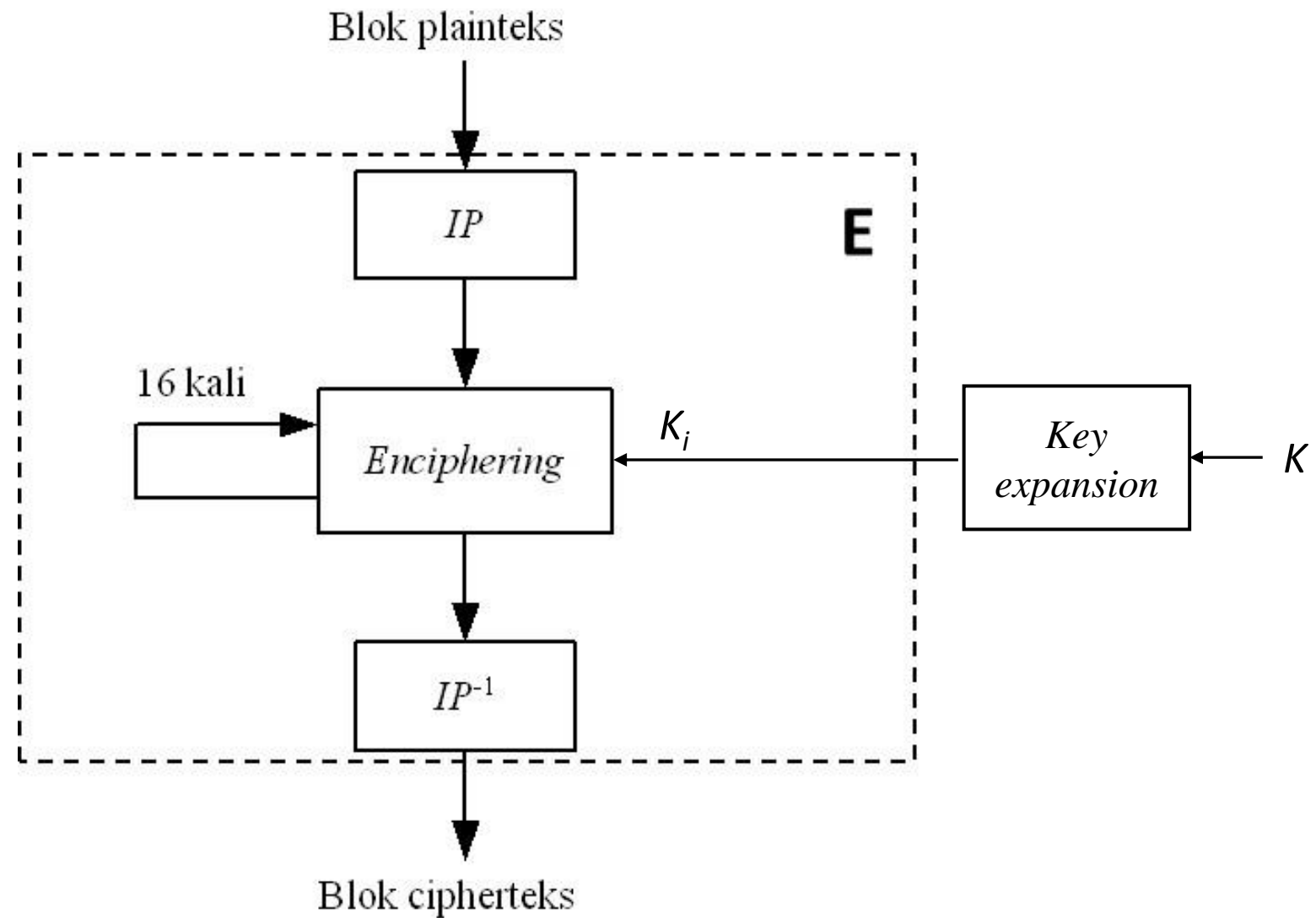
- Pada tahun 1972 NIST meminta standard enkripsi *cipher* blok.
- Pada tahun 1974 Horst Feistel di IBM mendesain cipher blok Lucifer (panjang kunci 128bit, Panjang blok 128 bit)
- Lucifer kemudian berkembang menjadi DES dengan beberapa masukan dari NSA (*National Security Agency*):
  - panjang kunci 56 bit, ukuran blok 64 bit
  - 16 putaran.
- Tahun 1976 DES disetujui oleh *National Bureau of Standard (NBS)* setelah penilaian kekuatannya oleh NSA.
- Sejak itu DES diimplementasikan secara luas, baik *software* maupun *hardware*.
- Tahun 1997-1998 DES berhasil dipecahkan dengan *brute force attack*
- Tahun 2001 DES diganti oleh AES

- Catat bahwa DES adalah sebuah standard, sedangkan algoritmanya sendiri bernama DEA (*Data Encryption Algorithm*). Kedua nama ini sering dikacaukan.
- DES termasuk ke dalam algoritma kriptografi kunci-simetri dan tergolong ke dalam *cipher* blok.
- DES beroperasi pada ukuran blok plainteks 64 bit.
- Panjang kunci eksternal = 64 bit (sesuai ukuran blok), tetapi hanya 56 bit yang dipakai (8 bit *parity* tidak digunakan). Bit parity = bit ke-8, 16, 24, dst.

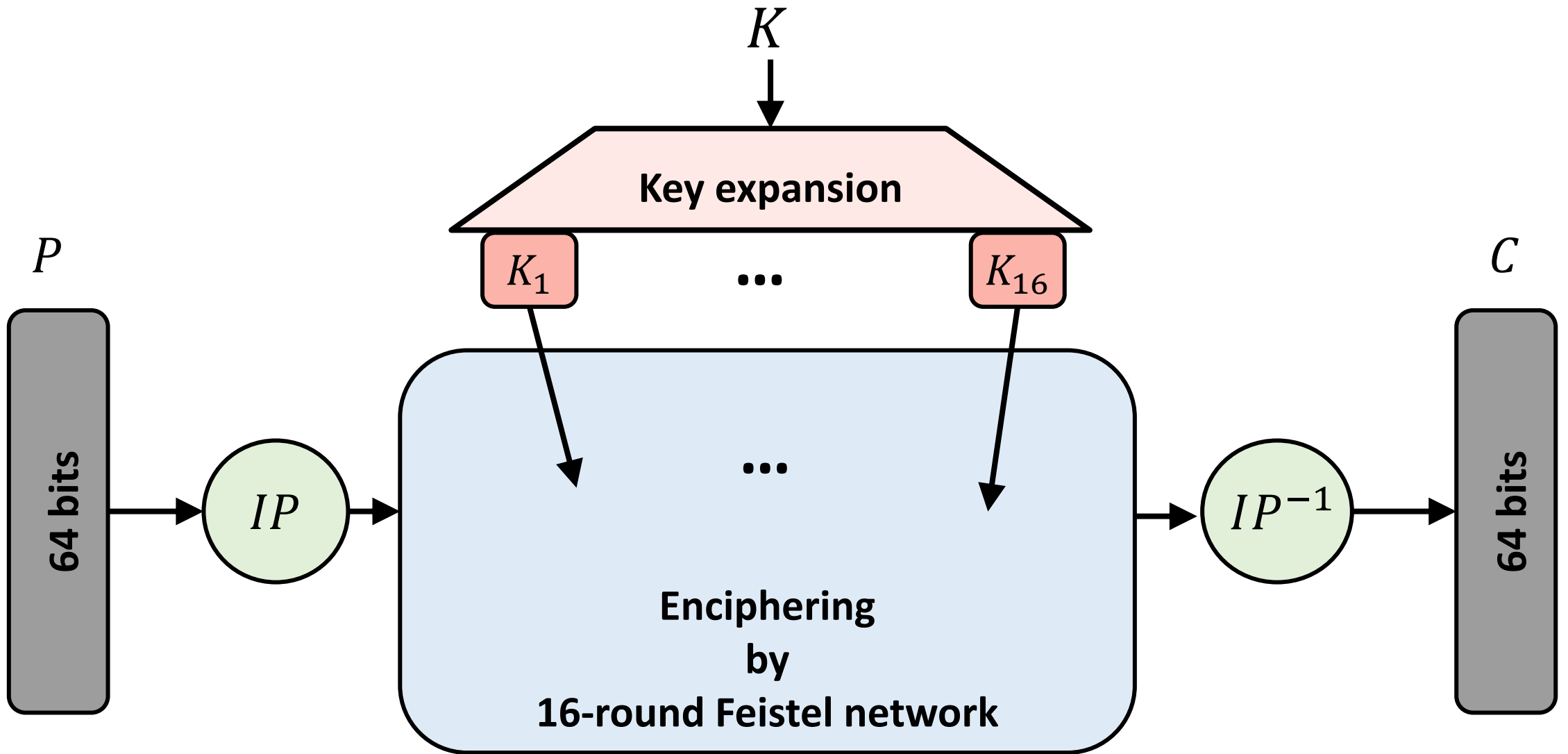


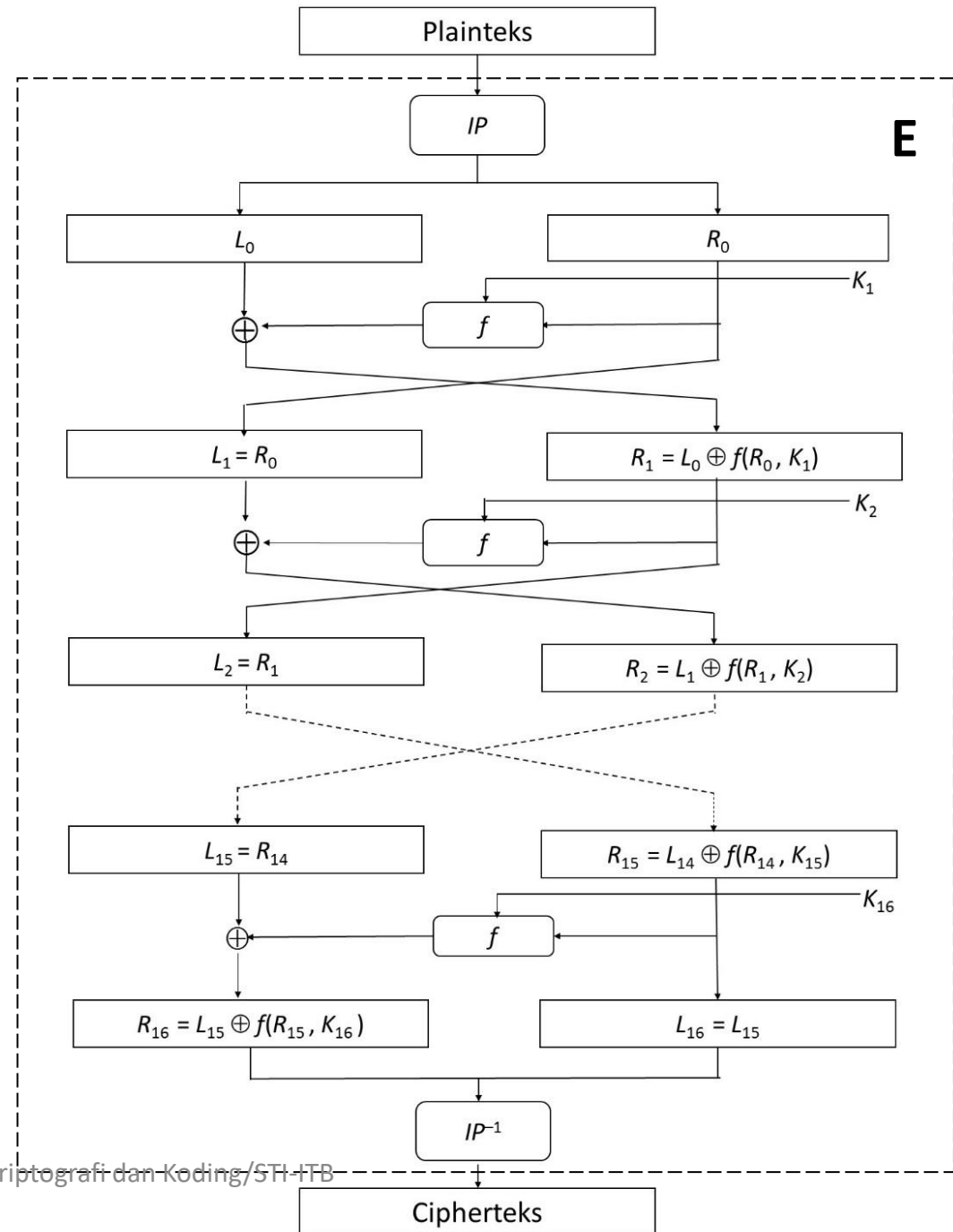
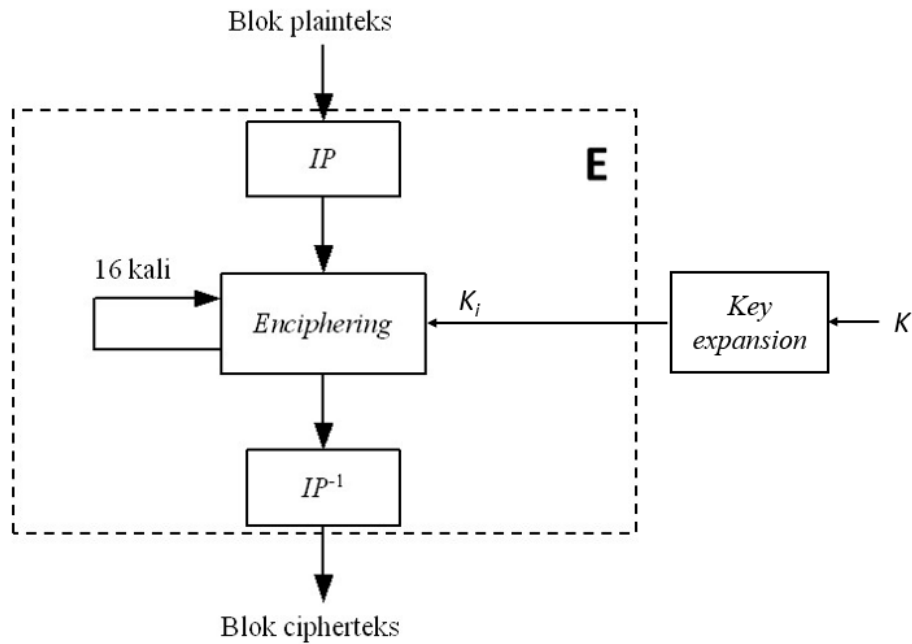


- Setiap blok plainteks dienkripsi dalam 16 putaran *enciphering*.
- Setiap putaran menggunakan kunci internal (kunci putaran) berbeda.
- Kunci internal sepanjang 48-bit dibangkitkan dari kunci eksternal
- Setiap blok plainteks mengalami permutasi awal ( $IP$ ), 16 putaran *enciphering*, dan inversi permutasi awal ( $IP^{-1}$ ). (lihat Gambar 1)

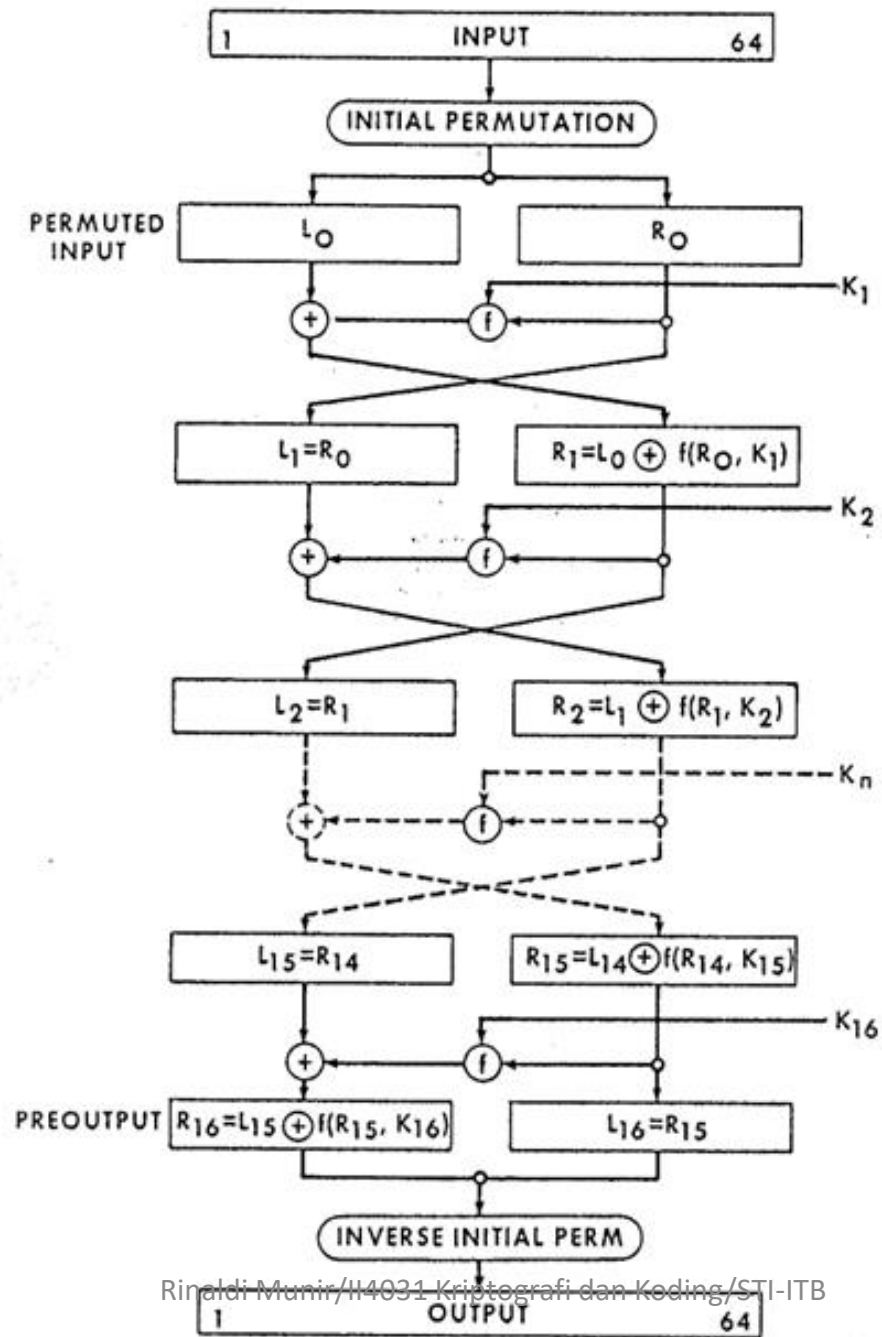


**Gambar 1** Skema global algoritma DES



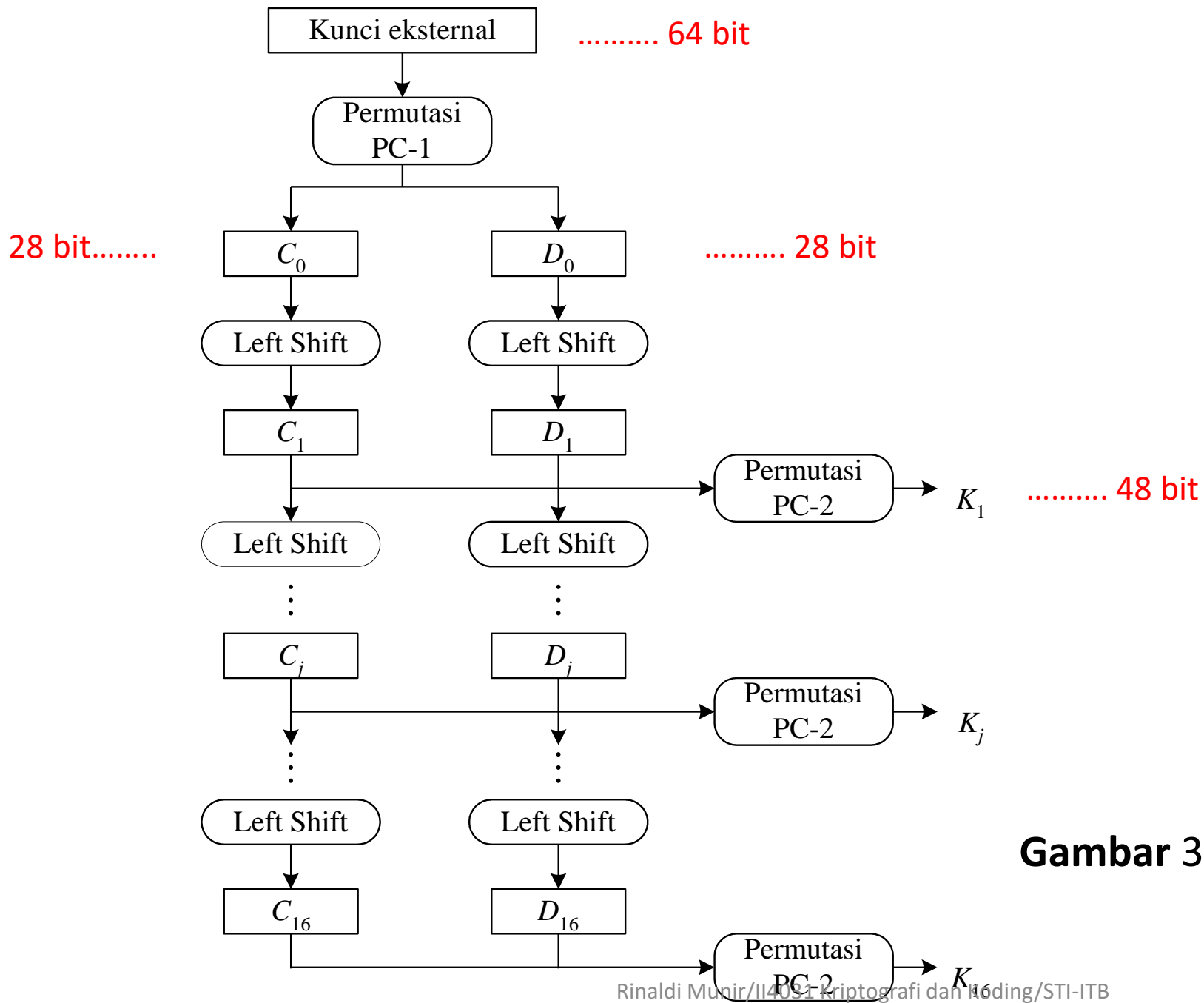


**Gambar 2. Algoritma Enkripsi dengan DES**

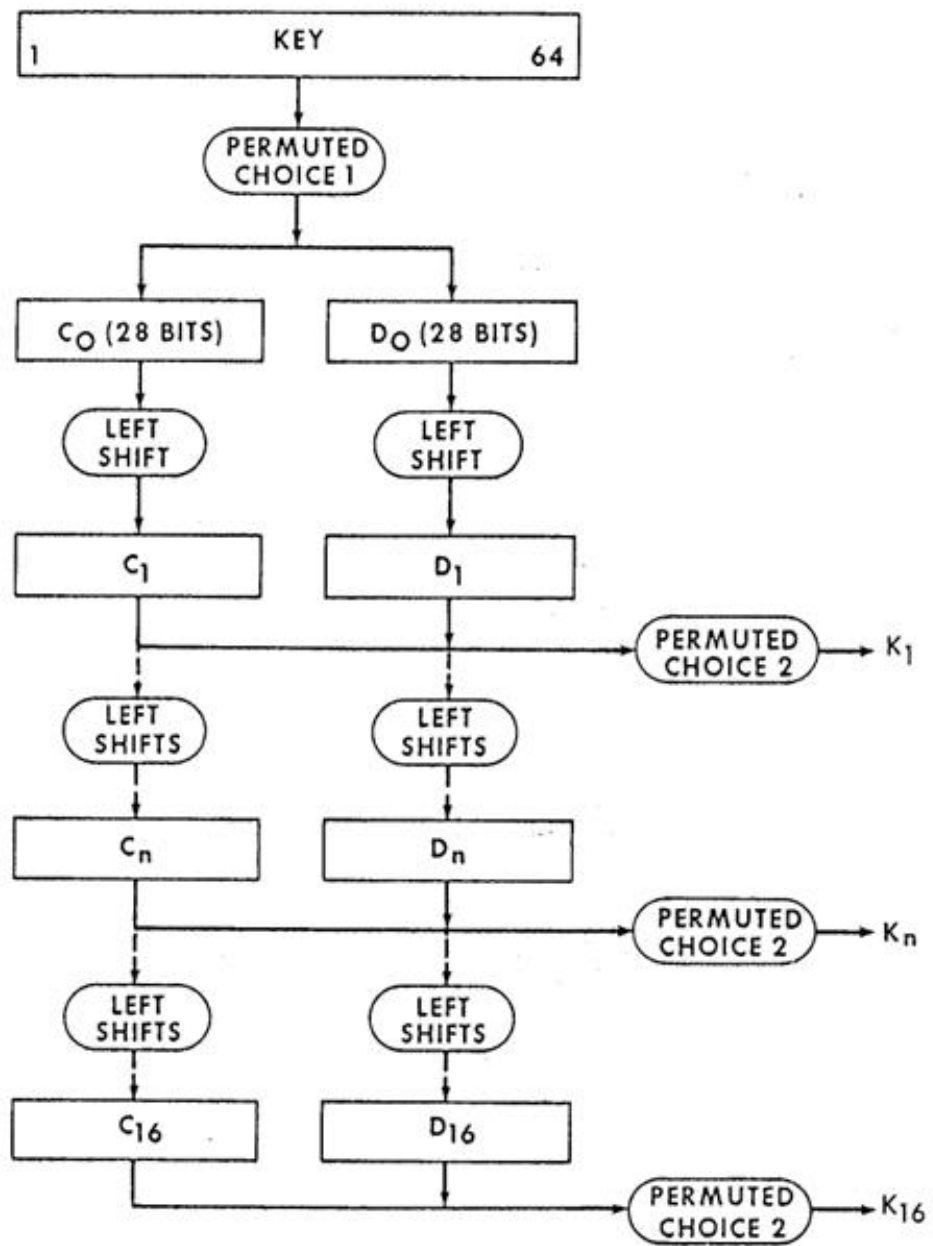


# Pembangkitan Kunci Internal (*key expansion*)

- Kunci internal = kunci setiap putaran = *subkey (round key)*
- Ada 16 putaran, jadi ada 16 kunci internal:  $K_1, K_2, \dots, K_{16}$
- Kunci internal dibangkitkan dari kunci eksternal (64 bit) yang diberikan oleh pengguna. Di dalam dokumen resmi DES pembangkitan kunci internal disebut *key schedule*.
- Panjang kunci internal adalah 48 bit
- Gambar 3 memperlihatkan proses pembangkitan kunci internal.



**Gambar 3.** Pembangkitan kunci internal





Matriks permutasi kompresi PC-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

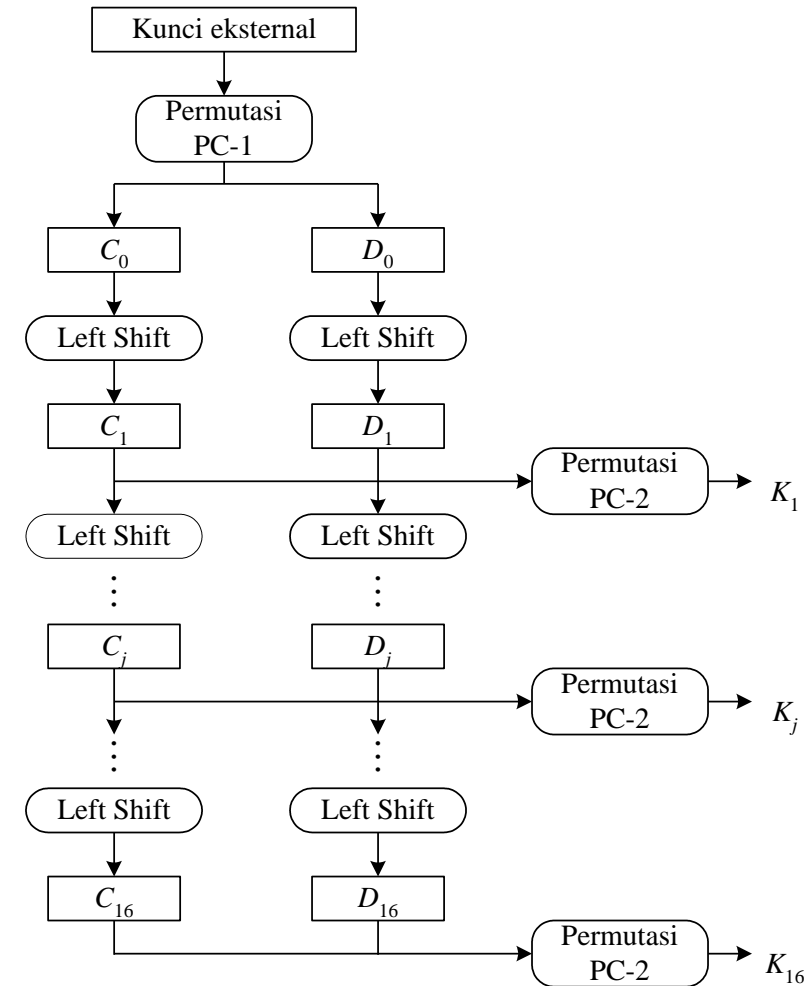
$C_0$ : berisi bit-bit dari  $K$  pada posisi

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

$D_0$ : berisi bit-bit dari  $K$  pada posisi

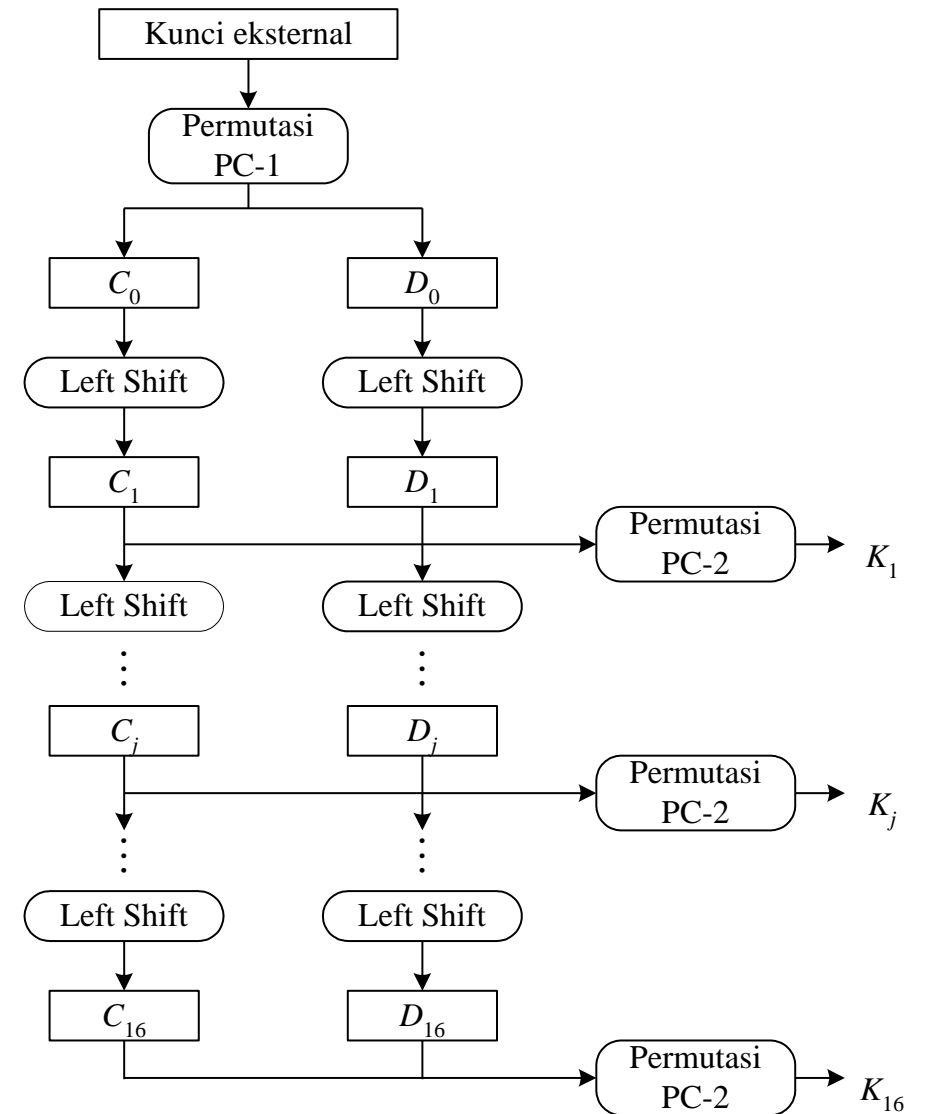
63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22  
 14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4

Perhatikan, bit-bit *parity*, yaitu bit ke-8, 16, 28, ... dibuang, sehingga menjadi 56 bit



**Tabel 1.** Jumlah pergeseran bit pada setiap putaran

Putaran, $i$	Jumlah pergeseran bit
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1



Matriks PC-2 berikut:

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

Jadi,  $K_i$  merupakan penggabungan bit-bit  $C_i$  pada posisi:

14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10

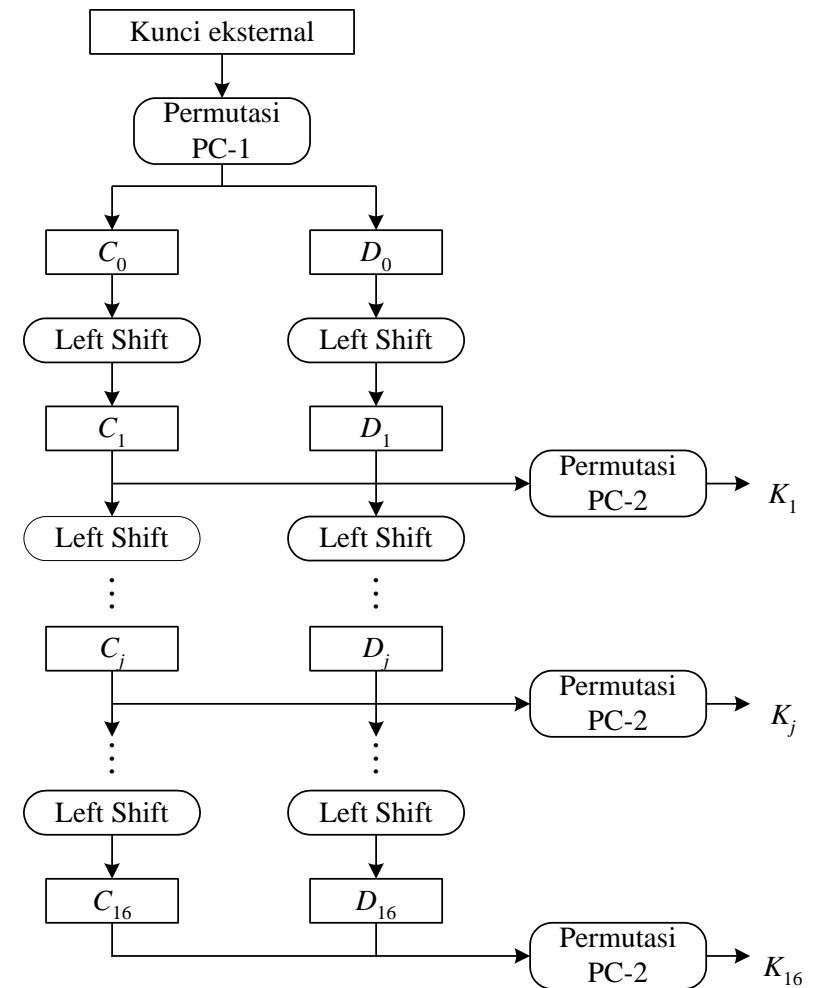
23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2

dengan bit-bit  $D_i$  pada posisi:

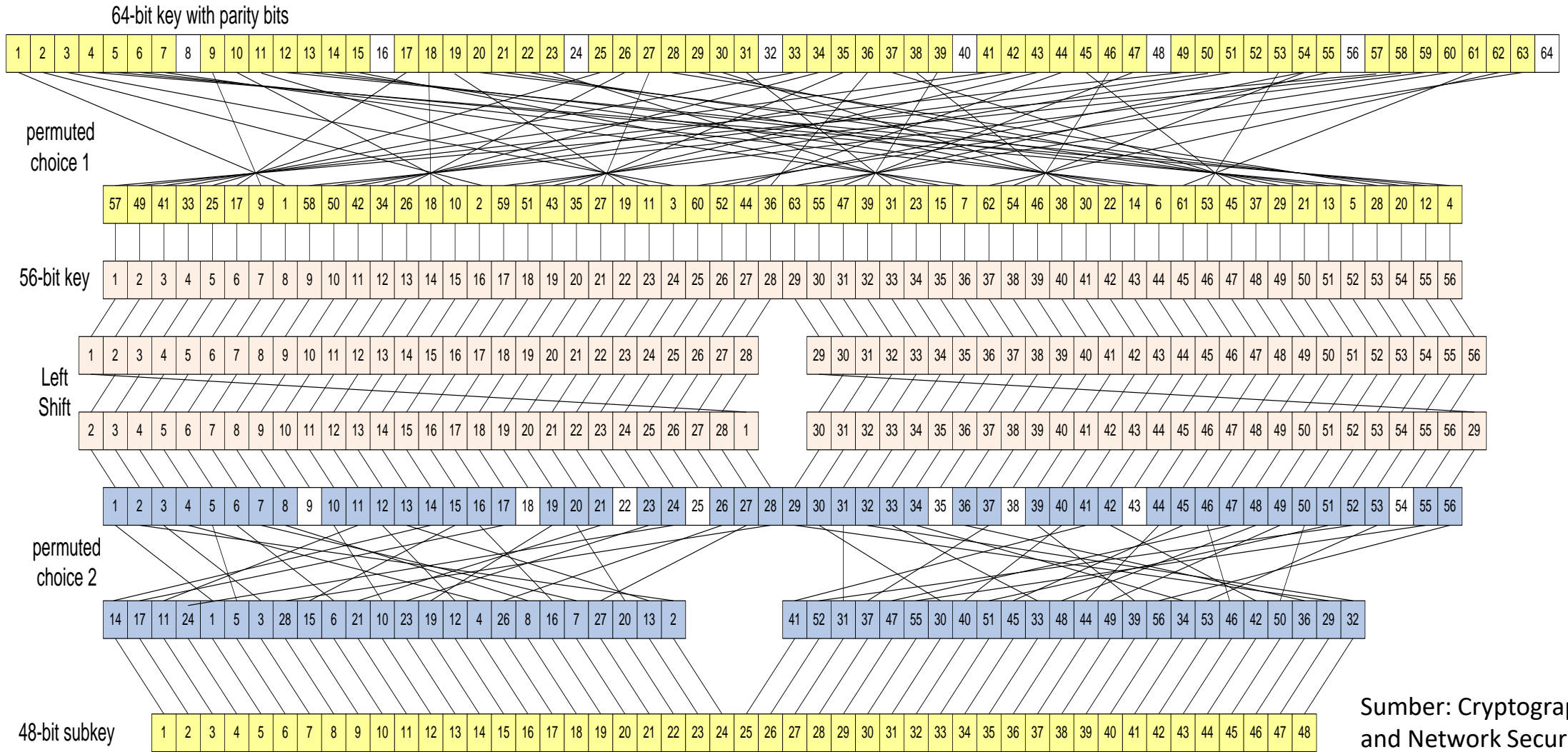
41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48

44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32

Setiap kunci internal  $K_i$  mempunyai panjang 48 bit.



# Pembangkitan kunci internal secara lengkap:

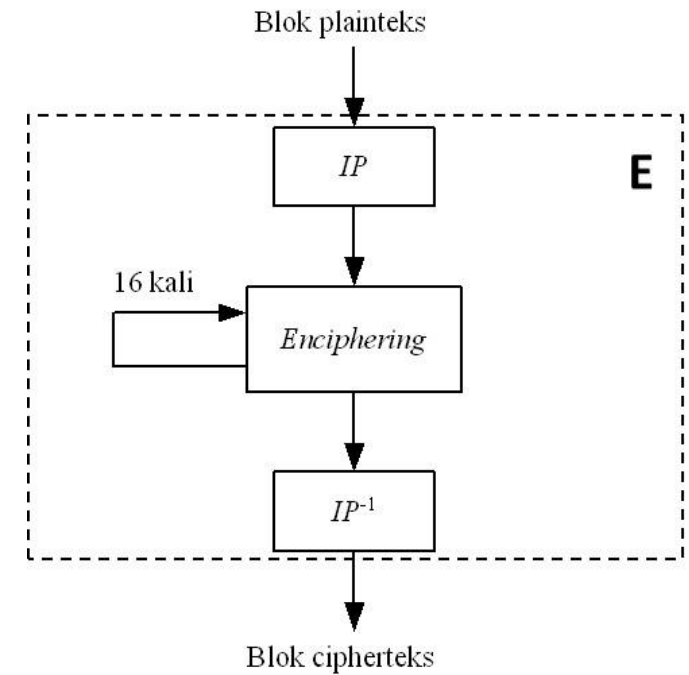


Sumber: Cryptography and Network Security Chapter 3, Lecture slides by Lawrie Brown Modified by Richard Newman

# Permutasi Awal

- Tujuan: mengacak plainteks sehingga urutan bit-bit di dalamnya berubah.
- Matriks permutasi awal ( $IP$ ):

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

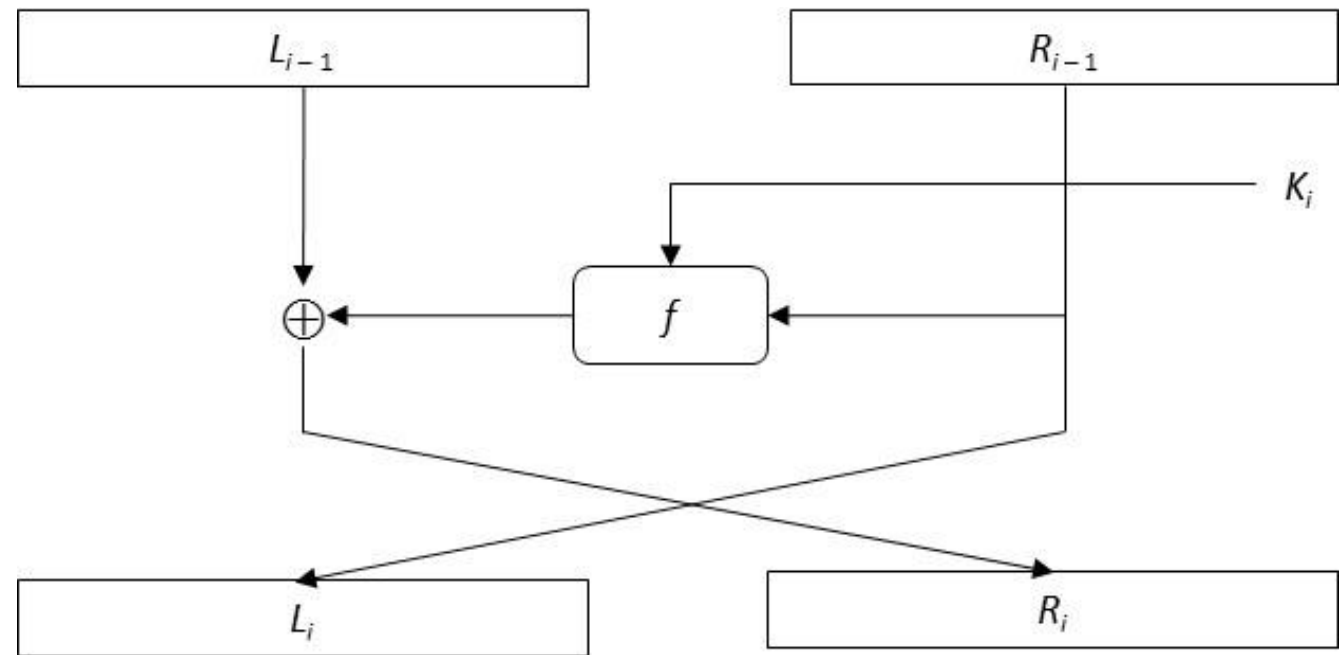
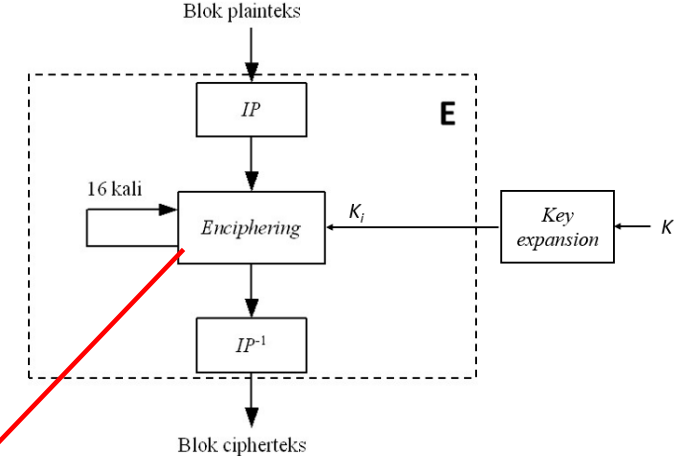


# Enciphering

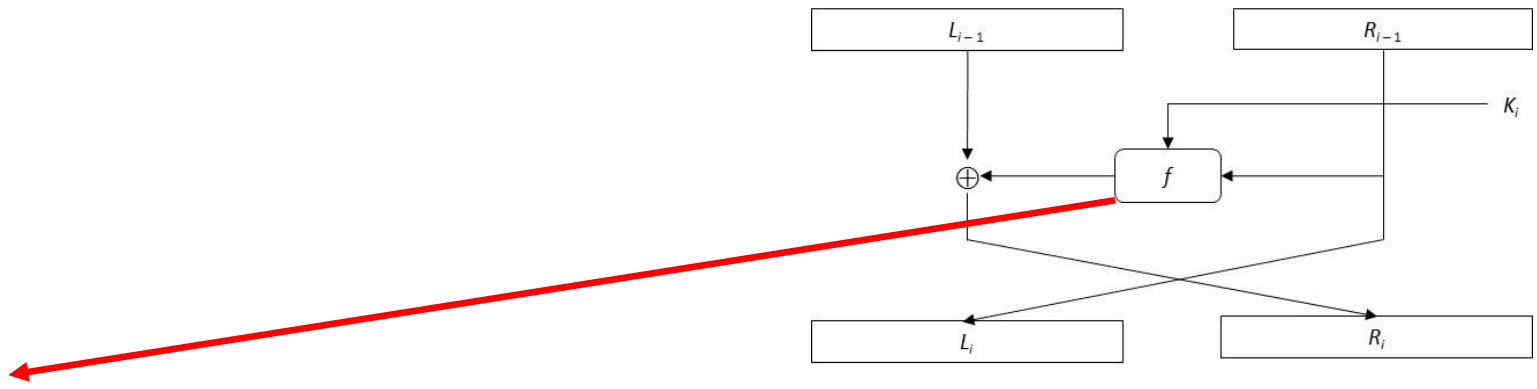
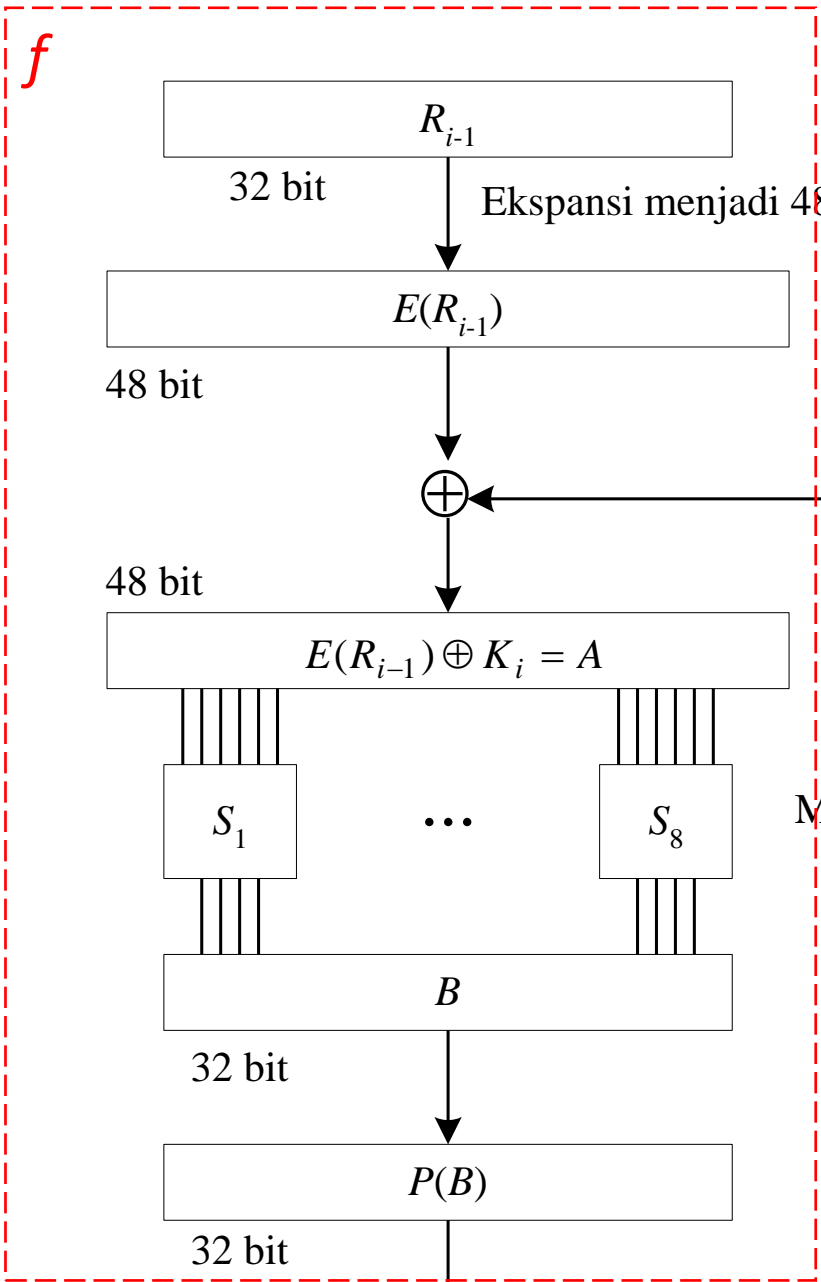
- Setiap blok plainteks mengalami 16 kali putaran *enciphering*
- Setiap putaran *enciphering* merupakan jaringan Feistel:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$



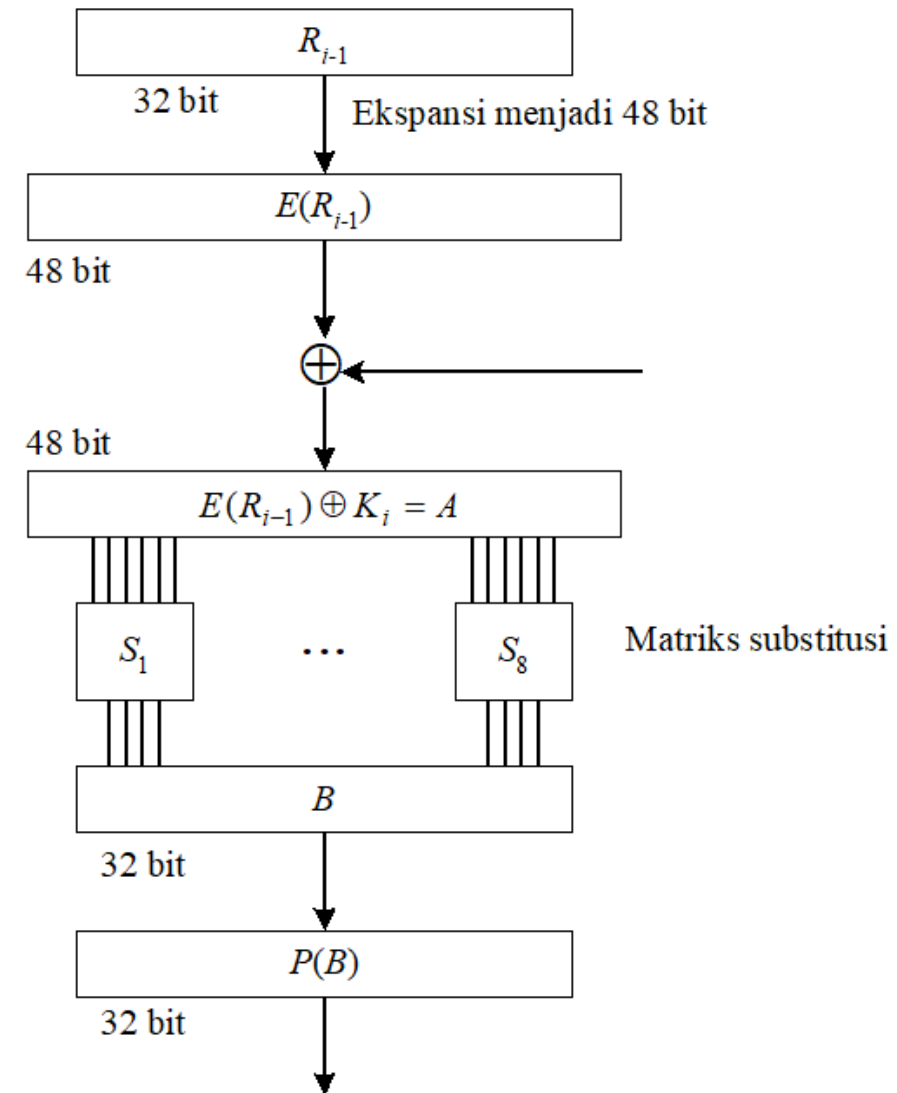
Gambar 4. Satu putaran *enciphering*



**Gambar 5.** Diagram komputasi fungsi *f*:

- $E$  adalah fungsi ekspansi yang memperluas blok  $R_{i-1}$  32-bit menjadi blok 48 bit.
- Fungsi ekspansi direalisasikan dengan matriks permutasi ekspansi:

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

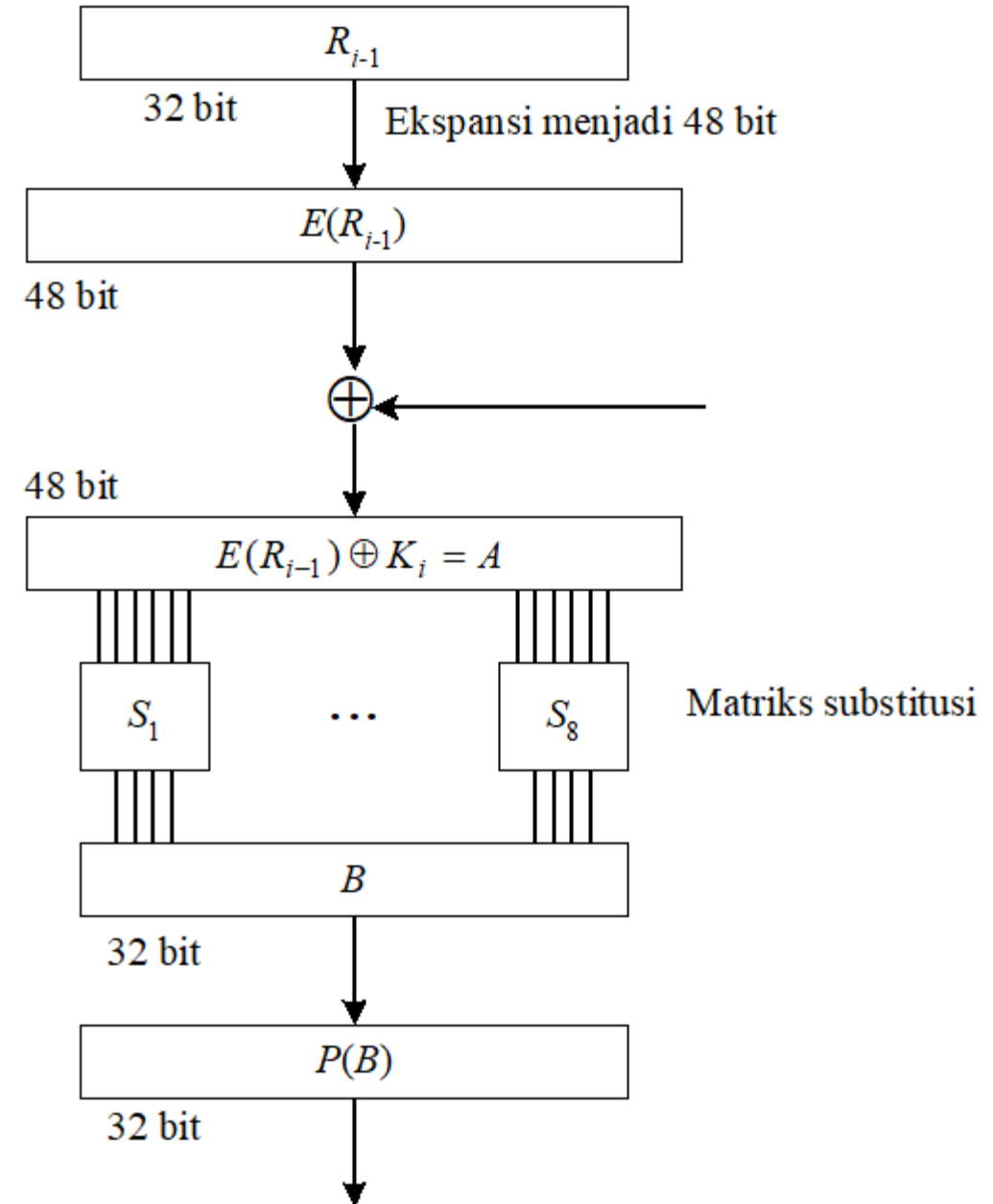




- Hasil ekspansi, yaitu  $E(R_{i-1})$  di-XOR-kan dengan  $K_i$  menghasilkan blok  $A$  berukuran 48-bit:

$$E(R_{i-1}) \oplus K_i = A$$

- Blok  $A$  dikelompokkan menjadi 8 kelompok, masing-masing 6 bit, dan menjadi masukan bagi proses substitusi.
- Ada 8 matriks substitusi, masing-masing dinyatakan dengan kotak-S.
- Kotak  $S$  menerima masukan 6 bit dan memberikan luaran 4 bit.



$S_1$ :

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

$S_2$ :

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

$S_3$ :

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

$S_4$ :

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Input: 101010

Baris = 10 = 2

Kolom = 0101 = 5

Luaran = 13 = 1101

$S_5$ :

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	16
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

$S_6$ :

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

- Setiap baris di dalam S-box adalah permutasi angka-angka 0, 1, 2, ..., 15.

$S_7$ :

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

- Mengubah satu bit di dalam input S-box mengakibatkan perubahan paling sedikit 2 bit pada output S-box

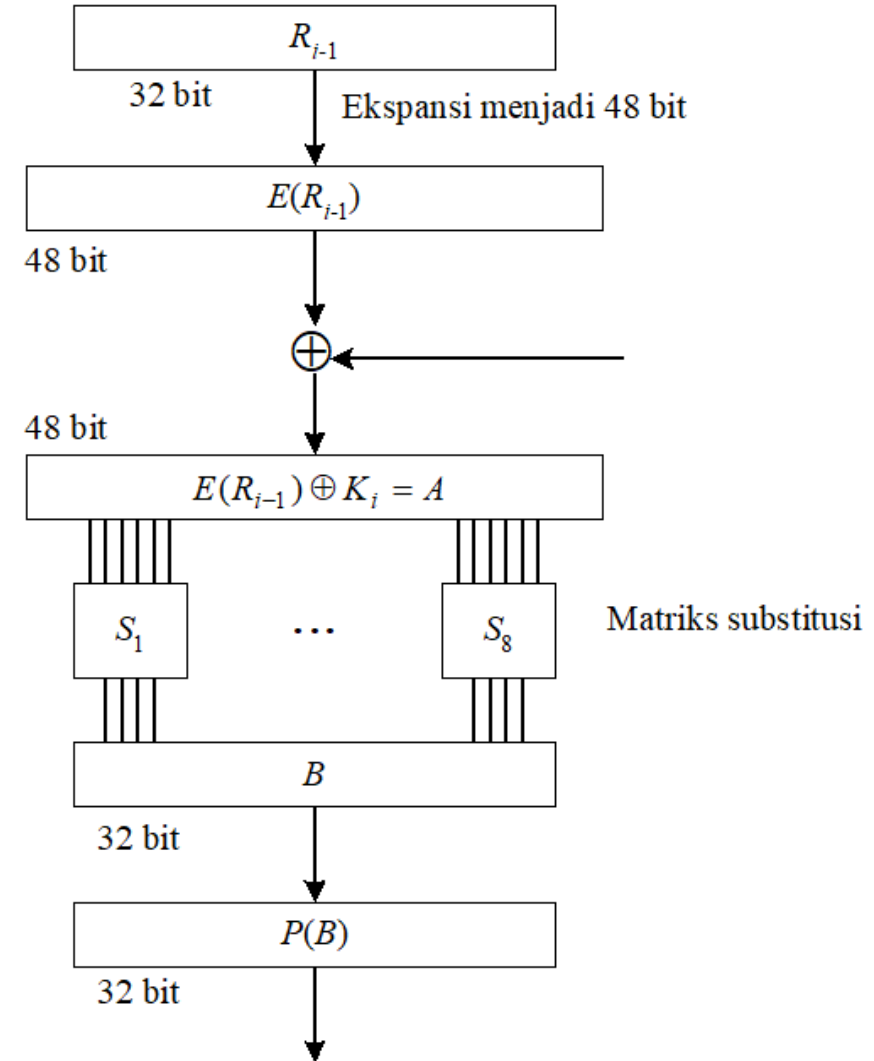
$S_8$ :

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

→ prinsip *confussion*

- Luaran proses substitusi adalah blok  $B$  yang panjangnya 32 bit.
- Blok  $B$  menjadi masukan untuk proses permutasi.
- Tujuan permutasi adalah untuk mengacak hasil proses substitusi kotak-S.
- Permutasi dilakukan dengan menggunakan matriks permutasi  $P$  ( $P$ -box) sbb:

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

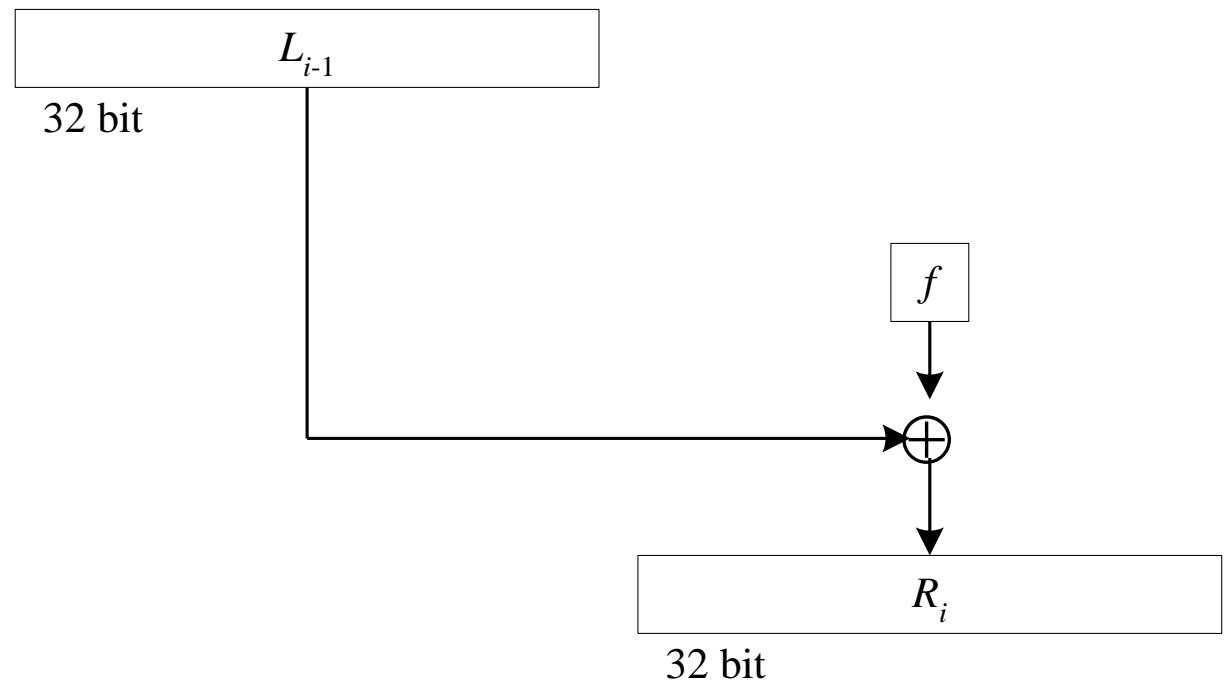


- $P(B)$  merupakan luaran dari fungsi  $f$ .
- Bit-bit  $P(B)$  di- $XOR$ -kan dengan  $L_{i-1}$  menghasilkan  $R_i$ :

$$R_i = L_{i-1} \oplus P(B)$$

- Jadi, luaran dari putaran ke- $i$  adalah

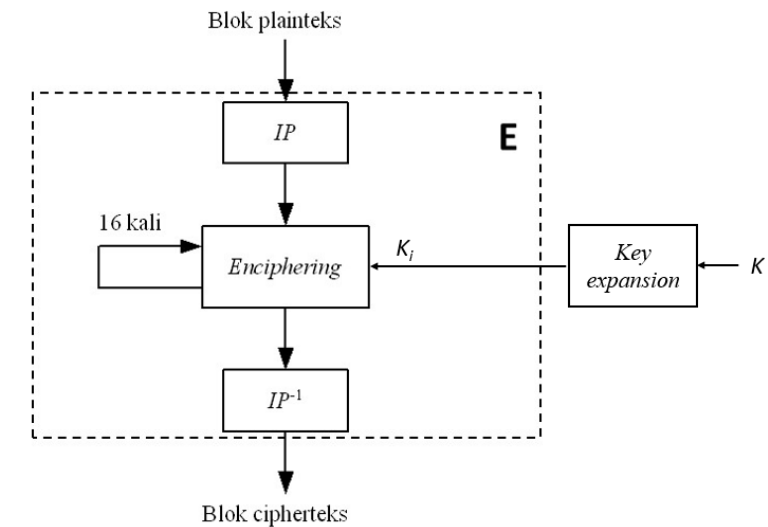
$$(L_i, R_i) = (R_{i-1}, L_{i-1} \oplus P(B))$$



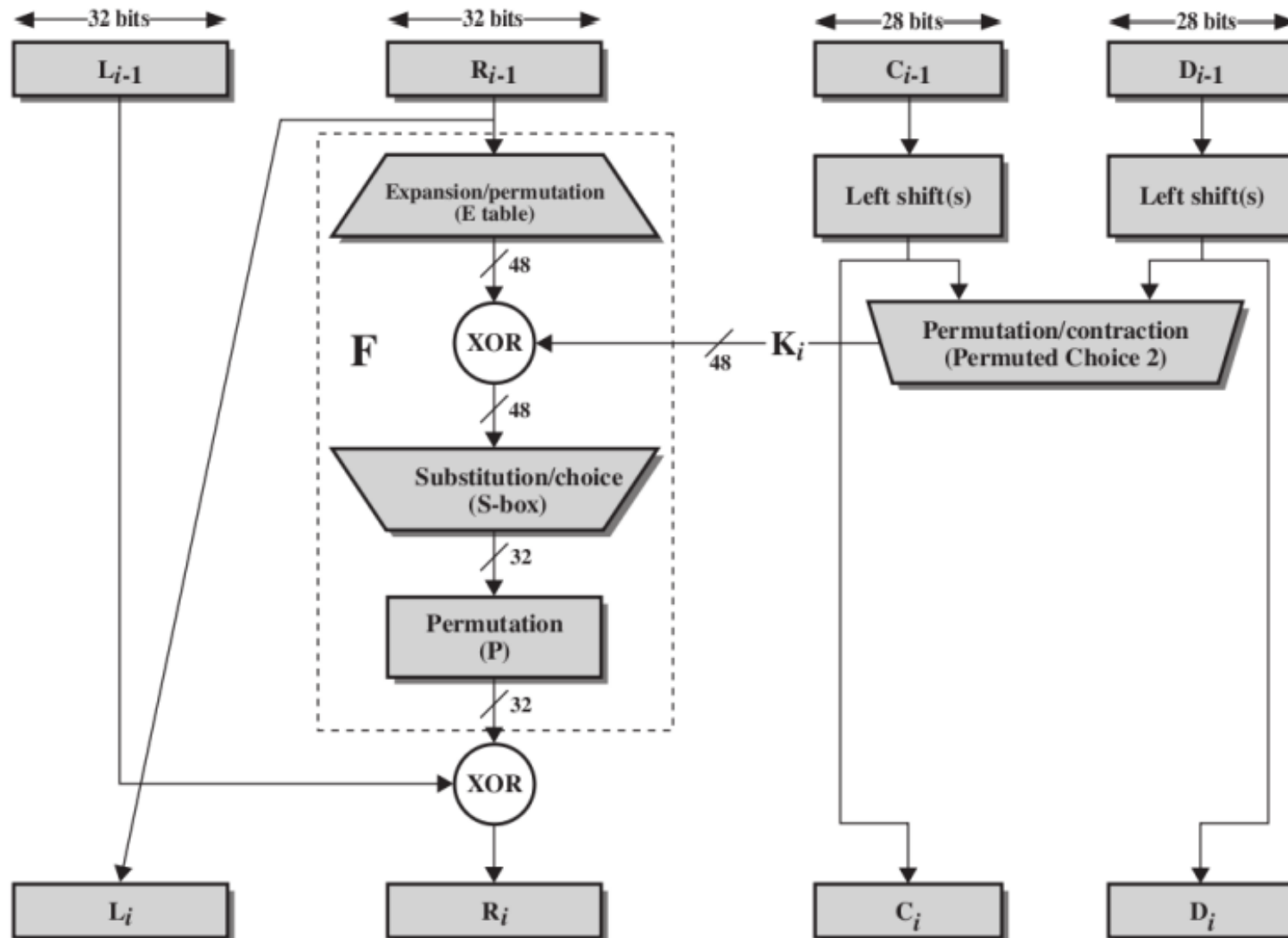
# Inversi Permutasi ( $IP^{-1}$ )

- Permutasi terakhir dilakukan setelah 16 kali putaran terhadap gabungan blok kiri dan blok kanan.
- Permutasi menggunakan matriks permutasi awal balikan ( $IP^{-1}$ ) sbb:

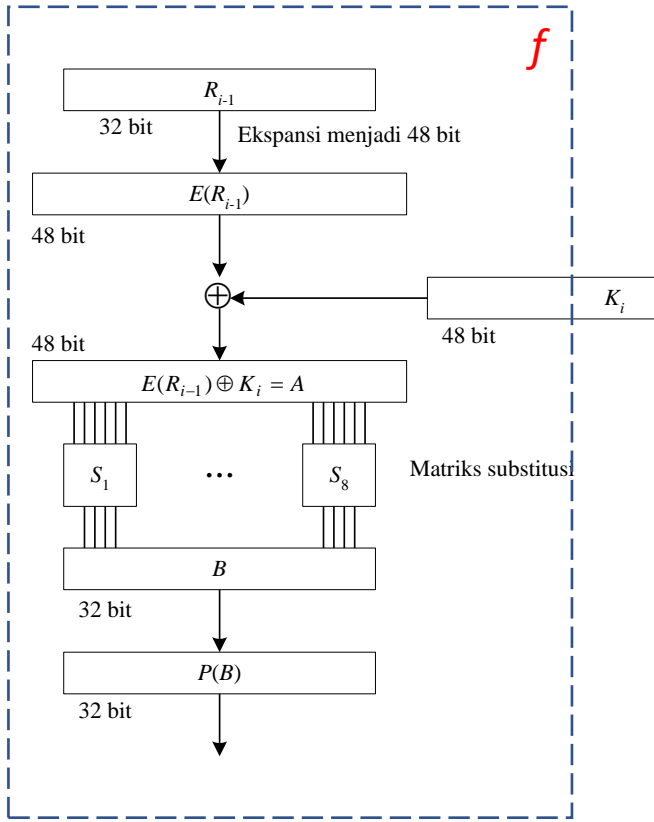
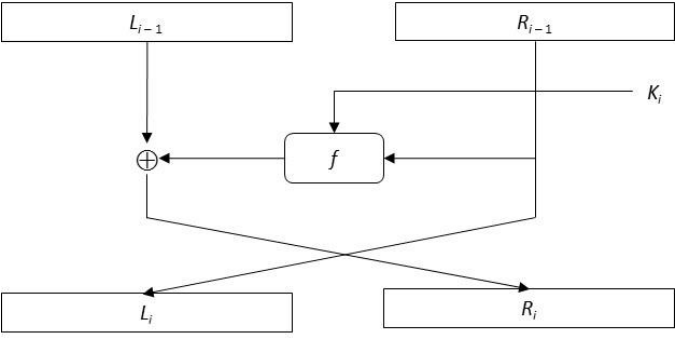
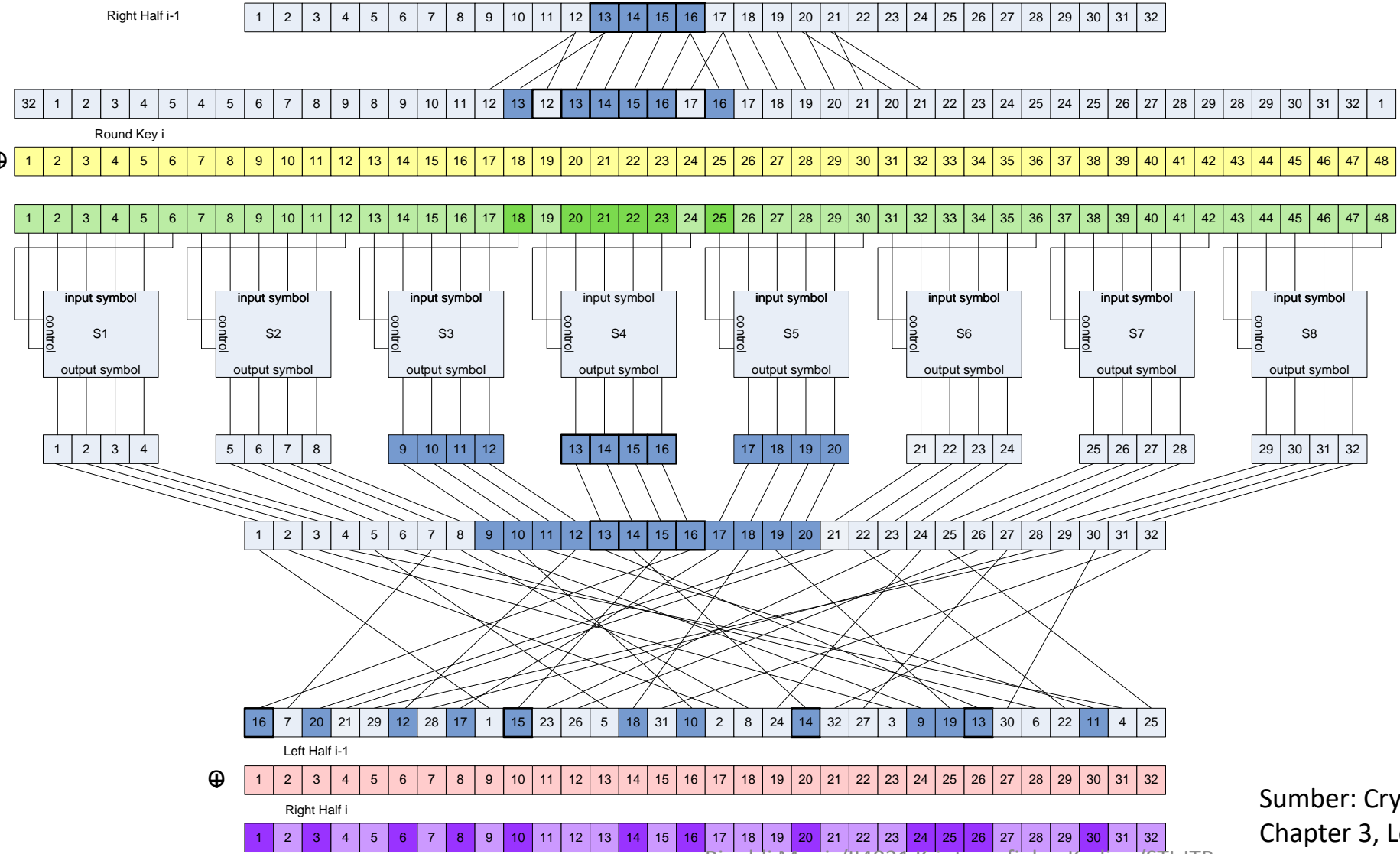
40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25



# Satu putaran di dalam DES:



# Satu putaran di dalam jaringan Feistel secara lengkap:



Sumber: Cryptography and Network Security  
 Chapter 3, Lecture slides by Lawrie Brown  
 Modified by Richard Newman



## Contoh hasil enkripsi DES pada setiap putaran:

Round	$K_i$	$L_i$	$R_i$
<b>IP</b>		5a005a00	3cf03c0f
<b>1</b>	1e030f03080d2930	3cf03c0f	bad22845
<b>2</b>	0a31293432242318	bad22845	99e9b723
<b>3</b>	23072318201d0c1d	99e9b723	0bae3b9e
<b>4</b>	05261d3824311a20	0bae3b9e	42415649
<b>5</b>	3325340136002c25	42415649	18b3fa41
<b>6</b>	123a2d0d04262a1c	18b3fa41	9616fe23
<b>7</b>	021f120b1c130611	9616fe23	67117cf2
<b>8</b>	1c10372a2832002b	67117cf2	c11bfc09
<b>9</b>	04292a380c341f03	c11bfc09	887fbc6c
<b>10</b>	2703212607280403	887fbc6c	600f7e8b
<b>11</b>	2826390c31261504	600f7e8b	f596506e
<b>12</b>	12071c241a0a0f08	f596506e	738538b8
<b>13</b>	300935393c0d100b	738538b8	c6a62c4e
<b>14</b>	311e09231321182a	c6a62c4e	56b0bd75
<b>15</b>	283d3e0227072528	56b0bd75	75e8fd8f
<b>16</b>	2921080b13143025	75e8fd8f	25896490
<b>IP<sup>-1</sup></b>		da02ce3a	89ecac3b

Sumber: Cryptography  
and Network Security  
Chapter 3, Lecture slides  
by Lawrie Brown  
Modified by Richard  
Newman

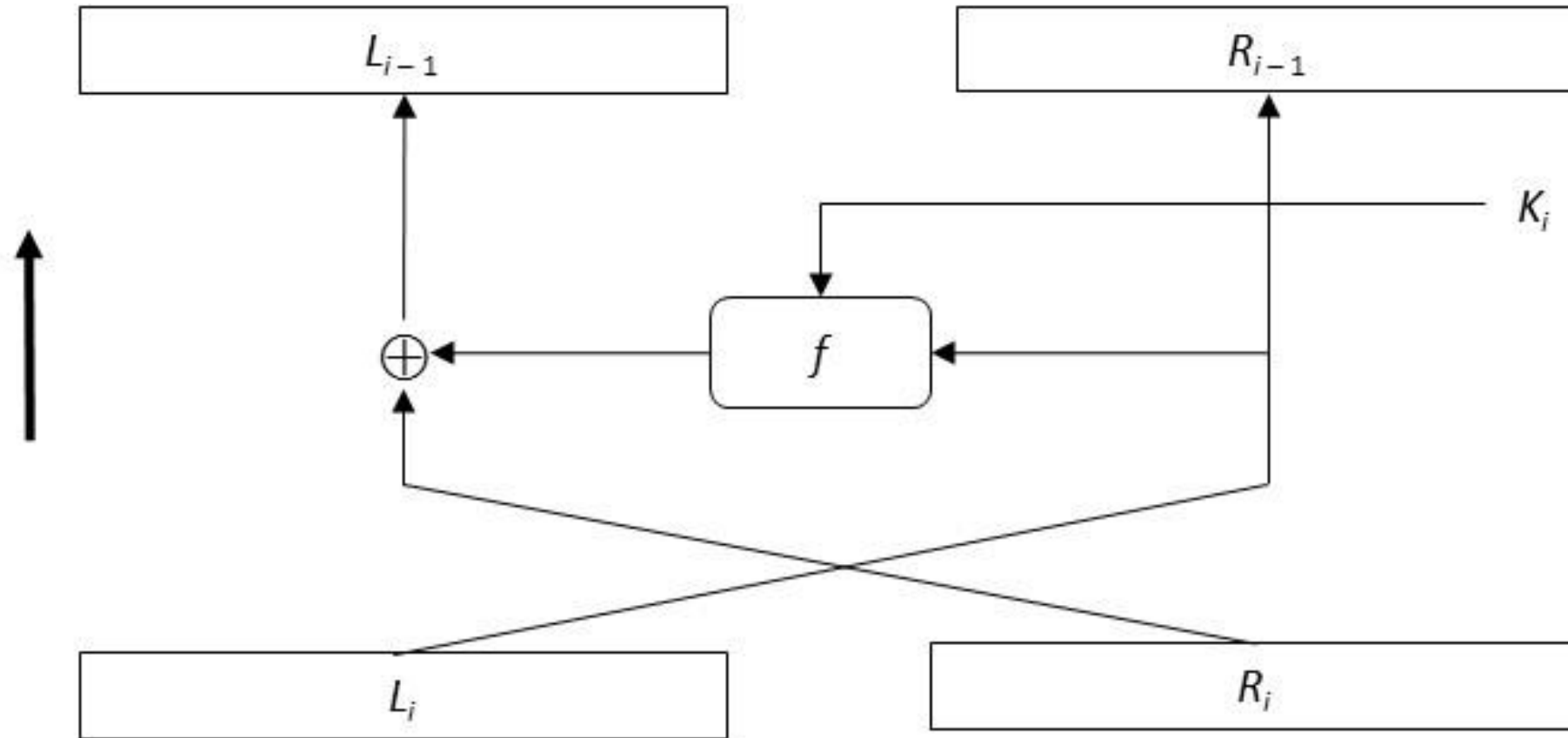
# Dekripsi

- Dekripsi terhadap cipherteks merupakan kebalikan dari proses enkripsi.
- DES menggunakan algoritma yang sama untuk proses enkripsi dan dekripsi.
- Pada proses dekripsi urutan kunci yang digunakan adalah  $K_{16}, K_{15}, \dots, K_1$ .
- Untuk tiap putaran 16, 15, ..., 1, luaran pada setiap putaran *deciphering* adalah

$$R_{i-1} = L_i$$

$$L_{i-1} = R_i \oplus f(R_{i-1}, K_i) = R_i \oplus f(L_i, K_i)$$

# Dekripsi:



$$R_{i-1} = L_i$$

$$L_{i-1} = R_i \oplus f(R_{i-1}, K_i) = R_i \oplus f(L_i, K_i)$$

# Efek longsor (*avalanche effect*)

- Efek longsor adalah properti kunci yang diinginkan dari algoritma enkripsi,
- Tujuan: menunjukkan perubahan satu bit pada plainteks atau kunci menghasilkan perubahan besar pada cipherteks
- Efek longsor membuat upaya untuk mencoba menebak kunci menjadi tidak mungkin
- DES menunjukkan efek longsor yang kuat

# Efek longSORan di dalam DES

(mengubah satu bit plainteks)

original →

modified →

Round		$\delta$	Round		$\delta$
	02468aceeca86420	1	9	c11bfc09887fbc6c	32
	12468aceeca86420			99f911532eed7d94	
1	3cf03c0fbad22845	1	10	887fbc6c600f7e8b	34
	3cf03c0fbad32845			2eed7d94d0f23094	
2	bad2284599e9b723	5	11	600f7e8bf596506e	37
	bad3284539a9b7a3			d0f23094455da9c4	
3	99e9b7230bae3b9e	18	12	f596506e738538b8	31
	39a9b7a3171cb8b3			455da9c47f6e3cf3	
4	0bae3b9e42415649	34	13	738538b8c6a62c4e	29
	171cb8b3ccaca55e			7f6e3cf34bc1a8d9	
5	4241564918b3fa41	37	14	c6a62c4e56b0bd75	33
	ccaca55ed16c3653			4bc1a8d91e07d409	
6	18b3fa419616fe23	33	15	56b0bd7575e8fd8f	31
	d16c3653cf402c68			1e07d4091ce2e6dc	
7	9616fe2367117cf2	32	16	75e8fd8f25896490	32
	cf402c682b2cefbc			1ce2e6dc365e5f59	
8	67117cf2c11bfc09	33	IP <sup>-1</sup>	da02ce3a89ecac3b	32
	2b2cefbc99f91153			057cde97d7683f2a	

Sumber: Cryptography and Network Security Chapter 3, Lecture slides by Lawrie Brown Modified by Richard Newman

# Efek longsor di dalam DES

(mengubah satu bit kunci)

Round		$\delta$	Round		$\delta$
	02468aceeca86420 02468aceeca86420	0	9	c11bfc09887fbc6c 548f1de471f64dfd	34
1	3cf03c0fbad22845 3cf03c0f9ad628c5	3	10	887fbc6c600f7e8b 71f64dfd4279876c	36
2	bad2284599e9b723 9ad628c59939136b	11	11	600f7e8bf596506e 4279876c399fdc0d	32
3	99e9b7230bae3b9e 9939136b768067b7	25	12	f596506e738538b8 399fdc0d6d208dbb	28
4	0bae3b9e42415649 768067b75a8807c5	29	13	738538b8c6a62c4e 6d208dbbb9bdeaaa	33
5	4241564918b3fa41 5a8807c5488dbe94	26	14	c6a62c4e56b0bd75 b9bdeeaad2c3a56f	30
6	18b3fa419616fe23 488dbe94aba7fe53	26	15	56b0bd7575e8fd8f d2c3a56f2765c1fb	33
7	9616fe2367117cf2 aba7fe53177d21e4	27	16	75e8fd8f25896490 2765c1fb01263dc4	30
8	67117cf2c11bfc09 177d21e4548f1de4	32	<b>IP-1</b>	da02ce3a89ecac3b ee92b50606b62b0b	30

# Mode DES

- DES dapat dioperasikan dengan mode ECB, CBC, OFB, CFB, dan mode *counter*.
- Namun karena kesederhanaannya, mode ECB lebih sering digunakan pada paket komersil, sehingga dikenal dengan nama DES-ECB.

# Kunci-kunci lemah DES

- **Kunci lemah DES** adalah kunci  $K$  sedemikian sehingga  $E_K(E_K(x)) = x$  untuk semua plainteks  $x$ , yang artinya enkripsi dan dekripsi menjadi sama.
  - Kunci lemah ini menghasilkan pembangkitan kunci putaran yang sama pada semua putaran.
- DES memiliki 4 kunci lemah (masing-masing panjangnya 56 bit), dinyatakan dalam kode heksadesimal sbb:
  - 0000000000000000
  - 0000000FFFFFFFFF
  - FFFFFFFF00000000
  - FFFFFFFFFFFFFFFF
- Kunci lemah seharusnya dihindari.





# Implementasi DES

- DES sudah diimplementasikan baik sebagai perangkat lunak maupun perangkat keras.
- Dalam bentuk perangkat keras, DES diimplementasikan di dalam *chip*. Setiap detik *chip* ini dapat mengenkripsikan 16,8 juta blok (atau 1 gigabit per detik).
- Implementasi DES ke dalam perangkat lunak dapat melakukan enkripsi 32.000 blok per detik (dijalankan pada komputer *mainframe* IBM 3090, yaitu komputer tercepat saat itu / tahun 1976).

# Keamanan DES

- Keamanan DES ditentukan oleh kunci.
- Panjang kunci eksternal DES hanya 64 bit, tetapi yang dipakai hanya 56 bit.
- Pada rancangan awal, panjang kunci yang diusulkan IBM adalah 128 bit, tetapi atas permintaan NSA, panjang kunci diperkecil menjadi 56 bit.
- Dengan panjang kunci 56 bit akan terdapat  $2^{56}$  atau 72.057.594.037.927.936 kemungkinan kunci.
- Ruang kunci (*key space*) DES sebanyak  $2^{56}$  terlalu kecil, tidak aman karena kunci dapat ditemukan dengan serangan *exhaustive key search (brute force attack)*
- Jika serangan *exhaustive key search (brute force attack)* dengan menggunakan prosesor paralel, andaikan dalam satu detik dapat dikerjakan satu juta serangan. Jadi seluruhnya diperlukan 1142 tahun untuk menemukan kunci yang benar.

- Misalkan kita mengetahui pasangan potongan plaintext ( $p$ ) dan cipherteks ( $c$ ) yang berkoresponden.
- Lakukan serangan *brute force* (atau *exhaustive key search attack*) dengan mencoba semua kemungkinan kunci:

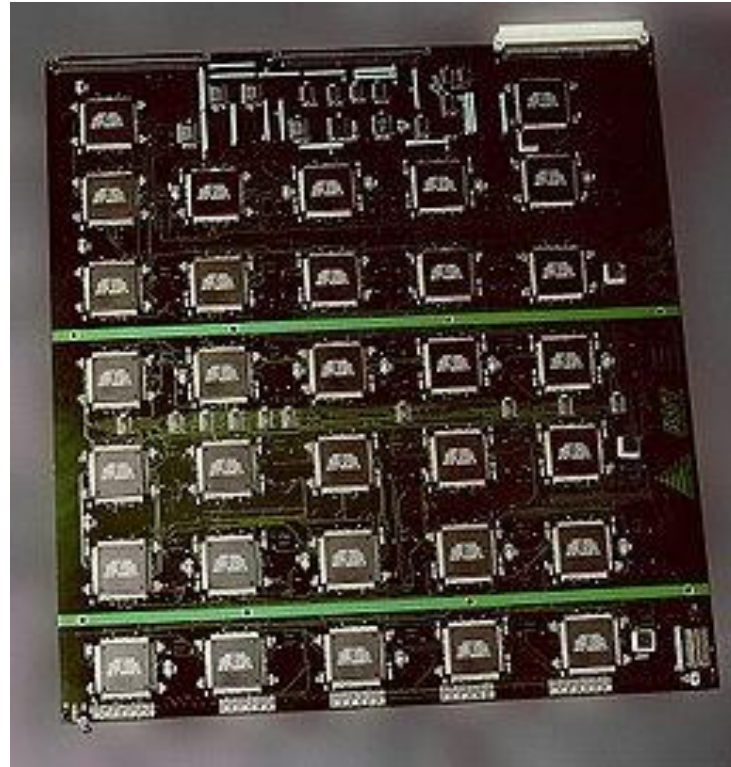
```
for (k=0; k<256; k++) {  
    if (DES(k,p)==c) {  
        printf("Key is %x\n", k);  
        break;  
    }  
}
```

- Kunci diharapkan ditemukan sebelum  $2^{55}$  iterasi.

Dikutip dari Wiki:

- In 1997, [RSA Security](#) sponsored a series of contests, offering a \$10,000 prize to the first team that broke a message encrypted with DES for the contest.
- That contest was won by the [DESCHALL Project](#), led by Rocke Verser, [Matt Curtin](#), and Justin Dolske, using idle cycles of thousands of computers across the Internet.

- Tahun 1998, *Electronic Frontier Foundation (EFE)* merancang dan membuat perangkat keras khusus untuk menemukan kunci DES secara *exhaustive key search* dengan biaya \$250.000 dan diharapkan dapat menemukan kunci selama 5 hari.
- Tahun 1999, kombinasi perangkat keras *EFE* dengan kolaborasi internet yang melibatkan lebih dari 100.000 komputer dapat menemukan kunci DES kurang dari 1 hari.



The [EFF](#)'s US\$250,000 [DES cracking machine](#) contained 1,856 custom chips and could brute force a DES key in a matter of days — the photo shows a DES Cracker circuit board fitted with several Deep Crack chips (Sumber Wikipedia).

- Their motivation was to show that DES was breakable in practice as well as in theory: *"There are many people who will not believe a truth until they can see it with their own eyes. Showing them a physical machine that can crack DES in a few days is the only way to convince some people that they really cannot trust their security to DES."*
- The machine brute-forced a key in a little more than 2 days search.

- Pengisian kotak-S DES masih menjadi misteri. Nilai-nilai awal S-box sudah diubah.
- Sebenarnya delapan putaran sudah cukup untuk membuat cipherteks sebagai fungsi acak dari setiap bit plainteks dan setiap bit kunci.
- Namun dari penelitian dengan melakukan kriptanalisis, DES dengan jumlah putaran yang kurang dari 16 ternyata dapat dipecahkan dengan *known-plaintext attack*.



# Demo online DES

<https://encode-decode.com/des-encrypt-online/>

The screenshot shows a web browser window with the URL <https://encode-decode.com/des-encrypt-online/>. The page title is "des encrypt & decrypt online". The site has a navigation menu with links for "encoding & decoding", "hash generation", "encryption & decryption", and "guide & faq".

The main interface includes a "supported encryptions:" dropdown menu currently set to "des". There are two text input fields: the left one contains the Indonesian text "Malam ini hujan turun deras sekali. Semoga tidak banjir. Kasihan orang2 di dekat sungai sana.", and the right one contains the encrypted string "QDQi4d9kP5WbvPG79EhMEcINwPKPGVVD/X2SISi5wP06UUyJP0b2cib7WT87e8oTKFItpvbkirJkJ6Untr+9Pn6V7hU2cAJqhViFBeeFU9RXg5TIq+torPmFrN3tVI/j".

Below the input fields are two green buttons: "Encrypt string →" and "← Decrypt string". On the left side, there is a vertical stack of social media sharing icons for Facebook, Twitter, Google+, Pinterest, LinkedIn, and Email. The Facebook share text is "mengapa".

At the bottom of the browser window, a video player is partially visible with the title "Data Encryption Standard (DES): Understanding the Limitations". The Windows taskbar at the very bottom shows the system tray with the date and time "8:36 PM 2/12/2023" and a temperature of "21°C".

# Program enkripsi – dekripsi DES dengan Python

```
#import library crypto dan base64
from Crypto.Cipher import DES
from Crypto import Random
import base64
```

```
def enkripsi (pesan):
    print("Plainteks: \n", pesan)
    print("Ketikkan kunci (8 karakter):")
    kunci = input()

    cipher = DES.new (kunci,DES.MODE_ECB)      #defenisikan mode DES ECB
    cipherteks = base64.b64encode (cipher.encrypt(pesan))  #enkripsi pesan

    print("\n")
    print("Cipherteks: \n", cipherteks)      #tampilkan ciphertext
```

```
def dekripsi(pesan):  
  
    print("Cipherteks: \n", pesan)  
  
    print("Ketikkan kunci (8 karakter):")  
    kunci = input()  
  
    cipher = DES.new(kunci, DES.MODE_ECB)  
  
    #dekripsi pesan  
    plainteks = cipher.decrypt (base64.b64decode(pesan))  
  
    print("\n")  
  
    #tampilkan plainteks  
    print ("Pesan setelah didekripsi adalah: \n", plainteks)
```

```

def ProgramEnkripsiDekripsiDES():
    print("-- Enkripsi dan dekripsi pesan dengan DES --")
    print("1: Enkripsi pesan")
    print("2: Dekripsi pesan")

    pilih = input()
    if pilih == '1':
        print("Ketikkan pesan yang akan dienkripsi:")
        pesan = input()
        n = len(pesan)
        if n % 8 != 0:
            pesan = pesan + ' ' * (8 - n % 8)    #padding dengan spasi
        print("Enkripsi pesan...")
        enkripsi(pesan)
    elif pilih == '2':
        print("Ketikkan pesan yang akan didekripsi:")
        pesan = input()
        print("Dekripsi pesan...")
        dekripsi(pesan)
    else:
        print("Pilihan salah")

```

# Run program:

Enkripsi

```
ProgramEnkripsiDekripsiDES()
```

```
-- Enkripsi dan dekripsi pesan dengan DES --
```

```
1: Enkripsi pesan
```

```
2: Dekripsi pesan
```

```
1
```

```
Ketikkan pesan yang akan dienkripsi:
```

```
Hari tanggal 17 Februari 2023, semoga kita sehat selalu gaess...
```

```
Enkripsi pesan...
```

```
Plainteks:
```

```
  Hari tanggal 17 Februari 2023, semoga kita sehat selalu gaess...
```

```
Ketikkan kunci (8 karakter):
```

```
abcdefgh
```

```
Cipherteks:
```

```
b'qjUUi+maltZluPer4W6RARzrbrGJMETx0d/Boq7YNzWv9f65xPT2y06RZyq2Q0sjyF16GLDVEwe2px4YtOx/gA=='
```

```
ProgramEnkripsiDekripsiDES()
```

```
-- Enkripsi dan dekripsi pesan dengan DES --
```

```
1: Enkripsi pesan
```

```
2: Dekripsi pesan
```

```
2
```

```
Ketikkan pesan yang akan didekripsi:
```

```
qjUUi+maltZluPer4W6RARZrbrGJMETx0d/Boq7YNzWv9f65xPT2y06RZyq2Q0sjyF16GLDVEwe2px4Yt0x/gA==
```

```
Dekripsi pesan...
```

```
Cipherteks:
```

```
qjUUi+maltZluPer4W6RARZrbrGJMETx0d/Boq7YNzWv9f65xPT2y06RZyq2Q0sjyF16GLDVEwe2px4Yt0x/gA==
```

```
Ketikkan kunci (8 karakter):
```

```
abcdefgh
```

```
Pesan setelah didekripsi adalah:
```

```
b'Hari tanggal 17 Februari 2023, semoga kita sehat selalu gaess...'
```