Algoritma Kriptografi Modern (Bagian 2)

Mode Operasi Cipher Blok

- Mode operasi: berkaitan dengan cara blok dioperasikan

- Ada 4 mode operasi *cipher* blok:
  1. *Electronic Code Book* (*ECB*)
  2. *Cipher Block Chaining* (*CBC*)
  3. *Cipher Feedback* (*CFB*)
  4. *Output Feedback (OFB)*

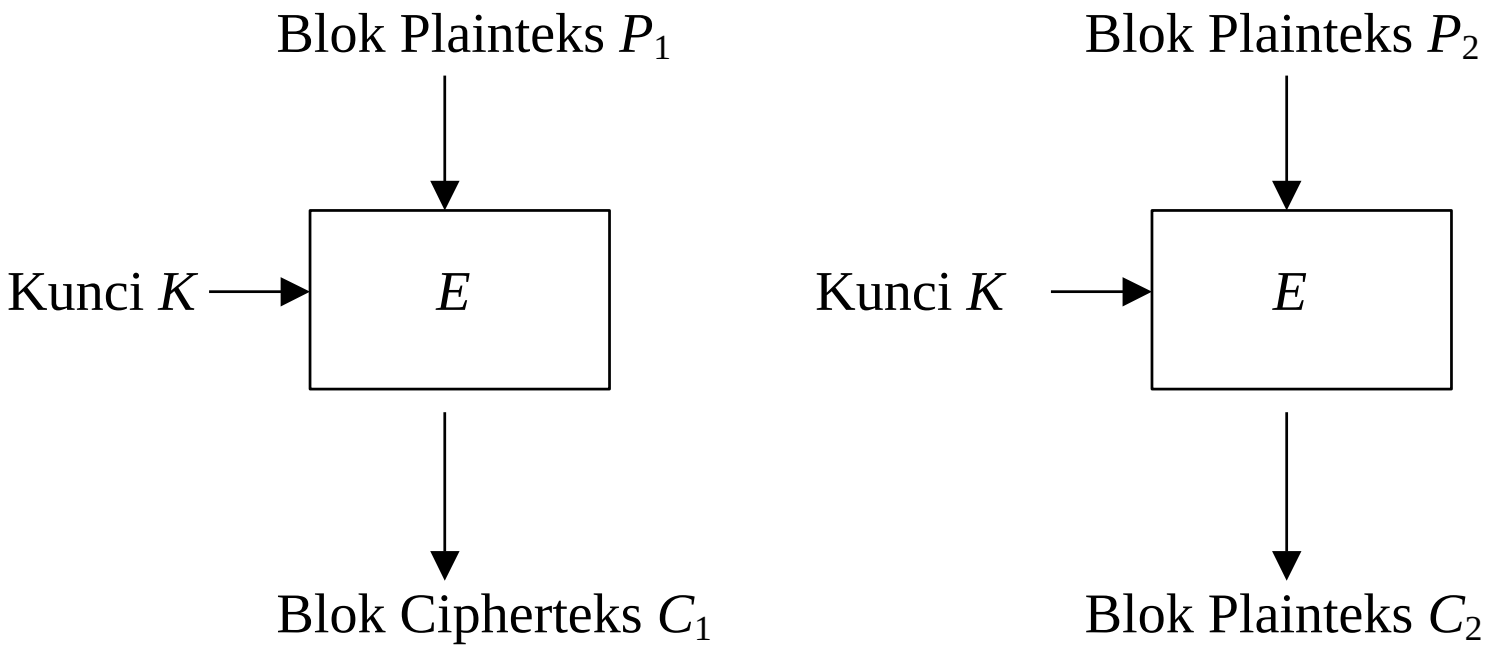
Electronic Code Book (ECB)

- Setiap blok plainteks P_i dienkripsi secara individual dan independen 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.



Gambar 9.4 Skema enkripsi dan dekripsi dengan mode *ECB*

- Contoh:

  Plainteks: `10100010001110101001`

  Bagi plainteks 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 adalah: XOR-kan blok plainteks P_i dengan *K*, kemudian geser secara *wrapping* bit-bit dari $P_i \oplus K$ satu posisi ke kiri.

Enkripsi:

	1010	0010	0011	1010	1001	
	1011	1011	1011	1011	1011	⊕
Hasil *XOR*:	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.

- 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*)

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

- Namun, semakin besar ukuran blok, semakin besar pula ukuran buku kodenya.

- Misalkan jika blok berukuran 64 bit, maka buku kode terdiri dari $2^{64} – 1$ 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.

Keuntungan Mode *ECB*

1. Karena tiap blok plainteks dienkripsi secara independen, maka kita tidak perlu mengenkripsi file secara linear.

   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 arsip (*file*) yang diakses secara acak, misalnya arsip-arsip basisdata.

- Jika basisdata dienkripsi dengan mode *ECB*, maka sembarang *record* dapat dienkripsi atau didekripsi 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 waktu dekripsi.

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

Kelemahan ECB

1. Karena bagian plainteks sering berulang (sehingga terdapat blok-blok plainteks yang sama), maka hasil enkripsinya menghasilkan blok cipherteks yang sama

   ➔ contoh berulang: spasi panjang

   ➔ mudah diserang secara statisitik

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

   Contoh: Seseorang mengirim pesan

   `Uang ditransfer lima satu juta rupiah`

Andaikan kriptanalis 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 kriptanalis berhasil mendekripsi keseluruhan blok cipherteks menjadi plainteks semula.

Kriptanalis membuang blok cipheteks ke-8 dan 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.

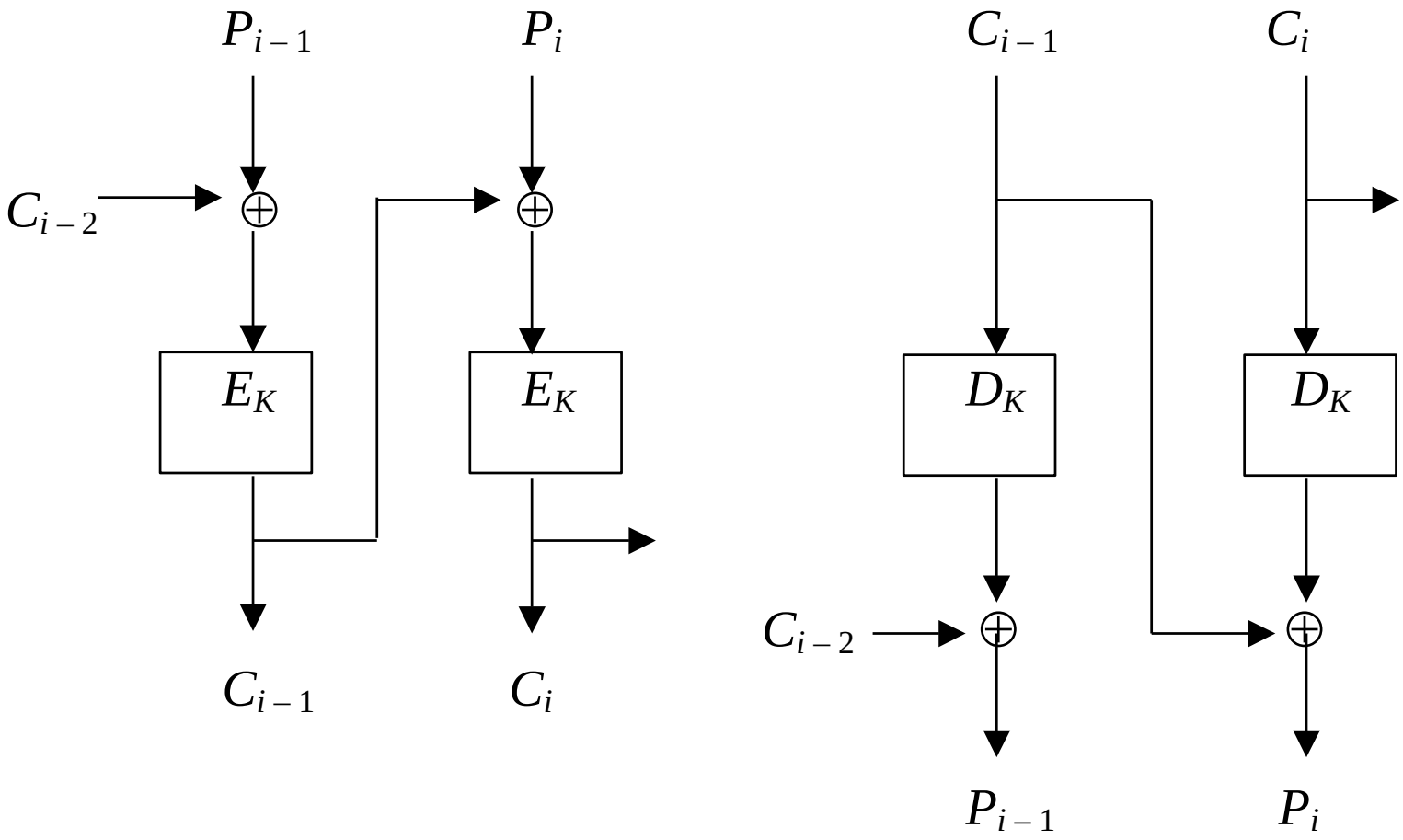
- Cara mengatasi kelemahan ini: enkripsi tiap blok individual bergantung pada semua blok-blok sebelumnya.

- Akibatnya, blok plainteks yang sama dienkripsi menjadi blok cipherteks berbeda.

- Prinsip ini mendasari mode *Cipher Block Chaining*.

Cipher Block Chaining (CBC)

- Tujuan: membuat ketergantungan antar blok.
- 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

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

Dekripsi

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

Gambar 8.5 Skema enkripsi dan dekripsi dengan mode *CBC*

- 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.

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)

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)

Misalkan fungsi enkripsi E yang sederhana (tetapi lemah) adalah dengan meng-XOR-kan blok plainteks P_i dengan K, kemudian geser secara *wrapping* bit-bit dari $P_i \oplus K$ satu posisi ke kiri.

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_2 = 1111 (atau F dalam HEX)

Demikian seterusnya, sehingga plainteks dan cipherteks hasilnya adalah:

Pesan (plainteks):	`A23A9`
Cipherteks (mode *ECB*):	`23124`
Cipherteks (mode *CBC*):	`27FBF`

Keuntungan Mode CBB

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

Inilah alasan utama penggunaan mode *CBC* digunakan.

Kelemahan Mode CBC

1. Kesalahan satu bit pada sebuah blok plainteks akan merambat pada blok cipherteks yang berkoresponden dan 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)

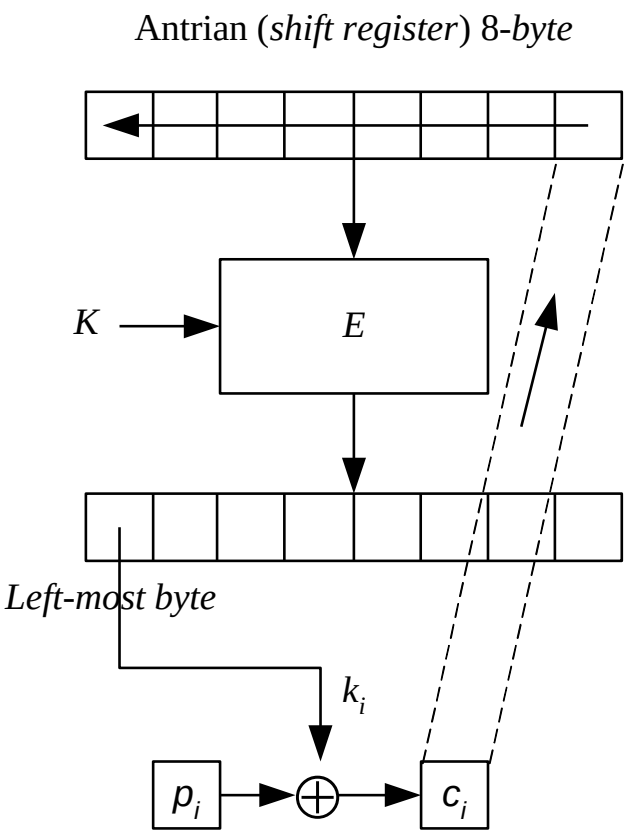
- Mengatasi kelemahan pada mode *CBC* jika diterapkan pada komunikasi data (ukuran blok yang belum lengkap)
- Data dienkripsikan dalam unit yang lebih kecil daripada ukuran blok.
- Unit yang dienkripsikan dapat berupa bit per bit (jadi seperti *cipher* aliran), 2 bit, 3-bit, dan seterusnya.
- Bila unit yang dienkripsikan satu karakter setiap kalinya, maka mode *CFB*-nya disebut *CFB* 8-bit.

- *CFB n*-bit mengenkripsi plainteks sebanyak *n* bit setiap kalinya, $n \leq m$ (*m* = ukuran blok).

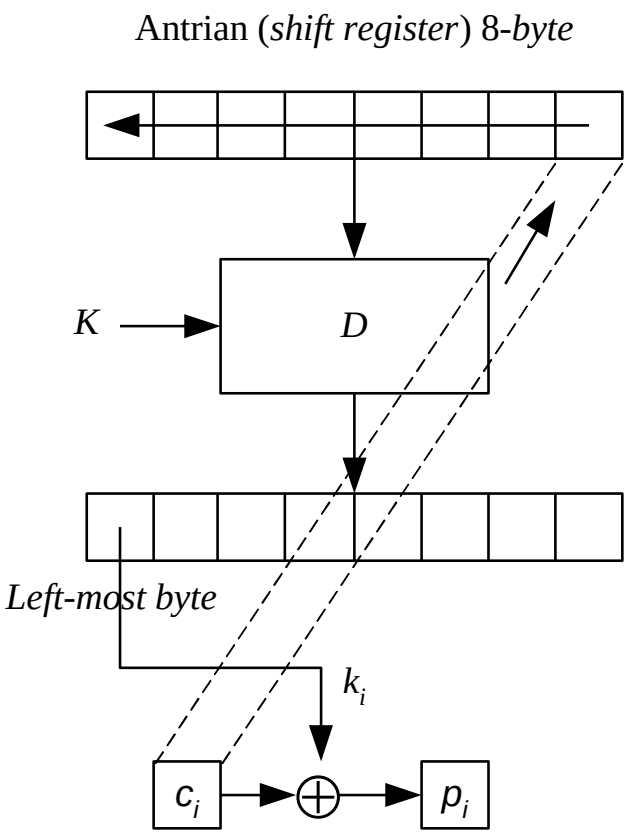
- Dengan kata lain, *CFB* mengenkripsikan *cipher* blok seperti pada *cipher* aliran.

- 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



(a) Enkripsi



(b) Dekripsi

Secara formal, mode *CFB n*-bit dapat dinyatakan sebagai:

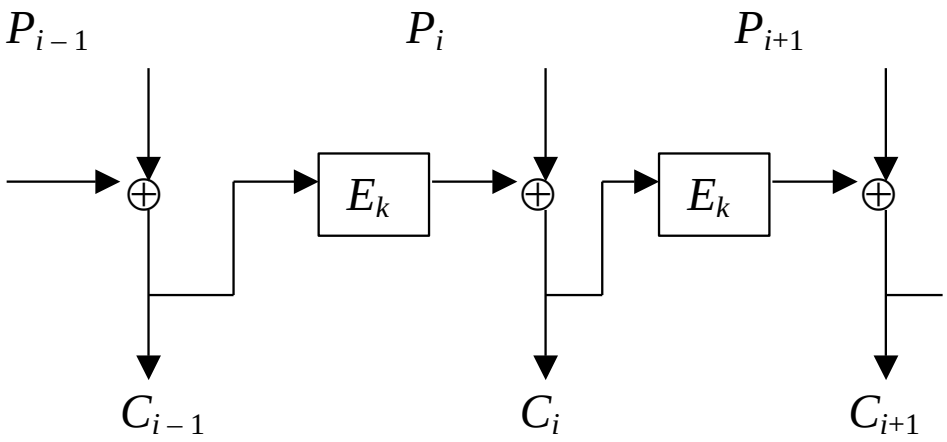
Proses Enkripsi: $C_i = P_i \oplus MSB_m(E_K(X_i))$
$X_{i+1} = LSB_{m-n}(X_i) \,||\, C_i$

Proses Dekripsi: $P_i = C_i \oplus MSB_m(D_K(X_i))$
$X_{i+1} = LSB_{m-n}(X_i) \,||\, C_i$

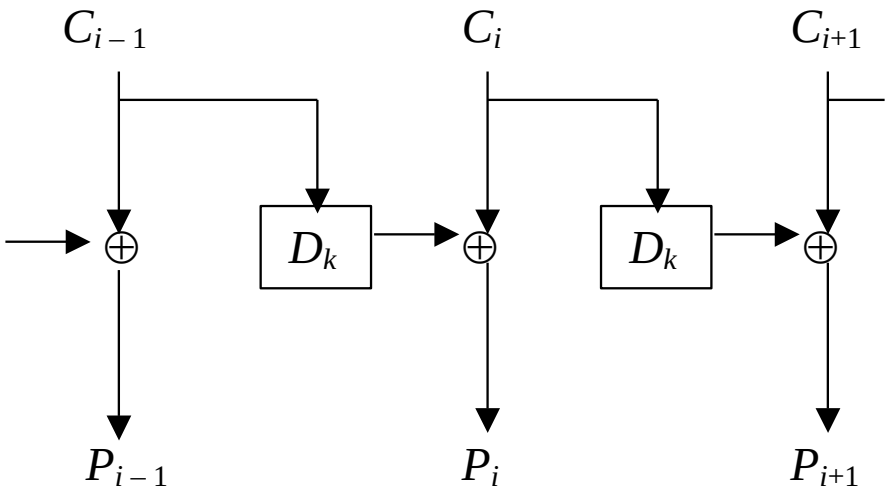
yang dalam hal ini,

X_i = isi antrian dengan X_1 adalah *IV*
E = fungsi enkripsi dengan algoritma *cipher* blok.
K = kunci
m = panjang blok enkripsi
n = panjang unit enkripsi
|| = operator penyambungan (*concatenation*)
MSB = Most Significant Byte
LSB = Least Significant Byte

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



Enkripsi *CFB*



Dekripsi *CFB*

- Dari Gambar di atas 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.

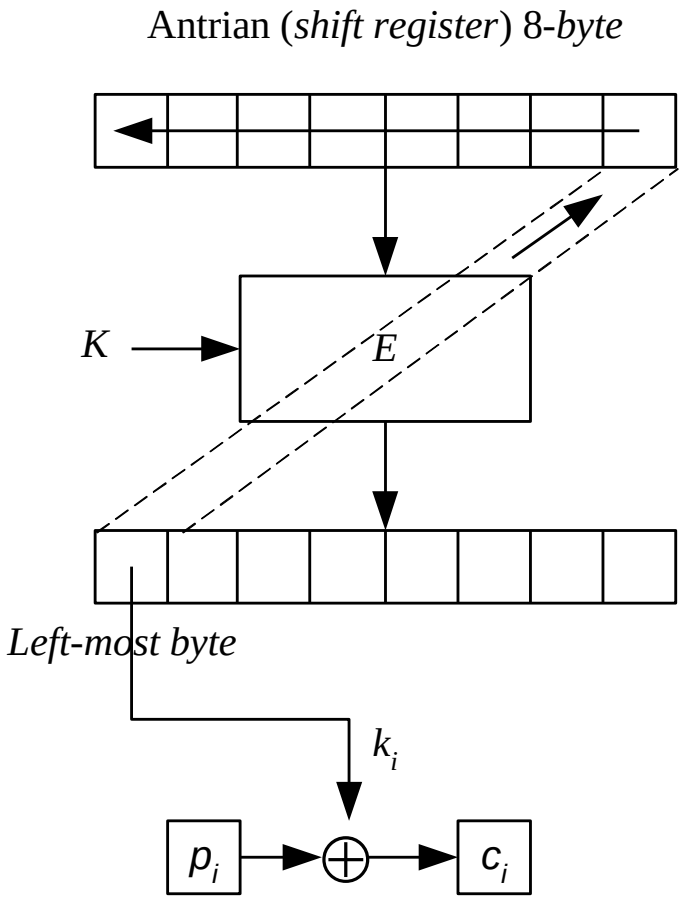
- Hal yang kebalikan terjadi pada proses dekripsi.

Output-Feedback (OFB)

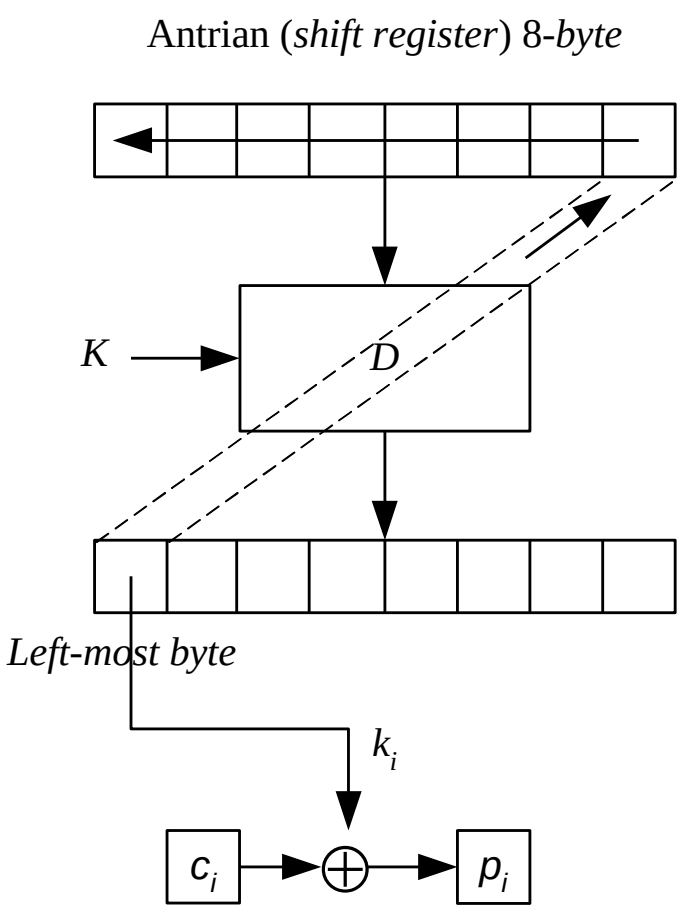
- Mode *OFB* mirip dengan mode *CFB*, kecuali *n*-bit dari hasil enkripsi terhadap antrian disalin menjadi elemen posisi paling kanan di antrian.

- Dekripsi dilakukan sebagai kebalikan dari proses enkripsi.

- Gambar berikut adalah mode *OFB* 8-bit yang bekerja pada blok berukuran 64-bit (setara dengan 8 *byte*).

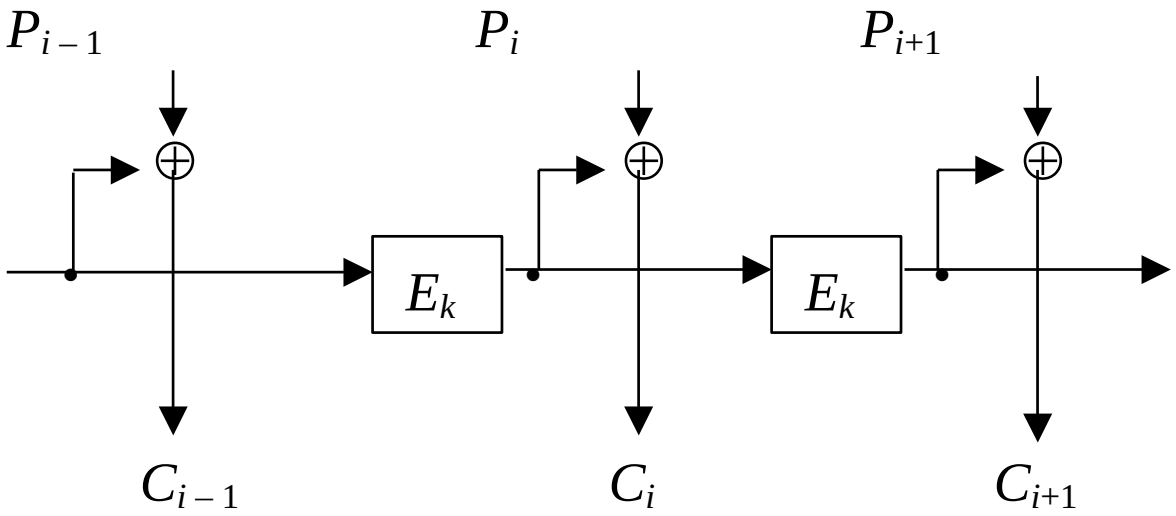


(a) Enkripsi



(b) Dekripsi

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



Enkripsi *OFB*

Gambar 8.9 Enkripsi mode *OFB* n-bit untuk blok n-bit

- 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 di-digitisasi, seperti suara atau video, yang dalam hal ini kesalahan 1-bit dapat ditolerir, tetapi penjalaran kesalahan tidak dibolehkan.

Prinsip-prinsip Perancangan *Cipher* Blok

1. Prinsip *Confusion* dan *Diffusion* dari Shannon.
2. *Cipher* berulang (*iterated cipher*)
3. Jaringan Feistel (*Feistel Network*)
4. Kotak-S (*S-box*)

Prinsip *Confusion* dan *Diffusion* dari Shannon.

- Banyak algoritma kriptografi klasik yang telah berhasil dipecahkan karena distribusi statistik plainteks dalam suatu bahasa diketahui.

- Claude Shannon dalam makalah klasiknya tahun 1949, *Communication theory of secrecy systems*, memperkenalkan prinsip *confusion* dan *diffusion* untuk membuat serangan statistik menjadi rumit.

- Dua prinsip tersebut menjadi panduan dalam merancang algoritma kriptografi.

1. *Confusion*

- Prinsip ini menyembunyikan hubungan apapun yang ada antara plainteks, cipherteks, dan kunci.

- Prinsip *confusion* membuat kriptanalis frustasi untuk mencari pola-pola statistik yang muncul pada cipherteks.

- *One-Time Pad* adalah contoh algoritma yang *confuse*.

- *Confusion* dapat direalisasikan dengan menggunakan algoritma substitusi yang kompleks.

- DES mengimplementasikan substitusi dengan menggunakan kotak-S.

2. Diffusion

- Prinsip ini menyebarkan pengaruh satu bit plainteks atau kunci ke sebanyak mungkin cipherteks.

- Sebagai contoh, pengubahan kecil pada plainteks sebanyak satu atau dua bit menghasilkan perubahan pada cipherteks yang tidak dapat diprediksi.

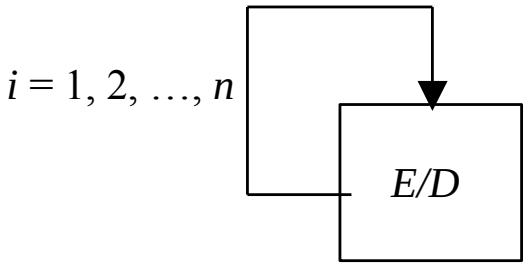
- Mode CBC dan CFB menggunakan prinsip ini

- Pada algoritma DES, *diffusion* direalisasikan dengan menggunakan operasi permutasi.

Cipher Berulang (*Iterated Cipher*)

- Fungsi transformasi sederhana yang mengubah plainteks menjadi cipherteks diulang sejumlah kali.

- Pada setiap putaran digunakan upa-kunci (*subkey*) atau kunci putaran (*round key*) yang dikombinasikan dengan plainteks.



- *Cpher* berulang dinyatakan sebagai

$C_i = f(C_{i-1}, K_i)$

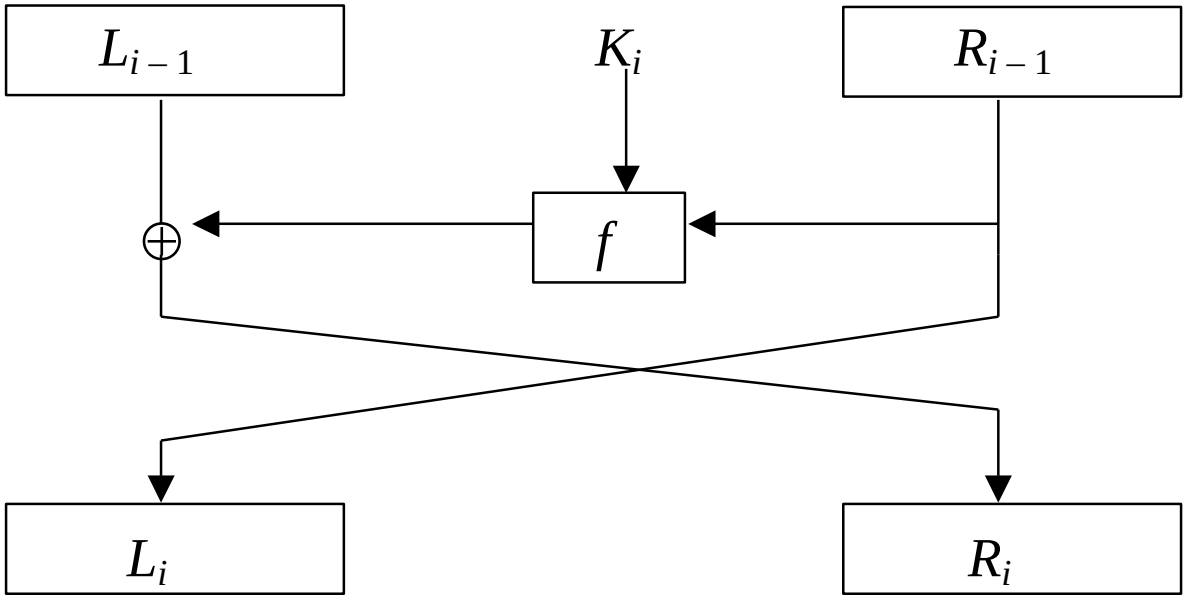
$i = 1, 2, \ldots, r$ (r adalah jumlah putaran).

K_i = upa-kunci (*subkey*) pada putaran ke-i

f = fungsi transformasi (di dalamnya terdapat operasi substitusi, permutasi, dan/atau ekspansi, kompresi).

Plainteks dinyatakan dengan C_0 dan cipherteks dinyatakan dengan C_r.

Jaringan Feistel (*Feistel Network*)



Gambar 8.10 Jaringan *Feistel*

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

- Jaringan *Feistel* banyak dipakai pada algoritma kriptografi *DES*, *LOKI*, *GOST*, *FEAL*, *Lucifer*, *Blowfish*, dan lain-lain karena model ini bersifat *reversible* untuk proses enkripsi dan dekripsi.

- Sifat *reversible* ini membuat kita tidak perlu membuat algoritma baru untuk mendekripsi cipherteks menjadi plainteks.

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

- Sifat *reversible* tidak bergantung pada fungsi *f* sehingga fungsi *f* dapat dibuat serumit mungkin.

Kotak-*S* (*S-box*)

- Kotak-*S* adalah matriks yang berisi substitusi sederhana yang memetakan satu atau lebih bit dengan satu atau lebih bit yang lain.

- Pada kebanyakan algoritma *cipher* blok, kotak-*S* memetakan *m* bit masukan menjadi *n* bit keluaran, sehingga kotak-*S* tersebut dinamakan kotak $m \times n$ *S-box*.

- Kotak-*S* merupakan satu-satunya langkah nirlanjar di dalam algoritma, karena operasinya adalah *look-up table*. Masukan dari operasi *look-up table* dijadikan sebagai indeks kotak-*S*, dan keluarannya adalah *entry* di dalam kotak-*S*.

Contoh: Kotak-*S* di dalam algoritma *DES* adalah 6 × 4 *S-box* yang berarti memetakan 6 bit masukan menjadi 4 bit keluaran. Salah satu kotak-*S* yang ada di dalam algoritma DES adalah sebagai berikut:

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

Baris diberi nomor dari 0 sampai 3
Kolom diberi nomor dari 0 sampai 15

Masukan untuk proses substitusi adalah 6 bit,

$$b_1b_2b_3b_4b_5b_6$$

Nomor baris dari tabel ditunjukkan oleh *string* bit b_1b_6
(menyatakan 0 sampai 3 desimal)

Nomor kolom ditunjukkan oleh *string* bit $b_2b_3b_4b_5$
(menyatakan 0 sampai 15)

- Misalkan masukan adalah 110100

  Nomor baris tabel = 10 (baris 2)

  Nomor kolom tabel = 1010 (kolom 10)

  Jadi, substitusi untuk 110100 adalah *entry* pada baris 2 dan kolom 10, yaitu 0100 (atau 4 desimal).

- DES mempunyai 8 buah kotak-S

- Pada AES kotak S hanya ada satu buah:

hex		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

S-BOX

19	a0	9a	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

hex	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

19

	a0	9a	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

hex	y															
	0	1	2	3	4	5	6	7				b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5				2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0				af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc				f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a				e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

d4