Bahan kuliah IF4020 Kriptografi

Kriptografi Modern
(Bagian 3: Block Cipher)

Oleh: Dr. Rinaldi Munir

Program Studi Informatika
Sekolah Teknik Elektro dan Informatika
ITB

Cipher Blok (*Block Cipher*)

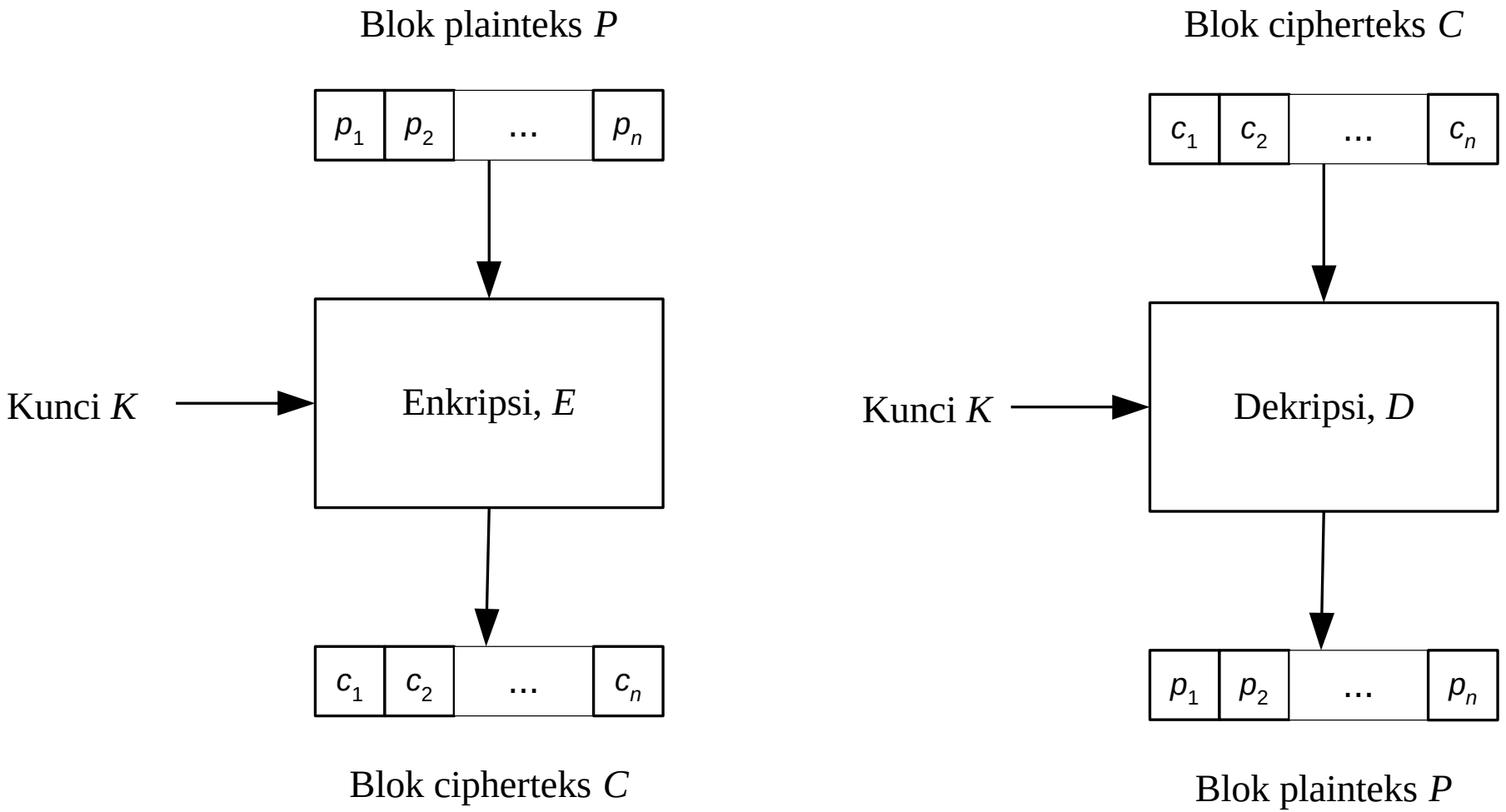
- Bit-bit plainteks dibagi menjadi blok-blok bit dengan panjang sama, misalnya 64 bit.
- Panjang blok cipherteks = panjang blok plainteks.
- Enkripsi dilakukan terhadap blok plainteks dengan bit-bit kunci
- Panjang kunci eksternal (yang diberikan oleh pengguna) tidak harus sama dengan panjang blok plainteks.

Blok plainteks berukuran n bit:

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

Blok cipherteks (C) berukuran n bit:

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



Skema enkripsi dan dekripsi pada *cipher* blok

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*

Electronic Code Book (ECB)

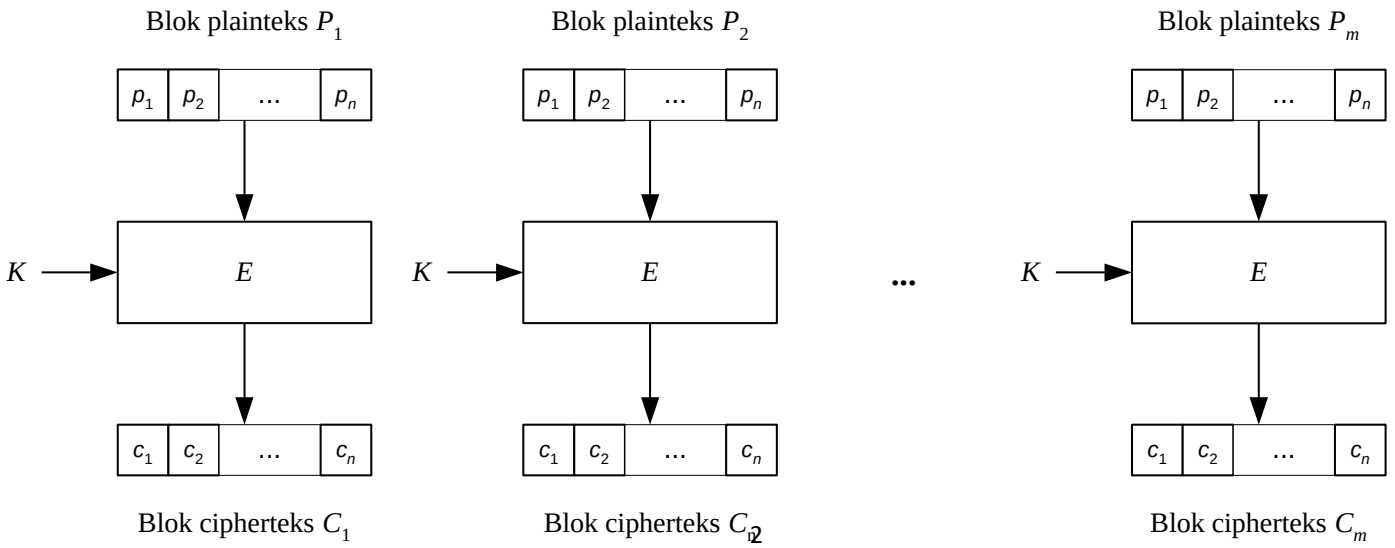
- Setiap blok plainteks P_i dienkripsi secara individual dan independen dari blok lainnya menjadi blok cipherteks C_i .

- Enkripsi: $C_i = E_K(P_i)$

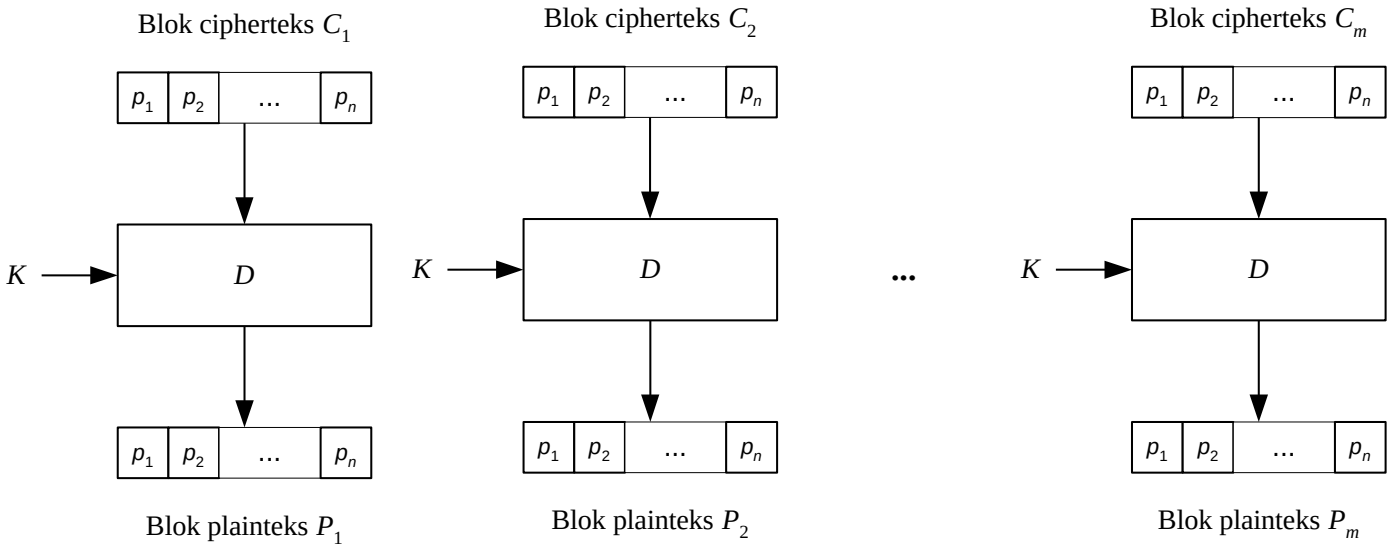
  Dekripsi: $P_i = D_K(C_i)$

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

Mode ECB



(a) Enkripsi



(b) Dekripsi

- 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:

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

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

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.

	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	

- 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 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 statisitik 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 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 mengethaui posisi pesan yang berkaitan dengan nominal uang, yaitu blok C_8 dan C_9.

Kriptanalis 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.

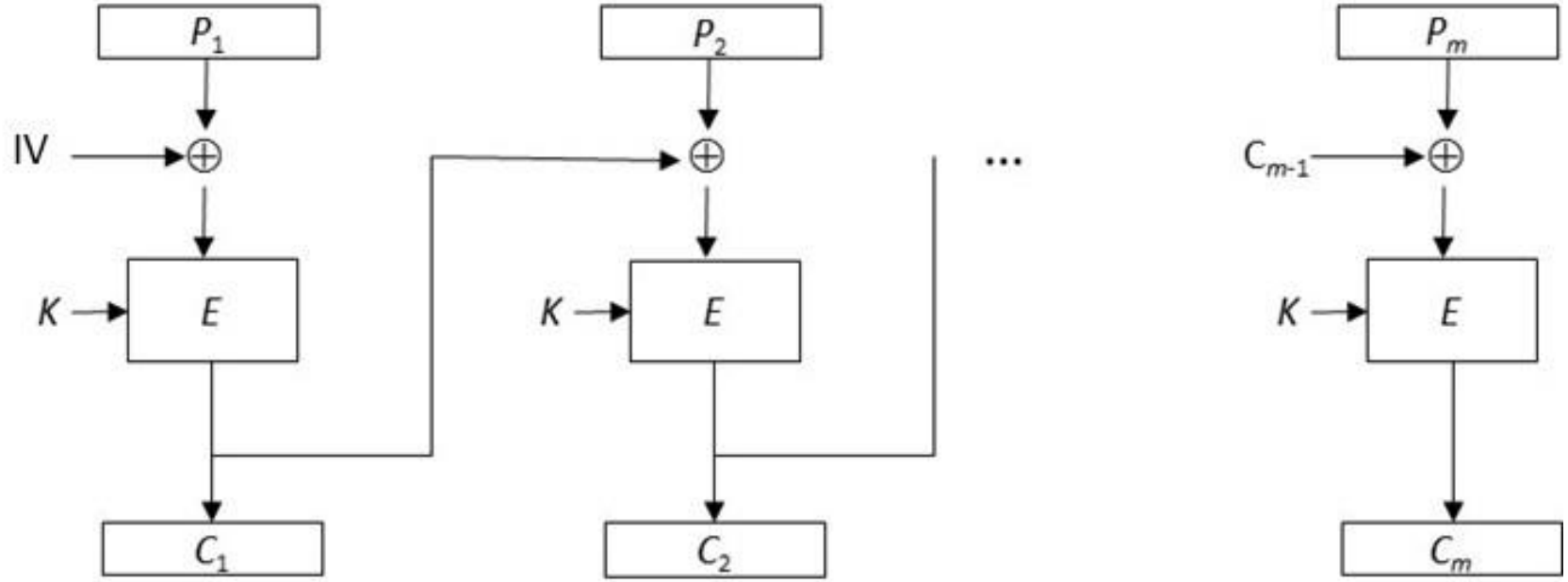
- Cara mengatasi kelemahan ini: enkripsi tiap blok individual bergantung pada semua blok-blok sebelumnya.

- Prinsip ini mendasari mode *Cipher Block Chaining* (CBC).

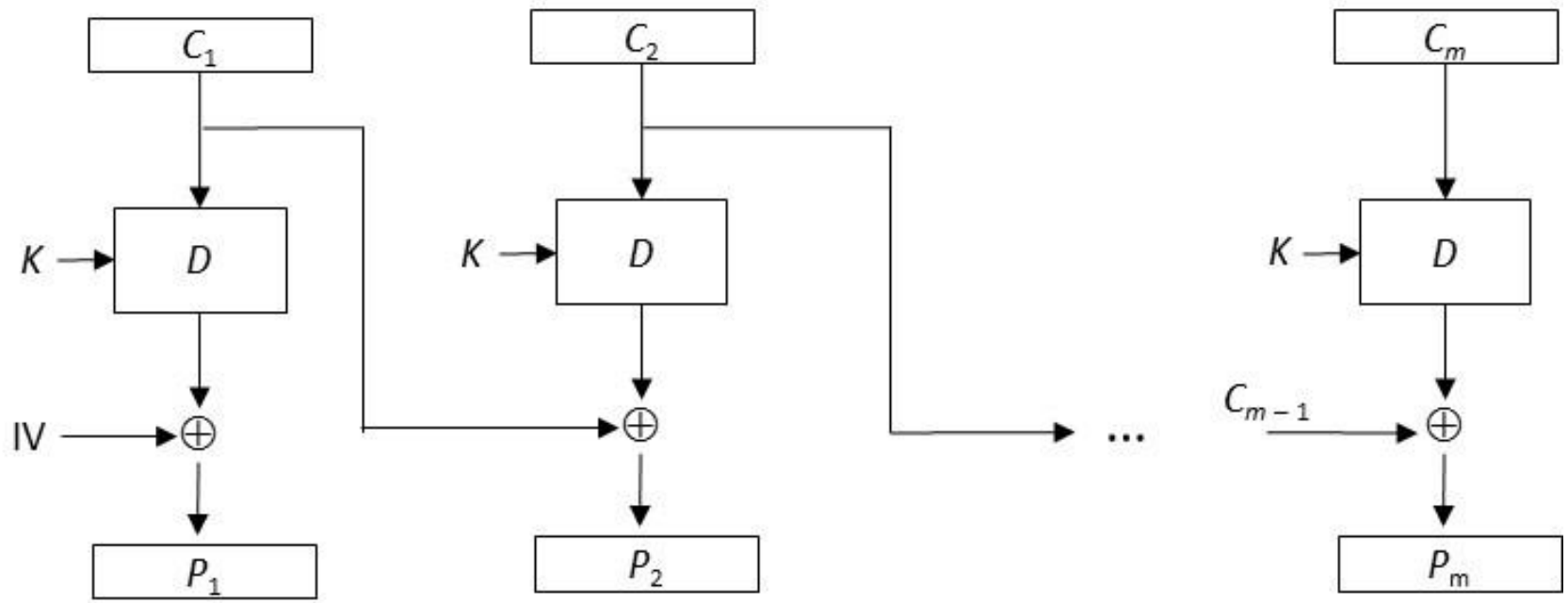
Cipher Block Chaining(CBC)

- Tujuan: membuat ketergantungan antar blok.
- 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*.

(a) Skema enkripsi mode CBC



(b) Skema dekripsi 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)

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.

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

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)

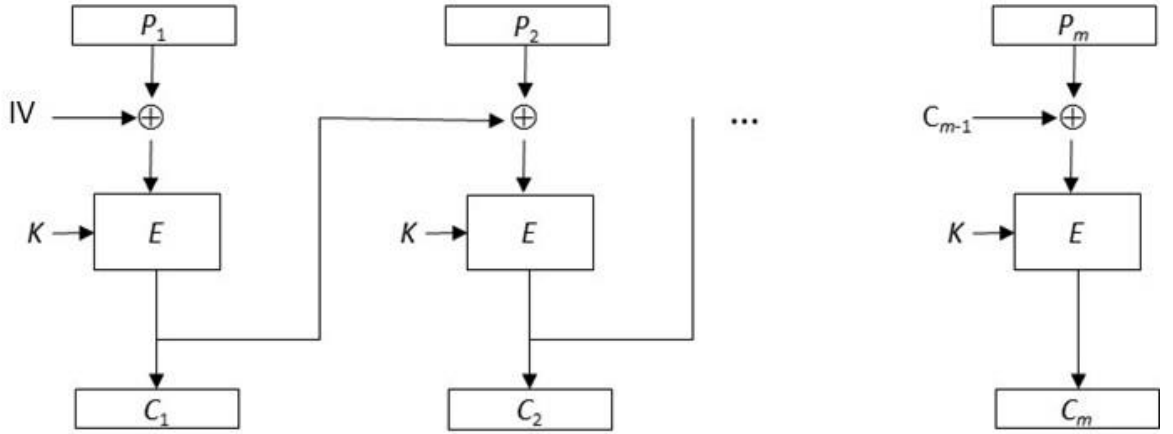
C_2 diperoleh sebagai berikut:

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

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

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

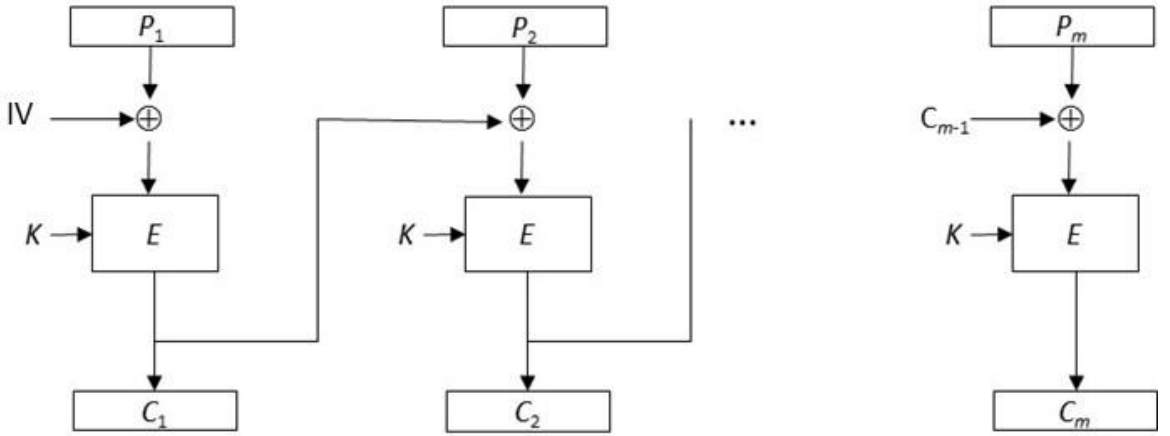
*B*lok-blok plainteks yang sama tidak selalu menghasilkan blok-blok cipherteks yang sama

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

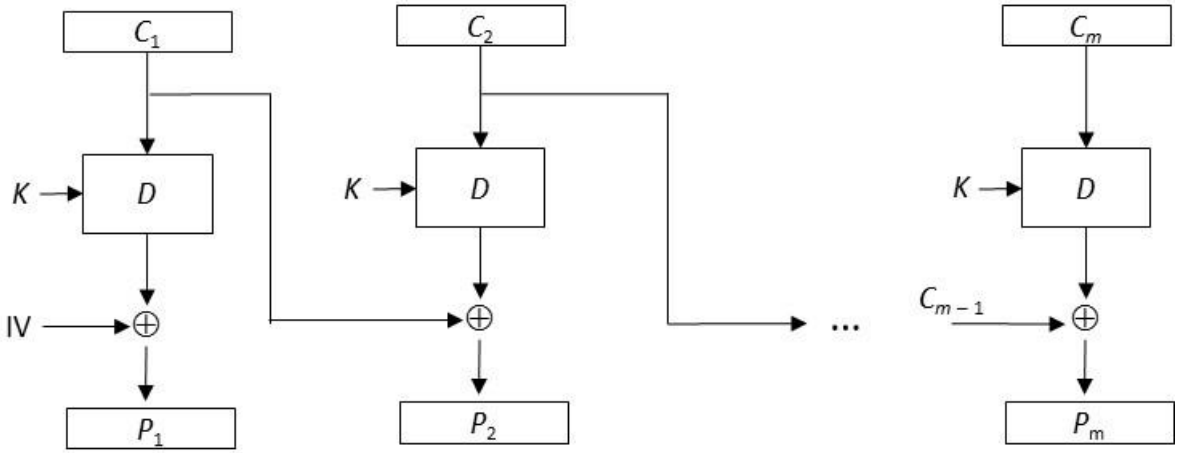
Inilah alasan utama penggunaan mode *CBC*.

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.

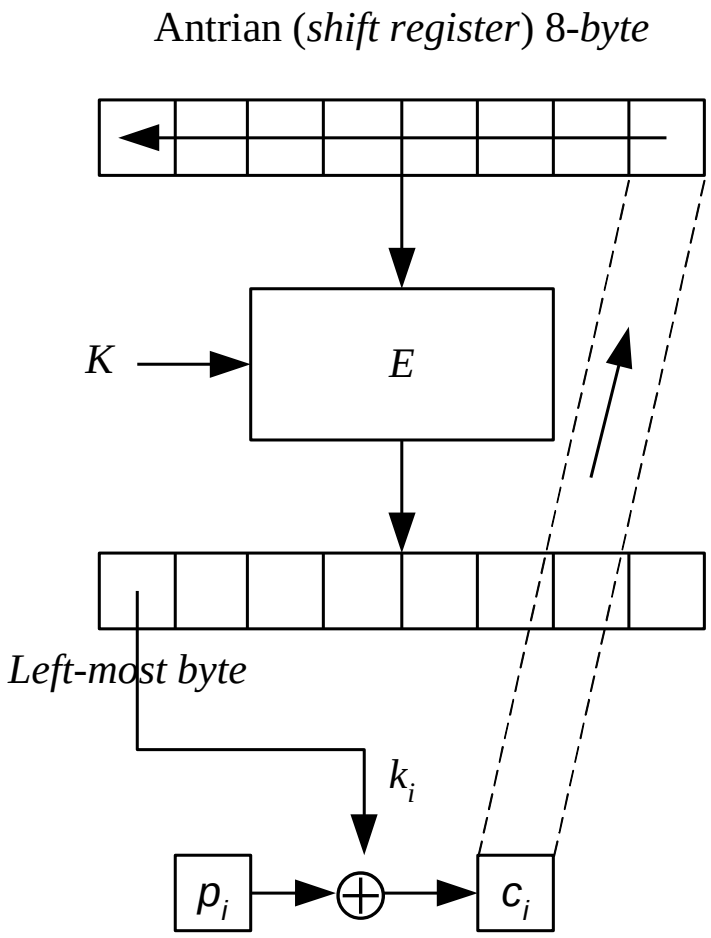
- *CFB n*-bit mengenkripsi plainteks sebanyak *n* bit setiap kalinya, $n \leq m$ (*m* = ukuran blok).

- Dengan kata lain, *CFB* n-bit meperlakukan *cipher* blok sama seperti pada *cipher* alir.

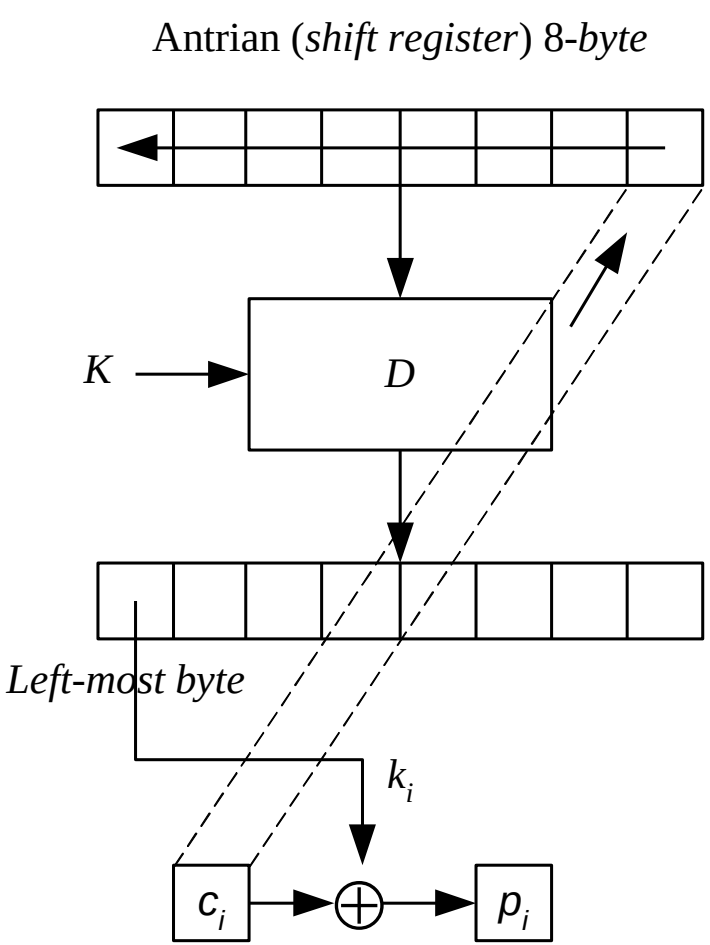
- Mode *CFB* membutuhkan sebuah antrian (*queue*) yang berukuran sama dengan ukuran blok masukan.

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

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

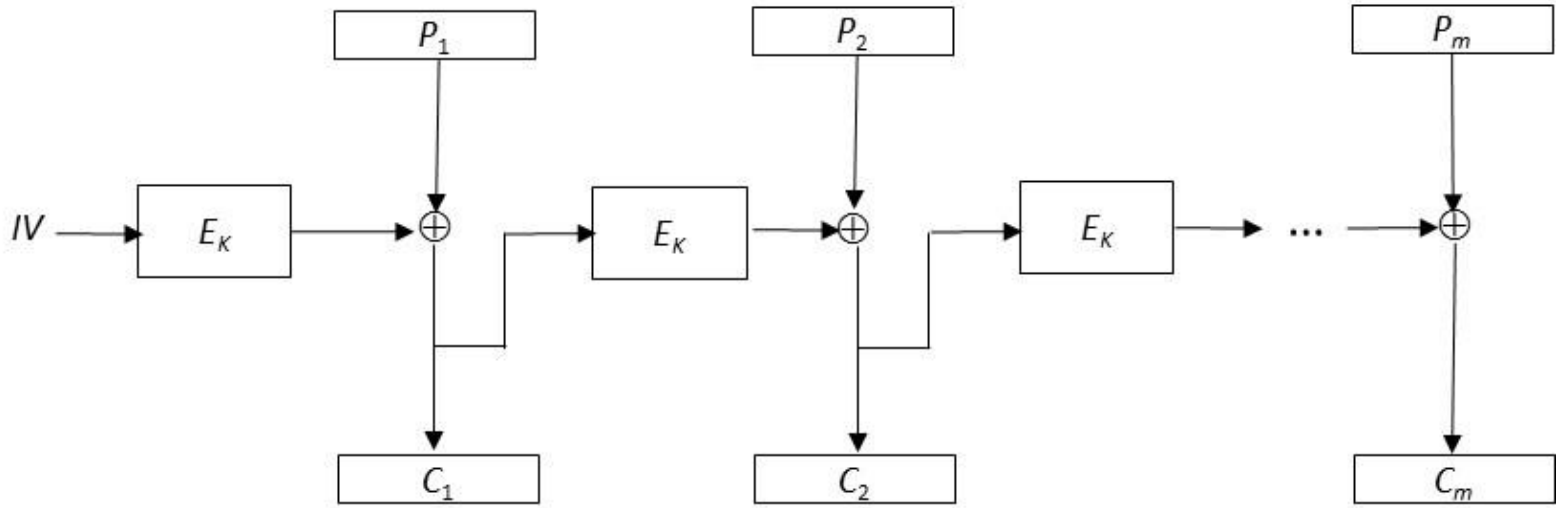
Proses Dekripsi: $P_i = C_i \oplus MSB_m(D_K(X_i))$
$X_{i+1} = LSB_{m-n}(X_i) \,||\, C_i$

yang dalam hal ini,

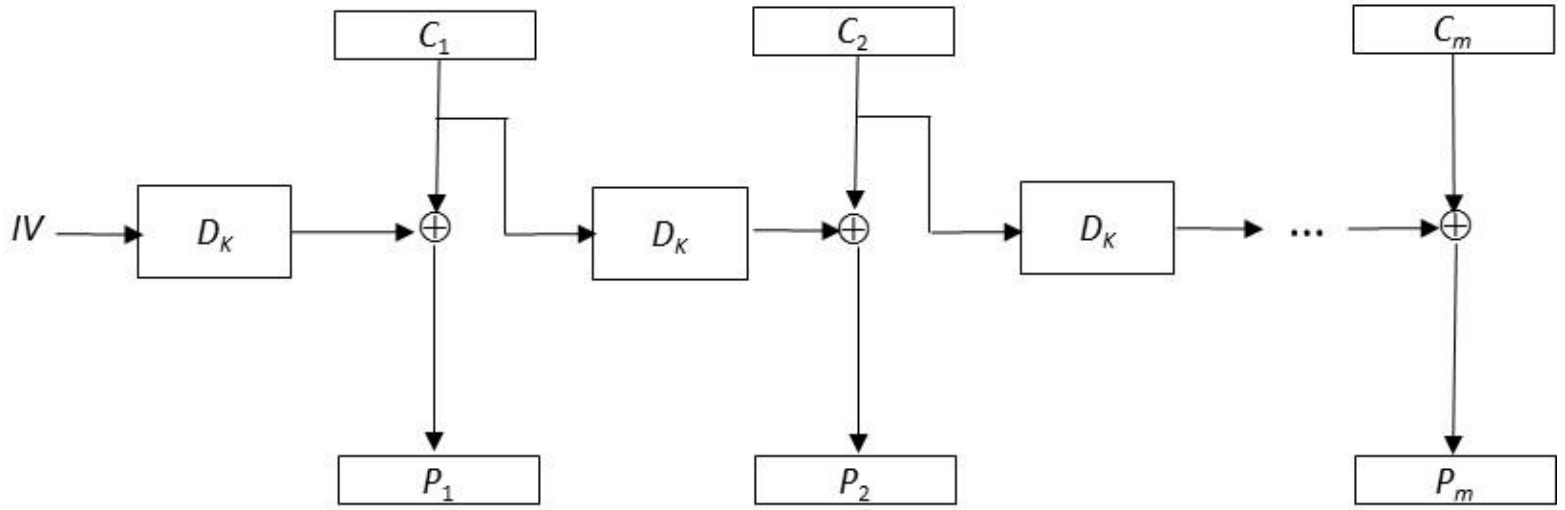
X_i = isi antrian dengan X_1 adalah *IV*
E = fungsi enkripsi dengan algoritma *cipher* blok.
K = kunci
m = panjang blok enkripsi
n = panjang unit enkripsi
|| = operator penyambungan (*concatenation*)
MSB = Most Significant Byte
LSB = Least Significant Byte

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

(a) Enkripsi

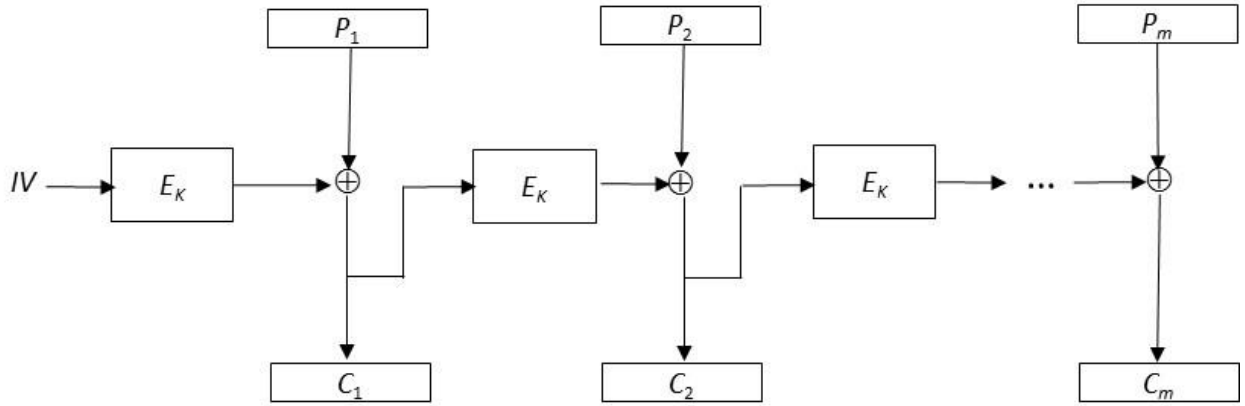


(b) Dekripsi



- Dari Gambar tersebut dapat dilihat bahwa:

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



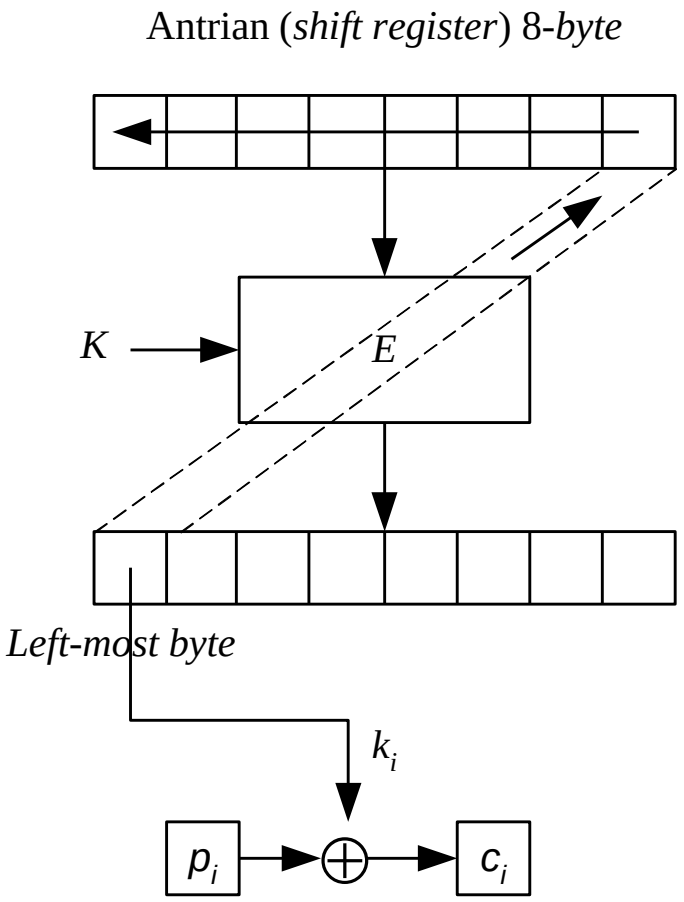
yang dalam hal ini, $C_0 = IV$.

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

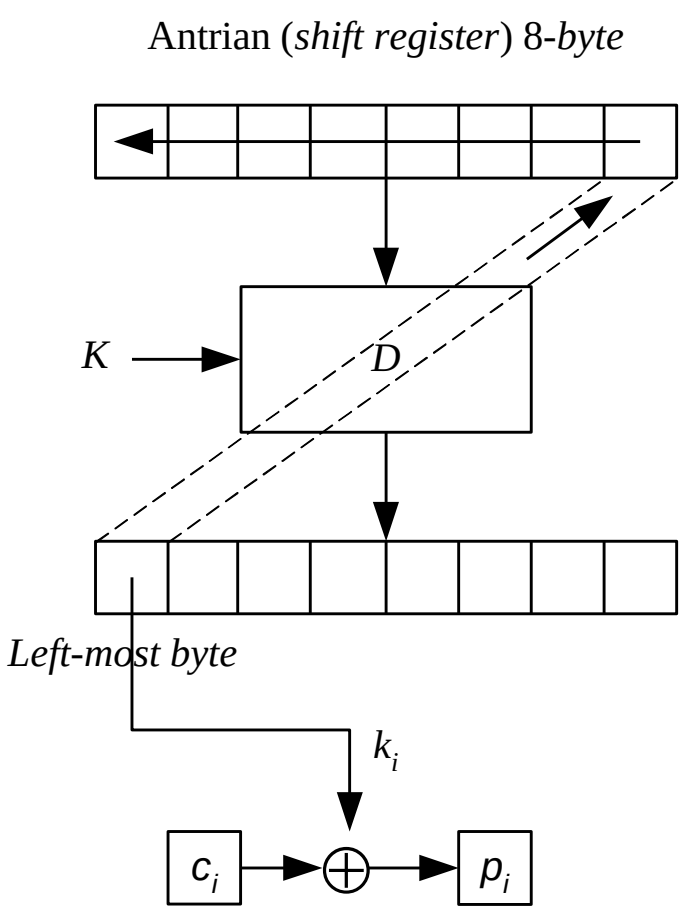
- Hal yang kebalikan terjadi pada proses dekripsi.

Output-Feedback (OFB)

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

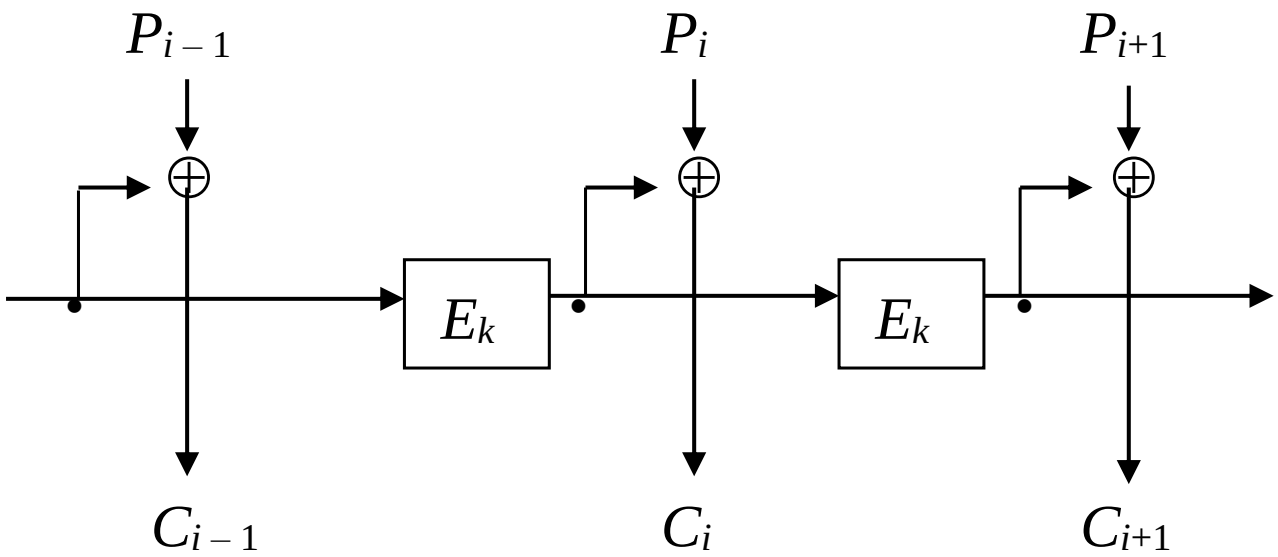


(a) Enkripsi



(b) Dekripsi

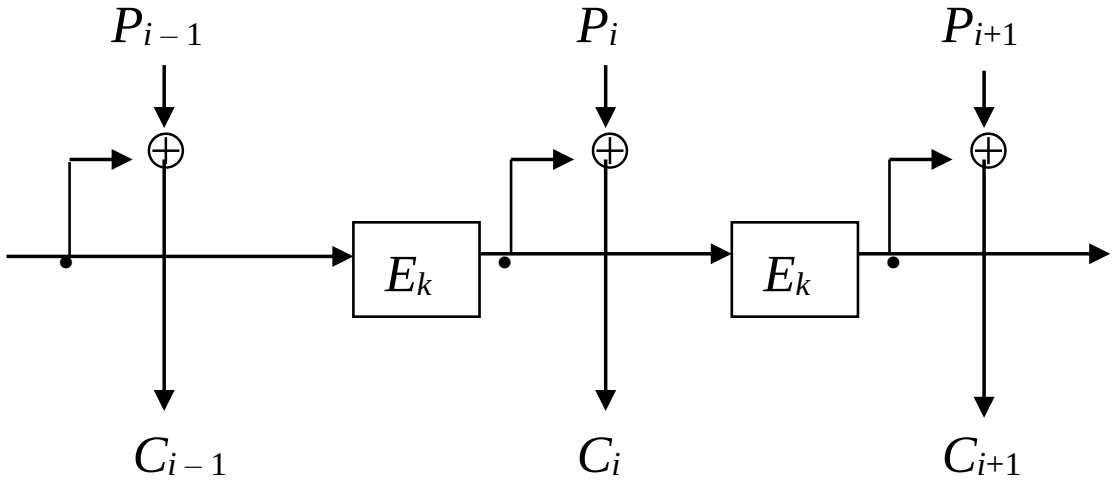
Jika $m = n$, maka mode *OFB* n-bit adalah seperti pada Gambar berikut



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.



Enkripsi *OFB*

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

- 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.

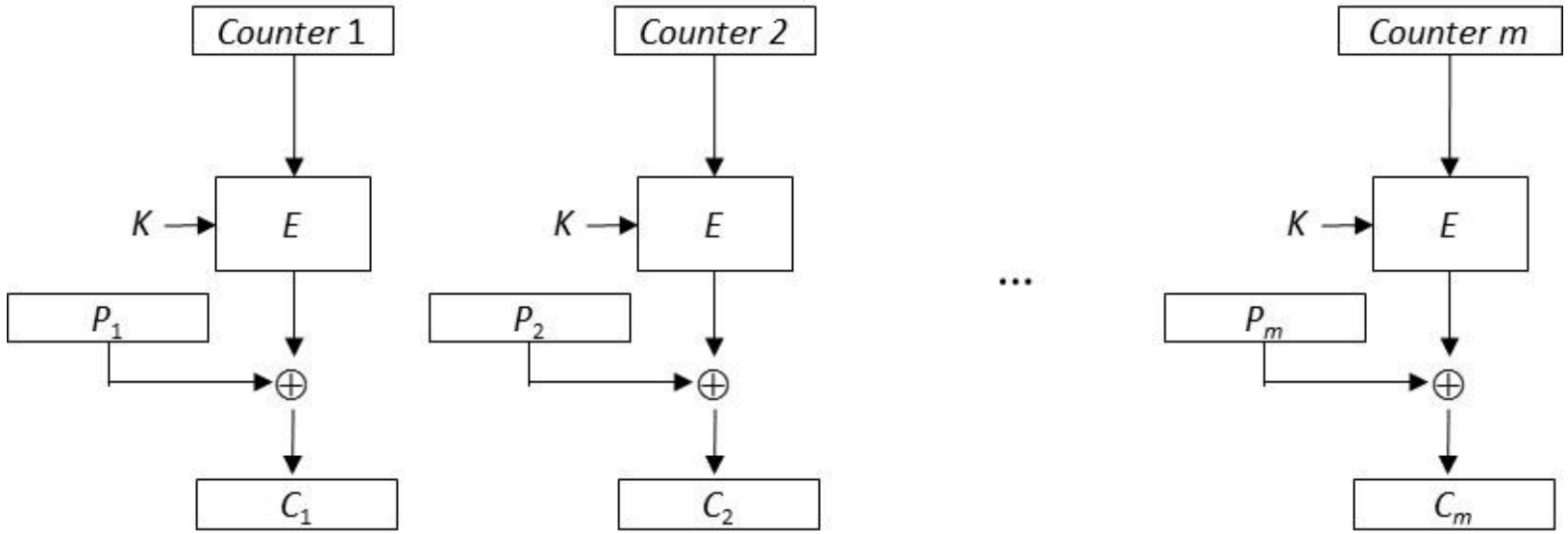
Counter Mode

- Mode *counter* tidak melakukan perantaian (*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 nilainya satu.

(a) Enkripsi



(b) Dekripsi