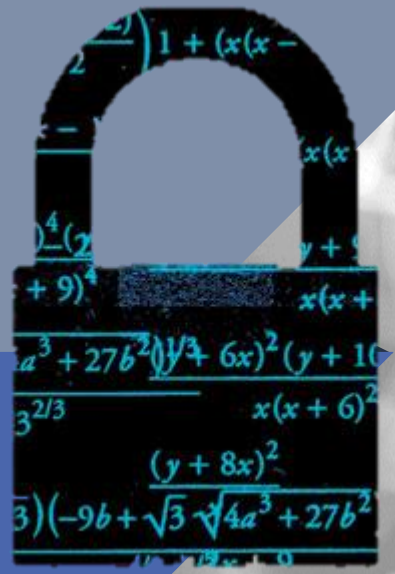
#12

Syahrul Imardi, MT

STMIK Amik Riau

MATAKULIAH

KEAMANAN PERANGKAT LUNAK



Praktikum KRIPTOGRAFI MODERN

P12

STMIK Amik Riau

MATAKULIAH
KEAMANAN PERANGKAT LUNAK

Syahrul Imardi, MT

P12 : Kriptografi Modern - Praktikum

Kategori *cipher* Berbasis Bit

1. *Cipher* Alir (*Stream Cipher*)

   - beroperasi pada bit tunggal

   - enkripsi/dekripsi bit per bit

2. *Cipher* Blok (*Block Cipher*)

   - beroperasi pada blok bit

     (contoh: 64-bit/blok = 8 karakter/blok)

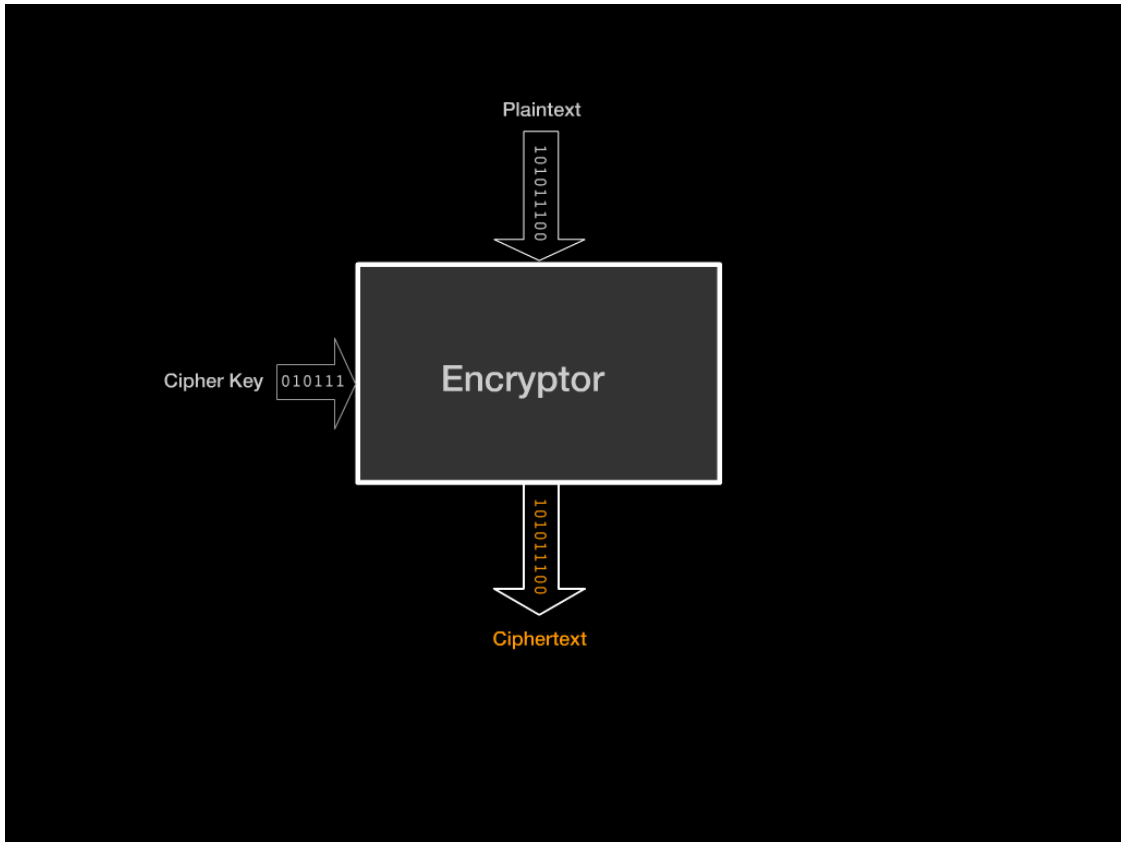
   - enkripsi/dekripsi blok per blok

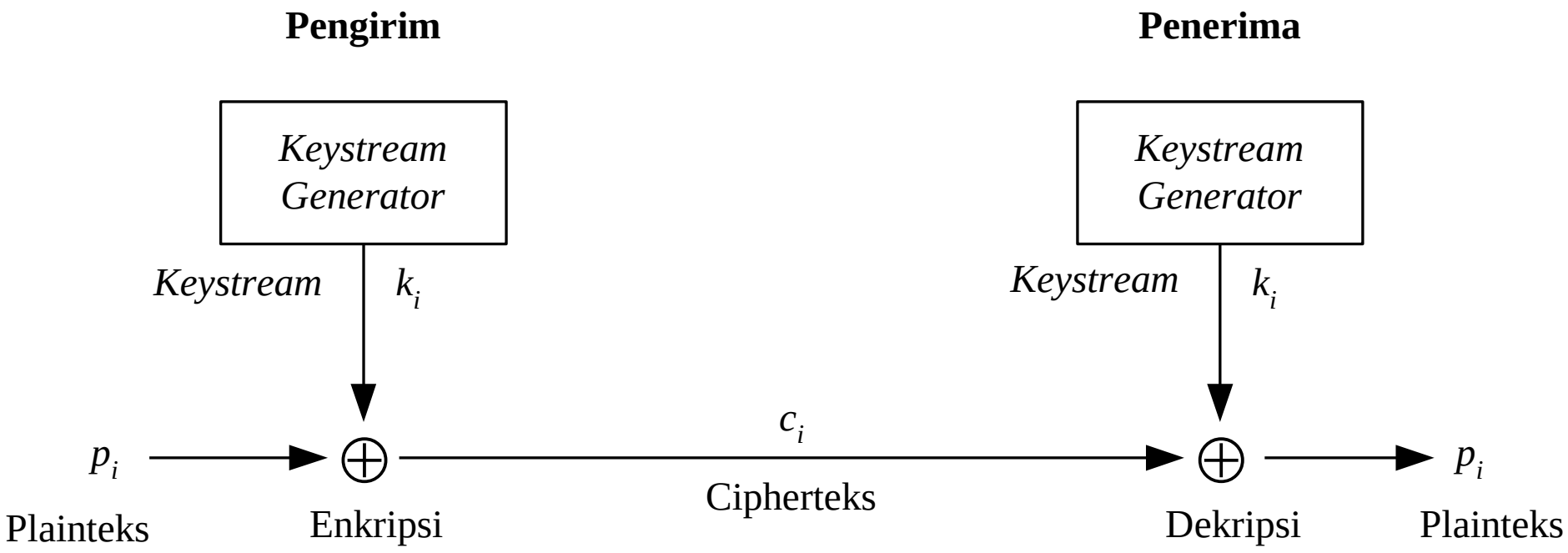
Cipher Alir

- Mengenkripsi plainteks menjadi chiperteks setiap bit per bit dengan bit-bit kunci (dinamakan bit *keystream)* atau *byte* per *byte* (1 *byte* setiap kali transformasi).

- Diperkenalkan oleh Vernam melalui algoritmanya, **Vernam *Cipher***.

- Vernam *cipher* diadopsi dari *one-time pad cipher*, yang dalam hal ini karakter diganti dengan bit (0 atau 1).





Gambar 1 Diagram *cipher* alir [MEY82]

- Bit-bit kunci untuk enkripsi/dekripsi disebut *keystream*

- *Keystream* dibangkitkan oleh *keystream generator.*

- *Keystream* di-XOR-kan dengan bit-bit plainteks, p_1, p_2, …, menghasilkan aliran bit-bit cipherteks:

$$c_i = p_i \oplus k_i$$

- Di sisi penerima dibangkitkan *keystream* yang sama untuk mendekripsi aliran bit-bit cipherteks:

$$p_i = c_i \oplus k_i$$

- Contoh:

Plainteks:	1100101010100110001		
Keystream:	1000110000101001101	⊕	Enkripsi
Cipherteks:	0100011010001111100		
Keystream:	1000110000101001101	⊕	Dekripsi
Plainteks:	1100101010100110001		

- Keamanan *cipher* alir bergantung seluruhnya pada *keystream generator*.
- Tinjau 3 kasus yang dihasilkan oleh *keystream generator*:
  1. *Keystream* seluruhnya 0
  2. *Keystream* berulang secara perodik
  3. *Keystream* benar-benar acak

- **Kasus 1**: Jika pembangkit mengeluarkan *keystream* yang seluruhnya nol,

  Keystream: 00000000000000000000000000000000…

- maka cipherteks = plainteks,

- sebab:

$$c_i = p_i \oplus 0 = p_i$$

dan proses enkripsi menjadi tak-berarti

- **Kasus 2**: Jika pembangkit mengeluarkan *kesytream* yang berulang secara periodik,

Kesytream: 11011011011011011011011011011…

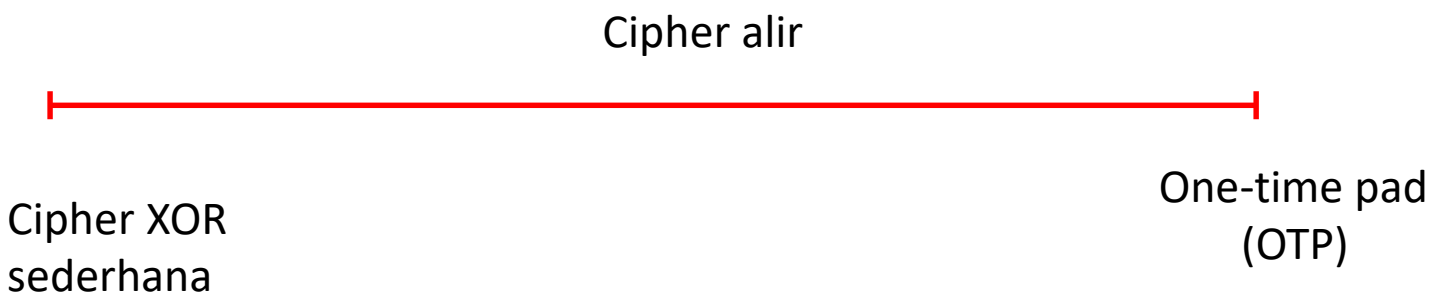
- maka algoritma enkripsinya = cipher XOR sederhana yang memiliki tingkat keamanan yang rendah.

- **Kasus 3**: Jika pembangkit mengeluarkan *keystream* benar-benar acak (*truly random*), maka algoritma enkripsinya = *one-time pad* dengan tingkat keamanan yang sempurna.

Keystream: 01101010010101110011010110010…

- Pada kasus ini, panjang *keystream* = panjang plainteks, dan kita mendapatkan *cipher* alir sebagai *unbreakable cipher*.

- **Kesimpulan**: Tingkat keamanan *cipher* alir terletak antara *cipher* XOR sederhana dengan *one-time pad*.



- Semakin acak keluaran yang dihasilkan oleh pembangkit *keystream*, semakin sulit kriptanalis memecahkan cipherteks.

Keystream Generator

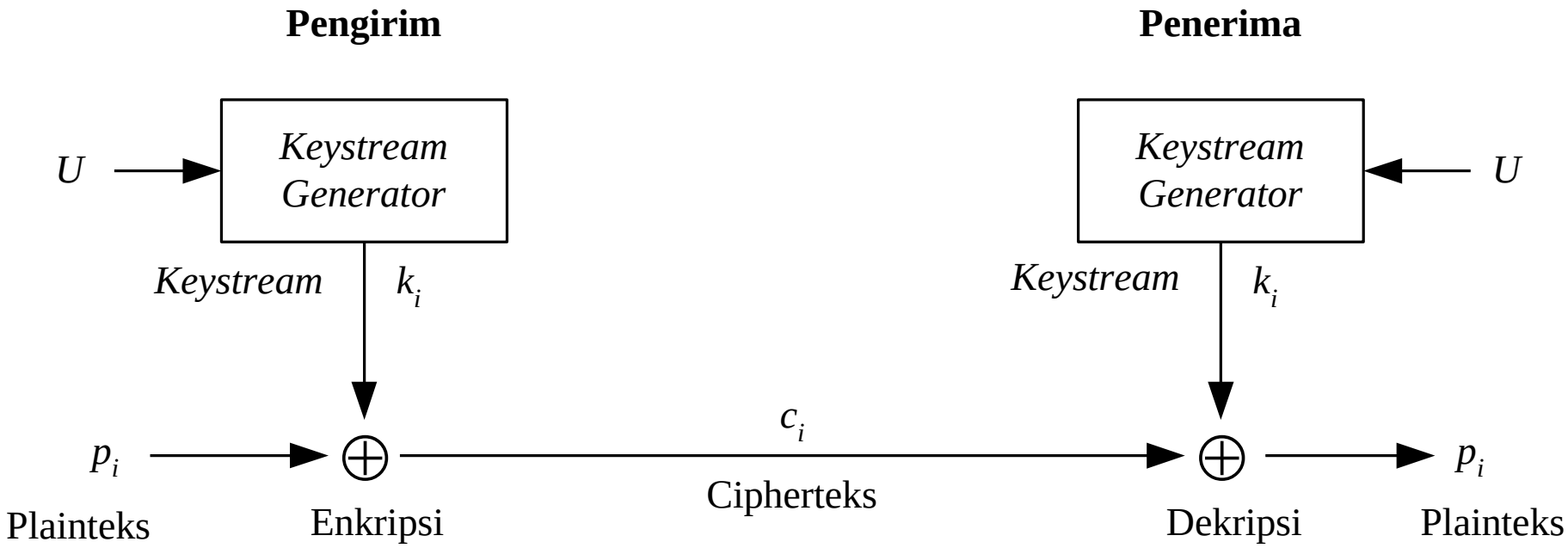
- *Keystream generator* diimplementasikan sebagai prosedur yang sama di sisi pengirim dan penerima pesan.

- *Keystream generator* dapat membangkitkan *keystream* berbasis bit per bit atau dalam bentuk blok-blok bit.

- Jika *keystream* berbentuk blok-blok bit, *cipher* blok dapat digunakan untuk untuk memperoleh *cipher* alir.

- *Keystream generator* menerima masukan sebuah kunci *U*. Luaran dari prosedur merupakan fungsi dari *U* (lihat Gambar 2). Pengirim dan penerima harus memiliki kunci *U* yang sama. Kunci *U* ini harus dijaga kerahasiaanya.

- *Keystream generator* menghasilkan bit-bit kunci yang di-XOR-kan dengan bit plainteks.



Gambar 2 *Cipher* aliran dengan pembangkit bit kunci-alir yang bergantung pada kunci *U* [MEY82].

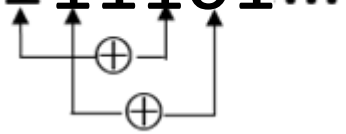
- Contoh: U = 1111

  (U adalah kunci empat-bit yang dipilih sembarang, kecuali 0000)

  Algoritma sederhana memperoleh *keystream*:

  XOR-kan bit ke-1 dengan bit ke-4 dari empat bit sebelumnya:

  1111010110010000 111101...

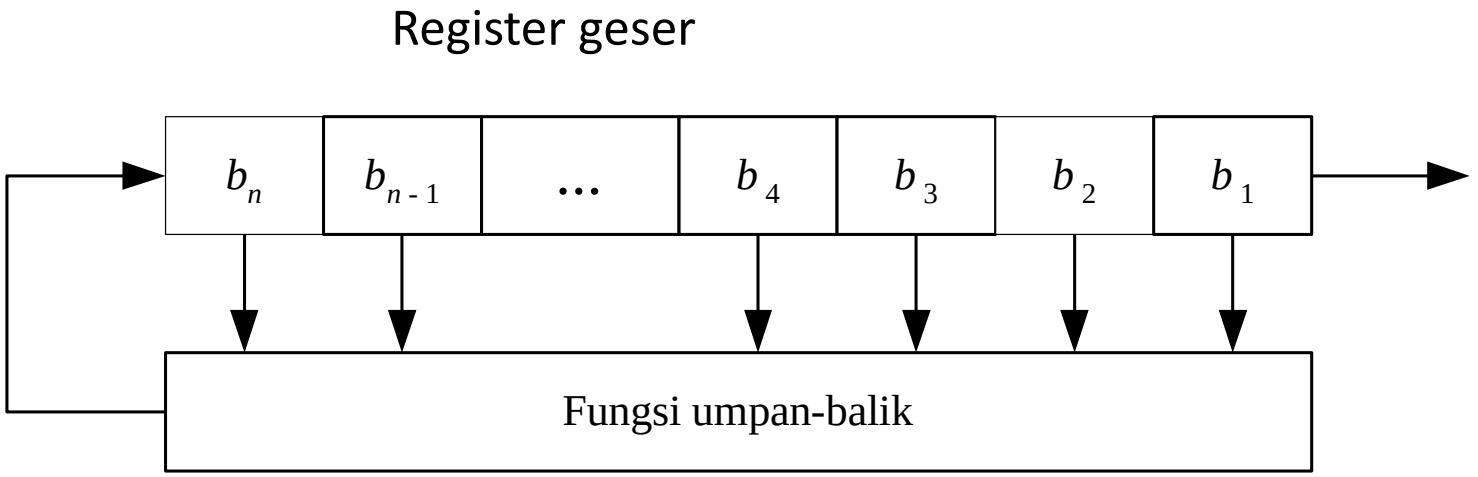
  

  dan akan berulang setiap 15 bit.

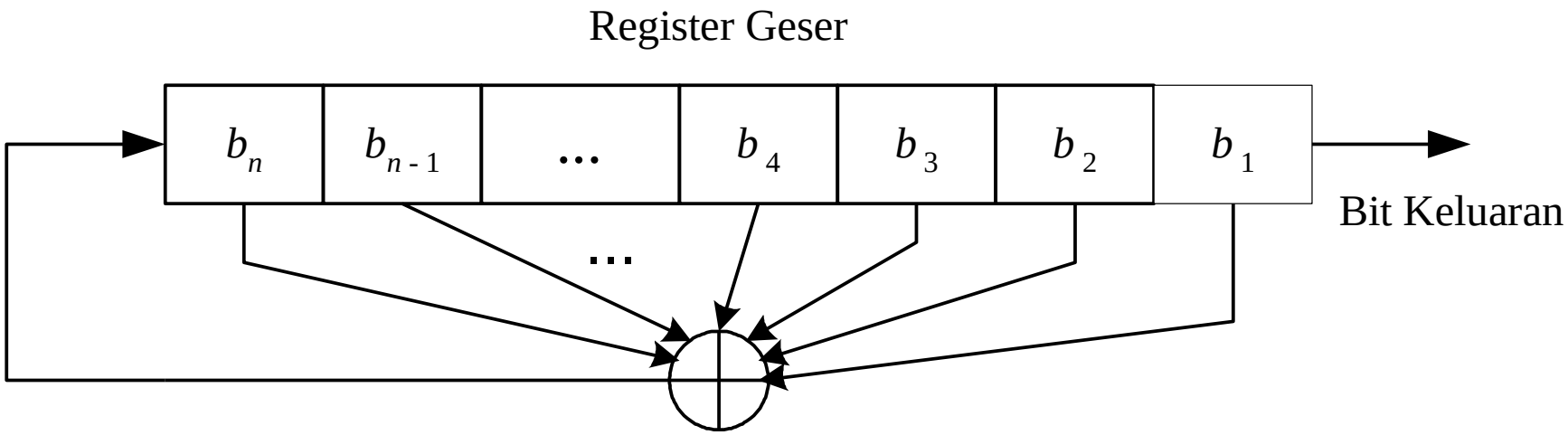
- Secara umum, jika panjang kunci U adalah n bit, maka bit-bit kunci tidak akan berulang sampai $2^n - 1$ bit.

Feedback Shift Register (FSR)

- *FSR* adalah contoh sebuah *keystream generator*.
- *FSR* terdiri dari dua bagian: register geser (n bit) dan fungsi umpan balik

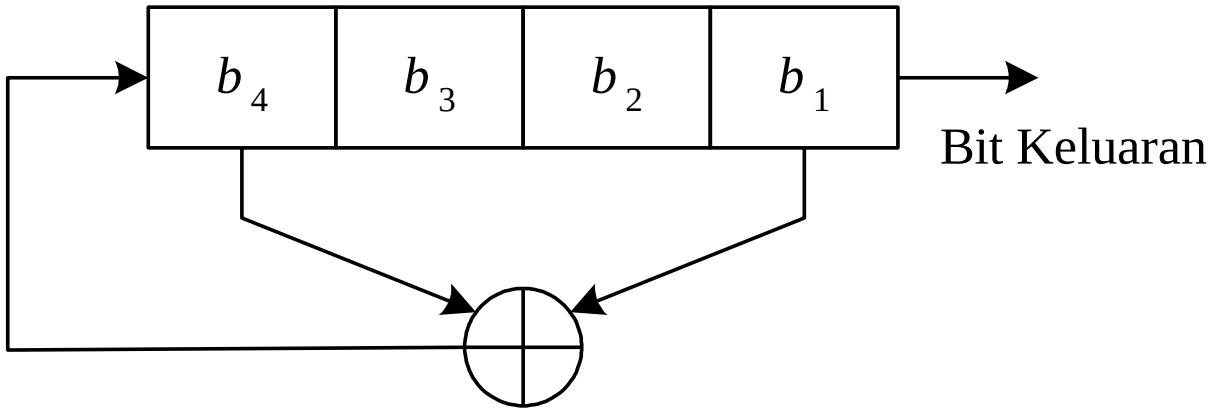


- Contoh FSR adalah LFSR (*Linear Feedback Shift Register*)



- Bit luaran LFSR menjadi *keystream*

- Contoh LFSR 4-bit



- Fungsi umpan balik:

$$b_4 = f(b_1, b_4) = b_1 \oplus b_4$$

- Contoh: jika LFSR 4-bit diinisialisasi dengan 1111

i	Isi Register	Bit Keluaran
0	1 1 1 1	
1	0 1 1 1	1
2	1 0 1 1	1
3	0 1 0 1	1
4	1 0 1 0	1
5	1 1 0 1	0
6	0 1 1 0	1
7	0 0 1 1	0
8	1 0 0 1	1
9	0 1 0 0	1
10	0 0 1 0	0
11	0 0 0 1	0
12	1 0 0 0	1
13	1 1 0 0	0
14	1 1 1 0	0

- Barisan bit acak: 1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 …
- Periode LFSR n-bit: $2^n - 1$

Serangan pada *Cipher* Alir

1. ***Known-plaintext attack***

   Kriptanalis mengetahui potongan *P* dan *C* yang berkoresponden.

   Hasil: *K* untuk potongan *P* tersebut, karena

$$P \oplus C = P \oplus (P \oplus K)$$
$$= (P \oplus P) \oplus K$$
$$= 0 \oplus K$$
$$= K$$

Contoh:

P	01100101		(karakter ‘e’)
K	00110101	$\oplus$	(karakter ‘5’)
C	01010000		(karakter ‘P’)
P	01100101	$\oplus$	(karakter ‘e’)
K	00110101		(karakter ‘5’)

2. *Ciphertext-only attack*

- Kriptanalis hanya memiliki beberapa potong cipherteks saja
- Namun kriptanalis mendeduksi bahwa cipherteks-cipherteks tersebut dihasilkan dari penggunaan potongan *keystream* yang sama yang digunakan lebih dari sekali terhadap beberapa potong plainteks yang berbeda (*keystream reuse attack*).
- Hal ini dapat terjadi karena bit-bit *keystream* yang dihasilkan oleh *keystream generator* memiliki periode yang berulang

- Contoh: Kriptanalis memiliki dua potongan cipherteks berbeda (C_1 dan C_2) yang merupakan hasil enkripsi dua buah plainteks dengan bit-bit *keystream* yang sama.

  XOR-kan kedua cipherteks tersebut:

$$C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K)$$
$$= (P_1 \oplus P_2) \oplus (K \oplus K)$$
$$= (P_1 \oplus P_2) \oplus 0$$
$$= (P_1 \oplus P_2)$$

- Jika P_1 atau P_2 tidak diketahui, dua buah plainteks yang ter-*XOR* satu sama lain ini dapat diterka dengan menggunakan statistik pesan.

- Misalnya dalam teks Bahasa Inggris, dua buah spasi ter-*XOR*, atau satu spasi dengan huruf ‘e’ yang paling sering muncul, dsb.

- Kriptanalis cukup cerdas untuk mendeduksi kedua plainteks tersebut.

3. ***Flip-bit attack***

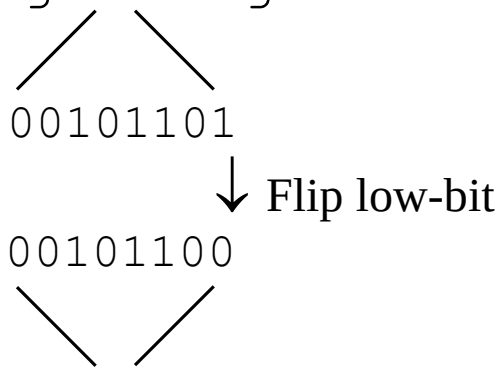
   Tujuan: mengubah bit cipherteks tertentu sehingga hasil dekripsinya berubah.

   Pengubahan dilakukan dengan membalikkan (*flip*) bit tertentu (0 menjadi 1, atau 1 menjadi 0).

Contoh 9.5:

P : `QT-TRNSFR US $00010,00 FRM ACCNT 123-67 TO`

C: `uhtr07hjLmkyR3j7Ukdhj38lkkldkYtr#)oknTkRgh`



C: `uhtr07hjLmkyR3j7Tkdhj38lkkldkYtr#)oknTkRgh`

P : `QT-TRNSFR US $10010,00 FRM ACCNT 123-67 TO`

Pengubahan 1 bit `U` dari cipherteks sehingga menjadi `T`.
Hasil dekripsi: $10,00 menjadi $ 10010,00

- Pengubah pesan tidak perlu mengetahui kunci, ia hanya perlu mengetahui posisi pesan yang diminati saja.

- Serangan semacam ini memanfaatkan karakteristik *cipher* alir yang sudah disebutkan di atas, bahwa kesalahan 1-bit pada cipherteks hanya menghasilkan kesalahan 1-bit pada plainteks hasil dekripsi.

Aplikasi *Cipher* Alir

- *Cipher* alir cocok untuk mengenkripsi aliran data yang terus menerus melalui saluran komunikasi, misalnya:
  1. Mengenkripsi data pada saluran yang menghubungkan antara dua buah komputer.
  2. Mengenkripsi suara pada jaringan telepon *mobile* GSM.

- Alasan: jika bit cipherteks yang diterima mengandung kesalahan, maka hal ini hanya menghasilkan satu bit kesalahan pada waktu dekripsi, karena tiap bit plainteks ditentukan hanya oleh satu bit cipherteks.

Referensi utama :

>> Michael Felderer , Riccardo Scandariato (editor) - Exploring Security in Software Architecture and Design, 2018.

>> Nancy R. Mead, Carol Woody - Cyber Security Engineering_ A Practical Approach for Systems and Software Assurance-Addison-Wesley Professional (2016)

>> James Helfrich - Security for Software Engineers-CRC Press (2019)

>> Pete Loshin - Simple Steps to Data Encryption_ A Practical Guide to Secure Computing-Syngress (2013)

>> Tevfik Bultan,Fang Yu,Muath Alkhalaf,Abdulbaki Aydin (auth.) - String Analysis for Software Verification and Security (2017)


Any
Questions?