II4031 Kriptografi dan Koding

Stream Cipher

Oleh: Rinaldi Munir

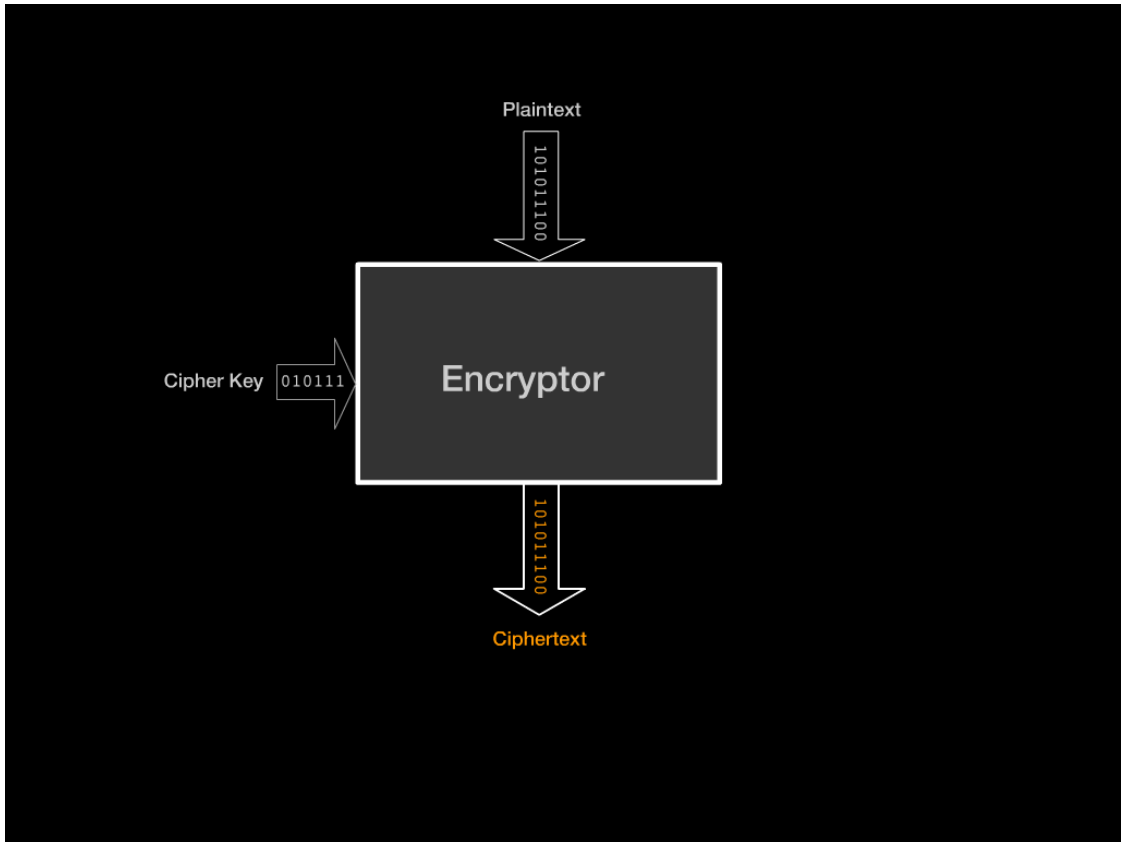
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
ITB

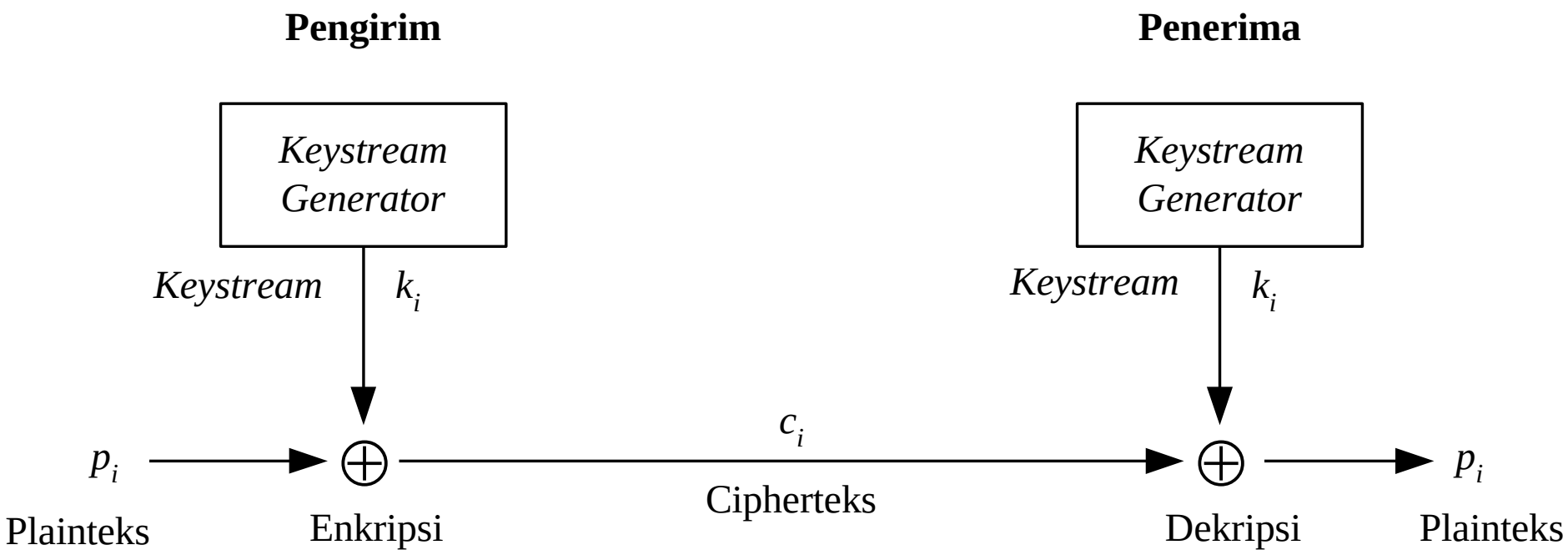
Cipher Alir

- 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…

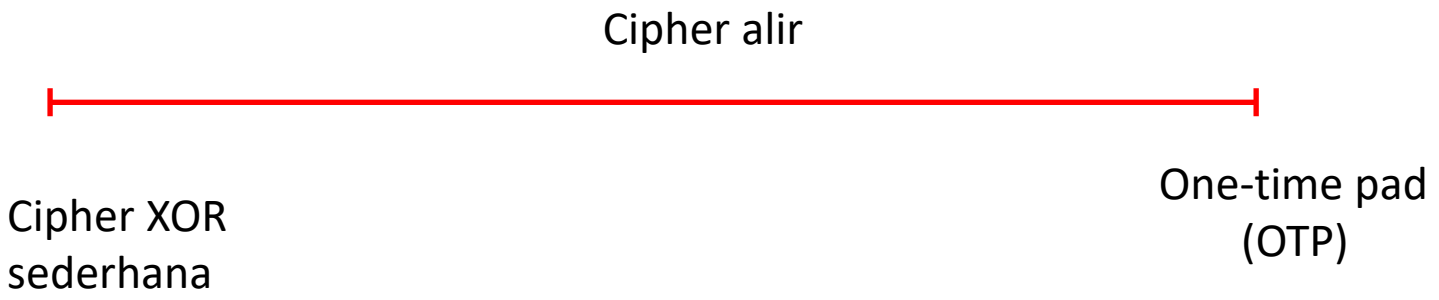
- maka algoritma enkripsinya = cipher XOR sederhana yang memiliki tingkat keamanan yang rendah.

- **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

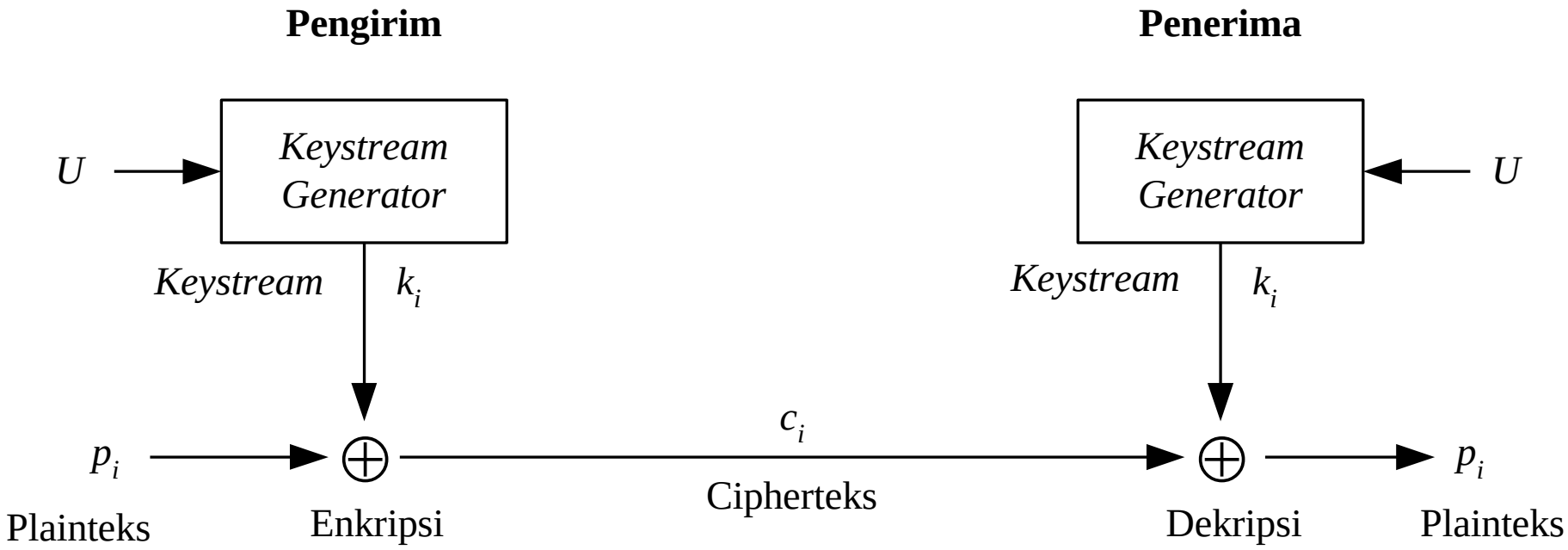
- *Keystream generator* diimplementasikan sebagai prosedur yang sama di sisi pengirim dan penerima pesan.

- *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].

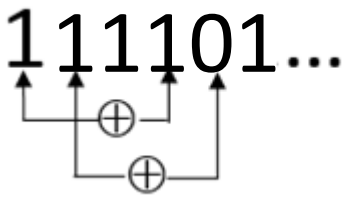
- Contoh: U = 1111

  (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:

  11110101100100**0**

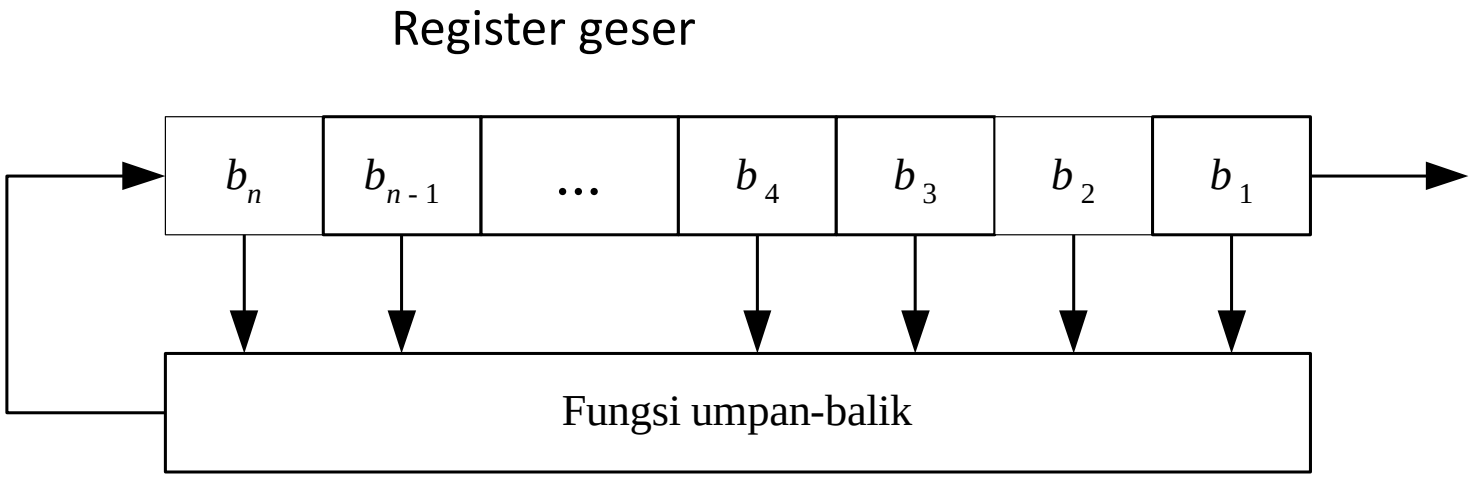
  

  dan akan berulang setiap 15 bit.

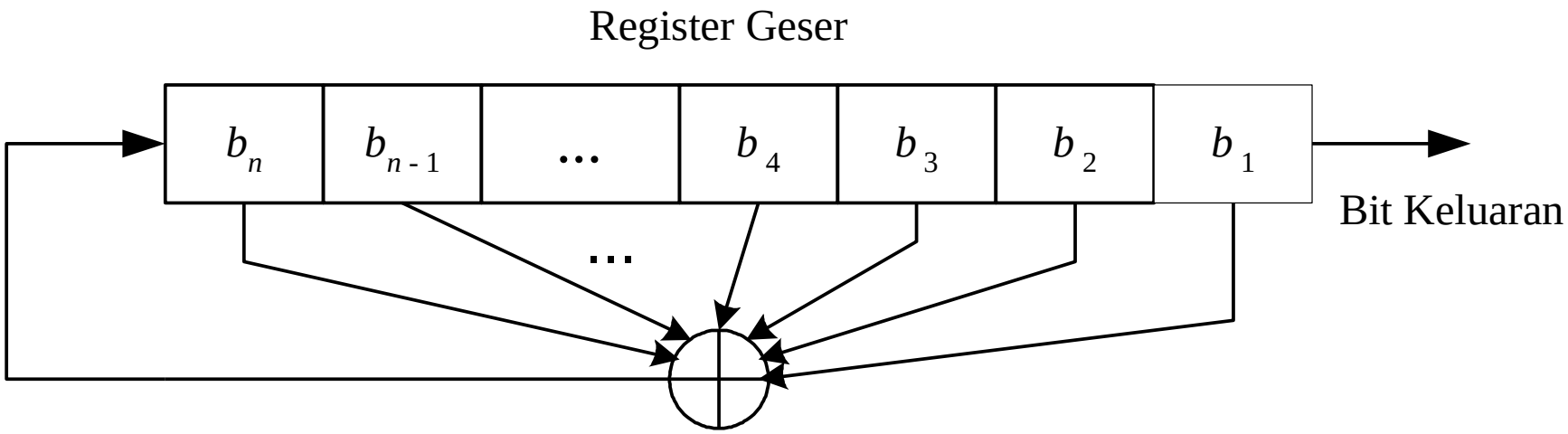
- Secara umum, jika panjang kunci U adalah n bit, maka bit-bit kunci tidak akan berulang sampai $2^n - 1$ bit.

Feedback Shift Register (FSR)

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

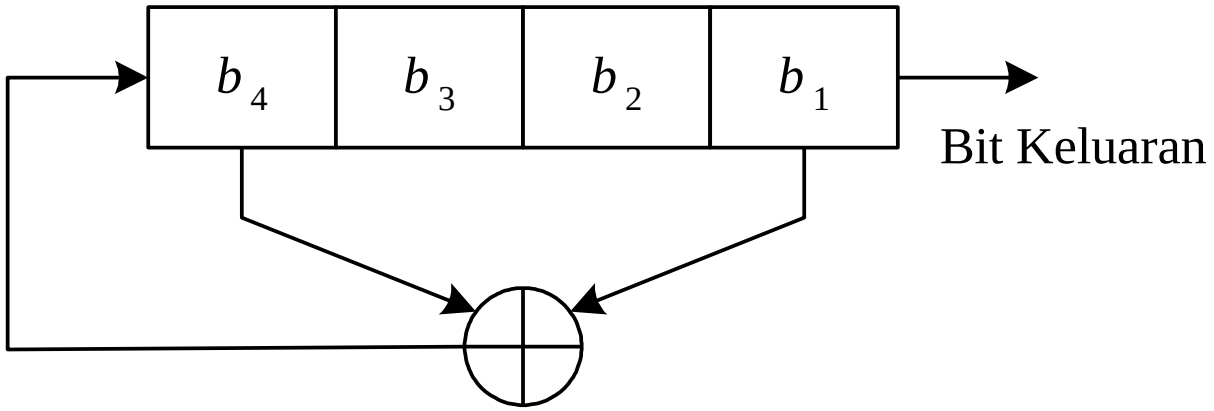


- Contoh FSR adalah LFSR (*Linear Feedback Shift Register*)



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

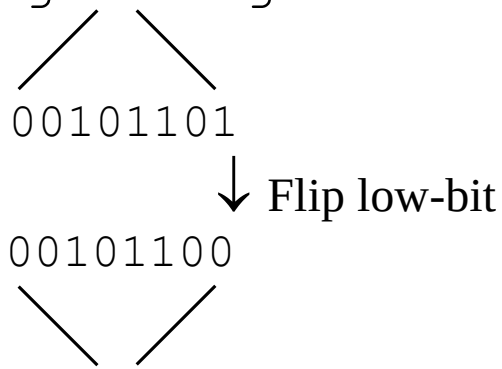
   Tujuan: mengubah bit cipherteks tertentu sehingga hasil dekripsinya berubah.

   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.

RC4

RC4

- Termasuk ke dalam *cipher* alir (*stream cipher*)
- Dibuat oleh Ron Rivest (1987) dari Laboratorium *RSA*
- *RC* adalah singkatan dari *Ron's Code*). Versi lain mengatakan *Rivest Cipher* .
- Digunakan sistem keamanan seperti:

  - protokol *SSL* (*Secure Socket Layer*).

  - WEP (Wired Equivalent Privacy)

  - WPA (Wi-fi Protect Access) untuk nirkabel

- *RC4* membangkitkan kunci-alir (*keystream*) dalam ukuran byte setiap kalinya, yang kemudian di-XOR-kan dengan karakter plainteks

- Jadi, *RC4* memproses data dalam ukuran *byte*, bukan dalam bit.

- Untuk membangkitkan kunci-alir, *cipher* menggunakan status internal yang terdiri dari:
  - Permutasi angka 0 sampai 255 di dalam larik S_0, S_1, …, S_{255}. Permutasi merupakan fungsi dari kunci *K* dengan panjang variabel.
  - Dua buah pencacah indeks, *i* dan *j*

- RC4 terdiri dari dua sub-proses:
  1. *Key-Scheduling Algorithm* (*KSA*)
  2. *Pseudo-random generation algorithm* (*PRGA*).

Key-Scheduling Algorithm (*KSA*)

1. Inisialisasi larik S: $S_0 = 0$, $S_1 = 1$, …, $S_{255} = 255$

```
for i ← 0 to 255 do
  S[i] ← i
end
```

0	1	2	3	4	5	6	7	…	…	…	252	253	254	255
0	1	2	3	4	5	6	7	…	…	…	252	253	254	255

3. Lakukan pengacakan (permutasi) nilai-nilai di dalam larik *S* berdasarkan kunci K:

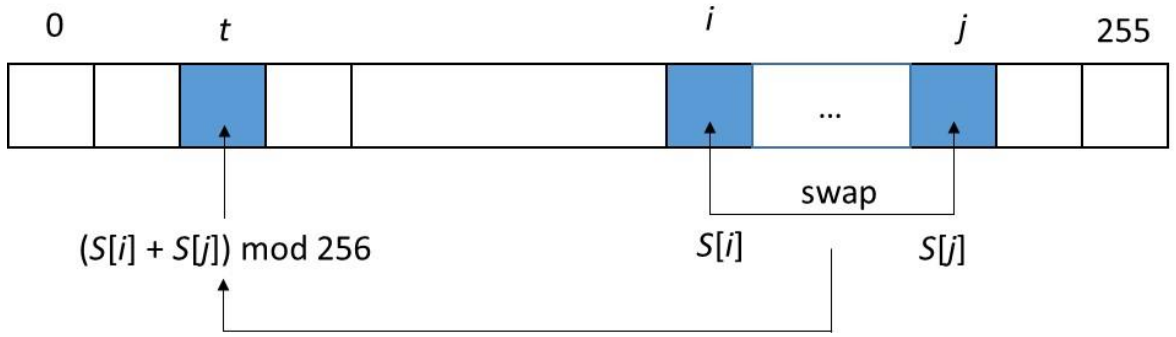
```
j ← 0
for i ← 0 to 255 do
  j ← (j + S[i] + K[i  mod Length(K)]) mod 256
  swap(S[i], S[j]) {* Pertukarkan S[i] & S[j] *}
end
```

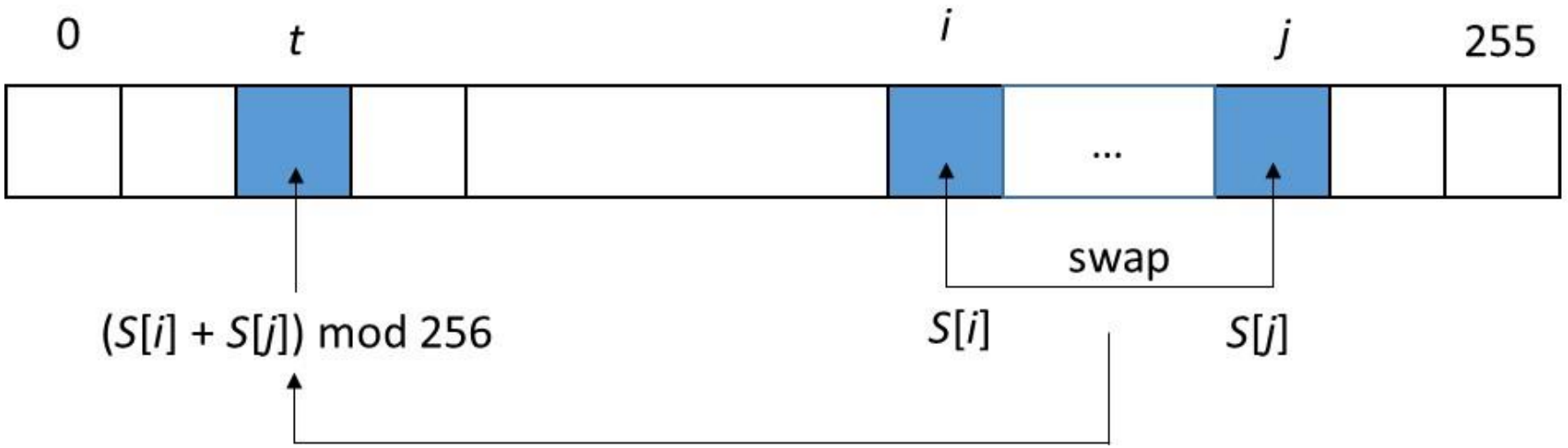
- Pemutasi ini menyebabkan elemen-elemen di dalam larik *S* teracak.

Pseudo-random generation algorithm (*PRGA*)

- *PRGA* membangkitkan kunci-alir (*keystream*) dengan cara mengambil nilai *S*[*i*] dan *S*[*j*], mempertukarkannya, lalu menjumlahkan keduanya dalam modulus 256.
- Kunci alir tersebut kemudian di-XOR-kan dengan sebuah karakter plainteks.

```
i ← 0
j ← 0
for idx ← 0 to Length(P) – 1 do
    i ← (i + 1) mod 256
    j ← (j + S[i]) mod 256
    swap(S[i], S[j])    {* Pertukarkan nilai S[i] dan S[j] *}
    t ← (S[i] + S[j]) mod 256
    u ← S[t]            (* keystream *)
    c ← u ⊕ P[idx]      { enkripsi }
end
```




0
t
i
j
255
...
swap
(S[i] + S[j]) mod 256
S[i]
S[j]

Keamanan RC4

- RC4 adalah *cipher* alir, maka ia tidak kuat terhadap serangan seperti *flip-bit attack* maupun serangan-serangan *stream attack* lainnya.

- Saat ini keamanan RC4 sudah berhasil dipecahkan dalam hitungan jam atau hari.

- Pada bulan Februari 2015, RC4 dilarang penggunaannya di dalam *Transport Layer Security* (TLS) seperti disebutkan di dalam RFC 7465 (https://tools.ietf.org/html/rfc7465).

Kode Program RC4 (dalam Bahasa C++)

- Enkripsi

```
// Enkripsi sembarang berkas dengan algoritma RC4.
#include <iostream>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
using namespace std;

main(int argc, char *argv[])
{
 FILE *Fin, *Fout;
 char p, c, u;
 string K;
 int S[256];
 int i, j, t, count;
```

```
Fin = fopen(argv[1], "rb");
 if (Fin == NULL) {
   cout << "Berkas " << argv[1] <<" tidak ada" << endl;
   exit(0);
 }

 Fout = fopen(argv[2], "wb");

 cout << "Kata kunci : "; cin >> K;
 cout <<"Enkripsi " << argv[1] << " menjadi " << argv[2] << "...";

 for (i = 0; i<256; i++) {
  S[i] = i;
 }
```

```
j = 0;
for (i=0; i<256; i++) {
   j = (j + S[i] + K[i % K.length()]) % 256;
   swap(S[i], S[j]); // Pertukarkan nilai S[i] dan S[j]
}

i = 0;   j = 0;
count = 0;
while (!feof(Fin)) {
   p = fgetc(Fin);
   count = count + 1;
   i = (i + 1) % 256;
   j = (j + S[i]) % 256;
   swap(S[i], S[j]);    // Pertukarkan nilai S[i] dan S[j]
   t = (S[i] + S[j]) % 256;
   u = S[t];   // keystream
   c = u ^ p;
   fputc(c, Fout);
}
cout << "Count = "<< count;
fclose(Fin);  fclose(Fout);
}
```

- Dekripsi

```
//Dekripsi sembarang berkas dengan algoritma RC4
#include <iostream>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
using namespace std;

main(int argc, char *argv[])
{
 FILE *Fin, *Fout;
 char p, c, u;
 string K;
 int S[256];
 int i, j, t, count;
```

```
Fin = fopen(argv[1], "rb");
 if (Fin == NULL) {
   cout << "Berkas " << argv[1] <<" tidak ada" << endl;
   exit(0);
 }

 Fout = fopen(argv[2], "wb");

 cout << "Kata kunci : "; cin >> K;
 cout <<"Dekripsi " << argv[1] << " menjadi " << argv[2] << "...";

 for (i = 0; i<256; i++) {
  S[i] = i;
 }
```

```
j = 0;
 for (i=0; i<256; i++) {
    j = (j + S[i] + K[i % K.length()]) % 256;
    swap(S[i], S[j]); // Pertukarkan nilai S[i] dan S[j]
 }

 i = 0;   j = 0;
 count = 0;
 while (!feof(Fin)) {
    c = fgetc(Fin);
    count = count + 1;
    i = (i + 1) % 256;
    j = (j + S[i]) % 256;
    swap(S[i], S[j]);    // Pertukarkan nilai S[i] dan S[j]
    t = (S[i] + S[j]) % 256;
    u = S[t];   // keystream
    p = u ^ c;
    fputc(p, Fout);
 }
 cout << "Count = " << count;
 fclose(Fin);  fclose(Fout);
}
```

A5

- *A5*: *cipher* aliran yang digunakan untuk mengenkripsi transmisi sinyal percakapan dari standard telepon seluler *GSM* (*Group Special Mobile*).

- Sinyal *GSM* dikirim sebagai barisan *frame*. Satu *frame* panjangnya 228 bit dan dikirim setiap 4,6 milidetik.

- *A5* digunakan untuk untuk menghasilkan kunci-alir (*keystream*) 228-bit yang kemudian di-*XOR*-kan dengan *frame*. Kunci eksternal (*session key*) panjangnya 64 bit.

- GSM merupakan standard telepon seluler Eropa. A5 dikembangkan oleh Perancis

- Tidak semua operator GSM mengimplementasikan enkripsi, bergantung regulasi (seperti di Indonesia)

- A5 ada dua versi:
    1. A5/1 : versi kuat A5, digunakan di Eropa
    2. A5/2 (Kasumi) : versi ekspor, lebih lemah

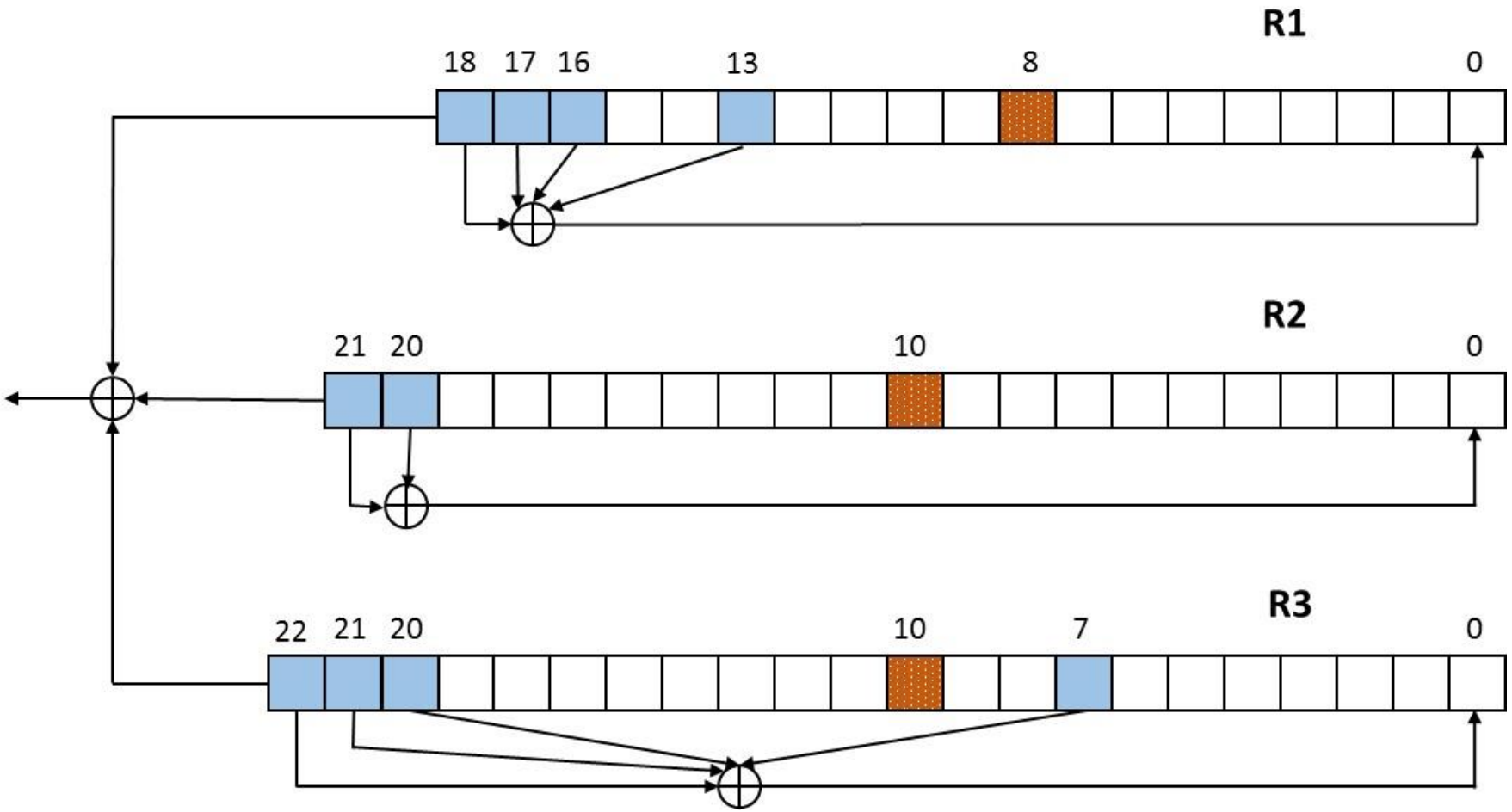
- Algoritma A5/1 pada awalnya rahasia, tetapi pada tahun 1994 melalui *reverse engineering*, algoritmanya terbongkar.

- *A5* terdiri dari 3 buah *LFSR* , masing-masing panjangnya 19, 22, dan 23 bit (total = 19 + 22 + 23 = 64).

- Bit-bit di dalam register diindeks dimana bit paling tidak penting (*LSB*) diindeks dengan 0 (elemen paling kanan).

- Luaran (*output*) dari *A5* adalah hasil *XOR* dari ketiga buah *LFSR* ini.

- *A5* menggunakan tiga buah kendali detak (*clock*) yang variabel:
  - Bit ke-8 pada register 1. Bit-bit detak pada bit 13, 16, 17, dan 18
  - Bit ke-10 pada register 2. Bit-bit detaknya adalah pada bit 20 dan 21
  - Bit ke-10 pada register 3. Bit-bit detaknya adalah pada bit 7, 20, 21, dan 22


R1
18
17
16
13
8
0
R2
21
20
10
0
R3
22
21
20
10
7
0

- Setiap register didetak dalam fase *stop/go* dengan menggunakan kaidah mayoritas:

  "Pada setiap putaran (*i*=1..64), bit-bit tengah setiap register diperiksa dan bit mayoritasnya (50% + 1) ditentukan. Jika bit tengah sebuah register sama dengan bit mayoritas, maka register tersebut didetak"

.

- Biasanya pada setiap putaran dua atau tiga buah register didetak. Peluang sebuah register didetak pada setiap putaran adalah ¾. Ketika sebuah register didetak, semua bit detaknya di-XOR-kan dan hasilnya diletakkan pada posisi LSB (posisi ke-0) dengan mekanisme pergeseran bit-bit ke kiri.

- Bit paling kiri (MSB) terlempar keluar. Bit yang terlempar dari setiap register di-XOR-kan bersama, bit inilah yang menjadi luaran dari ketiga buah register tadi.

Proses pembangkitan bit-bit acak di dalam A5/1 berdasarkan kunci sesi *K* adalah sebagai berikut:

1. Ketiga register pada awalnya diinisialisasi seluruh bitnya dengan 0. Selanjutnya dilakukan 64 putaran pertama, 64 bit kunci sesi *K* dicampur dengan bit register berdasarkan skema berikut:

   pada putaran $0 \leq i < 64$, bit $K[i]$ ditambahkan ke bit *LSB* dari setiap register *R* dengan menggunakan operasi *XOR*:

   $$R[0] = R[0] \oplus K[i]$$

2. Selanjutnya, ketiga register didetak selama 22 putaran tambahan. Selama 22 putaran tersebut, 22-bit dari nomor *frame* (F_n) dicampur dengan bit register berdasarkan skema berikut:

   pada putaran $0 \leq i < 22$, bit $F_n[i]$ ditambahkan ke bit *LSB* dari setiap register *R* dengan menggunakan operasi *XOR*:

   $$R[0] = R[0] \oplus F_n[i]$$

Isi register pada akhir putaran menyatakan kondisi awal untuk pembangkitan *frame* sepanjang 228-bit.

3. Ketiga register didetak selama 100 putaran dalam fase *stop/go* dengan menggunakan kaidah mayoritas, namun bit-bit luarannya dibuang (tidak dipakai).

4. Selanjutnya, ketiga register didetak selama 228 putaran dalam fase *stop/go* menggunakan kaidah mayoritas untuk menghasilkan bit-bit kunci-alir sepanjang 228 bit.

   Pada setiap putaran dihasilkan 1 bit yang merupakan hasil peng-XOR-an bit-bit MSB yang terlempar.

   Kunci-alir 228-bit inilah yang kemudian digunakan untuk mengenkripsi *frame* pesan sepanjang 228 bit.

- Algoritma A5/1 telah digunakan untuk mengenkripsi semua percakapan dan komunikasi data melalui telepon seluler GSM di Eropa.

- Program A5/1 ditanam di dalam *chip* pada kartu *Simcard*.

- Di negara-negara di mana operator telepon seluler dilarang melakukan enkripsi percakapan, seperti di Indonesia, maka program algoritma A5 tidak diaktifkan (*disabled*), sehingga semua sinyal terkirim dalam bentuk plainteks.

Keamanan A5

- Keamanan A5/1 terletak pada pembangkitan bit-bit acak yang dihasilkan oleh tiga buah LFSR di atas.

- Tujuan kriptanalisis A5/1 adalah untuk menemukan kunci sesi yang digunakan dalam pembangkitan bit-bit acak.

- Dalam serangan tersebut diasumsikan penyerang mengetahui luaran A5/1 pada periode awal komunikasi, untuk selanjutnya menemukan kunci sesi yang digunakan untuk mendekripsi sisa komunikasi selanjutnya.

- Serangan yang dilakukan terhadap A5/1 adalah *known-plaintext attack*.

- Serangan semacam ini pertama kali dilakukan oleh Ross Anderson pada tahun 1994.

- Ross Anderson mencoba menerka 42 bit isi register R1 dan R2, dan menurunkan 23 bit R3 dari 42 bit tersebut. Pekerjaan menemukan kunci tersebut dilakukan pada komputer PC dan membutuhkan waktu komputasi lebih dari sebulan.

- Pada tahun-tahun selanjutnya, para kriptanalis berhasil menemukan kunci hanya dalam waktu beberapa menit saja.

- Secara umum, A5/1 sudah berhasil dikriptanalisis.