IF4020 Kriptografi

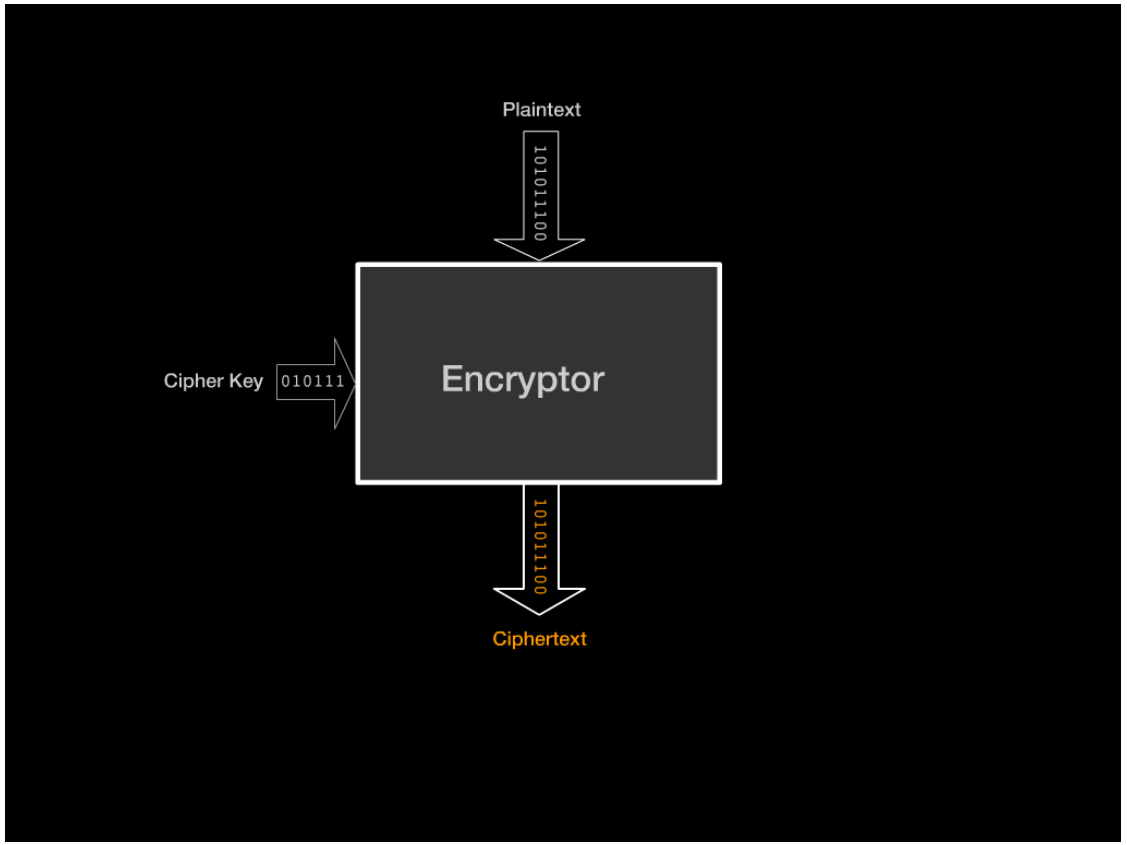
Stream Cipher

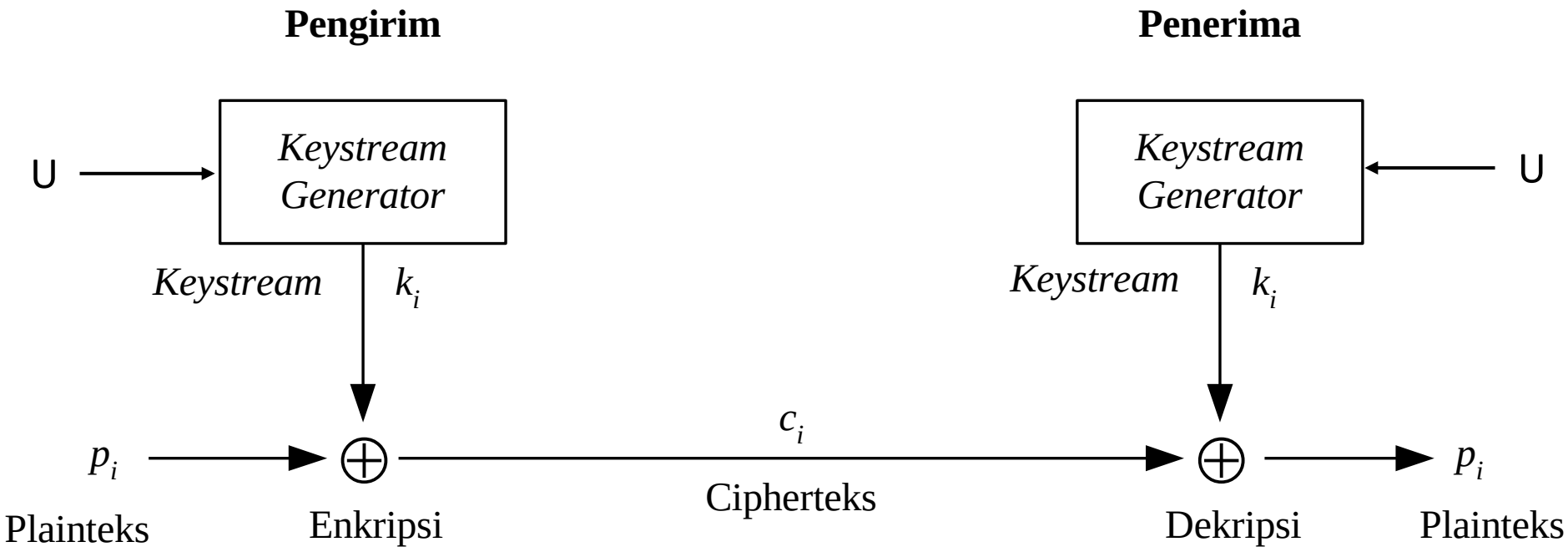
Oleh: Rinaldi Munir

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024

Cipher Alir (*Stream Cipher*)

- Mengenkripsi plainteks menjadi chiperteks setiap bit per bit dengan bit-bit kunci atau *byte* per *byte* (1 *byte* setiap kali enkripsi).
- Diperkenalkan oleh Vernam melalui algoritmanya, **Vernam *Cipher***.
- Vernam *cipher* diadopsi dari *one-time pad cipher*, yang dalam hal ini huruf diganti dengan bit (0 atau 1).





Gambar 1 Diagram *cipher* alir

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

- *Keystream* dibangkitkan oleh *keystream generator* berdasarkan umpan (*seed*) U.

- Enkripsi dan dekripsi dengan *cipher* alir komputasinya sederhana dan cepat

- *Keystream* c_1, c_2, … di-XOR-kan dengan aliran bit-bit plainteks, p_1, p_2, …, menghasilkan aliran bit-bit cipherteks c_1, c_2, …:

$$c_i = p_i \oplus k_i$$

- 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 (*trully random*)

- **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…

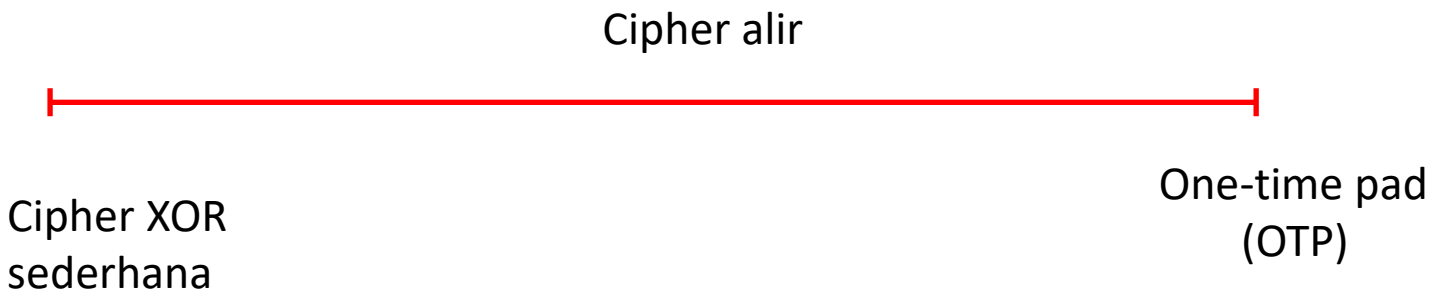
- maka algoritma enkripsinya = cipher XOR sederhana yang memiliki tingkat keamanan yang rendah.

- **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

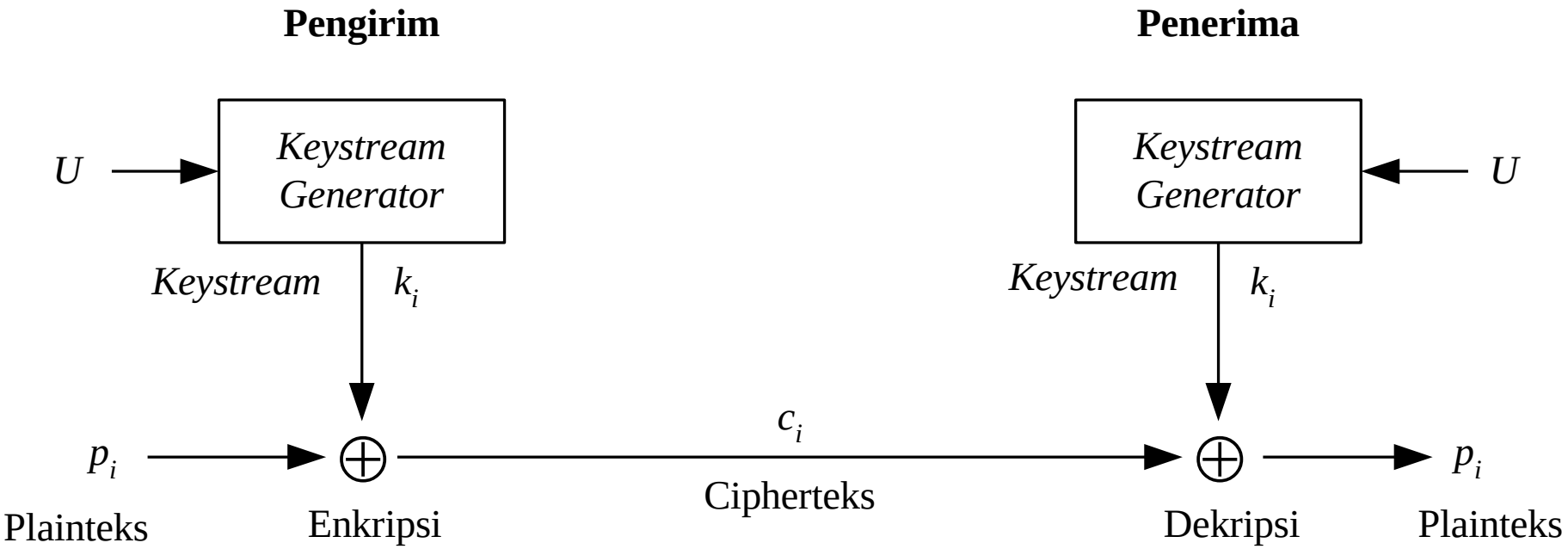
- *Keystream generator* diimplementasikan sebagai prosedur yang sama baik di sisi pengirim maupun di sisi penerima pesan.

- *Keystream generator* dapat membangkitkan *keystream* bit per bit *byte* per *byte*, atau blok-blok bit.

- Bit-bit *keystream* adalah bit-bit acak, namun bukan *true random*, tetapi *pseudorandom*, karena bit-bit *keystream* dapat dibangkitkan ulang baik di sisi pengirim maupun di sisi penerima pesan asalkan umpan (*seed*) U yang digunakan sama.

- *Keystream generator* menerima masukan sebuah umpan (*seed*) *U*. Luaran dari prosedur merupakan fungsi dari *U* (lihat Gambar 2). Pengirim dan penerima harus memiliki umpan *U* yang sama. Umpan *U* ini harus dijaga kerahasiaanya. Umpan *U* dapat dianggap sebagai kunci eksternal.

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



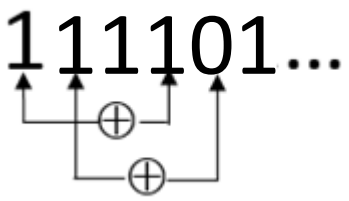
Gambar 2 *Cipher* aliran dengan pembangkit bit kunci-alir yang bergantung pada kunci *U* [MEY82].

- Contoh: Misalkan *U* = 1111

  Algoritma sederhana memperoleh *keystream* adalah sebagai berikut:

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

  1111010110010000

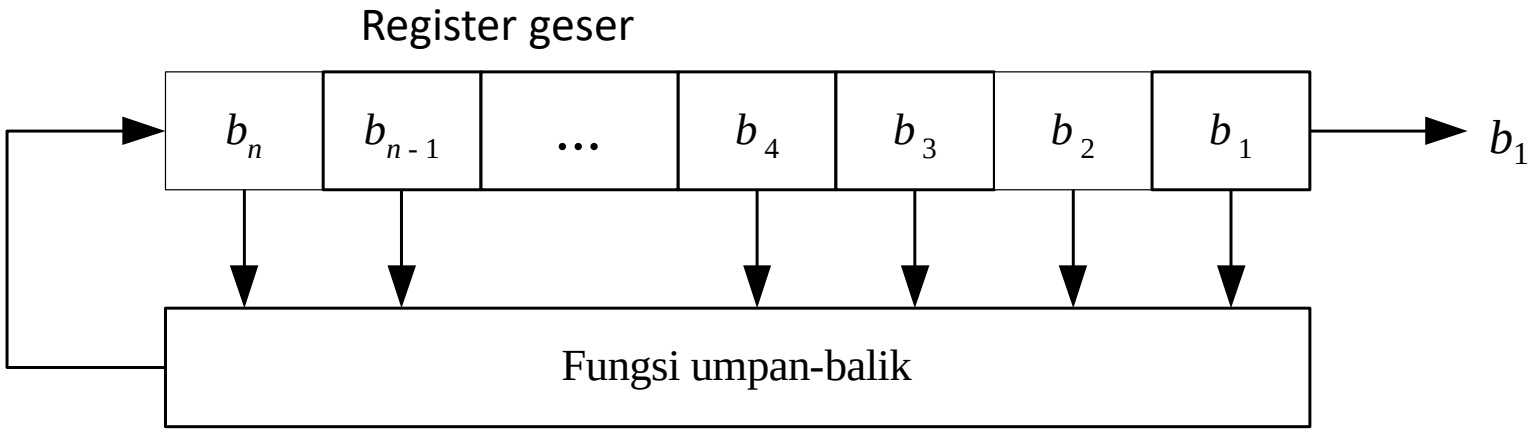
  

  dan akan berulang setiap 15 bit.

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

Feedback Shift Register (FSR)

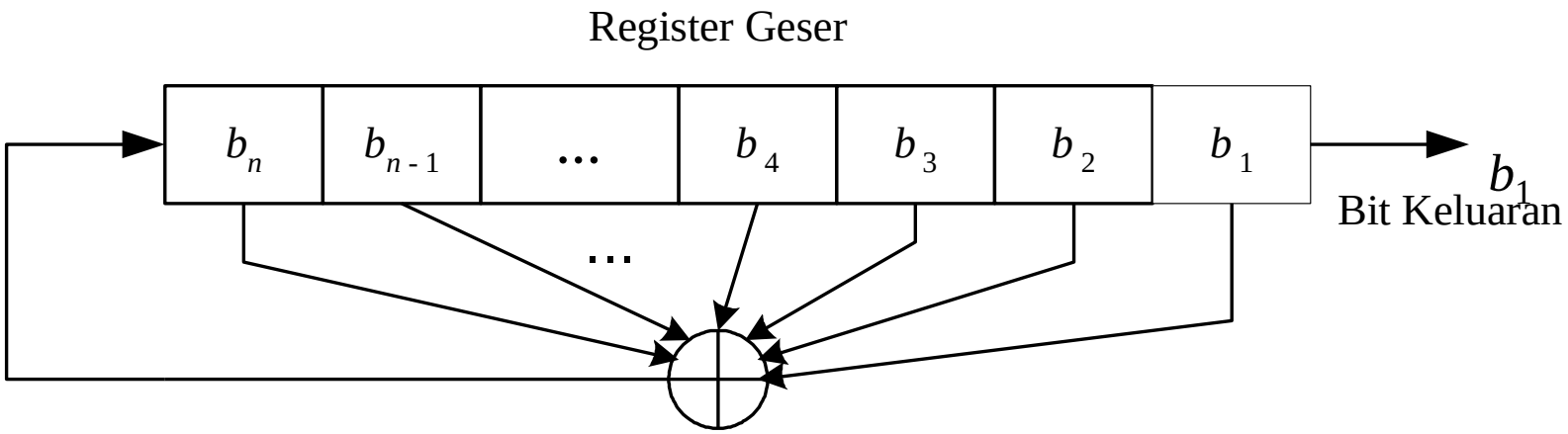
- *FSR* adalah contoh sebuah *keystream generator*.
- Umumnya diimplementasikan sebagai *hardware*.
- *FSR* terdiri dari dua bagian: register geser (n bit) dan fungsi umpan balik



Fungsi umpan baik : $f(b_n, b_{n-1}, b_2, b_1)$

$b_n = f(b_n, b_{n-1}, b_2, b_1)$

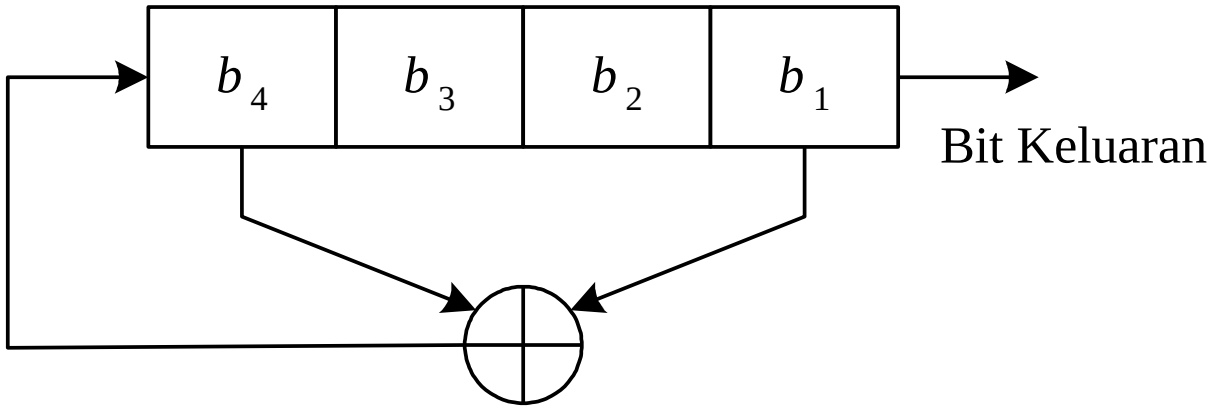
- Contoh FSR adalah LFSR (*Linear Feedback Shift Register*)



$$b_n = b_n \oplus b_{n-1} \oplus \ldots \oplus b_2 \oplus b_1$$

- Bit-bit di dalam register digeser satu bit ke kanan setiap kali pembangkitan bit luaran (= bit kunci alir atau *keystream*)
- Bit b_n digeser ke tempat b_{n-1}, b_{n-1} digeser ke b_{n-2}, dst, b_2 digeser ke b_1, b_1 terlempar ke luar
- Bit yang terlempar keluar (b_1) menjadi bit luaran LFSR
- Selanjutnya dihitung b_n yang baru, yaitu $b_n = b_n \oplus b_{n-1} \oplus \ldots \oplus b_2 \oplus b_1$

- 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 U = 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*

- terjadi jika kunci-alir yang sama digunakan dua kali terhadap potongan plainteks yang berbeda (*keystream reuse attack*)
- Misalkan kriptanalis memiliki dua potongan cipherteks berbeda (C_1 dan C_2) yang dienkripsi dengan *keystream* K yang sama
- XOR-kan kedua potongan 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 dan P_2 tidak diketahui, maka keduanya dapat diterka secara statistik. Misalnya P_1 dan P_2 adalah dua spasi yang ter-XOR satu sama lain atau spasi dengang huruf E yang sering muncul, dsb.

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

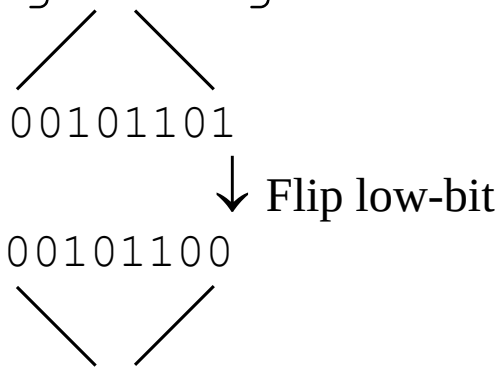
   Tujuan: mengubah bit cipherteks tertentu sehingga hasil dekripsinya berubah.

   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: `uhtr07hjLmkyR3j7`**`U`**`kdhj38lkkldkYtr#)oknTkRgh`



C: `uhtr07hjLmkyR3j7`**`T`**`kdhj38lkkldkYtr#)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.

- Selain itu, alasannya karena cipher alir itu komputasinya cepat (dibutuhkan untuk *real-time processing*)

Beberapa cipher alir dan tahun pembuatan

- RC4 (1987)
- A5 (1989)
- Rabbit (2003)
- Trivium (2004)
- SEAL (1997)
- Salsa20 (2004)
- Scream (2002)
- SOBER-128 (2003)
- WAKE (2003)
- Turing (2000-2003)
- Scream (2002)
- VEST (2005)
- SNOW (2003)
- CryptMT (2005)
- FISH (1993)
- Grain (2004)
- Isaac (1995)
- MICKEY (2004)

Stream cipher	Creation date	Speed (cycles per byte)	(bits)			Attack	
			Effective key-length	Initialization vector	Internal state	Best known	Computational complexity
A5/1	1989	?	54 or 64 (in 2G)	22 (in 2G)	64	Active KPA OR KPA time–memory tradeoff	~ 2 seconds OR $2^{39.91}$
A5/2	1989	?	54	114	64?	Active	4.6 milliseconds
Achterbahn-128/80	2006	1 (hardware)	80/128	80/128	297/351	Brute force for frame lengths $L \leq 2^{44}$. Correlation attack for $L \geq 2^{48}$.	2^{80} resp. 2^{128} for $L \leq 2^{44}$.
CryptMT	2005	?	Variable	up to 19968	19968	— (2008)	— (2008)
Crypto-1	Pre-1994	?	48	16	48	Active KPA (2008)	40 ms OR 2^{48} (2008)[2]
E0 (cipher)	Pre-1999	?	Variable (usually 128)	4	132	KPA (2005)	2^{38} (2005)[3]
FISH	1993	?	Variable	?	?	Known-plaintext attack	2^{11}
Grain	Pre-2004	?	80	64	160	Key derivation	2^{43}
HC-256	Pre-2004	4 (W_{P4})	256	256	65536	?	?
ISAAC	1996	2.375 ($W_{64\text{-bit}}$) – 4.6875 ($W_{32\text{-bit}}$)	8–8288 (usually 40–256)	—	8288	(2006) First-round weak-internal-state-derivation	4.67×10^{1240} (2001)
MICKEY	Pre-2004	?	80	Variable (0 to 80)	200	Differential Fault Attack (2013)	$2^{32.5}$ (2013)[4]
MUGI	1998–2002	?	128	128	1216	— (2002)	~ 2^{82}

PANAMA	1998	2	256	128?	1216?	Hash collisions (2001)	2^{82}
Phelix	Pre-2004	up to 8 (W_{x86})	256 + a 128-bit nonce	128?	?	Differential (2006)	2^{37}
Pike	1994	?	Variable	?	?	— (2004)	— (2004)
Py	Pre-2004	2.6	8–2048? (usually 40–256?)	64	8320	Cryptanalytic theory (2006)	2^{75}
Rabbit	2003-Feb	3.7(W_{P3}) – 9.7(W_{ARM7})	128	64	512	— (2006)	— (2006)
RC4	1987	7 W_{P5}[5]	8–2048 (usually 40–256)	RC4 does not take an IV. If one desires an IV, it must be mixed into the key somehow.	2064	Shamir initial-bytes key-derivation OR KPA	2^{13} OR 2^{33}
Salsa20	Pre-2004	4.24 (W_{G4}) – 11.84 (W_{P4})	256	a 64-bit nonce + a 64-bit stream position	512	Probabilistic neutral bits method	2^{251} for 8 rounds (2007)
Scream	2002	4–5 (W_{soft})	128 + a 128-bit nonce	32?	64-bit round function	?	?
SEAL	1997	?	?	32?	?	?	?
SNOW	Pre-2003	?	128 or 256	32	?	?	?
SOBER-128	2003	?	up to 128	?	?	Message forge	2^{-6}
SOSEMANUK	Pre-2004	?	128	128	?	?	?
Trivium	Pre-2004	4 (W_{x86}) – 8 (W_{LG})	80	80	288	Brute force attack (2006)	2^{135}
Turing	2000–2003	5.5 (W_{x86})	?	160	?	?	?
VEST	2005	42 (W_{ASIC}) – 64 (W_{FPGA})	Variable (usually 80–256)	Variable (usually 80–256)	256–800	— (2006)	— (2006)
WAKE	1993	?	?	?	8192	CPA & CCA	Vulnerable

(Sumber: https://en.wikipedia.org/wiki/Stream_cipher

RC4

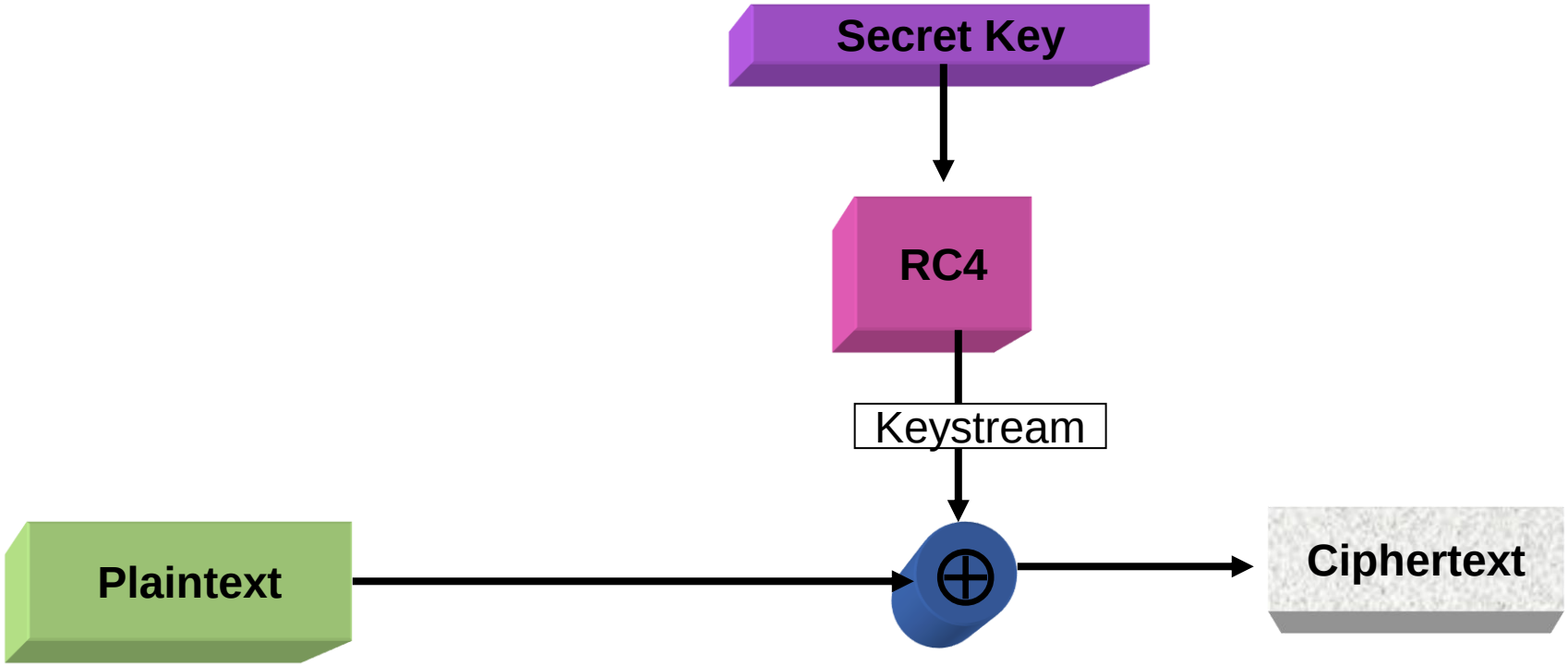
- *Cipher* alir yang paling populer
- Dibuat oleh Ronald Rivest (1987) dari Laboratorium *RSA*
- *RC* adalah singkatan dari *Ron's Code*. Versi lain mengatakan *Rivest Cipher* .
- Digunakan di dalam beberapa sistem keamanan seperti:

  - protokol *SSL* (*Secure Socket Layer*) dan TLS (*Transport Layer Security*)

  - Standard IEEE 802.11 *wireless* LAN: WEP (*Wired Equivalent Privacy*)

  - WPA (*Wi-fi Protocol Access*) untuk nirkabel

Diagram Blok RC4:



- Kunci rahasia (*secret key*) memiliki panjang maksimal 256 karakter (1 karakter = 1*byte*). Jika panjang kunci kurang dari 256 *byte* maka karakter-karakter kunci diulang secara periodik.
- RC4 menghasilkan luaran berupa kunci-alir (*keystream*) dengan panjang tak terbatas

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

- 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 rahasia *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 rahasia K (kunci eksternal dari pengguna):

```
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
```

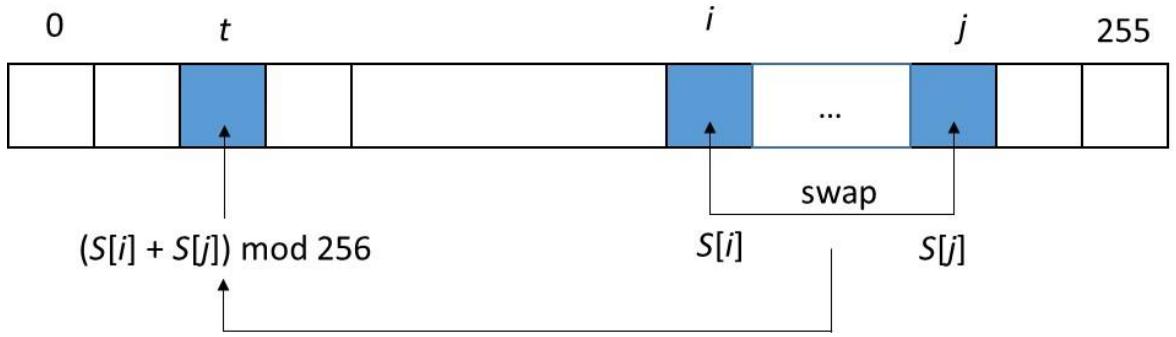
- Permutasi ini menyebabkan elemen-elemen di dalam larik *S* teracak.

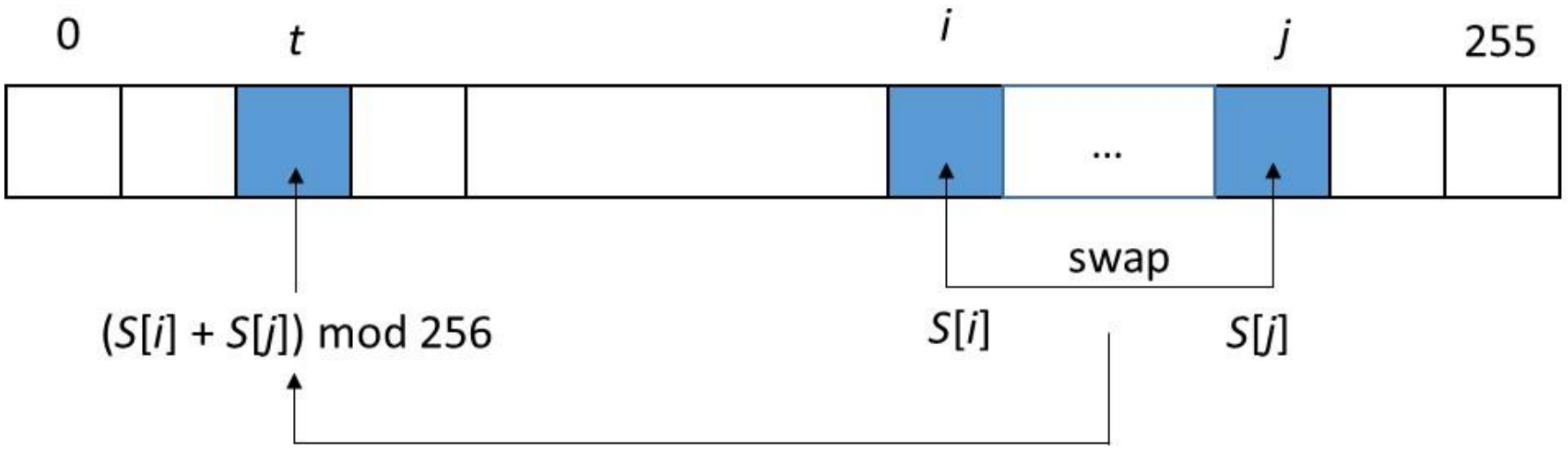
- `K[i mod Length(K)]` menyatakan karakter-karakter kunci diulang secara periodik jika panjangnya kurang dari 256

Pseudo-random generation algorithm (*PRGA*)

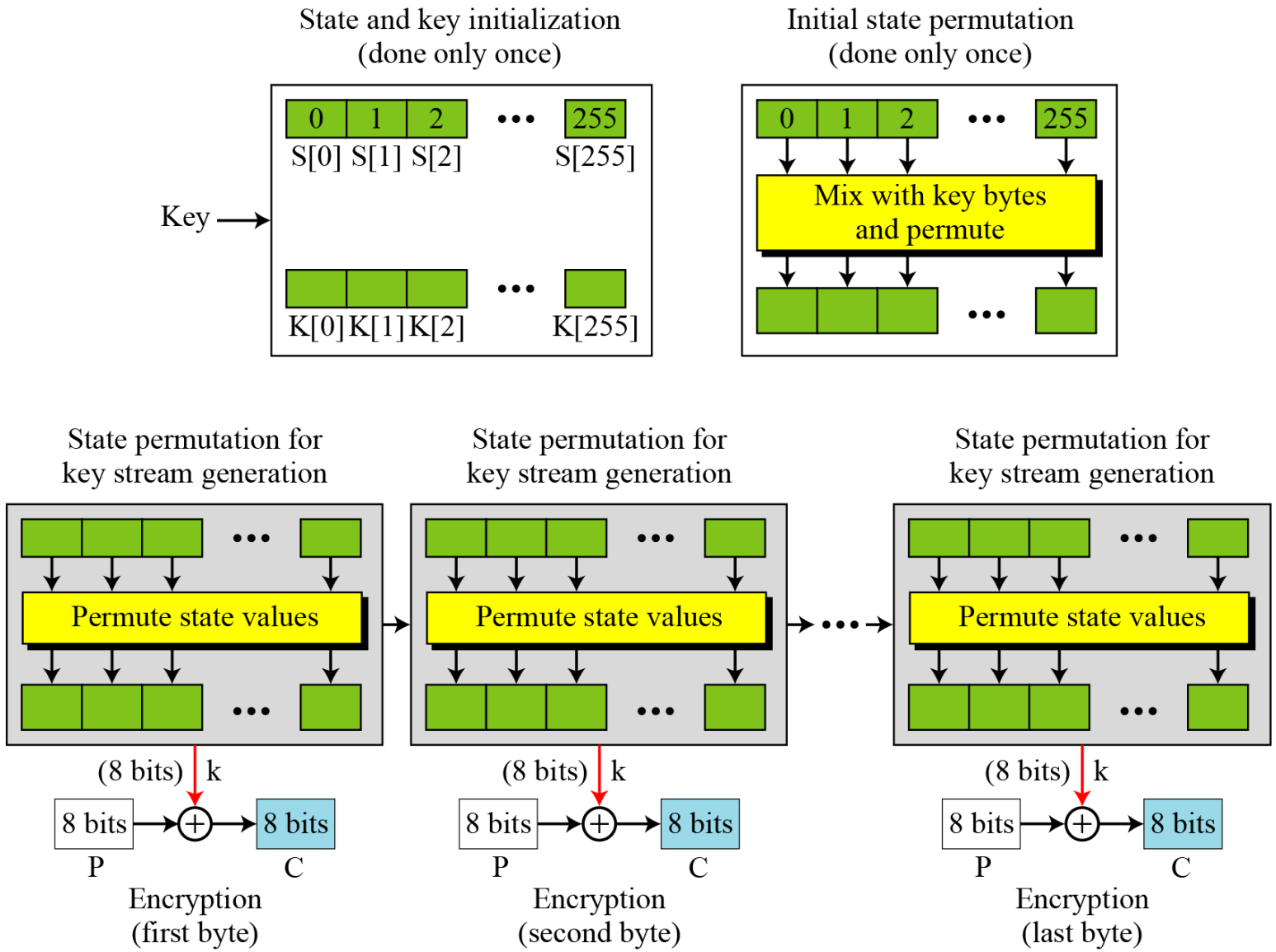
- *PRGA* membangkitkan kunci-alir (*keystream*) dengan cara mengambil nilai *S*[*i*] dan *S*[*j*], mempertukarkannya, lalu menjumlahkan keduanya dalam modulus 256.
- Kunci alir tersebut kemudian di-XOR-kan dengan sebuah karakter plainteks.

```
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]

- Operasi RC4 secara keseluruhan:



Keamanan RC4

- RC4 memiliki kelemahan, yaitu *byte-byte* pada *keystream* awal pembangkitan memiliki korelasi yang tinggi dengan beberapa *byte* awal kunci
- Oleh karena itu direkomendasikan membuang 256 sampai 512 *byte keystream* awal
- 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).
- Beberapa varian RC4 telah dibuat untuk mengatasi kelemahannya, yaitu Spritz, RC4A, VMPC, dan RC4+.

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;
```

```
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;
while (!feof(Fin)) {
    p = fgetc(Fin);
    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);
 }
 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;
```

```
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;
while (!feof(Fin)) {
    c = fgetc(Fin);
    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);
 }
 fclose(Fin);  fclose(Fout);
}
```

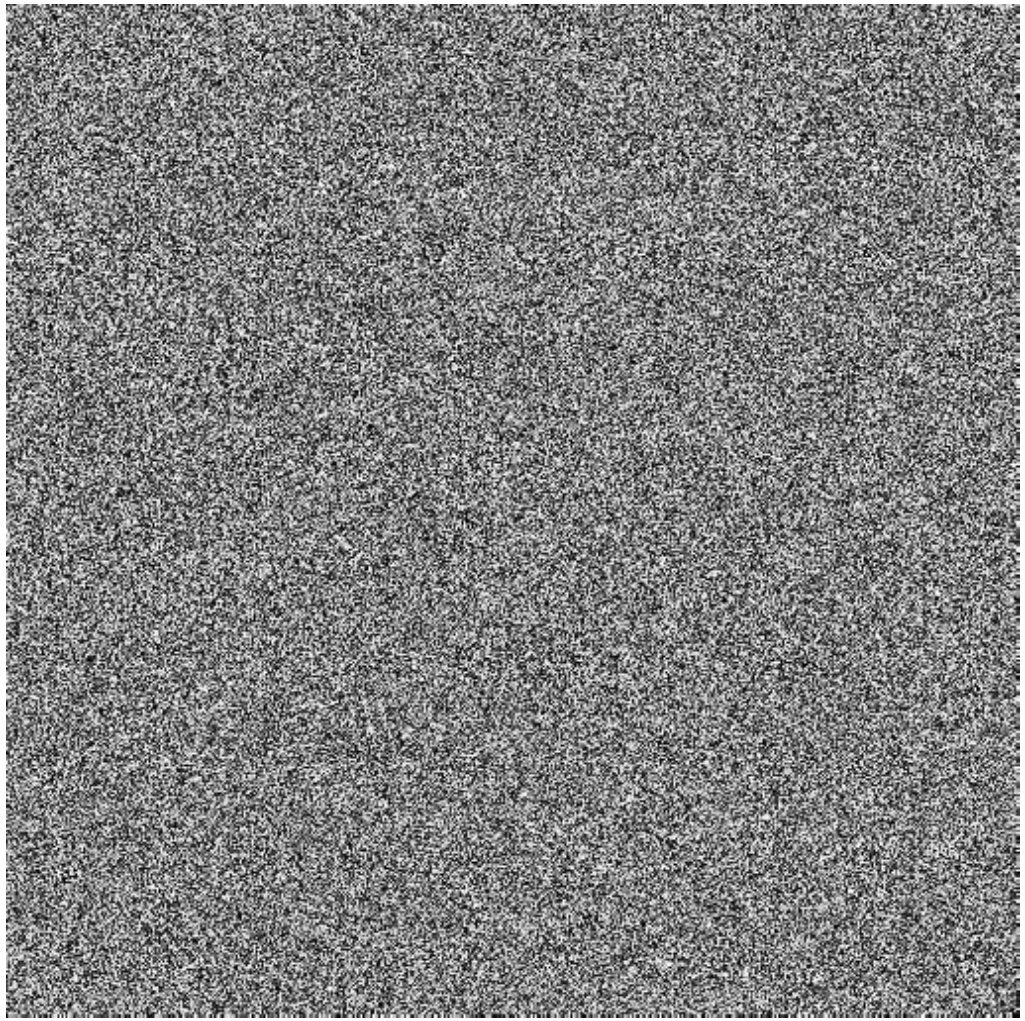
Enkripsi citra dengan RC4

- RC4 cocok digunakan untuk mengenkripsi file citra (*image*) tanpa merusak header filenya,
- karena citra terdiri atas sejumlah *pixel*, setiap pixel berukuran 1 byte (*grayscale image*) sampai 3 byte (*color image*).

- Ingatlah bahwa RC4 membangkitkan *keystream* berupa rangkaian *byte*

- Dengan mengenkripsi setiap *byte pixel* dengan setiap *byte keystream*, pixel-pixel citra terenkripsi.

Plain-image



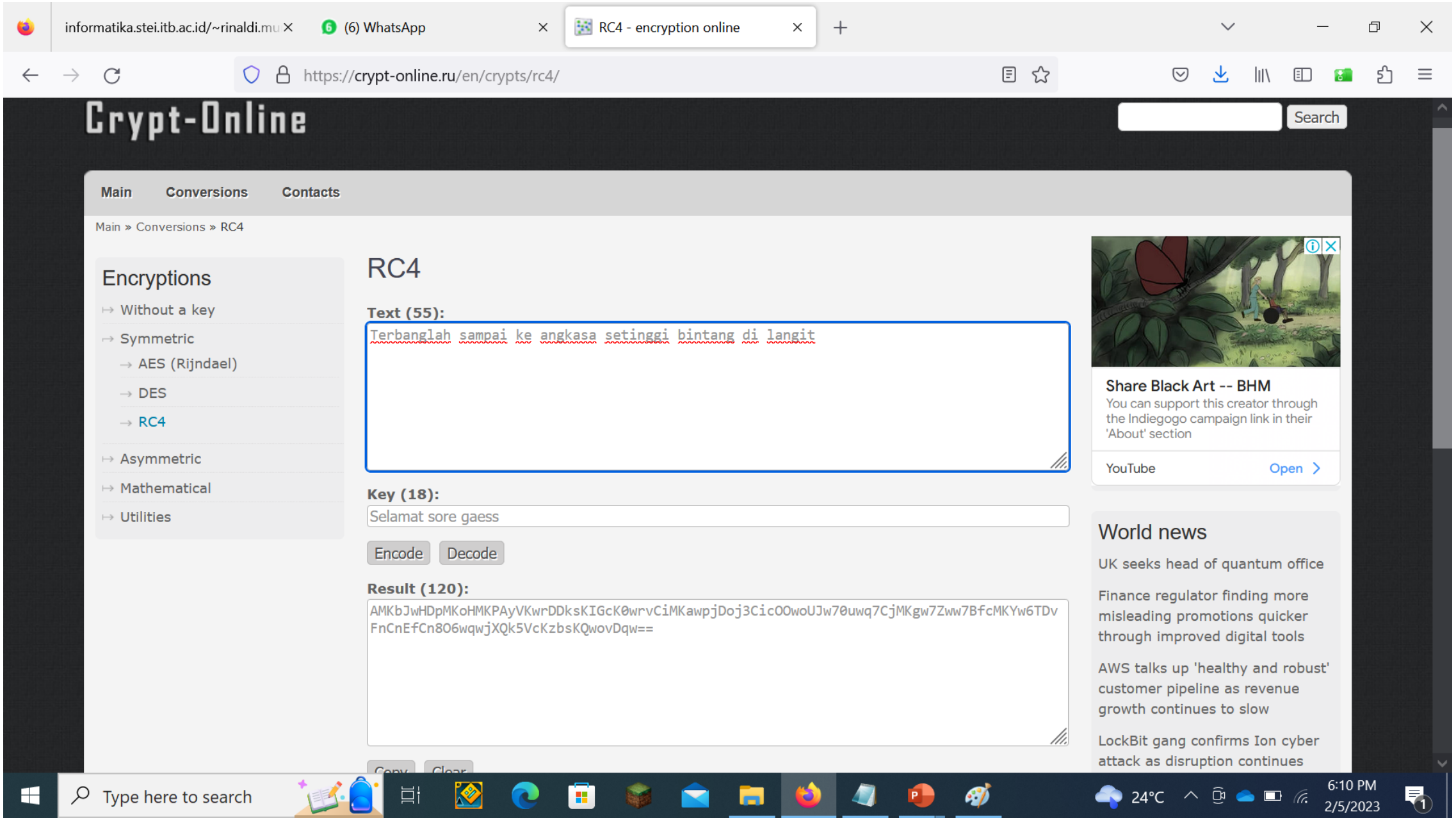
Encrypted image

Demo RC4 Online

1. https://crypt-online.ru/en/crypts/rc4/
2. https://www.lddgo.net/en/encrypt/rc4

https://crypt-online.ru/en/crypts/rc4/

Enkripsi



Dekripsi

informatika.stei.itb.ac.id/~rinaldi.mu | (6) WhatsApp | RC4 - encryption online

https://crypt-online.ru/en/crypts/rc4/

Crypt-Online

Search

Main | Conversions | Contacts

Main » Conversions » RC4

Encryptions

- Without a key
- Symmetric
  - AES (Rijndael)
  - DES
  - RC4
- Asymmetric
- Mathematical
- Utilities

RC4

Text (120):

AMKbJwHDpMKoHMKPAyVKwrDDksKIGcK0wrvCiMKawpjDoj3CicOOwoUJw70uwq7CjMKgw7Zww7BfcMKYw6TDv
FnCnEfCn8O6wqwjXQk5VcKzbsKQwovDqw==

Key (18):

Selamat sore gaess

Encode | Decode

Result (55):

Terbanglah sampai ke angkasa setinggi bintang di langit

Share Black Art -- BHM

You can support this creator through the Indiegogo campaign link in their 'About' section

YouTube Open

World news

UK seeks head of quantum office

Finance regulator finding more misleading promotions quicker through improved digital tools

AWS talks up 'healthy and robust' customer pipeline as revenue growth continues to slow

LockBit gang confirms Ion cyber attack as disruption continues

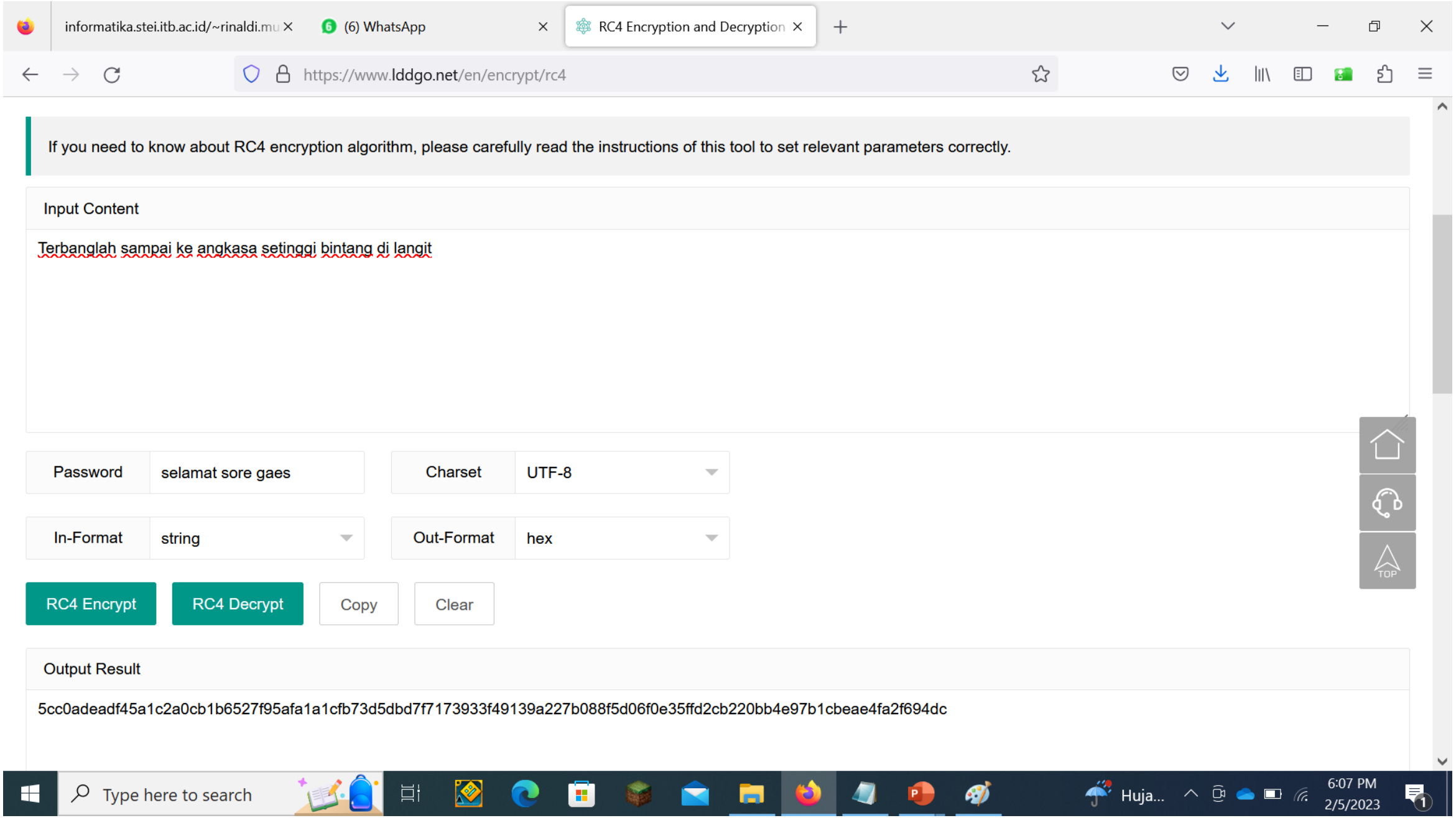
Type here to search

Senja

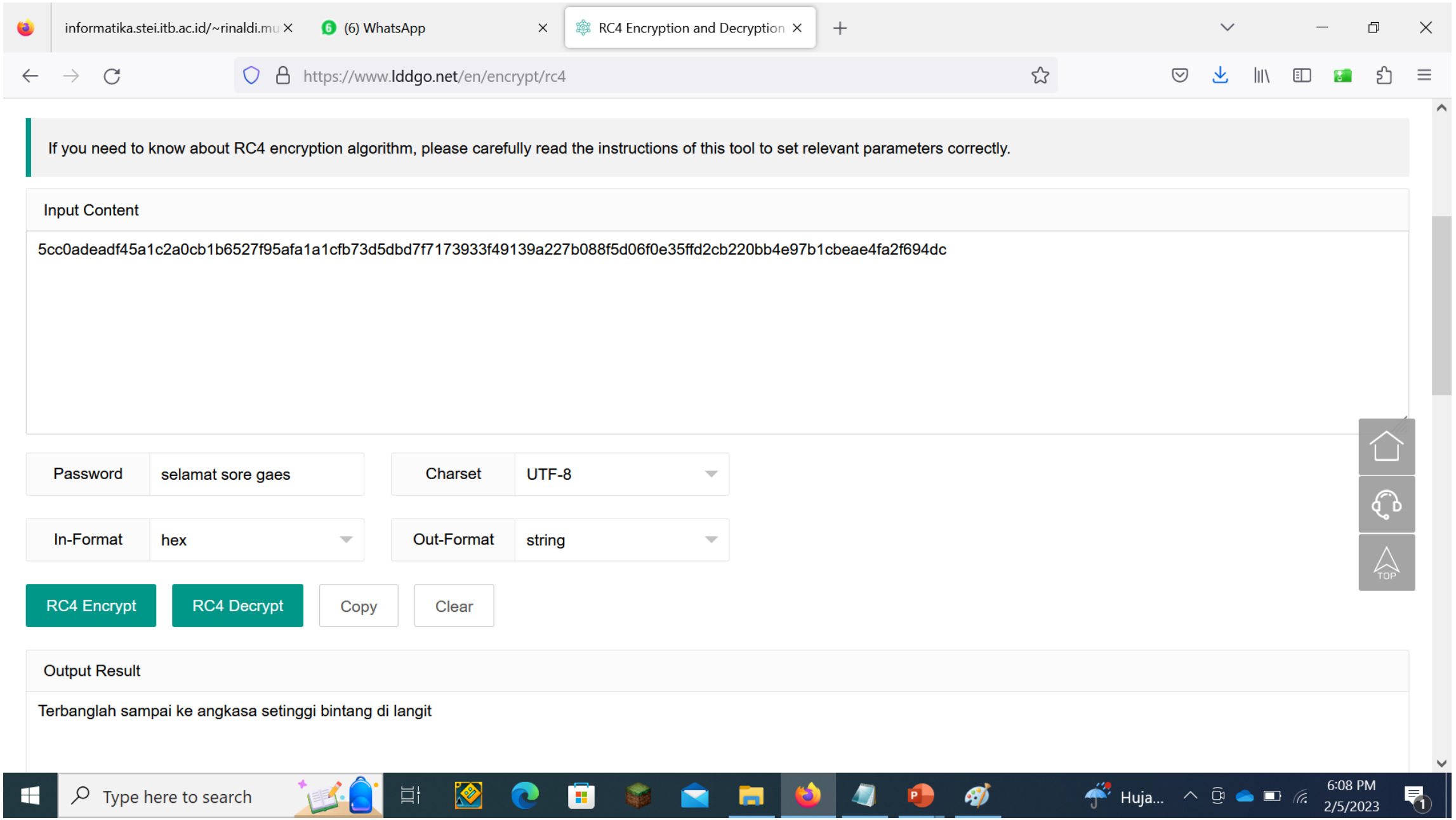
6:12 PM
2/5/2023

https://www.lddgo.net/en/encrypt/rc4

Enkripsi



Dekripsi


informatika.stei.itb.ac.id/~rinaldi.mu
(6) WhatsApp
RC4 Encryption and Decryption
https://www.lddgo.net/en/encrypt/rc4
If you need to know about RC4 encryption algorithm, please carefully read the instructions of this tool to set relevant parameters correctly.
Input Content
5cc0adeadf45a1c2a0cb1b6527f95afa1a1cfb73d5dbd7f7173933f49139a227b088f5d06f0e35ffd2cb220bb4e97b1cbeae4fa2f694dc
Password
selamat sore gaes
Charset
UTF-8
In-Format
hex
Out-Format
string
RC4 Encrypt
RC4 Decrypt
Copy
Clear
Output Result
Terbanglah sampai ke angkasa setinggi bintang di langit
TOP
Type here to search
Huja...
6:08 PM
2/5/2023

A5

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

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

- *A5* menghasilkan kunci-alir (*keystream*) sepanjang 228-bit yang kemudian bit-bitnya di-*XOR*-kan dengan bit-bit *frame* pesan sepanjang 228 bit. 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

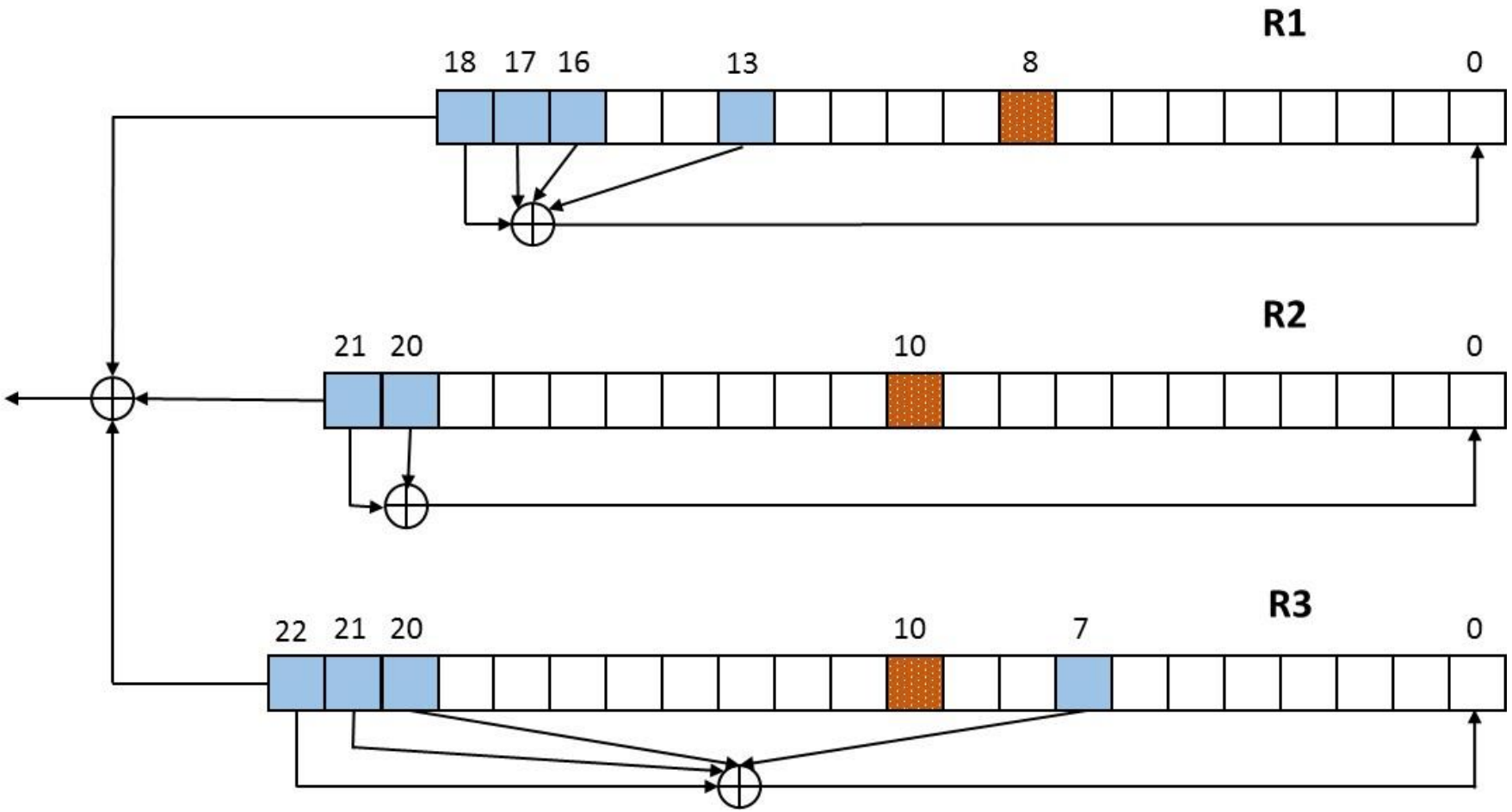
- Algoritma A5/1 pada awalnya rahasia, tetapi pada tahun 1994 melalui *reverse engineering*, algoritmanya terbongkar.

- *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, yaitu 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*:

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

   - tiap register kemudian didetak

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*:

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

   - tiap register kemudian didetak

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*.

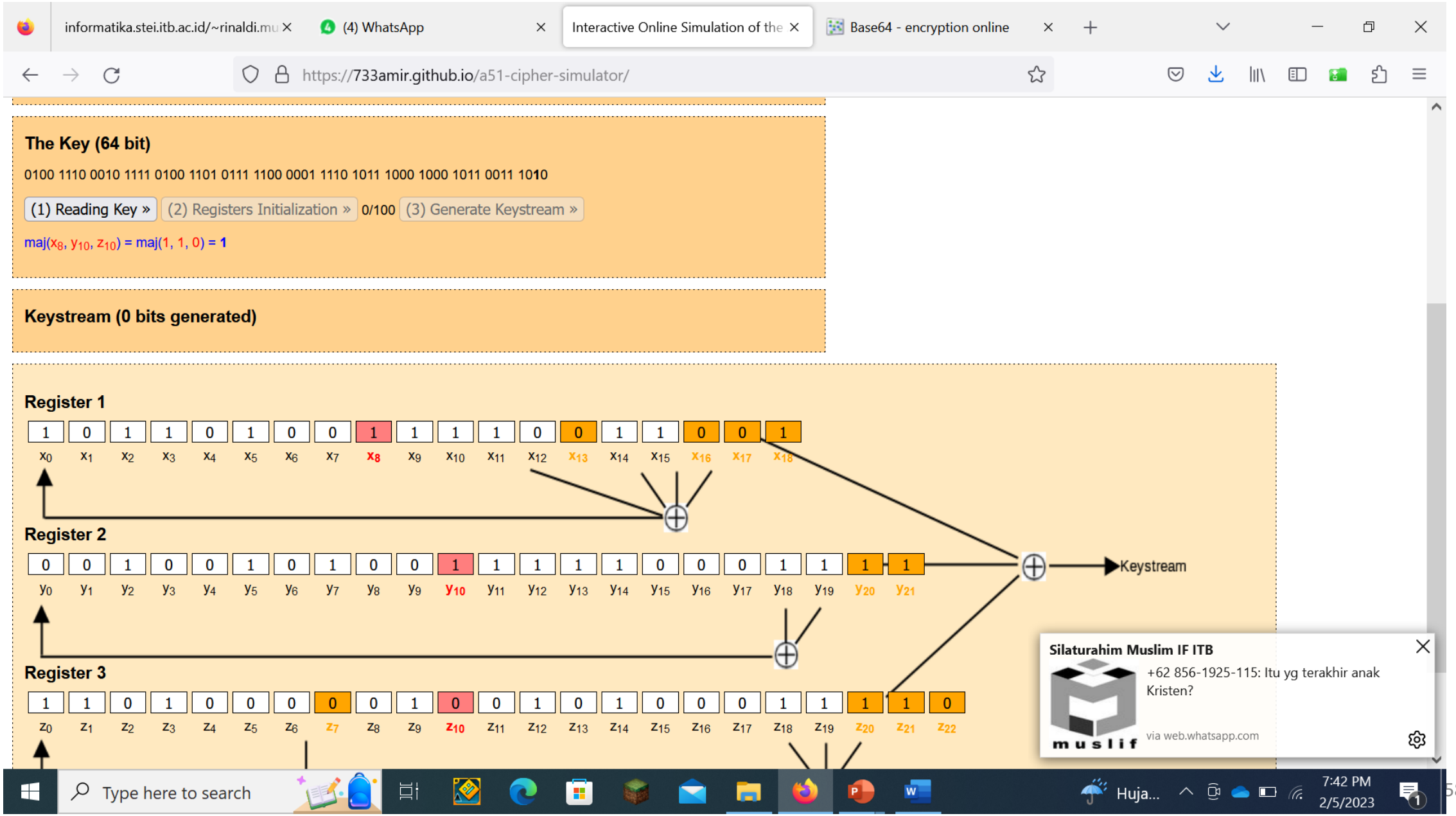
- Serangan semacam ini pertama kali dilakukan oleh Ross Anderson pada tahun 1994.

- 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.

Interactive Online Simulation of the A5/1 Cipher (https://733amir.github.io/a51-cipher-simulator/)



Trivium

- Trivium adalah *cipher-alir* modern, merupakan salah satu cipher alir yang di-submit pada kompetisi *stream cipher* eSTREAM pada tahun 2004 - 2008.

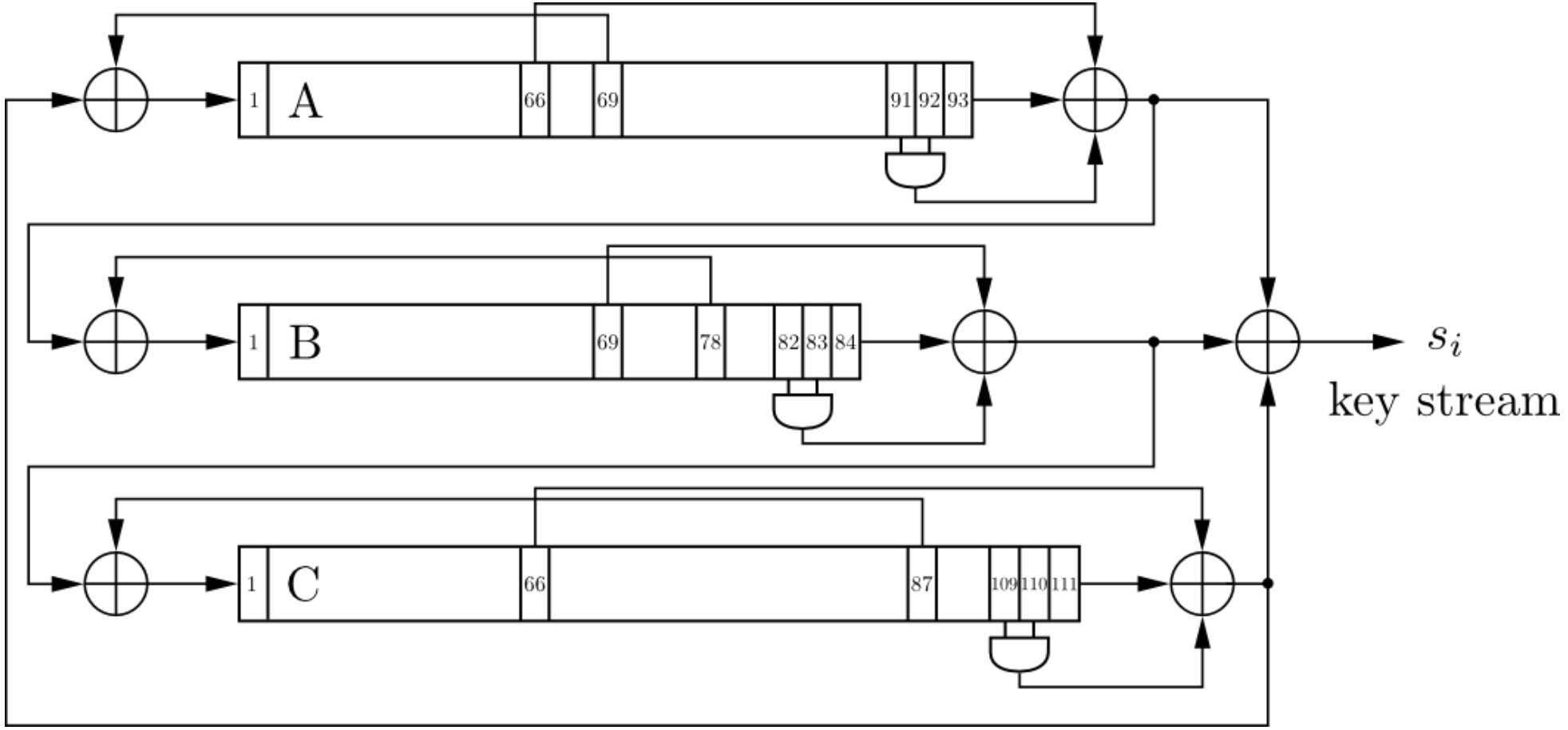
- Dibuat oleh Christophe De Cannière dan Bart Preneel.

- Menggunakan tiga buah *nonlinear LFSRs* (NLFSR) dengan panjang masing-masing 93, 84, dan 111 bit.
- Melakukan XOR terhadap tiga buah luaran NLFSR untuk membangkitkan bit kunci-alir.
- Minimalis jika diimplementasikan pada *hardware*:

  - total sel register: 288

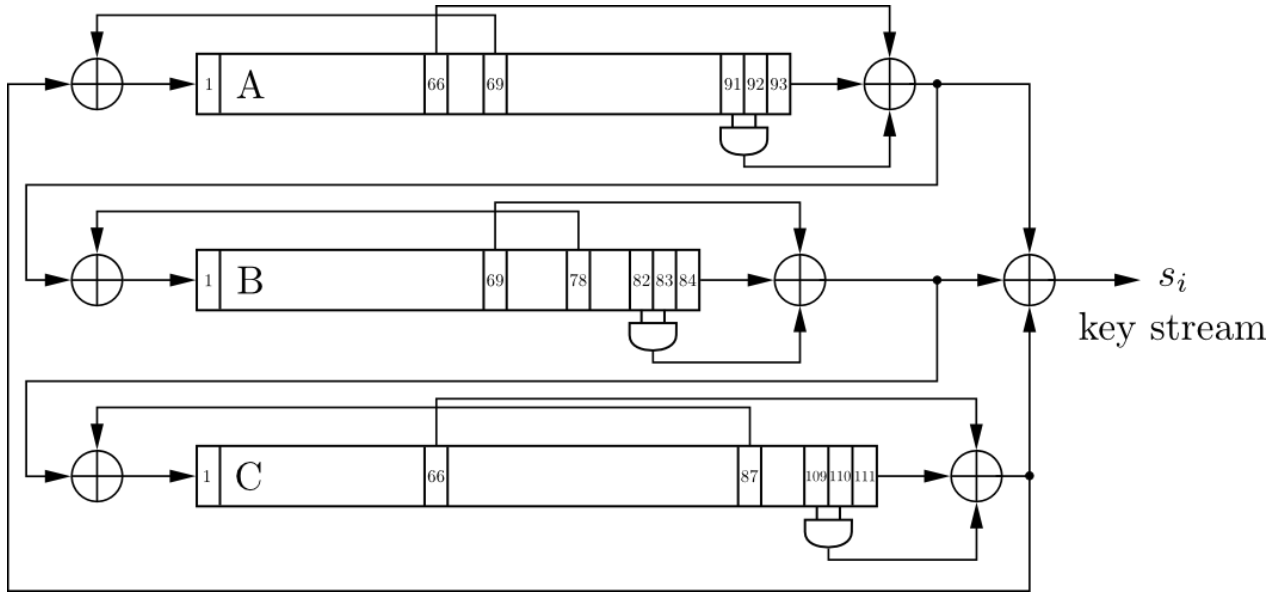
  - tiga buah gerbang AND

  - tujuh buah gerbang XOR



	Register length	Feedback bit	Feedforward bit	AND inputs
A	93	69	66	91, 92
B	84	78	69	82, 83
C	111	87	66	109, 110

Sumber: Chapter 2 of *Understanding Cryptography* by Christof Paar and Jan Pelzl



Inisialisasi:

- Masukkan 80-bit *initialization vector* (IV) ke dalam A
- Masukkan 80-bit kunci ke dalam B
- Set c_{109} , c_{110} , c_{111} =1, sedangkan bit lainnya 0

Pemanasan:

- Detak (*clock*) NLFSR 4 x 288 = 1152 kali tanpa menghasilkan luaran (luaran diabaikan)

Encryption:

- XOR-kan ketiga buah luaran NLFSR untuk membangkitkan kunci-alir s_i
- XOR-kan s_i dengan bit plainteks p_i

Sumber: Chapter 2 of *Understanding Cryptography* by Christof Paar and Jan Pelzl