

| noersilo | jenny | meysita | melando |

<http://www.noersilo.web.id>

[noersilo@yahoo.com](mailto:noersilo@yahoo.com)

## DESAIN ALGORITMA STREAM D'JAZT

### 3.1 Pengenalan Algoritma D'JAZT Cipher

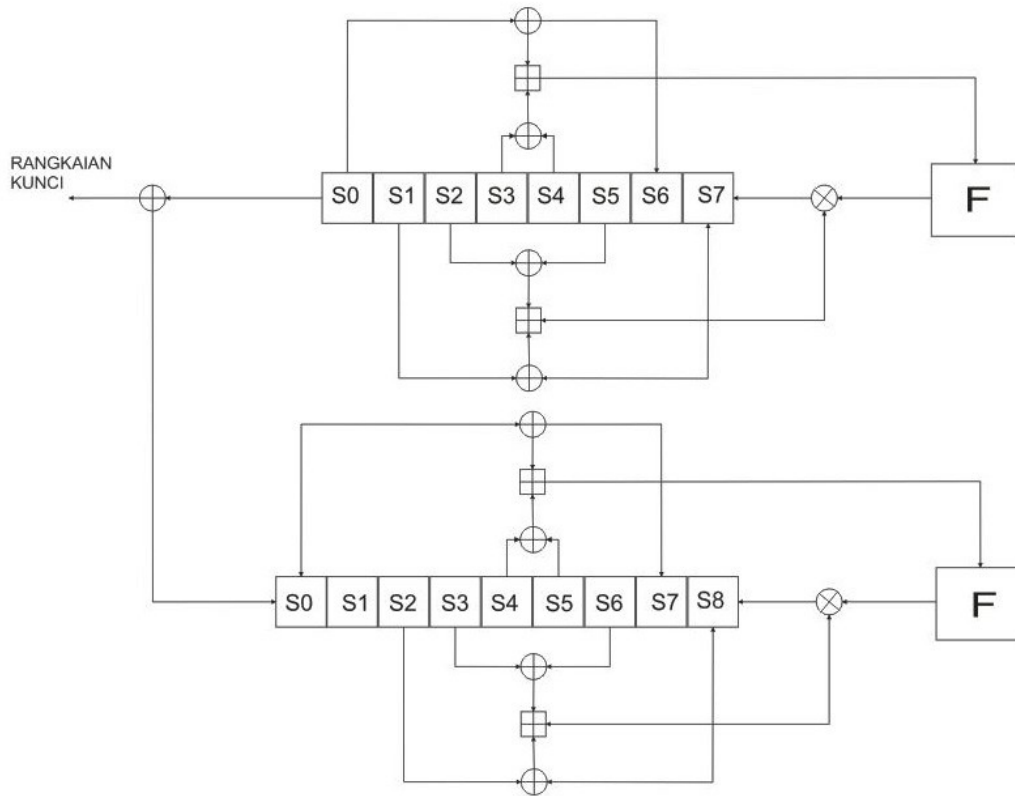
Kekuatan algoritma stream cipher terletak pada keacakan rangkaian kunci yang dihasilkan bukan tergantung pada kerahasiaan algoritmanya. Proses enkripsi pada stream cipher umumnya hanya meng-XOR plaintext dengan rangkaian kunci stream yang dihasilkan dari algoritma pembangkitan kunci.

Proses pembangkitan rangkaian kunci stream cipher yang berorientasi pada bit dengan menggunakan operasi Linier Feedback Shift Register (LFSR) akan menghasilkan rangkaian kunci yang sangat mudah diinverskan, sehingga algoritma ini dapat dikatakan lemah. Oleh karena itu diperlukan fungsi-fungsi tambahan, sehingga operasi pengacakan rangkaian kunci outputnya menjadi fungsi *Non-Linear Feedback Shift Register (NLFSR)*, hal ini penting untuk dilakukan agar dalam pembangkitan rangkaian kunci outputnya tidak mudah untuk diinverskan, selain itu periode rangkaian kuncinya juga akan bertambah panjang. Untuk mengatasi masalah tersebut kami mencoba membuat algoritma stream cipher yang kami beri nama D'JAZT berbasis NLFSR dan berorientasi pada byte (8-bit). Hal ini dimaksudkan agar rangkaian kunci yang dihasilkan memiliki keacakan yang baik, panjang periode yang maksimum dan yang terakhir adalah cepat secara komputasi.

### 3.2 Algoritma D'JAZT Stream Cipher

Algoritma D'JAZT adalah algoritma *Synchronous Stream Cipher* yang berbasis NLFSR. Algoritma ini berorientasi pada byte (8-bit). Algoritma ini terdiri atas bagian pembangkitan rangkaian kunci dengan penambahan fungsi F. Bagian pembangkitan rangkaian kunci ini terdiri atas 2 buah LFSR (R1 dan R2), yang mempunyai nilai inputan state yang berlainan dengan output dari masing-masing LFSR di-XOR untuk menghasilkan rangkaian kunci. LFSR 1 terdiri dari 8-stages dan

LFSR 2 terdiri dari 9-stages. State-state tersebut melalui peng-XORan dan penjumlahan modulo  $2^8$ , serta perkalian modulo  $2^8$ . Untuk skema algoritma dari stream cipher D'JAZT dapat dilihat pada gambar.



Gambar 6. Skema Algoritma D'JAZT Cipher

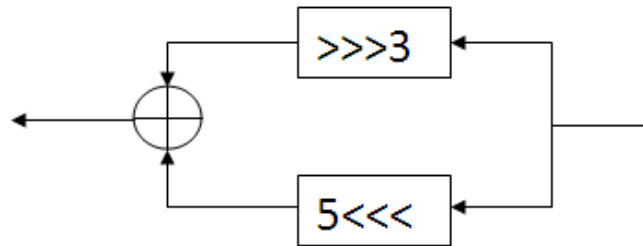
Algoritma ini menggunakan 2 buah LFSR, dimana dalam satu LFSR setiap stage mengalami pengoperasian XOR dengan stage yang lainnya, dan hasil XOR akan dioperasikan dengan penjumlahan modulo  $2^8$ , sehingga output yang dihasilkan oleh masing-masing LFSR maksimal. Operasi yang digunakan di tiap LFSR terdiri dari fungsi XOR, penjumlahan, perkalian, dan pergeseran. Operasi ini menggunakan modulo  $2^8$ .

Berikut fungsi penjumlahan:

$$S_1(x) = R_1 + R_2 \pmod{2^8}$$

$$S_2(x) = R_3 + R_4 \pmod{2^8}$$

Selanjutnya output dari  $S_1$  akan masuk ke dalam fungsi  $F$  dan mengalami pergeseran setiap bitnya.  $S_1$  akan melakukan pergeseran ke kiri sebanyak 5 kali dan ke kanan sebanyak 3 kali dari output awal  $S_1$ . Lalu, output dari fungsi  $F$  ini akan dikalikan dengan output dari  $S_2$ , dan menghasilkan feedback dari LFSR tsb. Untuk Skema dari fungsi  $F$  dapat dilihat pada gambar.



Gambar 8. Skema Fungsi  $F$  D'JAZT Cipher

Jumlah stages pada masing-masing LFSR berbeda, hal ini karena  $\text{gcd}(8,9)=1$ , sehingga ketika output dari kedua LFSR saling dioperasikan untuk menghasilkan rangkaian kunci yang berperiode maksimal.

Rangkaian kunci stream yang dihasilkan dari algoritma ini didapatkan dari peng-XORan output dari masing-masing LFSR. Rangkaian kunci yang dihasilkan berjumlah 8 byte atau 64 bit. Rangkaian kunci ini diharapkan memenuhi standar keacakan kunci dengan uji statistik, dan dapat menghasilkan periode yang maksimum. Rangkaian kunci direpresentasikan dalam bentuk hexadecimal.

## BAB IV

### ANALISIS

#### 4.1 Parameter Synchronous Stream Cipher

- a. Periode rangkaian kunci yang dihasilkan harus maksimum

Bagian pembangkit kunci pada algoritma D'JAZT cipher adalah NLFSR. Hal ini penting untuk dilakukan agar dalam pembangkitan rangkaian kunci outputnya tidak mudah untuk diinverskan, selain itu periode rangkaian kuncinya juga akan bertambah panjang. NLFSR berasal dari 2 buah LFSR dengan jumlah stage menghasilkan  $\text{gcd}=1$ , sehingga menghasilkan periode yang maksimal. Operasi yang digunakan ialah kombinasi dari pengoperasian XOR, penjumlahan modulo  $2^8$ , dan perkalian, sehingga output yang dihasilkan oleh masing-masing LFSR benar-benar acak. NLFSR juga berorientasi pada byte (8-bit). Hal ini dimaksudkan agar rangkaian kunci yang dihasilkan memiliki keacakan yang baik dan panjang periode yang maksimum.

- b. Rangkaian kunci yang dihasilkan harus random

Rangkaian kunci yang dihasilkan algoritma D'JAZT cipher dapat dikatakan random, karena menggunakan NLFSR. NLFSR menggunakan operasi XOR, penjumlahan, perkalian, pergeseran modulo  $2^8$  sebagai feedback yang menambah keacakan pada NLFSR.

- c. Nonlinearity

Nonlinearity pada algoritma D'JAZT cipher terletak pada NLFSR. Sebagai feedback, operasi yang digunakan di tiap LFSR terdiri dari fungsi XOR, penjumlahan, perkalian, dan pergeseran. Operasi ini menggunakan modulo  $2^8$ . Operasi XOR pada LFSR didasarkan pada persamaan polinomial-polinomial primitive yang digunakan untuk mencegah pola.

- d. Untraceable

Salah satu syarat algoritma *synchronous stream cipher* yang baik adalah cryptanalisis tidak dapat dengan mudah mendapatkan *key input* dengan hanya

mendapatkan *encryption key*. Algoritma stream cipher D'JAZT cipher menggunakan NLFSR yang tidak mudah untuk diinverskan, sehingga ketika seorang cryptanalisis ingin mentrace kembali bit input dari key generator, maka dia harus mengetahui dahulu fungsi-fungsi polinomial dan operasi-operasi yang ada dan menginverskannya. Selain itu algoritma D'JAZT cipher yang juga berbasis NLFSR menggunakan fungsi-fungsi nonlinear, hal ini tentu saja mempersulit cryptanalisis melakukan *attack*.

e. No Error Propagation

Karena algoritma D'JAZT cipher merupakan algoritma *synchronous stream cipher*, maka kesalahan dekripsi satu *ciphertext* hanya akan mempengaruhi *plaintext* yang bersesuaian, tidak akan berpengaruh terhadap *ciphertext* yang lain. Artinya *ciphertext* lain akan dapat didekripsi secara benar.

## 4.2 Design

Design algoritma D'JAZT cipher sederhana dan proses matematis yang dilakukan tidak rumit. Desain algoritma D'JAZT cipher sangat cocok untuk diimplementasikan untuk enkripsi data dengan kemampuan pemroses data serta memori yang kecil, contohnya untuk aplikasi-aplikasi mobile pada *handphone*, *pda*, maupun pada mikrokontroler.

## 4.3 Kekuatan

Kekuatan pada algoritma D'JAZT cipher terletak pada nonlinearitas yang memungkinkan NLFSR untuk memperoleh panjang periode kunci yang maksimum.

## **BAB V**

### **PENUTUP**

#### **5.1 Kesimpulan**

Keamanan suatu operasi pada stream cipher tidak hanya tergantung pada operasi standar LFSR saja, perlu adanya tambahan operasi yang membuat kenonlinearan operasi tersebut sehingga menambah keacakan output yang berpengaruh pada keamanannya. D'JAZT merupakan salah satu desain stream cipher yang berbasiskan byte (8 bit) dan menggunakan operasi NLFSR. Desain utama dari algoritma ini menggunakan 2-LFSR berbeda dengan operasi yang sama yang kemudian dikombinasikan. Fungsi dari penggunaan 2-LFSR ini adalah untuk mendapatkan rangkaian kunci acak dan maksimal. Pada algoritma ini LFSRnya berisi 8-stage yang masing-masing stagenya berisi 1-byte(8 bit). Fungsi yang digunakan cukup sederhana dengan hanya menggunakan fungsi XOR, penjumlahan, perkalian dan pergeseran bit.

#### **5.2 Saran**

- Perlu dilakukan pengujian terhadap kekuatan algoritma D'JAZT
- Perlunya pengembangan kompleksitas fungsi non linear pada algoritma D'JAZT

## SOURCE CODE

```
/*
 * D'JAZT STREAM CIPHER ALGORITHM
 * NURSILO;
 * JENNY ;
 * MEYSITA;
 * MELANDO;
 */

package streamc4;
class dec2bin{
    int Tobin(int tes){
        int temp2[]=new int[8];
        int temp;
        for(int i=0;i<8;i++){
            temp=(tes&0xff)%2;
            tes = tes/2;
            temp2[i]=temp;
        }
        for(int j=7;j>=0;j--){
            System.out.print(temp2[j]);
        }
        if(j==4){
            System.out.print(" ");
        }
        System.out.print(" ");
    }
    return tes;
}

}

public class Main {

    public static void main(String[] args) {
        byte[]St={0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x22};
        byte[]Bt={0x44,0x11,0x11,0x11,0x22,0x33,0x44,0x55,0x11};
        dec2bin bin=new dec2bin();

        /* Proses Operasi */
        for(int i=0;i<10000;i++)
        {
            byte xor1 = (byte) ((St[0] ^ St[6]) & 0xff);
            byte xor2 = (byte) ((St[3] ^ St[4]) & 0xff);
            byte xor3 = (byte) ((St[2] ^ St[5]) & 0xff);
            byte xor4 = (byte) ((St[1] ^ St[7]) & 0xff);
            byte add1 = (byte) ((xor1 + xor2) % 256);
            byte add2 = (byte) ((xor3 + xor4) % 256);

            byte xor5 = (byte) ((Bt[0] ^ Bt[7]) & 0xff);
            byte xor6 = (byte) ((Bt[4] ^ Bt[5]) & 0xff);
            byte xor7 = (byte) ((Bt[3] ^ Bt[6]) & 0xff);
            byte xor8 = (byte) ((Bt[2] ^ Bt[8]) & 0xff);
            byte add3 = (byte) ((xor5 + xor6) % 256);
            byte add4 = (byte) ((xor7 + xor8) % 256);

            if (i!=0 & i%8==0){
                System.out.println();
            }

            /* Fungsi F ....*/

            byte a = (byte) (add1 << 5);
        }
    }
}
```

```

byte b = (byte) (add1 >> 3);
byte c = (byte) (a ^ b);
byte d = (byte) ((add2 * c) % 256);

byte e = (byte) (add3 << 5);
byte f = (byte) (add3 >> 3);
byte g = (byte) (e ^ f);
byte h = (byte) ((add4 * g) % 256);

byte rk = (byte)(St[0] ^ Bt[0]);

//Tampilan layout

System.out.println("Shift ke- "+ i);
System.out.println("LFSR_1");
System.out.print("St[0]          = ");
bin.Tobin(St[0]);System.out.println();
System.out.print("St[i]          = ");
bin.Tobin(St[0]);bin.Tobin(St[1]);bin.Tobin(St[2]);bin.To
bin(St[3]);bin.Tobin(St[4]);bin.Tobin(St[5]);bin.Tobin(St[6]);bin.
Tobin(St[7]);
System.out.println();
System.out.print("St[0] ^ St[6]   = ");
bin.Tobin(xor1);System.out.println();
System.out.print("St[3] ^ St[4]   = ");
bin.Tobin(xor2);System.out.println();
System.out.print("St[2] ^ St[5]   = ");
bin.Tobin(xor3);System.out.println();
System.out.print("St[1] ^ St[7]   = ");
bin.Tobin(xor4);System.out.println();
System.out.print("Add_1          = ");
bin.Tobin(add1);System.out.println();
System.out.print("Add_2          = ");
bin.Tobin(add2);System.out.println();
System.out.print("Hasil Fungsi f = ");
bin.Tobin(c);System.out.println();
System.out.print("Feedback LFSR_1 = ");
bin.Tobin(d);System.out.println();
System.out.println();

System.out.println("LFSR_2");

System.out.print("Bt[0]          = ");
bin.Tobin(Bt[0]);System.out.println();

System.out.print("Bt[i]          = ");
bin.Tobin(Bt[0]);bin.Tobin(Bt[1]);bin.Tobin(Bt[2]);bin.To
bin(Bt[3]);bin.Tobin(Bt[4]);bin.Tobin(Bt[5]);bin.Tobin(Bt[6]);bin.
Tobin(Bt[7]);
System.out.println();

System.out.print("Bt[0] ^ Bt[6]   = ");
bin.Tobin(xor5);System.out.println();
System.out.print("Bt[3] ^ Bt[4]   = ");
bin.Tobin(xor6);System.out.println();
System.out.print("Bt[2] ^ Bt[5]   = ");
bin.Tobin(xor7);System.out.println();
System.out.print("Bt[1] ^ Bt[7]   = ");
bin.Tobin(xor8);System.out.println();
System.out.print("Add_3          = ");
bin.Tobin(add3);System.out.println();
System.out.print("Add_4          = ");
bin.Tobin(add4);System.out.println();

```

```

        System.out.print("Hasil Fungsi f = ");
        bin.Tobin(g);System.out.println();
        System.out.print("Feedback LFSR_2 = ");
        bin.Tobin(h);System.out.println();
        System.out.println();
        System.out.print("Key Stream ke-"+ i);System.out.print("
= ");
        bin.Tobin(rk);System.out.println();
        System.out.println();

        /*Pergeseran LFSR*/

        for (int j=1;j<=7;j++)
        {
            St[j-1]=St[j];
            Bt[j-1]=Bt[j];
            if (j==7)
            {
                St[7]=d;
                Bt[7]=h;
            }
        }
    }
}

```

## TEST VECTOR I

### Test vector dengan 10 kali pergeseran LFSR

```

init:
deps-jar:
compile:
run:
Shift ke- 0
LFSR_1
St[0]          = 0001 0001
St[i]          = 0001 0001 0010 0010 0011 0011 0100 0100 0101 0101
    0110 0110 0111 0111 0010 0010
St[0] ^ St[6] = 0110 0110
St[3] ^ St[4] = 0001 0001
St[2] ^ St[5] = 0101 0101
St[1] ^ St[7] = 0000 0000
Add_1         = 0111 0111
Add_2         = 0101 0101
Hasil Fungsi f = 0001 0010
Feedback LFSR_1 = 0000 0110

LFSR_2
Bt[0]          = 0100 0100
Bt[i]          = 0100 0100 0001 0001 0001 0001 0001 0001 0010 0010
    0011 0011 0100 0100 0101 0101
Bt[0] ^ Bt[6] = 0001 0001
Bt[3] ^ Bt[4] = 0001 0001
Bt[2] ^ Bt[5] = 0101 0101
Bt[1] ^ Bt[7] = 0000 0000
Add_3         = 0010 0010

```

Add\_4 = 0101 0101  
Hasil Fungsi f = 0100 0100  
Feedback LFSR\_2 = 0110 1100

Key Stream ke-0 = 0101 0101

Shift ke- 1

LFSR\_1

St[0] = 0010 0010  
St[i] = 0010 0010 0011 0011 0100 0100 0101 0101 0110 0110  
0111 0111 0010 0010 0000 0110  
St[0] ^ St[6] = 0000 0000  
St[3] ^ St[4] = 0011 0011  
St[2] ^ St[5] = 0011 0011  
St[1] ^ St[7] = 0011 0101  
Add\_1 = 0011 0011  
Add\_2 = 0110 1000  
Hasil Fungsi f = 0110 0110  
Feedback LFSR\_1 = 0111 0000

LFSR\_2

Bt[0] = 0001 0001  
Bt[i] = 0001 0001 0001 0001 0001 0001 0010 0010 0011 0011  
0100 0100 0101 0101 0110 1100  
Bt[0] ^ Bt[6] = 0111 1011  
Bt[3] ^ Bt[4] = 0111 0111  
Bt[2] ^ Bt[5] = 0111 0111  
Bt[1] ^ Bt[7] = 0000 0000  
Add\_3 = 0000 0100  
Add\_4 = 0111 0111  
Hasil Fungsi f = 0111 1111  
Feedback LFSR\_2 = 0000 1001

Key Stream ke-1 = 0011 0011

Shift ke- 2

LFSR\_1

St[0] = 0011 0011  
St[i] = 0011 0011 0100 0100 0101 0101 0110 0110 0111 0111  
0010 0010 0000 0110 0111 0000  
St[0] ^ St[6] = 0011 0101  
St[3] ^ St[4] = 0001 0001  
St[2] ^ St[5] = 0111 0111  
St[1] ^ St[7] = 0011 0100  
Add\_1 = 0100 0110  
Add\_2 = 0101 0101  
Hasil Fungsi f = 0011 1000  
Feedback LFSR\_1 = 0110 1000

LFSR\_2

Bt[0] = 0001 0001  
Bt[i] = 0001 0001 0001 0001 0010 0010 0011 0011 0100 0100  
0101 0101 0110 1100 0000 1001  
Bt[0] ^ Bt[6] = 0001 1000  
Bt[3] ^ Bt[4] = 0001 0001  
Bt[2] ^ Bt[5] = 0101 1001  
Bt[1] ^ Bt[7] = 0011 0011  
Add\_3 = 0010 1001  
Add\_4 = 0010 0110  
Hasil Fungsi f = 0010 0101  
Feedback LFSR\_2 = 0111 1110

Key Stream ke-2 = 0010 0010

Shift ke- 3

LFSR\_1

St[0] = 0100 0100  
St[i] = 0100 0100 0101 0101 0110 0110 0111 0111 0010 0010  
0000 0110 0111 0000 0110 1000  
St[0] ^ St[6] = 0011 0100  
St[3] ^ St[4] = 0101 0101  
St[2] ^ St[5] = 0110 0000  
St[1] ^ St[7] = 0011 0011  
Add\_1 = 0111 0111  
Add\_2 = 0010 1101  
Hasil Fungsi f = 0010 1111  
Feedback LFSR\_1 = 0100 0011

LFSR\_2

Bt[0] = 0001 0001  
Bt[i] = 0001 0001 0010 0010 0011 0011 0100 0100 0101 0101  
0110 1100 0000 1001 0111 1110  
Bt[0] ^ Bt[6] = 0110 1101  
Bt[3] ^ Bt[4] = 0011 1111  
Bt[2] ^ Bt[5] = 0100 1101  
Bt[1] ^ Bt[7] = 0010 0010  
Add\_3 = 0101 0100  
Add\_4 = 0110 1111  
Hasil Fungsi f = 0111 0110  
Feedback LFSR\_2 = 0010 1010

Key Stream ke-3 = 0101 0101

Shift ke- 4

LFSR\_1

St[0] = 0101 0101  
St[i] = 0101 0101 0110 0110 0111 0111 0010 0010 0000 0110  
0111 0000 0110 1000 0100 0011  
St[0] ^ St[6] = 0011 0011  
St[3] ^ St[4] = 0010 0100  
St[2] ^ St[5] = 0000 0111  
St[1] ^ St[7] = 0010 0101  
Add\_1 = 0000 1111  
Add\_2 = 0001 1110  
Hasil Fungsi f = 0010 0010  
Feedback LFSR\_1 = 0000 0100

LFSR\_2

Bt[0] = 0010 0010  
Bt[i] = 0010 0010 0011 0011 0100 0100 0101 0101 0110 1100  
0000 1001 0111 1110 0010 1010  
Bt[0] ^ Bt[6] = 0000 1100  
Bt[3] ^ Bt[4] = 0110 0011  
Bt[2] ^ Bt[5] = 0010 1001  
Bt[1] ^ Bt[7] = 0101 0101  
Add\_3 = 0110 1111  
Add\_4 = 0010 1100  
Hasil Fungsi f = 0010 1110  
Feedback LFSR\_2 = 0001 1000

Key Stream ke-4 = 0111 0111

Shift ke- 5

LFSR\_1

St[0] = 0110 0110

```

St[i]           = 0110 0110 0111 0111 0010 0010 0000 0110 0111 0000
  0110 1000 0100 0011 0000 0100
St[0] ^ St[6]  = 0010 0101
St[3] ^ St[4]  = 0111 0110
St[2] ^ St[5]  = 0100 0110
St[1] ^ St[7]  = 0111 0101
Add_1           = 0101 0001
Add_2           = 0100 0101
Hasil Fungsi f = 0010 1010
Feedback LFSR_1 = 0101 0010

```

```

LFSR_2
Bt[0]           = 0011 0011
Bt[i]           = 0011 0011 0100 0100 0101 0101 0110 1100 0000 1001
  0111 1110 0010 1010 0001 1000
Bt[0] ^ Bt[6]  = 0010 1011
Bt[3] ^ Bt[4]  = 0111 0101
Bt[2] ^ Bt[5]  = 0100 0010
Bt[1] ^ Bt[7]  = 0100 0100
Add_3           = 0100 1010
Add_4           = 0111 1010
Hasil Fungsi f = 0011 0110
Feedback LFSR_2 = 0100 0100

```

Key Stream ke-5 = 0101 0101

Shift ke- 6

```

LFSR_1
St[0]           = 0111 0111
St[i]           = 0111 0111 0010 0010 0000 0110 0111 0000 0110 1000
  0100 0011 0000 0100 0101 0010
St[0] ^ St[6]  = 0111 0101
St[3] ^ St[4]  = 0001 1000
St[2] ^ St[5]  = 0100 0101
St[1] ^ St[7]  = 0111 0000
Add_1           = 0111 0011
Add_2           = 0010 1011
Hasil Fungsi f = 0110 1110
Feedback LFSR_1 = 0111 1010

```

```

LFSR_2
Bt[0]           = 0100 0100
Bt[i]           = 0100 0100 0101 0101 0110 1100 0000 1001 0111 1110
  0010 1010 0001 1000 0100 0100
Bt[0] ^ Bt[6]  = 0000 0000
Bt[3] ^ Bt[4]  = 0101 0100
Bt[2] ^ Bt[5]  = 0001 0001
Bt[1] ^ Bt[7]  = 0111 1011
Add_3           = 0101 0100
Add_4           = 0110 1010
Hasil Fungsi f = 0111 0110
Feedback LFSR_2 = 0010 0100

```

Key Stream ke-6 = 0011 0011

Shift ke- 7

```

LFSR_1
St[0]           = 0010 0010
St[i]           = 0010 0010 0000 0110 0111 0000 0110 1000 0100 0011
  0000 0100 0101 0010 0111 1010
St[0] ^ St[6]  = 0111 0000
St[3] ^ St[4]  = 0010 0101
St[2] ^ St[5]  = 0111 0100

```

$St[1] \wedge St[7] = 0111\ 1100$   
 $Add\_1 = 0110\ 1011$   
 $Add\_2 = 0000\ 1000$   
 Hasil Fungsi f = 0101 0010  
 Feedback LFSR\_1 = 0111 0000

LFSR\_2  
 $Bt[0] = 0101\ 0101$   
 $Bt[i] = 0101\ 0101\ 0110\ 1100\ 0000\ 1001\ 0111\ 1110\ 0010\ 1010$   
 $0001\ 1000\ 0100\ 0100\ 0010\ 0100$   
 $Bt[0] \wedge Bt[6] = 0111\ 0111$   
 $Bt[3] \wedge Bt[4] = 0011\ 0010$   
 $Bt[2] \wedge Bt[5] = 0011\ 1010$   
 $Bt[1] \wedge Bt[7] = 0001\ 1000$   
 $Add\_3 = 0101\ 0111$   
 $Add\_4 = 0010\ 0010$   
 Hasil Fungsi f = 0001 0110  
 Feedback LFSR\_2 = 0001 0100

Key Stream ke-7 = 0111 0111

Shift ke- 8

LFSR\_1  
 $St[0] = 0000\ 0110$   
 $St[i] = 0000\ 0110\ 0111\ 0000\ 0110\ 1000\ 0100\ 0011\ 0000\ 0100$   
 $0101\ 0010\ 0111\ 1010\ 0111\ 0000$   
 $St[0] \wedge St[6] = 0111\ 1100$   
 $St[3] \wedge St[4] = 0100\ 0001$   
 $St[2] \wedge St[5] = 0011\ 0110$   
 $St[1] \wedge St[7] = 0010\ 0000$   
 $Add\_1 = 0100\ 0011$   
 $Add\_2 = 0101\ 0110$   
 Hasil Fungsi f = 0101 0111  
 Feedback LFSR\_1 = 0011 1010

LFSR\_2  
 $Bt[0] = 0110\ 1100$   
 $Bt[i] = 0110\ 1100\ 0000\ 1001\ 0111\ 1110\ 0010\ 1010\ 0001\ 1000$   
 $0100\ 0100\ 0010\ 0100\ 0001\ 0100$   
 $Bt[0] \wedge Bt[6] = 0111\ 1000$   
 $Bt[3] \wedge Bt[4] = 0101\ 1100$   
 $Bt[2] \wedge Bt[5] = 0000\ 1010$   
 $Bt[1] \wedge Bt[7] = 0110\ 1101$   
 $Add\_3 = 0010\ 1100$   
 $Add\_4 = 0110\ 0011$   
 Hasil Fungsi f = 0111 1010  
 Feedback LFSR\_2 = 0010 1110

Key Stream ke-8 = 0110 1110

Shift ke- 9

LFSR\_1  
 $St[0] = 0111\ 0000$   
 $St[i] = 0111\ 0000\ 0110\ 1000\ 0100\ 0011\ 0000\ 0100\ 0101\ 0010$   
 $0111\ 1010\ 0111\ 0000\ 0011\ 1010$   
 $St[0] \wedge St[6] = 0010\ 0000$   
 $St[3] \wedge St[4] = 0101\ 0010$   
 $St[2] \wedge St[5] = 0011\ 1001$   
 $St[1] \wedge St[7] = 0101\ 1110$   
 $Add\_1 = 0111\ 0010$   
 $Add\_2 = 0010\ 0101$   
 Hasil Fungsi f = 0011 0001

Feedback LFSR\_1 = 0001 0101

LFSR\_2

Bt[0] = 0000 1001  
Bt[i] = 0000 1001 0111 1110 0010 1010 0001 1000 0100 0100  
0010 0100 0001 0100 0010 1110  
Bt[0] ^ Bt[6] = 0010 0101  
Bt[3] ^ Bt[4] = 0110 1000  
Bt[2] ^ Bt[5] = 0000 1100  
Bt[1] ^ Bt[7] = 0011 1001  
Add\_3 = 0111 0011  
Add\_4 = 0100 0101  
Hasil Fungsi f = 0110 1110  
Feedback LFSR\_2 = 0101 1010

Key Stream ke-9 = 0111 1001

BUILD SUCCESSFUL (total time: 0 seconds)

## TEST VECTOR II INPUT EXTREME

**Test vector dengan 10 kali pergeseran LFSR**

init:

deps-jar:

Compiling 1 source file to F:\Documents and Settings\pemakai1\My Documents\NetBeansProjects\streamc4\build\classes

compile:

run:

Shift ke- 0

LFSR\_1

St[0] = 0000 0000  
St[i] = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000  
St[0] ^ St[6] = 0000 0000  
St[3] ^ St[4] = 0000 0000  
St[2] ^ St[5] = 0000 0000  
St[1] ^ St[7] = 0000 0000  
Add\_1 = 0000 0000  
Add\_2 = 0000 0000  
Hasil Fungsi f = 0000 0000  
Feedback LFSR\_1 = 0000 0000

LFSR\_2

Bt[0] = 0000 0000  
Bt[i] = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000  
Bt[0] ^ Bt[6] = 0000 0000  
Bt[3] ^ Bt[4] = 0000 0000  
Bt[2] ^ Bt[5] = 0000 0000  
Bt[1] ^ Bt[7] = 0000 0000  
Add\_3 = 0000 0000  
Add\_4 = 0000 0000  
Hasil Fungsi f = 0000 0000  
Feedback LFSR\_2 = 0000 0000

Key Stream ke-0 = 0000 0000

Shift ke- 1

LFSR\_1

St[0] = 0000 0000

```

St[i]          = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  0000 0000 0000 0000 0000 0000 0000
St[0] ^ St[6] = 0000 0000
St[3] ^ St[4] = 0000 0000
St[2] ^ St[5] = 0000 0000
St[1] ^ St[7] = 0000 0000
Add_1          = 0000 0000
Add_2          = 0000 0000
Hasil Fungsi f = 0000 0000
Feedback LFSR_1 = 0000 0000

```

```

LFSR_2
Bt[0]          = 0000 0000
Bt[i]          = 0000 0000 0000 0000 0000 0000 0000 0000 0000
  0000 0000 0000 0000 0000 0000
Bt[0] ^ Bt[6] = 0000 0000
Bt[3] ^ Bt[4] = 0000 0000
Bt[2] ^ Bt[5] = 0000 0000
Bt[1] ^ Bt[7] = 0000 0000
Add_3          = 0000 0000
Add_4          = 0000 0000
Hasil Fungsi f = 0000 0000
Feedback LFSR_2 = 0000 0000

```

Key Stream ke-1 = 0000 0000

Shift ke- 2

```

LFSR_1
St[0]          = 0000 0000
St[i]          = 0000 0000 0000 0000 0000 0000 0000 0000 0000
  0000 0000 0000 0000 0000 0000
St[0] ^ St[6] = 0000 0000
St[3] ^ St[4] = 0000 0000
St[2] ^ St[5] = 0000 0000
St[1] ^ St[7] = 0000 0000
Add_1          = 0000 0000
Add_2          = 0000 0000
Hasil Fungsi f = 0000 0000
Feedback LFSR_1 = 0000 0000

```

```

LFSR_2
Bt[0]          = 0000 0000
Bt[i]          = 0000 0000 0000 0000 0000 0000 0000 0000 0000
  0000 0000 0000 0000 0000 0000
Bt[0] ^ Bt[6] = 0000 0000
Bt[3] ^ Bt[4] = 0000 0000
Bt[2] ^ Bt[5] = 0000 0000
Bt[1] ^ Bt[7] = 0000 0000
Add_3          = 0000 0000
Add_4          = 0000 0000
Hasil Fungsi f = 0000 0000
Feedback LFSR_2 = 0000 0000

```

Key Stream ke-2 = 0000 0000

Shift ke- 3

```

LFSR_1
St[0]          = 0000 0000
St[i]          = 0000 0000 0000 0000 0000 0000 0000 0000 0000
  0000 0000 0000 0000 0000 0000
St[0] ^ St[6] = 0000 0000
St[3] ^ St[4] = 0000 0000
St[2] ^ St[5] = 0000 0000

```

St[1] ^ St[7] = 0000 0000  
Add\_1 = 0000 0000  
Add\_2 = 0000 0000  
Hasil Fungsi f = 0000 0000  
Feedback LFSR\_1 = 0000 0000

LFSR\_2  
Bt[0] = 0000 0000  
Bt[i] = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000  
Bt[0] ^ Bt[6] = 0000 0000  
Bt[3] ^ Bt[4] = 0000 0000  
Bt[2] ^ Bt[5] = 0000 0000  
Bt[1] ^ Bt[7] = 0000 0000  
Add\_3 = 0000 0000  
Add\_4 = 0000 0000  
Hasil Fungsi f = 0000 0000  
Feedback LFSR\_2 = 0000 0000

Key Stream ke-3 = 0000 0000

Shift ke- 4

LFSR\_1  
St[0] = 0000 0000  
St[i] = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000  
St[0] ^ St[6] = 0000 0000  
St[3] ^ St[4] = 0000 0000  
St[2] ^ St[5] = 0000 0000  
St[1] ^ St[7] = 0000 0000  
Add\_1 = 0000 0000  
Add\_2 = 0000 0000  
Hasil Fungsi f = 0000 0000  
Feedback LFSR\_1 = 0000 0000

LFSR\_2  
Bt[0] = 0000 0000  
Bt[i] = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000  
Bt[0] ^ Bt[6] = 0000 0000  
Bt[3] ^ Bt[4] = 0000 0000  
Bt[2] ^ Bt[5] = 0000 0000  
Bt[1] ^ Bt[7] = 0000 0000  
Add\_3 = 0000 0000  
Add\_4 = 0000 0000  
Hasil Fungsi f = 0000 0000  
Feedback LFSR\_2 = 0000 0000

Key Stream ke-4 = 0000 0000

Shift ke- 5

LFSR\_1  
St[0] = 0000 0000  
St[i] = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000  
St[0] ^ St[6] = 0000 0000  
St[3] ^ St[4] = 0000 0000  
St[2] ^ St[5] = 0000 0000  
St[1] ^ St[7] = 0000 0000  
Add\_1 = 0000 0000  
Add\_2 = 0000 0000  
Hasil Fungsi f = 0000 0000  
Feedback LFSR\_1 = 0000 0000



Bt[0] ^ Bt[6] = 0000 0000  
 Bt[3] ^ Bt[4] = 0000 0000  
 Bt[2] ^ Bt[5] = 0000 0000  
 Bt[1] ^ Bt[7] = 0000 0000  
 Add\_3 = 0000 0000  
 Add\_4 = 0000 0000  
 Hasil Fungsi f = 0000 0000  
 Feedback LFSR\_2 = 0000 0000

Key Stream ke-7 = 0000 0000

Shift ke- 8

LFSR\_1  
 St[0] = 0000 0000  
 St[i] = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  
     0000 0000 0000 0000 0000 0000  
 St[0] ^ St[6] = 0000 0000  
 St[3] ^ St[4] = 0000 0000  
 St[2] ^ St[5] = 0000 0000  
 St[1] ^ St[7] = 0000 0000  
 Add\_1 = 0000 0000  
 Add\_2 = 0000 0000  
 Hasil Fungsi f = 0000 0000  
 Feedback LFSR\_1 = 0000 0000

LFSR\_2  
 Bt[0] = 0000 0000  
 Bt[i] = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  
     0000 0000 0000 0000 0000 0000  
 Bt[0] ^ Bt[6] = 0000 0000  
 Bt[3] ^ Bt[4] = 0000 0000  
 Bt[2] ^ Bt[5] = 0000 0000  
 Bt[1] ^ Bt[7] = 0000 0000  
 Add\_3 = 0000 0000  
 Add\_4 = 0000 0000  
 Hasil Fungsi f = 0000 0000  
 Feedback LFSR\_2 = 0000 0000

Key Stream ke-8 = 0000 0000

Shift ke- 9

LFSR\_1  
 St[0] = 0000 0000  
 St[i] = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  
     0000 0000 0000 0000 0000 0000  
 St[0] ^ St[6] = 0000 0000  
 St[3] ^ St[4] = 0000 0000  
 St[2] ^ St[5] = 0000 0000  
 St[1] ^ St[7] = 0000 0000  
 Add\_1 = 0000 0000  
 Add\_2 = 0000 0000  
 Hasil Fungsi f = 0000 0000  
 Feedback LFSR\_1 = 0000 0000

LFSR\_2  
 Bt[0] = 0000 0000  
 Bt[i] = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  
     0000 0000 0000 0000 0000 0000  
 Bt[0] ^ Bt[6] = 0000 0000  
 Bt[3] ^ Bt[4] = 0000 0000  
 Bt[2] ^ Bt[5] = 0000 0000  
 Bt[1] ^ Bt[7] = 0000 0000

Add\_3 = 0000 0000  
Add\_4 = 0000 0000  
Hasil Fungsi f = 0000 0000  
Feedback LFSR\_2 = 0000 0000

Key Stream ke-9 = 0000 0000

BUILD SUCCESSFUL (total time: 0 seconds)