Implementasi algortima GOST dengan Python

Muhammad Gema Akbar (13510099)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
mgemaakbar@itb.ac.id

***Abstract*—Algoritma GOST adalah salah satu algortima kriptografi jenis block cipher, ditemukan oleh pemerintah Russia dan Soviet. Makalah ini berisi penjelasan tentang algoritma GOST, bagaimana cara kerjanya dan hal yang paling utama adalah bagaimana algortima GOST mungkin diimplementasikan dalam bahasa Python.**

***Index Terms*—algoritma, block cipher, cipher, dekripsi, enkripsi, feistel cipher,implementasi, kriptografi, kunci simetris, python.**

I. PENDAHULUAN

GOST adalah salah satu algoritma *block cipher* kunci simetris, diciptakan oleh pemerintah Russia dan Soviet. GOST selain juga menjadi algoritma enkripsi, GOST menjadi dasar dari algoritma fungsi hash GOST. Algoritma GOST ini diimplementasikan dalam OpenSSL dan *library* kriptografi lainnya, digunakan oleh bank-bank di Russia. [1]

Secara struktural, GOST memiliki kemiripan dengan algoritma block cipher lainnya yaitu DES (*Data Encryption Standard*). GOST termasuk algortima *block cipher* dengan tipe perancangan *Feistel cipher* GOST memiliki ukuran block sepanjang 64-bit dan panjang kunci 256 bit. GOST memiliki S-Box yang dapat memiliki

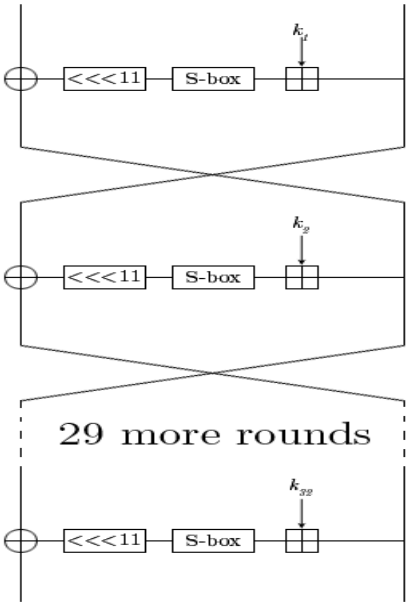


Fig0. Diagram GOST

isi sekitar 354 bit informasi, yang membuat ukuran kunci dapat bertambah menjadi 610 bit.

Pada dasarnya GOST tergolong Feistel *cipher*, memiliki 32 round. Fungsi round dari GOST adalah penambahan *subkey* module 2^32, dan taruh hasilnya di layer S-Boxes, dan rotasi hasil ke kiri sebanyak 11 bit. Hasil dari itu adalah output dari fungsi round. Satu baris adalah 32 bit.[2]

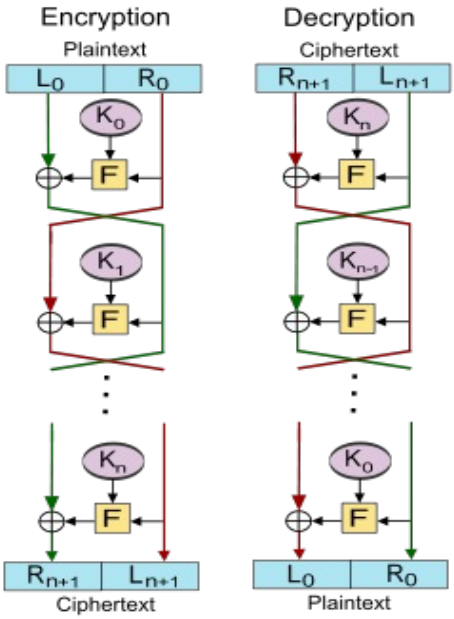


Fig 02 Diagram feistel cipher

Subkey, atau bagian dari kunci di tentukan dalam urutan yang telah kita tentukan, urutan kunci dapat dilakukan speerti ini: pecah kunci 256 bit menjadi 8 32-bit subkey, dan setiap kunci digunakan sebanyak 4 kali dalam algortima. 24 round pertama menggunakan subkunci secara berurutan, 8 round terakhir menggunakan dalam urutan yang terbalik.[3]

S-box menerima input 4 bit dan menghasilkan output 4

bit output. Substitusi S-box dalam fungsi round terdiri dari 8 s-box 4x4. S-box tergantung dengan implementasi, orang yang ingin merahasiakan komunikasi menggunakan GOST harus menggunakan s-box yang sama. Untuk keamanan tambahan S-box dapat dirahasiakan. Dalam standar asli dimana GOST ditentukan tidak ada s-box yang diberikan. Contoh S-box yang akan digunakan dalam program (digunakan Central Bank of Russian Federation) [4]

1	4 10 9 2 13 8 0 14 6 11 1 12 7 15 5 3
2	14 11 4 12 6 13 15 10 2 3 8 1 0 7 5 9
3	5 8 1 13 10 3 4 2 14 15 12 7 6 0 9 11
4	7 13 10 1 0 8 9 15 14 4 6 12 11 2 5 3
5	6 12 7 1 5 15 13 8 4 10 9 14 0 3 11 2
6	4 11 10 0 7 2 1 13 3 6 8 5 9 12 15 14
7	13 11 4 1 3 15 5 9 0 10 14 7 6 8 2 12
8	1 15 13 0 5 7 10 4 9 2 3 14 6 11 8 12

II. IMPLEMENTASI

Bagian ini akan membahas detail dari implementasi yang dilakukan dari algoritma GOST *cipher*. Implementasi dilakukan tanpa menggunakan *library* kriptografi sama sekali, hanya menggunan operasi dan tipe data yang secara *default* ada dalam Python. Walaupun telah disebutkan bahwa algortima GOST memiliki kemiripan dengan algoritma DES yang sama-sama *block cipher,* ternyata implementasi algortima GOST dapat dibuat dengan cukup sederhana, terbukti karena sudah dibandingkan dengan beberapa implementasi algoritma DES dalam Python yang dapat mencapai kurang lebih 400 baris kode.

```
sbox = (
    (4, 10, 9, 2, 13, 8, 0, 14, 6, 11,
1, 12, 7, 15, 5, 3),
    (14, 11, 4, 12, 6, 13, 15, 10, 2,
3, 8, 1, 0, 7, 5, 9),
    (5, 8, 1, 13, 10, 3, 4, 2, 14, 15,
12, 7, 6, 0, 9, 11),
    (7, 13, 10, 1, 0, 8, 9, 15, 14, 4,
6, 12, 11, 2, 5, 3),
    (6, 12, 7, 1, 5, 15, 13, 8, 4, 10,
9, 14, 0, 3, 11, 2),
    (4, 11, 10, 0, 7, 2, 1, 13, 3, 6,
8, 5, 9, 12, 15, 14),
    (13, 11, 4, 1, 3, 15, 5, 9, 0, 10,
14, 7, 6, 8, 2, 12),
    (1, 15, 13, 0, 5, 7, 10, 4, 9, 2,
3, 14, 6, 11, 8, 12),
)
```

Fig. 1. Kode yang berisi array s-box

Mula-mula kita mengisi suatu *array of integer array* (senarai yang berisi senarai bilangan bulat) yang berisi S-box. S-box yang digunakan dalam program implementasi algoritma GOST ini adalah S-box yang digunakan oleh Cenral Bank of Russian Federation.

Dalam Fig.1 yang dicantumkan diatas, indeks pertama dari senarai adalah senarai **(4, 10, 9, 2, 13, 8, 0, 14, 6, 11, 1, 12, 7, 15, 5, 3).** Dalam elemen senarai ini, angka 4 berada pada indeks 0, angka 10 berada pada indeks 1, dan seterusnya. Maka, untuk mengakses angka 4, cara pengaksesannya adalah dengan mengacu pada `sbox[0][0]`, jika kita ingin mengakses angka 10 maka kita harus mengacu `sbox[0][1]`.

```
def panjangBit(x):
    assert x >= 0
    return len(bin(x)) - 2
```

Fig 2. Kode fungsi untuk menghitung panjang bilangan biner

Kita membuat fungsi yang digunakan untuk menghitung panjang dari bilangan biner. Bilangan tersebut harus bernilai positif. Fungsi ini akan digunakan dalam asersi dalam fungsi lain.

```
def fungsiF(input, key):
    assert panjangBit(input) <= 32
    assert panjangBit(key) <= 32

    temp = input ^ key

    output = 0
    for i in range(8):
        output |= ((sbox[i][(temp >>
(4 * i)) & 0b1111]) << (4 * i))

    a=(output>>11)
    b=(output<<21)
          c=(a|b)
    output = (c & 0xFFFFFFFF)

    return output
```

Fig 3. Kode fungsi FungsiF

Fungsi fungsiF akan digunakan dalam fungsi-fungsi lain, yang akan dibahas nanti. FungsiF menerima input kunci dan input teks. Panjang input diasersikan tidak akan lebih dari 32. Panjang kunci juga diaserksikan tidak akan lebih dari 32, keduanya menggunakan fungsi yang telah dibuat di awal.

```
temp = input ^ key
```

Pertama-tama input teks ke fungsiF akan di lakukan operasi And dengan kunci yang digunakan untuk algortima GOST cipher, lalu simpan dalam suatu variabel, dalam program yang dibuat nama variabel tersebut adalah 'temp'.

```
for i in range(8):
        output |= ((sbox[i][(temp >>
(4 * i)) & 0b1111]) << (4 * i))
```

Lalu dalam program digunakan *loop* sebanyak 8 kali. *Loop* akan melakukan operasi logika OR terhadap suatu variable yang nantinya akan menjadi nilai kembalian FungsiF, dalam program ini nama variabel tersebut adalah 'output'.

```
(temp >> (4 * i)) & 0b1111) = x
```

Dalam beberapa paragraf, hasil operasi diatas akan disebut sebagai x demi kemudahan dalam pengacuan. Operasi yang dilakukan adalah *shift* kanan sebanyak 4*i, dimana I adalah variabel indeks yang menyatakan sudah loop keberapa, lalu hasilnya dilakukan operasi AND terhadap angka 1111 (angka basis biner), lalu hasilnya dijadikan indeks untuk mengacu elemen s-box. X akan digunakan untuk mengacu elemen horizontal dari sbox.

```
(temp >> (4 * i)) & 0b1111]) << (4 *
I) == y
```

Hasil dari operasi logika diatas akan disebut sebagai 'y' demi kemudahan daalam pengacuan. Sbox[i][X] dilakukan pergeseran ke kiri sebanyak I*4.

```
output |= ((sbox[i][(temp >> (4 * i))
& 0b1111]) << (4 * I))
```

Y akan dilakukan operasi OR dengan variabel 'output', dan operasi OR ini dilakukan sebanyak 8 kali terhadap variabel 'output', sesuai dengan banyaknya pengulangan *loop*.

```
output = ((output >> 11) | (output <<
(32 - 11))) & 0xFFFFFFFF
```

Setelah *loop*, nilai variabel 'output' menjadi nilai kembalian dari fungsi FungsiF.

```
def enkripsiRound(inputKiri,
inputKanan, kunciRound):
    outputKiri = inputKanan
    outputKanan = inputKiri ^
FungsiF(inputKanan, kunciRound)

    return outputKiri, outputKanan
```

Fig 4. Fungsi enkripsiRound

Fungsi enkripsiRound menerima 3 parameter, inputKiri, inputKanan, kunciRound. Fungsi enkripsiRound berguna untuk menukar inputKiri dengan inputKanan, tapi sebelumnya inputKiri dilakukan operasi AND terlebih dahulu dengan hasil dari FungsiF, FungsiF menerima input inputKanan dan kunciRound. Operasi tersebut dilakukan oleh baris kode

```
outputKanan = inputKiri ^
FungsiF(inputKanan, kunciRound)
```

Selanjutnya fungsi enkripsiRoudn mengembalikan *tuple* outputKiri dan outputKanan. Fungsi enkripsiRound ini akan digunakan dalam fungsi utama untuk enkripsi.

```
def dekripsiRound(inputKiri,
inputKanan, kunciRound):
    outputKanan = inputKiri
    outputKiri = inputKanan ^
fungsiF(inputKiri, kunciRound)

    return outputKiri, outputKanan
```

Fig. 5 Kode fungsi dekripsiRound

Fungsi dekripsiRound menerima 3 input, inputKiri, inputKanan, dan kunciRound. Fungsi dekripsiRound ini pada dasarnya adalah kebalikan dari fungsi enkripsiRound, dan fungsi dekripsiRound akan digunakan dalam fungsi dekripsi utama. Fungsi dekripsiRound akan mengembalikan tuple outputKiri dan outputKanan, outputKanan adalah inputKiri, sementara outputKiri adalah operasi AND dari inputKanan dan output fungsiF yang menerima parameter inputKiri dan kunciRound sebagai kuncinya.

```
def setKey(self, master_key):
        assert panjangBit(master_key)
<= 256
        for i in range(8):
            self.master_key[i] =
(master_key >> (32 * i)) & 0xFFFFFFFF
```

Fig 6. Fungsi setKey

Fungsi setKey menerima input master_key, yaitu kunci yang sebenarnya adalah kunci masukan pengguna. Kunci masukan pengguna diasersikan panjangnya tidak lebih dari 256. Lalu per byte master_key dilakukan *shift* kanan sebanyak 32*i, dimana I adalah indeks *loop*, *loop* dilakukan sebanyak 8 kali.

```
    def encrypt(self, plaintext):

        assert panjangBit(plaintext)
<= 64
        teksKiri = plaintext >> 32
        teksKanan = plaintext &
0xFFFFFFFF

        for i in range(24):
            teksKiri, teksKanan =
enkripsiRound(teksKiri, teksKanan,
self.master_key[i % 8])

        for i in range(8):
            teksKiri, teksKanan =
enkripsiRound(teksKiri, teksKanan,
self.master_key[7 - i])

        return (teksKiri << 32) |
teksKanan
```

Fig 7. Kode fungsi utama enkripsi (encrypt)

Fungsi encrypt adalah fungsi utama enkripsi dari implementasi algoritma GOST. Fungsi ini menerima masukan plainteks, dan juga kunci. Panjang plainteks yang diterima fungsi ini diasersikan tidak lebih dari 64 bit panjangnya.

Dalam fungsi ini plainteks dibagi menjadi dua, dua bagian tersebut adalah variabel teksKiri dan teksKanan. TeksKiri adalah hasil *shift* kanan 32 dari plainteks masukan, teksKanan adalah hasil operasi AND plainteks dan 0xFFFFFFFF.

Selanjutnya dilakukan *loop* sebanyak 24 kali yang berisi pemanggilan fungsi enkripsiRound. Fungsi enkripsiRound menerima input teksKiri, teksKanan dan 8 bit bagian dari master_key. Elemen senarai master_key yang diacu adalah master_key[i%8] dimana I adalah indeks dari *loop*. Ini loopnya:

```
for i in range(24):
            teksKiri, teksKanan =
enkripsiRound(teksKiri, teksKanan,
self.master_key[i % 8])
```

Selanjutnya dilakukan *loop* sebanyak 8 kali yang berisi pemanggilan fungsi enkripsiRound. Di loop kali ini fungsi enkripsiRound menerima input teksKiri dan teksKanan, bedanya adalah parameter roundKey menerima input elemen master_key[7-i] dimana I adalah indeks dari *loop*.

Berikut adalah loop yang dimaksud:

```
for i in range(8):
            teksKiri, teksKanan =
enkripsiRound(teksKiri, teksKanan,
self.master_key[7 - i])
```

Diakhir fungsi dikembalikan `(teksKiri << 32) | teksKanan`. Hasil teksKiri di *shift* kiri sebanyak 32 kali., lalu hasilnya dioperasikan dengan operand AND.

```
def decrypt(self, ciphertext):

        assert panjangBit(ciphertext)
<= 64
        teksKiri = ciphertext >> 32
        teksKanan = ciphertext &
0xFFFFFFFF

        for i in range(8):
            teksKiri, teksKanan =
dekripsiRound(teksKiri, teksKanan,
self.master_key[i])

        for i in range(24):
            teksKiri, teksKanan =
dekripsiRound(teksKiri, teksKanan,
self.master_key[(7 - i) % 8])

        return (teksKiri << 32) |
teksKanan
```

Fig 8. Fungsi dekripsi utama (decrypt)

Fungsi dekripsi utama dari implementasi algortima GOST pada dasarnya adalah kebalikan atau inverse dari fungsi utama enkripsi. Didalamnya dimanfaatkan fungsi dekripsiRound yang telah dibuat sebelumnya. Fungsi dekripsi utama menerima teks yang telah dienkripsi, dan kunci masukan pengguna program.

Fungsi dekripsi utama melakukan pembagian terhadap teks yang telah dienkripsi menjadi dua bagian, dua bagian tersebut dinyatakan dalam dua variabel teksKiri dan teksKanan. TeksKiri adalah *shift* kanan sebanyak 32 kali dari teks terenkripsi, teksKanan adalah operasi AND dari teks terenkripsi dan 0xFFFFFFFF.

Selanjutnya dibuat loop yang melakukan pemanggilan fungsi dekripsiRound. Fungsi dekripsiRound menerima input teks terenkripsi yang telah dibagi 2 (teksKiri dan teksKanan) dan kunci utama master_key. Elemen senarai master_key yang diacu adalah master_key[i] dimana I adalah indeks dari *senarai*. Loop ini pada dasarnya adalah kebalikan atau inverse dari loop dalam fungsi utama enkripsi yang dilakukan sebanyak 8 kali juga dan mengakses elemen senarai master_key[7-i]. Kembalian dari fungsi dekripsiRound nilainya dimasukan ke tuple teksKiri dan teksKanan. Ini loopnya:

```
for i in range(8):
            teksKiri, teksKanan =
dekripsiRound(teksKiri, teksKanan,
self.master_key[i])
```

Selanjutnya dibuat loop yang melakukan fungsi dekripsiRound, banyaknya pengulangan yang dilakukan adalah 24 kali. Loop ini pada dasarnya adalah kebalikan atau inverse dari loop yang digunakan dalam fungsi utama enkripsi. Dalam loop ini, fungsi dekripsiRound menerima input teksKiri, teksKanan hasil loop sebelumnya, dan juga menerima input kunci (elemen senarai yang diacu adalah master_key[(7-i)]%8). Terlihat bahwa indeks elemen senarai yang diakses/diacu adalah inverse atau kebalikan dari loop yang ada di fungsi utama enkripsi. Nilai kembalian dari hasil pemanggilan fungsi dekripsiRound dalam loop disimpan dalam tuple teksKiri,teksKanan secara berurutan. Ini loopnya:

```
for i in range(24):
            teksKiri, teksKanan =
dekripsiRound(teksKiri, teksKanan,
self.master_key[(7 - i) % 8])
```

Nilai kembalian terakhir dari fungsi utama dekripsi adalah `(teksKiri << 32) | teksKanan` yaitu shift kiri 32 bit teksKiri dan hasilnya dilakukan operasi or dengan teksKanan

II. Pengujian Program

Dalam bagian ini akan dilakukan pengujian terhadap program implementasi algoritma GOST. Kunci yang menjadi masukan algortima adalah 0x1111222233334444555566667777888899990000aaaa bbbbccccddddeeeeffff dan teks yang akan dienkripsi adalah 0xabcdef0987654321.

Source code yang digunakan untuk pengujian :

```
text = 0xabcdef0987654321
key =
0x1111222233334444555566667777888899999
0000aaaabbbbccccddddeeeeffff

    my_GOST = GOST()
    my_GOST.set_key(key)

    num = 1000

    for i in range(num):
        text = my_GOST.encrypt(text)
    print hex(text)

    for i in range(num):
        text = my_GOST.decrypt(text)
    print hex(text)
```

Hasil pengujian membuktikan bahwa program berjalan dengan baik

Hasil enkripsi :0xe3108973d8c65d13L
Hasil dekripsi: 0xabcdef0987654321L

```
C:\Users\anon\Desktop>python gost.py
0xe3108973d8c65d13L
0xabcdef0987654321L
```

Fig 09 Screenshot hasil pengujian/

III. Kesimpulan

Algortima GOST adalah salah satu algoritma block cipher, memiliki kemiripan dengan block cipher lainnya yaitu DES (Data Encryption Standard) . Walaupun GOST memiliki kemiripan dengan algortima lain, seperti DES,

dalam makalah ini terbukti lebih sedikit kode yang digunakan dibandingkan implementasi DES. Lalu algoritma GOST mungkin diimplementasikan dalam bahasa python dengan cara yang cukup sederhana.

REFERENCES

[1] Martin Albrecht: Algebraic Attacks against the Courtois Toy Cipher, In Cryptologia, Vol.32 Iss. 3 July 2008, ppp.220-276

[2] Martin Albrecht and Gregor Leander: An All-inOne Approach to Differential Cryptanalysis for Small Block Ciphers, pre print available at ecprint.iacr.org/2012/401

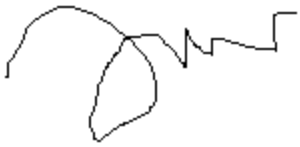
[3] Ludmila K. Babenko, Evgeniya Ischukova, Ekaterina Maro: Research about strength of GOST 28147-89 encryption algortithm

[4] Ludmila K. Babenko, Evgeniya Ischukova, Differential analysis GOST encryption algorithm, In SIN 2010, pp. 149-157, ACM, 2010

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Maret 2014



ttd

Muhammad Gema Akbar 13510099