PEMANFAATAN METODE GOST UNTUK ENKRIPSI DAN DEKRIPSI FILE

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar
Sarjana dalam bidang Ilmu Komputer

oleh:
RIA TANAYA
0310960064-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA

2008


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

PEMANFAATAN METODE GOST
UNTUK ENKRIPSI DAN DEKRIPSI FILE

Oleh:
RIA TANAYA
0310960064-96

Setelah dipertahankan di depan Majelis Penguji
Pada tanggal
dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana dalam bidang Ilmu Komputer

Pembimbing I	Pembimbing II
Drs. Achmad Ridok,M.Kom	**Bayu Rahayudi,ST.,MT**
NIP. 131 090 392	**NIP. 132 318 424**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Agus Suryanto, MSc.
NIP. 132 126 049


UNIVERSITAS BRAWIJAYA

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	**: Ria Tanaya**
NIM	**: 0310960064**
Jurusan	**: Matematika**
Penulis tugas Akhir berjudul	**: Pemanfaatan Metode GOST Untuk Enkripsi dan Dekripsi File**

Dengan ini menyatakan bahwa :

1. **Isi dari skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam skripsi ini.**
2. **Apabila dikemudian hari ternyata skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang,
Yang menyatakan,

(Ria Tanaya)
NIM. 0310960064

PEMANFAATAN METODE GOST
UNTUK ENKRIPSI DAN DEKRIPSI FILE

ABSTRAK

Data dan informasi merupakan sumber daya yang sangat berharga sehingga perlu dijaga keamanan dan kerahasiaan datanya. Kriptografi muncul sebagai sebuah ilmu yang menawarkan beragam metode dalam rangka untuk menjaga keamanan data. Pemilihan metode yang tepat akan sangat menentukan tercapainya tujuan keamanan. Pada algoritma kriptografi modern, kerahasiaan data berasal dari adanya algoritma yang kuat dan dipublikasikan dengan kunci yang panjang. Dapat dikatakan bahwa metode dengan rentang penggunaan kunci yang panjang dan algoritma yang kuat akan bersifat lebih aman. Metode *GOST* muncul sebagai sebuah algoritma kriptografi yang menawarkan keamanan data yang cukup handal dengan jumlah bit kunci sebanyak 256 bit dan jumlah putaran sebanyak 32 *round* (putaran). Metode *GOST* menggunakan 64-bit *block cipher* , delapan buah *S-Box*, dan operasi *XOR* serta *Rotate Left Shift.* Penggunaan metode *GOST* pada awalnya ditujukan untuk menangani keamanan sebuah data teks, namun dengan kemampuan yang dimiliki metode *GOST* dirasa mampu untuk menangani proses keamanan data secara umum. Pada penelitian ini akan dibangun sebuah perangkat lunak untuk mengimplementasikan kemampuan metode *GOST* dalam menangani keamanan data. Selain itu sebuah analisis juga akan dilakukan untuk mengetahui keefektifan waktu proses, konsistensi ukuran *file*, dan menghitung prosentase *avalanche effect* untuk mengetahui ketahanan dari metode *GOST*. Dari uji coba yang telah dilakukan diperoleh waktu proses enkripsi mencapai 1092,20 *byte* per detik pada data sebesar 5461 *byte* dan waktu proses dekripsi mencapai 915,43 *byte* per detik pada data sebesar 6400 *byte*. Ketahanan algoritma *GOST* cukup handal, dengan prosentase *avalanche effect* pengubahan kedua hampir mencapai 100%.

PEMANFAATAN METODE GOST
UNTUK ENKRIPSI DAN DEKRIPSI FILE

ABSTRACT

Data and information is the most valuable resource that required protecting, both data secret and its security. Cryptography emerged as a science that offering a lot of method to protect data. Choosing one or more method will determine how secure our data kept and how success the security target. At modern cryptography algorithm, data security can be reached from a great algorithm which publicized with a long key. It equals that a long key and a great algorithm will provide a high security level. GOST (Gosudarstvennyi Standard) as an encryption algorithm, provide a level of security information with 256 bit key and 32 rotations. GOST have 64-bit block cipher, eight S-Box, XOR operation and Rotate Left Shift. Initially, usages of GOST are for handle data text security, but later GOST felt able to handle data security process in general. This research will create software to implement the ability of GOST in handling data security. Some analysis will be conducted to know effectiveness of time process, file size consistency, and percentage of avalanche effect as a characteristic of strength ability. The result of analysis obtained some conclusions. Time process for encryption up to 1092,20 byte per second at 5461 byte data, and time process for decryption up to 915,43 byte per second at 6400 byte data. Percentage of avalanche effect up to 100% at a second change.

Kata Pengantar

Alhamdulillahi rabbil 'alamin. Puji syukur penulis panjatkan kehadirat Allah SWT, karena atas segala rahmat dan limpahan hidayahnya, skripsi yang berjudul "*Pemanfaatan Metode GOST Untuk Enkripsi dan Dekripsi File*" ini dapat berjalan dengan baik. Skripsi ini disusun dan diajukan sebagai syarat untuk memperoleh gelar sarjana pada program studi Ilmu Komputer, jurusan Matematika, fakultas MIPA, universitas Brawijaya.

Dalam penyelesaian tugas akhir ini, penulis telah mendapat begitu banyak bantuan baik moral maupun materiil dari banyak pihak. Atas bantuan yang telah diberikan, penulis ingin menyampaikan penghargaan dan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Bapak Drs. Achmad Ridok,M.Kom dan Bapak Bayu Rahayudi,ST.,MT selaku pembimbing. Terima kasih atas segala saran, bantuan, kearifan, waktu dan bimbingannya.
2. Bapak Wayan Firdaus Mahmudy, SSi., MT selaku Ketua Program Studi Ilmu Komputer.
3. Bapak Nurul Hidayat,S.Pd.,MSc selaku Penasihat Akademik.
4. Bapak Dr. Agus Suryanto,MSc selaku Ketua Jurusan Matematika.
5. Segenap bapak dan ibu dosen yang telah memberikan ilmunya kepada penulis.
6. Segenap staf dan karyawan di Jurusan Matematika FMIPA Universitas Brawijaya.
7. Bapak, Ibu, kakak dan adik. Terima kasih atas doa, dukungan dan semangat yang tiada henti.
8. Hendrik Wijaya, S.Kom terima kasih atas bimbingan dan kesabarannya.
9. Sahabat-sahabat terdekat Phepi, Ulan, Rio, Boim dan ilkomers `03, terima kasih atas semua bantuan yang telah diberikan.
10. Warga Watu Gong 31, tempat seribu satu hal baru.
11. Warga Tapak Jalak 5. Tempat dua ribu dua hal yang baru.
12. Pihak lain yang telah membantu terselesaikannnya Skripsi ini yang tidak bisa penulis sebutkan satu-persatu.

Semoga penulisan laporan ini bermanfaat bagi pembaca sekalian. Penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan, dan mengandung banyak kekurangan, sehingga dengan segala kerendahan hati penulis mengharapkan kritik dan saran yang membangun dari pembaca.

Malang, 21 Juli 2008

Penulis

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

BAB I
PENDAHULUAN

1.1 Latar Belakang

Data dan informasi merupakan sumber daya yang sangat berharga, sehingga perlu dijaga keamanan dan kerahasiaan datanya. Ilmu yang mempelajari tentang pengamanan data atau informasi disebut sebagai kriptografi (Rinaldi, 2006). Terdapat beragam metode yang digunakan dalam kriptografi. Setiap metode memiliki kelebihan dan kekurangan masing-masing. *User* dapat memilih metode kriptografi yang sesuai dengan kebutuhan dan tujuan yang ingin dicapai. Diantara banyak pilihan yang ditawarkan, akan terdapat satu atau beberapa metode yang paling banyak digunakan. Dapat dikatakan metode-metode ini dianggap oleh *user* sebagai metode terbaik, sehingga dengan menggunakan metode ini kebutuhan keamanan data akan tercapai.

Data Encryption Standard (*DES*) merupakan algoritma kriptografi yang paling banyak digunakan dari awal dipublikasikan pada tahun 1975 sampai menjadi sebuah algoritma standart pada tahun 1977 (Rinaldi, 2006). Keunggulan yang dimiliki oleh *DES* terletak pada jumlah kunci sebesar 56 *bit* dan putaran sebanyak 16 *round*. Dengan kemampuan yang dimilikinya pada waktu itu, *DES* dianggap sebagai algoritma kriptografi yang cukup handal. Namun seiring dengan berkembangnya teknologi, *DES* tidak lagi dipandang cukup kuat untuk menangani keamanan data. Kedudukan *DES* digantikan oleh *Rijndael*, yang terpilih sebagai algoritma standart AES *(Advanced Encryption Standard)* pada tahun 2001. Kemampuan *Rijndael* lebih baik pada jumlah kunci yang fleksibel yaitu 128, 196, 256 *bit* dan dengan jumlah putaran 10 *round* (Kurniawan, 2003).

Salah satu algoritma yang muncul seiring dengan perkembangan *DES* adalah *GOST*. *GOST* dan *DES* secara struktural memiliki banyak kesamaan. Karena pada dasarnya *GOST* dibuat sebagai alternatif atas algoritma *DES*. *GOST* adalah salah satu metode kriptografi modern yang menggunakan algoritma simetrik. *GOST* merupakan singkatan dari "*Gosudarstvennyy Standard*" yang berarti "*State Standard"* atau Standart Pemerintah. Metode ini dikembangkan oleh pemerintah Uni Soviet pada masa perang dingin untuk menyembunyikan data atau informasi yang bersifat rahasia

pada saat komunikasi. Algoritmanya merupakan algoritma enkripsi yang memiliki jumlah proses sebanyak 32 *round* dan menggunakan 64 bit *block cipher* dengan 256 bit *key*. Metode *GOST* juga menggunakan 8 buah *S-Box* yang permanen dan operasi *XOR* serta *Rotate Left Shift* (Rinaldi, 2006).

Beberapa kelebihan algoritma *GOST* jika dibandingkan dengan algoritma *DES* dan *Rijndael* sebagai algoritma standart adalah pada jumlah *bit key* dan banyaknya putaran (*round*). Semakin panjang kunci yang digunakan, dan semakin banyak putaran yang dilakukan, dapat dikatakan algoritma kriptografi tersebut akan semakin aman. Hal ini dikarenakan untuk algoritma kriptografi modern yang menggunakan kunci (*key*), kerahasiaan berasal dari adanya algoritma yang kuat dan dipublikasikan dengan kunci yang panjang (Menezes, 1996). Metode *GOST* juga dapat menyediakan sebuah tingkat keamanan informasi yang fleksibel, yang dapat digunakan untuk melindungi informasi pada sistem komputer dan jaringan komputer serta dapat diimplementasikan baik untuk perangkat keras atau perangkat lunak (Josef,1994).

Penggunaan metode *GOST* pada awalnya ditujukan untuk menangani proses enkripsi dan dekripsi *file text*. Namun dengan kemampuan yang dimiliki metode *GOST*, metode ini dirasa mampu untuk menangani proses enkripsi dan dekripsi *file* secara umum, dalam berbagai tipe *file*. Selama sebuah *file* dapat dirubah ke dalam bentuk biner, maka metode *GOST* dapat diterapkan untuk melakukan enkripsi dan dekripsi terhadap *file* tersebut. Dengan memanfaatkan algoritma *GOST*, jumlah kunci dan banyaknya putaran, *file* yang terenkripsi menggunakan metode GOST telah dirubah ke dalam bentuk yang lebih aman, berupa *file* dengan tipe data yang berbeda yang hanya dapat dibuka kembali menggunakan perangkat lunak dekripsi metode *GOST*.

Maka berdasarkan latar belakang yang telah dipaparkan, dalam skripsi ini, diangkat judul yaitu **"Pemanfaatan Metode GOST Untuk Enkripsi dan Dekripsi File".**

1.2 Rumusan Masalah

Rumusan masalah yang akan dijadikan obyek penelitian adalah bagaimana mengimplementasikan metode *GOST* untuk melakukan proses enkripsi dan dekripsi *file* dengan menggunakan berbagai macam *input* data dalam bentuk dan ukuran yang berbeda.

1.3 Tujuan Penelitian

Sesuai dengan rumusan masalah di atas, maka tujuan penelitian ini adalah sebagai berikut :

1. Melakukan enkripsi dan dekripsi *file* menggunakan metode *GOST*.
2. Melakukan analisis pada penerapan beberapa kasus dengan *input* data yang berbeda. Analisis yang dilakukan menyangkut analisis ukuran (*size*) *file*, analisis waktu proses, dan analisis ketahanan algoritma.

1.4 Manfaat Penulisan

Manfaat dari penyusunan skripsi ini adalah sebagai berikut :

1. Mengetahui kemampuan metode *GOST* dalam menangani proses enkripsi dan dekripsi *file*.
2. Mengetahui penerapan metode *GOST* untuk beberapa kasus dengan *input* data yang berbeda.

1.5 Batasan Masalah

Batasan masalah pada penelitian ini adalah sebagai berikut :

1. Obyek yang diteliti adalah perangkat lunak enkripsi dan dekripsi metode *GOST*.
2. Jenis *input*an yang digunakan dalam aplikasi berupa *file* dalam bentuk dan ukuran yang beragam.
3. Data yang digunakan dalam penelitian ini adalah lima jenis *file* dalam batasan ukuran *file* yang telah ditentukan.

BAB II
DASAR TEORI

2.1 Kriptografi

Secara etimologi, kata kriptografi berasal dari gabungan dua kata dalam bahasa Yunani yaitu "*kriptos*" dan "*graphia*". Kata *kriptos* mendeskripsikan sesuatu yang disembunyikan, rahasia atau misterius. Sedangkan kata *graphia* berarti tulisan. Kriptografi, secara umum adalah ilmu dan seni untuk menjaga kerahasiaan berita (Schneier,1996). Selain pengertian tersebut terdapat pula pengertian ilmu yang mempelajari teknik-teknik matematika yang berhubungan dengan aspek keamanan informasi seperti kerahasiaan data, keabsahan data, integritas data, serta autentikasi data (Menezes, 1996).

Terdapat tiga komponen penting dalam kriptografi, yaitu :

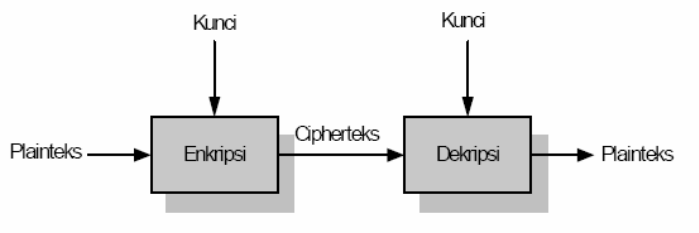
1. *Plaintext* atau *Cleartext,* merupakan sumber berita/pesan/teks asli.
2. *Ciphertext,* merupakan data sandi atau teks yang sudah diproses (diacak/digantikan).
3. Algoritma dan Kunci. Algoritma kriptografi adalah fungsi matematika untuk enkripsi dan dekripsi sedangkan kunci merupakan sederetan bit yang diperlukan untuk mengenkripsi dan mendekripsi data.

Dalam menjaga kerahasiaan data kriptografi mentransformasikan *plaintext* ke bentuk *ciphertext* yang tidak dapat dikenali. *Ciphertext* ini kemudian dikirimkan oleh pengirim (*sender*) kepada penerima (*receiver*). Setelah sampai di penerima, *ciphertext* ditranformasikan kembali ke bentuk *plaintext* agar dapat dikenali. Proses untuk menyamarkan pesan dengan cara sedemikian rupa untuk menyembunyikan isi aslinya disebut enkripsi. Pesan yang telah dienkripsi disebut *ciphertext*. Proses pengembalian sebuah *ciphertext* ke *plaintext* disebut dekripsi. Secara umum, proses enkripsi dan dekripsi dengan menggunakan kunci dapat dilihat pada gambar 2.1

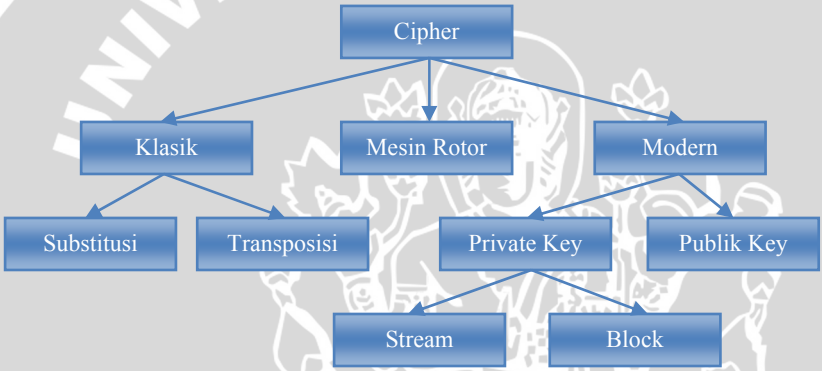
2.2 Algoritma Kriptografi

Algoritma kriptografi (*cryptographic algorithm*) disebut *cipher,* merupakan persamaan matematik yang digunakan dalam proses enkripsi dan deskripsi (Schneier, 1996). Secara umum, algoritma

kriptografi *(cipher)* dibagi menjadi beberapa bagian yang dapat dilihat pada gambar 2.2.



Gambar 2.1 Proses enkripsi dan dekripsi menggunakan sebuah kunci



Gambar 2.2 *Taksonomi Cipher*

Dalam kriptografi yang berbasiskan kunci, terdapat dua buah bentuk umum algoritma, yaitu Algoritma Kunci Privat *(Private Key Algoritm)* dan Algoritma Kunci Publik *(Public Key Algoritm)*.

2.2.1 Algoritma Kunci Privat

Algoritma kunci privat dikenal dengan sebutan algoritma simetrik. Simetrik karena algoritma ini menggunakan sebuah kunci yang sama, baik dalam melakukan enkripsi maupun dekripsi. Karena kunci memegang peranan yang sangat penting, maka algoritma ini disebut juga dengan algoritma kunci rahasia (*Secret key algorithm)*. Rumusan notasi matematik algoritma kunci privat dapat dilihat pada persamaan 2.1 (Schneier,1996).

$$E_K(M) = C \quad ; \quad D_K(C) = M \quad ; \quad D_K(E_K(M)) = M \qquad (2.1)$$

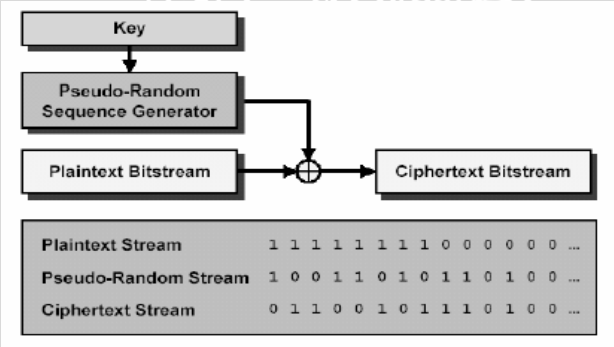
Algoritma simetrik dapat dibagi menjadi dua kategori, yaitu *stream cipher,* dan *block cipher*.

Stream Cipher

Stream cipher adalah jenis algoritma enkripsi simetrik yang mentransformasikan data karakter per karakter. *Stream cipher* digunakan untuk blok data yang lebih kecil, biasanya ukuran bit. *Stream cipher* menghasilkan *keystream*, suatu barisan bit yang digunakan sebagai kunci. *Keystream* dibangkitkan dari sebuah pembangkit yang dinamakan *Keystream Generator* atau *Pseudo Random Sequence Generator.* Proses enkripsi dicapai dengan menggabungkan *keystream* dengan *plaintext* dan operasi XOR. Gambar mengenai proses enkripsi algoritma *Stream Cipher* dapat dilihat pada gambar 2.3.

Contoh algoritma yang digunakan dalam *stream cipher* adalah :

- RC4
  Algoritma ini ditemukan pada tahun 1987 oleh Ronald Rivest dan menjadi simbol keamanan RSA (Rivest Shamir Adleman). RC4 menggunakan panjang kunci dari 1 sampai 256 bit yang digunakan untuk menginisialisasikan tabel sepanjang 256 bit.
- Tagima
  Algoritma yang digunakan Jerman dalam Perang Dunia II, dan berhasil dipecahkan oleh tawanan Amerika Serikat.

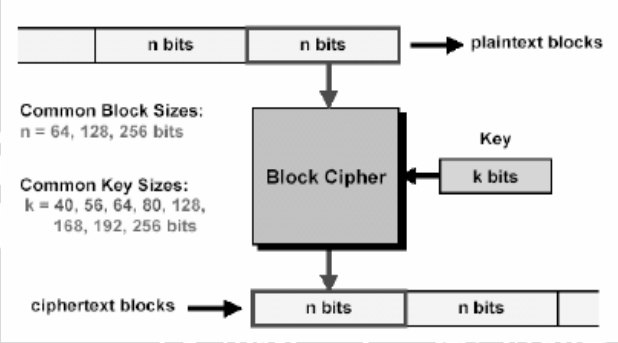


Gambar 2.3 Enkripsi Algoritma *Stream Cipher*

Block Cipher

Block cipher adalah sebuah fungsi yang memetakan n-*bit* blok *plaintext* menjadi n-*bit ciphertext*. Fungsi tersebut terdiri dari sebuah

algoritma dan sebuah kunci. Hasil pemetaan dari *plaintext* ke *ciphertext* akan berbeda tergantung pada kunci yang digunakan. Pada umumnya, *block cipher* memproses *plaintext* dengan blok yang relatif panjang lebih dari 64 bit (biasanya 64 bit atau 128 bit blok). Untuk panjang blok 64 bit, algoritma *block cipher* akan mengenkripsi 8 karakter pada setiap kali penyandian (1 karakter = 8 bit dalam pengkodean ASCII). Gambar mengenai proses enkripsi algoritma *block cipher* dapat dilihat pada gambar 2.4.



Gambar 2.4 Algoritma *Block Cipher*

Pembagian rangkaian *plaintext* ke dalam blok-blok berpengaruh pada keamanan. Blok yang terlalu kecil rentan terhadap kebocoran kriptografi, sedangkan blok yang terlalu panjang akan mengurangi efisiensi proses komputasi.
Contoh:

```
Plainteks dalam bit :   100111010110

            1001 1101 0110      (bit blok panjang 4)
             9    13   6        (bil. Bulat 0 – 15)

            100 111 010 110     (bit blok panjang 3)
             4   7   2   6      (bil. Bulat 0 – 7)
```

Bila panjang rangkaian bit tidak habis dibagi dengan ukuran blok yang ditetapkan, terjadi proses *padding*. *Padding* menambahkan *byte-byte dummy* berupa karakter *null* pada *byte-byte* sisa yang masih kosong pada blok terakhir *plaintext*, sehingga ukurannya menjadi sama dengan ukuran blok penyandian. Pola penambahan dilakukan

secara teratur. Misalnya ditambahkan bit 0 semua, atau bit 1 semua, atau bit 0 dan bit 1 berselang-seling.
Contoh :

```
Plainteks dalam bit :   100111010110

                  10011 10101 10000  (bit blok panjang 5)
                    19    21    16   (bil. Bulat 0 – 31)
```

Proses dasar dari *block cipher* adalah substitusi pada blok kecil dan permutasi pada blok lebar. Proses ini disebut "*product cipher"*. Satu langkah proses ini disebut satu *round* dan kemudian diulang-ulang. Substitusi adalah suatu proses dimana jika kita memiliki *input* *k*-bit dengan kemungkinan 2^k, kita harus menentukan pasangan setiap *k*-bit yang lebarnya juga *k*-bit. Permutasi mempunyai definisi untuk setiap bit dari *input k*-bit, tiap bit akan ditukar posisinya ke tempat lain. Misalnya bit ke-1 dari *input* menjadi bit ke-13. Lalu bit ke-2 dari *input* menjadi bit ke 61, dan seterusnya (Mahyudin, 2006).

GOST merupakan salah satu contoh algoritma *block cipher.* Selain *GOST*, terdapat beberapa algoritma yang digunakan dalam *block cipher* antara lain :

- DES (*Data Encryption Standard*)
  DES dikembangkan pada tahun 1970-an oleh IBM dan dijadikan standart oleh pemerintah Amerika Serikat. DES menggunakan 56 *bit* kunci. Varian dari DES adalah *Triple* DES, dihasilkan dengan pemakaian algoritma DES selama 3 kali operasi.
- IDEA (*International Data Encryption Algorithm*)
  IDEA dikembangkan oleh James L. Massey dan Xuejia Lai di Zurich, Swiss. IDEA menggunakan kunci 128 *bit*, dianggap aman dan merupakan algoritma yang cukup baru.
- *Blowfish*
  Blowfish dibuat oleh Bruce Schneier. Dengan ukuran *block cipher* 64 *bit* dan panjang kunci sampai dengan 448 *bit*. Contoh aplikasi *blowfish* seperti Nautilus dan PGPfone.
- *Safer*
  Safer dikembangkan oleh James L. Massey, memiliki dua jenis ukuran kunci yaitu 64 *bit* dan 128 *bit*.

2.2.2 Algoritma Kunci Publik

Kriptografi kunci publik, dikenal dengan sebutan kriptografi asimetrik, merupakan bentuk kriptografi di mana seorang pengguna

memiliki sepasang kunci kriptografi, kunci publik (*public key*) dan kunci privat (*private key*). Kunci privat disimpan secara rahasia, sementara kunci publik dipublikasikan secara luas. Sebuah pesan yang dienkripsi dengan kunci publik hanya dapat didekripsi dengan kunci privat yang berpasangan dengannya. Secara matematis, proses algoritma kunci publik dapat dinotasikan seperti pada persamaan 2.2 (Schneier,1996).

$$E_{K1}(M) = C \quad ; \quad D_{K2}(C) = M \tag{2.2}$$

E_{k1} adalah fungsi enkripsi menggunakan kunci publik, dan D_{k2} adalah fungsi dekripsi yang menggunakan kunci privat.

Contoh algoritma yang digunakan dalam kriptografi kunci publik antara lain :

- RSA (Rivers, Shamir, dan Adleman)
  Merupakan algoritma yang paling banyak digunakan, dapat digunakan untuk proses enkripsi dan sertifikasi digital. Tingkat keamanannya terletak pada kesulitan untuk memfaktorkan bilangan *integer* yang sangat besar.
- El Gamal
  Algoritma yang tingkat keamanan didasarkan pada kesulitan untuk menghitung logaritma diskrit (*dicrete algorithm*).
- LUC
  LUC merupakan sistem enkripsi kunci umum, yang menggunakan fungsi *lucas* selain dari fungsi pemangkatan.

2.3 Landasan Matematis Kriptografi

2.3.1 Aritmatika Modulo

Aritmatika modulo (*modular arithmethic*) memainkan peran yang penting dalam komputasi integer, khususnya pada aplikasi kriptografi. Operator yang digunakan pada aritmatika modulo adalah *mod*. Operator *mod* jika digunakan pada pembagian bilangan bulat memberikan sisa pembagian sebagai kembaliannya. Sebagai contoh 53 mod 5 memberikan hasil = 10 dan sisa = 3. Maka 53 mod 5 = 3. Definisi dari operator *mod* adalah sebagai berikut (Wicaksono, 2006) :

> *Misalkan a adalah bilangan bulat dan m adalah bilangan bulat > 0. Operasi a mod m memberikan sisa jika a dibagi dengan m. Dengan kata lain a mod m = r sedemikian sehingga* $a = mq + r$ *, dengan* $0 \leq r < m$.

Operasi modulo ini ditujukan untuk mengontrol panjang *bit* data agar tidak lebih dari 32 *bit*.

2.3.2 Operator XOR

Operator biner yang sering digunakan dalam *cipher* yang beroperasi dalam mode bit adalah XOR atau *exclusive-or*. Notasi matematis untuk operator XOR adalah ⊕. Operator XOR dioperasikan pada dua bit dengan aturan seperti yang tertera pada tabel 2.1.

Tabel 2.1 *Operasi XOR*

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Dari tabel 2.1 dapat dilihat sifat-sifat unik operasi XOR yaitu,

- A ⊕ A = 0
- A ⊕ 0 = A
- A ⊕ 1 = A', dengan A' adalah komplemen dari A.

Misalkan *a*, *b*, dan *c* adalah peubah Boolean, hukum-hukum yang terkait dengan operator XOR adalah :

(i) $a \oplus a = 0$

(ii) $a \oplus b = b \oplus a$ (Hukum komutatif)

(iii) $a \oplus (b \oplus c) = (a \oplus b) \oplus c$ (Hukum asosiatif)

Jika dua rangkaian bit dioperasikan dengan XOR, maka operasinya dilakukan dengan meng-XORkan setiap bit yang berkoresponden dari kedua rangkaian bit tersebut.

Contoh :

```
10011 ⊕ 11001 = 01010
hasilnya diperoleh sebagai berikut:
  1   0   0   1   1
  1   1   0   0   1  ⊕
  0   1   0   1   0
```

Algoritma enkripsi sederhana yang menggunakan XOR adalah dengan meng-XORkan *plaintext* (*P*) dengan kunci (*K*) menghasilkan *ciphertext*. Persamaannya dapat ditulis sebagai persamaan 2.3.

$$C = P \oplus K \quad (2.3)$$

Karena meng-XOR-kan nilai yang sama dua kali menghasilkan nilai semula, maka proses dekripsi menggunakan persamaan 2.4.

$$P = C \oplus K \quad (2.4)$$

Contoh:

```
plainteks     01100101    (karakter 'e')
kunci         00110101    (karakter '5')

cipherteks    01010000    (karakter 'P')
kunci         00110101    (karakter '5')

plainteks     01100101    (karakter 'e')
```

2.3.3 Rotasi

Rotasi *bit* merupakan operasi *bit* dengan memutar suatu barisan *bit* sebanyak yang diinginkan. *Bit* yang telah tergeser tidak akan hilang karena *bit* tersebut akan dipindahkan ke sisi barisan *bit* yang berlawanan dengan arah putaran rotasi *bit*. Rotasi *bit* dibagi atas :

1. Operasi Rotasi Kiri (*Rotate Left*) yaitu pemutaran barisan *bit* ke kiri sebanyak nilai yang diberikan secara per *bit*. Kemudian *bit* kosong yang telah tergeser di sebelah kanannya akan digantikan dengan *bit* yang telah tergeser di sebelah kirinya. Operasi *Rotate Left* biasanya dilambangkan dengan "<<<".
   Contoh :
   00111111 <<< 1 = 01111110
   01111110 <<< 2 = 11111001
   11111000 <<< 3 = 11000111
2. Operasi Rotasi Kanan (*Rotate Right*) yaitu pemutaran barisan *bit* ke kanan sebanyak nilai yang diberikan secara per *bit*. Kemudian *bit* kosong yang telah tergeser di sebelah kirinya akan digantikan dengan *bit* yang telah tergeser di sebelah kanannya. Operasi *Rotate Right* biasanya dilambangkan dengan "<<<".
   Contoh :
   00111111 >>> 1 = 10011111
   01111110 >>> 2 = 10011111
   11111000 >>> 3 = 00011111

2.4 Metode *GOST*

2.4.1 Struktur *GOST*

GOST (Gosudarstvennyi Standard) berarti standart pemerintah. Metode *GOST* dikembangkan oleh Uni Soviet pada tahun 1970 sebagai alternatif atas algoritma enkripsi standart Amerika Serikat, *DES*. Hal ini menyebabkan secara struktural, *GOST* mirip dengan *DES*.

GOST merupakan algoritma enkripsi simetrik sederhana yang menggunakan 64 bit *block cipher*, dengan panjang kunci eksternal 256 bit dan jumlah putaran sebanyak 32 *round* (putaran). *GOST* memiliki kunci internal sebanyak 8 buah (K_0 sampai K_7) yang digunakan untuk setiap putaran. Karena ada 32 putaran, maka 8 buah kunci internal ini dijadualkan penggunaannya. Metode *GOST* juga menggunakan 8 buah *Substitution-Box (S-Box)* yang berbeda-beda dan operasi *XOR* serta *Left Circular Shift* (Rinaldi,2006).

Kelemahan *GOST* yang diketahui sampai saat ini adalah karena *key schedule*-nya yang sederhana sehingga pada keadaan tertentu menjadi titik lemahnya terhadap metode kriptanalisis. Kelebihan dari metode *GOST* ini adalah kecepatannya yang cukup baik, walaupun tidak secepat *Blowfish* tetapi lebih cepat dari *IDEA*. Untuk kecepatan enkripsi di memori sebesar 5MB data adalah 3.492,620 Kbyte/detik pada Pentium Pro 200 MHz dan 2.466,700 Kbyte/detik pada Pentium MMX 200 MHz (Sukmawan,1996).

Komponen dari metode *GOST* antara lain :

- *Key Store Unit* (*KSU*) menyimpan 256 *bit string* dengan menggunakan 32 *bit register* (K_0, K_1, …, K_7).
- Dua buah 32 *bit register* (R_1, R_2).
- 32 *bit adder modulo* 2^{32} (CM_1).
- *Bitwise Adder XOR* (CM_2).
- *Substitusion block* (S) yaitu berupa 8 buah 64 *bit S-Box*.
- *Rotasi Left shift register* (R) sebanyak 11 *bit*.

Secara umum proses-proses yang terjadi pada metode *GOST* dapat dilihat pada gambar 2.5.

2.4.2 Proses Pembentukan Kunci

Proses pembentukan kunci dapat dilihat pada penjabaran berikut ini :

1. *Input key* eksternal sepanjang 256 *bit* (k_1, k_2, k_3, k_4,…, k_{256}) dibagi ke dalam delapan bagian yang masing-masing panjangnya 32 bit.
   1 : ($k_{1, \ldots,} k_{32}$)
   2 : ($k_{33, \ldots,} k_{64}$)
   3 : (k_{65}, …, k_{96})
   4 : (k_{97}, … , k_{128})
   5 : (k_{129}, …, k_{160})
   6 : (k_{161}, … ,k_{192})
   7 : (k_{193}, …, k_{224})
   8 : (k_{225}, …, k_{256})
2. Delapan bagian tersebut dimasukkan ke dalam delapan buah *KSU* dengan aturan seperti berikut :
   $K_0 = (k_{32}, \ldots, k_1)$
   $K_1 = (k_{64}, \ldots, k_{33})$
   $K_2 = (k_{96}, \ldots, k_{65})$
   $K_3 = (k_{128}, \ldots, k_{97})$
   $K_4 = (k_{160}, \ldots, k_{129})$
   $K_5 = (k_{192}, \ldots, k_{161})$
   $K_6 = (k_{224}, \ldots, k_{193})$
   $K_7 = (k_{256}, \ldots, k_{225})$
3. K_0 sampai dengan K_7 merupakan kunci internal yang akan dipakai pada tiap putaran *(round)* dengan aturan sebagai berikut :
   Putaran 0 – 7 : $K_0, K_1, K_2, \ldots, K_7$
   Putaran 8 – 15 : $K_0, K_1, K_2, \ldots, K_7$
   Putaran 16 – 23 : $K_0, K_1, K_2, \ldots, K_7$
   Putaran 24 – 31 : $K_7, K_6, K_5, \ldots, K_0$

2.4.3 Proses Enkripsi

Proses enkripsi dari metode *GOST* untuk satu putaran *(round)* dapat dapat dilihat pada penjabaran berikut ini :

1. *Plaintext* dibagi ke dalam blok-blok sepanjang 64 *bit* (T_0). T_0 dapat dituliskan sebagai persamaan 2.5.

$$T_0 = (a_1(0), a_2(0), \ldots, a_{32}(0), b_1(0), \ldots, b_{32}(0)) \quad (2.5)$$

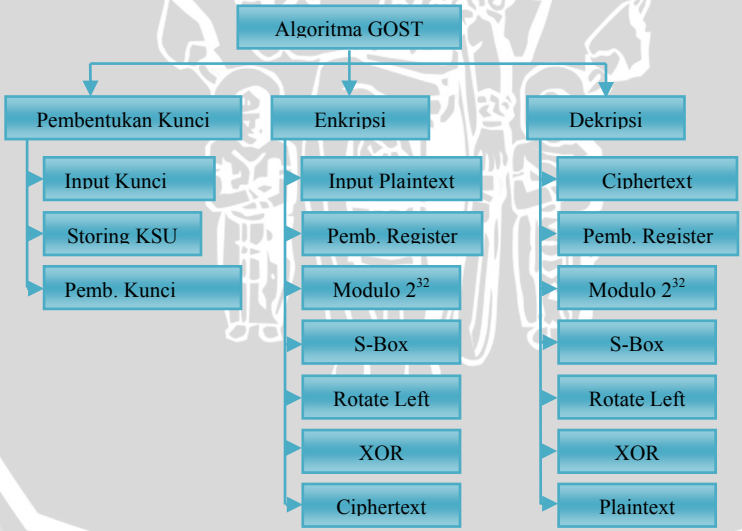
2. T_0 dibagi menjadi 2 buah bagian 32 *bit*, yaitu R_1 dan R_2. R_1 dan R_2 dapat dituliskan sebagai persamaan 2.6 dan 2.7.

$$R_1 = a_{32}(0), a_{31}(0), \ldots, a_1(0) \quad (2.6)$$
$$R_2 = b_{32}(0), b_{31}(0), \ldots, b_1(0) \quad (2.7)$$

3. Pada putaran pertama, *register* R_1 dijumlahkan dengan *partial key* K_0 modulo 2^{32}. Hasil dari penjumlahan modulo 2^{32} berupa 32 bit dan disimpan ke dalam CM_1. CM_1 dapat dituliskan sebagai persamaan 2.8.

$$CM_1 = (R_1 + K_0) \bmod 2^{32} \quad (2.8)$$

4. 32 bit CM_1 dibagi menjadi delapan bagian, masing-masing terdiri dari 4 bit. Setiap bagian dimasukkan ke dalam tabel *S-Box* yang berbeda, 4 bit pertama menjadi *input* dari *S-Box* pertama, 4 bit kedua menjadi *S-Box* kedua, dan seterusnya. Setiap 4 bit ditransformasikan ke bentuk 4 bit yang lain melalui *S-Box* yang bersesuaian. *S-Box* yang digunakan pada metode *GOST* ada 8 buah, masing – masing terdiri atas 16 kolom. Setiap kotak berisi permutasi angka 0 sampai 15. S-Box Metode *GOST* secara lengkap dapat dilihat pada tabel 2.2.



Gambar 2.5 Proses dalam Metode *GOST*

Penjelasan cara kerja *S-Box* dari Metode *GOST* dapat dilihat pada tabel 2.3. *Input* biner diubah menjadi bilangan desimal dan hasilnya menjadi urutan bilangan dalam *S-Box*.

Tabel 2.2 *S-Box* dari Metode *GOST*

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

Tabel 2.3 Penjelasan Cara Kerja *S-Box* dari Metode *GOST*

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

Contoh, jika data *input* ke *S-Box* adalah 5 (setelah didesimalkan dari digit biner) maka dicari data pada posisi ke-5. *Output* yang dihasilkan adalah 8.

5. Kedelapan hasil yang didapat dari substitusi ke *S-Box* digabungkan kembali menjadi 32 *bit* dan kemudian dilakukan rotasi *left shift* sebanyak 11 bit.
6. Hasil rotasi kemudian di-*XOR*-kan dengan R_2 oleh *adder* CM_2. CM_2 dapat dituliskan sebagai persamaan 2.9.

$$CM_2 = R_1(\text{hasil dari } \textit{rotate left shift}) \textit{ XOR } R_2 \quad (2.9)$$

7. $R_1 = CM_2$ (2.10)
8. $R_2 = R_1$ sebelum proses (2.11)

Proses penjumlahan modulo 2^{32}, *S-Box*, *Rotate Left Shift* dilakukan sebanyak 32 putaran *(round)* dengan penggunaan kunci pada masing-masing putaran berbeda-beda sesuai dengan aturan penjadualan yang ditetapkan. Setelah 32 putaran *adder* CM_2 bernilai sama dengan R_1 sedangkan R_2 menyimpan nilai R_1 sebelumnya. Nilai dari R_1 dan R_2 ini adalah 64 bit, dan merupakan *ciphertext*.

Enkripsi metode *GOST* secara umum dapat dituliskan dalam persamaan 2.12 sampai dengan 2.17 (Josef Pieprzyk,1994).

$$a(j) = ((a(j-1) \boxplus K_{(j-1) \pmod 8}) \text{ SR} \oplus b(j-i)) \quad (2.12)$$

$$b(j) = a(j-1) \quad (2.13)$$

untuk j = 1, … , 24 dan

$$a(j) = ((a(j-1) \boxplus K_{32-j}) \text{ SR} \oplus b(j-i)) \quad (2.14)$$

$$b(j) = a(j-1) \quad (2.15)$$

untuk j = 25, … , 31. Dan *ciphertext*nya (j=32) adalah :

$$a(32) = a(31) \quad (2.16)$$

$$b(32) = ((a(31) \boxplus K_0) \text{ SR} \oplus b(31)) \quad (2.17)$$

Dimana $a(0) = (a_{32}(0),\ldots, a_1(0))$ dan $b(0) = b_{32}(0),\ldots,b_1(0))$ berturut-turut adalah isi dari R_1 dan R_2. Vektor $a(j) = (a_{32}(j),\ldots, a_1(j))$ dan $b(j) = b_{32}(j),\ldots,b_1(j))$ berturut-turut adalah isi dari R_1 dan R_2, setelah j putaran enkripsi. Tanda $\oplus$ dan $\boxplus$ pada persamaan 2.12, 2.14 dan 2.17 adalah tanda operasi *XOR* dan penjumlahan modulo 2^{32}. S menunjukkan proses *S-Box* dan R untuk proses *Rotate Left Shift*. *Ciphertext* blok yang terbentuk dapat ditulis sebagai persamaan 2.18.

$$T_e = (a_1(32), a_2(32), \ldots, a_{32}(32), b_1(32), \ldots, b_{32}(32)) \quad (2.18)$$

Proses enkripsi dari metode *GOST* secara umum dapat digambarkan dalam bentuk skema seperti yang ditunjukkan pada gambar 2.6.

2.4.4 Proses Dekripsi

Proses dekripsi merupakan kebalikan dari proses enkripsi. Dalam hal ini 64 bit *ciphertext* yang dibagi ke dalam 2 *register* R_1 dan R_2. R_1 dan R_2 dapat dituliskan sebagai persamaan 2.19 dan 2.20.

$$T_e = (a_1(32), a_2(32), \ldots, a_{32}(32), b_1(32), \ldots, b_{32}(32))$$

$$R_1 = a_{32}(32), a_{31}(32), \ldots, a_1(32) \quad (2.19)$$

$$R_2 = b_{32}(32), b_{31}(32), \ldots, b_1(32) \quad (2.20)$$

Untuk penggunaan kunci masing-masing putaran pada proses dekripsi adalah sebagai berikut :

Putaran 0 – 7 : $K_0, K_1, K_2, \ldots, K_7$

Putaran 8 – 15 : $K_7, K_6, K_5, \ldots, K_0$

Putaran 16 – 23 : $K_7, K_6, K_5, \ldots, K_0$

Putaran 24 – 31 : $K_7, K_6, K_5, \ldots, K_0$

Algoritma yang digunakan untuk proses dekripsi sama dengan proses enkripsi. Persamaan dekripsi metode *GOST* secara umum dapat dituliskan dalam persamaan 2.21 sampai dengan 2.26 (Josef Pieprzyk,1994).

$$a(32\text{-}j) = ((a(32\text{-}j+1) \boxplus K_{(j-1) \pmod 8}) \text{ SR} \oplus b(32\text{-}j+1)) \quad (2.21)$$
$$b(32\text{-}j) = a(32\text{-}j+1) \quad (2.22)$$

untuk j = 1, … , 8 dan

$$a(32\text{-}j) = ((a(32\text{-}j+1) \boxplus K_{(32\text{-}j) \pmod 8}) \text{ SR} \oplus b(32\text{-}j+1)) \quad (2.23)$$
$$b(32\text{-}j) = a(32\text{-}j+1) \quad (2.24)$$

untuk j = 9, … , 31. Dan (a(0),b(0)) untuk j = 32 adalah :

$$a(0) = a(1) \quad (2.25)$$
$$b(0) = ((a(1) \boxplus K_0) \text{ SR} \oplus b(1)) \quad (2.26)$$

Plaintext yang dihasilkan pada proses dekripsi dapat dituliskan sebagai persamaan 2.29. Persamaan 2.29 merupakan hasil penggabungan dari persamaan 2.27 dan 2.28.

$$R_1 = b_{32}(0), b_{31}(0), \ldots, b_1(0) \quad (2.27)$$
$$R_2 = a_{32}(0), a_{31}(0), \ldots, a_1(0) \quad (2.28)$$
$$T_0 = (a_1(0), a_2(0), \ldots, a_{32}(0), b_1(0), \ldots, b_{32}(0)) \quad (2.29)$$

Proses dekripsi dari metode *GOST* dapat digambarkan dalam bentuk skema seperti yang ditunjukkan pada gambar 2.7.

2.5 *File*

Sebuah *file* adalah sebutan untuk serangkaian digit biner. Pada tingkatan yang lebih tinggi rangkaian digit biner ini dapat merepresentasikan sebuah nilai *integer*, sederetan karakter, atau dapat juga berupa rangkaian instruksi *software* aplikasi untuk dijalankan oleh komputer (Ramadhani,2008). Sebuah *file* memiliki ukuran yang diukur dalam satuan *bytes*.

File memiliki atribut yang berisi informasi *file* antara lain nama *file*, tipe, lokasi disimpan, ukuran, waktu pembuatan dan identitas pembuat, proteksi, dan beberapa informasi lain tentang *file*. Pada sebuah *file* juga dapat dilakukan operasi-operasi *file* antara lain membuat, menulis, membaca, menghapus, mencari, membuka, menutup, menghapus dengan menyisakan atribut dan beberapa operasi *file* lainnya.

2.5.1 Format *File*

Format *file* merupakan sebuah cara untuk mengkodekan informasi yang ada di dalam *file* (Wikipedia,2008). Karena sebuah *file* hanya dapat menyimpan data dalam serangkaian digit biner (bit-bit), maka sistem komputer harus mempunyai sebuah cara untuk mengkonversikan informasi dari *file* ke dalam sebuah rangkaian instruksi yang dapat dibaca oleh aplikasi *software* yang sesuai sehingga pada akhirnya dapat dipahami oleh *user*. Terdapat format *file* yang berbeda untuk jenis-jenis informasi yang berbeda pula.

Format *file* secara umum dapat dibagi menjadi dua kategori, yaitu format *file* teks dan format *file* biner (www.anycount. com,2008).

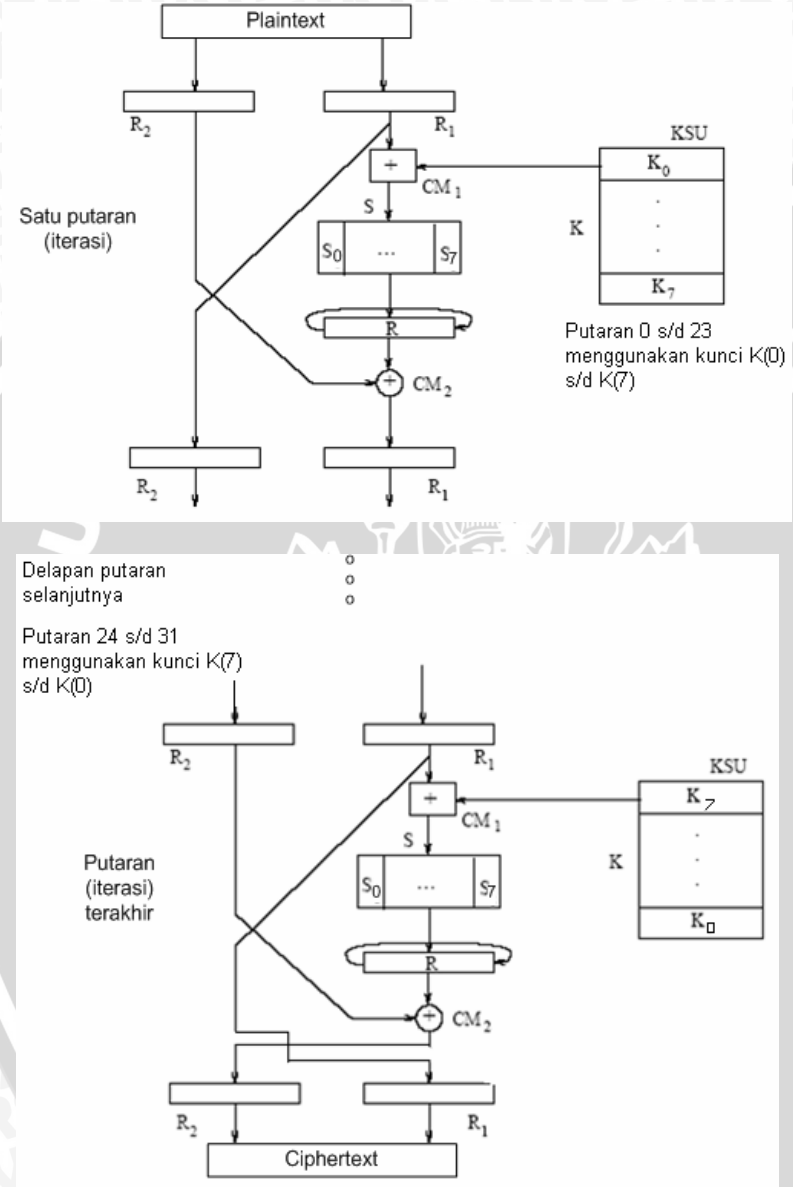
- **Format *File* Teks**

File teks adalah sebuah *file* yang berkorespondensi *one-to-one* antara *byte-byte* dengan karakter-karakter (huruf dan angka) yang terdapat di dalamnya (www.anycount.com,2008). Representasi digit biner ke dalam sebuah karakter dilakukan melalui sebuah sistem spesifik dengan cara merelasikan bilangan-bilangan biner di dalam *file* ke dalam karakter teks. Sistem ini dinamakan *encoding*. Jenis *encoding* yang umum digunakan untuk format *file* teks antara lain *Unicode UTF-8*, *Unicode UTF-16*, *ISO 8859*, dan ASCII. *File* dengan format *file* teks disimpan dalam ekstensi *.txt*.

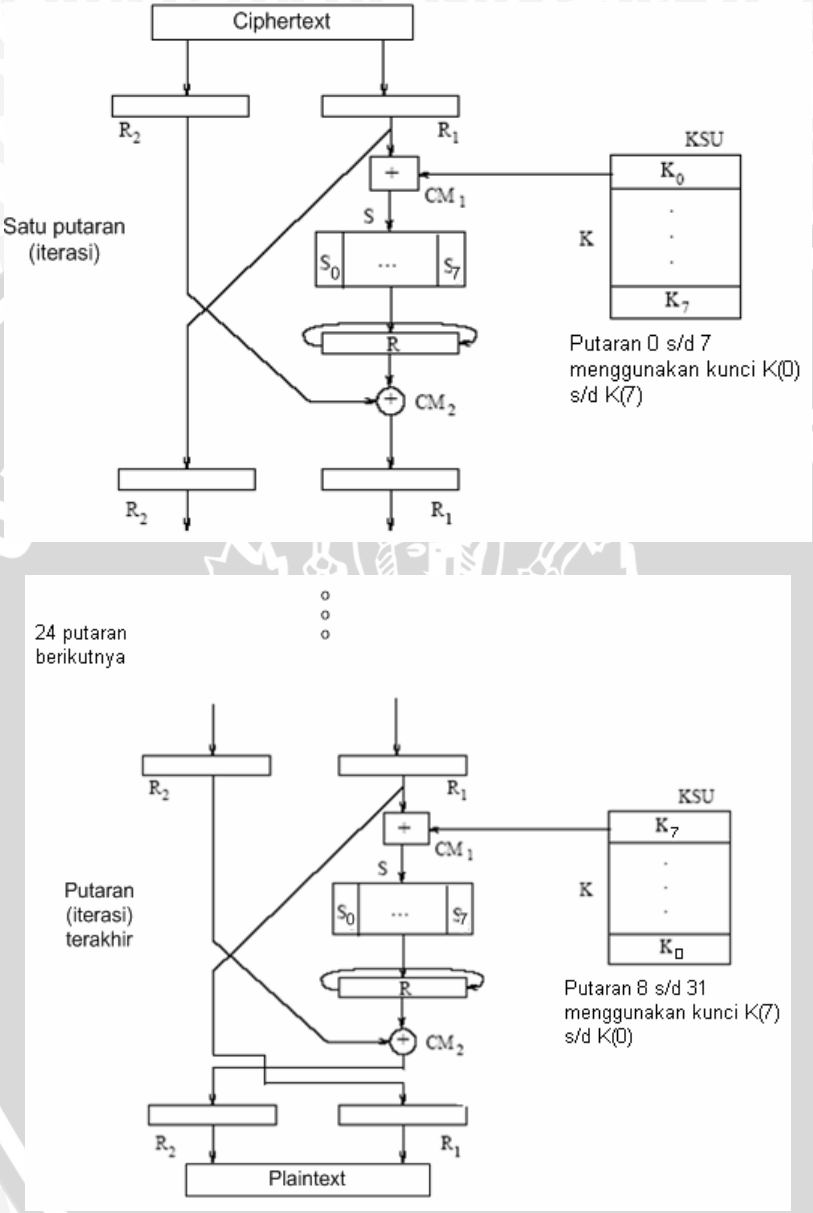
Secara umum, *file* teks mengandung karakter-karakter ASCII dengan beberapa karakter kontrol, seperti tabulasi dan enter, dengan tidak mengandung informasi-informasi tambahan seperti informasi mengenai huruf, *hyperlink*, ataupun gambar. Sebuah *file* dengan format *file* teks dapat ditampilkan dengan perangkat lunak *text editor* sederhana seperti *Notepad*, *Word Processor* dan sebagainya.

- **Format *File* Biner**

File biner adalah sebuah *file* yang dapat berisikan beragam tipe data, dikodekan ke dalam bentuk biner sebagai media penyimpanan dan untuk tujuan pemrosesan (Wikipedia,2008). *File* biner berisikan sebuah format informasi yang hanya dapat dimengerti oleh aplikasi dan proses tertentu. Jika *user* dapat membaca secara langsung sebuah *file* teks yang hanya berisikan karakter-karakter ASCII, maka pembacaan *file* biner harus dilakukan dengan menggunakan perangkat lunak yang sesuai.



Gambar 2.6 Skema Proses Enkripsi dengan Metode *GOST* (Josef Pieprzyk,1994)



Gambar 2.7 Skema Proses Dekripsi dengan Metode *GOST* (Josef Pieprzyk,1994)

Sebagai contoh, hanya *Microsoft Word* dan beberapa perangkat lunak pemrosesan teks lain yang dapat menangani format informasi dalam sebuah *file Word*. *Executable file*s, *compiled programs*, SAS dan SPSS *system file*s, *spreadsheets*, *compressed file*s, dan *file-file* gambar (*image*) adalah beberapa contoh dari *file* biner. *File-file* ini disimpan dengan ekstensi tertentu sesuai dengan kaidah yang telah ditentukan. Beberapa contoh *file* biner dengan ekstensi masing-masing dapat dilihat pada gambar 2.8.

file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

Gambar 2.8 Berbagai Macam Tipe *File* Biner
(**Sumber: Silberschatz, Galvin, Gagne.** *Operating System Concepts)*

File biner dikodekan dalam *byte*, yang berari bahwa setiap digit biner yang ada dikelompokkan ke dalam sebuah grup yang masing-masing berisi delapan digit biner (www.anycount.com,2008). Jika *user* menampilkan *file* ini menggunakan *text editor* misalnya sebuah *Notepad*, maka tiap grup delapan bit ini akan ditranslasikan sebagai sebuah karakter. *File* biner yang hanya terdiri atas data tekstual tanpa

berbagai macam format informasi yang menyertainya dinamakan *file plaintext*. Di dalam beberapa kasus, *file plaintext* dibedakan dengan *file* biner. Hal ini dikarenakan *file* biner dibuat dengan fungsi yang lebih luas dibandingkan dengan *plaintext*.

Diantara berbagai tipe *file* biner yang ada, terdapat beberapa jenis tipe *file* yang paling banyak digunakan *user* dalam berkomunikasi dan bertukar informasi. Antara lain *file* multimedia dimana pada setiap digit binernya berisikan informasi mengenai suara dan gambar.

- **Format *File* Gambar**

Terdapat dua kategori besar untuk format *file* gambar, yaitu format bitmap dan format vektor. Untuk kepentingan dalam pengolahan citra yang biasanya dilakukan pada setiap *pixel* penyusun citra, format bitmap akan lebih mudah digunakan. Format bitmap menyimpan data kode gambar secara digital dan lengkap (Usman,2005). Format bitmap dapat dikelompokkan menjadi 3 macam, yaitu: bitmap tidak terkompres, bitmap terkompres, dan *PostScript*. Pada gambar bitmap tidak terkompres matriks yang terdiri dari kedalaman warna disimpan apa adanya, sehingga kualitas gambar yang disimpan sama dengan kualitas gambar ketika diambil (*scan* atau foto digital). Macam-macam format ini antara lain *bmp*, *tiff*, dan *ras*. Gambar bitmap terkompres susunan matriksnya masih sama dengan bitmap tidak terkompres, bedanya terletak pada teknik pengompresan dari tiap-tiap gambar. Format yang biasa digunakan adalah format *JPEG (joint picture enhanced graphics)* sering disebut dengan istilah *loosy compression.*

Kompresi seperti ini hanya baik pada gambar yang terdiri atas jutaan warna *(true color bitmap)*, misalnya foto, namun sangat jelek pada gambar yang hanya terdiri dari 256 warna (seperti hasil gambar sketsa/kanvas). Gambar yang hanya terdiri dari 256 warna hanya baik dikompres dalam format *GIF (graphic interchange format)* atau *PNG (portable network graphics)*.

Gambar terformat *PostScript*®, seperti *EPS (Encapsulated PostScript)* memiliki dua lapis gambar di luar *header*. Lapis paling bawah adalah raster yang biasanya menggunakan format *tiff*, sedangkan lapis atasnya adalah perintah-perintah postscript yang hanya dikenal oleh printer atau penampil *postscript* seperti *gv*, *Gsview* atau *ghostview* (Warmada,2004). Format *file* gambar yang

umum digunakan dalam pemrograman pengolah citra ditampilkan pada tabel 2.4.

Tabel 2.4 Format *file* grafik Bitmap

Nama Format	Ekstensi	Kegunaan
Microsoft Windows Bitmap Format	BMP	Format umum untuk menyimpan citra bitmap. Dikembangkan oleh Microsoft
Comp*user*ve Graphics Interchange Format	GIF	Format umum citra yang dirancang untuk keperluan transmisi melalui modem
Aldus Tagged Image *File* Format	TIFF	Format kompleks dan miltiguna. Dikembangkan oleh Aldus dan Microsoft
Joint Picture Enhanced Graphics	JPEG	Format umum untuk menayangkan warna dengan kedalaman 24-bit *true color*. Mengkompresi gambar dengan sifat *lossy*.
ZSoft Pengolahan Citra Paintbrush Formats	PCX	Dirancang untuk menyimpan citra layar dan merupakan format bitmap yang didukung luas

2.6 Avalanche Effect

Avalanche effect adalah salah satu karakteristik yang menjadi acuan untuk menentukan baik atau tidaknya ketahanan suatu algoritma kriptografi (khususnya *block cipher* dan *hash*). *Avalanche effect* dalam kriptografi dapat dilihat ketika dilakukan perubahan kecil pada *plaintext* maupun *key* yang akan menyebabkan perubahan signifikan terhadap *ciphertext* yang dihasilkan. Dengan kata lain, perubahan satu bit pada *plaintext* maupun *key* akan menghasilkan perubahan banyak bit pada *ciphertext*. Suatu *avalanche effect* dikatakan baik jika perubahan bit yang dihasilkan berkisar antara 45-60% (sekitar separuhnya, 50 % adalah hasil yang sangat baik) (Rivest, 1998). Hal ini dikarenakan perubahan tersebut berarti membuat perbedaan yang cukup sulit untuk kriptanalis melakukan serangan. Nilai *avalanche effect* dirumuskan dengan persamaan 2.30.

$$Avalanche\_Effect(AE) = \frac{\sum bit\_berubah}{\sum bit\_total} * 100\% \qquad (2.30)$$

Pengukuran *avalanche effect* meliputi pengaruh dari perubahan satu bit *plaintext*, kunci enkripsi, dan *ciphertext.*

BAB III
METODOLOGI DAN PERANCANGAN

Pada bab metode dan perancangan ini akan dibahas mengenai metode yang digunakan dalam pembuatan perangkat lunak enkripsi dan dekripsi metode *GOST* antara lain :

1. Melakukan studi literatur mengenai metode *GOST,* dengan melakukan pemahaman konsep enkripsi dan dekripsi menggunakan metode *GOST*.
2. Melakukan perancangan untuk membangun perangkat lunak. Perancangan yang dilakukan meliputi perancangan proses *input*, proses pembentukan kunci, proses enkripsi, proses dekripsi, perancangan *interface* dan perancangan uji coba perangkat lunak.
3. Implementasi hasil perancangan ke dalam bahasa pemrograman *Borland Delphi 7.0*.
4. Melakukan uji coba enkripsi dan dekripsi *file* menggunakan perangkat lunak yang telah dibuat.
5. Melakukan analisis pada penerapan beberapa kasus dengan *input* data yang berbeda.
6. Melakukan evaluasi terhadap perangkat lunak yang telah dibuat.

Diagram alir pembuatan perangkat lunak enkripsi dan dekripsi metode *GOST* dapat dilihat pada Gambar 3.1.

3.1 Analisis Perangkat Lunak

Pada subbab analisis akan dibahas mengenai semua hal yang diperlukan dalam proses pembuatan perangkat lunak enkripsi dan dekripsi metode *GOST*.

3.1.1 Deskripsi Umum Perangkat Lunak

Perangkat lunak enkripsi dan dekripsi metode *GOST* dibuat untuk *user* yang ingin menjaga kerahasiaan *file* yang dimilikinya. Perangkat lunak ini merupakan sebuah sistem yang menerima masukan berupa sebarang tipe *file* (*plain-file*), yang kemudian akan dilakukan enkripsi menghasilkan *file* dalam format berbeda (*cipher-file*). Informasi dan data yang terdapat dalam sebuah *cipher-file* telah disandikan menggunakan sebuah kunci dan hanya dapat dibuka kembali oleh pihak pemilik kunci (*key*). Rancangan format *file* hasil

enkripsi (*cipher-file)* dengan metode GOST adalah *GOST Encrypted File* (*.gef*).

Skema proses enkripsi dan dekripsi *file* menggunakan metode *GOST* dapat dilihat pada gambar 3.2.

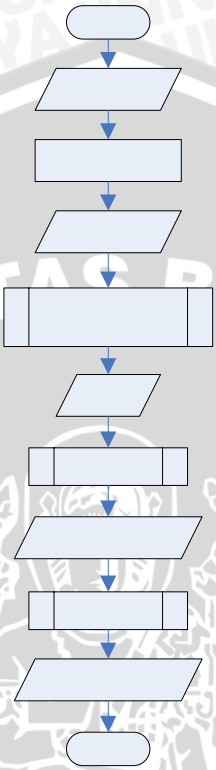
Gambar 3.1 Diagram Alir Pembuatan Perangkat Lunak Enkripsi Dekripsi Metode *GOST*

Gambar 3.2 Skema Proses Enkripsi dan Dekripsi *File* Menggunakan Metode GOST

Metode *GOST* merupakan metode enkripsi yang beroperasi dalam digit biner. Oleh karena itu untuk setiap data *file* yang akan diolah menggunakan metode *GOST* harus terlebih dahulu dirubah menjadi bentuk biner.

Secara umum, proses-proses yang terjadi sewaktu *user* menggunakan perangkat lunak enkripsi dekripsi metode *GOST* adalah :

1. *User* diminta untuk meng*input*kan *file* yang akan dienkripsi atau *file .gef* yang akan didekripsi. Data *file* kemudian dirubah ke dalam bentuk biner.



Gambar 3.3 Diagram Langkah Perangkat Lunak Enkripsi Dan Dekripsi *File*

2. *User* diminta untuk memasukkan kata kunci *(key)*. Kemudian akan dilakukan proses pembentukan kunci seperti yang telah dibahas pada bab 2, subbab 2.4.2 Proses Pembentukan Kunci.
3. Dari data biner yang didapatkan dari *file input* akan dilakukan proses enkripsi menggunakan *key*. Proses enkripsi *file* seperti yang telah dijelaskan pada bab 2, subbab 2.4.3 Proses Enkripsi.
4. Hasil enkripsi berupa *cipher file* akan didekripsi kembali menggunakan kunci menghasilkan *file* awal (*plain file*). Proses dekripsi seperti yang telah dijelaskan pada bab 2, subbab 2.4.4 Proses Dekripsi.

Proses-proses yang dilakukan oleh perangkat lunak dapat digambarkan dalam diagram langkah perangkat lunak (gambar 3.3).

3.1.2 Batasan Perangkat Lunak

Batasan dari perangkat lunak yang akan dibangun adalah :

1. *Input*an untuk kata kunci berupa karakter ASCII.
2. *File* hasil enkripsi berupa *file .gef* dan tidak dapat dilihat bentuknya *(preview*).

3.2 Perancangan Perangkat Lunak

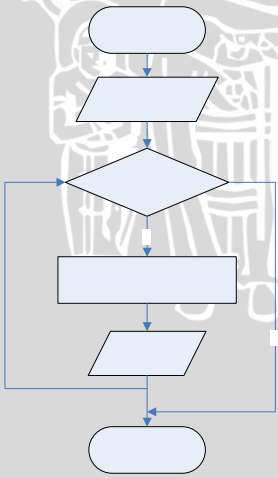
Berdasarkan deskripsi umum yang telah dijelaskan, berikut ini akan dibahas mengenai arsitektur dan proses yang terjadi pada perangkat lunak yang akan dibangun.

3.2.1 Perancangan Proses *Input* Perangkat Lunak

Data yang di*input*kan pada perangkat lunak ini, seperti yang telah dijelaskan sebelumnya berupa sebarang tipe *file*.

Input Plain-File

Input plain-file diperoleh dari *input*an *user* secara langsung pada waktu *running program*. *User* dapat memilih sebarang tipe *file* sebagai *file input*. Setelah *file* dipilih akan dilakukan proses pembacaan *file*. Data hasil pembacaan akan disimpan dalam sebuah variabel hasil untuk nantinya dapat diproses lebih lanjut. *Flowchart* proses *input plain-file* dapat dilihat pada gambar 3.4.



Gambar 3.4 *Flowchart* Proses *Input Plain-File*

3.2.2 Perancangan Proses Pembentukan Kunci

Pada proses ini, perangkat lunak akan meminta *user* untuk meng*input*kan sejumlah karakter sebagai *key* eksternal. Kemudian akan dilakukan proses *padding*. Proses *padding* dilakukan untuk mengetahui jumlah *key* yang dimasukkan. Bila *key* kurang dari ketentuan (<32 karakter), perangkat lunak akan menambahkan karakter #0 (NULL) pada blok terakhir *key* sejumlah kekurangannya, sehingga ukurannya menjadi sama dengan ukuran blok penyandian. Sedangkan bila *input* kata kunci lebih dari ketentuan (>32 karakter), perangkat lunak hanya akan mengenali *key* yang dimasukkan sampai sejumlah 32 karakter, karakter lainnya dianggap tidak ada. Setelah didapatkan *input* 32 karakter *key*, kemudian akan dilakukan pengubahan *key* ke dalam bentuk biner dengan menggunakan fungsi *RealToBin*. Hasilnya adalah *key* biner 256 bit. Proses *padding* ini dilakukan dalam fungsi *Padding*.

Key biner 256 bit kemudian akan dibagi ke dalam delapan kelompok yang masing-masing terdiri atas 32 bit *key* dan dimasukkan ke dalam KSU (*Key Store Unit)* (subbab 2.4.2 Proses Pembentukan Kunci). Pengelompokan *key* dan *storing* ke KSU dilakukan pada fungsi *Bentuk_Kunci* dengan menggunakan aturan-aturan *GOST*. Untuk *flowchart* dari pembentukan kata kunci diatas, dapat dilihat pada gambar 3.5. Sedangkan *flowchart* proses *Padding* dan *Bentuk_Kunci* dapat dilihat pada gambar 3.6 dan 3.7.

3.2.3 Perancangan Proses Enkripsi

Proses enkripsi metode *GOST* dilakukan sebanyak 32 putaran *(round)*. Untuk tiap-tiap putaran, sub proses yang berlangsung antara lain adalah pembentukan register awal, operasi penjumlahan modulo 2^{32}, substitusi ke dalam *S-Box*, *Rotate Left Shift* 11 bit, dan operasi *XOR*.

***Input Plain-file*.**

Proses enkripsi metode *GOST* memerlukan *input* data dalam bentuk biner. Data yang akan dienkripsi diambil dari data *file input* yang telah ditampung dalam variabel *Hasil*. Jumlah data dalam variabel *Hasil* tidak selamanya genap. Karena metode GOST mengenkripsi data dalam blok data sebesar 8 karakter per blok, maka tetap diperlukan proses *padding* untuk data dalam variabel Hasil.

Proses *padding* yang dilakukan untuk data enkripsi sama dengan proses *padding* untuk kunci. Hanya saja konstanta bilangan yang digunakan adalah 8, sesuai dengan blok data *GOST* sebesar 8 karakter. Setelah mendapatkan proses *padding* dan terbentuk data biner dari *file*, data biner ini akan dipecah menjadi blok-blok kecil sebesar 64 bit per blok (T_i). Tiap blok 64 bit ini akan melalui proses enkripsi sebanyak 32 putaran *(round)*.

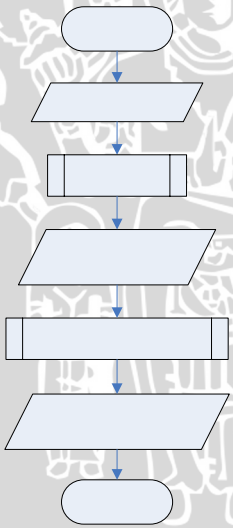
Pembentukan Register Awal

Register awal yang dibentuk adalah R_1 dan R_2. R_1 dan R_2 dapat dikatakan sebagai nilai kanan dan kiri dari 64 bit blok biner (T_i). Misalkan 64 bit blok biner awal sebagai T_0, maka R_1 dan R_2 adalah 32 bit blok biner sisi kiri T_0 dan 32 bit blok biner sisi kanan T_0.

$$T_0 = (a_1(0), a_2(0), \ldots, a_{32}(0), b_1(0), \ldots, b_{32}(0))$$

$$R_1 = a_{32}(0), a_{31}(0), \ldots, a_1(0)$$

$$R_2 = b_{32}(0), b_{31}(0), \ldots, b_1(0)$$



Gambar 3.5 *Flowchart* Proses Pembentukan Kunci

Operasi penjumlahan modulo 2^{32}

Pada putaran pertama, *register* R_1 dijumlahkan dengan *partial key* K_0 modulo 2^{32}. Hasil dari penjumlahan modulo 2^{32} berupa 32 bit dan disimpan ke dalam CM_1.

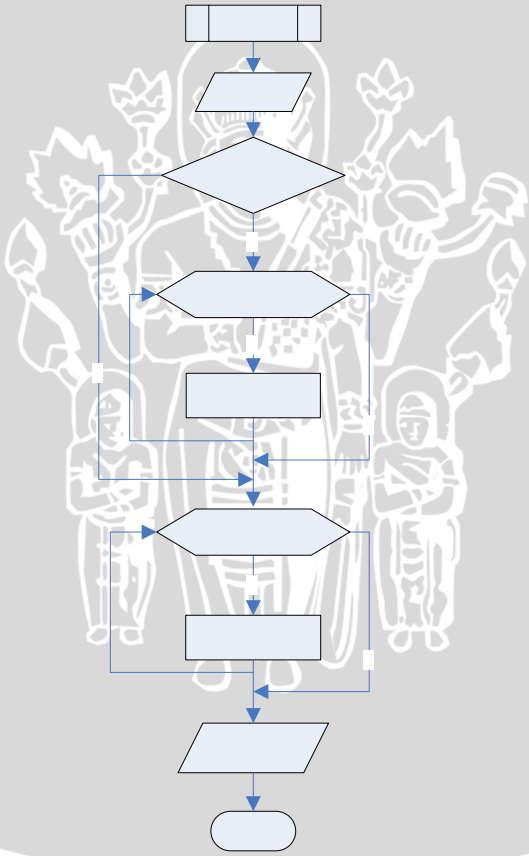
start

Input : ke

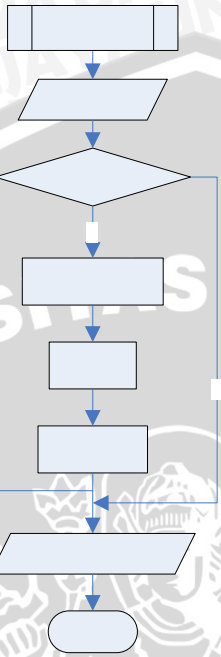
$$CM_1 = (R_1 + K_0) \ mod \ 2^{32}$$

Substitusi ke dalam *S-Box*

Dari 32 bit CM_1 yang telah terbentuk, dibagi menjadi delapan bagian, masing-masing terdiri dari 4 bit. Setiap bagian dimasukkan ke dalam tabel *S-Box* yang berbeda, 4 bit pertama menjadi *input* dari *S-Box* pertama, 4 bit kedua menjadi *S-Box* kedua, dan seterusnya. Setiap 4 bit ditransformasikan ke bentuk 4 bit yang lain melalui *S-Box* yang bersesuaian. Delapan bagian hasil transformasi ini kemudian digabungkan kembali. Proses substitusi *S-Box* seperti yang telah dijelaskan pada bab 2 subbab 2.4.3 Proses Enkripsi, tabel 2.2 dan tabel 2.3.



Gambar 3.6 *Flowchart* Proses *Padding*



Gambar 3.7 *Flowchart* Proses *Bentuk_Kunci*

***Rotate Left Shift* 11 bit**

Kedelapan hasil yang didapat dari substitusi ke S-Box digabungkan kembali menjadi 32 bit dan kemudian dilakukan rotasi *left shift* sebanyak 11 bit. Hasil dari substitusi dan rotasi ini akan cukup acak, dan cukup menyulitkan kriptanalis yang ingin mendapatkan potongan blok biner secara manual.

Operasi XOR

Hasil rotasi kemudian di-XOR-kan dengan R_2 oleh *adder* CM_2.

$CM_2 = R_1$(hasil dari *rotate left shift*) XOR R_2

$R_1 = CM_2$

$R_2 = R_1$ sebelum proses

Proses penjumlahan modulo 2^{32}, *S-Box*, *Rotate Left Shift* dilakukan sebanyak 32 putaran *(round)* dengan penggunaan kunci internal pada masing-masing putaran berbeda-beda sesuai dengan aturan penjadualan yang ditetapkan. Setelah 32 putaran, *adder* CM_2 bernilai sama dengan R_2 sedangkan R_1 menyimpan nilai R_1

sebelumnya. Gabungan dari nilai R_1 dan R_2 adalah 64 bit, dan merupakan *cipher-file*.

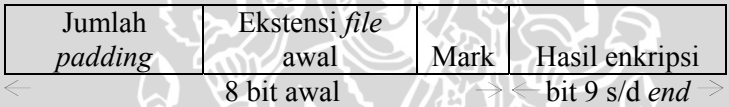
Flowchart dari proses enkripsi untuk satu putaran (*round*), dapat dilihat pada gambar 3.8.

3.2.4 Perancangan *File* Hasil Enkripsi

File hasil enkripsi (*cipher-file*) disimpan dalam bentuk *file* berformat *GOST Encrypted File* (*.gef*). *File* ini tidak dapat di*preview* atau dilihat bentuknya tanpa menggunakan kunci dan perangkat lunak enkripsi dan dekripsi metode *GOST*.

Struktur Data *File .gef*

File .gef akan mendapatkan tambahan bit sebesar delapan bit. Tambahan sebesar delapan bit ini akan menyimpan informasi yang dibutuhkan dalam proses dekripsi. Stuktur *file .gef* terbagi ke dalam empat bagian yang dapat ditunjukkan pada gambar 3.9.

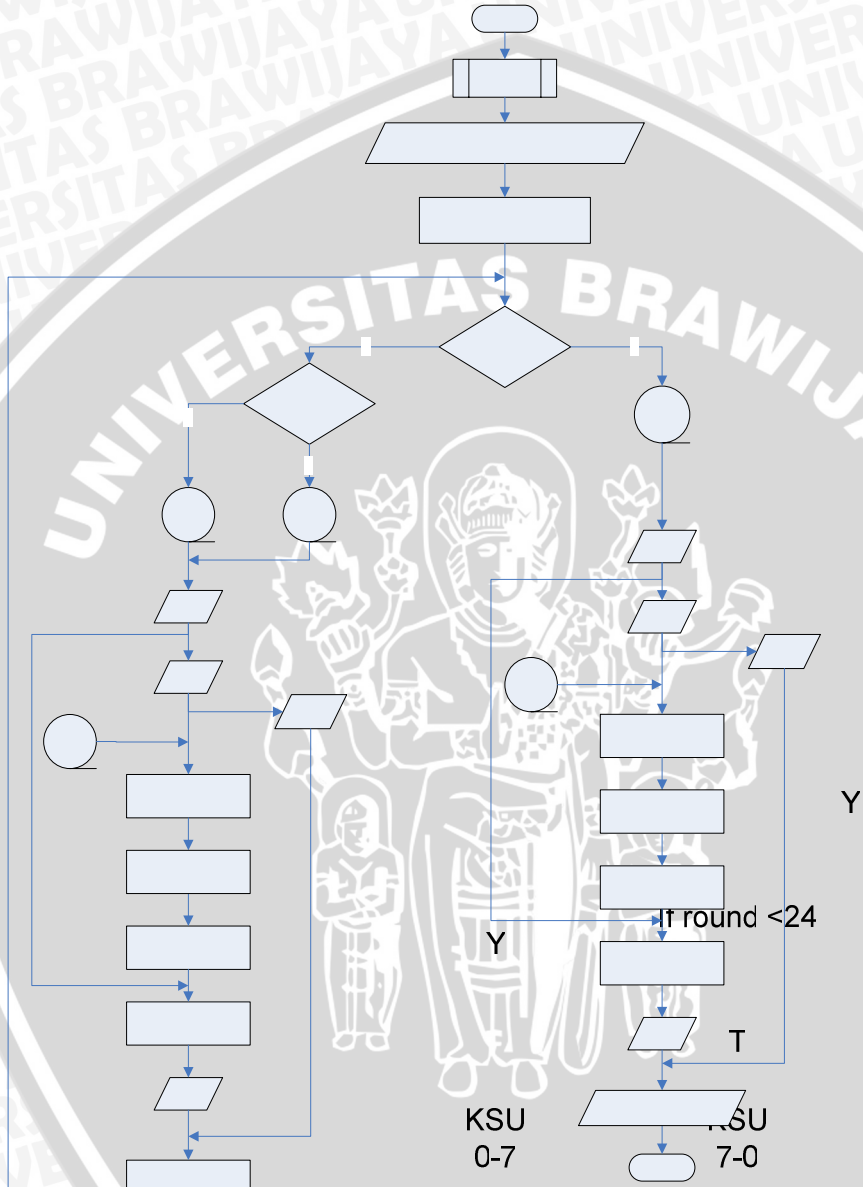


Gambar 3.9 Struktur *File .gef*

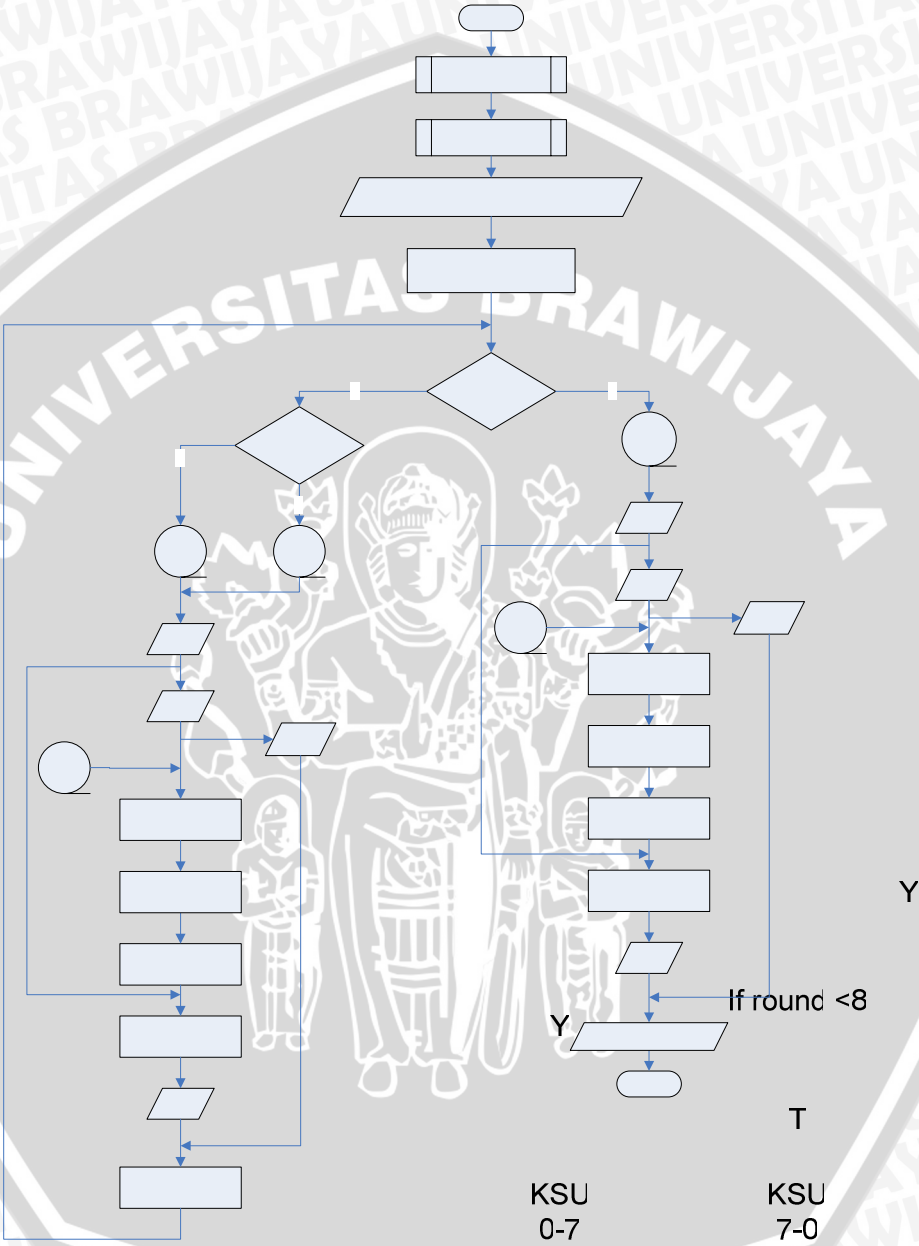
Delapan bit pertama dari *file .gef* akan menyimpan jumlah *padding text* yang terjadi, ekstensi *file* awal sebelum dilakukan proses enkripsi dan sebuah penanda (*mark*). Jumlah *padding* dan ekstensi *file* awal yang tersimpan akan digunakan dalam proses dekripsi untuk mengembalikan *file* hasil dekripsi sesuai dengan ekstensi dan ukuran semula. Besarnya alokasi tempat untuk jumlah *padding* adalah satu bit data, sedangkan untuk menyimpan ekstensi awal *file* adalah enam bit data, dan untuk *mark* adalah satu bit data. Bit ke sembilan sampai dengan akhir menunjukkan hasil enkripsi dari *file* (*cipher -file*).

3.2.5 Perancangan Proses Dekripsi

Proses dekripsi metode *GOST* merupakan kebalikan dari proses enkripsi. Dilakukan sebanyak 32 putaran *(round)*. Untuk tiap-tiap putaran, sub proses yang berlangsung sama seperti proses yang dilakukan pada waktu enkripsi, hanya saja data yang diolah adalah *cipher-file* hasil proses enkripsi (T_e).



Gambar 3.8 *Flowchart* Proses Enkripsi



Gambar 3.10 *Flowchart* Proses Dekripsi

Untuk melakukan proses dekripsi diperlukan sebuah kunci yang tepat, yaitu kunci yang sama, yang digunakan untuk proses enkripsi. Dari kata kunci yang di*input*kan, akan dilakukan proses *padding*, pengelompokan *key* dan *storing* KSU (fungsi Bentuk_Kunci) dengan aturan penggunaan kunci internal yang berbeda (sub bab 2.4.4 Proses Dekripsi).

Dari sebuah *file .gef* yang akan didekripsi terlebih dahulu akan dibaca delapan bit awal pada *file* yang menunjukkan informasi mengenai jumlah *padding* dan ekstensi dari *file* awal. Data yang akan menjadi *cipher-file* untuk proses dekripsi adalah data dari bit ke sembilan sampai dengan bit terakhir. Sedangkan untuk hasil proses dekripsi yang akan dituliskan kepada *file* hasil adalah sejumlah bit hasil dekripsi dikurangi dengan jumlah *padding*. *Flowchart* dari proses dekripsi untuk satu putaran (*round*), dapat dilihat pada gambar 3.10.

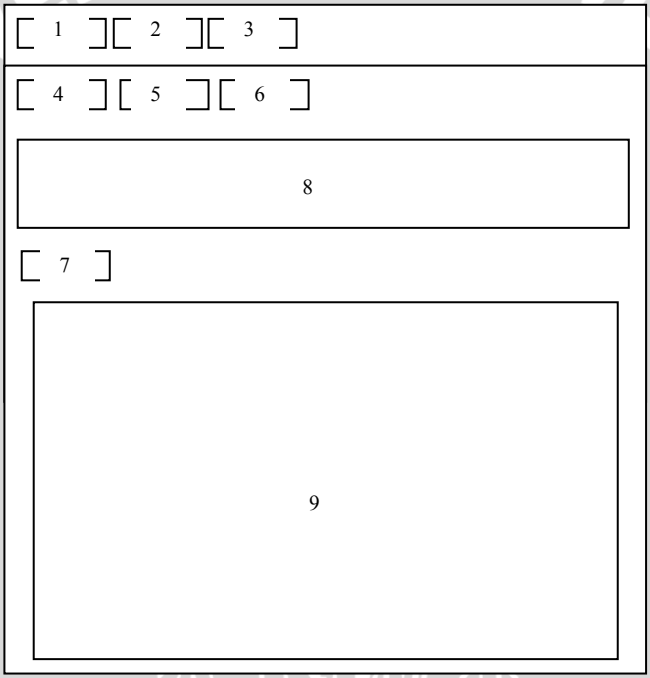
3.3 Perancangan *Interface*

Gambar 3.11 menunjukkan rancangan *interface* perangkat lunak enkripsi dan dekripsi metode *GOST*. *Interface* perangkat lunak ini terdiri dari tiga buah menu, tiga buah *image*, satu *checkbox*, satu buah *panel* dan sebuah memo.

Keterangan gambar 3.11 :

1. Menu *File*.
   Menu *File* terdiri atas submenu *Open File* dan *Exit*. Submenu *Open* digunakan untuk membuka *file* yang akan dienkripsi atau didekripsi. Submenu *Exit* untuk keluar dari aplikasi.
2. Menu *Process*.
   Terdiri atas submenu *Encrypt* dan *Decrypt*.
3. Menu *About*
   Untuk menampilkan keterangan mengenai perangkat lunak.
4. *ImageOpenFile*
   Jika di klik akan memunculkan perintah untuk membuka *file*.
5. *ImageEncrypt*
   Jika di klik fungsinya sama dengan submenu *Encrypt*, yaitu untuk menjalankan proses enkripsi.
6. *ImageDecrypt*
   Jika di klik fungsinya sama dengan submenu *Decrypt*, yaitu untuk menjalankan proses dekripsi.

7. *CheckBox View Process*
   CheckBox ini dapat dipilih untuk memunculkan hasil proses yang sedang berlangsung pada komponen memo.
8. Panel *File Properties*
   Pada panel ini akan ditampilkan keterangan-keterangan dari *file* yang dipilih, antara lain nama *file*, ukuran (*size*) *file*, status *file*, *path* tempat *file* tersimpan, dan lama waktu proses.
9. Memo. Sebuah memo akan menampilkan keterangan mengenai proses yang sedang berjalan dalam kondisi apabila *check box view process* dipilih atau dicentang.



Gambar 3.11 Rancangan *Interface* Perangkat Lunak

3.4 Perancangan Uji Coba dan Evaluasi

Perangkat lunak enkripsi dan dekripsi metode *GOST* dapat menerima *input*an berupa sebarang tipe *file*. Namun untuk mempermudah analisis data yang akan dilakukan, objek uji coba dilakukan pada beberapa tipe *file* antara lain *file* teks dan *file* multimedia (gambar dan suara). Untuk *file* teks akan digunakan *file*

.txt, sedangkan untuk *file* multimedia akan digunakan *file .jpg*, dan *.gif*, (untuk gambar) dan *file .wav* dan *.mp3* (untuk suara).

Parameter yang digunakan dalam uji coba adalah ukuran (*size*) *file*, waktu proses, dan *avalanche effect*. Besarnya masing-masing *file* yang diujikan dikelompokkan menjadi tiga, yaitu *file* berukuran kurang 10 KB, *file* berukuran 10 KB sampai dengan 50 KB dan *file* berukuran lebih dari 50 KB.

Untuk uji coba menggunakan parameter ukuran (*size*) *file*, pada setiap *file* yang diujikan akan dihitung ukuran (*size*) *file* awal (sebelum dilakukan proses enkripsi maupun dekripsi), ukuran (*size*) *file* setelah proses enkripsi dan ukuran (*size*) *file* setelah proses dekripsi. Tujuan dari uji coba menggunakan parameter ukuran (*size*) *file* adalah untuk mengetahui pengaruh proses *padding* pada *file* setelah dilakukan proses enkripsi dan dekripsi. Bentuk tabel hasil pengujian untuk parameter ukuran (*size*) *file* dapat dilihat pada tabel 3.1.

Untuk uji coba menggunakan parameter waktu proses, pada setiap *file* yang diujikan akan dihitung lama waktu proses enkripsi dan dekripsi. Tujuan dari uji coba menggunakan parameter waktu proses adalah untuk mengetahui pengaruh tipe *file* dan ukuran *file* terhadap waktu proses enkripsi dan dekripsi. Bentuk tabel hasil pengujian untuk parameter waktu proses dapat dilihat pada tabel 3.2.

Untuk uji coba dengan parameter *avalanche effect*, *file* yang diujikan adalah *file .txt* dengan beberapa ukuran *file* yang berbeda. Penggunaan *file* teks dinilai lebih mudah untuk proses analisis, karena dapat diamati secara langsung perubahan yang terjadi. Beberapa perubahan data yang dilakukan dalam uji coba dengan parameter *avalanche effect* adalah :

1. Perubahan satu bit *plaintext* terhadap *ciphertext* dengan kunci enkripsi yang sama.
2. Perubahan satu bit kunci enkripsi terhadap *ciphertext* dengan *plaintext* yang sama.
3. Perubahan satu bit *ciphertext* terhadap *plaintext* dengan kunci dekripsi yang sama.

Tujuan dari uji coba menggunakan parameter *avalanche effect* adalah untuk mengetahui ketahanan algoritma kriptografi metode *GOST*. Bentuk tabel hasil pengujian untuk parameter *avalanche effect* dapat dilihat pada tabel 3.3.

Keterangan Tabel 3.1 :

- Kolom Tipe *File* berisikan tipe *file* dari *file* uji.
- Kolom Ukuran Awal berisikan besarnya ukuran (*size*) awal *file* sebelum dilakukan proses enkripsi dan dekripsi.
- Kolom Ukuran Enkripsi berisikan besarnya ukuran (*size*) *file* setelah melalui proses enkripsi.
- Kolom Ukuran Dekripsi berisikan besarnya ukuran (*size*) *file* setelah melalui proses dekripsi.
- Ukuran besarnya *file* dalam satuan *byte*. Dimana 1 KB = 1024 *Byte*.

Keterangan Tabel 3.2 :

- Kolom Tipe *File* berisikan tipe *file* dari *file* uji.
- Kolom Ukuran *File* berisikan besarnya ukuran *file* sebelum dilakukan proses enkripsi dan dekripsi.
- Kolom Waktu Proses Enkripsi berisikan lamanya waktu yang diperlukan untuk proses enkripsi *file*.
- Kolom Waktu Proses Dekripsi berisikan lamanya waktu yang diperlukan untuk proses dekripsi *file*.
- Ukuran penghitungan waktu proses menggunakan satuan detik.

Tabel 3.1 Rancangan tabel hasil uji untuk parameter ukuran (*size*) *file.*

Tipe File	*Ukuran Awal*	*Ukuran Enkripsi*	*Ukuran Dekripsi*

Tabel 3.2 Rancangan tabel hasil uji untuk parameter waktu proses

Tipe File	*Ukuran File*	*Waktu Proses Enkripsi*	*Waktu Proses Dekripsi*

Tabel 3.3 Rancangan tabel hasil uji untuk parameter *Avalanche Effect*

Data	*Avalanche Effect (%)*
Rata-Rata	

Keterangan Tabel 3.3 :

- Kolom Data berisikan nama dari *file* uji.
- Kolom *Avalance Effect (%)* berisikan besarnya prosentase nilai *avalance effect* dari *file* uji (persamaan 2.30).
- Rata-rata adalah jumlah keseluruhan *Avalanche Effect (%)* yang dihasilkan oleh setiap data dibagi dengan banyaknya data yang diujikan.

3.5 Contoh Perhitungan Matematis Metode GOST

Sebagai contoh perhitungan matematis, digunakan sebuah *file* teks berukuran 8 *byte* yang berisi 8 buah karakter. Kata kunci yang di*input*kan sebanyak 31 karakter untuk mengetahui proses *padding* yang dilakukan.

3.5.1 Perhitungan Matematis Proses Pembentukan Kunci

Proses pembentukan kunci ini memerlukan *input data key* dengan panjang 256 bit atau 32 buah karakter. Proses ini dapat dilihat pada contoh berikut ini.

Misalkan :
key : ‘Kriptografi Metoda GOST, Tanaya’.
Maka proses pembentukan kunci dari *key* di atas adalah sebagai berikut :

1. Kunci = ‘Kriptografi Metoda GOST, Tanaya’.
   Karena kata kunci kurang dari 32 karakter, dilakukan *padding* misalnya dengan menambahkan satu karakter spasi di akhir kata kunci.
2. Ubah kunci ke bentuk biner. Dapat dilihat pada tabel 3.4.
   Hasil konversi kunci ke bentuk biner
   = (k(1), k(2), k(3), ... , k(256))

=01001011011100100110100101110000011101000110111101100111001001100001011001100110100100100000010011010110011101000011011110110010001100001000001001101011001010111010001101111011001000110000100100000010001110100111101010011010101000010110000100000010101000110000101101110011000010111100101100001001000000

Tabel 3.4 Konversi *Key* ke Biner

No	String	Biner	No	String	Biner
1	K	01001011	**17**	d	01100100
2	r	01110010	**18**	a	01100001
3	i	01101001	**19**	spasi	00100000
4	p	01110000	**20**	G	01000111
5	t	01110100	**21**	O	01001111
6	o	01101111	**22**	S	01010011
7	g	01100111	**23**	T	01010100
8	r	01110010	**24**	,	00101100
9	a	01100001	**25**	spasi	00100000
10	f	01100110	**26**	T	01010100
11	i	01101001	**27**	a	01100001
12	spasi	00100000	**28**	n	01101110
13	M	01001101	**29**	a	01100001
14	e	01100101	**30**	y	01111001
15	t	01110100	**31**	a	01100001
16	o	01101111	**32**	spasi	00100000

3. Kelompokkan hasil yang didapat pada $K_0 - K_7$ (*storing* KSU)
   K(0)= k(32),...,k(1)= 00001110100101100100111011010010
   K(1)= k(64),...,k(33)= 01001110111001101111011000101110
   K(2)= k(96),...,k(65)= 00000100100101100110011010000110
   K(3)=k(128),...,k(97)= 11110110001011101010011010110010
   K(4)=k(160),..,k(129)=11100010000001001000011000100110
   K(5)=k(192),..,k(161)=00110100001010101100101011110010
   K(6)=k(224),..,k(193)=01110110100001100010101000000100
   K(7)=k(256),..,k(225)=00000100100001101001111010000110

4. Kunci K(0) - K(7) merupaka kunci internal yang akan digunakan dalam proses enkripsi dan dekripsi.
   PROSES ENKRIPSI
   = Putaran 0 - 7 : K(0), K(1), K(2), ... , K(7)

Putaran 8 - 15 : K(0), K(1), K(2), ... , K(7)
Putaran 16 - 23 : K(0), K(1), K(2), ... , K(7)
Putaran 24 - 31 : K(7), K(6), K(5), ... , K(0)

PROSES DEKRIPSI

= Putaran 0 - 7 : K(0), K(1), K(2), ... , K(7)
Putaran 8 - 15 : K(7), K(6), K(5), ... , K(0)
Putaran 16 - 23 : K(7), K(6), K(5), ... , K(0)
Putaran 24 - 31 : K(7), K(6), K(5), ... , K(0)

3.5.2 Perhitungan Matematis Proses Enkripsi

Proses enkripsi dari metoda *GOST* memproses *input* data *plaintext* 64 bit atau 8 karakter dengan melalui 32 tahapan putaran *(round)*. Misalkan diambil hasil pembentukan kunci di atas dan *plaintext* 'ENKRIPSI', maka proses enkripsinya adalah sebagai berikut :

PROSES ENKRIPSI - PUTARAN 0

Plaintext = 'ENKRIPSI'

T_0 = 'ENKRIPSI'

Tabel 3.5 Konversi *Plaintext* ke Biner

No	String	Biner
1	E	01000101
2	N	01001110
3	K	01001011
4	R	01010010
5	I	01001001
6	P	01010000
7	S	01010011
8	I	01001001

Konversi ke biner. Dapat dilihat pada tabel 3.5.

=0100010101001110010010110101001001001001010100000101001101001001

1. R_1 (0) = 01001010110100100111001010100010
   R_2 (0) = 10010010110010100000101010010010

2. $R_1(0) + K_0 \bmod 2^{32}$
   $R_1(0)$ = 1255305890

K(0) = 244731602

---------- +

CM_1 = 1500037492 mod 2^{32}

= 1500037492

= 01011001011010001100000101110100

3. Pecah menjadi 8 kelompok dan masukkan ke *S-Box*.

0101 = 5 = SBOX(0) = 8 = 1000
1001 = 9 = SBOX(1) = 3 = 0011
0110 = 6 = SBOX(2) = 4 = 0100
1000 = 8 = SBOX(3) = 14 = 1110
1100 = 12 = SBOX(4) = 0 = 0000
0001 = 1 = SBOX(5) = 11 = 1011
0111 = 7 = SBOX(6) = 9 = 1001
0100 = 4 = SBOX(7) = 5 = 0101

4. Hasil digabungkan kembali dan lakukan *Rotate Left Shift* sebanyak 11 kali.

$R_1(0)$ = 10000011010011100000101110010101

RLS(1)=00000110100111000001011100101011
RLS(2)=00001101001110000010111001010110
RLS(3)=00011010011100000101110010101100
RLS(4)=00110100111000001011100101011000
RLS(5)=01101001110000010111001010110000
RLS(6)=11010011100000101110010101100000
RLS(7)=10100111000001011100101011000001
RLS(8)=01001110000010111001010110000011
RLS(9)=10011100000101110010101100000110
RLS(10)=00111000001011100101011000001101
RLS(11)=01110000010111001010110000011010

5. $CM_2 = R_{1\ (Hasil\ Rotasi)}$ (0) XOR R_2 (0)

R_1(0) = 01110000010111001010110000011010
R_2(0) = 10010010110010100000101010010010

---------- XOR

CM_2 = 11100010100101101010011010001000

6. $R_1(1) = CM_2$

$R_2(1) = R_1(0)$ sebelum proses.

= 01001010110100100111001010100010

PROSES ENKRIPSI - PUTARAN 1

1. $R_2(1)$ = 01001010110100100111001010100010
   $R_1(1)$ = 11100010100101101010011010001000
2. $R_1(1) + K(1) \bmod 2^{32}$
   $R_1(1)$ = 3801523848
   $K(1)$ = 1323759150
   ---------- +
   CM_1 = $5125282998 \bmod 2^{32}$
   = 830315702
   = 00110001011111011001110010110110
3. Pecah menjadi 8 kelompok dan masukkan ke *S-Box*.
   0011 = 3 = SBOX(0) = 2 = 0010
   0001 = 1 = SBOX(1) = 11 = 1011
   0111 = 7 = SBOX(2) = 2 = 0010
   1101 = 13 = SBOX(3) = 2 = 0010
   1001 = 9 = SBOX(4) = 10 = 1010
   1100 = 12 = SBOX(5) = 9 = 1001
   1011 = 11 = SBOX(6) = 7 = 0111
   0110 = 6 = SBOX(7) = 10 = 1010
4. Hasil digabungkan kembali dan lakukan *Rotate Left Shift* sebanyak 11 kali.
   $R_1(1)$ = 00101011001000101010100101111010

   RLS(1)=01010110010001010101001011110100
   RLS(2)=10101100100010101010010111101000
   RLS(3)=01011001000101010100101111010001
   RLS(4)=10110010001010101001011110100010
   RLS(5)=01100100010101010010111101000101
   RLS(6)=11001000101010100101111010001010
   RLS(7)=10010001010101001011110100010101
   RLS(8)=00100010101010010111101000101011
   RLS(9)=01000101010100101111010001010110
   RLS(10)=10001010101001011110100010101100
   RLS(11)=00010101010010111101000101011001
5. $CM_2 = R_{1\ (Hasil\ Rotasi)}(1)$ XOR $R_2(1)$
   $R_1(1)$ = 00010101010010111101000101011001
   $R_2(1)$ = 01001010110100100111001010100010
   ---------- XOR
   CM_2 = 01011111100110011010001111111011

6. $R_1(2) = CM_2$
   $R_2(2) = R_1(1)$ sebelum proses.
   = 11100010100101110101001101000100 0

Langkah-langkah enkripsi untuk putaran ke dua sampai dengan putaran ke enam adalah sama dengan langkah-langkah enkripsi pada putaran ke-0 dan ke-1. Hanya saja penggunaan kunci internal untuk putaran kedua sampai keenam adalah K(2) sampai dengan K(6).

PROSES ENKRIPSI - PUTARAN 7

1. $R_2(7)$ = 01000111101101000010110101001101
   $R_1(7)$ = 00000001001011100100011111011111
2. $R_1(7) + K(7) \bmod 2^{32}$
   $R_1(7)$ = 19810271
   K(7) = 75931270
   ---------- +
   CM_1 = 95741541 $\bmod 2^{32}$
   = 95741541
   = 00000101101101001110011001100101
3. Pecah menjadi 8 kelompok dan masukkan ke *S-Box*.
   0000 = 0 = SBOX(0) = 4 = 0100
   0101 = 5 = SBOX(1) = 13 = 1101
   1011 = 11 = SBOX(2) = 7 = 0111
   0100 = 4 = SBOX(3) = 0 = 0000
   1110 = 14 = SBOX(4) = 11 = 1011
   0110 = 6 = SBOX(5) = 1 = 0001
   0110 = 6 = SBOX(6) = 5 = 0101
   0101 = 5 = SBOX(7) = 7 = 0111
4. Hasil digabungkan kembali dan lakukan *Rotate Left Shift* sebanyak 11 kali.
   $R_1(7)$ = 01001101011100001011000101010111

   RLS(1)=10011010111000010110001010101110
   RLS(2)=00110101110000101100010101011101
   RLS(3)=01101011100001011000101010111010
   RLS(4)=11010111000010110001010101110100
   RLS(5)=10101110000101100010101011101001
   RLS(6)=01011100001011000101010111010011
   RLS(7)=10111000010110001010101110100110
   RLS(8)=01110000101100010101011101001101
   RLS(9)=11100001011000101010111010011010

RLS(10)=11000010110001010101110100110101
RLS(11)=10000101100010101011101001101011

5. $CM_2 = R_{1\ (Hasil\ Rotasi)}(7)$ XOR $R_2(7)$
   $R_1(7)$ = 10000101100010101011101001101011
   $R_2(7)$ = 01000111101101000010110101001101
   ---------- XOR
   CM_2 = 11000010001111101001011100100110
6. $R_1(8) = CM_2$
   $R_2(8) = R_1(1)$ sebelum proses.
   = 00000001001011100100011111011111

Untuk putaran ke-8 sampai dengan putaran ke-23 menggunakan kunci K(0) sampai dengan K(7) dengan cara yang sama seperti pada langkah-langkah sebelumnya.

PROSES ENKRIPSI - PUTARAN 8

1. $R_2(8)$ = 00000001001011100100011111011111
   $R_1(8)$ = 11000010001111101001011100100110
2. $R_1(8) + K(0) \bmod 2^{32}$
   $R_1(8)$ = 3258881830
   K(0) = 244731602
   ---------- +
   CM_1 = $3503613432 \bmod 2^{32}$
   = 3503613432
   = 11010000110101001110010111111000
3. Pecah menjadi 8 kelompok dan masukkan ke *S-Box*.
   1101 = 13 = SBOX(0) = 15 = 1111
   0000 = 0 = SBOX(1) = 14 = 1110
   1101 = 13 = SBOX(2) = 0 = 0000
   0100 = 4 = SBOX(3) = 0 = 0000
   1110 = 14 = SBOX(4) = 11 = 1011
   0101 = 5 = SBOX(5) = 2 = 0010
   1111 = 15 = SBOX(6) = 12 = 1100
   1000 = 8 = SBOX(7) = 9 = 1001
4. Hasil digabungkan kembali dan lakukan Rotate Left Shift sebanyak 11 kali.
   $R_1(8)$ = 11111110000000001011001011001001

   RLS(1)=11111100000000010110010110010011
   RLS(2)=11111000000000101100101100100111

RLS(3)=11110000000001011001011001001111
RLS(4)=11100000000010110010110010011111
RLS(5)=11000000000101100101100100111111
RLS(6)=10000000001011001011001001111111
RLS(7)=00000000010110010110010011111111
RLS(8)=00000000101100101100100111111110
RLS(9)=00000001011001011001001111111100
RLS(10)=00000010110010110010011111111000
RLS(11)=00000101100101100100111111110000

5. $CM_2 = R_{1\ (Hasil\ Rotasi)}\ (8)$ XOR $R_2\ (8)$

$R_1(8)$ = 00000101100101100100111111110000
$R_2(8)$ = 00000001001011100100011111011111
---------- XOR
CM_2 = 00000100101110000000100000101111

6. $R_1(9) = CM_2$

$R_2(9) = R_1$ (1) sebelum proses.
= 11000010001111101001011100100110

0
0
0

Untuk putaran ke-24 sampai dengan putaran ke-31, digunakan kunci internal K(7) sampai dengan K(0)

PROSES ENKRIPSI - PUTARAN 24

1. $R_2(24)$ = 11010101011011011001011010110111
$R_1(24)$ = 00110010101110011100101110101100
2. $R_1(24) + K(7) \bmod 2^{32}$

$R_1(24)$ = 851037100
K(7) = 75931270
---------- +
CM_1 = 926968370 mod 2^{32}
= 926968370
= 00110111010000000110101000110010

3. Pecah menjadi 8 kelompok dan masukkan ke *S-Box*.

0011 = 3 = SBOX(0) = 2 = 0010
0111 = 7 = SBOX(1) = 10 = 1010
0100 = 4 = SBOX(2) = 10 = 1010
0000 = 0 = SBOX(3) = 7 = 0111

0110 = 6 = SBOX(4) = 13 = 1101
1010 = 10 = SBOX(5) = 8 = 1000
0011 = 3 = SBOX(6) = 1 = 0001
0010 = 2 = SBOX(7) = 13 = 1101

4. Hasil digabungkan kembali dan lakukan Rotate Left Shift sebanyak 11 kali.

$R_1(24)$ = 001010101010011111011000000011101
RLS(1)=010101010100111110110000000111010
RLS(2)=101010101001111101100000001110100
RLS(3)=010101010011111011000000011101001
RLS(4)=101010100111110110000000111010010
RLS(5)=010101001111101100000001110100101
RLS(6)=101010011111011000000011101001010
RLS(7)=010100111110110000000111010010101
RLS(8)=101001111101100000001110100101010
RLS(9)=010011111011000000011101001010101
RLS(10)=100111110110000000111010010101010
RLS(11)=001111101100000001110100101010101

5. $CM_2 = R_{1\ (Hasil\ Rotasi)}\ (24)$ XOR R_2 (24)
$R_1(24)$ = 001111101100000011101001010101
$R_2(24)$ = 110101010110110110010110101101 11
---------- XOR
CM_2 = 111010111010110101111111111100010

6. $R_1(25) = CM_2$
$R_2(25) = R_1(1)$ sebelum proses.
= 001100101011100111001011010 1100
0
0
0

PROSES ENKRIPSI - PUTARAN 31

1. $R_1(31)$ = 000100110101110101111110100110101
$R_2(31)$ = 001101000000111101010010111101 01

2. $R_1(31) + K(0) \bmod 2^{32}$
$R_1(31)$ = 324894005
$K(0)$ = 244731602
---------- +
CM_1 = 569625607 $\bmod 2^{32}$
= 569625607
= 00100001111100111100110000000111

3. Pecah menjadi 8 kelompok dan masukkan ke SBox.
   0010 = 2 = SBOX(0) = 9 = 1001
   0001 = 1 = SBOX(1) = 11 = 1011
   1111 = 15 = SBOX(2) = 11 = 1011
   0011 = 3 = SBOX(3) = 1 = 0001
   1100 = 12 = SBOX(4) = 0 = 0000
   1100 = 12 = SBOX(5) = 9 = 1001
   0000 = 0 = SBOX(6) = 13 = 1101
   0111 = 7 = SBOX(7) = 4 = 0100
4. Hasil digabungkan kembali dan lakukan Rotate Left Shift sebanyak 11 kali.
   $R_1(31)$ = 10011011101100010000100111010100
   RLS(1)=00110111011000100001001110101001
   RLS(2)=01101110110001000010011101010010
   RLS(3)=11011101100010000100111010100100
   RLS(4)=10111011000100001001110101001001
   RLS(5)=01110110001000010011101010010011
   RLS(6)=11101100010000100111010100100110
   RLS(7)=11011000100001001110101001001101
   RLS(8)=10110001000010011101010010011011
   RLS(9)=01100010000100111010100100110111
   RLS(10)=11000100001001110101001001101110
   RLS(11)=10001000010011101010010011011101
5. $R_1(32) = R_1(31)$ sebelum proses.
   = 00010011010111010111110100110101
6. $CM_2 = R_{1\ (Hasil\ Rotasi)}$ (31) XOR R_2 (31)
   R_1 (31) = 10001000010011101010010011011101
   R_2 (31) = 00110100000011110101001011110101
   ---------- XOR
   CM_2 = 10111100010000011111011000101000

   $R_2(32) = CM_2 = b(32), b(31), ... b(1)$
   $R_1(32) = a(32), a(31), ... a(1)$
   Ciphertext = a(1), ... a(32), b(1), ... b(32)
   Ciphertext dalam biner
   =10101100101111101011101011001000000101000110111110
   00001000111101 = ubah ke ASCII
   = ¼╛ ‖ ╚¶oé=

Karakter "¼┘ ╢ ╚¶oé=" merupakan *ciphertext* hasil dari proses enkripsi *plaintext* "ENKRIPSI" menggunakan kata kunci 'Kriptografi Metoda GOST, Tanaya'. Jumlah *padding* yang dilakukan adalah satu karakter terhadap kata kunci. Jumlah karakter hasil enkripsi sama dengan jumlah karakter *ciphertext*, yaitu sebanyak delapan karakter. Perubahan bentuk biner pada tiap karakter dapat dilihat pada tabel 3.6.

Tabel 3.6 Perubahan Bentuk Biner Karakter Hasil Enkripsi

No	*Plaintext*		*Ciphertext*	
	String	**Biner**	**String**	**Biner**
1	E	01000101	¼	10101100
2	N	01001110	┘	10111110
3	K	01001011	╢	10111010
4	R	01010010	╚	11001000
5	I	01001001	¶	00010100
6	P	01010000	o	01101111
7	S	01010011	é	10000010
8	I	01001001	=	00111101

3.5.3 Perhitungan Matematis Proses Dekripsi

Proses dekripsi merupakan kebalikan dari proses enkripsi. Proses dekripsi dari metoda GOST menggunakan algoritma yang sama dengan proses enkripsi. Misalkan diambil hasil pembentukan kunci dan *ciphertext* di atas, maka proses dekripsinya adalah sebagai berikut :

PROSES DEKRIPSI - PUTARAN 0
Ciphertext = "¼┘ ╢ ╚¶oé="
T_e = ¼┘ ╢ ╚¶oé=

Konversi *ciphertext* ke biner. Dapat dilihat pada Tabel 3.7
=10101100101111101011101011001000000101000110111100000100011101

1. R_1 (0) = 00010011010111010111110100110101
   R_2 (0) = 10111100010000011111011000101000

Tabel 3.7 Konversi *Ciphertext* ke Biner

No	String	Biner
1	¼	10101100
2	╡	10111110
3	╢	10111010
4	╚	11001000
5	¶	00010100
6	o	01101111
7	é	10000010
8	=	00111101

2. $R_1(0) + K(0) \bmod 2^{32}$
   $R_1(0)$ = 324894005
   K(0) = 244731602
   ---------- +
   CM_1 = 569625607 mod 2^{32}
   = 569625607
   = 00100001111100111100110000000111
3. Pecah menjadi 8 kelompok dan masukkan ke *S-Box*.
   0010 = 2 = SBOX(0) = 9 = 1001
   0001 = 1 = SBOX(1) = 11 = 1011
   1111 = 15 = SBOX(2) = 11 = 1011
   0011 = 3 = SBOX(3) = 1 = 0001
   1100 = 12 = SBOX(4) = 0 = 0000
   1100 = 12 = SBOX(5) = 9 = 1001
   0000 = 0 = SBOX(6) = 13 = 1101
   0111 = 7 = SBOX(7) = 4 = 0100
4. Hasil digabungkan kembali dan lakukan *Rotate Left Shift* sebanyak 11 kali.
   $R_1(0)$ = 10011011101100010000100111010100

   RLS(1)=00110111011000100001001110101001
   RLS(2)=01101110110001000010011101010010
   RLS(3)=11011101100010000100111010100100
   RLS(4)=10111011000100001001110101001001
   RLS(5)=01110110001000010011101010010011
   RLS(6)=11101100010000100111010100100110
   RLS(7)=11011000100001001110101001001101

RLS(8)=10110001000010011101010010011011
RLS(9)=01100010000100111010100100110111
RLS(10)=11000100001001110101001001101110
RLS(11)=10001000010011101010010011011101

5. $CM_2 = R_{1\ (Hasil\ Rotasi)}\ (0)$ XOR $R_2\ (0)$
   $R_1(\ 0)$ = 10001000010011101010010011011101
   $R_2(\ 0)$ = 10111100010000011111011000101000
   ---------- XOR
   CM_2 = 00110100000011110101001011110101
6. $R_1(1) = CM_2$
   $R_2(1) = R_1(0)$ sebelum proses.
   = 00010011010111010111110100110101

PROSES DEKRIPSI - PUTARAN 1

1. $R_2(1)$ = 00010011010111010111110100110101
   $R_1(1)$ = 00110100000011110101001011110101
2. $R_1(1) + K(1) \bmod 2^{32}$
   $R_1(1)$ = 873419509
   $K(1)$ = 1323759150
   ---------- +
   CM_1 = 2197178659 mod 2^{32}
   = 2197178659
   = 10000010111101100100100100100011
3. Pecah menjadi 8 kelompok dan masukkan ke *S-Box*.
   1000 = 8 = SBOX(0) = 6 = 0110
   0010 = 2 = SBOX(1) = 4 = 0100
   1111 = 15 = SBOX(2) = 11 = 1011
   0110 = 6 = SBOX(3) = 9 = 1001
   0100 = 4 = SBOX(4) = 5 = 0101
   1001 = 9 = SBOX(5) = 6 = 0110
   0010 = 2 = SBOX(6) = 4 = 0100
   0011 = 3 = SBOX(7) = 0 = 0000
4. Hasil digabungkan kembali dan lakukan Rotate Left Shift sebanyak 11 kali.
   $R_1(1)$ = 01100100101110010101011001000000

   RLS(1)=11001001011100101010110010000000
   RLS(2)=10010010111001010101100100000001

RLS(3)=00100101110010101011001000000011
RLS(4)=01001011100101010110010000000110
RLS(5)=10010111001010101100100000001100
RLS(6)=00101110010101011001000000011001
RLS(7)=01011100101010110010000000110010
RLS(8)=10111001010101100100000001100100
RLS(9)=01110010101011001000000011001001
RLS(10)=11100101010110010000000110010010
RLS(11)=11001010101100100000001100100101

5. $CM_2 = R_{1\ (Hasil\ Rotasi)}\ (1)$ XOR $R_2\ (1)$
   $R_1(1)$ = 11001010101100100000001100100101
   $R_2(1)$ = 00010011010111010111110100110101
   ---------- XOR
   CM_2 = 11011001111011110111111000010000
6. $R_1(2) = CM_2$
   $R_2(2) = R_1\ (1)$ sebelum proses.
   = 00110100000011110101001011110101

o
o
o

Untuk putaran ke-0 sampai dengan putaran ke-7, kunci internal yang digunakan adalah K(0) sampai dengan K(7).

PROSES DEKRIPSI - PUTARAN 7

1. $R_1\ (7)$ = 00110010101110011100101110101100
   $R_2\ (7)$ = 11101011101011010111111111100010
2. $R_1(7) + K(7) \bmod 2^{32}$
   $R_1(7)$ = 851037100
   K(7) = 75931270
   ---------- +
   CM_1 = 926968370 $\bmod 2^{32}$
   = 926968370
   = 00110111010000000110101000110010
3. Pecah menjadi 8 kelompok dan masukkan ke *S-Box*.
   0011 = 3 = SBOX(0) = 2 = 0010
   0111 = 7 = SBOX(1) = 10 = 1010
   0100 = 4 = SBOX(2) = 10 = 1010
   0000 = 0 = SBOX(3) = 7 = 0111
   0110 = 6 = SBOX(4) = 13 = 1101
   1010 = 10 = SBOX(5) = 8 = 1000

0011 = 3 = SBOX(6) = 1 = 0001
0010 = 2 = SBOX(7) = 13 = 1101

4. Hasil digabungkan kembali dan lakukan *Rotate Left Shift* sebanyak 11 kali.

$R_1(7)$ = 00101010101001111101100000011101

RLS(1)=01010101010011111011000000111010
RLS(2)=10101010100111110110000001110100
RLS(3)=01010101001111101100000011101001
RLS(4)=10101010011111011000000111010010
RLS(5)=01010100111110110000001110100101
RLS(6)=10101001111101100000011101001010
RLS(7)=01010011111011000000111010010101
RLS(8)=10100111110110000001110100101010
RLS(9)=01001111101100000011101001010101
RLS(10)=10011111011000000111010010101010
RLS(11)=00111110110000001110100101010101

5. $CM_2 = R_{1\ (Hasil\ Rotasi)}(7)$ XOR $R_2(7)$

$R_1(7)$ = 00111110110000001110100101010101
$R_2(7)$ = 11101011101011010111111111100010
---------- XOR
CM_2 = 11010101011011011001011010110111

6. $R_1(8) = CM_2$
$R_2(8) = R_1(7)$ sebelum proses.
= 00110010101110011100101110101100

PROSES DEKRIPSI - PUTARAN 8

1. $R_1(8)$ = 11010101011011011001011010110111
$R_2(8)$ = 00110010101110011100101110101100
2. $R_1(8) + K(7) \bmod 2^{32}$

$R_1(8)$ = 3580729015
K(7) = 75931270
---------- +
CM_1 = 3656660285 mod 2^{32}
= 3656660285
= 11011001111101000011010100111101

3. Pecah menjadi 8 kelompok dan masukkan ke *S-Box*.

1101 = 13 = SBOX(0) = 15 = 1111
1001 = 9 = SBOX(1) = 3 = 0011

1111 = 15 = SBOX(2) = 11 = 1011
0100 = 4 = SBOX(3) = 0 = 0000
0011 = 3 = SBOX(4) = 1 = 0001
0101 = 5 = SBOX(5) = 2 = 0010
0011 = 3 = SBOX(6) = 1 = 0001
1101 = 13 = SBOX(7) = 11 = 1011

4. Hasil digabungkan kembali dan lakukan *Rotate Left Shift* sebanyak 11 kali.

$R_1(8)$ = 11110011101100000001001000011011

RLS(1)=11100111011000000010010000110111
RLS(2)=11001110110000000100100001101111
RLS(3)=10011101100000001001000011011111
RLS(4)=00111011000000010010000110111111
RLS(5)=01110110000000100100001101111110
RLS(6)=11101100000001001000011011111100
RLS(7)=11011000000100100001101111111001
RLS(8)=10110000000100100001101111110011
RLS(9)=01100000001001000011011111100111
RLS(10)=11000000010010000110111111001110
RLS(11)=10000000100100001101111110011101

5. $CM_2 = R_{1\ (Hasil\ Rotasi)}(8)$ XOR $R_2(8)$

$R_1(8)$ = 10000000100100001101111110011101
$R_2(8)$ = 00110010101110011100101110101100
---------- XOR
CM_2 = 10110010001010010001010000110001

6. $R_1(9) = CM_2$
$R_2(9) = R_1(8)$ sebelum proses.
= 11010101011011011001011010110111

o
o
o

Untuk putaran ke-8 sampai dengan putaran ke-31, kunci internal yang digunakan secara berturut-turut adalah dari K(7) sampai dengan K(0).

PROSES DEKRIPSI - PUTARAN 31

1. $R_2(31)$ = 11100010100101101010011010001000
$R_1(31)$ = 01001010110100100111001010100010

2. $R_1(31) + K(0) \bmod 2^{32}$

   $R_1(31)$ = 1255305890
   K(0) = 244731602
   ---------- +
   CM_1 = 1500037492 mod 2^{32}
   = 1500037492
   = 01011001011010001100000101110100

3. Pecah menjadi 8 kelompok dan masukkan ke SBox.

   0101 = 5 = SBOX(0) = 8 = 1000
   1001 = 9 = SBOX(1) = 3 = 0011
   0110 = 6 = SBOX(2) = 4 = 0100
   1000 = 8 = SBOX(3) = 14 = 1110
   1100 = 12 = SBOX(4) = 0 = 0000
   0001 = 1 = SBOX(5) = 11 = 1011
   0111 = 7 = SBOX(6) = 9 = 1001
   0100 = 4 = SBOX(7) = 5 = 0101

4. Hasil digabungkan kembali dan lakukan Rotate Left Shift sebanyak 11 kali.

   $R_1(31)$ = 10000011010011100000101110010101

   RLS(1)=00000110100111000001011100101011
   RLS(2)=00001101001110000010111001010110
   RLS(3)=00011010011100000101110010101100
   RLS(4)=00110100111000001011100101011000
   RLS(5)=01101001110000010111001010110000
   RLS(6)=11010011100000101110010101100000
   RLS(7)=10100111000001011100101011000001
   RLS(8)=01001110000010111001010110000011
   RLS(9)=10011100000101110010101100000110
   RLS(10)=00111000001011100101011000001101
   RLS(11)=01110000010111001010110000011010

5. $R_1(32) = R_1(31)$ sebelum proses.
   = 01001010110100100111001010100010

6. $CM_2 = R_{1\,(\text{Hasil Rotasi})}(31)$ XOR $R_2(31)$

   $R_1(31)$ = 01110000010111001010110000011010
   $R_2(31)$ = 11100010100101101010011010001000
   ---------- XOR
   CM_2 = 10010010110010100000101010010010

$R_2(32) = CM_2 = b(32), b(31), ... b(1)$
$R_1(32) = a(32), a(31), ... a(1)$
Plaintext = a(1), ... a(32), b(1), ... b(32)
Plaintext dalam biner :
=01000101010011100100101101010010010010010101000001
01001101001001
= ubah ke ASCII
= ENKRIPSI

Tabel 3.8 Perubahan Bentuk Biner Karakter Hasil Dekripsi

No	*Ciphertext*		*Plaintext*	
	String	**Biner**	**String**	**Biner**
1	¼	10101100	E	01000101
2	╛	10111110	N	01001110
3	╗	10111010	K	01001011
4	└	11001000	R	01010010
5	¶	00010100	I	01001001
6	o	01101111	P	01010000
7	é	10000010	S	01010011
8	=	00111101	I	01001001

Karakter "ENKRIPSI" merupakan hasil proses dekripsi dari *ciphertext* "¼╛ ╗ └¶oé=" dengan menggunakan kunci yang sama untuk proses enkripsi yaitu 'Kriptografi Metoda GOST, Tanaya'. Jumlah karakter hasil dekripsi sama dengan jumlah karakter *ciphertext* dan merupakan karakter-karakter *plaintext* awal sebelum dilakukan proses enkripsi dan dekripsi. Hal ini berarti bahwa, proses enkripsi dan dekripsi telah berhasil dilakukan, dengan *output* hasil dekripsi sama dengan karakter awal. Perubahan bentuk biner pada tiap karakter dapat dilihat pada tabel 3.8.


UNIVERSITAS BRAWIJAYA

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

Pada bab implementasi dan pembahasan akan dilakukan implementasi perangkat lunak, serta analisis hasil yang dikeluarkan oleh perangkat lunak.

4.1 Lingkungan Implementasi

Lingkungan implementasi meliputi lingkungan perangkat keras serta lingkungan perangkat lunak.

4.1.1 Lingkungan Perangkat Keras

Perangkat keras yang digunakan dalam penyusunan perangkat lunak enkripsi dan dekripsi metode *GOST* ini adalah :

1. *Processor Mobile Intel*(R) *Pentium*(R) 4 CPU 2.40GHz.
2. RAM 1.00 GB
3. *Harddisk* dengan kapasitas 80 GB
4. Monitor
5. *Keyboard*
6. *Mouse*

4.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan sistem ini adalah :

1. Sistem Operasi *Microsoft Windows XP Professional*
2. *Borland Delphi 7.0*

4.2 Implementasi Program

Berdasarkan perancangan perangkat lunak pada subbab 3.2 maka pada subbab ini akan dibahas mengenai implementasi dari perancangan tersebut.

4.2.1 Proses *Input File*

Input plain-file dapat diperoleh dari *input*an *user* secara langsung pada waktu *running program*. Pada proses *input*, *file* atau *file .gef* dibuka dari sebuah komponen *OpenDialog*. *Sourcecode* fungsi *input plain-file* dapat dilihat pada *sourcecode* 4.1.

FromF dideklarasikan sebagai sebuah variabel *file*, sesuai dengan *file input* yang dipilih. Setelah *file* dibuka dilakukan proses

pembacaan *file*. Proses pembacaan ini dilakukan dengan menggunakan prosedur *BlockRead*. Penggunaan prosedur *BlockRead* dianggap lebih mudah dan lebih sesuai untuk perangkat lunak yang dibangun. *BlockRead* dapat dioperasikan untuk semua tipe *file*, selain itu dengan *BlockRead* dapat ditentukan besarnya blok pembacaan data yang diinginkan.

```
procedure TForm1.ImageOpenClick(Sender: TObject);
var
  FromF: file;
  NumRead: Integer;
  Buf:array [1..8] of Char;
  i:integer;
begin
    if OpenDialog1.Execute then
    begin
        statusOpenF:=true;
        AssignFile(FromF, OpenDialog1.FileName);
        Reset(FromF, 1);
        hasil:='';
        repeat
            BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
            for i:=1 to numread do
                Hasil:=Hasil+buf[i];
        until (numread <> 8);
        eks:= rightstr(opendialog1.FileName,length
              (opendialog1.FileName)- Pos('.',
               opendialog1.FileName));
        CloseFile(FromF);
    end
    else
        statusOpenF:=False;
end;
```

***Sourcecode* 4.1** *Sourcecode* Prosedur *ImageOpenClick.*

Buf pada prosedur *BlockRead* dideklarasikan sebagai sebuah *array[1..8] of Char* yang berarti bahwa blok pembacaan data yang dilakukan adalah tiap 8 *byte* = 8 karakter = 64 bit dari *file FromF*. Hal ini sesuai dengan aturan metode *GOST*, yaitu pemrosesan dilakukan tiap 64 bit *block cipher*. *Output* dari fungsi *Open Plain-file* adalah data pembacaan dari *file input* yang disimpan dalam variabel Hasil.

4.2.2 Proses Pembentukan Kunci

Proses pembentukan kunci memerlukan *input* data berupa karakter-karakter kunci. Pembentukan kunci berlangsung pada fungsi Bentuk_Kunci. Sebelum dibentuk menjadi sebuah kunci, karakter-karakter *input user* untuk *key* terlebih dahulu divalidasi melalui fungsi *Padding*. *Padding* dilakukan dengan menambahkan karakter #0 (NULL) sebanyak kekurangan yang ada. Pada akhir fungsi *Padding*, karakter kunci dirubah ke dalam bentuk biner melalui fungsi *RealToBin*. Hasil biner dari kata kunci ini sebanyak 256 bit dan disimpan dalam variabel Hasil.

Hasil biner 256 bit kemudian dikelompokkan sesuai dengan aturan pembentukan kunci pada subbab 2.4.2. *Output* dari fungsi Bentuk_Kunci adalah delapan buah *key* internal. *Sourcecode fungsi Padding* dan prosedur Bentuk_Kunci dapat dilihat pada *sourcecode* 4.2 dan 4.3.

```
Function TForm1.Padding(Jenis:byte;text:string):String;
// Jenis 1 untuk padding text
// Jenis 2 untuk padding kunci
begin
    JumlPadding:=0;
    if jenis=1 then
        JumlHuruf:=8
    else
    begin
        JumlHuruf:=32;
        if length(text)>32 then
            text:=leftstr(text,32);
    end;
    if (length(text)mod jumlhuruf) > 0 then
        JumlPadding:=jumlhuruf-(length(text)mod
              jumlhuruf);
    if length(text)mod jumlhuruf >0 then
    begin
        For i:= 1 to jumlhuruf-(length(text)mod
              jumlhuruf) do
            text:=text + #0;
    end;
    Hasil:='';
    For i:=1 to length(text) do
        Hasil:=Hasil + Realtobin(ord(text[i]));
    Padding:=Hasil;
```

***Sourcecode* 4.2** *Sourcecode* Fungsi *Padding*

4.2.3 Proses Enkripsi

Proses enkripsi metode *GOST* dilakukan sebanyak 32 putaran (*round*), untuk setiap putaran, langkah yang dilakukan seperti telah dijelaskan pada subbab 2.4.3. Dari blok pertama yang didapatkan (T_0) (persamaan 2.5) dibentuk register awal R_1 dan R_2 (persamaan 2.6 dan 2.7).

Kemudian dilakukan penjumlahan modulo antara R_1 dan kunci internal sesuai dengan kaidah *GOST* (*sourcecode* 4.3). Hasil akhir penjumlahan modulo disimpan dalam variabel *cm1* bertipe *string* (persamaan 2.8). *Sourcecode* operasi penjumlahan modulo 2^{32} dapat dilihat pada *sourcecode* 4.4.

```
Procedure TForm1.Bentuk_Kunci(Text: string);
var
  i:integer;
  Temp:string;
  index:integer;
begin
    i:=1;
    index:=0;
    while i<length(text)do
    begin
        Temp:=midstr(text,i,32);
        temp:=ReverseString(temp);
        k[index]:=temp;
        inc(i,32);
        inc(index);
    end;
end;
```

***Sourcecode* 4.3** *Sourcecode* Prosedur Bentuk_Kunci

Substitusi *S-Box* pada metode *GOST* dilakukan dengan membagi biner *cm1* menjadi delapan bagian, masing-masing empat bit. Kemudian dimasukkan ke *S-Box* yang bersesuaian. Pada implementasi program, *S-Box* metode *GOST* dideklarasikan sebagai sebuah konstanta.

Output dari proses substitusi *S-Box* adalah *cm1* yang merupakan hasil penggabungan dari tiap empat bit *S-Box*. *Sourcecode* operasi substitusi *S-Box* dapat dilihat pada *sourcecode* 4.5.

```
    i:=1;
    Bentuk_kunci(Padding(2,Kunci));
    text:=Padding(1,Text);
    while i<length(text)do
    begin
        Temp:=midstr(text,i,64);
        R1[0]:=ReverseString(midstr(temp,1,32));
        R2[0]:=ReverseString(midstr(temp,33,64));
        j:=0;m:=0;
        while j<=31 do
        begin
            if (m>7) and (j<24) then m:=0;
            if j=24 then m:=7;
            cm1:=realtobin((bintoreal(R1[j])+
                    bintoreal(K[m]))mod Modulo);
            tempbin:='';
            for l:=1 to 32-(length(cm1)) do
                Tempbin:=tempbin+'0';
            cm1:=tempbin+cm1;
```

***Sourcecode* 4.4** *Sourcecode* Operasi Penjumlahan Modulo 2^{32}

Setelah melalui proses substitusi *S-Box*, data diolah dalam fungsi RLS. Fungsi RLS (*Rotate Left Shift*) menggunakan fungsi Delphi *shl* yang akan menggeser satu bit ke arah kiri. Pergeseran bit ini dilakukan sebanyak 11 bit. *Sourcecode* fungsi *Rotate Left Shift* dapat dilihat pada *sourcecode* 4.6.

```
// Konstanta S-Box
const
SBox : array[1..8,0..15] of byte = (
  (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)
);
// Substitusi S-Box
cm:='';
for l:=1 to 8 do
     cm:=cm+realtobin4(sbox[l,bintoreal(midstr(cm1,(l-
1)*4+1,4))]);
cm1:=cm;
```

***Sourcecode* 4.5** *Sourcecode* Operasi Substitusi *S-Box*

HasilRLS digunakan untuk menyimpan hasil dari rotasi 11 bit yang dilakukan. Selanjutnya, hasil RLS di-XOR-kan dengan register R_2 sesuai dengan persamaan 2.9. Hasil XOR disimpan sebagai bit biner dalam variabel *cm2*. *cm2* merupakan register R_1 untuk putaran selanjutnya (persamaan 2.10), sedangkan register R_1 sebelum proses digunakan sebagai register R_2 untuk putaran selanjutnya (persamaan 2.11). Pada akhir putaran (putaran 31), register $R_1[31]$ merupakan register $R_1[32]$ (persamaan 2.16), dan *cm2* merupakan register $R_2[32]$ (persamaan 2.17). Pasangan $R_1[32]$ dan $R_2[32]$ merupakan pasangan *cipher* (persamaan 2.18) yang disimpan dalam variabel *chipertext1*. *Sourcecode* operasi *XOR* dan *CM2* dapat dilihat pada *sourcecode* 4.7.

```
Function TForm1.RLS(Angka:longWord):LongWord;
Var
   i:integer;
   Temp:longWord;
begin
    for i:=1 to 11 do
    begin
        if Angka >= pangkat(2,31) then
            Temp:=((Angka-pangkat(2,31))shl 1)+1
        else
            Temp:=Angka shl 1;
        Angka:=temp;
    end;
    RLS:=Angka;
end;
//fungsi RLS dipanggil dalam fungsi enkripsi
HasilRLS:=RLS(bintoreal(cm1));
```

***Sourcecode* 4.6** *Sourcecode* Fungsi RLS (*Rotate Left Shift*)

Pasangan *cipher* $R_1[32]$ dan $R_2[32]$ yang terbentuk masih dalam rangkaian digit biner. Untuk itu diperlukan sebuah proses untuk merubah digit-digit biner ini menjadi sebuah karakter. Dari deretan digit biner, dilakukan proses *midstr* untuk setiap delapan digit, kemudian digunakan fungsi *BinToReal* untuk mengubahnya ke dalam bentuk real. Fungsi *chr* digunakan untuk mengetahui karakter dari sebuah nilai real yang dimasukkan. *Cipher* dalam bentuk karakter-karakter disimpan dalam variabel *chipertext*. *Sourcecode* operasi pembentukan *ciphertext* dapat dilihat pada *sourcecode* 4.8.

```
cm2:= realtobin(HasilRLS xor bintoreal(R2[j]));
      tempbin:='';
      for l:=1 to 32-(length(cm2)) do
          Tempbin:=tempbin+'0';
      cm2:=tempbin+cm2;
      inc(j);
      if j<24 then
          inc(m)
      else
          dec(m);
      R1[j]:=cm2;
      R2[j]:=R1[j-1];
   end;
   R2[32]:=cm2;
   R1[32]:=R1[31];
   ChiperText1:=ChiperText1+reversestring(R1[32])+
        reversestring(R2[32]);
   inc(i,64);
```

***Sourcecode* 4.7** *Sourcecode* Operasi XOR dan CM2

```
while i <= length(ChiperText1) do
begin
   temp:=midstr(ChiperText1,i,8);
   chipertext:=chipertext + chr(bintoreal(temp));
   inc(i,8);
end;
Enkripsi:=chipertext;
```

***Sourcecode* 4.8** *Sourcecode* Operasi Pembentukan *Ciphertext*

4.2.4 Proses Penyimpanan *Cipher-File*

Proses penyimpanan *cipher-file* ke dalam *file* tujuan (*file .gef*) dilakukan dengan menggunakan fungsi *BlockWrite*. Fungsi ini dipanggil setelah proses enkripsi dijalankan. Dengan menggunakan *BlockWrite*, penulisan dapat dilakukan per blok penyandian yaitu per delapan karakter.

Terdapat beberapa bit tambahan yang dituliskan ke dalam *file .gef*, yaitu delapan bit tambahan yang berisikan informasi mengenai jumlah *padding text* dan ekstensi *file* awal. Jumlah *padding text* telah dihitung sebelumnya pada fungsi *Padding* dengan perintah sebagai berikut :

```
JumlPadding:=jumlhuruf-(length(text)mod jumlhuruf);
```

sedangkan ekstensi *file* awal dapat diketahui dari variable eks pada prosedur *ImageOpenClick* dengan perintah sebagai berikut :

```
eks:= rightstr(opendialog1.FileName,length(opendialog1.
      FileName)-Pos('.',opendialog1.FileName));
```

Untuk membentuk delapan bit tambahan, dilakukan perulangan dengan perintah sebagai berikut :

```
temp:=''
for i:=1 to  (8-length(eks))-1 do
begin
   if i>1 then
         temp:= temp + #0
   else
         temp:= temp + inttostr(jumlpadding);
end;
eks:=temp+eks+'.';
hasil1:=eks+hasil1;
```

Dari perintah diatas, dihasilkan delapan bit tambahan yang berisikan jumlah *padding text* pada bit pertama, ekstensi *file* awal pada bit ke dua sampai dengan bit ke tujuh, dan *mark* yaitu karakter titik pada bit ke delapan. Delapan bit tambahan ini disimpan dalam variabel *eks*, kemudian ditambahkan dengan variabel *Hasil1* yang merupakan variabel hasil enkripsi. Dari *Hasil1* dilakukan *BlockWrite* untuk *file* tujuan (*file .gef*). *Sourcecode* penyimpanan *cipher-file* dapat dilihat pada *sourcecode* 4.9.

4.2.5 Proses Dekripsi

Secara umum fungsi-fungsi yang digunakan dalam proses dekripsi sama dengan proses enkripsi. Beberapa perbedaan terdapat pada aturan penggunaan *key* pada tiap putaran dan proses *padding*. Pada proses dekripsi *padding text* akan selalu bernilai nol. Hal ini dikarenakan proses *padding* telah dilakukan pada *cipher-file* hasil proses enkripsi, sehingga ukuran *cipher-file* telah sesuai dengan ukuran blok yang ditetapkan.

Sebelum fungsi dekripsi dijalankan, terdapat beberapa hal yang dilakukan berkaitan dengan delapan bit tambahan, antara lain :

1. Mengetahui jumlah *padding text* dari *file .gef*. Jumlah *padding text* disimpan dalam variabel *Juml1* bertipe *integer* dengan perintah

   ```
   Juml1:= strtoint(leftstr(hasil,1));
   ```

2. Mendapatkan informasi ekstensi awal dari *file*. Ekstensi awal disimpan dalam variabel *eks* dengan perintah

   ```
   hasil:= midstr(hasil,2,length(hasil)-1);
   eks:= trim(leftstr(hasil,pos('.',hasil)-1));
   ```

```
Bentuk_kunci(Padding(2,Kunci));
text:=Padding(1,Text);
Plaintext1:='';
while i<length(text)do
begin
    Temp:=midstr(text,i,64);
    R1[0]:=ReverseString(midstr(temp,1,32));
    R2[0]:=ReverseString(midstr(temp,33,64));
    j:=0;m:=0;
    while j<=31 do
    begin
        if j=0 then m:=0;
        if j=8 then m:=7;
        if (j > 7) and (m < 0) then m:=7;
        cm1:=realtobin((bintoreal(R1[j])+
                bintoreal(K[m]))mod Modulo);
        tempbin:='';
        for l:=1 to 32-(length(cm1)) do
            Tempbin:=tempbin+'0';
        cm1:=tempbin+cm1;
        cm:='';
        for l:=1 to 8 do
            cm:=cm + realtobin4(sbox[l,bintoreal
              (midstr(cm1,(l-1)*4+1,4))]);
        cm1:=cm;
        HasilRLS:=RLS(bintoreal(cm1));
        cm2:= realtobin(HasilRLS xor bintoreal
             (R2[j]));
        tempbin:='';
        for l:=1 to 32-(length(cm2)) do
            Tempbin:=tempbin+'0';
        cm2:=tempbin+cm2;
        inc(j);
        if j<8 then
            inc(m)
        else
            dec(m);
        R1[j]:=cm2;
        R2[j]:=R1[j-1];
    end;
    R2[32]:=cm2;
    R1[32]:=R1[31];
    Plaintext1:= Plaintext1+reversestring
               (R1[32])+reversestring(R2[32]);
    inc(i,64);
```

***Sourcecode* 4.10** *Sourcecode* Proses Dekripsi

```
//fungsi BlockWrite file .gef

if SaveDialog1.Execute then
begin
   AssignFile(ToF, SaveDialog1.FileName + '.GEF');
   Rewrite(ToF, 1);
   hasil1:=enkripsi(hasil,Kunci);
   temp:='';
   for i:=1 to  8-length(eks)-1 do
   begin
      if i>1 then
         temp:= temp + #0
      else
         temp:= temp + inttostr(jumlpadding);
   end;
   eks:=temp+eks+'.';
   hasil1:=eks+hasil1;
   i:=1;
   while i <= length(hasil1) do
   begin
     numread:=8;
     temp:=midstr(hasil1,i,8);
     for j:=1 to 8 do
     begin
         buf[j]:=temp[j];
     end;
     repeat
         BlockWrite(ToF,Buf, NumRead, NumWritten);
         numread:=0;
     until (NumRead <> 8) or (NumWritten <> NumRead);
     inc(i,8);
   end;
   CloseFile(ToF);
end;
```

***Sourcecode* 4.9** *Sourcecode* Proses Penyimpanan *Cipher-File*

3. Mendapatkan data hasil enkripsi dari *file .gef* dengan perintah

```
hasil:= midstr(hasil,pos('.',hasil)+1,length
        (hasil)- pos('.',hasil));
```

Variabel *Hasil* merupakan variabel yang menyimpan data hasil enkripsi.

4. Proses Dekripsi. Proses dekripsi *file .gef* dijalankan dengan memanggil fungsi dekripsi dengan parameter hasil dan kunci.

```
hasil1:=dekripsi(hasil,Kunci);
```

5. Setelah proses dekripsi dijalankan, variable *hasil1* berisi data *file* hasil dekripsi. Kemudian dijalankan perintah

```
hasil1:=leftstr(hasil1,length(hasil1)-juml1);
```

Perintah ini berfungsi untuk mengurangi panjang data *file* hasil dekripsi sebanyak jumlah *padding*. Hal ini berguna agar ukuran *file* kembali seperti ukuran awal. *Sourcecode* proses dekripsi dapat dilihat pada *sourcecode* 4.10.

Pada proses dekripsi, pasangan *plaintext* $R_1[32]$ dan $R_2[32]$ yang terbentuk juga masih dalam rangkaian digit biner. Diperlukan fungsi *midstr*, *BinToReal* dan *chr* untuk mengubah digit biner menjadi karakter. *Plaintext* dalam bentuk karakter-karakter disimpan dalam variabel *Plaintext*. *Sourcecode* Operasi Pembentukan *plaintext* dapat dilihat pada *sourcecode* 4.11.

```
while i <= length(PlainText1) do
begin
   temp:=midstr(PlainText1,i,8);
   PlainText := PlainText + chr(bintoreal(temp));
   inc(i,8);
end;
Enkripsi:=PlainText;
```

***Sourcecode* 4.11** *Sourcecode* Operasi Pembentukan *Ciphertext*

```
// pembentukan file hasil dekripsi

if SaveDialog1.Execute then
begin
    AssignFile(ToF, SaveDialog1.FileName + '.'+ eks);
    Rewrite(ToF, 1);
    i:=1;
    while i <= length(hasil1) do
    begin
       numread:=1;
       buf1:=hasil1[i];
       repeat
           BlockWrite(ToF,Buf1, NumRead, NumWritten);
           numread:=0;
       until (NumRead=0) or (NumWritten <> NumRead);
    inc(i);
    end;
    CloseFile(ToF);
end;
```

***Sourcecode* 4.12** *Sourcecode* Pembentukan *File* Hasil Dekripsi

4.2.6 Proses Pembentukan *File* Hasil Dekripsi

File hasil merupakan *file* yang dihasilkan dari proses dekripsi. *File* hasil berukuran dan berekstensi sama dengan *file* awal sebelum dilakukan proses enkripsi. Pembentukan *file* hasil diperoleh dari penulisan data hasil dekripsi ke dalam *file* melalui fungsi *BlockWrite*. *Sourcecode* pembentukan *file* hasil dekripsi dapat dilihat pada *sourcecode* 4.12.

4.3 Implementasi *Interface*

Berdasarkan rancangan *interface* pada subbab 3.3, maka dihasilkan *interface form* utama seperti pada gambar 4.1.

Gambar 4.1 *Form* Utama

Pada saat menjalankan perintah enkripsi atau dekripsi, *interface* perangkat lunak dapat dilihat seperti pada gambar 4.2 dan 4.3.

Dari gambar 4.2 dapat dilihat informasi dari *file* coba1.txt dengan ukuran (*size*) sebesar 5461 *byte* berada dalam *path* F:\FINALLLL\File Uji\Coba1.txt. Status *un-protected*, berarti bahwa *file* ini bukan *file .gef*. *File* ini melalui proses enkripsi selama 8 detik, menghasilkan *cipher-file* seperti yang ditampilkan pada memo.

Sedangkan pada gambar 4.3 dapat dilihat bahwa *file* coba1.gef dengan status *protected* (merupakan *file .gef*) dengan ukuran (*size*) sebesar 5472 *byte*, berada pada *path* F:\FINALLLL\File Uji\coba1.gef. *File* ini melalui proses dekripsi selama 7 detik menghasilkan *plain-file* seperti yang ditampilkan pada memo.

Gambar 4.2 *Form* Hasil Enkripsi

4.4 Implementasi Uji Coba

Uji coba perangkat lunak enkripsi dan dekripsi metode *GOST* dilakukan pada beberapa tipe *file* yang berbeda ukurannya. Tipe *file* yang diujikan seperti yang telah disebutkan pada subbab 3.4 adalah *file .txt*, *file .jpg*, *file .gif*, *file .wav*, dan *file .mp3*. Ukuran *file* uji dikelompokkan menjadi tiga yaitu ukuran (*size*) *file* kurang dari 10 KB, 10 KB sampai dengan 50 KB, dan ukuran (*size*) *file* lebih besar dari 50 KB. Tabel 4.1 menunjukkan data *file* uji dan ukuran (*size*) masing-masing *file*.

Sebagai *file* uji coba untuk analisis dengan parameter *avalanche effect*, digunakan lima buah *file* teks dengan ukuran panjang bit yang berbeda, yang dapat dilihat pada tabel 4.2.

Gambar 4.3 *Form* Hasil Dekripsi

4.4.1 Hasil Uji

Sebagaimana yang telah dijelaskan sebelumnya pada subbab 3.4, terdapat tiga parameter yang diujikan pada perangkat lunak yang telah dibangun. Tabel hasil uji secara lengkap dapat dilihat pada tabel 4.3 sampai dengan tabel 4.7.

Pada tabel 4.3 dapat diketahui data hasil pengujian dengan parameter ukuran (*size*) *file*. Selisih ukuran (*size*) *file* antara *file* hasil enkripsi dengan *file* awal tidak terlalu signifikan. Selisih terbesar adalah 15 *byte*. Sedangkan untuk ukuran (*size*) *file* hasil dekripsi dengan *file* awal adalah sama. Pada tabel 4.4 data hasil uji coba untuk parameter waktu proses enkripsi dan dekripsi menunjukkan bahwa waktu proses enkripsi dan dekripsi metode *GOST* sangat lambat. Waktu proses enkripsi terbesar adalah 591 detik pada data sebesar 74.212 *byte* sedangkan waktu enkripsi terkecil adalah 4 detik pada data sebesar 4.224 *byte*. Untuk waktu proses dekripsi terbesar adalah 589 detik pada data sebesar 74.224 *byte* dan waktu dekripsi terkecil adalah 5 detik pada data sebesar 4.232 *byte*.

Tabel 4.1 Data *file* uji

Tipe *File*	Nama	Ukuran *File* (byte)
.txt	Coba1	5461
	Coba2	37490
	Coba3	74212
.jpg	Image1	7063
	Image2	30025
	Image3	55144
.gif	Gambar1	9101
	Gambar2	22331
	Gambar3	51236
.wav	Ring1	6400
	Ring2	38930
	Ring3	55776
.mp3	Sing1	4224
	Sing2	28800
	Sing3	60186

Tabel 4.2 Data *file* uji untuk parameter *avalanche effect*

Nama	Ukuran *File* (byte)
Text1.txt	10
Text2.txt	50
Text3.txt	80
Text4.txt	120
Text5.txt	160

Hasil uji coba menggunakan parameter *avalanche effect* untuk masing-masing perubahan dapat dilihat pada tabel 4.5, tabel 4.6 dan tabel 4.7. Perhitungan prosentase *avalanche effect* dapat dicontohkan sebagai berikut :

Contoh :

Data uji pertama (file Text1.txt) sebesar 10 *byte*.

Plaintext : ENKRIPSI !

Analisis :

1. Perubahan pertama (perubahan satu bit *plaintext* terhadap *ciphertext* dengan kunci enkripsi yang sama).
   Hasil enkripsi :
   #Ø +¹F¦ûÎìæs'
   Pengubahan satu bit plaintext : E menjadi e
   Hasil enkripsi :
   Õ□□½¡•WGûÎìæs'
   Bit yang berubah = 8 = #Ø +¹F¦
   Prosentase *avalanche effect* = (8/16) x 100% = 50%
2. Perubahan kedua (perubahan satu bit kunci enkripsi terhadap *ciphertext* dengan *plaintext* yang sama).
   Kata Kunci : tanaya
   Hasil Enkripsi :
   #Ø +¹F¦ûÎìæs'
   Pengubahan satu bit kunci : tanaya menjadi tqnaya
   Hasil enkripsi :
   T™Á4®¡ÄG□jhÓì®Ð□
   Bit yang berubah = 16 = #Ø +¹F¦ûÎìæs'
   Prosentase *avalanche effect* = (16/16) x 100% = 100%
3. Perubahan ketiga (perubahan satu bit *ciphertext* terhadap *plaintext* dengan kunci dekripsi yang sama).

Ciphertext : #Ø +¹F¦ûÎìæs'
Ubah *ciphertext* : #Ø a¹F¦ûÎìæs'
Hasil Dekripsi : ;□□□ƒËeÊ !
Prosentase *avalanche effect* = (8/10) x 100% = 80%

Tabel 4.3 Tabel hasil uji coba untuk parameter ukuran (*size*) *file.*

Tipe File	*Ukuran Awal (byte)*	*Ukuran Enkripsi (byte)*	*Ukuran Dekripsi (byte)*
.txt	5461	5472	5461
.txt	37490	37504	37490
.txt	74212	74224	74212
.jpg	7063	7072	7063
.jpg	30025	30040	30025
.jpg	55144	55152	55144
.gif	9101	9112	9101
.gif	22331	22344	22331
.gif	51236	51248	51236
.wav	6400	6408	6400
.wav	38930	38944	38930
.wav	55776	55784	55776
.mp3	4224	4232	4224
.mp3	28800	28808	28800
.mp3	60186	60200	60186

Tabel 4.5 Tabel hasil uji coba parameter *avalanche effect* perubahan pertama.

Data	*Avalanche Effect (%)*
Text1	50%
Text2	14.29%
Text3	10 %
Text4	6.67%
Text5	5%
Rata-Rata	17.19%

Tabel 4.4 Tabel hasil uji coba untuk parameter waktu proses enkripsi dan dekripsi.

Tipe File	*Ukuran File (byte)*	*Waktu Proses Enkripsi (detik)*	*Waktu Proses Dekripsi (detik)*
.txt	5461	5	6
.txt	37490	168	169
.txt	74212	591	589
.jpg	7063	8	8
.jpg	30025	74	74
.jpg	55144	340	339
.gif	9101	10	11
.gif	22331	44	45
.gif	51236	299	297
.wav	6400	7	7
.wav	38930	182	182
.wav	55776	350	348
.mp3	4224	4	5
.mp3	28800	68	69
.mp3	60186	401	398

Tabel 4.6 Tabel hasil uji coba parameter *avalanche effect* perubahan kedua

Data	*Avalanche Effect (%)*
Text1	100%
Text2	100%
Text3	100%
Text4	100%
Text5	100%
Rata-Rata	100%

Tabel 4.7 Tabel hasil uji coba parameter *avalanche effect* perubahan ketiga

Data	*Avalanche Effect (%)*
Text1	80%
Text2	16%
Text3	28.75%
Text4	45%
Text5	56.25%
Rata-Rata	45.2%

Untuk hasil uji coba dengan parameter *avalanche effect,* dari tabel 4.5, tabel 4.6, dan tabel 4.7 dapat diketahui bahwa setiap pengubahan yang dilakukan pada satu bit data menghasilkan prosentase *avalanche effect* yang berbeda-beda pada ukuran data yang berbeda. Rata-rata yang dihasilkan untuk pengubahan satu bit *plaintext* dengan kunci enkripsi yang sama adalah 17,19% (tabel 4.5). Untuk rata-rata yang dihasilkan pada pengubahan satu bit kunci enkripsi dengan *plaintext* yang sama adalah 100% (tabel 4.6) dan untuk pengubahan satu bit *ciphertext* dengan kunci dekripsi yang sama menghasilkan nilai rata-rata sebesar 45,2% (tabel 4.7). Rata-rata hasil yang didapatkan untuk pengubahan pertama (perubahan satu bit *plaintext* terhadap *ciphertext* dengan kunci enkripsi yang sama) dan ketiga (perubahan satu bit *ciphertext* terhadap *plaintext* dengan kunci dekripsi yang sama) cukup jauh dari hasil optimal yang diinginkan yaitu sebesar 50%. Sedangkan rata-rata hasil untuk pengubahan kedua (perubahan satu bit kunci enkripsi terhadap *ciphertext* dengan *plaintext* yang sama) sangat optimal. Grafik yang menunjukkan perbandingan dari ketiga hasil uji coba menggunakan parameter *avalanche effect* dapat dilihat pada gambar 4.4.

4.4.2 Analisis Hasil

Dari hasil uji coba dengan parameter ukuran (*size*) *file* dapat dibentuk tabel ukuran (*size*) rata-rata *file* enkripsi dan dekripsi yang dapat dilihat pada tabel 4.8. Ukuran rata-rata untuk *file* awal .txt adalah 39054,33 *byte*, ukuran (*size*) enkripsi adalah 39066,67 *byte* dan ukuran (*size*) *file* dekripsi adalah 39054,33 *byte*. Untuk ukuran rata-rata *file* awal .jpg adalah 30744 *byte*, ukuran (*size*) enkripsi

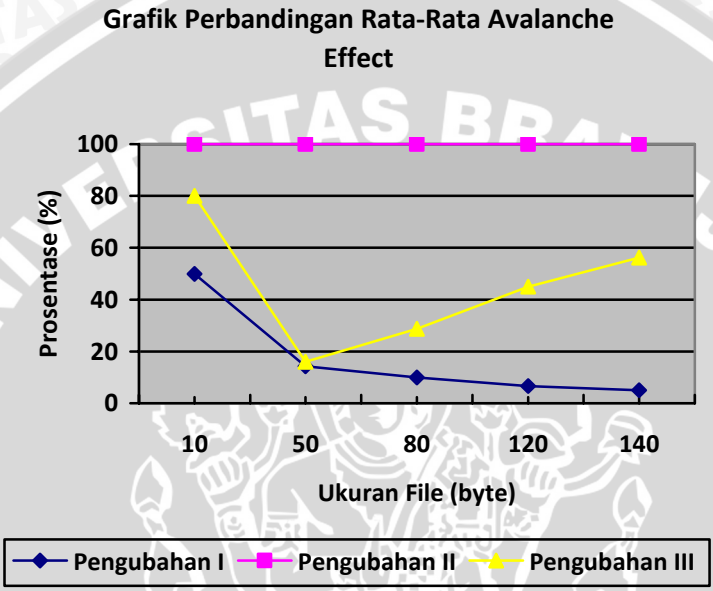
adalah 30754,67 *byte* dan ukuran (*size*) *file* dekripsi adalah 30744 *byte*.

Tabel 4.8 Tabel ukuran (*size*) rata-rata *file* enkripsi dekripsi

Tipe File	*Ukuran Awal (byte)*	*Ukuran Enkripsi (byte)*	*Ukuran Dekripsi (byte)*
.txt	5461	5472	5461
.txt	37490	37504	37490
.txt	74212	74224	74212
Rata-rata	39054.33	39066.67	39054.33
.jpg	7063	7072	7063
.jpg	30025	30040	30025
.jpg	55144	55152	55144
Rata-rata	30744	30754.67	30744
.gif	9101	9112	9101
.gif	22331	22344	22331
.gif	51236	51248	51236
Rata-rata	27556	27568	27556
.wav	6400	6408	6400
.wav	38930	38944	38930
.wav	55776	55784	55776
Rata-rata	33702	33712	33702
.mp3	4224	4232	4224
.mp3	28800	28808	28800
.mp3	60186	60200	60186
Rata-rata	31070	31080	31070

Ukuran rata-rata untuk *file* awal .gif adalah 27556 *byte*, ukuran (*size*) enkripsi adalah 27568 *byte* dan ukuran (*size*) *file* dekripsi adalah 27556 *byte*. Untuk ukuran rata-rata *file* awal .wav adalah 33702 *byte*, ukuran (*size*) enkripsi adalah 33712 *byte* dan ukuran

(*size*) *file* dekripsi adalah 33702 *byte*. Ukuran rata-rata untuk *file* awal .mp3 adalah 31070 *byte*, ukuran (*size*) enkripsi adalah 31080 *byte* dan ukuran (*size*) *file* dekripsi adalah 31070 *byte*. Grafik ukuran rata-rata *file* enkripsi dekripsi dapat dilihat pada gambar 4.5.



Gambar 4.4 Grafik perbandingan rata-rata *avalanche effect*

Ukuran *file .gef* lebih besar bila dibandingkan dengan *file* awal karena adanya proses *padding* dan penambahan delapan bit informasi tambahan pada *file .gef*. Selisihnya sebesar delapan bit ditambah dengan jumlah *padding*. Perbedaan ini tidak dapat ditunjukkan dengan jelas pada grafik, sehingga tampak besarnya bar untuk ukuran (*size*) *file* awal, *file* enkripsi, dan *file* dekripsi relatif sama.

Berdasarkan hasil uji coba mengenai pengaruh ukuran *file* terhadap waktu proses enkripsi dan dekripsi, dapat dibentuk tabel kecepatan rata-rata proses enkripsi dan dekripsi *byte* data per detik (tabel 4.9). Data pada kolom rata-rata enkripsi (*byte*/detik) diperoleh dari pembagian ukuran (*size*) tiap *file* dengan waktu proses enkripsi, sedangkan kolom rata-rata dekripsi (*byte*/detik) diperoleh dari pembagian ukuran (*size*) tiap *file* hasil enkripsi dengan waktu proses dekripsinya.

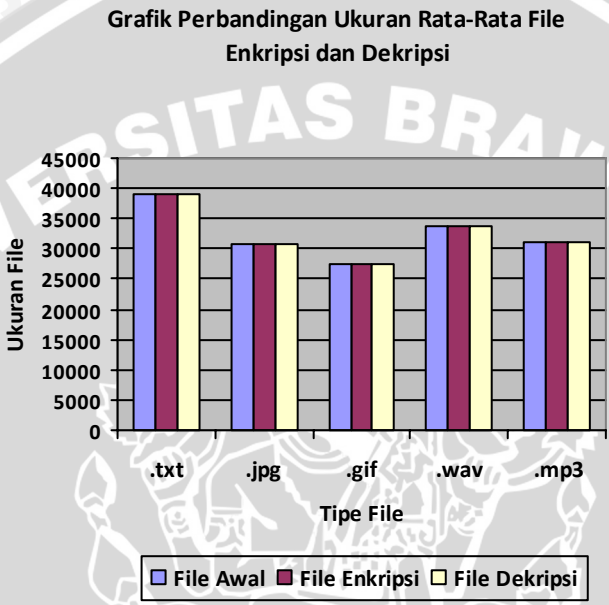
Tabel 4.9 Tabel kecepatan rata-rata proses enkripsi dan dekripsi *byte* data per detik

Tipe File	*Ukuran File (byte)*	*Waktu Proses Enkripsi (detik)*	*Waktu Proses Dekripsi (detik)*	*Rata-Rata Enkripsi (byte/detik)*	*Rata-Rata Dekripsi (byte/detik)*
.txt	5461	5	6	1092.20	912.00
.txt	37490	168	169	223.15	221.92
.txt	74212	591	589	125.57	126.02
.jpg	7063	8	8	882.88	884.00
.jpg	30025	74	74	405.74	405.95
.jpg	55144	340	339	162.19	162.69
.gif	9101	10	11	910.10	828.36
.gif	22331	44	45	507.52	496.53
.gif	51236	299	297	171.36	172.55
.wav	6400	7	7	914.29	915.43
.wav	38930	182	182	213.90	213.98
.wav	55776	350	348	159.36	160.30
.mp3	4224	4	5	1056.00	846.40
.mp3	28800	68	69	423.53	417.51
.mp3	60186	401	398	150.09	151.26

Dari tabel 4.9 dapat dibentuk grafik yang menunjukkan kecepatan rata-rata proses enkripsi dan dekripsi *byte* data per detik yang dapat dilihat pada gambar 4.6 dan 4.7. Dari gambar 4.6 dapat dilihat kecepatan rata-rata proses enkripsi terbesar yaitu 1092,20 *byte* per detik pada data .txt sebesar 5461 *byte*, dan kecepatan rata-rata proses enkripsi terkecil yaitu 125,57 *byte* per detik pada data .txt sebesar 74212 *byte*. Dari gambar 4.7 dapat dilihat kecepatan rata-rata proses dekripsi terbesar yaitu 915,43 *byte* per detik pada data .wav sebesar 6408 *byte*, dan kecepatan rata-rata proses enkripsi terkecil yaitu 126,02 *byte* per detik pada data .txt sebesar 74224 *byte*.

Grafik waktu proses enkripsi dan dekripsi untuk tiap tipe *file* menunjukkan garis yang cukup linear dan menurun. Hal ini berarti bahwa kecepatan waktu proses enkripsi dan dekripsi pada data dengan ukuran (*size*) kecil akan lebih tinggi bila dibandingkan dengan kecepatan waktu proses enkripsi dan dekripsi pada data

dengan ukuran (*size*) yang besar. Dapat dikatakan bahwa besaran yang berpengaruh terhadap waktu proses enkripsi dan dekripsi adalah besarnya ukuran (*size*) *file*.

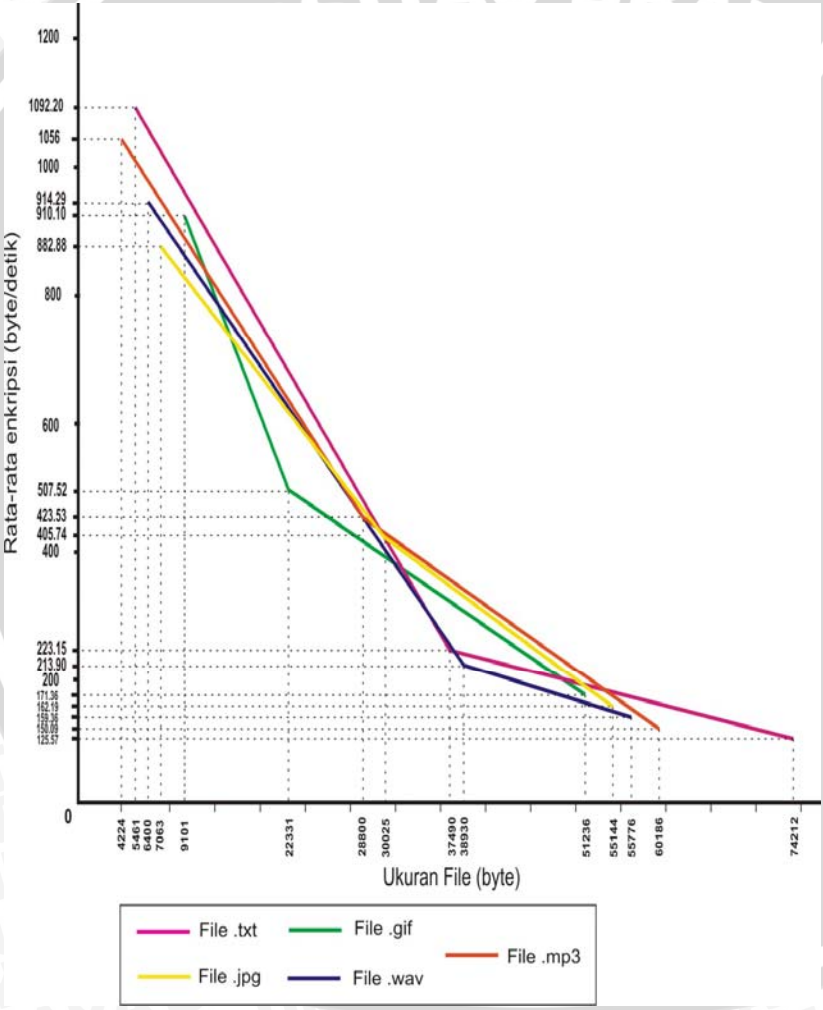


Gambar 4.5 Grafik perbandingan ukuran rata-rata *file* enkripsi dan dekripsi

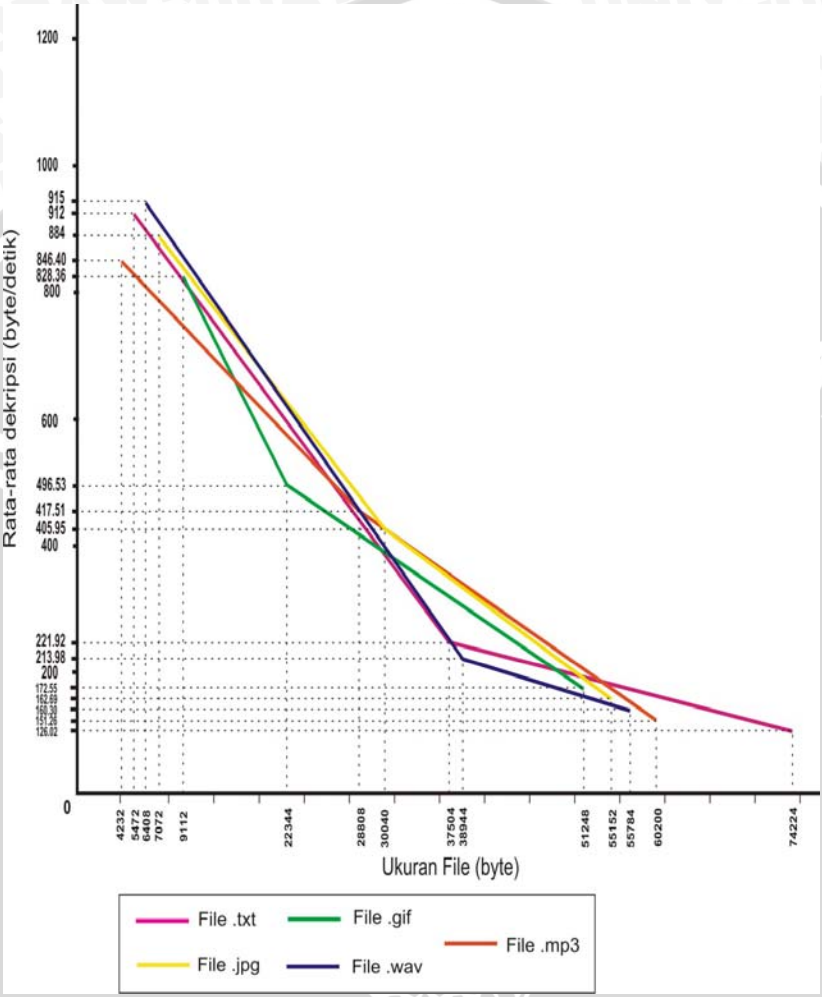
Prosentase *avalanche effect* untuk masing-masing pengubahan yang terdapat pada gambar 4.4 menunjukkan bahwa rata-rata prosentase tertinggi adalah sebesar 100% untuk pengubahan satu bit kunci enkripsi dengan *plaintext* yang sama. Prosentase terbesar kedua adalah untuk pengubahan satu bit *ciphertext* dengan kunci dekripsi yang sama dengan nilai rata-rata sebesar 45,2%. Prosentase terkecil yaitu untuk pengubahan satu bit *plaintext* dengan kunci enkripsi yang sama dengan nilai rata-rata sebesar 17,19%.

Secara umum, urutan grafik prosentase *avalanche effect* dari yang tertinggi adalah rata-rata prosentase pengubahan pertama, rata-rata prosentase pengubahan kedua, dan rata-rata prosentase pengubahan ketiga. Berdasarkan hasil ini dapat dikatakan bahwa ketahanan metode *GOST* sangat baik dalam menangani pengubahan

kunci karena pengubahan satu karakter kunci akan mengubah 100% *ciphertext* yang terbentuk, cukup aman dalam menangani pengubahan *ciphertext* karena pengubahan satu karakter *ciphertext* akan mengubah 45,2% *plaintext* yang dihasilkan, dan kurang aman dalam menangani pengubahan *plaintext* karena pengubahan satu karakter *plaintext* hanya akan mengubah 17,19% *ciphertext* yang terbentuk.



Gambar 4.6 Grafik kecepatan rata-rata proses enkripsi



Gambar 4.7 Grafik kecepatan rata-rata proses dekripsi

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan yang diperoleh dari hasil penulisan laporan ini adalah:

1. Perangkat lunak yang dibangun dapat melakukan enkripsi dan dekripsi untuk semua tipe *file*.
2. Selisih terbesar ukuran *file* awal dan *file* hasil enkripsi adalah 15 *byte*. Selisih didapatkan dari proses *padding* dan penambahan bit informasi.
3. Grafik rata-rata waktu proses enkripsi dan dekripsi berdasarkan ukuran *file* (*ascending*) cukup linear dan menurun. Besaran yang berpengaruh terhadap waktu proses enkripsi dan dekripsi adalah besarnya ukuran (*size*) *file*.
4. Rata-rata prosentase *avalanche effect* untuk perubahan satu bit kunci enkripsi dengan *plaintext* yang sama sebesar 100%, rata-rata prosentase *avalanche effect* untuk perubahan satu bit *ciphertext* dengan kunci dekripsi yang sama adalah 45,2%, dan rata-rata prosentase *avalanche effect* untuk perubahan satu bit *plaintext* dengan kunci enkripsi yang sama adalah 17,19%.

5.2 Saran

Beberapa saran untuk pengembangan lebih lanjut yang dapat diberikan oleh penulis adalah:

1. Data uji untuk analisis proses enkripsi dan dekripsi diperbanyak dengan ukuran (*size*) *file* dan jenis *file* yang lebih bervariasi sehingga didapatkan hasil analisis yang lebih akurat.
2. Melakukan percobaan dengan jumlah *kunci (key)* yang beragam agar dapat terlihat perubahan-perubahan yang terjadi pada data hasil enkripsi.


UNIVERSITAS BRAWIJAYA

DAFTAR PUSTAKA

http://www.gnome.org/projects/gnumeric/doc/file-format-text.shtml. Diakses pada tanggal 29 Mei 2008

http://www.anycount.com/wordcounting_software/word_count/about_txt_file_format.htm. Diakses pada tanggal 29 Mei 2008

http://dhani.singcat.com/IT/hardware.php?page=harddisk. Diakses pada tanggal 29 Mei 2008

Mahyudin, Rezza. 2006. *Algoritma Message Digest 5 (MD5) Dalam Aplikasi Kriptografi*. Program Studi Teknik Informatika, Institut Teknologi Bandung.http://www.informatika.org/~rinaldi/Kriptografi/2006-2007/bahan kuliah2006.htm. Diakses pada tanggal 25 September 2007

Menezes, Alfred J., Paul C van Oorschot, dan Scott A. Vanstone. 1996. *Handbook of Applied Cryptography*. CRC Press. http://www.cacr.math.uwaterloo.ca.hac. Diakses pada tanggal 25 September 2007

Munir, Rinaldi. 2006. *Kriptografi*. Informatika Bandung. Bandung.

Pieprzyk, Josef dan Tombak, Leonid. 1994. *Soviet Encryption Algoritm*. Department of Computer Science University of Wollongong. Wollongong, NSW 2500. http://citeseer.ist.psu.edu/rd/0%2C7705%2C1%2C0.25%2CDownload/http://citeseer.ist.psu.edu/cache/papers/cs/4165/http:zSzzSzwww.ce.chalmers.sezSz~stefanpzSzSecurityzSztr-94-10.pdf/soviet-encryption-algorithm.pdf. Diakses pada tanggal 3 September 2007

Rivest, Ronald L; Robshaw, M.J.B; Sidney, R; and Yin, Y.L. 1998. *The RC6 Block Cipher*. AES Submission

Schneier, Bruce. 1996. *Applied Cryptography: Protocols, Algorithms, and Source Code in C. 2nd ed*. John Wiley and Sons. New Jersey.

Sukmawan, Budi. 1996. *Metoda Enkripsi GOST*. http://www.informatika.org/~rinaldi/Kriptografi/2006-2007/bahan kuliah2006.htm. Diakses pada tanggal 25 September 2007

Usman, Ahmad. 2005. *Pengolahan Citra Digital dan Teknik Pemrogramannya*. Graha Ilmu. Yogyakarta.

Warmada, I Wayan. 2004. *Geokomputasi*. Fakultas Teknik Universitas Gajahmada. Yogyakarta

Wicaksono, Nasrul Kukuh. 2006. *Penerapan Teori Bilangan Bulat dalam Kriptografi dan Aplikasinya dalam Kehidupan Sehari-hari.* Program Studi Teknik Informatika, Institut Teknologi Bandung.http ://www.informatika.org/~rinaldi/Kriptografi/2006-2007/bahankuliah 2006.htm. Diakses pada tanggal 25 September 2007

www.tedi-h.com/related/bOMif2ROzAEKEwiriZmS8KiOAhUGhW UKHS0Wj88QBRgBIAgwtaTSATgN/Cryptography+Tutorial. Diakses pada tanggal 3 September 2007

www.users.zetnet.co.uk/hopwood/crypto/scan/cs.html Diakses pada tanggal 3 September 2007

_. 2005. *Pengantar Kriptografi*. www.indocisc.com Diakses pada tanggal 1 September 2007