Lembar Pengesahan

Pengembangan Algoritma Kriptografi GOST (*Gosudarstvenny Standart*) untuk Peningkatan Keamanan dalam Penyandian Data

GOST (*Gosudarstvenny Standart*) Cryptography Algorithm Development to Increase Security for Data Encryption

Sylviani

113041046

Tugas Akhir ini telah diterima dan disahkan untuk memenuhi sebagian dari syarat
untuk memperoleh gelar Sarjana Teknik Fakultas Informatika
Institut Teknologi Telkom

Bandung, 29 Juli 2011

Menyetujui

Pembimbing I	Pembimbing II
Tjokorda Agung Budi Wirayuda, ST., MT.	Niken Dwi Wahyu Cahyani, ST., M.Kom.
NIP: 06830333-1	NIP: 00750199-1

ABSTRAK

Masalah keamanan dan kerahasiaan data merupakan salah satu aspek penting dalam hal pertukaran informasi. Salah satu solusi untuk menjaga keamanan dan kerahasiaan pada proses pengiriman data adalah dengan teknik enkripsi. Enkripsi adalah proses pengubahan data dari yang bisa dimengerti menjadi sebuah kode yang tidak bisa dimengerti.

GOST merupakan salah satu algoritma enkripsi *block cipher* 32 putaran yang menggunakan struktur *Feistel Network* dan masukan kunci sepanjang 256 bit. Dengan struktur Feistel, algoritma enkripsi memiliki struktur yang sama dengan dekripsi. Perbedaannya hanya terletak pada subkunci yang digunakan. Subkunci dihasilkan dari proses penjadwalan kunci (*key-schedule*). Algoritma *key-schedule* pada GOST sangatlah sederhana, sehingga pada keadaan tertentu hal ini menyebabkan rentan terhadap *related-key attack*.

Salah satu pendekatan untuk mencegah serangan tersebut adalah dengan memaksimalkan *avalanche* pada kunci yaitu dengan cara melewatkan kunci utama (masukan pengguna) ke suatu fungsi hash yang kuat sebelum kunci tersebut diperluas dengan algoritma *key-schedule*.

Tugas akhir ini membahas tentang modifikasi algoritma GOST untuk meningkatkan keamanan terhadap *related-key attack* dengan menambahkan fungsi hash yaitu HAVAL dan SHA-256 sebelum proses penjadwalan kuncinya. Dari pengujian yang telah dilakukan, disimpulkan bahwa *related key attack* tidak bisa diterapkan lagi dan diperoleh peningkatan nilai *avalanche effect* pada algoritma modifikasi GOST.

Kata kunci: *block cipher*, GOST, *key-schedule, related-key attack, Feistel Network, sub-key, avalanche effect.*

ABSTRACT

Issues of security and confidentiality of data is one of important aspects in terms of information exchange. One of the solutions to maintain the security and confidentiality of the data transmission process is the encryption technique. Encryption is the process of transforming data from which can be understood to be unreadable.

GOST is one block cipher encryption algorithm that uses a 32 round Feistel network structure and 256-bit key. With a Feistel structure, the encryption algorithm has the same structure as the decryption. The difference lies only in the sub-key is used. Sub-key generated from the key scheduling process (key-schedule). Key scheduling algorithm in the GOST is very simple, so that in certain circumstances this causes vulnerable to related-key attack.

One approach to preventing that attacks is to maximize the avalanche on the key that is by passing through a strong hash function before the key is expanded to the key scheduling algorithm.

In this final project, the key scheduling was design by adding one-way hash function, HAVAL and SHA-256, to improve security againts related key attack. From the tests performed that has been done, it was concluded that the related key attack is no longer applicable and obtained an increase in the value of the avalanche effect from the modification of algorithm GOST.

***Keywords**: GOST, block cipher, Feistel Network, sub-key, key-schedule, related-key attack.*

Lembar Persembahan

Alhamdulillahirobbil'alamin, puji syukur atas segala anugrah, cinta, kasih sayang, dan ilmu yang selalu dilimpahkan oleh Allah SWT dan Sholawat serta salam teruntuk Rasullullah Muhammad SAW, tabi'-tabi'in, dan shohabat-shohabiah. Dengan izin-Nyalah penulis dapat menyelesaikan tugas akhir yang berjudul **"Pengembangan Algoritma Kriptografi GOST (*Gosudarstvenny Standart*) untuk Peningkatan Keamanan dalam Penyandian Data"**. Kelancaran dalam mempersiapkan dan menyelesaikan Tugas Akhir ini tidak terlepas dari bantuan berbagai pihak. Oleh karena itu, penulis ingin mengucapkan terima kasih yang sangat mendalam kepada:

1. Allah SWT, Subhanallah tak henti-hentinya memberikan anugrah yang seringkali tak disadari oleh penulis, segala pujian dan senandung keagungan takkan pernah berhenti dilantunkan untuk-Mu.
2. Kedua orangtua penulis, Ibunda Rosilawati Yunus dan Ayahanda Gunawan Said atas semua do'a, pengorbanan dan kasih sayang tulus yang tak pernah berhenti mengalir untuk penulis. Segenap do'a akan selalu dipanjatkan untuk mereka "*ALLAHUMMAGHFIRLII WALIWAA LIDAYYA WARHAMHUMAA KAMAA ROBBAYAANII SHOGHIIROO*"
3. *My lovely brother*, Haryono (alm.) "*Rest in peace there, always watching us from up above, we always love you"*, Haryanto dan Rowan Al-Warits, semoga kita bisa selalu berkumpul, bercanda, dan tertawa bersama karena *support* kalian sangat dibutuhkan sampai puzzle kehidupan penulis tersusun dengan sempurna, amin. *My lovely sister* Ai Sriwahyuni, yang telah menghadirkan keponakan yang lucu-lucu di rumah. *My big family* yang ada di Belitung, Jakarta, Bogor, Lampung, dan Surabaya atas segala doa, *support* serta nasehat-nasehat "bawel"-nya kepada penulis agar segera menyelesaikan kuliah ini. *My besties* Dini Fronitasari *"thanks for everything, Life is just an adventure and I'm ready right now".*
4. Bapak Tjokorda Agung Budi Wirayuda, ST., MT., dan Ibu Niken Dwi Wahyu Cahyani, ST., M.Kom., selaku pembimbing yang telah memberikan kesempatan, petunjuk, pengarahan, pencerahan, waktu, koreksi, bimbingan untuk membimbing penulis dalam menyelesaikan tugas akhir ini.
5. *My Special Edition* Ipda Pol. Anton JX, atas segala perhatian, ceramah, dan motivasi yang selalu diberikan kepada penulis selama 3 tahun terakhir. Pesan yang selalu diingat oleh penulis "tak kan lari gunung dikejar". "*Let you be the last for become the first in my life".*
6. *My 2nd Brain* Bapak Joni Gozali, yang telah memberikan kontribusi yang besar, pemahaman materi, penjelasan detail serta telah meluangkan waktunya untuk membantu menyelesaikan tugas akhir ini.

7. Seluruh Dosen dan Staf Fakultas Informatika IT Telkom, *Spesial thanks to "The Problem Solver"* Bapak Bayu Erfianto *feat* Ibu Vera Suryani, Dosen Wali terbaik Bapak Fazmah Arif Yulianto, serta Admin Fakultas Informatika Bapak Iwan Kurniawan dan Bapak Zakaria atas segala ilmu dan pengalaman yang diberikan serta kemudahan-kemudahan dan keramahannya terhadap mahasiswa.
8. Seluruh penghuni PGA yang selalu setia menjadi teman, sahabat, adik, dan kakak selama penulis berada di Bandung atas kebersamaan, suka duka, tangis dan tawa selama 7 tahun ini. Tak ada yang sia-sia dari semua yang pernah kita lewati bersama-sama sekalipun telah menguras keringat dan air mata.
9. Teman-teman seperjuangan, senasib dan sepenanggungan angkatan 2004 yang masih tersisa di akhir-akhir masa studi ini, Alinda Prima Damayanti, Panji Copri Sasongko, Tunggul Raedi, Shoffan Azizurrahman, Tomy Angga Putranto, Sabdo Hastopo, Dimas Rizky Fajar, Kurniawan Zakaria, Lina Yurnalita, Ronald Hutabarat, Lau Rencius, Roy M. Silaban dan kawan-kawan, atas perjuangan panjang yang telah kita lalui bersama-sama, semoga ini menjadi awal yang baik untuk menapaki perjalanan selanjutnya. Amin.
10. *My Enemies* Dek Iky dan Oja atas canda tawa yang selalu hadir saat kita bersama, tiada hari tanpa celaan yang tiada tara karena tiada mencela dan dicela sama dengan HAMPA serta *Anjang's Family* yang telah menghantarkan penulis ke tempat-tempat terindah di Indonesia ini, perjalanan panjang selanjutnya akan selalu ditunggu.
11. *My 2nd Family*, Papa, Mama, Ipda pol. Andi Muhammad Akbar Mekuo, Niul, Dek Dhika, dan Juanita Nur Aulia atas segala motivasi dan semangat yang luar biasa yang telah diberikan kepada penulis.
12. Rika Ayu Utami, Reny Renaningsih dan *my bro* Andri Maulana atas segala bantuan kepada penulis baik do'a, waktu maupun tenaga dalam mendukung penyelesaian Tugas Akhir ini selama 3 bulan terakhir.
13. Dan semua pihak yang telah ikut membantu dan memberikan motivasi, dorongan serta semangat kepada penulis dalam menyelesaikan tugas akhir ini yang tidak bisa disebutkan satu persatu.

Kata Pengantar

Puji syukur penulis ucapkan atas kebesaran, nikmat dan karunia Allah SWT yang telah melimpahkan segala nikmat dan berkah-Nya sehingga Tugas Akhir yang berjudul "**Pengembangan Algoritma Kriptografi GOST (*Gosudarstvenny Standart*) untuk Peningkatan Keamanan dalam Penyandian Data**" ini dapat terselesaikan dengan baik.

Tugas Akhir ini disusun sebagai salah satu persyaratan untuk menyelesaikan program pendidikan Sarjana Teknik Fakultas Informatika di Institut Teknologi Telkom. Penulis sangat menyadari bahwa dalam penulisan tugas akhir ini masih terdapat banyak kekurangan. Oleh karena itu, penulis sangat mengharapkann kritik dan saran yang positif untuk perbaikan terhadap tugas akhir ini sebagai ilmu di masa yang akan datang.

Bandung, 29 Juli 2011

Sylviani

DAFTAR ISI

Daftar Gambar

Daftar Tabel

Daftar Istilah

Attacker	Penyerang
Ciphertext	pesan yang terenkripsi
File input	file yang akan dienkripsi
File output	file yang terenkripsi
Key schedule	proses penjadwalan kunci untuk digunakan dalam setiap putaran algoritma GOST
Padding	proses penambahan bit-bit *dummy*
Passphrase	masukan dari pengguna berupa kunci yang akan digunakan untuk melakukan enkripsi maupun dekripsi
Password	Kunci
Plaintext	pesan yang akan dienkripsi
Secure	Aman
User	pengguna

1. PENDAHULUAN

1.1. Latar Belakang

Keamanan data merupakan hal yang sangat penting dalam menjaga kerahasiaan informasi, terutama yang berisi informasi sensitif dan rahasia. Pertukaran informasi melalui jaringan publik baik yang bersifat tertutup (*intranet*) maupun yang bersifat terbuka (*internet*) memungkinkan mendapat ancaman dari pihak-pihak yang tidak bertanggungjawab, dimana informasi tersebut akan sangat mudah disadap dan diketahui isinya. Untuk mencegah hal tersebut, maka kita membutuhkan pengamanan pada sistem informasi yang kita gunakan.

Metode kriptografi dapat digunakan untuk mengamankan data yang bersifat rahasia agar data tersebut tidak diketahui oleh pihak lain yang tidak memiliki hak akses. Prinsip kriptografi adalah sebelum data asli (*plaintext*) dikirim atau ditransmisikan, di tempat pengirim data terlebih dahulu disandikan (proses enkripsi). Jadi, data yang dikirim adalah data berupa teks tersandi (*chipertext*). Walaupun *hacker* dapat menyadap komunikasi, *hacker* tidak dapat membaca dan mengerti makna dari komunikasi yang terjadi karena data yang berhasil disadap berupa teks tersandi. Di tempat penerima, teks tersandi dikembalikan lagi menjadi teks semula atau teks asli (proses dekripsi).

Salah satu contoh metode kriptografi adalah algoritma GOST (*Gosudarstvenny Standart*). Algoritma GOST merupakan suatu algoritma *block cipher* yang dikembangkan oleh pemerintah Uni Soviet pada masa perang dingin untuk menyembunyikan data atau informasi yang bersifat rahasia. Algoritma ini merupakan algoritma enkripsi sederhana yang menggunakan 64-bit *block cipher* dengan panjang kunci 256 bit. Algoritma GOST juga menggunakan 8 buah *S-Box* 4 bit yang memiliki nilai yang berbeda-beda. Karena terdiri dari 32 putaran dalam prosesnya, maka algoritma GOST memiliki penjadwalan kunci (*key-schedule)*. Penjadwalan kunci pada algoritma GOST sangatlah sederhana, dimana subkunci dihasilkan dengan membagi 256 bit kunci menjadi delapan 32 bit subkunci: k_1, k_2, …, k_8. Hal inilah yang menyebabkan pada keadaan tertentu algoritma ini menjadi lemah sehingga rentan terhadap metoda kriptanalisis seperti *Related-key Attack.* Metoda kriptanalisis ini tentunya berpengaruh terhadap keamanan dalam penyandian data karena selalu berusaha mencari kelemahan dari sistem pengamanan data tersebut untuk bisa mendapatkan informasi rahasia yang ada di dalamnya secara ilegal.

1.2. Perumusan Masalah

Berdasarkan latar belakang pemilihan judul, maka yang menjadi permasalahan dalam tugas akhir ini adalah bagaimana membuat pengembangan terhadap algoritma GOST agar dapat meningkatkan keamanan dalam penyandian data sehingga tahan terhadap metode *Related-key Attack.*

1.3. Batasan Masalah

Adapun ruang lingkup yang menjadi batasan masalah dalam tugas akhir ini antara lain:

1. Aplikasi hanya bisa memproses pesan berupa file teks saja (.txt).
2. Mode operasi yang digunakan adalah *Electronic Code Book* (ECB)
3. Analisis performansi menggunakan parameter waktu proses yang dibutuhkan untuk melakukan proses enkripsi maupun dekripsi dan parameter tingkat keamanan dengan pengujian nilai *avalanche effect*.

1.4. Tujuan

Penyusunan tugas akhir ini bertujuan untuk menganalisis performansi dari algoritma pengembangan GOST dan meningkatkan keamanan dalam enkripsi dan dekripsi data, salah satunya tahan terhadap metode *Related-key Attack*.

1.5. Hipotesa Awal

Salah satu pendekatan untuk mencegah *Related-key Attack* adalah memaksimalkan *avalanche* pada kunci yaitu dengan cara melewatkan kunci ke sebuah fungsi hash sebelum kunci dimasukkan ke proses penjadwalan kunci.

1.6. Metodologi Penyelesaian Masalah

Metodologi yang digunakan dalam menyelesaikan Tugas Akhir ini adalah sebagai berikut:

a. Studi Literatur, membaca dan mencari sumber-sumber informasi dari buku-buku, jurnal dan *internet* yang bertujuan untuk mempelajari dan memahami lebih mendalam mengenai:
    1. Sejarah dan konsep-konsep dasar kriptografi [2][8][12][13].
    2. Teknik-teknik dasar pemrograman dengan menggunakan bahasa pemograman Microsoft Visual Basic 6.0 [15].
    3. Metode kriptografi GOST [2][10][12][13][16][17].
    4. Metode *Related-key Attack* [3][4][5][6][7][11].
b. Melakukan analisis spesifikasi kebutuhan sistem yang akan dibangun.
c. Melakukan perancangan sistem dari hasil analisis spesifikasi yang sudah didapat dengan membuat Diagram Aliran Data, spesifikasi proses dan kamus data untuk sistem yang akan dibangun.
d. Melakukan implementasi dengan menerapkan algoritma kriptografi untuk proses enkripsi dan dekripsi ke sistem yaitu algoritma GOST yang sudah dimodifikasi pada penjadwalan kuncinya.
e. Tahap analisis hasil pengujian sistem yaitu dengan melakukan analisis perbandingan performansi antara algoritma GOST standart dengan algoritma GOST yang sudah dikembangkan menggunakan parameter tingkat keamanan, *avalanche effect* dan waktu proses yang dibutuhkan untuk melakukan proses enkripsi dan dekripsi.

f. Membuat laporan tugas akhir berupa dokumentasi dari semua tahapan proses yang sudah dilakukan selama pengerjaan tugas akhir yang berisi tentang dasar teori dan hasil akhir dari Tugas Akhir ini.

1.7. Sistematika Penulisan

Sistematika penulisan Tugas Akhir ini disusun sebagai berikut:

BAB I PENDAHULUAN

Bab ini berisi latar belakang masalah, perumusan masalah, batasan masalah, tujuan, metodologi penyelesaian masalah, dan sistematika penulisan.

BAB II LANDASAN TEORI

Bab ini membahas tentang teori-teori yang menjadi landasan konseptual dalam penyelesaian Tugas Akhir ini, meliputi pengertian, sejarah, konsep-konsep, algoritma dan penerapan kriptografi.

BAB III ANALISIS DAN PERANCANGAN SISTEM

Bab ini membahas tentang pengumpulan data analisis dan perancangan aplikasi Enkripsi dan Dekripsi dengan menggunakan metode GOST yang sudah dimodifikasi.

BAB IV ANALISIS PENGUJIAN SISTEM

Bab ini membahas tentang analisis hasil dari pengujian sistem.

BAB V PENUTUP

Bab ini berisi tentang kesimpulan dan saran-saran secara keseluruhan dari hasil analisis dan implementasi Tugas Akhir ini.

2. LANDASAN TEORI

2.1. Kriptografi

2.1.1. Pengertian Kriptografi

Kata kriptografi (*cryptography*) berasal dari bahasa Yunani, yaitu *kriptos* yang artinya *secret* (rahasia) dan *graphein* yang artinya *writing* (tulisan). Jadi kriptografi berarti *secret writing* (tulisan rahasia). Ada beberapa definisi kriptografi yang telah dikemukakan di dalam berbagai literatur. Definisi yang dipakai di dalam buku-buku yang lama (sebelum tahun 1980-an) menyatakan bahwa kriptografi adalah ilmu dan seni untuk menjaga kerahasiaan pesan dengan cara menyandikannya ke dalam bentuk yang tidak dapat dimengerti maknanya. Definisi ini mungkin cocok pada masa lalu dimana kriptografi digunakan untuk keamanan komunikasi penting seperti komunikasi di kalangan militer, diplomat dan mata-mata. Namun saat ini kriptografi lebih dari sekadar *privacy*, tetapi juga untuk tujuan *data integrity*, *authentication*, dan *non-repudiation*.

Definisi kriptografi yang dikemukakan di dalam buku '*Applied Cryptography, 2nd edition*' oleh Bruce Schneir yaitu "Kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan (*cryptography is the art and science of keeping messages secure*)". Kata "seni" di dalam definisi di atas berasal dari fakta sejarah bahwa pada masa-masa awal sejarah kriptografi, setiap orang mungkin mempunyai cara yang unik untuk merahasiakan pesan. Cara-cara unik tersebut mungkin berbeda-beda pada setiap pelaku kriptografi dimana setiap cara menulis pesan rahasia mempunyai nilai estetika tersendiri sehingga kriptografi berkembang menjadi sebuah seni dalam merahasiakan pesan.

2.1.2. Sejarah

Menurut sejarahnya, kriptografi sudah lama digunakan oleh tentara Sparta di Yunani pada permulaan tahun 400 SM. Mereka menggunakan alat yang disebut *scytale*. Alat ini terdiri dari sebuah pita panjang dari daun papyrus yang dililitkan pada sebatang selinder. Pesan yang akan dikirim ditulis secara horizontal (baris per baris). Bila pita dilepaskan, maka huruf-huruf di dalamnya telah tersusun membentuk sebuah pesan rahasia. Untuk membaca pesan, penerima melilitkan kembali silinder yang diameternya sama dengan diameter silinder pengirim. Teknik kriptografi ini dikenal dengan nama transposisi *cipher*, ini merupakan teknik kriptografi yang pertama kali digunakan sekaligus sebagai metode enkripsi tertua. Gambar *scytale* dapat dilihat pada gambar berikut ini [13]:

Gambar 2.1 Bentuk *Scytale*

Bidang ilmu ini terus berkembang seiring dengan kemajuan peradaban manusia. Selain itu, ilmu kriptografi juga memegang peranan penting dalam strategi peperangan yang terjadi dalam sejarah manusia, dimulai dari "Caesar Chiper" pada zaman Romawi Kuno, "Playfair Chiper" yang digunakan Inggris, "ADFVGX Chiper" yang digunakan pada Perang Dunia I, sampai algoritma-algoritma kriptografi Rotor yang mulai populer pada Perang Dunia II seperti Typex, Purple, dan mesin kriptografi legendaris Enigma di Jerman. Induk dari keilmuan dari kriptografi sebenarnya adalah matematika, khususnya teori aljabar yang mendasar yaitu ilmu bilangan. Oleh karena itu kriptografi semakin berkembang ketika komputer ditemukan. Sebab dengan penemuan komputer memungkinkan dilakukanya perhitungan yang rumit dan komplek dalam waktu yang relatif singkat, suatu hal yang sebelumnya tidak dapat dilakukan. Dari hal tersebut lahirlah banyak teori dan algotitma penyandian data yang semakin kompleks dan sulit dipecahkan.

2.1.3. Tujuan dan Manfaat

Kriptografi bertujuan untuk memberi layanan keamanan yang juga dinamakan sebagai aspek-aspek keamanan sebagai berikut [17]:

1. **Kerahasiaan (*confidentiality*)**, yaitu layanan yang ditujukan untuk menjaga agar pesan tidak dapat dibaca oleh pihak-pihak yang tidak berhak. Di dalam kriptografi, layanan ini direalisasikan dengan menyandikan pesan menjadi cipherteks.
2. **Integritas data (*data integrity*)**, yaitu layanan yang menjamin bahwa pesan masih asli/ utuh atau belum pernah dimanipulasi selama proses pengiriman. Untuk menjaga integritas data, sistem harus memiliki kemampuan untuk mendeteksi manipulasi pesan oleh pihak-pihak yang tidak berhak. Layanan ini direalisasikan dengan menggunakan tanda tangan digital (*digital signature*).
3. **Otentikasi (*authentication*)**, yaitu layanan yang berhubungan dengan identifikasi, termasuk mengidentifikasi kebenaran pihak-pihak yang berkomunikasi (*user authentication*). Dua pihak yang saling berkomunikasi harus dapat mengotentikasi satu sama lain sehingga ia dapat memastikan sumber pesan. Otentikasi sumber pesan secara implisit juga memberikan kepastian integritas data, sebab jika pesan telah dimodifikasi berarti sumber pesan sudah tidak benar.
4. **Nirpenyangkalan (*non-repudiation*)**, yaitu layanan untuk mencegah entitas yang berkomunikasi melakukan penyangkalan, yaitu pengirim pesan menyangkal melakukan pengiriman atau penerima pesan menyangkal telah menerima pesan.

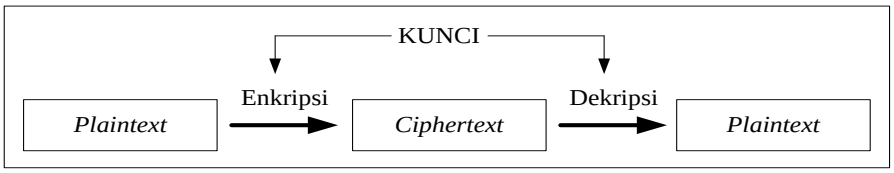
Beberapa manfaat dari kriptografi adalah pengamanan untuk transaksi di mesin ATM, E-Commerce, transaksi dengan kartu kredit, secure e-mail, percakapan telepon atau pengaktifan peluru kendali. Oleh karena itu kriptografi banyak digunakan di bidang diplomat, militer, bisnis, telekomunikasi, jaringan komputer, keuangan, perbankan, dan pendidikan. Singkatnya, dimana suatu kerahasiaan data amat diperlukan maka disitulah kriptografi memegang peranan penting.

2.2. Jenis Algoritma Kriptografi

Terdapat dua jenis algoritma kriptografi berdasarkan jenis kuncinya, yaitu algoritma simetri dan algoritma asimetri [12].

2.2.1. Algoritma Simetri

Algoritma Simetri atau disebut juga dengan algoritma konvensional adalah algoritma yang menggunakan kunci enkripsi yang sama dengan kunci dekripsinya. Algoritma Simetri sering juga disebut dengan algoritma kunci rahasia atau algoritma satu kunci yang mengharuskan pengirim dan penerima menyetujui suatu kunci tertentu sebelum mereka dapat berkomunikasi dengan aman. Membocorkan kunci berarti bahwa pihak lain dapat melakukan proses enkripsi dan dekripsi terhadap pesan yang sifatnya rahasia. Beberapa algoritma yang termasuk algoritma kunci simetri adalah One-Time-Pad (OTP), DES, RC2, RC4, RC5, RC6, WAKE, IDEA, TwoFish, FEAL, SAFER, CAST, AES, Blowfish, GOST, A5 dan lain-lain [12].



Gambar 2.2 Proses Enkripsi dan Dekripsi

Dalam dunia kriptografi, *password* sering disebut sebagai kunci. Pesan asli yang belum dikodekan disebut plainteks. Plainteks tidak harus berupa teks, namun dapat berupa file gambar (.gif, .jpg), file biner (.exe, .com, .ocx), file suara (.wav, .mp3), dan lain-lain. File yang telah disandikan disebut cipherteks. Enkripsi adalah proses pengubahan pesan asli menjadi karakter yang tidak dapat dibaca. Sedangkan dekripsi adalah proses pengubahan karakter yang tidak dapat dibaca menjadi pesan asli. Kita sering menggunakan notasi matematika untuk mempermudah penulisan dan analisis, sehingga kriptografi modern selalu berhubungan dengan matematika. Dengan pesan asal P dan kode rahasia C yang diperoleh dari enkripsi dengan kunci K, kita dapat menuliskan:

$$\mathbf{C = E_k\,(P)}$$

Notasi ini menyatakan bahwa C dihasilkan oleh fungsi Enkripsi E yang dioperasikan terhadap masukan P dengan kunci K. Operasi ini dilakukan oleh pengirim pesan. Penerima pesan melakukan operasi sebaliknya:

$$\mathbf{P = E_k\,(C)}$$

Operasi ini adalah proses dekripsi untuk kembali mendapatkan pesan asal P. Pemecah kode (*cryptanalyst*) sering kali hanya memiliki C dan harus menemukan nilai P.

Algoritma simetri dapat dibagi dalam dua kategori. Jenis pertama beroperasi pada plainteks yang berupa satu bit tunggal pada satu waktu, yang disebut *stream ciphers*. Jenis kedua beroperasi pada plainteks dalam grup bit-bit yang disebut blok (*block ciphers*). Untuk algoritma komputer modern, ukuran blok dasarnya adalah 64 bit atau 128 bit, cukup besar untuk menghindari analisis pemecahan kode dan cukup kecil agar dapat bekerja dengan cepat [12].

2.2.1.1. *Stream Cipher*

Algoritma *stream cipher* bekerja dengan cara mengenkrip karakter demi karakter (biasanya digit *binary*) dari plainteks. *Stream cipher* jauh lebih cepat dibandingkan dengan algoritma *block cipher*. Algoritma ini secara umum digunakan untuk mengenkrip plainteks yang kecil biasanya dalam ukuran bit. Salah satu contoh algoritma *stream cipher* adalah algoritma RC4 yang dikembangkan pada tahun 1987 oleh Ronald Rivest [12].

2.2.1.2. *Block Cipher*

Block cipher adalah fungsi yang memetakan n-bit blok plainteks menjadi n-bit cipherteks. Hasil pemetaan dari plainteks ke cipherteks akan berbeda tergantung pada kunci yang digunakan. Dengan demikian, algoritma *block cipher* akan melakukan proses enkripsi dan dekripsi untuk setiap blok bit. Bila blok terakhir dari plainteks tidak mencukupi ukuran blok bit yang ditentukan, maka pada blok terakhir tersebut harus dilakukan *padding*. Proses *padding* adalah proses penambahan *byte-byte dummy* berupa karakter *null* pada *byte-byte* sisa yang masih kosong pada blok terakhir plainteks, sehingga ukurannya menjadi sama dengan ukuran blok penyandian. Beberapa contoh algoritma *block cipher* adalah DES, Blowfish, RC2, RC5, IDEA dan GOST [12].

2.2.2. Algoritma Asimetri

Algoritma Asimetri atau disebut juga dengan algoritma kunci publik, didesain sedemikian rupa sehingga kunci yang digunakan untuk enkripsi berbeda dengan kunci yang digunakan untuk dekripsi. Kunci dekripsi tidak dapat dihitung dari kunci enkripsi. Algoritma ini disebut kunci publik karena kunci enkripsi dapat dibuat publik yang berarti semua pihak boleh mengetahuinya. Pihak manapun dapat menggunakan kunci enkripsi untuk melakukan enkripsi terhadap pesan, namun hanya orang tertentu (calon penerima pesan dan sekaligus pemilik kunci dekripsi yang merupakan pasangan kunci publik), yang dapat melakukan dekripsi terhadap pesan tersebut. Dalam sistem ini, kunci enkripsi sering disebut kunci publik, sementara kunci dekripsi sering disebut kunci privat. Yang termasuk algoritma asimetri adalah ECC, LUC, RSA, El Gamal dan DH. Enkripsi dengan kunci publik K_e dinyatakan sebagai:

$$\mathbf{E_{Ke}\,(P) = C}$$

Dengan kunci privat (K_d) sebagai pasangan kunci publik (K_e), dekripsi dengan kunci privat yang bersesuaian dapat dinyatakan dengan:

$$\mathbf{D_{Kd}\,(C) = P}$$

Di sini, K_e merupakan pasangan K_d. Artinya tidak ada K_d lain yang dapat digunakan untuk melakukan dekripsi kode C yang merupakan hasil enkripsi dengan kunci K_e. Sebaliknya, pesan dapat dienkripsi dengan kunci privat dan didekripsi dengan kunci publik. Artinya kunci privat dan kunci publik digunakan secara berlawanan dengan tujuan yang berbeda. Sifat ini berlaku untuk algoritma kunci publik tertentu semacam RSA [12].

2.3. Algoritma GOST

GOST merupakan singkatan dari "*Gosudarstvennyi Standard*" atau "*Government Standard*". Metode GOST merupakan algoritma *block cipher* yang dikembangkan oleh seseorang yang berkebangsaan Uni Soviet. Metode ini kemudian dikembangkan oleh pemerintah Uni Soviet pada masa perang dingin untuk menyembunyikan data atau informasi yang bersifat rahasia pada saat komunikasi. Algoritma ini merupakan suatu algoritma enkripsi sederhana yang memiliki jumlah proses sebanyak 32 *round* (putaran) dan menggunakan 64 bit *block cipher* dengan 256 bit *key*. Metode GOST juga menggunakan 8 buah *S-Box* yang berbeda-beda dan operasi *XOR* serta *Left Circular Shift*.

Kelemahan GOST yang diketahui sampai saat ini adalah karena *key schedule*-nya yang sederhana sehingga pada suatu keadaan tertentu menjadi titik lemahnya terhadap metode kriptanalisis seperti *Related-key Cryptanalysis*. Tetapi hal ini dapat diatasi dengan melewatkan kunci kepada fungsi *hash* yang kuat secara kriptografi seperti fungsi *hash* SHA-1, kemudian menggunakan hasil *hash* untuk *input* inisialisasi kunci. Kelebihan dari metode GOST ini adalah kecepatannya yang cukup baik, walaupun tidak secepat *Blowfish* tetapi GOST lebih cepat dari IDEA [2]. Komponen dari metode GOST adalah sebagai berikut [14]:

1. *Key Store Unit* (KSU) menyimpan 256 *bit string* dengan menggunakan 32 *bit register* (K0, K1, …, K7).
2. Dua buah 32 *bit register* (L_i, R_i).
3. 32 *bit adder modulo* 2^{32} (CM1).
4. *Bitwise Adder XOR* (CM2).
5. *Substitusion block* (S) yaitu berupa 8 buah 64 *bit S-Box*.
6. *Rotasi Left* (R) sebanyak 11 *bit*.

2.3.1. Proses Pembentukan Kunci

Proses pembentukan kunci dapat dilihat pada penjabaran berikut ini:

1. *Input key* berupa 256 *bit key* dengan perincian k_1, k_2, k_3, k_4, …, k_{256}.
2. *Input key* ini akan dikelompokkan dan dimasukkan ke dalam 8 buah KSU dengan aturan seperti berikut [14]:

$K_0 = (k_{32}, \ldots, k_1)$ $\quad$ $K_4 = (k_{160}, \ldots, k_{129})$
$K_1 = (k_{64}, \ldots, k_{33})$ $\quad$ $K_5 = (k_{192}, \ldots, k_{161})$
$K_2 = (k_{96}, \ldots, k_{65})$ $\quad$ $K_6 = (k_{224}, \ldots, k_{193})$
$K_3 = (k_{128}, \ldots, k_{97})$ $\quad$ $K_7 = (k_{256}, \ldots, k_{225})$

2.3.2. Proses Enkripsi

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

1. 64 *bit* plainteks dibagi menjadi 2 buah bagian 32 *bit*, yaitu L_i dan R_i.
   Contoh:
   Input a(1), a(2), …, a(32), b(1), b(2), …, b(32)
   L_0 = a(32), a(31), …, a(1)
   R_0 = b(32), b(31), …, b(1)
2. $(R_i + K_i) \bmod 2^{32}$. Hasil dari penjumlahan modulo 2^{32} ini adalah 32 bit.

3. Hasil dari penjumlahan modulo 2^{32} dibagi menjadi 8 bagian, dimana setiap bagian 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. *S-Box* yang digunakan pada metode GOST dapat dilihat pada tabel berikut [17]:

Tabel 2.1 *S-Box* dari Metode GOST

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

Cara melihat *S-Box* yaitu *input* biner diubah menjadi bilangan desimal dan hasilnya menjadi urutan bilangan dalam *S-Box*.

4. Hasil yang didapat dari substitusi ke *S-Box* digabungkan kembali menjadi 32 *bit* dan kemudian dilakukan rotasi kiri sebanyak 11 bit.
5. $R_{i+1} = R_i$ (hasil dari *rotate left*) XOR L_i.
6. $L_{i+1} = R_i$ sebelum dilakukan proses.

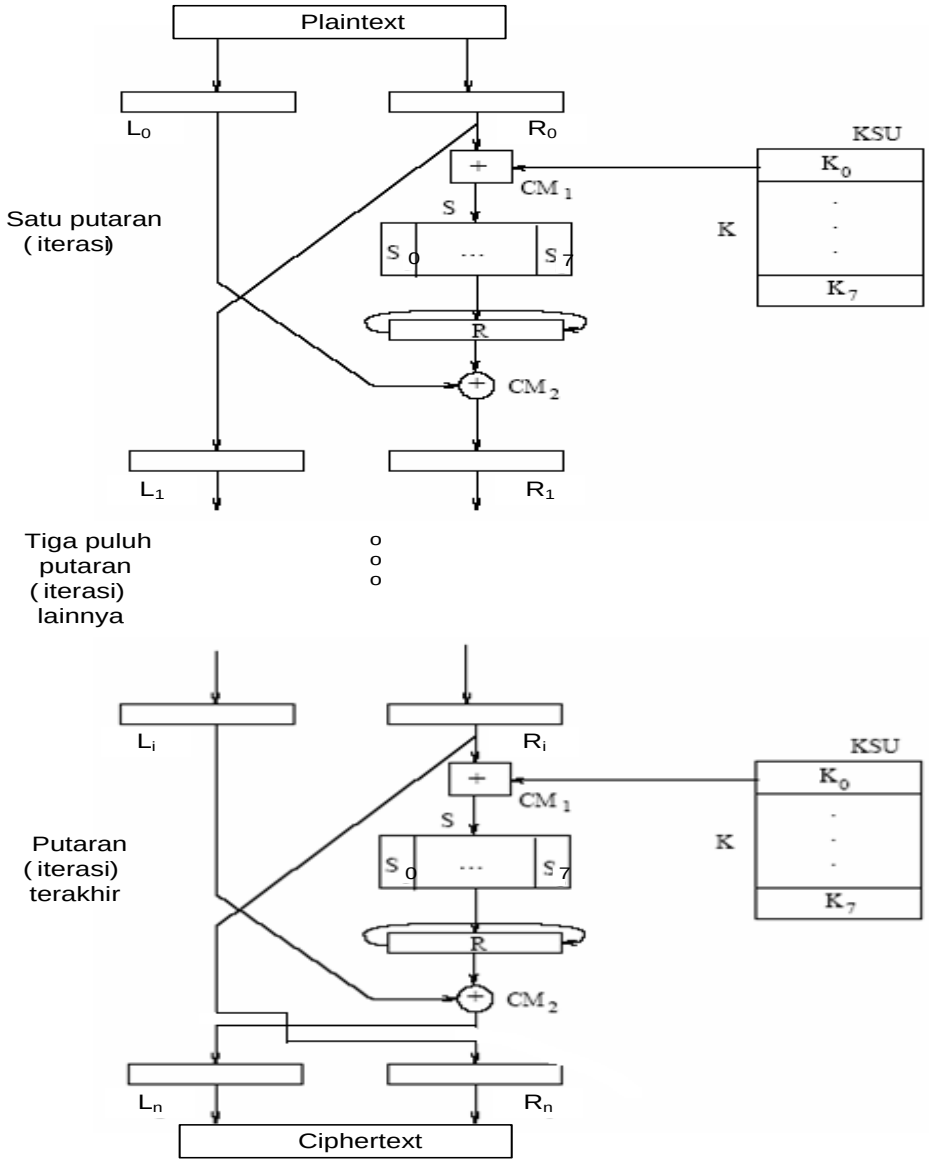
Proses penjumlahan modulo 2^{32}, *S-Box* dan *Rotate Left* dilakukan sebanyak 32 putaran dengan penggunaan kunci sesuai dengan aturan berikut ini:

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 : $K_7, K_6, K_5, \ldots, K_0$

Untuk putaran ke-31, langkah 5 dan 6 agak sedikit berbeda. Langkah 5 dan 6 untuk putaran 31 adalah sebagai berikut:
$L_{32} = L_{31}$ XOR R_{31} setelah proses
$R_{32} = R_{31}$ sebelum dilakukan proses

Sehingga, cipherteks yang dihasilkan adalah
L_{32} : b(32), b(31), …, b(1)
R_{32} : a(32), a(31), …, a(1)
T = a(1), …, a(32), b(1), …, b(32)

Proses enkripsi dari metode GOST dapat digambarkan dalam bentuk skema seperti gambar berikut ini:



Gambar 2.3 Skema Proses Enkripsi Metode GOST

2.3.3. Proses Dekripsi

Proses dekripsi merupakan kebalikan dari proses enkripsi. Penggunaan kunci pada masing-masing putaran pada proses dekripsi adalah sebagai berikut:

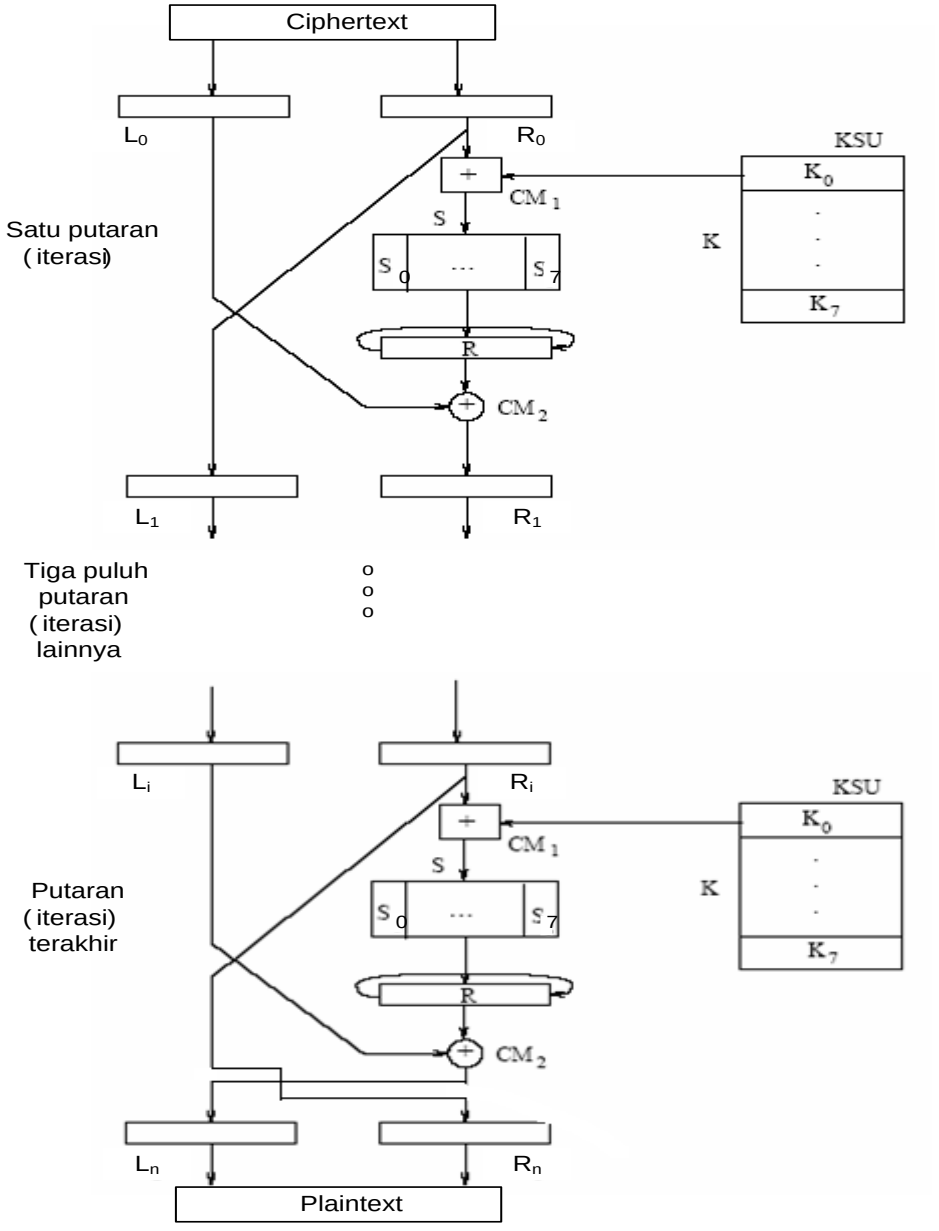
Putaran 0 – 7 : $K_0, K_1, K_2, \ldots, K_7$
Putaran 8 – 15 : $K_7, K_6, K_5, \ldots, K_0$
Putaran 16 – 23 : $K_7, K_6, K_5, \ldots, K_0$
Putaran 24 – 31 : $K_7, K_6, K_5, \ldots, K_0$

Algoritma yang digunakan untuk proses dekripsi sama dengan proses enkripsi dengan aturan untuk langkah 5 dan 6 pada putaran ke-31 adalah sebagai berikut:

L_{32}= L_{31} XOR R_{31} setelah proses
R_{32}= R_{31} sebelum dilakukan proses

Plainteks yang dihasilkan pada proses dekripsi adalah:
L_{32} = b(32), b(31), …, b(1)
R_{32} = a(32), a(31), …, a(1)
P = a(1), …, a(32), b(1), …, b(32)

Proses dekripsi dari algoritma GOST dapat dilihat dari gambar 2.4 berikut:



Gambar 2.4 Skema Proses Dekripsi Metode GOST

2.4. *Avalanche Effect (AE)*

Avalanche Effect adalah pengaruh perubahan 1 bit pada plainteks atau kunci yang menyebabkan perubahan yang signifikan terhadap cipherteks. Suatu algoritma dikatakan memiliki *avalanche effect* yang baik jika perubahan satu bit saja pada input menghasilkan perubahan sekitar setengah jumlah bit total pada outputnya. Salah satu fungsi a*valanche effect* adalah untuk melihat tingkat keamanan yang menjadi acuan untuk menentukan baik atau tidaknya sebuah algoritma kriptografi. Nilai *Avalanche Effect* dapat dihitung dengan persamaan berikut ini:

$$AE = \frac{\text{jumlah bit berubah}}{\text{jumlah bit total}} \times 100\%$$

Keterangan:

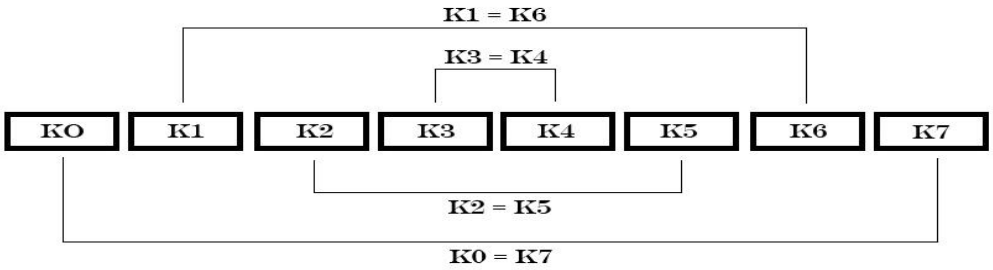
- Jumlah bit berubah = jumlah bit-bit cipherteks yang berubah akibat perubahan 1 bit pada plainteks maupun kunci.
- Jumlah bit total = jumlah bit total cipherteks.

2.5. Kunci Lemah dan Kunci Setengah Lemah

Dalam kriptografi dikenal istilah kunci lemah (*weak-key*) dan kunci setengah lemah (*semi weak-key*). Kunci lemah adalah kunci yang apabila mengenkripsi suatu plainteks kemudian dienkripsi lagi menggunakan kunci yang sama, maka cipherteksnya akan menghasilkan plainteks itu sendiri. Sedangkan yang disebut dengan kunci setengah lemah adalah sepasang kunci yang memiliki sifat jika sebuah plainteks dienkripsi dengan suatu kunci, maka akan dapat dienkripsi dengan kunci yang lain yang merupakan pasangannya. Adanya kunci lemah pada suatu algoritma kriptografi disebabkan oleh mekanisme penjadwalan kunci atau pembangkitan subkunci. Misalkan K_L adalah kunci lemah, E adalah fungsi enkripsi, D adalah fungsi dekripsi, P adalah plainteks, dan C adalah cipherteks, maka persamaan berikut menunjukan fenomena kunci lemah:

$$E_{KL}(P) = C$$
$$D_{KL}(C) = E_{KL}(C) = P$$

GOST merupakan salah satu algoritma kriptografi yang memiliki banyak kunci lemah yaitu sebanyak $2^{256/2}$ atau 2^{128} kunci lemah. Hal ini disebabkan oleh proses penjadwalan kunci pada GOST yang sangat sederhana. Adapun struktur kunci lemah pada GOST ditunjukkan oleh gambar berikut:



Gambar 2.5 Struktur Kunci Lemah pada Algoritma GOST

2.6. *Related Key Attack (RKA)*

Related-key Attack pertama kali diperkenalkan oleh Eli Biham dan Adi Shamir. Serangan ini merupakan jenis serangan *chosen key attack* yang menyerang kelemahan algoritma penjadwalan kunci pada suatu *block cipher*. Dalam serangan ini, *attacker* memperoleh cipherteks dari beberapa plainteks tertentu yang dienkripsi dengan beberapa kunci yang memiliki hubungan atau relasi satu sama lain. Sebagai contoh, kedua kunci tersebut memiliki relasi yaitu hanya memiliki perbedaan 1 bit saja. Adapun tujuan dari tipe serangan ini adalah menemukan kunci rahasia yang dipakai untuk mengenkripsi plainteks dan hal itu dapat dicapai apabila penyerang dapat memilih dan menemukan hubungan diantara kunci-kunci tersebut [3]. Serangan ini akan menurunkan kompleksitas serangan dari *brute force attack* [4].

GOST merupakan salah satu contoh *block cipher* yang dapat diserang dengan menggunakan metode *related-key attack*. Hal ini dikarenakan proses pembentukan subkunci pada GOST yang sangat sederhana [2]. Adapun contoh serangan *related-key* pada algoritma GOST, misalkan *attacker* diasumsikan mengetahui hubungan dari 2 buah kunci berukuran 256 bit yang hanya memiliki perbedaan satu bit yaitu pada bit ke-i maka dapat ditulis dengan persamaan:

$$K \oplus K' = e_i$$

dimana e_i merupakan 256 bit string biner, dimana bit ke-i nilainya satu dan bit yang lainya bernilai 0. Maka hubungan/ relasi diantara dua kunci tersebut dapat didefinisikan sebagai berikut:

$$K0' = K0$$
$$K1' = K1$$
$$K2' = K2$$
$$K3' = K3$$
$$K4' = K4$$
$$K5' = K5$$
$$K6' = K6$$
$$K7' = K7 \oplus e_i$$

Dengan melakukan analisis terhadap kunci yang memiliki relasi seperti dijelaskan di atas, maka *attacker* akan dapat melewatkan 7 putaran pertama dengan mudah karena tidak terdapat perbedaan pada subkunci yang digunakan. Hal ini menyebabkan cipherteks yang dihasilkan pada 7 putaran pertama akan sama. Dengan menggunakan analisis yang telah diperoleh dari relasi kunci tersebut, maka *attacker* dapat menemukan n-bit subkunci untuk putaran tertentu dengan tingkat kesuksesan sekitar 91,7% [11].

2.7. Fungsi Hash Satu Arah

2.7.1. Pengenalan

Fungsi hash adalah fungsi yang menerima masukan string yang panjangnya sembarang dan mengonversinya menjadi string keluaran yang panjangnya tetap (*fixed*), yang umumnya berukuran jauh lebih kecil daripada ukuran semula. Fungsi hash satu arah (*one-way hash function*) berfungsi sebagai:

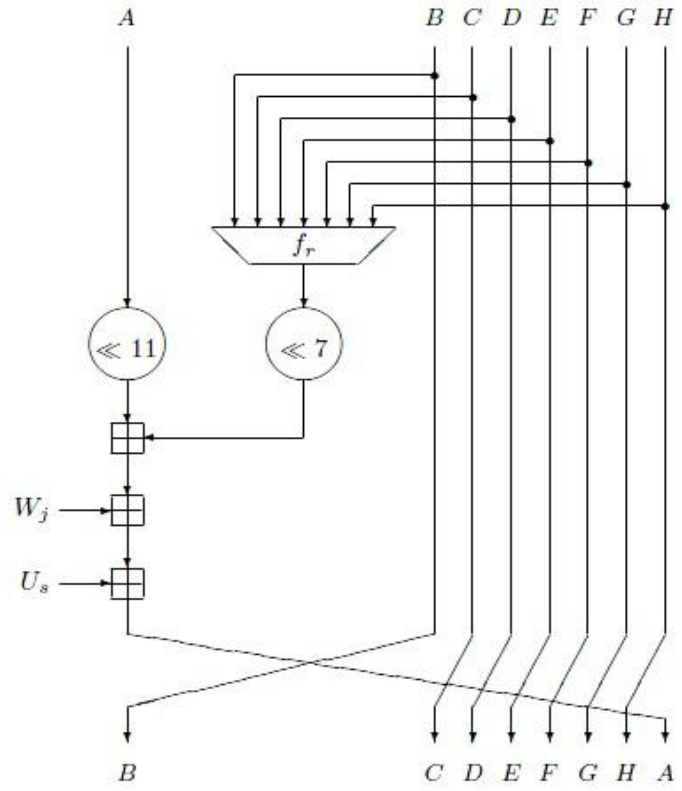
1. Sidik jari (*fingerprint*): Membuat sidik jari dari suatu dokumen atau pesan M yang mana sidik jari merupakan suatu identitas dari si pengirim pesan.
2. Fungsi kompresi: Fungsi kompresi, dokumen D (yang besarnya dapat bervariasi) yang akan di-hash disebut *pre-image*, sedangkan keluarannya yang memiliki ukuran tetap dalam bentuk aslinya dan pada dasarnya masukan lebih besar daripada keluaran, seolah-olah mengalami kompresi, namun hasil dari kompresi tidak bisa dikembalikan ke bentuk awalnya.
3. *Messages digest*: dianggap intisari dari suatu dokumen, namun tidak demikian karena intisari dokumen merupakan ringkasan dokumen yang dapat dipahami maknanya. *Messages digest* tidak demikian, karena dengan sidik jari orang lain tidak mengerti asli dari dokumen tersebut.

Sifat dari fungsi hash adalah sebagai berikut [2]:

1. Diberikan D, mudah dihitung H(D) = h.
2. Diberikan h, yang tidak mungkin untuk mendapatkan M^1 sehingga H(D) = h.
3. Diberikan D, sulit untuk bisa mendapatkan (D^1), sehingga H(D) = H(D^1). Jika diperoleh dokumen seperti ini maka disebut dengan *collision* (tabrakan).
4. Sulit untuk mendapatkan dua dokumen D dan D^1 sehingga H(D) = H(D^1). Kebanyakan kriptanalis berhasil mendapat D^1 sehingga H(D) = H(D^1) yang berarti algoritma dari fungsi hash belum berhasil dibobol. Dibutuhkan waktu lama untuk bisa menjebol algoritma fungsi *hash*. Contoh algoritma fungsi *hash* satu arah adalah HAVAL, MD-4, MD-5, SHA dan SHS.

2.7.2. Fungsi HAVAL-256

HAVAL merupakan sebuah fungsi hash yang memiliki 15 level keamanan yang berbeda – beda, yang jumlah putarannya dapat dipilih di antara 3, 4, dan 5, dan panjang *hash* dapat dipilih di antara 128 bit sampai dengan 256 bit dengan peningkatan sebanyak 32 bit. Struktur HAVAL hampir menyerupai MD4. Panjang blok HAVAL adalah 1024 bit. Metode *padding* dari MD4 telah diperluas. Setelah dilakukan *padding* dengan bit 1 atau 0, 16 bit tambahan dibuat yang merupakan gabungan versi yang digunakan (3 bit), jumlah putaran dari fungsi kompresi (3 bit), dan panjang *hash* (10 bit) yang kemudian ditambahkan sebelum 64 bit panjang pesan. HAVAL akan melakukan *padding* terlebih dahulu terhadap pesan tersebut dan panjang pesan tersebut setelah dilakukan *padding* ialah kelipatan 1024. *Padding* tersebut tetap dilakukan walaupun panjang pesan telah mencapai kelipatan 1024. Blok terakhir dari pesan yang telah di-*padding* mengandung jumlah bit dari pesan asli yang belum di-*padding*, jumlah bit yang dibutuhkan di dalam *digest*, dan jumlah putaran setiap blok pesan yang diproses. Secara umum, proses pembentukan *message digest* pada HAVAL dapat dilihat dari gambar berikut ini [18]:



Gambar 2.6 Proses HAVAL

Misalkan pesan yang telah di-*padding* itu adalah B_{n-1}, B_{n-2}, …, B_0, dimana setiap B_i merupakan sebuah blok 1024 bit. HAVAL dimulai dari blok B_0 dan sebuah 8-*word* (256 bit) string konstan $D_0 = D_{0,7}D_{0,6}…D_{0,0}$, yang diambil dari hasil pecahan $\Pi = 3.1415…$, dan memproses pesan B_{n-1}, B_{n-2}, …, B_0 secara blok per blok. Lebih tepatnya, pesan tersebut dipadatkan dengan mengulangi perhitungan:

$$D_{i+1} = H(D_i, B_i)$$

dimana i merupakan angka yang berada pada range 0 sampai *n-1* dan H disebut sebagai *updating algorithm* dari HAVAL. Akhirnya, 256 bit terakhir dari string D_n disesuaikan dengan panjang *digest* yang dispesifikasikan di blok terakhir B_{n-1} dan *string* tersebut menjadi *message digest* dari pesan M. Proses HAVAL tersebut dapat dirangkum ke dalam tiga langkah, antara lain [18]:

1. *Padding* pesan M sehingga panjangnya menjadi kelipatan 1024. Blok terakhir dari pesan yang telah di-*padding* mengindikasikan panjang dari pesan asli M sebelum di-*padding*, panjang yang dibutuhkan dari *digest* M, jumlah putaran setiap blok yang diproses dan versi HAVAL yang digunakan.
2. Hitung secara berulang $D_{i+1} = H(D_i, B_i)$ dengan i dari 0 sampai *n-1* dimana D_0 adalah sebuah *string* tetap 8-*word* (256 bit) dan n adalah jumlah total blok yang ada di dalam pesan yang telah di-*padding*.
3. Sesuaikan nilai 256-bit D_n yang diperoleh berdasarkan perhitungan di atas berdasarkan panjang *digest* yang telah dispesifikasikan pada blok terakhir B_{n-1} dan hasil dari penyesuaian nilai tersebut menjadi *message digest* dari pesan M.

2.7.2.1. Padding

Tujuan dari melakukan *padding* pada pesan adalah untuk membuat panjang pesan menjadi *kelipatan* 1024 dan membiarkan pesan mengindikasikan panjang dari pesan asli, jumlah bit *digest*, jumlah putaran dan versi dari HAVAL yang digunakan. HAVAL menggunakan 64-bit *field* MSGLENG untuk mendeskripsikan panjang pesan yang belum di*padding*, 10-bit *field* DGSTLENG untuk mendeskripsikan jumlah bit yang dibutuhkan di dalam sebuah *digest*, 3-bit *field* PASS untuk mendeskripsikan jumlah putaran dari setiap blok pesan yang diproses dan 3-bit *field* VERSION untuk mengindikasikan versi HAVAL. Jumlah bit di dalam *digest* dapat berupa 128, 160, 192, 224, dan 256 sedangkan jumlah putaran dapat berupa 3, 4, dan 5. Untuk versi HAVAL diisi dengan angka 1. HAVAL melakukan *padding* terhadap pesan dengan menambahkan bit 1 setelah *most significant bit* dari pesan, yang kemudian diikuti dengan bit 0 sampai panjang pesan yang baru mencapai 944 mod 1024. Kemudian HAVAL akan melakukan *padding* dengan bit VERSION, PASS, DGSTLENG, dan MSGLENG.

2.7.2.2. Updating Algorithm H

Algoritma ini memproses sebuah blok ke dalam 3, 4, atau 5 putaran, sesuai dengan yang didefinisikan di dalam *field* PASS pada blok terakhir dari pesan. Kelima putaran tersebut dinotasikan dengan H1, H2, H3, H4, dan H5. Misalkan masukan untuk H ialah (D_{in}, B) dimana D_{in} adalah *string* berukuran 8-*word* dan B adalah blok dengan ukuran 32-*word* (1024 bit) maka D_{out} merupakan hasil dari fungsi H terhadap masukan (D_{in}, B). Proses H dapat digambarkan sebagai berikut [18]:

$$
\begin{aligned}
E_0 &= D_{in};\\
E_1 &= H_1(E_0, B);\\
E_2 &= H_2(E_1, B);\\
E_3 &= H_3(E_2, B);\\
E_4 &= H_4(E_3, B);\ \text{(if PASS=4, 5)}\\
E_5 &= H_5(E_4, B);\ \text{(if PASS=5)}\\
D_{out} &= \begin{cases} E_3 \boxplus E_0 \text{ if PASS=3} \\ E_4 \boxplus E_0 \text{ if PASS=4} \\ E_5 \boxplus E_0 \text{ if PASS=5} \end{cases}
\end{aligned}
$$

Setiap H1, H2, H3, H4, dan H5 memiliki 32 putaran operasi dan setiap putaran akan memproses *word* yang berbeda dari B. Urutan *word* yang akan diproses oleh B berbeda – beda tergantung oleh putarannya. Fungsi yang ada di dalam H1, H2, H3, H4, dan H5 ialah sebagai berikut [18]:

$$
\begin{aligned}
f_1(x_6, x_5, x_4, x_3, x_2, x_1, x_0) &= x_1x_4 \oplus x_2x_5 \oplus x_3x_6 \oplus x_0x_1 \oplus x_0\\
f_2(x_6, x_5, x_4, x_3, x_2, x_1, x_0) &= x_1x_2x_3 \oplus x_2x_4x_5 \oplus x_1x_2 \oplus x_1x_4 \oplus\\
&\quad x_2x_6 \oplus x_3x_5 \oplus x_4x_5 \oplus x_0x_2 \oplus x_0\\
f_3(x_6, x_5, x_4, x_3, x_2, x_1, x_0) &= x_1x_2x_3 \oplus x_1x_4 \oplus x_2x_5 \oplus x_3x_6 \oplus x_0x_3 \oplus x_0\\
f_4(x_6, x_5, x_4, x_3, x_2, x_1, x_0) &= x_1x_2x_3 \oplus x_2x_4x_5 \oplus x_3x_4x_6 \oplus\\
&\quad x_1x_4 \oplus x_2x_6 \oplus x_3x_4 \oplus x_3x_5 \oplus\\
&\quad x_3x_6 \oplus x_4x_5 \oplus x_4x_6 \oplus x_0x_4 \oplus x_0\\
f_5(x_6, x_5, x_4, x_3, x_2, x_1, x_0) &= x_1x_4 \oplus x_2x_5 \oplus x_3x_6 \oplus x_0x_1x_2x_3 \oplus x_0x_5 \oplus x_0
\end{aligned}
$$

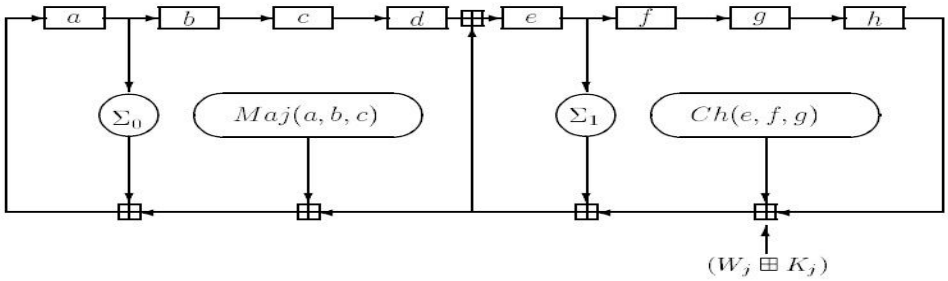
2.7.2.3. Mengubah Hasil Terakhir dari H

$D_n = D_{n,7}D_{n,6}\ldots D_{n,0}$ merupakan hasil dari H yang berukuran 256 bit. Dn akan langsung menjadi *message digest* jika panjang *message digest* yang diminta adalah 256 bit. Jika ternyata panjang yang diminta berbeda, maka Dn tersebut akan diubah menjadi *string* dengan panjang yang sesuai.

2.7.3. Fungsi SHA-256

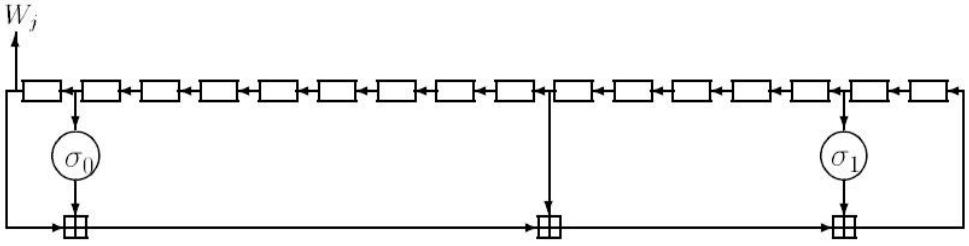
SHA dibuat oleh NIST dan digunakan bersama DSS (*Digital Signature Standard*). SHA dinyatakan sebagai standard fungsi hash satu-arah. SHA dapat dianggap sebagai kelanjutan pendahulunya yaitu MD5 yang telah digunakan secara luas. SHA disebut aman (*secure*) karena ia dirancang sedemikian rupa sehingga secara komputasi tidak mungkin menemukan pesan yang berkorespondensi dengan *message digest* yang diberikan. Anggota pertama keluarga SHA adalah SHA-0 yang dipublikasikan pada tahun 1993. SHA-0 sering diacu sebagai SHA saja. Berikutnya pada tahun 1995, SHA-1 dipublikasikan. Empat varian lain juga telah dipublikasikan yaitu SHA-224, SHA-256, SHA-384 dan SHA-512. Keempat varian ini diacu sebagai SHA-2.

SHA-256 sebagai salah satu varian dari SHA-2 beroperasi hampir sama seperti MD4, MD5, dan SHA-1, yaitu menerima masukan berupa pesan dengan ukuran maksimum 2^{64} bit (2.147.483.648 gigabyte) akan tetapi menghasilkan *message digest* yang panjang 256 bit, lebih panjang dari *message digest* yang dihasilkan oleh MD5 yang hanya 128 bit dan SHA-1 sepanjang 160 bit saja. Dalam SHA-256 terdapat 2 buah komponen utama, yaitu fungsi kompresi *(compression function)* dan penjadwalan pesan (*message schedule)*. Fungsi kompresi pada SHA-256 dapat dilihat dari gambar berikut ini [1]:



Gambar 2.7 Fungsi Kompresi pada SHA-256

Sedangkan penjadwalan pesan pada SHA-256 dapat digambarkan sebagai berikut:



Gambar 2.8 Penjadwalan Pesan pada SHA-256

Adapun langkah-langkah pembuatan *message digest* dengan SHA-256 secara garis besar adalah sebagai berikut [1]:

1. *Message Padding*
   Pesan ditambah dengan sejumlah bit pengganjal sedemikian sehingga panjang pesan (dalam satuan bit) kongruen dengan 448 modulo 512. Angka 512 ini muncul karena SHA-256 memproses pesan dalam blok-blok yang berukuran 512 bit. Pesan dengan panjang 448 bit pun tetap ditambah dengan bit-bit pengganjal. Bit-bit pengganjal terdiri dari sebuah bit 1 diikuti dengan sisanya bit 0. Pada *message padding*, akan ditambahkan satu buah bit "1" diikuti oleh k buah bit "0" serta 64 bit integer yang merupakan nilai desimal dari panjang pesan asli yang ditulis dalam biner. Sebagai contoh pesan (dalam ASCII) "abc" mempunyai panjang 24 bit. Sebelum pesan menjadi input SHA-256, dilakukan *message padding* sebagai berikut:

   01100001 01100010 01100011 **1** **000...0** (423) **000...011000** (64)

2. Inisialisasi penyangga (*buffer*) *message digest* (MD)
   SHA-256 membutuhkan 8 buah penyangga (*buffer*) yang masing-masing panjangnya 32 bit. Total panjang penyangga adalah 8 x 32 = 256 bit. Setiap penyangga diinisialisasi dengan nilai-nilai (dalam HEX) sebagai berikut:
   $H(0)^1$ = 6a09e667
   $H(0)^2$ = bb67ae85
   $H(0)^3$ = 3c6ef372
   $H(0)^4$ = a54ff53a
   $H(0)^5$ = 510e527f
   $H(0)^6$ = 9b05688c
   $H(0)^7$ = 1f83d9ab
   $H(0)^8$ = 5be0cd19

3. Pengolahan pesan dalam blok berukuran 512 bit.
   Blok pesan M dibagi menjadi N buah blok $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$ masing-masing sepanjang 512 bit. 32 bit pesan pada blok ke-i ditulis dengan $M_0^{(i)}$, 32 bit berikutnya $M_1^{(i)}$ dan seterusnya sampai dengan 32 bit terakhir $M_{15}^{(i)}$. Setiap blok 512 bit diproses bersama dengan penyangga MD menjadi keluaran 256 bit dan ini disebut proses H_{SHA}. Proses H_{SHA} terdiri dari 64 buah putaran dan masing-masing putaran menggunakan bilangan penambah K_0, ..., K_{63} yang berbeda-beda, yaitu [1]:

428a2f98	71374491	b5c0fbcf	e9b5dba5	3956c25b	59f111f1	923f82a4	ab1c5ed5
d807aa98	12835b01	243185be	550c7dc3	72be5d74	80deb1fe	9bdc06a7	c19bf174
e49b69c1	efbe4786	0fc19dc6	240ca1cc	2de92c6f	4a7484aa	5cb0a9dc	76f988da
983e5152	a831c66d	b00327c8	bf597fc7	c6e00bf3	d5a79147	06ca6351	14292967
27b70a85	2e1b2138	4d2c6dfc	53380d13	650a7354	766a0abb	81c2c92e	92722c85
a2bfe8a1	a81a664b	c24b8b70	c76c51a3	d192e819	d6990624	f40e3585	106aa070
19a4c116	1e376c08	2748774c	34b0bcb5	391c0cb3	4ed8aa4a	5b9cca4f	682e6ff3
748f82ee	78a5636f	84c87814	8cc70208	90befffa	a4506ceb	bef9a3f7	c67178f2

Terdapat enam buah fungsi logika dalam SHA-256, yaitu [1]:

$$Ch\ (x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$$
$$Maj\ (x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$
$$\sum_0 (x) = S^2(x) \oplus S^{13}(x) \oplus S^{22}(x)$$
$$\sum_1 (x) = S^6(x) \oplus S^{11}(x) \oplus S^{25}(x)$$
$$\sigma_0 (x) = S^7(x) \oplus S^{18}(x) \oplus R^3(x)$$
$$\sigma_1 (x) = S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x)$$

Catatan: operator logika AND, NOT, XOR masing-masing dilambangkan dengan $\wedge$, $\neg$, $\oplus$

Pada proses penjadwalan pesan, nilai W_j untuk j = 0,1, ..., 15 berasal dari 16 *word* pada blok yang sedang diproses, sedangkan nilai W_j berikutnya didapatkan dari persamaan [1]:

$$W_j \leftarrow \sigma_1 (W_{j-2}) + W_{j-7} + \sigma_0 (W_{j-15}) + (W_{j-16})$$

2.8. Landasan Matematis Kriptografi

2.8.1. Penjumlahan Modulo

Pada metode GOST, operasi aritmetika modular yang dipakai adalah operasi penjumlahan modulo 2^{32}. Operasi penjumlahan modulo ini ditujukan untuk mengontrol panjang bit data agar hasil penjumlahan tidak lebih dari 32 bit. Contoh penjumlahan modulo 2^{32} dapat dilihat pada contoh di bawah ini:

```
A =     1011 0111 1101 1111 0110 0011 1100 0111
B =     1110 1010 0101 0110 1100 1010 0101 1101 +
C =  1 1010 0010 0011 0110 0010 1110 0010 0100
```

Hasil penjumlahan di atas berjumlah 33 bit. Selanjutnya, dilakukan proses modulo 2^{32} dan hasilnya dapat dilihat di bawah ini:

```
C  =  1 1010 0010 0011 0110 0010 1110 0010 0100
C  =  7016427044 (bentuk desimal)
C  =  7016427044 mod 2^32
C  =  2721459748 (bentuk desimal)
C  =  1010 0010 0011 0110 0010 1110 0010 0100(bentuk biner)
```

Dari hasil perhitungan di atas, terlihat bahwa hasil operasi penjumlahan modulo 2^{32} akan membuang bit ke-33. Bila hasil penjumlahan tidak melebihi 32 bit maka operasi modulo akan mendapatkan hasil yang sama seperti operasi penjumlahan [17].

2.8.2. Operasi XOR

Operasi XOR (*Exclusive OR*) dilambangkan dengan simbol ⊕. Kemungkinan-kemungkinan nilai operasi XOR ini dapat dilihat pada tabel 2.2 berikut ini:

Tabel 2.2 Operasi XOR

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

2.8.3. Rotasi Bit

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 kesisi 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 "<<<". Berikut contoh operasi *Rotate Left*.
   - 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 ">>>". Berikut contoh operasi *Rotate Right* [17]:
   - 00111111 >>> 1 = 10011111
   - 01111110 >>> 2 = 10011111
   - 11111000 >>> 3 = 00011111

2.9. Means Square Error (MSE)

Analisis performansi dilakukan dengan menghitung *avalanche effect* sesuai dengan teori *Strict Avalanche Criterion (SAC),* dimana suatu algoritma dikatakan baik apabila *avalanche effect*-nya sebesar setengahnya atau 50 %. Dengan demikan dilakukan perhitungan selisih rata-rata untuk mendapatkan metode yang terbaik dengan menggunakan perhitungan *means square error* dengan persamaan berikut:

$$MSE(\theta) = E_\theta[(\delta(\mathbf{X}) - \theta)^2]$$

dimana

ø = 50 (parameter untuk nilai *avalanche effect* terbaik)

X = rata-rata *distance error* ke ø

3. ANALISIS DAN PERANCANGAN SISTEM

Pada bab ini akan dibahas mengenai tahap analisis kebutuhan dan perancangan sistem yang akan dibangun. Analisis sistem dilakukan dengan menggunakan metoda analisis terstruktur.

3.1. Analisis Kebutuhan Sistem

Dari analisa yang didapat, sistem yang dibangun dalam Tugas Akhir ini harus memenuhi kebutuhan sebagai berikut:

1. Sistem dapat melakukan proses enkripsi dan proses dekripsi terhadap teks.
2. Sistem mengimplementasikan algoritma kriptografi GOST dengan penambahan fungsi hash HAVAL-256 dan SHA-256 pada algoritma penjadwalan kuncinya.
3. *User* dapat memasukkan sendiri kunci yang akan digunakan.
4. Aplikasi dapat menampilkan proses pembentukan kunci.
5. Sistem dapat menampilkan waktu *real-time* yang dibutuhkan untuk melakukan proses enkripsi dan proses dekripsi.
6. Sistem dapat menampilkan nilai *Avalanche Effect* dari modifikasi algoritma GOST.
7. Sistem dapat membuka (*load*) dan menyimpan (*save*) file teks hasil enkripsi atau dekripsi.

3.2. Perancangan Sistem

Untuk mengimplementasikan algoritma pengembangan GOST dalam Tugas Akhir ini dirancang sebuah aplikasi yang sederhana, namun memuat seluruh aspek baik proses enkripsi, dekripsi maupun pembentukan *sub-key*. Perancangan aplikasi ini dilakukan dengan menggunakan metode desain yang berorientasi aliran data. Tools yang digunakan berupa Diagram Aliran Data, Spesifikasi Proses, dan Kamus Data.

Adapun alasan penggunaan perancangan yang berorientasi objek pada aliran data adalah karena Diagram Aliran Data memiliki sejumlah sifat yang dapat digunakan untuk enjamin kejelasan dari sistem yang digambarkan, kelengkapan penggambaran, serta tidak menimbulkan ambiguitas.

3.2.1. Diagram Aliran Data

3.2.1.1. Diagram Konteks (Diagram Aliran Data Level 0)

Perancangan dimulai dengan pembentukan diagram konteks yang merupakan penggambaran sistem secara global. Penggambaran tersebut meliputi:

1. Keseluruhan sistem yang dimodelkan sebagai suatu proses.
2. Entiti-entiti luar yang berinteraksi dengan sistem.
3. Aliran data yang terjadi diantara sistem dengan entiti-entiti luar.