IMPLEMENTASI *PSEUDO RANDOM NUMBER GENERATOR* (PRNG) MENGGUNAKAN ALGORITME *MERSENNE TWISTER* (MT)

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI REKAYASA KOMPUTER

Diajukan untuk Memenuhi Persyaratan

Memperoleh Gelar Sarjana Teknik

Disusun oleh:

WAHYU ADHITIYA RACHMAN

NIM. 0810633086 - 63

KEMENTERIAN RISET, TEKNOLOGI, DAN PENDIDIKAN TINGGI

UNIVERSITAS BRAWIJAYA

FAKULTAS TEKNIK

MALANG

2015

LEMBAR PERSETUJUAN

IMPLEMENTASI *PSEUDO RANDOM NUMBER GENERATOR* (PRNG) MENGGUNAKAN ALGORITME *MERSENNE TWISTER* (MT)

SKRIPSI

TEKNIK ELEKTRO

KONSENTRASI REKAYASA KOMPUTER

Diajukan untuk Memenuhi Persyaratan
Memperoleh Gelar Sarjana Teknik

Disusun oleh:
WAHYU ADHITIYA RACHMAN
NIM. 0810633086 - 63

Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Dr. Ir. Muhammad Aswin, M.T.	**Adharul Muttaqin, ST., M.T.**
NIP. 19640626 199002 1 001	**NIP. 19760121 200501 1 001**

PENGANTAR

Puji syukur kehadirat Allah SWT, Sang Maha Pencipta yang telah memberikan limpahan rahmat dan hidayah sehingga penulis dapat menyelesaikan Tugas Akhir dengan judul "Implementasi *Pseudo Random Number Generator* (PRNG) Menggunakan Algoritme *Mersenne Twister* (MT)" sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik di Jurusan Teknik Elektro Fakultas Teknik Universitas Brawijaya Malang.

Tidak banyak yang bisa penulis sampaikan kecuali ungkapan terima kasih kepada berbagai pihak yang telah dengan tulus ikhlas memberikan bimbingan, arahan, dan dukungan hingga penulisan tugas akhir ini dapat terselesaikan. Pada kesempatan kali ini, penulis mengucapkan banyak terima kasih kepada:

1. Bapak M. Azis Muslim, S.T., M.T., Ph.D. selaku Ketua Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya.
2. Bapak Hadi Suyono, S.T., M.T., Ph.D. selaku Sekretaris Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya.
3. Bapak Ali Mustofa, S.T., M.T. selaku Ketua Program Studi Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya.
4. Bapak Dr. Ir. Muhammad Aswin, M.T. selaku dosen pembimbing I yang telah banyak memberikan bimbingan, masukan dan arahan dalam penyusunan tugas akhir ini.
5. Bapak Adharul Muttaqin, S.T., M.T. selaku dosen pembimbing II yang telah banyak memberikan bimbingan, masukan dan arahan dalam penyusunan tugas akhir ini.
6. Bapak Waru Djurianto, S.T., MT., selaku penguji dan KKDK konsentrasi Teknik Informatika dan Komputer.
7. Bapak Raden Arief Setyawan, S.T., M.T. selaku penguji dalam ujian skripsi dan dosen pembimbing akadekmik saat ini.
8. Bapak Dr. Eng. Panca Mudjirahardjo, S.T., M.T. selaku penguji dalam ujian skripsi
9. Bapak dan Ibu Dosen serta seluruh karyawan jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya.

10. Ayahanda Ir. Dwi Admojo Kristyantyo, ibunda Sri Martianingsih serta keluarga besar saya yang telah memberikan banyak dukungan secara moral maupun materi.
11. Seluruh keluarga besar Margono dan keluarga besar Mariati yang telah banyak memberi motifasi dan dukungan dalam mengerjakan tugas akhir ini.
12. Saudara Ferdy Hendrawan, saudari Nimas Wahyu Choironik dan teman – teman E08 selaku teman seperjuangan dalam mengerjakan tugas akhir ini yang telah memberikan banyak motivasi dan dukungan sarana dan prasarana.
13. Saudara Zulhad Aliansyah, S.T.dan Fikri Aulia, S.T. yang telah membantu dalam mengerjakan tugas akhir ini.
14. Seluruh teman – teman Gang IV no 12a Robith, Aditya, Royun, Yudhi, Yudha, Aryo, Hendra, Theo Malang yang secara langsung maupun tidak langsung telah memberikan banyak motivasi dan dukungan sarana dan prasarana.
15. Seluruh teman – teman Chummy Community Denpasar Meita, Dewi, Lili, Dyan, Thedy, Panji, Irfan yang secara langsung maupun tidak langsung telah memberikan banyak motivasi dan dukungan sarana dan prasarana.
16. Saudari Rindi Lestari beserta keluarga yang telah banyak memberi motifasi dan dukungan dalam mengerjakan tugas akhir ini
17. Seluruh teman - teman mahasiswa Teknik Elektro (terutama asisten Laboratorium Komputer)
18. Seluruh Follower adith_3977, yang banyak memberikan banyak referensi dan pengetahuan yang berhubungan dengan skripsi ini.
19. Seluruh member J-Rockstars, yang banyak memberikan banyak referensi dan pengetahuan yang berhubungan dengan skripsi ini.
20. Serta semua pihak yang tidak bisa disebutkan satu – per satu baik yang terlibat secara langsung dan tidak langsung demi terselesaikannya skripsi ini.

Penulis menyadari sepenuhnya bahwa Tugas Akhir ini masih banyak kekurangan. Segala kritik dan saran yang membangun akan penulis terima demi kesempurnaan skripsi ini. Harapan penulis semoga skripsi ini bermanfaat bagi pembaca dan khususnya mahasiswa jurusan teknik elektro dimasa yang akan datang.

Malang, Agustus 2015

Penyusun

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Wahyu Adhitiya Rachman, Jurusan Teknik Elektro, Fakultas Teknik, Universitas Brawijaya, Agustus 2015, *Implementasi Pseudo Random Number Generator(PRNG) menggunakan Algoritme Mersenne Twister (MT)* , Dosen Pembimbing : Dr. Ir. Muhammad Aswin, M.T., Adharul Muttaqin, ST., MT

Sistem keacakan data adalah hal yang penting dalam aplikasi keamanan data. Karena semakin tinggi acak nilai suatu data kemungkinan data itu ditebak akan semakin kecil.

Pseudo Random Number Generator (PRNG) yang ada saat ini masih memiliki tingkat ke acakan yang rendah. Diharapkan dengan menggabungkan *multi* PRNG dapat meningkatkan nilai keacakan dalam suatu data sehingga data menjadi lebih sulit untuk di tebak dan hasil keluaran menjadi semakin baik. Algoritme yang digunakan adalah *Mersenne Twister* karena jika dibandingkan dengan algorime lainnya *Mersenne Twister* memiliki tingkat keacakan data yang cukup tinggi serta proses eksekusinya yang memakan waktu singkat serta bit yang digunakan juga relatif lebih kecil yaitu 32-bit. Hasil pengujian dari *Mersenne Twister* akan di autokorelasi untuk mengetahui ada tidaknya perulangan data.

Kata Kunci : Autokorelasi, *Mersenne Twister*, *Random Number*, PRNG.

BAB I
PENDAHULUAN

1.1 Latar Belakang

Masalah keamanan dan kerahasiaan data merupakan suatu aspek yang penting. Pesan, data, atau informasi akan tidak berguna lagi apabila di tengah jalan disadap atau dibajak oleh orang yang tidak berkepentingan. Kriptografi merupakan solusi dalam menangani masalah keraharasiaan dan keamanan data. Kriptografi merupakan ilmu dan seni untuk mengamankan pesan. Seiring dengan perkembangan zaman, metode kriptografi diterapkan dalam berbagai aspek kebutuhan manusia seperti untuk mengamankan data sehingga saat ini banyak bermunculan algoritme modern.

Mersenne Twister adalah salah satu metode kriptografi modern yang populer karena di samping prosesnya yang memakan waktu lebih singkat, *Mersenne Twister* juga menggunakan memori yang lebih sedikit. *Mersenne Twister* melakukan penggacakan bilangan semu acak dengan menggunakan operasi eksklusif *or* untuk mendapatkan sebuah bilangan acak. (Andresta Ramadhan,2013)

PRNG (*Pseudo Random Number Generator*) adalah salah satu jenis *generator* yang dapat menghasilkan bit semu acak. PRNG sering digunakan karena mampu menghasilkan bit semu acak dengan periode maksimal yang panjang dan mudah diaplikasikan dalam berbagai hal. Namun seiring dengan perkembangan zaman, penggunaan sebuah PRNG sebagai *generator* bit semu acak rawan terhadap serangan kriptanalisis karena bit semu acak yang dihasilkan *generator* PRNG akan mengalami perulangan di setiap periodenya. Oleh karena itu diperlukan fungsi-fungsi tambahan karena kekuatan algoritme *Mersenne Twister* terletak pada keacakan rangkaian bit semu acak yang dihasilkan bukan tergantung pada kerahasiaan algoritmenya.

PRNG menggunakan polinomial dalam menghasilkan bit semu acak. Polinomial berfungsi sebagai fungsi *feedback* dari *generator* PRNG. Penggunaan jenis polinomial mempengaruhi tingkat keacakan bit semu acak yang dihasilkan

PRNG. Berdasarkan semua hal yang telah dijabarkan di atas, maka pada skripsi ini dilakukan pengembangan algoritme enkripsi dekripsi berbasis PRNG dengan menggunakan polinomial primitif sebagai *generator* yang mampu menghasilkan bit semu acak yang lebih tahan terhadap serangan kriptanalisis.

.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan sebelumnya, maka rumusan masalah ditekankan pada :

1. Bagaimana cara memperbaiki algoritme pembangkitan bilangan acak semu0
2. Bagaimana cara mendapatkan keacakan tinggi dengan sumber terbatas

1.3 Batasan Masalah

Agar pembahasan lebih terarah, maka penulis menetapkan beberapa batasan masalah sebagai berikut :

1. Hanya membahas tentang proses *random number*
2. Berjalan pada perangkat dengan sumber daya terbatas
3. Tidak membahas proses enkripsi

1.4 Tujuan

Tujuan dari skripsi ini adalah :

Tujuan tugas akhir ini adalah merancang dan memperbaiki suatu pembangkit bilangan acak semu.

1.5 Manfaat

Manfaat yang diharapkan bisa diperoleh dari skripsi ini adalah dapat memahami serta memperbaiki nilai keacakan algoritme *Mersenne Twister* dengan

menggunakan *multi* PRNG sehingga dapat dikembangakan untuk lebih baik lagi nantinya dapat diterapakan dalam berbagai aktifitas yang ada.

1.6 Sistematika Penulisan

Sistematika penulisan laporan skripsi ini adalah sebagai berikut :

BAB I Pendahuluan

Dalam bab ini berisi tentang latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika penulisan dari laporan tugas akhri ini.

BAB II Tinjauan Pustaka

Dalam bab ini akan dijabarkan mengenai landasan teori yang digunakan serta yang mendukung pelaksanaan penulisan skripsi ini.

BAB III Metodologi Penelitian

Dalam bab ini akan dijelaskan tentang metode yang digunakan penulis untuk menyelesaikan skripsi.

BAB IV Perancangan dan Implementasi

Dalam bab ini akan dijelaskan langkah – langkah dalam perancangan sistem pengecakan angka.

BAB V Pengujian

Dalam bab ini akan disampaikan hasil pengujian dari aplikasi yang telah dibuat.

BAB VI Penutup

Dalam bab ini berisikan kesimpulan dan saran dari penulis.

BAB II
TINJAUAN PUSTAKA

2.1 Kriptografi

Kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan. Kriptografi berasal dari bahasa Yunani yakni *kriptos* yang artinya tersembunyi dan *graphia* yang artinya sesuatu yang tertulis, sehingga kriptografi dapat disebut sebagai sesuatu yang tertulis secara rahasia. Kriptografi merupakan suatu bidang ilmu yang mempelajari tentang bagaimana merahasiakan suatu informasi penting ke dalam bentuk yang tidak dapat dibaca oleh siapapun, serta mengembalikannya kembali menjadi informasi semula dengan menggunakan berbagai macam teknik yang telah ada sehingga informasi tersebut tidak dapat diketahui oleh pihak manapun yang tidak berkepentingan. Terdapat berbagai macam definisi mengenai kriptografi, namun pada intinya kriptografi adalah teknik yang digunakan untuk menjamin keamanan pertukaran data.

2.2 Sejarah Kriptografi

Sebelum era modern, kriptografi hanya bersangkutan dengan keterpercayaan pesan dengan mengubah pesan yang dapat dimengerti menjadi tidak dapat dimengerti, kemudian proses tersebut dibalikkan kembali untuk dapat dibaca. Tujuannyaadalah agar pesan tersebut tidak dapat dibaca oleh penyusup dan mata-mata tanpa pengetahuan khusus yang menyangkut kunci-kunci dekripsi. Enkripsi digunakan untuk menjaga kerahasiaan dalam berkomunikasi, seperti komunikasi yang dilakukan mata-mata, pemimpin militer, dan diplomat.

Tulisan rahasia yang pertama memerlukan tidak lebih dari sekedar pena dan kertas, karena kebanyakan orang pada masa itu tidak dapat membaca. Ketika sudah lebih banyak orang dapat dapat membaca, tentu dibutuhkan kriptografi yang sebenarnya. Cipher klasik yang utama berbentuk cipher transposisi. Chiper transposisi mengubah urutan huruf yang dituliskan dalam sebuah pesan. Sedangkan chiper substitusi mengubah huruf ke dalam huruf lain (misalnya kalimat “abcde” menjadi “efghi”, dengan mengubah setiap huruf menjadi huruf Latin). Contoh chiper substitusi kuno adalah *Caesar cipher*. *Caesar chiper*

menggunakan prinsip mengganti huruf-huruf dalam pesan menjadi beberapa huruf setelah huruf asli. Metode ini dinamai berdasarkan laporan bahwa yang menggunakannya pertama kali adalah Julius Caesar. Ia menggunakannya dengan mengganti huruf pesan asli dengan 3 huruf setelah huruf asli (Caesar menggunakannya untuk berkomunikasi dengan jendral jendralnya saat ekpedisi militer). Penggunaan kriptografi yang paling awal diketahui adalah *ciphertext* yang di pahat pada batu di Mesir (sekitar 1900BC), tapi hal ini mungkin dilakukan hanya untuk kesenangan kaum terpelajar. Penggunaan yang tertua berikutnya adalah resep roti dari Mesopotamia.

Peralatan dan penanganan yang berbeda telah digunakan untuk bekerja dengan cipher. Salah satu yang terdahulu digunakan oleh bangsa Yunani kuno adalah sebuah tongkat, dimana tongkat tersebut digunakan bangsa Sparta untuk mentransposisi cipher. Pada abad pertengahan, penanganan lain telah ditemukan, contohnya adalah cipher grille, yang digunakan juga untuk steganografi. Seiring dengan pengembangan cipher polialphabetik, semakin banyak pula bentuk-bentuk alat dan penanganan cipher, misalnya adalah cipher disk buatan Alberti, skema Johannes Trithemius tabula recta, dan *multi* silinder buatan Thomas Jefferson. Banyak mesin pengenkripsi dan pendekripsi diciptakan dan dikembangkan pada abad ke 20 dan beberapa yang dipatenkan. Perkembangan komputer digital dan elektronika setelah perang dunia kedua memungkinkan untuk membuat chiper yang lebih kompleks lagi. Berbeda dengan chiper kuno yang mengenkripsi dengan tulisan dan bahasa, komputer memungkinkan enkripsi bermacam macam data dalam format biner. Cipher komputer dapat dikategorikan berdasarkan operasi bilangan biner yang dilakukannya. Cipher yang modern telah melewati ilmu kriptanalisis. Jenis cipher ini pada umumnya paling efisien dan sangat sulit untuk memecahkannya.

2.3 Tujuan Kriptografi

Tujuan utama dari kriptografi akan dijelaskan sebagai berikut:

1. Kerahasiaan (*confidentiality*)

Kerahasiaan bertujuan untuk melindungi suatu informasi dari semua pihak yang tidak berhak atas informasi tersebut. Terdapat beberapa cara yang dapat digunakan untuk menjaga kerahasiaan suatu informasi, mulai dari penjagaan secara fisik misalnya menyimpan data pada suatu tempat khusus sampai dengan penggunaan algoritme matematika untuk mengubah bentuk informasi menjadi tidak terbaca.

2. Integritas data (*data integrity*)

Integritas data bertujuan untuk mencegah terjadinya pengubahan informasi oleh pihak-pihak yang tidak berhak atas informasi tersebut.Manipulasi data yang dimaksud di sini meliputi penyisipan, penghapusan, maupun penggantian data.

3. Otentikasi (*authentication***)**

Otentikasi merupakan identifikasi yang dilakukan oleh masing–masing pihak yang saling berkomunikasi, maksudnya beberapa pihak yang berkomunikasi harus mengidentifikasi satu sama lainnya. Informasi yang didapat oleh suatu pihak dari pihak lain harus diidentifikasi untuk memastikan keaslian dari informasi yang diterima. Identifikasi terhadap suatu informasi dapat berupa tanggal pembuatan informasi, isi informasi, waktu kirim dan hal-hal lainnya yang berhubungan dengan informasi tersebut.

4. *Non-repudiation*

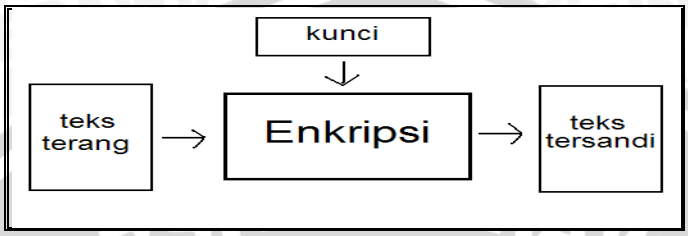
Nirpenyalahgunaan (non-repudiation), adalah layanan untuk mencegah entitas yang berkomunikasi melakukan penyangkalan, yaitu pengiriman pesan menyangkal melakukan pengiriman atau penerima pesan menyangkal telah menerima pesan.

2.4 Sistem Kriptografi

Salah satu tujuan dari dilakukannya proses penyandian adalah untuk membuat data yang dikirimkan tidak dapat dimengerti oleh pihak lain selain yang memiliki akses terhadap data tersebut. Proses penyandian terdiri atas dua tahapan, yaitu:

2.4.1 Enkripsi

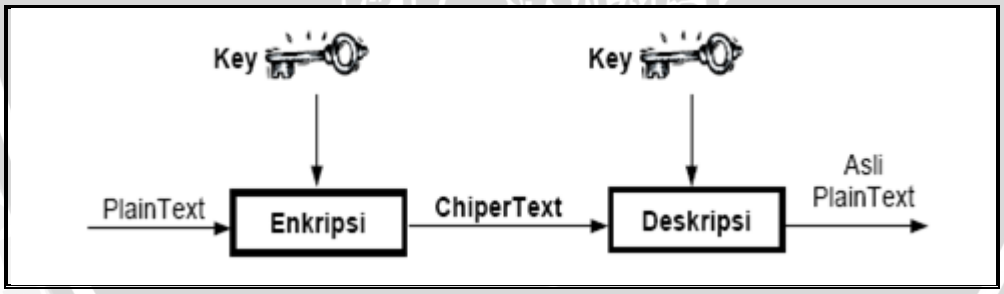
Enkripsi merupakan proses mengubah *plaintext* menjadi *ciphertext* yang tidak bisa dimengerti. Proses enkripsi biasa dilakukan sebelum pesan dikirimkan. Untuk meningkatkan keamanan enkripsi pesan, pada proses enkripsi ditambahkan kunci yang juga diperlukan untuk proses dekripsi, seperti pada Gambar 2.1.



Gambar 2.1 Proses Enkripsi Dengan Kunci
Sumber: id.wkipedia.org

2.4.2 Dekripsi

Dekripsi merupakan proses untuk mengubah cipherteks kembali menjadi plainteks agar pesan dapat dimengerti. Proses dekripsi biasanya dilakukan oleh penerima pesanagar pesan yang diterima dapat dimengerti. Untuk proses enkripsi yang menggunakan kunci maka proses dekripsi harus dilakukan dengan menggunakan kunci, seperti pada Gambar 2.2



Gambar 2.2 Proses Dekripsi Dengan Kunci
Sumber: rezqiwati.wordpress.com

Kunci yang digunakan pada proses dekripsi dapat berbeda dengan kunci yang digunakan pada proses enkripsi, disebut juga kriptografi kunci publik. Sebaliknya, jika kunci yang digunakan sama, disebut juga kriptografi kunci simetri. Berikut adalah istilah istilah yang berkaitan dengan proses penyandian pesan,antara lain:

1. Kunci

   Kunci yang yang dimaksud di sini adalah kunci yang dipakai untuk melakukan enkripsi dan dekripsi. Kunci terbagi menjadi dua bagian, yakni kunci pribadi (*private key*) dan kunci umum (*public key*).

2. *Chipertext*

   Merupakan suatu pesan yang sudah melalui proses enkripsi. Pesan yang ada pada *ciphertext* tidak bisa dibaca karena berisi karakter-karakter yang tidak memiliki makna (arti).

3. *Plaintext*

   Sering juga disebut *cleartext*; merupakan suatu pesan bermakna yang ditulis atau diketik dan *plaintext* itulah yang akan diproses menggunakan algoritme kriptografi agar menjadi *chipertext*.

4. Pesan

   Pesan bisa berupa data atau infomasi yang dikirim (melalui kurir, saluran komunikasi data, dan sebagainya) atau yang disimpan di dalam media perekaman (kertas, storage, dan sebagainya).

5. *Cryptanalysis*

   Bisa diartikan sebagai analisis sandi atau suatu ilmu untuk mendapatkan *plaintext* tanpa harus mengetahui kunci secara wajar. Jika suatu *ciphertext* berhasil menjadi *plaintext* tanpa menggunakan kunci yang sah, maka proses tersebut dinamakan *breaking code* yang dilakukan oleh para *cryptanalyst*. Analisis sandi juga mampu menemukan kelemahan dari suatu algoritme kriptografi dan akhirnya bisa menemukan kunci atau *plaintext* dari *chipertext* yang dienkripsi menggunakan algoritme tertentu.

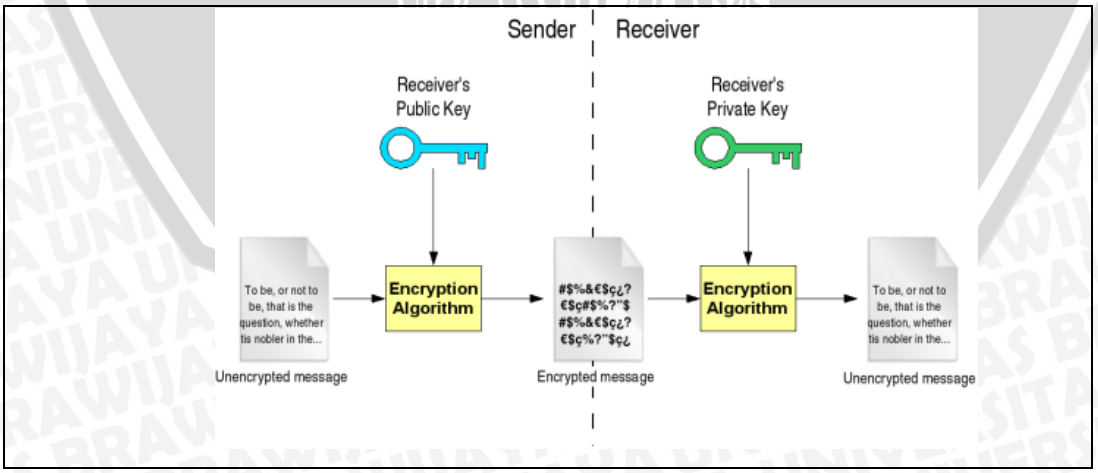
2.5 Kriptografi Berdasarkan Jenis Kunci

Berdasarkan pemakaian kunci, sistem kriptografi (c*ryptosystems*) dapat digolongkan atas atas 2 (dua) jenis sistem yakni sistem kriptografi kunci publik (*public key cryptography*) dan sistem kriptografi kunci rahasia (*secret key*

cryptography). Dalam sistem kriptografi kunci rahasia yang dikenal juga dengan *symmetric cryptosystems*, pihak pengirim dan penerima bersama-sama menyepakati sebuah kunci rahasia yang akan digunakan dalam proses enkripsi dan dekripsi tanpa diketahui oleh pihak lain. Sedangkan dalam sistem kriptografi kunci publik atau dikenal dengan *asymmetric cryptosystem*, pihak pengirim maupun pihak penerima mendapatkan sepasang kunci yakni kunci publik (*public key*) dan kunci rahasia (*private key*) di mana kunci public dipublikasikan dan kunci rahasia tetap dirahasiakan. Enkripsi dilakukan dengan menggunakan kunci publik sedangkan dekripsi dilakukan dengan menggunakan kunci rahasia.

2.5.1 Kriptografi Kunci Asimetris

Algoritme asimetris adalah suatu algoritme kriptografi dimana kunci untuk enkripsi yang digunakan berbeda dengan kunci dekripsi. Kunci enkripsi dinamakan sebagai kunci public yaitu kunci yang bebas diketahui oleh siapapun, sedangkan kunci dekripsi dinamakan kunci kunci privat yaitu kunci yang hanya boleh diketahui oleh penerima pesan. Contoh dari algoritme kriptografi asimetris adalah RSA, Elgamal, dan lain-lain. Kelebihan dari kriptografi kunci asimetris yaitu masalah keamanan pada distribusi kunci dapat lebih baik dan manajemen kunci yang lebih baik karena jumlah kunci yang sedikit. Sementara itu, kelemahan kunci asimetirs antara lain kecepatan yang lebih rendah bila dibandingkan dengan algoritme simetris untuk tingkat keamanan sama, kunci yang digunakan lebih panjang dibandingkan dengan algoritme simetris.

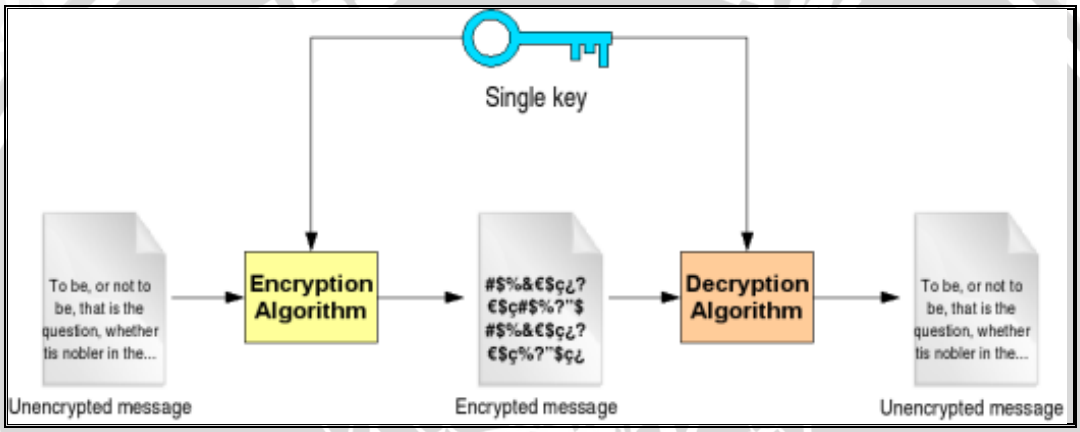


Gambar 2.3 Skema Kriptografi Kunci Asimetris
Sumber: gdp.globus.org

Pada Gambar 2.3 merupakan diagram proses enkripsi dan dekripsi pada algoritme asimetris. Pada proses tersebut *plaintext* dienkripsi menggunakan kunci enkripsi *public key* sehingga menghasilkan *ciphertext*. *Ciphertext* didekripsikan kembali menggunakan kunci *private key* dekripsi, artinya kunci yang dipakai pada saat pendekripsian berbeda dengan kunci ketika pengenkripsian

2.5.2 Kriptografi Kunci Simetris

Kriptografi kunci simetri menggunakan kunci yang sama untuk proses enkripsi dan dekripsi. Kunci tersebut harus ditentukan sebelumnya oleh pihak penerima dan pengirim pesan, seperti pada Gambar 2.4.



Gambar 2.4 Skema Kriptografi Kunci Simetris
Sumber: gdp.globus.org

Kelemahan dari kriptografi kunci simetri adalah kesulitan dalam pendistribusian kunci karena pengirim dan penerima pesan harus mengetahui kunci yang sama maka kunci harus dikirimkan atau diberitahukan pada pihak lainnya. Masalah lainnya adalah pada banyaknya kunci yang harus digunakan, untuk setiap pasangan pengirim dan penerima pesan harus ada paling tidak satu buah kunci yang dapat digunakan untuk melakukan enkripsi antara mereka.

2.6 Tipe dan Mode Algoritme Kunci Simetri

Algoritme kunci simetri memiliki dua tipe yang umum digunakan berdasarkan mode bitnya, yaitu:

1. Cipher blok, yang beroperasi pada blok-blok plainteks. Ukuran blok dapat bermacam- macam tergantung pada algoritme enkripsi yang digunakan, biasanya berkisar antara 64 atau 128 *bit*. Hasil enkripsi dari suatu plainteks yang sama, akan menghasilkan cipherteks yang sama.
2. Cipher aliran, yang beroperasi pada aliran plainteks satu *bit* atau *byte* setiap waktu. Hasil enkripsi dari suatu plainteks yang sama belum tentu menghasilkan cipherteks yang sama.

2.7 Teknik Subtitusi

Teknik subtitusi ini merupakan penggantian setiap karakter dari *plaintext* dengan karakter lainnya, ada empat istilah dari subtitusi *cipher* diantaranya adalah : *monoalphabet, polyalphabet, monograph,* dan *polygraph.* Contoh teknik subtitusi adalah Caesar *Cipher*, *Playfair Cipher*, *Shift Cipher*, Hill *Cipher*, dan Vigenere *Cipher*. Subtitusi Caesar *Cipher* merupakan pencetus dalam dunia kriptografi, penyandian ini dilakukan pada zaman Pemerintahan **Julius Caesar** dengan menggeser atau mengganti posisi huruf *alphabet.* Misal pergeseran yang dilakukan sebanyak tiga kali, berarti kunci dekripsinya adalah (*n – 3*). Sebenarnya pergeseran yang dilakukan tergantung keinginan dari kesepakatan pihak pengirim dan penerima, ini yang dinamakan kunci.. Misal kunci yang digunakan (*(n x 2) - 1*) atau yang lainnya. Contoh pesan yang akan disandikan dari algoritme Caesar *Cipher* dengan kunci *(n + 3)* sebagai berikut :

Plaintext = S E L A M A T

Ciphertext = V H O D P D W

Dengan subtitusi atau mengeser tiga kali sehingga huruf S = V, E = H, L = O, ……, T = W. Caesar *Cipher* ini dapat dipecahkan dengan cara *Brute Force Attack* suatu bentuk dari sebuah serangan dengan mencoba kemungkinan-kemungkinan pola untuk menemukan kunci rahasia sampai kunci tersebut ditemukan. Banyak kemungkinan kunci yang dapat digunakan oleh Caesar *Cipher* sehingga cukup merespon para kriptoanalis, walaupun sederhana akan tetapi butuh cukup waktu untuk memecahkannya karena penggunaan Enkripsi Klasik tidak semudah sekarang dengan bantuan komputer.

2.8 Teknik Tranposisi

Teknik ini menggunakan permutasi karakter, dengan menggunakan teknik ini pesan yang asli tidak dapat dibaca kecuali pihak yang memiliki kunci untuk mengembalikan pesan tersebut kebentuk semula atau mendekripsikannya. Sebagai contoh :

Ada enam kunci yang digunakan untuk melakukan permutasi *cipher* yaitu:

Posisi *Plaintext*	**1**	**2**	**3**	**4**	**5**	**6**
Posisi *Ciphertext*	**3**	**5**	**1**	**6**	**4**	**2**

Untuk mendekripsikan digunakan juga 6 kunci *invers cipher* yaitu :

Posisi *Ciphertext*	**1**	**2**	**3**	**4**	**5**	**6**
Posisi *Plaintext*	**3**	**6**	**1**	**5**	**2**	**4**

Sehingga sebuah pesan

Plaintext = T E R I M A K A S I H

Terlebih dahulu kalimat tersebut dibagi menjadi 6 *block* dan apabila terjadi kekurangan pada *block* bisa ditambahkan dengan huruf yang disepakati, misal "X".

Posisi = 1 2 3 4 5 6 1 2 3 4 5 6

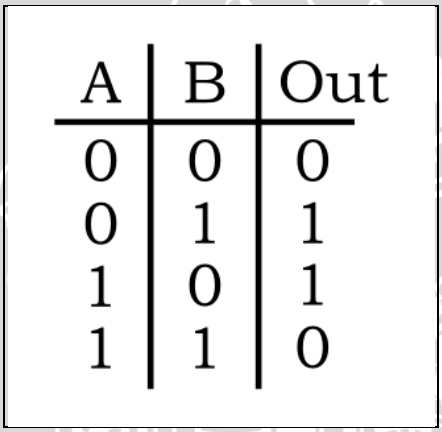
Plaintext = T E R I M A K A S I H X
Ciphertext = R A T M E I S X K H A I

Untuk mendekripsikan *ciphertext*nya, maka harus melakukan hal yang sama seperti *cipher*nya dengan menggunakan kunci *invers cipher* dari permutasi tersebut. Banyak teknik lain permutasi seperti *zig-zag***,** *segitiga*, *spiral*, dan

diagonal. Denganbeberapa macam pola Teknik Tranposisi (Permutasi) maka dapat dilakukan untuk menyandikan atau menyembunyikan pesan secara aman dari pihak yang tidak berwenang. Dari kombinasi teknikteknik inilah yang menjadi dasar dari pembentukan algoritme kriptografi yang dikenal dengan Kriptografi Modern

2.9 Operasi XOR

Jika meng-XOR-kan dua buah bit yang sama nilainya, maka operasi XOR akan menghasilkan nilai bit "0" (nol) dan jika meng-XOR-kan dua buah bit yang masing masing nilai bitnya berbeda, maka akan menghasilkan nilai bit "1" (satu). Aturan yang berlaku untuk operasi XOR dapat dilihat pada gambar di bawah ini



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Gambar 2.5 Tabel Kebenaran XOR
Sumber: eeweb.com

Jika nilai A di-XOR-kan dengan nilai B sebanyak 2 (dua) kali, maka akan didapatkan nilai A kembali. Karena sifat istimewa yang dimiliki operasi XOR tersebut sehingga operasi XOR cenderung dipakai dalam proses enkripsi dan dekripsi yang memiliki kunci yang sama.

2.10 Bilangan Acak

Bilangan acak adalah deretan nilai yang acak dan tidak dapat diprediksi semuanya. Untuk menghasilkan bilangan acak merupakan hal yang sulit,

kebanyakan pembangkit bilangan acak (*random number generator* = RNG) mempunyai beberapa bagian yang dapat diprediksi dan berhubungan. Kebanyakan RNG mengulang *string* yang sama setelah melakukan *n* putaran.

2.11 Pembangkit Bilangan Acak

Random number generator (RNG) adalah suatu piranti komputasional yang dirancang menghasilkan suatu urutan nilai yang tidak dapat ditebak polanya dengan mudah, sehingga urutan nilai tersebut dapat dianggap sebagai suatu keadaan acak (*random*). RNG ini tidak dapat diterapkan dalam prakteknya. Bilangan acak yang dihasilkan oleh komputer sekalipun tidak benar-benar acak dan kebanyakan bilangan acak yang diterapkan dalam kriptografi juga tidak benar-benar acak, tetapi hanya berupa acak semu. Ini berarti bahwa bilangan acak yang dihasilkan itu dapat ditebak susunan atau urutan nilainya. Dalam kriptografi, bilangan acak sering dibangkitkan dengan menggunakan pembangkit bilangan acak semu (*pseudo random number generator*).

2.12 Pembangkit Bilangan Acak Semu

Suatu *pseudo random number generator* (PRNG) merupakan suatu algoritme yang menghasilkan suatu urutan nilai dimana elemen-elemennya bergantung pada setiap nilai yang dihasilkan. *Output* dari PRNG tidak betul-betul acak, tetapi hanya mirip dengan properti dari nilai acak.Kebanyakan algoritme dari *pseudo random number generator* ditujukan untuk menghasilkan suatu sampel yang secara seragam terdistribusi. PRNG ini sering digunakan dalam kriptografi pada proses pembentukan kunci dari metoda kriptografi. Tingkat kerumitan dari PRNG ini menentukan tingkat keamanan dari metoda kriptografi.Semakin rumit (kompleks) PRNG yang digunakan maka semakin tinggi tingkat keamanan dari metoda kriptografi.

2.13 *Mersenne Twister*

Algoitma *mersenne twister* merupakan salah satu PRNG (*Pseudo Random Number Generator*) yang di kembangaakan Pada tahun 1997 , penemuan

algoritme *Mersenne twister* oleh Matsumoto dan Takuji Nishimura, menghilangkan masalah yang dihadapai oleh pembangkit-pembangkit acak semu sebelumnya. *Mersenne Twister* memiliki periode yang sangat besar yaitu 2^{19937}-1 iterasi yang tidak akan selesai dihitung secara komputasi selamanya.. Keunggulan lainnya adalah memiliki keseragaman keluaran hingga dimensi 623 (untuk yang 32 bit), dan waktu eksekusinya lebih cepat daripada pembangkit acak yang baik lainnya. *Mersenne Twister* sekarang ini telah menjadi pilihan utama sebagai pembangkit bilangan acak semu untuk simulasi dan pemodelan Meskipun *Mersenne Twister* sesuai untuk dipakai dalam banya kaplikasi, MT yang murni tidak cocok dipakai dalam kriptografi. Masalahnya adalah MT serupa dengan generalized feedback shift register sehingga keluaran yang dihasilkan dapat diprediksi dengan mudah.

Algoritme umum ditandai dengan jumlah sebagai berikut:

w: panjang kata dalam bit

n: tingkat perulangan

m: kata tengah , sebuah offset yang digunakan dalam relasi rekurensi mendefinisikan seri $\boldsymbol{x}$, $1 \leq m < n$

r: titik pemisahan satu kata , atau jumlah bit dari bitmask rendah $0 \leq r \leq w - 1$

a : koefisien rasional yang normal matriks bentuk twister

b, *c*: TGFSR(R) percampuran bitmasks

s, *t*: TGFSR(R) pergeseran bit percampuran

u , d , l : tambahan *Mersenne Twister* percampuran sedikit pergeseran

dengan pembatasan yang $2^{nw-r} - 1$ adalah ke utamaan Mersenne . Pilihan ini menyederhanakan tes primitif dan k - distribusi pengujian yang dibutuhkan dalam pencarian parameter .

Seri x didefinisikan sebagai rangkaian w - bit jumlah dengan relasi rekurensi :

$$x_{k+n} := x_{k+m} \oplus ({x_k}^u \mid {x_{k+1}}^l)A \qquad k = 0, 1, \ldots$$

Dimana | menunjukkan bitwise or $\oplus$ *bitwise exclusive or* (XOR), x_k^u berarti naik $w - r$bit. x_k dan x_{k+1}^l berarti rendah r bit x_{k+1}persilangan tranformasi A di definisikan dalam bentuk normal

$$A = \begin{pmatrix} 0 & I_{w-1} \\ a_{w-1} & (a_{w-2}, \ldots, a_0) \end{pmatrix}$$

Dengan I_{n-1} sebagai (n - 1) × (n - 1) matriks identitas . Bentuk normal rasional memiliki manfaat yang perkalian dengan A dapat efisien dinyatakan sebagai :

$$\boldsymbol{x}A = \begin{cases} \boldsymbol{x} \gg 1 & x_0 = 0 \\ (\boldsymbol{x} \gg 1) \oplus \boldsymbol{a} & x_0 = 1 \end{cases}$$

Dimana X_0 lebih rendah dari x

Seperti TGFSR (R) , *Mersenne Twister* mengalir dengan teranformasi perulangan untuk mengkompensasi berkurangnya dimensi dari *equidistribution* . Perhatikan bahwa ini setara dengan menggunakan matriks A ' di mana A ' = T - 1AT untuk T matriks yang dapat dibalik , dan karena itu analisis polinomial karakteristik yang disebutkan di bawah masih memegang .

Seperti A, kita memilih transformasi perulangan menjadi mudah dihitung , dan tidak benar-benar membangun T itu sendiri . Perulangan didefinisikan dalam kasus *Mersenne Twister* sebagai

$$\begin{aligned} \boldsymbol{y} &:= \boldsymbol{x} \oplus ((\boldsymbol{x} >> u) \,\&\, \boldsymbol{d}) \\ \boldsymbol{y} &:= \boldsymbol{y} \oplus ((\boldsymbol{y} << s) \,\&\, \boldsymbol{b}) \\ \boldsymbol{y} &:= \boldsymbol{y} \oplus ((\boldsymbol{y} << t) \,\&\, \boldsymbol{c}) \\ \boldsymbol{z} &:= \boldsymbol{y} \oplus (\boldsymbol{y} >> l) \end{aligned}$$

di mana x adalah nilai berikutnya dari seri , ya sementara nilai menengah, z nilai kembali dari algoritme , dengan << , >> sebagai bitwise kiri dan pergeseran kanan , dan & sebagai bitwise dan transformasi pertama dan terakhir ditambahkan dalam rangka meningkatkan lebih rendah sedikit equidistribution . Dari milik TGFSR , $s + t \geq \lfloor w/2 \rfloor - 1$ diperlukan untuk mencapai batas atas *equidistribution* untuk bit atas.

2.13.1 Desain *Mersenne Twister*

Mersenne Twister menghasilkan suatu urutan bilangan semu acak yang selanjutnya disebut sebagai word dengan menggunakan rekursi lanjar Gaussian Field GF(2) Dengan wi (i=0,1,2,…) adalah bilangan bulat 32 bit, yang masing masing dianggap sebagai sebuiah vector baris berdimensi 32 terhadap dua *field* elemen GF(2). Operator biner ⊕ melambangkan operasi bit XOR yaitu penambahan sebagai vector. Notasi heksadesimal 0x80000000 merupakan vector yang komponennya adalah nol semuanya kecuali bit satu yang terkiri. Jadi ((wi&0x8000000) |(w1+i&0x7fffffff)) adalah vector baris yang diperoleh dengan menyambungkan MSB dari wi dan dengan semua bit dari w1+I kecuali MSBnya. Vektor ini kemudian dikalikan dengan MAtriks A berukuran 32x32. MAtriks A ini adalah berbentuk

$$\begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ a_{31} & a_{30} & \cdots & \cdots & a_0 \end{pmatrix}$$

Sehingga perkalian ini dihitung dengan

xA = shiftkanan(x) (jika LSB dari x adalah 0)

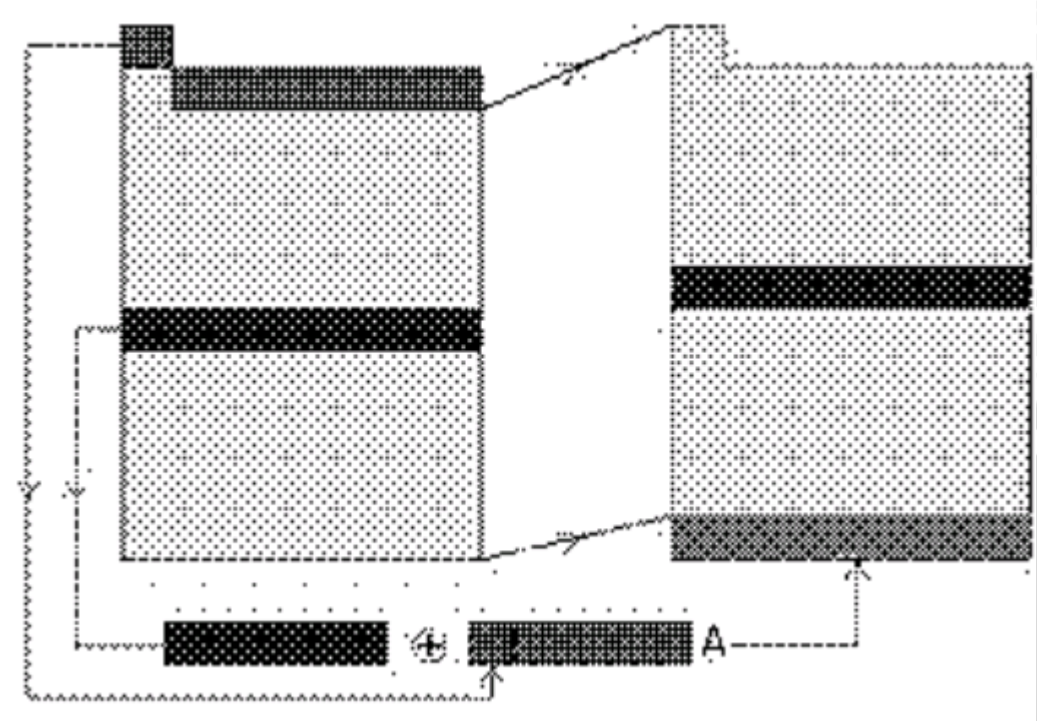
atau

xA = shiftkiri(x) (jika LSB dari x adalah 1)

Konstanta ini dipilih sehingga periode dari bilangan yang adihasilkan adalah 2^{19937}-1. Jumlah dari elemen yang bukan nol pada karakterisitik polinom dari fungsi transisi adalah 153.

Gambar berikut menggambarkan state transisi dari MT. State nya terdiri atas 623 word + 1 bit. State selanjutnya diperoleh dengan menggeser satu word ke atas, dan menyelipkan sebuah word baru yang dihitung secara lanjar dari bagian yang dibuang dan elemen bagian tengah. Pada implementasi pada perangkat

lunak, digunakan teknik round-robin (yaitu memanfaatkan pointer) untuk masalah efifiensi pembangkitan bilangan.



Gambar 2.6 Gambar Mersenna Twister

Kelebihan dari konfigurasi ini dibandingkan dengan LFSR biasa dengan okefisien misalnya pada GF(2^{32}) adalah:

1. Tidak diperlukan operasi yang mahal seperti polynomial perkalian modulo
2. Terdapat algoritme yang secara efisien mengecek periode maksimal dengan teori Galois
3. Kecepatan dalam menghasilkan bilangan tidak bergantung pada a, yang membuat pencarian parameter menjadi lebuiih mudah. Hal ini berbeda dengan LFSR dimana kecepatan menghasilkan bilangannya bergantung pada pemilihan koefien tertentu.

Untuk menginisialisasi *Mersenne Twister*, diperlukan bilangan $w_0,w_1,w_2,\ldots,w_{623}$ sebagai state awal , dimana semua 31 bit dari w_0 kecuali MSB-nya diacuhkan dalam menghasilkan word selanjutnya. Jadi, ruang state memiliki

624x32-31 = 19937 bit. Urutan keluaran dari *Mersenne Twister* adalah w^{624},w^{625},… , yaitu MT melewatkan isi dari state awal.

Inisialisasi pada (MT) adalah sangat penting karena sifat kelanjaran dan kejarangan pada rekursinya. Jika state awal memiliki terlalu banyak bit nol, maka urutan bilangan keluaran akan memiliki kecenderungan yang sama untuk keluaran yang melebihi 1000. Pertama, w_0,w_1,…,w_{623} diset nilainya dengan nilai yang rumit melalui rekursi

$$\begin{aligned} w_0 &\leftarrow 19650218, \\ w_i &\leftarrow ((w_{i-1} \oplus (w_{i-1} >> 30)) + i) \times 1812433253 \end{aligned}$$

(1= i = 623), dengan w i dianggap sebagai variable bilangan bulat integer 32 bit unsigned, dan setiap operasi aritmatika adalah dalam modulo 2_{32}. Notasi >> 30 berarti pergeseran bit ke kanan sebanyak 30 bit (Konstanta 19650218 adalah hari ulang tahun salah satu pencipta *Mersenne Twister*. Konstanta 1812433252 adalah pengali untuk pembangkit kongruen lanjar, dipilih acak tanpa alasan tertentu).

Rekursi ini dipilih sehingga memiliki property difusi bit yang bagus. Perkalian dengan konstanta memiliki peroperti pencampuran bit yang bagus kecuali bahwa difusi untuk informasi bit dilakukan dari kanan ke kiri. Dua bit paling sifnifikan (yang mengumpulkan informasi dari semua bit setelah perkalian) dikirimkan ke kedua LSB bit dari w_i dengan meng-XOR-kan untuk mengomplemenkan perkalian. Lambang operator assignment yang dipakai menandakan bijeksi.

Penambahan dengan I bermanfaat untuk menghindari fenomena berikut. Misalkan I tidak ditambahkan pada rekursi pada inisialisasi, misalkan pula bahwa nilai awal w0 dipilih sembarang pula dan w0,…,w623 adalah urutan keluaran. Apabila pada inisialisasi yang lain dipakai nilai awal w0’, yang akan menghasilkan keluaran w0’,…,w623’. Yang menjadi masalah adalah mungkin terjadi bahwaq w0’=w 1 secara kebetulan (atau w0’=w2 atau semacamnya) maka wi’=wi+1 untuk i=0,1,…,622. Kesamaan tersebut pada nilai awal menghasilkan

keluaran yang saling berhuungan untuk 100 keluaran atau lebih. Penambahan dengan I menghindari fenomena ini.

Kelebihan dari *Mersenne Twister*

Versi yang umum digunakan untuk Algoritme *Mersenne Twister* MT 19937 yang menghasilakan urutan bilangan bulat 32-bit memiliki sifatyang diniginkan sebagai berikut :

1. Memiliki panjang periode $12^{19937} - 1$. Sementara panjang periode belum menjamin ke acakan *generator* bilangan acaka semu. periode pendek (seperti 2^{32} umum di banyak paket perangkat lunak yang lebih tua) dapat menjadi masalah
2. Distribusi K ke 32-bit keakuratnanya setiap $1 \leq k \leq 623$
3. Melewati berbagai test untuk keacakan static termasuk diehard test

Kelemahan *Mersenne Twister*

1. Pengkomputasinya membutukan ruang yang besar pada CPU (Periode diatas 2^{512} sudah cukup untuk aplikasi apaun). Pada tahun 2011 Saito & Matsumoto mengusulkan versi *Mersenne Twister* untuk mengatasi masalah ini . Versi kecil , TinyMT , hanya menggunakan 127 bit ruang static.
2. Mulai saat ini, *Mersenne Twister* cukup lambat , kecuali digunakan pelaksanaan SFMT.
3. Dapat melewati sebagaian besar test tapi tidak semua test pada test ke acakan TESTU01 yang ketat.
4. Ini membutuhkan waktu yang lama untuk menghasikan keluaran dari test keacakan. Jika keadaan awal tidak acak terutama jika kadaan awal memiliki angka 0 yang banyak. Ituadalah kosekuensi dari *generator* yang singkat. Dimulai dari adaan yang sama biasanya akan menampikan hampir menpilkan urutan yang sama pada banyak iterasi, sebelum akhirnya divergen, pada 2002 di perbarui untuk meningkatkan iterasi sehingga mencapai ke adaan yang tidak mungkin.

BAB III
METODOLOGI PENELITIAN

Dalam penyusunan skripsi ini, dirancang suatu perancangan Aplikasi system pengacakan bilangan semu menggunakan metode *Mersenne Twister*. Adapun langkah – langkah yang akan dikerjakan meliputi sebagai berikut :

1. Studi literature
2. Abstraksi system
3. Pengujian dan analisis system
4. Kesimpulan dan saran

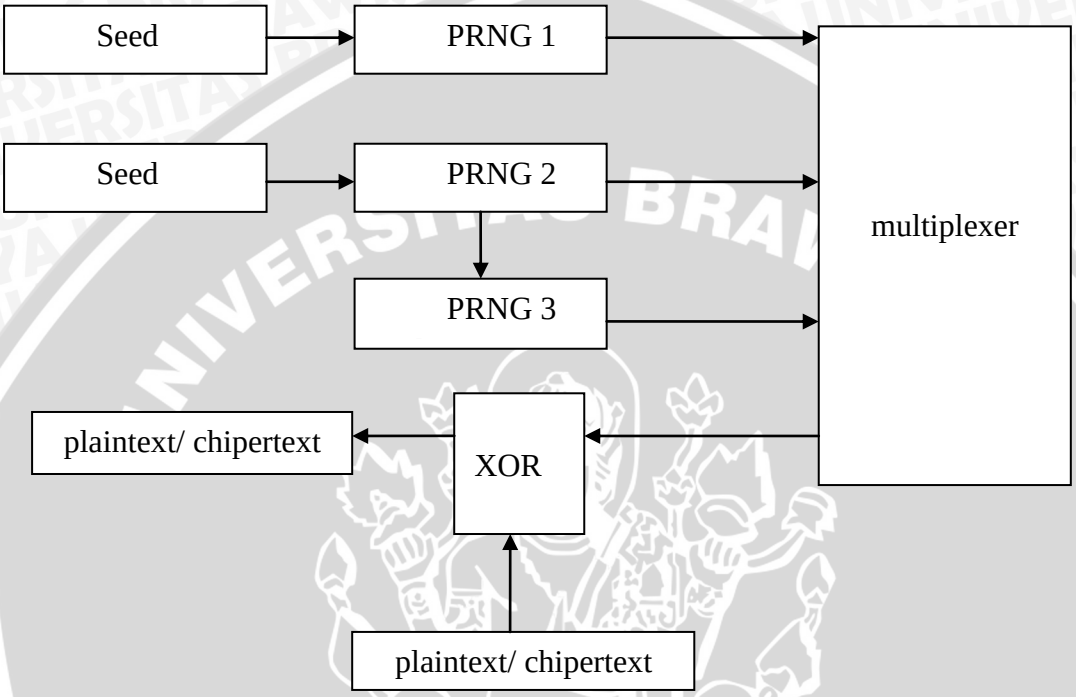
3.1 Abstraksi Sistem

Kriptografi mempunyai tujuan menjaga keamanan suatu data sehingga dapat terjamin bahwa informasi dari data terjaga keasliannya dan tertuju pada pihak yang berhak mendapatkannya. Seiring dengan perkembangan zaman, penggunaan PRNG sebagai *generator* bit semi acak pada metode *Mersenne Twister* tidak aman terhadap serangan kriptanalisis oleh karena itu perlu dilakukan pengenbangan untuk meningkatkan keacakan bilangan acak semu berbasis PRNG agar lebih tahan terhadap serangan kriptanalisis.

Perangkat lunak yang dibangun bertujuan melakukan enkripsi dekripsi pada data dengan melakukan pengembangan pada algoritme pembangkit bilangan semu acak berbasis PRNG sehingga meningkatkan ketahanan terhadap serangan kriptanalisis.

Data dan kunci enkripsi diberikan oleh pengguna di mana data berfungsi sebagai *bilangan* dan kunci enkripsi dipakai sebagai *initial state* dari bilangan acak semu yang akan dibuat. Setelah *generator* PRNG menghasilkan bit semi acak yang panjangnya sama dengan panjang *bilangan*. Kemudian data *bilangan* di kombinasikan dengan operasi XOR dengan bit semu acak agar menghasilkan *kode* yang tidak terbaca lagi informasi yang terkandung di dalamnya. Untuk mengembalikan *kode* menjadi *bilangan* seperti semula, pengguna cukup

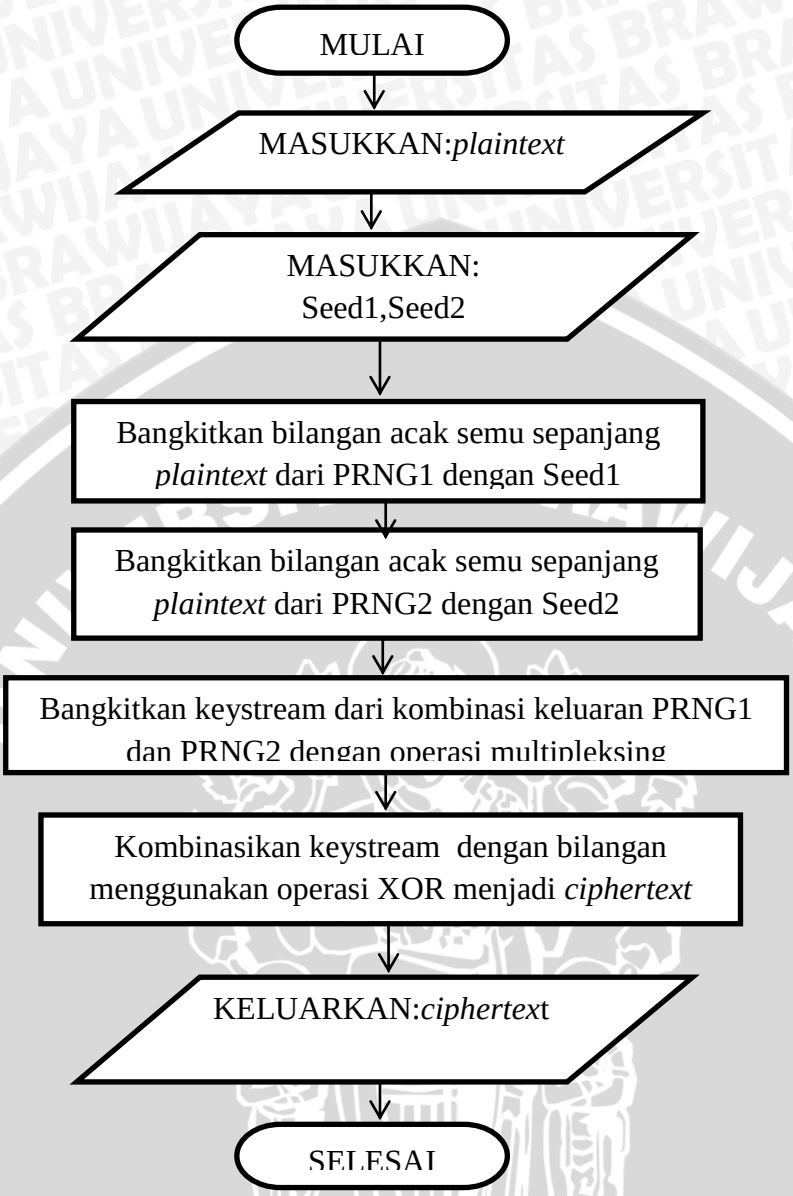
memasukkan *kode* dan kunci yang sama dengan kunci enkripsi. Proses kombinasi *kode* dengan kunci yang sama menggunakan operasi XOR akan mengembalikan *kode* menjadi *bilangan* yang sama.



Gambar 3.1 Diagram Blok sistem

3.2 Diagram Alir

Untuk membantu memahami abstraksi sistem, Berikut ini merupakan diagram alir dari proses enkripsi data. Diagram pertama memperlihatkan proses enkripsi dimulai dari pengguna mulai memasukkan data dan kunci sampai dengan data menjadi *kode*.



Gambar 3.2 Proses Enkripsi Data

Gambar 3.3 Proses Dekripsi Data

3.3 Pengujian dan Analisis System

Pada tahap ini pengujian menggunakan 10 masukan yang akan di uji hal ini dilakukan untuk memastikan bahwa system yang telah di rancang memiliki tingkat kesalahan atau error yang sangat kecil dan dengan akurasi yang tinggi. Agar dapat diketahui apakah system yang telah dirancang dapat bekerja dengan baik sesuai dengan perancangan yang telah dibuat sebelumnya maka akan dilakukan beberapa pengujian yang berangkaian atau berurutan. Tahap rangkaian penguajian adalah sebagai berikut :

Pengujian *Mersenne Twister* dengan menggunakan *single* PRNG dan *multi* PRNG

3.4 Kesimpulan dan Saran

Pada bagian ini akan dilakukan penarikan kesimpulan yang di hasilkan dari hasil pengujian dan analisis terhadap aplikasi PRNG menggunakan *Mersenne Twister*. Tahapan selanjutnya adalah membuat saran yg berguna sebagai masukan pada penelitian untuk menyempurnakan kekurangan yang ada sehingga di dapat perbaikan system yang lebih baik pada penelitian yang berikutnya.

BAB IV
PERANCANGAN DAN IMPLEMENTASI

4.1 Pembahasan

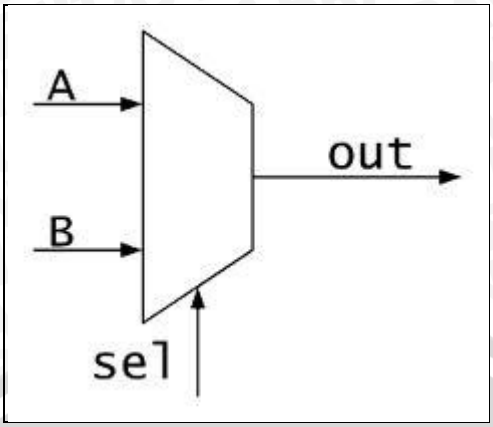
Enkripsi data atau juga dikenal sebagai penyandian data mempunyai tujuan salah satunya menjaga kerahasiaan data sehingga data tidak dapat disalahgunakan oleh pihak-pihak yang tidak bertanggung jawab.

Dalam masalah transfer data, enkripsi menjadi perihal yang sangat penting karena dalam proses pengiriman data ada peluang data dicuri atau disalahgunakan oleh pihak yang tidak bertanggung jawab sangatlah besar. Mengingat masalah kecepatan waktu dalam proses pengiriman data adalah sesuatu yang sangat penting untuk diperhatikan, diperlukan suatu algoritme enkripsi yang tepat untuk dapat mengeksekusi data secara cepat sehingga proses enkripsi nantinya tidak akan membuat proses pengiriman data itu sendiri menjadi tersendat.

Algoritme enkripsi dan dekripsi data *stream cipher* adalah algoritme kriptografi yang mampu melakukan proses penyandian data dengan sangat cepat dan tidak memakan memori yang relatif banyak sehingga prosesnya pun tidak akan memberatkan komputer.

Semakin berkembangnya kemajuan teknologi, kriptanalisis semakin berkembangan sehingga penggunaan sebuah *generator* saja untuk membangkitkan bilangan acak semu dalam algoritme *stream cipher* menjadi tidak aman lagi. *Multi* register adalah salah satu cara untuk mengatasi kriptanalisis sebuah *generator* bilangan acak semu. Dengan menggunakan *multi* register, periode dari sebuah bilangan acak semu akan bertambah panjang dan pola dari bilangan acak semu itu sendiri akan lebih sulit di analisis. Dalam skripsi ini metode *multi* register yang dipakai adalah menggunakan metode *multipleksing* data.

Multipleksing adalah proses memilih jalur keluaran dari beberapa saluran untuk dikeluarkan dalam sebuah saluran saja. *Multipleksing* memiliki beberapa saluran masukan dimana untuk menentukan saluran mana yang nantinya akan dikeluarkan tergantung dari nilai fungsi *select* dari *multipleksing* itu sendiri.



Gambar 4.1 Proses *Multipleksing* Secara Umum
Sumber: id.wikipedia.org

Dalam perancangan kali ini akan dipakai tiga buah *generator* berbasis PRNG dimana *generator* pertama dan *generator* kedua fungsinya sebagai masukan *multiplekser*,dan *generator* nomor tiga fungsinya sebagai fungsi *select* dari *multiplekser*. Bit yang dipakai untuk mengenkripsi *plaintext* berasal dari dari bit *generator* PRNG nomor satu atau *generator* PRNG nomor dua tergantung dari bit keluaran *generator* PRNG nomor 3. Apabila bit dari *generator* nomor 3 bernilai "1", maka bit yang akan diambil adalah bit keluaran dari PRNG nomor 2 dan sebaliknya apabila bit dari *generator* nomor 3 bernilai "0", maka yang akan diambil adalah bit dari *generator* nomor 1.

4.2 Perancangan Secara Umum

Perancangan aplikasi secara umum merupakan langkah awal sebagai acuan untuk pembuatan aplikasi lebih lanjut. Perancangan ini didahului dengan menentukan langkah – langkah paling dasar pada aplikasi *Pseudo Random Number Generator*

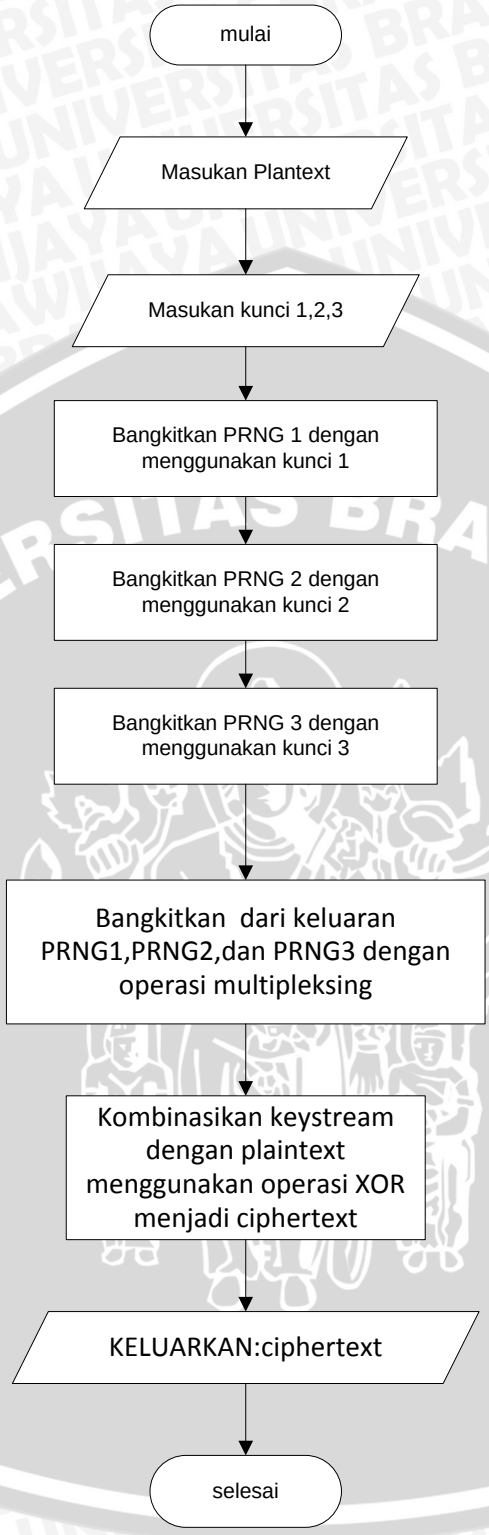
4.3 Perancangan Program

Algoritme enkripsi dekripsi stream cipher sebenarnya merupakan algoritme yang dapat diterapkan pada semua jenis baik *file* teks, *file* gambar, *file* audio, maupun *file* video karena prinsip dasarnya algoritme ini bekerja

berdasarkan bit atau byte. Pada program ini obyek yang akan dienkripsi hanya akan dibatasi pada *text* saja. Bilangan acak semu yang dihasilkan berasal dari 3 buah *generator* bilangan acak semu dimana jumlah seed masing masing *generator* berbeda beda antara satu dengan yang lainnya. Keluaran dari 3 buah *generator* ini nantinya akan disatukan menggunakan proses *multipleksing.*

PRNG pertama seed dimulai dari 0 sebanyak 32bit
PRNG kedua seed dimulai dari 1 sebanyak 32bit
PRNG ketiga seed dimulai dari 0 sebanyak 1bit

Berikut adalah diagram alir proses enkripsi yang menggambarkan proses enkripsi dari tahap awal hingga *plaintext* berubah menjadi *ciphertext.*

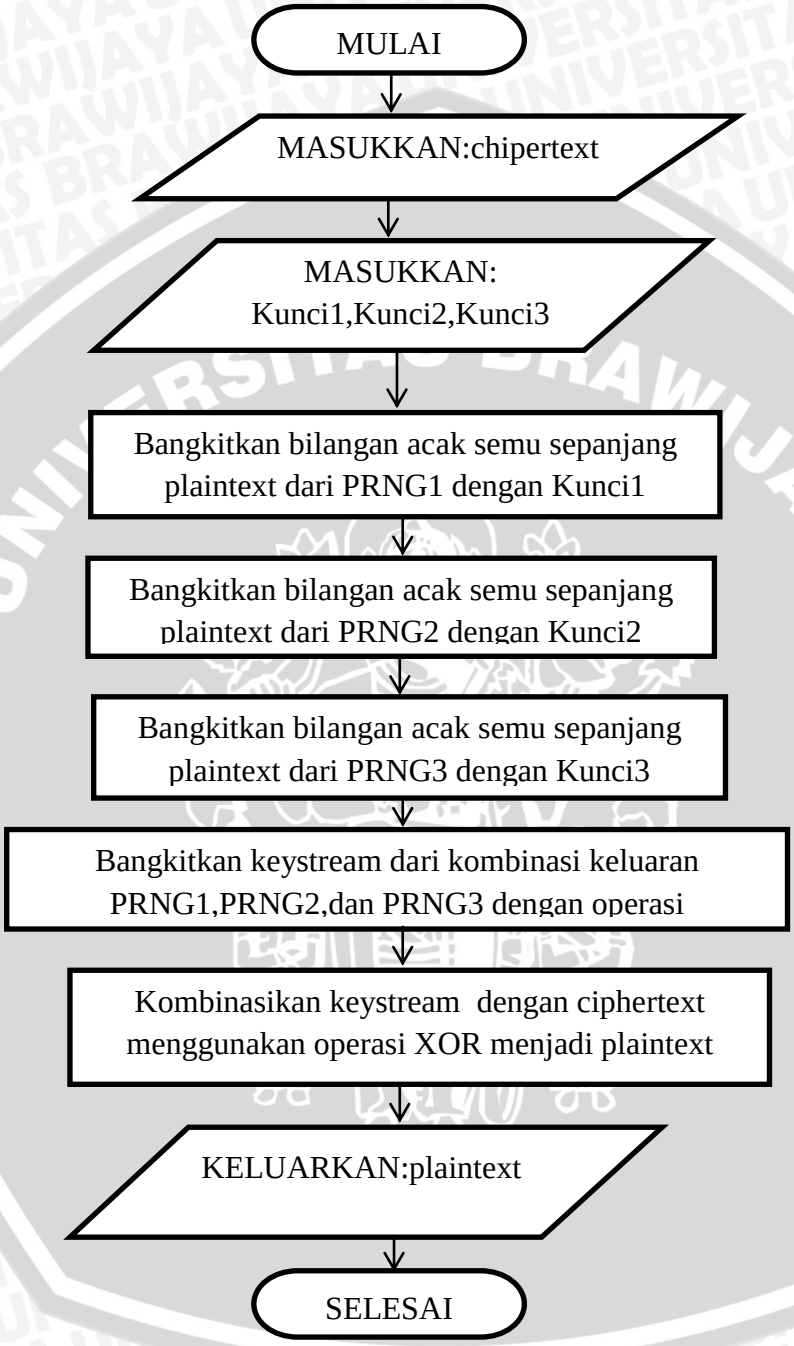


Gambar 4.2 Proses Enkripsi

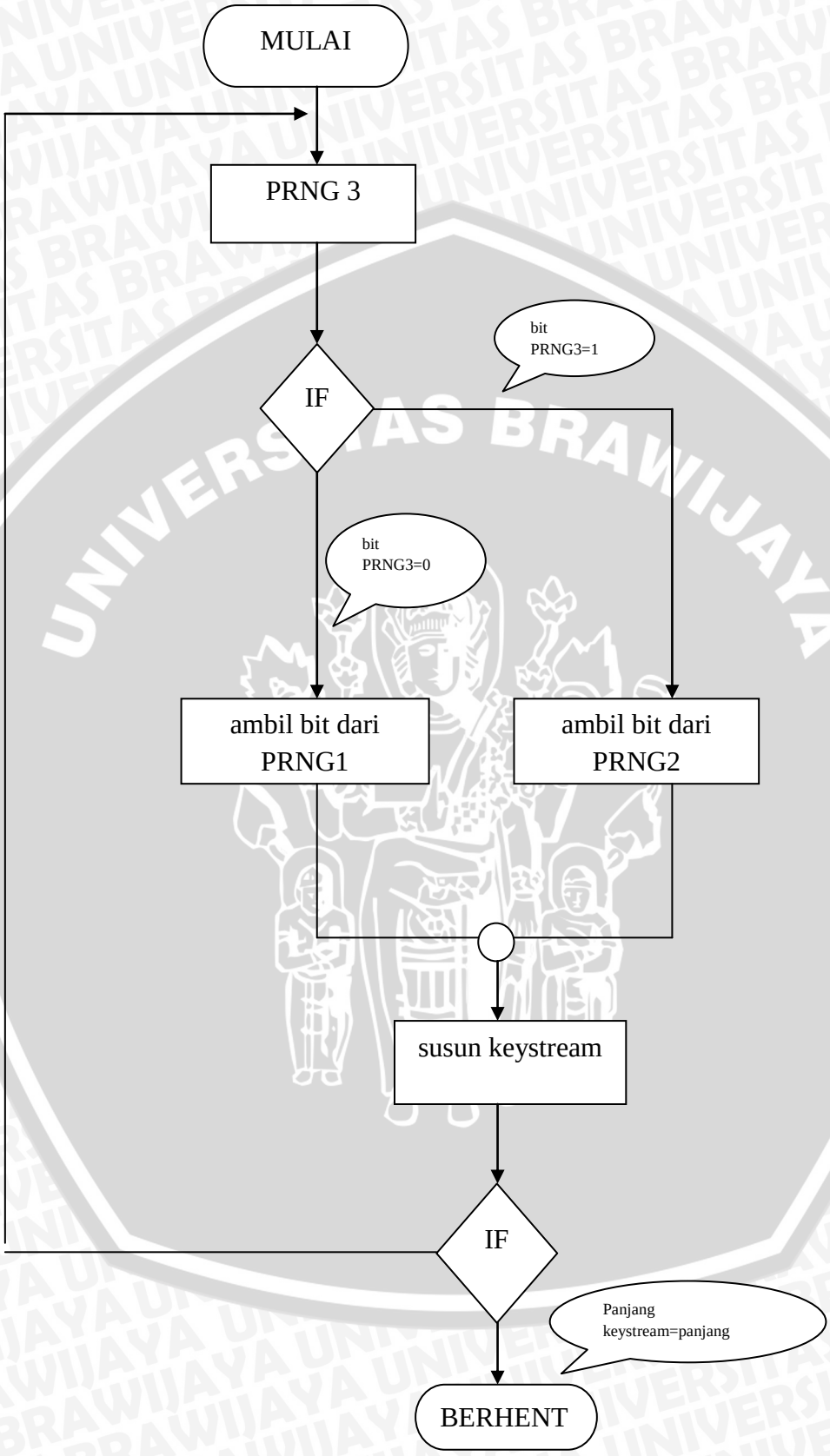
Pertama tama pengguna harus memasukkan kunci yang nantinya akan berfungsi sebagai *seed* dari *generator* pembangkit bilangan acak semu. Karena disini menggunakan 3 buah *generator* PRNG , maka pengguna harus memasukkan 3 buah kunci atau dengan kata lain sama dengan memasukkan 3 buah *initial state* untuk 3 buah *generator* PRNG yang masing masing kunci. panjangnya adalah 32 bit, dimana untuk masing masing kunci akan direpresentasikan dengan bilangan dari 0 - 4.294.770.687 Pemilihan kunci sepanjang 32 bit ini bertujuan untuk mempermudah pengguna dalam proses memasukkan kunci enkripsi dekripsi. Namun pada prosesnya setelah diubah pada mode bit, panjang kunci tersebut akan dilakukan proses *padding* supaya panjang kunci atau panjang *initial state* yang dimasukkan user sebelumnya pada akhirnya akan bertambah panjang memenuhi jumlah state untuk masing masing PRNG. Untuk kunci nomor 1 akan dimulai dari 0 bit, untuk kunci nomor 2 akan dimulai dari 3 bit, dan untuk kunci nomor 3 akan dimulai 0 bit. *Keystream* akan terbentuk setelah bilangan acak semu dari *generator* nomor 1 dan bilangan acak semu dari *generator* nomor 2 di*multipleksing* dengan bilangan acak semu dari *generator* nomor 3. Kemudian bit dari *keystream* akan di XOR dengann bit *plaintext* sehingga menjadi *ciphertext*.

Pada proses dekripsinya tidak jauh berbeda dengan proses enkripsi. Pada proses dekripsi pengguna akan diminta memasukkan 3 buah kunci yang sama dengan kunci yang sama dengan kunci yang dipakai untuk proses enkripsi supaya *ciphertext* bisa kembali menjadi *plaintext* yang mengandung informasi asli. Kunci terdiri dari 32 bit yang direpresentasikan dengan bilangan antara 0 - 4.294.770.687. 3 buah kunci tersebut fungsinya sama seperti pada proses enkripsi sebelumnya, yaitu sebagai *initial state* dari 3 buah *generator* PRNG yang nantinya setelah proses *multipleksing* akan terbentuk *keystream*. Bit bit dari keystream akan diXOR dengan bit bit *ciphertext*. Berdasarkan tabel kebenaran dari operasi XOR, apabila sebuah bit “x” diXOR dengan bit lain sebanyak dua kali, dimana bit yang dipakai pada proses XOR pertama tadi sama dengan bit yang dipakai dengan bit pada proses XOR kedua, maka hasil keluaran terakhirnya sama dengan bit “x”. Berdasarkan karakteristik dari tabel kebenaran tersebut, apabila bit *ciphertext* di XOR dengan bit bilangan acak semu yang sama seperti bit yang dipakai pada

proses enkripsi, maka keluarannya adalah bit *plaintext* semula dengan kata lain proses akan mengembalikan *ciphertext* menjadi *plaintext* semula.



Gambar 4.3 Proses Deskripsi



Gambar 4.4 Proses *Multipleksing*

Diagram alir berikutnya ini akan menggambarkan penerapan proses *multipleksing* pada algoritme enkripsi dekripsi yang akan dirancang. Masukan *multiplekser* adalah bit dari *generator* PRNG nomor satu dan bit dari *generator* PRNG nomor dua, sementara bit dari *generator* nomor tiga berfungsi sebagai *select multiplekser*. Apabila keluaran bit dari *generator* nomor tiga bernilai "1", maka bit penyusun keystream diambil dari bit *generator* nomor dua dan sebaliknya apabila bit keluaran dari *generator* nomor tiga bernilai "0" maka bit penyususn keystream diambil dari bit keluaran dari *generator* nomor satu. Proses *multipleksing* akan terus berlanjut sampai bit bit *plaintext* terakhir.

4.4 Algoritme

Pada bagian algoritme ini akan dijelaskan semua algoritme yang dipakai dalam proses perancangan.

```
function a=acaka(seed1,seed2,totaldata)
    rng(seed1);
    for n=1:totaldata
        ranvar1(n)=randi(2.^32);
    end
    rng(seed2);
    for n=1:totaldata
        ranvar2(n)=randi(2.^32);
    end
    for n=1:totaldata
        MUX_OUT(n)=muxx(ranvar1(n),ranvar2(n));
    end
    a=MUX_OUT;

    end


function m=enkripsi(mykeynumber)
fileidread=fopen('masaru.txt','r');
a=fscanf(fileidread,'%c')

for i=1:size(a,2)
    charToAsc=uint32(a(i))
    b(i)=bitxor(charToAsc,mykeynumber)
end
fileidwrite=fopen('hasil_enkripsi.txt','w');
fprintf(fileidwrite,'%d ',b);

fclose(fileidread)
fclose(fileidwrite)

function decrip(mykeynumber)
```

```
fileidread=fopen('hasil_enkripsi.txt','r');
fileidwrite=fopen('hasil_dekripsi.txt','w');

a=fscanf(fileidread,'%d')

for i=1:size(a,1)
    temp=a(i)
    b(i)=bitxor(temp,mykeynumber);

end

fprintf(fileidwrite,'%c',b);
fclose(fileidread)
fclose(fileidwrite)
```

4.5 Algoritme Membangkitkan Bilangan Acak Semu *Mersenne Twister*

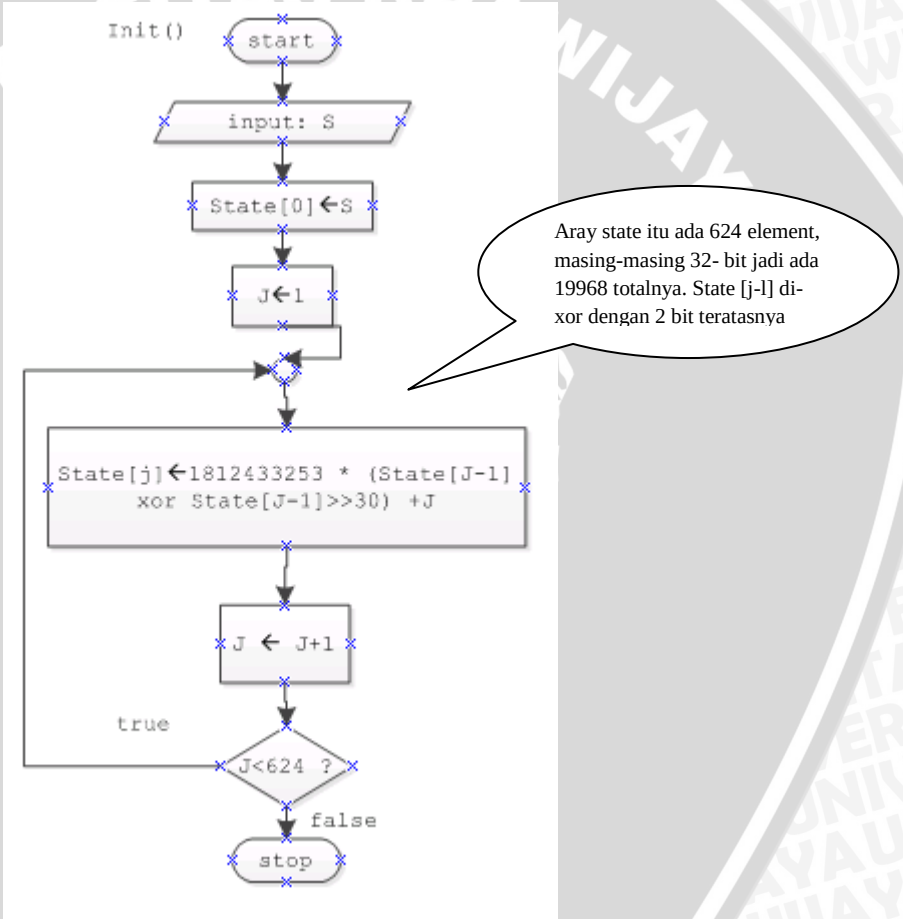
```
#include <stdio.h>

/* Period parameters */
#define N 624
#define M 397
#define MATRIX_A 0x9908b0dfUL   /* constant vector a */
#define UMASK 0x80000000UL /* most significant w-r bits */
#define LMASK 0x7fffffffUL /* least significant r bits */
#define MIXBITS(u,v) ( ((u) & UMASK) | ((v) & LMASK) )
#define TWIST(u,v) ((MIXBITS(u,v) >> 1) ^ ((v)&1UL ? MATRIX_A : 0UL))

static unsigned long state[N]; /* the array for the state vector  */
static int left = 1;
static int initf = 0;
static unsigned long *next;

/* initializes state[N] with a seed */
void init_genrand(unsigned long s)
{
    int j;
    state[0]= s & 0xffffffffUL;
    for (j=1; j<N; j++) {
        state[j] = (1812433253UL * (state[j-1] ^ (state[j-1] >> 30)) + j);
        state[j] &= 0xffffffffUL;  /* for >32 bit machines */
    }
    left = 1; initf = 1;
}

/* initialize by an array with array-length */
/* init_key is the array for initializing keys */
/* key_length is its length */
/* slight change for C++, 2004/2/26 */
void init_by_array(unsigned long init_key[], int key_length)
```

```
{
    int i, j, k;
    init_genrand(19650218UL);
    i=1; j=0;
    k = (N>key_length ? N : key_length);
    for (; k; k--) {
        state[i] = (state[i] ^ ((state[i-1] ^ (state[i-1] >> 30)) * 1664525UL))
          + init_key[j] + j; /* non linear */
        state[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
        i++; j++;
        if (i>=N) { state[0] = state[N-1]; i=1; }
        if (j>=key_length) j=0;
    }
    for (k=N-1; k; k--) {
        state[i] = (state[i] ^ ((state[i-1] ^ (state[i-1] >> 30)) * 1566083941UL))
          - i; /* non linear */
        state[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
        i++;
        if (i>=N) { state[0] = state[N-1]; i=1; }
    }

    state[0] = 0x80000000UL; /* MSB is 1; assuring non-zero initial array */
    left = 1; initf = 1;
}

static void next_state(void)
{
    unsigned long *p=state;
    int j;

    /* if init_genrand() has not been called, */
    /* a default initial seed is used         */
    if (initf==0) init_genrand(5489UL);

    left = N;
    next = state;

    for (j=N-M+1; --j; p++)
        *p = p[M] ^ TWIST(p[0], p[1]);

    for (j=M; --j; p++)
        *p = p[M-N] ^ TWIST(p[0], p[1]);

    *p = p[M-N] ^ TWIST(p[0], state[0]);
}

/* generates a random number on [0,0xffffffff]-interval */
unsigned long genrand_int32(void)
{
    unsigned long y;
```

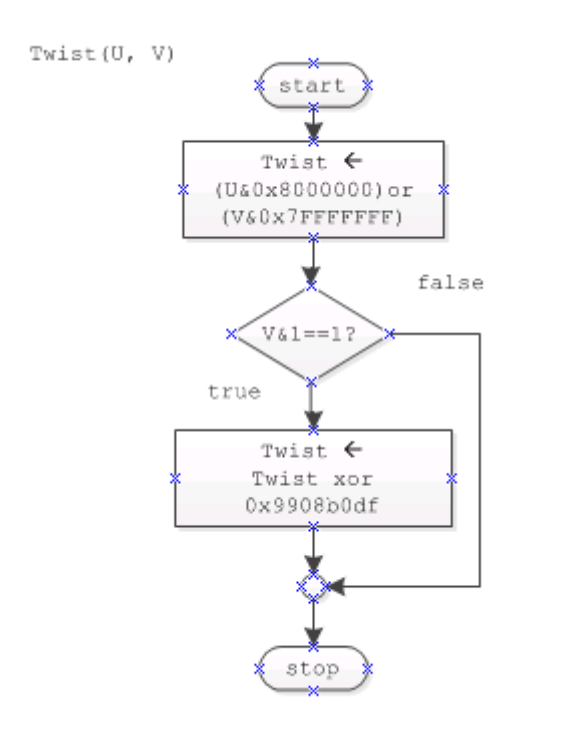
```
    if (--left == 0) next_state();
    y = *next++;

    /* Tempering */
    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680UL;
    y ^= (y << 15) & 0xefc60000UL;
    y ^= (y >> 18);

    return y;
}

/* generates a random number on [0,0x7fffffff]-interval */
long genrand_int31(void)
{
    unsigned long y;

    if (--left == 0) next_state();
    y = *next++;

    /* Tempering */
    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680UL;
    y ^= (y << 15) & 0xefc60000UL;
    y ^= (y >> 18);

    return (long)(y>>1);
}

/* generates a random number on [0,1]-real-interval */
double genrand_real1(void)
{
    unsigned long y;

    if (--left == 0) next_state();
    y = *next++;

    /* Tempering */
    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680UL;
    y ^= (y << 15) & 0xefc60000UL;
    y ^= (y >> 18);

    return (double)y * (1.0/4294967295.0);
    /* divided by 2^32-1 */
}

/* generates a random number on [0,1)-real-interval */
double genrand_real2(void)
```

```
{
    unsigned long y;

    if (--left == 0) next_state();
    y = *next++;

    /* Tempering */
    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680UL;
    y ^= (y << 15) & 0xefc60000UL;
    y ^= (y >> 18);

    return (double)y * (1.0/4294967296.0);
    /* divided by 2^32 */
}

/* generates a random number on (0,1)-real-interval */
double genrand_real3(void)
{
    unsigned long y;

    if (--left == 0) next_state();
    y = *next++;

    /* Tempering */
    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680UL;
    y ^= (y << 15) & 0xefc60000UL;
    y ^= (y >> 18);

    return ((double)y + 0.5) * (1.0/4294967296.0);
    /* divided by 2^32 */
}

/* generates a random number on [0,1) with 53-bit resolution*/
double genrand_res53(void)
{
    unsigned long a=genrand_int32()>>5, b=genrand_int32()>>6;
    return(a*67108864.0+b)*(1.0/9007199254740992.0);
}

int main(void)
{
    int i;
    unsigned long init[4]={0x123, 0x234, 0x345, 0x456}, length=4;
    init_by_array(init, length);
    /* This is an example of initializing by an array.       */
    /* You may use init_genrand(seed) with any 32bit integer */
    /* as a seed for a simpler initialization                */
    printf("1000 outputs of genrand_int32()\n");
```

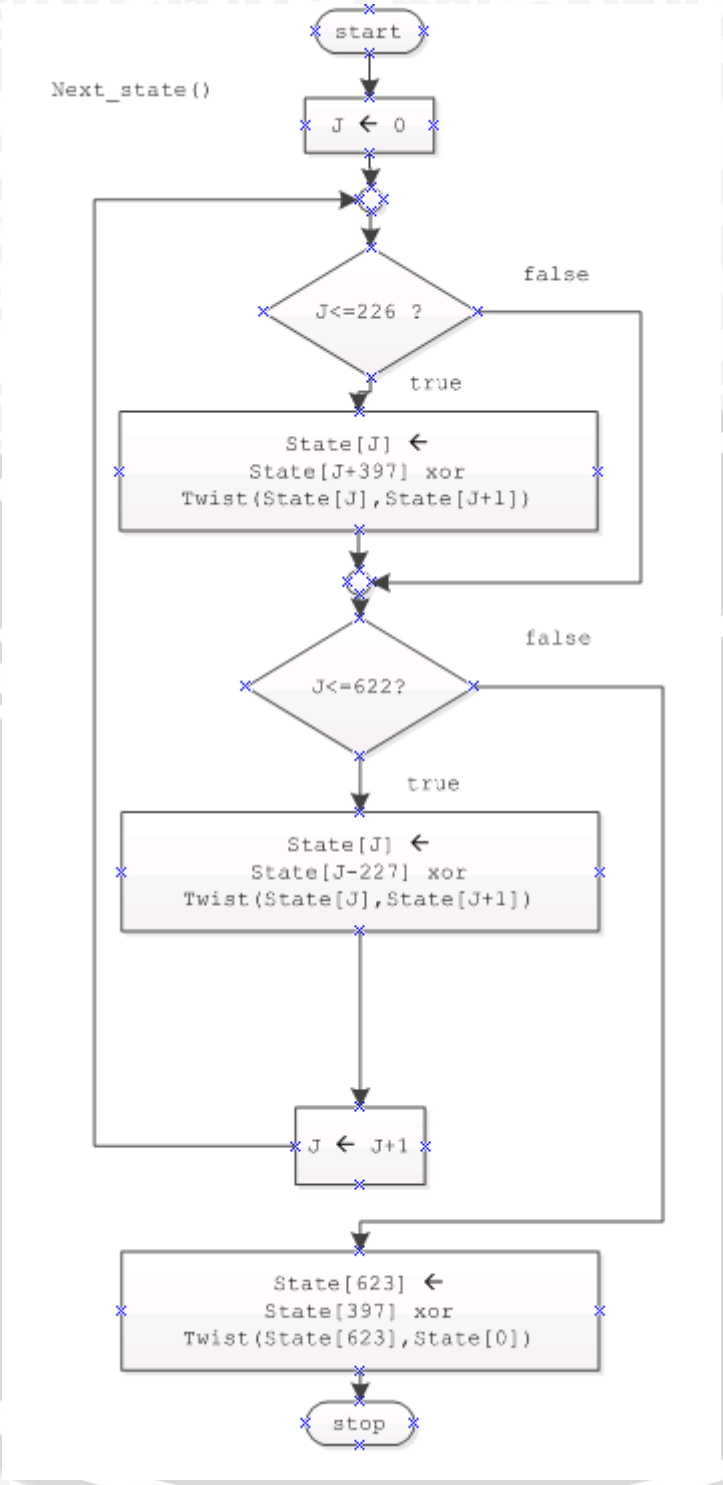
```
    for (i=0; i<1000; i++) {
      printf("%10lu ", genrand_int32());
      if (i%5==4) printf("\n");
    }
    printf("\n1000 outputs of genrand_real2()\n");
    for (i=0; i<1000; i++) {
      printf("%10.8f ", genrand_real2());
      if (i%5==4) printf("\n");
    }

    return 0;
}
```



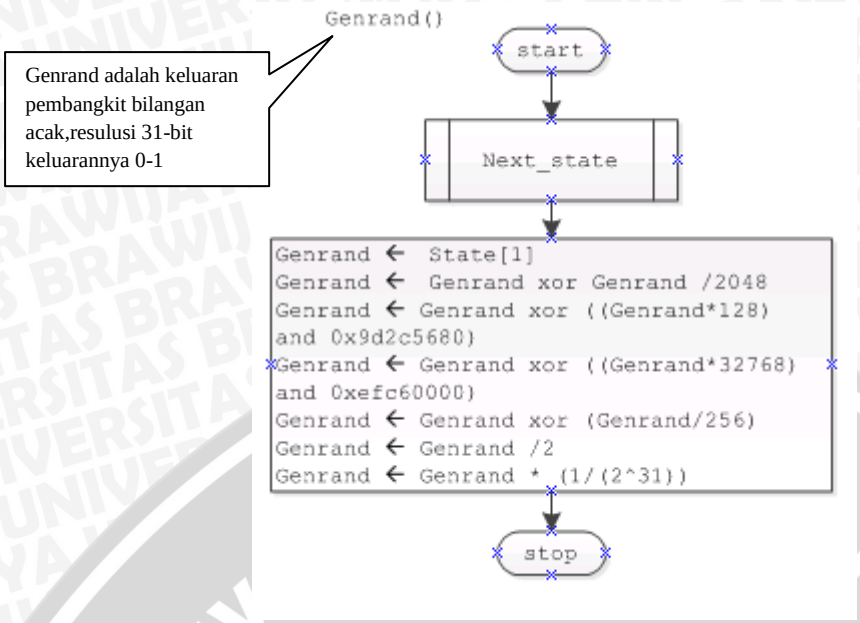
Gambar 4.5 Diagram Alir *Mersenne Twister*



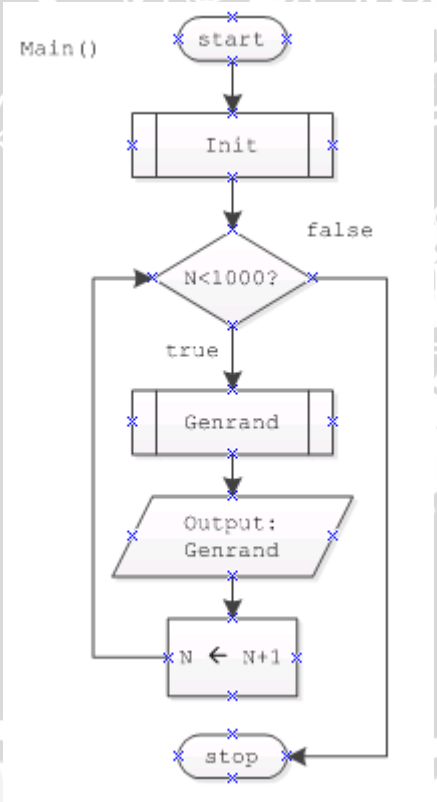
Gambar 4.6 Diagram Alir *Mersenne Twister*



Gambar 4.7 Diagram Alir *Mersenne Twister*



Gambar 4.8 Diagram Alir *Mersenne Twister*



Gambar 4.9 Diagram Alir *Mersenne Twister*

BAB V
PENGUJIAN DAN ANALISIS

Pada bab ini dilakukan proses pengujian dan analisis terhadap *single Mersenne Twister* serta *multi Mersenne Twister* yang telah dibangun. Proses pengujian yang akan dilakukan yaitu pengujian statistik keacakan bilangan acak semu yang dihasilkan *generator* pada program MATLAB.

5.1 Pengujian Autokorelasi

Pada uji ini, diasumsikan bahwa adalah barisan bit dengan panjang *n* dan merupakan variabel acak. karena merupakan kejadian saling bebas dimana . Tujuan dari uji autokorelasi adalah untuk memeriksa korelasi antara barisan bit dengan versi pergeserannya (Menezes,dkk.,1996).

Misalkan *d* adalah bilangan bulat dengan $1 \leq d \leq [\frac{n}{2}]$. Jumlah bit dalam barisan *s* yang tidak sama dengan barisan *s* yang telah digeser sejauh *d* adalah A(d)= $\sum_{i=o}^{n-d-1} Si \oplus$ dengan $\oplus$ adalah operasi XOR. Operasi XOR merupakan operasi logika bitwise yang bekerja dengan membandingkan dua buah bit yang apabila pada salah satu bitnya bernilai benar (dinotasikan dengan bit 1) maka hasil akhir operasi XOR tersebut adalah benar. Namun, bila kedua bit yang akan dibandingkan bernilai salah (dinotasikan dengan bit 0) atau benar maka hasil akhirnya adalah salah. Statistik ujinya adalah :

$$X_5=\frac{2(A(d)-\frac{n-d}{2}}{\sqrt{n-d}}$$

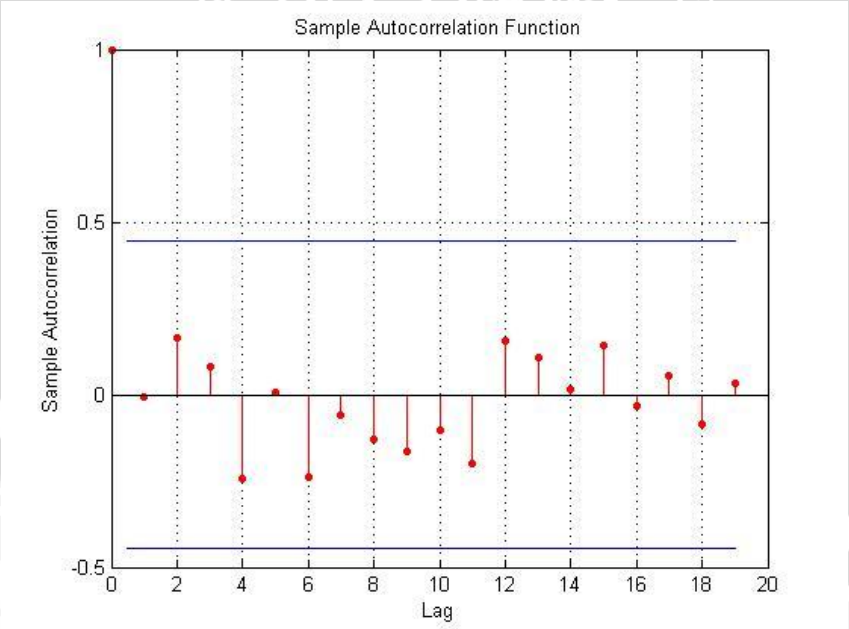
Mendekati distribusi N(0,1) jika $n - d \geq 10$

5.1.1 Pengujian *Single* PRNG

Hasil pengujian dengan menggunakan uji autokorelasi pada barisan yang dihasilkan oleh PRNG *Mersenne Twister* ditampilkan dalam bentuk tabel. Table memperlihatkan hasil uji autokorelasi *single seed* berukuran 32 bit yang dibangkitkan PRNG *Mersenne Twister*.

Tabel 5.1 Hasil Uji Seed 54321

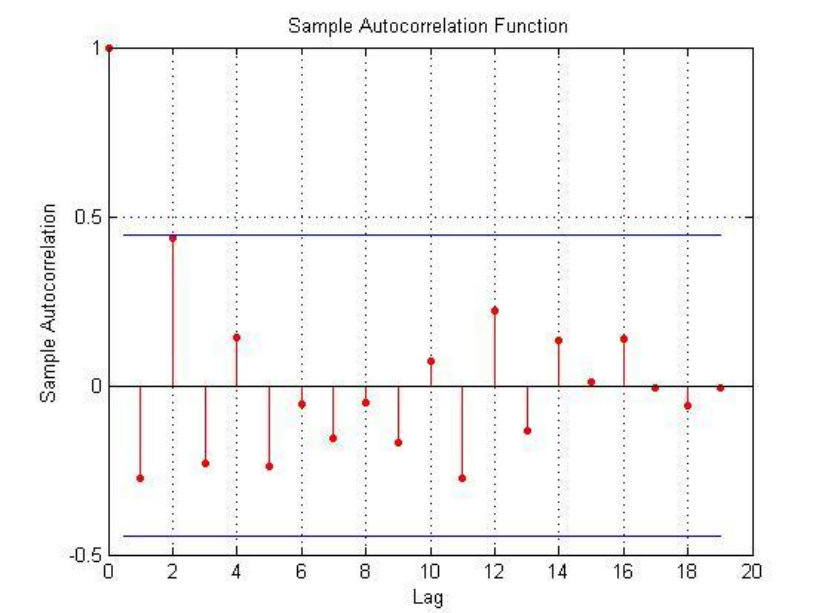
No.	Seed	Random
1	54321	3.9155
2	54321	2.6793
3	54321	3.4008
4	54321	1.8459
5	54321	2.3322
6	54321	1.7758
7	54321	0.3678
8	54321	3.3398
9	54321	2.1000
10	54321	0.2170
11	54321	2.3125
12	54321	0.1781
13	54321	3.4707
14	54321	4.2746
15	54321	2.7432
16	54321	4.0624
17	54321	3.1642
18	54321	3.5307
19	54321	0.4917
20	54321	3.1569



Gambar 5.1 Grafik autokorelasi dari seed 54321

Tabel 5.2 Hasil Uji Seed 54321

No.	Seed	Random
1	1	3.9155
2	1	2.6793
3	1	3.4008
4	1	1.8459
5	1	2.3322
6	1	1.7758
7	1	0.3678
8	1	3.3398
9	1	2.1000
10	1	0.2170
11	1	2.3125
12	1	0.1781
13	1	3.4707
14	1	4.2746
15	1	2.7432
16	1	4.0624
17	1	3.1642
18	1	3.5307
19	1	0.4917
20	1	3.1569



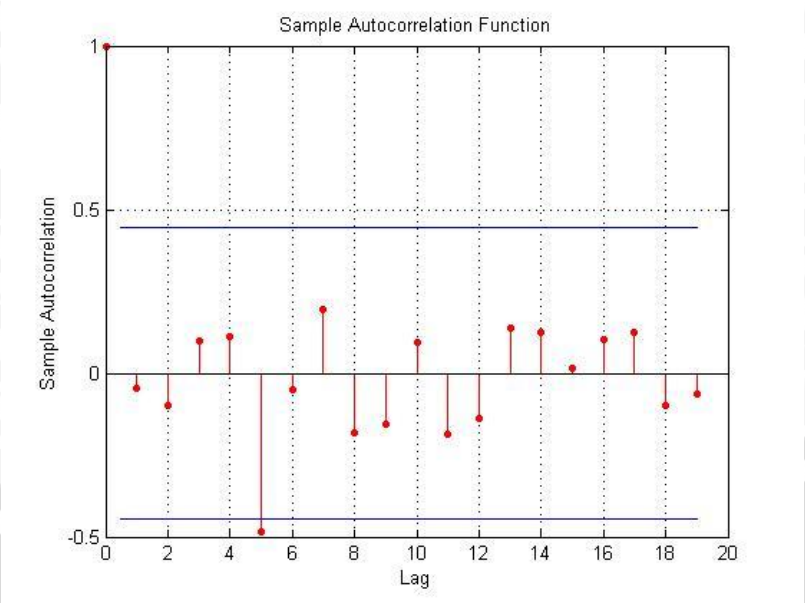
Gambar 5.2 grafik autokorelasi dari seed

5.1.2 Pengujian *Multi* PRNG

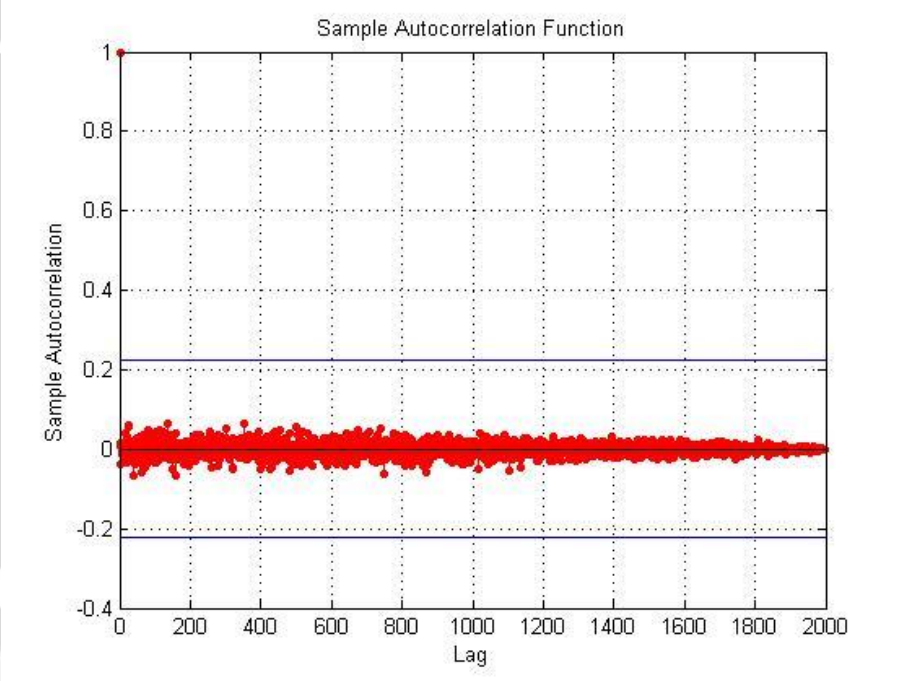
Selain menerapkan uji autokorelasi dengan menggunakan *single* PRNG , dilakukan juga pengujian *multi seed* oleh dua buah PRNG *Mersenne Twister*. Hasil uji beserta kesimpulan *multi* PRNG tersebut disajikan pada Tabel berikut :

Tabel 5.3 Hasil Uji *Multi* PRNG dari Seed 54321,1

No.	Seed	Random
1	54321,1	3.9154
2	54321,1	2.6793
3	54321,1	0.4912
4	54321,1	1.8459
5	54321,1	2.3322
6	54321,1	0.3965
7	54321,1	0.7999
8	54321,1	3.3398
9	54321,1	2.1000
10	54321,1	0.2170
11	54321,1	1.8004
12	54321,1	2.9429
13	54321,1	0.8781
14	54321,1	4.2745
15	54321,1	2.7432
16	54321,1	4.0624
17	54321,1	1.7923
18	54321,1	3.5307
19	54321,1	0.6029
20	54321,1	0.8508



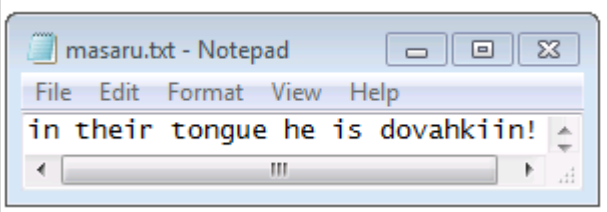
Gambar 5.3 Grafik Autokorelasi dari *multi* PRNG Seed 54321,1



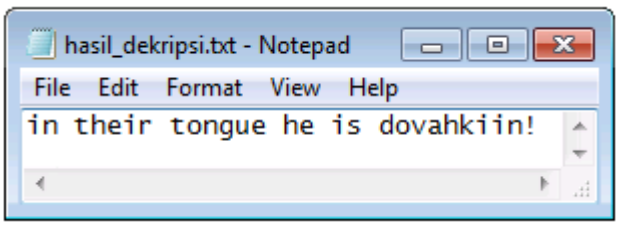
Gambar 5.4 Grafik Autokorelasi dari 54321,1 sebanyak 2000 lags

Tabel 5.4 Hasil Uji Enkripsi Deskripsi dengan menggunakan kunci 491237

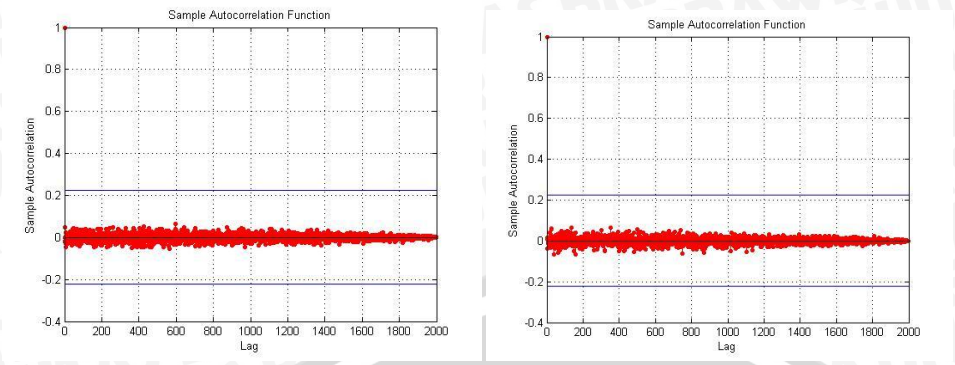
No.	Hasil Enkripsi	Hasil Deskripsi	No.	Hasil Enkripsi	Hasil Deskripsi
1	491148	I	21	491158	S
2	491147	N	22	491205	
3	491205		23	491137	D
4	491153	T	24	491146	O
5	491149	H	25	491155	V
6	491136	E	26	491140	A
7	491148	I	27	491149	H
8	491159	R	28	491150	K
9	491205		29	491148	I
10	491153	T	30	491148	I
11	491146	O	31	491147	N
12	491147	N	32	491204	!
13	491138	G			
14	491152	U			
15	491136	E			
16	491205				
17	491149	H			
18	491136	E			
19	491205				
20	491148	I			



Gambar 5.5 Teks yang akan di uji



Gambar 5.6 Teks hasil deskripsi



Gambar 5.7 Perbandingan *single* PRNG dan *multi* PRNG 2000 lags

5.2 Analisi Pengujian

Pada kenyataannya aplikasi PRNG menggunakan Mersenne Twister ini tidak selalu berjalan dengan baik. Ada beberapa faktor yang bisa menyebabkan pengujian yang dilakukan menjadi kurang baik. Berikut adalah beberapa faktor-faktor yang menyebabkan kegagalan tersebut.

1. Pada gambar Grafik 5.2 data ke-2 menunjukan kenaikan yang signifikan yang menjukan keacakan angkanya kurang bagus.
2. Pada gambar Grafik 5.3 data ke-5 menunjukan kenaikan yang signifikan yang menjukan keacakan angkanya kurang bagus.
3. Pada gambar Grafik 5.3 data ke-1 sampai data ke-1200 grafik masih terlihat tidak stabil yang menjukan keacakan angkanya kurang bagus.

Grafik perlahan terlihat konsisten setelah data 1200 dengan di tunjukan pada grafik 5.3 yang semakin lama semakin mengerucut mengecil. Ini menunjukan nilai keacakan angkanya menjadi semakin baik.

BAB VI
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berdasarkan hasil penelitian dapat disimpulkan bahwa :

1. Pada uji autokorelasi, jumlah bit dalam suatu barisan yang tidak sama dengan versi pergeserannya merupakan kejadian binomial sehingga diperoleh bahwa statistik uji autokorelasi mendekati distribusi normal baku.
2. Panjang barisan bit minimal agar hasil uji autokorelasi valid dan jumlah pergeseran maksimal adalah sebesar dengan *n* adalah panjang barisan bit dan *d* adalah banyaknya pergeseran.
3. Hasil pengujian autokorelasi terhadap satu barisan bit yang berasal dari *single* PRNG *Mersenne Twister* maupun terhadap *multi* PRNG *Mersenne Twister* menunjukkan uji autokorelasi cukup efektif untuk mendeteksi adanya korelasi antara barisan bit dengan versi pergeserannya baik ketika pergeserannya kecil (d=1) maupun ketika pergeseran bitnya besar ($d \approx \lfloor \frac{n}{2} \rfloor$).
4. Pada uji autokorelasi di lag awal masih ada ketidak stabilan sehingga masih ada grafik yang naik turun hingga lag mencapai panjang 1200 baru grafik terlihat stabil.
5. Pengacakan angka menggunakan *multi* PRNG lebih efektif untuk meningkatkan nilai keracakan PRNG.

6.2 Saran

Dalam perancangan dan pembuatan aplikasi ini masih terdapat kekurangan dan kelemahan, oleh karena itu masih diperlukan penyempurnaan untuk pengembangan kedepannya. Berikut adalah beberapa hal yang perlu diperhatikan dan disempurnakan :

1. Diharapkan penelitian selanjutnya dapat ditingkatkan lagi untuk proses enkripsi yang lebih baik.
2. Diharapkan penelitian selanjutnya dapat ditingkatkan lagi untuk proses keacakan bilangannya
3. Diharapkan dapat dikembangkan lagi algoritme dari *Pseudo Random Number Generator* yang ada.

DAFTAR PUSTAKA

Andresta Ramadhan. 2013.Perbandingan Algoritme Linear Congruential *Generator*s, BlumBlumShub, dan MersenneTwister untuk Membangkitkan Bilangan Acak Semu. Bandung:Institut Teknologi Bandung.

Anggun Triyogo. 2012.Pengembangan Algoritma Enkripsi Deskripsi Berbasis LFSR Menggunakan Polinomial Primitif. Malang:Universitas Brawijaya

Anonymous. 2009.Mersenne Twister. https://en.wikipedia.org/wiki/Mersenne_Twister. Diakses tanggal 27 februari 2015.

Matsumoto, Takuji Nishimura. 1997.Mersenne Twister. Hiroshima:Universitas Hiroshima.

Sari Agustini Hafman, Arif Fachru Rozi. 2013.Analisis Teoritis dan Penerapan Uji Autokorelasi dari *Five Basic Test* untuk Menguji Keacakan Barisan Bit. Malang:Lembaga Sandi Negara.