BAB II

LANDASAN TEORI

2.1 Pengertian Simulasi

Simulasi adalah suatu cara untuk menduplikasi atau menggambarkan ciri , tampilan, dan karakteristik dari suatu sistem yang nyata secara matematis. Simulasi juga bisa dikatakan sebagai sebuah tiruan dari sistem yang dinamis dengan menggunakan model komputer yang digunakan untuk mengevaluasi dan meningkatkan kerja sistem [6]. Simulasi merupakan salah satu teknik yang sering kali digunakan dalam suatu penelitian dan manajemen ilmu pengetahuan, dan oleh semua indikasinya hal tersebut sangat populer dikembangkan. Dari proses simulasi kita dapat mempelajari karakter maupun sifat operasional dari sistem fisik maupun abstrak.

Pengertian lain tentang simulasi adalah sebagai berikut :

a. Cara untuk mereproduksi kondisi situasi dengan menggunakan model, untuk mempelajari, menguji, melatih dll. (Oxford American Dictionary ,1980)
b. Pemodelan dari sebuah proses atau sistem dimana model meniru respon dari sistem nyata untuk setiap kejadian yang terjadi setiap saat. (Schriber,1987).

2.2 Pengacakan (Randomize)

Kata ‘acak’ (*random*) sering digunakan untuk menyatakan sesuatu yang serba kurang teratur dan tanpa tujuan yang jelas. Contohnya, pada saat melemparkan sebuah dadu, angka yang akan muncul hampir tidak dapat diprediksi. Kemungkinan hasil prediksi benar hanya bergantung pada faktor keberuntungan saja. Selain itu, kondisi acak juga muncul dalam beberapa bidang berikut ini :

1. Bidang penelitian.

   Ketika melakukan suatu eksperimen, semua variabel telah dikontrol dan ditetapkan, namun sisa proses yang didapat tetap akan bervariasi. Variasi sisa ini merupakan salah satu hal yang tidak dapat dikontrol. Sekalipun jika kemungkinan ini dapat dikontrol secara sempurna, hasil pengujian juga akan menunjukkan hal yang sama.

2. Bidang matematika.

   Teori probabilitas matematika yang muncul dengan maksud menjelaskan formula matematika dari perubahan peluang. Dalam ilmu Statistik dipakai untuk menduga distribusi probabilitas dari suatu kumpulan dari observasi empiris. Nilai *random* juga memegang peranan penting dalam simulasi karena pada dasarnya keadaan yang disimulasikan memang bersifat tidak pasti.

3. Bidang Kriptografi

   Akses pada suatu sumber *randomize* dengan kualitas tinggi secara absolut bersifat kritikal dalam banyak aplikasi kriptografi. Sebagai contoh, suatu pilihan *non-random* mungkin menghasilkan suatu penyusupan dalam kanal

komunikasi yang dipercayai telah diamankan. Misalnya pada mesin Enigma dan artikel mengenai *one-time pad* merupakan konsekuensi dari perhitungan yang gagal. Kunci yang digunakan pada Enigma tidak secara acak dalam banyak kasus di mana memungkinkan kriptanalis sekutu untuk memecahkan lalu lintas komunikasi tersebut.

Terdapat suatu teknik untuk mengkombinasikan data bias *random* untuk menghasilkan kualitas *randomize* yang baik. Randomize diturunkan oleh beberapa komponen *input output* (seperti contoh *disk drive*) mempunyai cukup *entropy* untuk diekstraksi untuk tujuan kriptografi. Suatu perhatian lebih untuk memastikan praktek terbaik yang diikuti oleh sistem aktual dan suatu generasi nilai acak untuk dipecahkan oleh penyerang.

4. Bidang Permainan

   Randomize merupakan pusat dari *games of chance* dan penting dalam industri perjudian. Penggambaran *random* sering digunakan untuk membuat keputusan di mana tidak ada rasional atau basis yang adil untuk membuat suatu pilihan deterministik.

5. Bidang Peluang

   Penerapan bilangan acak terlihat secara nyata dalam bidang peluang ini. Contoh kasus yang paling sering ditemukan adalah pada waktu diadakan kuis ataupun undian berhadiah yang dilakukan oleh stasiun televisi. Peluang seseorang untuk terpilih dalam sebuah kuis ataupun undian berhadiah itu ditentukan secara acak.

2.3 Bilangan Acak

Bilangan acak adalah deretan nilai yang acak dan tidak dapat diprediksi secara keseluruhan. 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. Sedangkan ada beberapa RNG lainnya menghasilkan nilai acak dengan berfokus pada suatu area tertentu dan mendistribusikannya secara seragam.

2.3.1 Pembangkit Bilangan Acak (Random Number Generator)

Random number generator (RNG) adalah suatu peralatan komputasional yang dirancang untuk menghasilkan suatu urutan nilai yang tidak dapat ditebak polanya dengan mudah, sehingga urutan nilai tersebut dapat dianggap sebagai suatu keadaan acak (*random*) [2].

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*).

2.3.2 Pembangkit Bilangan Acak Semu (Pseudo Random Number Generator)

Suatu *pseudo random number generator* (PRNG) merupakan suatu algoritma 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 algoritma 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.

2.3.3 Algoritma Pembangkit Bilangan Acak Semu

Semua deretan bilangan acak yang dibangkitkan dari rumus matematika, serumit apapun, dianggap sebagai deret acak semu, karena dapat diulang pembangkitannya [7]. Sementara itu, banyak produk *software* yang dinyatakan sebagai produk yang aman karena menggunakan bilangan acak semacam OTP (*One Time Programmable*). Namun karena OTP ini dibangkitkan dari bilangan acak semu, maka keamanan yang diperoleh juga semu. Pembangkit bilangan acak yang sering diimplementasikan adalah *Linier Congruential Generator* (LCG) dan *Linear Feedback Shift Register* (LFSR). Beberapa algoritma lainnya yang dapat

digunakan untuk membangkitkan deretan bilangan acak semu adalah sebagai berikut :

1. *Non Linear Feedback Shift Register* (NLFSR).
2. *Indirection, Shift, Accumulate, Add and Count* (ISAAC).
3. *Lagged Fibonacci Generator* (LFG).
4. *Mersenne Twister*.
5. *Fortuna*.
6. *Blum-Blum Shub.*

2.3.3.1 Linear Congruential Generator (LCG)

Linear Congruential Generator (LCG) mewakili salah satu algoritma *pseudo random number* yang tertua dan paling populer. Teori dari algoritma ini mudah dipahami dan dapat diimplementasikan secara cepat. Keuntungan dari LCG adalah operasinya yang sangat cepat.

LCG dapat didefinisikan dengan rumusan berikut :

$$\mathbf{X_n = (aX_{n-1} + b) \bmod m}$$

Periode dari LCG umumnya adalah sebesar nilai m.

Masalah pada LCG adalah *lower-order bit* yang digenerasi mempunyai periode yang lebih pendek dari deretan secara keseluruhan jika M di-set menjadi pangkat 2. Tanpa desain yang benar, dengan m yang sangat besar, bisa jadi perode bilangan acak yang dihasilkan tidak akan maksimal, bahkan mungkin jauh lebih pendek daripada periode maksimalnya. Menurut para ahli kriptografi, LCG tidak dapat digunakan untuk kriptografi karena urutan keluarannya dapat diprediksi.

Bahkan mereka juga dapat membongkar urutan bilangan acak yang dibangkitkan oleh pembangkit kuadratik. Walaupun LCG tidak bermanfaat untuk kriptografi, namun LCG bermanfaat untuk keperluan lainnya seperti simulasi.[8]

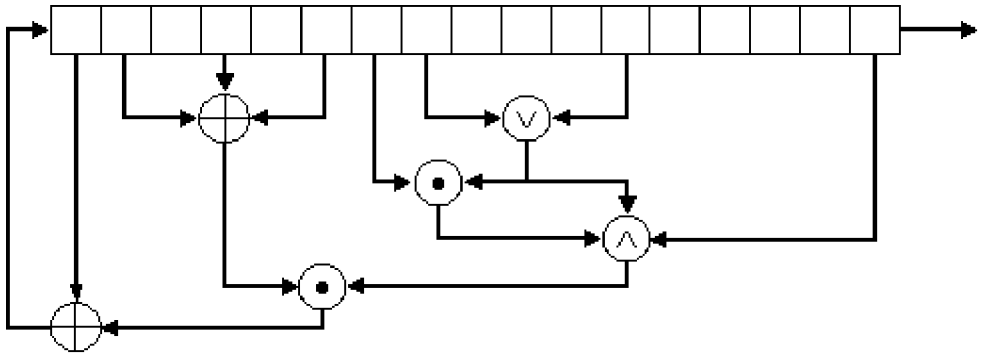
2.3.3.2 Non Linear Feedback Shift Register (NLFSR)

Mudah untuk membayangkan suatu rangkaian umpan balik (*feedback*) yang lebih rumit daripada yang digunakan LFSRs atau FCSRs. Masalahnya adalah tidak terdapat suatu teori matematika untuk menganalisanya. Secara khusus, beberapa masalah dengan rangkaian umpan balik *shift register nonlinear* adalah :

a. Mungkin terdapat prasangka (bias), seperti terdapat lebih banyak 1 daripada 0 atau lebih sedikit langkah daripada yang diharapkan pada keluaran.
b. Periode maksimum dari rangkaian mungkin jauh lebih rendah daripada yang diharapkan.
c. Periode rangkaian dapat berbeda untuk nilai awal yang berbeda.
d. Rangkaian dapat muncul secara acak untuk sesaat, tetapi kemudian terjadi "*dead end*" pada suatu nilai tunggal. Ini dapat diselesaikan dengan meng-XOR-kan fungsi *nonlinear* dengan bit paling kanan.

Sebaliknya, jika tidak ada teori untuk menganalisis sisi keamanan umpan balik *nonlinear shift register*, terdapat sedikit perangkat (*tool*) kriptanalis *stream cipher* yang didasarkan pada metode tersebut. Kita dapat menggunakan umpan balik *nonlinear shift register* pada rancangan *stream cipher*, tetapi kita perlu berhati-hati [8].

Dalam sebuah umpan balik *nonlinear shift register*, fungsi umpan balik dapat berupa apa saja yang diinginkan yang terlihat pada Gambar 2.1 berikut ini.



Gambar 2.1 A nonlinear-feedback shift register

2.3.3.3 ISAAC

Indirection, Shift, Accumulate, Add and Count (ISAAC) merupakan *pseudo random number generator* yang didesain oleh Bob Jenkins pada tahun 1996 untuk tujuan keamanan kriptografi.

Algoritma ini diinspirasi oleh RC4. ISAAC menggunakan sebuah *array* dari 256 yang berisi nilai 4 *byte integer* sebagai *internal state*, menulis hasilnya ke 256 *integer array* yang lain, di mana *array* tersebut hanya dibaca sekali pada suatu waktu hingga menjadi kosong, pada *point* di mana mereka dikomputasi ulang lagi [2].

2.3.3.4 Lagged Fibonacci Generator (LFG)

Lagged Fibonacci Generator (LFG) merupakan suatu contoh pseudo random number generator. LFG sering disebut juga sebagai *Additive Generators*. Jenis ini ditujukan pada peningkatan ‘standar’ dari LCG. Algoritma ini berdasarkan pada generasi urutan atau deret Fibonacci.

Deret Fibonacci dapat didefinisikan sebagai berikut :

$$\mathbf{F_n = F_{n-1} + F_{n-2}}$$

Jadi suatu rumusan baru adalah penjumlahan dari dua term dalam deret. Ini dapat digenerasi dengan rumusan ini :

$$\mathbf{S_n = S_{n-j} * S_{n-k} \bmod m} \; ; \qquad \text{di mana } 0 < j < k.$$

LFG mempunyai periode maksimum $(2^k - 1) * 2^{m-1}$ jika operasi penambahan atau XOR digunakan untuk mengkombinasikan nilai sebelumnya. Jika operasi perkalian yang digunakan, maka maksimum periode adalah $(2^k - 1) * 2^{m-3}$. Untuk memperoleh nilai periode maksimum dapat digunakan persamaan polinomial berikut:

$$\mathbf{y = x^k + x^j + 1}$$

Inisialisasi dari LFG merupakan masalah yang sangat kompleks, setiap periode maksimum dari LFG mempunyai kemungkinan siklus yang sangat besar, semuanya berbeda. Memilih suatu siklus adalah mungkin, tetapi metode untuk melakukan ini mungkin membahayakan *output* subsekuen *randomize*. Selain itu, *output* dari LFC sangat sensitif pada kondisi awal, dan efek statistikal mungkin muncul secara inisial ataupun secara periodikal dalam urutan *ouput* kecuali jika tindakan ekstra hati-hati diambil.[6]

2.3.3.5 Mersenne Twister

Mersenne Twister merupakan suatu pseudo random number generator yang dikembangkan pada tahun 1997 oleh Makoto Matsumoto dan Takuji Nishimura. Algoritma ini menawarkan generasi yang cepat dari bilangan acak

kualitas tinggi, telah dirancang secara khusus untuk mengatasi setiap cacat yang ditemukan pada algoritma terdahulu.

Terdapat dua varian dari algoritma, yang paling baru dan lebih umum dipakai adalah Mersenne Twister MT 19937. Algoritma ini mempunyai properti sebagai berikut :

a. Dirancang untuk mempunyai suatu periode kolosal 129937 – 1. Periode ini menjelaskan sesuai dengan namanya, suatu *Mersenne prime*, dan beberapa jaminan atas algoritma bergantung pada penggunaan internal dari *Mersenne prime*.
b. *Mersenne Twister* mempunyai suatu *equidistribution dimensional high order*. Ini berarti secara *default*, terdapat korelasi serial yang dapat diabaikan antara nilai suksesif dalam deret *output.*
c. Algroritma lebih cepat daripada algoritma yang ada tetapi menggunakan generator yang lebih statistikal.
d. Algoritma ini bersifat statistikal secara acak dalam semua bit outputnya.

Algoritma ini sendiri merupakan suatu *twisted generalised shift feedback register* atau TGSFR. Kata ‘twist’ merupakan suatu transformasi di mana menjamin *equidistribution* dari nilai yang digenerasi dalam 623 dimensi.

Tidak seperti Blum-Blum Shub, algoritma ini mempunyai bentuk bersifat *native* dan tidak cocok untuk kriptografi. Pada banyak aplikasi, bagaimana pun juga algoritma ini cepat dan telah menjadi algoritma *pseudo random generator* pilihan.

2.3.3.6 Fortuna

Fortuna merupakan salah satu jenis *crytographically secure pseudo random number generator* yang didesain oleh Niels Ferguson dan Bruce Schenier. Bentuk persamaan algoritma ini adalah :

$$\mathbf{X_n = X_{n-1}^3 \bmod c}$$

2.3.3.7 Blum-Blum Shub (BBS)

Blum-Blum Shub (BBS) merupakan suatu pseudo random number generator yang diajukan pada tahun 1986 oleh Lenore Blum, Manuel Blum dan Michael Shub. BBS mempunyai bentuk :

$$\mathbf{X_{n+1} = X_n^2 \bmod m}$$

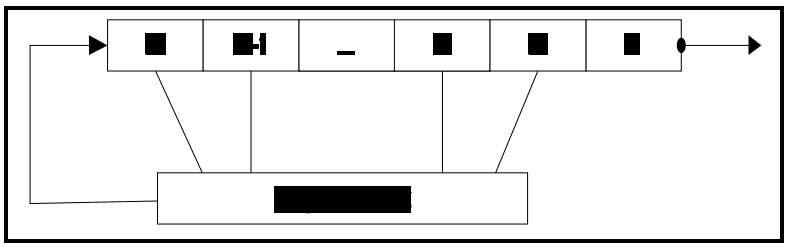
dengan m merupakan hasil dari perkalian dua buah bilangan prima besar p dan q dan *output*-nya dalam *least significant bit* dari X_n di mana hal yang sama sebagai *parity* dari X_n. Dua buah bilangan prima p dan q harus kongruen terhadap 3 mod 4 dan *greatest common divisor* (GCD) harus kecil.

Generator ini sering digunakan untuk aplikasi kriptografi, karena generator ini tidak begitu cepat. Bagaimanpun juga, generator ini mempunyai bukti keamanan yang kuat, di mana berhubungan dengan kualitas generator karena sulitnya faktorisasi integer [8].

2.4 Linear Feedback Shift Register (LFSR)

Suatu *Linear Feedback Shift Register* (LFSR) adalah suatu mekanisme untuk menghasilkan sekuens bit biner. *Register* memiliki sebarisan sel yang ditentukan oleh vektor inisialisasi yang biasanya menjadi kunci rahasia. Tingkah laku *register* diatur oleh sebuah *counter* (*clock*). Pada setiap saat isi sel dari

register digeser (*shift*) ke kanan sejauh satu posisi, dan hasil operasi XOR terhadap subset dari isi sel ditempatkan pada sel paling kiri [8]. Satu bit dari *output* biasanya diturunkan selama prosedur *update* ini.



Gambar 2.2 Linear Feedback Shift Register (LFSR)

Register geser umpan balik ini (Feedback Shift Register) mempunyai 2 bagian yang sangat penting :

1. Register Geser

   Yaitu barisan bit bit (bn, bn-1, bn-2, ... b4,b3,b2,b1) yang panjangnya n (disebut juga register geser n- bit)

2. Fungsi Umpan Balik

   Yaitu suatu fungsi yang menerima masukan dari register geser dan mengembalikan nilai fungsi ke register geser.

Tiap kali sebuah bit dibutuhkan, semua bit didalam register digeser 1 bit ke kanan. Bit paling kiri (bn) dihitung sebagai fungsi bit bit lain di dalam register tersebut. Keluaran dari register geser adalah 1 bit (yaitu bit b1 yang tergeser). Periode register geser adalah 1bit (yaitu bit b1 yang tergeser). Periode register geser adalah panjang keluaran sebelum ia berulang kembali.

LFSR cepat dan mudah untuk diimplementasikan baik secara *hardware* maupun secara *software*. Dengan pemilihan yang cermat terhadap alat kontrol (*pad*) *feedback* (bit tertentu yang digunakan) sekuens yang dihasilkan dapat memiliki performansi yang baik secara statistik. Akan tetapi, sekuens yang dihasilkan oleh LFSR tunggal tidak cukup aman karena sebuah kerangka matematik yang kuat telah dikembangkan selama bertahun-tahun yang memungkinkan analisis yang mudah terhadap hasil tersebut. Namun demikian, LFSR sangat berguna sebagai blok bangunan dalam sistem yang lebih aman.[8]

Suatu *shift register cascade* merupakan sebuah set dari LFSR yang dihubungkan bersama-sama dengan cara tertentu sedemikian hingga perlakuan dari LFSR sebelumnya dalam bentuk aliran yang menyerupai air terjun. Sifat dan tingkah laku yang saling tergantung ini biasanya ditempuh dengan menggunakan satu LFSR untuk mengontrol LFSR berikutnya. Sebagai contoh, satu *register* mungkin melangkah lebih dulu satu fase jika *output register* di depannya adalah 1 dan maju dua langkah jika hasil = 0. Beberapa konfigurasi yang berbeda mungkin dan pemilihan parameter tertentu nampaknya menawarkan tingkat keamanan yang baik. Sesuai dengan sifat keaktifannya, LFSR yang digunakan dapat terbagi menjadi dua jenis, yaitu :

1. LFSR yang selalu aktif.
2. LFSR yang hanya akan aktif kalau di-*clock* oleh LFSR lain.

Beberapa jenis generator yang hanya menggunakan LFSR yang selalu aktif adalah :

a. *Geffe Generator*.
b. *Jennings Generator*.
c. *Threshold Generator*.
d. *Multispeed Inner-Product Generator*.
e. *Gollmann Cascade*.

Sedangkan, beberapa jenis generator yang menggunakan kedua jenis LFSR di atas adalah :

a. *Beth-Piper Stop-and-Go Generator*.
b. *Alternating Stop-and-Go Generator*.
c. *Bilateral Stop-and-Go Generator*.

Selain itu, terdapat suatu generator yang menggunakan LFSR yang di-*clock* oleh dirinya sendiri. Generator ini disebut sebagai *Self-Decimated Generators*.

2.4.1 Geffe Generator

Generator ini menggunakan tiga atau lebih LFSR. Keluaran generator-generator ini dikombinasikan secara nonlinier. Beberapa LFSR merupakan masukan ke multiplexer, dan ssatu multiplexer mengontrol keluar multiplexer. Misalkan digunakan 3 generator LFSR dan a_i, b_i dan c_i adalah keluaran dari ketiga generator tersebut. Maka keluaran Generator Geffe adalah :

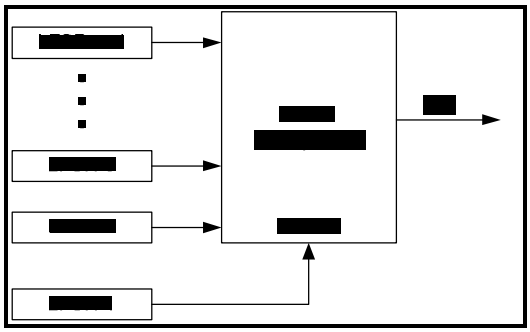
$$\mathbf{z_i = (a_i \text{ and } b_i) \text{ xor } ((\text{not } b_i) \text{ and } c_i)}$$

Jika LFSR memiliki panjang n_1, n_2 dan n_3 berturut-turut, maka kompleksitas linier generatornya adalah :

$$\mathbf{(n_1 + 1).n_2 + n_1.n_3}$$

Periode generator ini paling sedikit kelipatan bersama dari periode masing-masing generator. Meskipun kelihatan sangat aman, ternyata menurut ilmu kriptografi, generator ini lemah melawan *correlation attack*. *Correlation attack* mencari hubungan antara keluaran generator (z_i) dengan keluar subgenerator (a_i, b_i, c_i) yang terdapat di dalamnya. Dengan mengamati urutan keluaran z_i, maka dapat diperkirakan nilai keluaran subgeneratornya (a_i, b_i, c_i). Dari attack ini diperoleh bahwa keluaran generator sama dengan keluaran LFSR-2 yaitu keluaran b_i dalam 75 % waktunya. Artinya perbedaan antara z_i dan b_i hanya sekitar 25 %. Kemudian dari urutan keluaran b_i dapat dianalisis susunan umpan balik generator LFSR-2 serta menerka nilai awal isi LFSR-2 dan menjalankan generator keseluruhan. Bila 75 % waktu yang digunakan menunjukkan bahwa b_i sesuai dengan z_i, maka terkaan LFSR-2 telah benar.

Selain memilih di antara dua buah LFSR, generator ini dapat memilih di antara k LFSR, selama k adalah hasil pemangkatan dari 2. Di dalam generator ini terdapat k + 1 LFSR total. LFSR-1 harus di-*clock* $\log_2 k$ kali lebih cepat dari k LFSR lain.

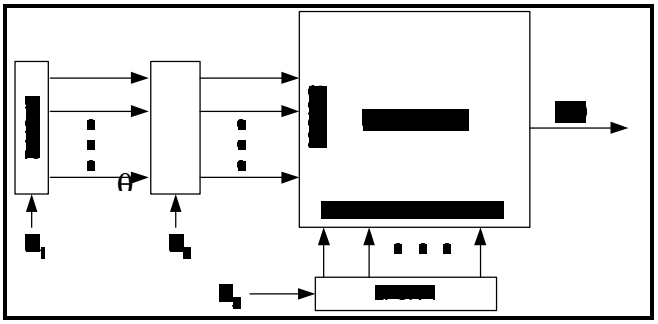


Gambar 2.3 Sketsa gambar Geffe Generator

Walaupun generator ini kelihatan lebih kompleks daripada *Geffe Generator,* namun jenis *correlation attack* yang sama dapat memecahkannya.

2.4.2 Jennings Generator

Generator ini menggunakan sebuah multiplexer untuk mengkombinasikan dua buah LFSR. *Multiplexer* dikontrol oleh LFSR-1, memilih 1 bit dari LFSR-2 untuk setiap *output* bit. Selain itu, juga terdapat satu fungsi untuk memetakan *output* dari LFSR-2 ke *input* dari *multiplexer*.[8]



Gambar 2.4 Sketsa gambar Jennings Generator [8]

Kuncinya terletak pada keadaan awal (*initial state*) dari dua buah LFSR dan fungsi pemetaannya. Walaupun generator ini memiliki properti statistik yang luar biasa, namun generator ini dapat dipecahkan oleh Ross Anderson dengan menggunakan *meet-in-the-middle consistency attack* dan *linear consistency attack*.

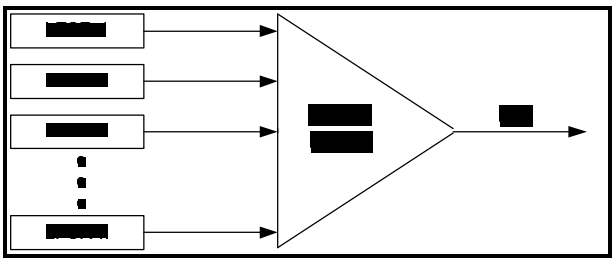
2.4.3 Threshold Generator

Generator ini mencoba untuk mengambil konsep seputar masalah sekuritas dari generator terdahulu dengan menggunakan sejumlah variasi dari LFSR. Teorinya adalah jika menggunakan lebih banyak LFSR, maka akan lebih sulit untuk memecahkan *cipher*-nya.

Pastikan bahwa panjang dari semua LFSR adalah bilangan prima relatif dan semua polinomial *feedback* adalah primitif untuk memaksimalkan periodenya. Jika lebih dari setengah bit *output* adalah 1, maka ouput dari generator adalah 1. Jika lebih dari setengah bit *output* adalah 0, maka output dari generator adalah 0.

Dengan tiga buah LFSR, output dari generator dapat ditulis sebagai berikut :

$$b = (a_1 \wedge a_2) \oplus (a_1 \wedge a_3) \oplus (a_2 \wedge a_3)$$



Gambar 2.5 Sketsa gambar Threshold Generator

Ini sangat mirip dengan *Geffe Generator*, kecuali generator ini memiliki kompleksitas linier yang lebih besar :

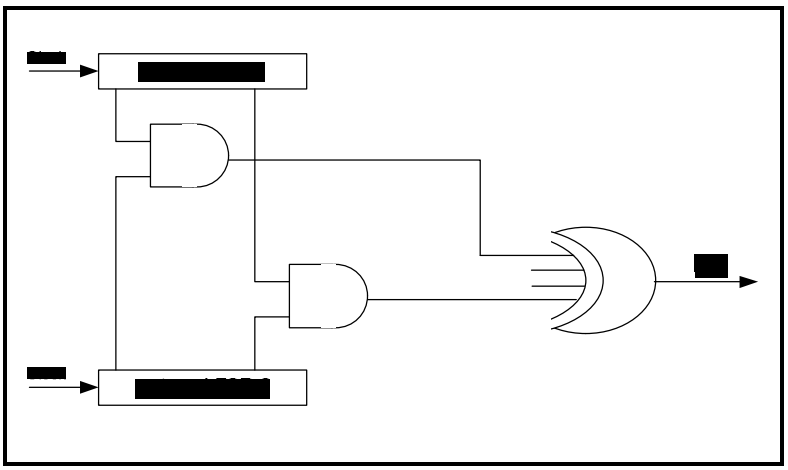
$$\mathbf{n_1.n_2 + n_2.n_3 + n_2.n_3}$$

di mana n_1, n_2 dan n_3 adalah panjang dari LFSR pertama, kedua dan ketiga.

Generator ini tidak terlalu hebat. Setiap bit *output* dari generator masih memiliki beberapa informasi mengenai keadaan dari LFSR dan dapat dipecahkan dengan *correlation attack*.[8]

2.4.4 Multispeed Inner-Product Generator

Generator ini diperkenalkan oleh Massey dan Rueppel, dengan menggunakan dua buah LFSR yang di-*clock* pada dua kecepatan yang berbeda. LFSR-2 di-*clock* d kali secepat LFSR-1. Bit individu dari kedua LFSR di-ANDkan bersama-sama dan kemudian di-XORkan satu sama lain untuk menghasilkan bit *output* terakhir dari generator.



Gambar 2.6 Sketsa gambar Multispeed inner-product Generator

Walaupun generator ini memiliki kompleksitas linier yang tinggi dan memproses properti statistik yang sempurna, namun tetap dapat dipecahkan oleh sebuah *linear consistency attack* [8]. Jika n_1 adalah panjang dari LFSR-1, n_2

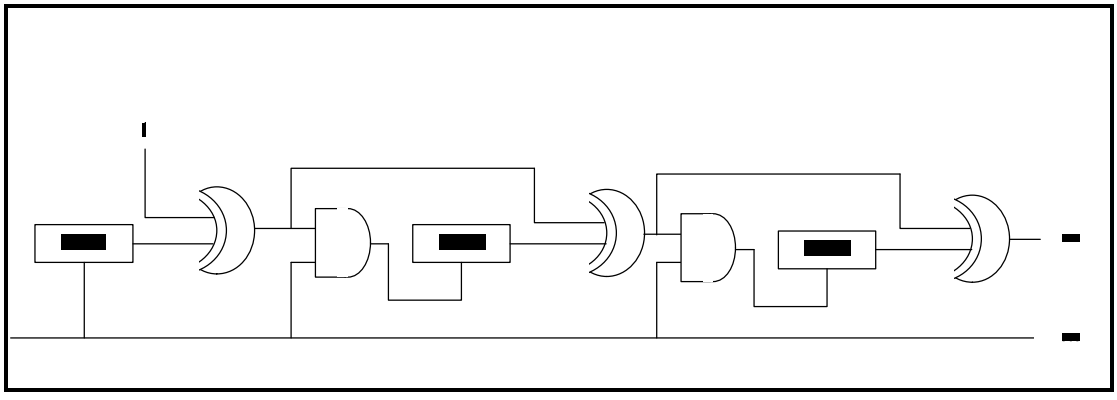
adalah panjang dari LFSR-2 dan d adalah perkalian kecepatan di antara keduanya, maka keadaan internal dari generator dapat dikembalikan dari sekuens *output* dengan panjang :

$$\mathbf{n_1 + n_2 + \log_2 d}$$

2.4.5 Gollmann Cascade Generator

Gollmann cascade adalah sebuah versi yang lebih tangguh dari *stop-and-go generator*. Generator ini terdiri dari sebuah seri dari LFSR, dengan *clock* dari setiap generator dikontrol oleh LFSR sebelumnya. Jika *output* dari LFSR-1 adalah 1 pada waktu t – 1, maka LFSR-2 di-*clock*. Jika *output* dari LFSR-2 adalah 1 pada waktu t – 1, maka LFSR-2 di-*clock*, dan seterusnya. *Output* dari LFSR terakhir adalah *output* dari generator. Jika semua LFSR memiliki panjang yang sama, n, kompleksitas linier dari sistem dengan k LFSR adalah :

$$\mathbf{n(2^n - 1)^{k-1}}$$



Gambar 2.7 Sketsa gambar Gollmann Cascade

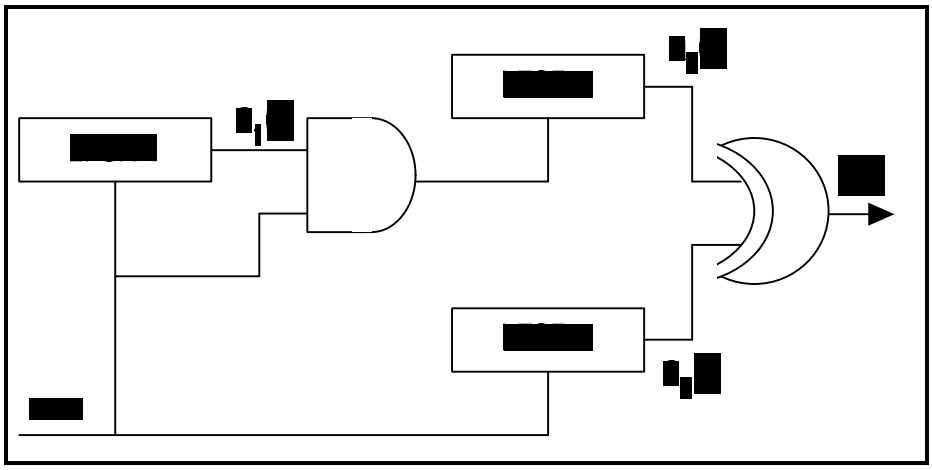
Cascade (bertindih) adalah sebuah ide yang bagus. Secara konsep, hal ini sangat sederhana dan dapat digunakan untuk menghasilkan sekuens dengan periode yang besar, kompleksitas linier yang besar, dan properti statistik yang

bagus. Generator ini rentan terhadap serangan yang dinamakan *lock-in*. Ini merupakan sebuah teknik di mana seorang kriptanalis merekonstruksikan *input* ke *shift register* terakhir dalam *cascade*, kemudian melakukan proses untuk memecahkan *cascade* secara *register* demi *register*. Ini merupakan sebuah masalah serius dalam beberapa situasi dan melemahkan panjang kunci efektif dari algoritma, tetapi pencegahan dapat dilakukan untuk mengurangi serangan.

Analisis lebih lanjut mengindikasikan bahwa sekuens mendekati *random* (acak) ketika k besar. Berdasarkan serangan akhir-akhir ini pada Gollmann cascade yang pendek, direkomendasikan untuk menggunakan k dengan panjang minimal 15. Lebih baik menggunakan banyak LFSR yang pendek daripada sedikit LFSR yang panjang [8].

2.4.6 Beth-Piper Stop-and-Go Generator

Generator Beth Piper Step and Go ini dikembangkan oleh Thomas Beth dan Fred Piper pada tahun 80-an yang namanya kemudian diabadikan sebagai nama generator tersebut. Generator ini menggunakan konfigurasi 3 buah LFSR yang digabungkan dengan suatu fungsi logika AND dan XOR. Generator ini menggunakan *output* dari satu LFSR untuk mengontrol *clock* dari LFSR lain. *Clock input* dari LFSR-2 dikontrol oleh *output* dari LFSR-1, sehinga LFSR-2 dapat berubah keadaannya pada waktu t hanya jika *output* dari LFSR-1 adalah bit 1 pada waktu t – 1.



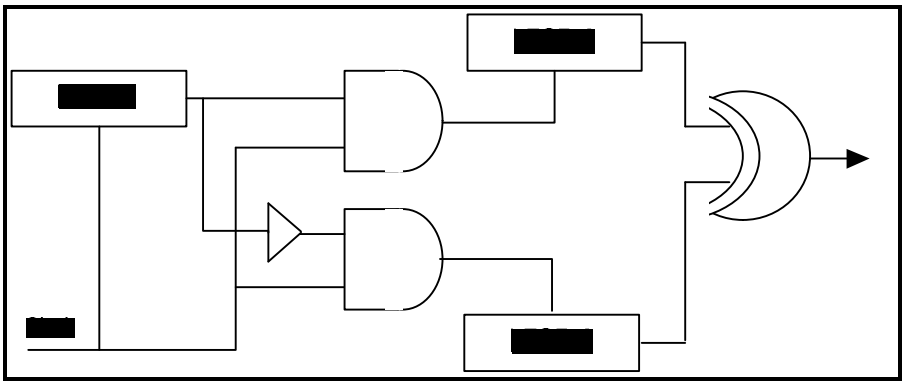
Gambar 2.8 Sketsa gambar Beth-Piper Stop-and-Go Generator

Tidak ada orang yang mampu untuk membuktikan hasil mengenai kompleksitas linier generator ini dalam kasus umum. Namun, generator ini dapat dipecahkan oleh *correlation attack* [8].

2.4.7 Alternating Stop-and-Go Generator

Generator ini menggunakan tiga buah LFSR dengan panjang yang berbeda. LFSR-2 di-*clock* ketika *output* dari LFSR-1 adalah 1, LFSR-3 di-*clock* ketika *output* dari LFSR-1 adalah 0. *Output* dari generator ini adalah hasil XOR dari LFSR-2 dan LFSR-3 [8].

Generator ini memiliki sebuah periode yang panjang dan kompleksitas linier yang besar.

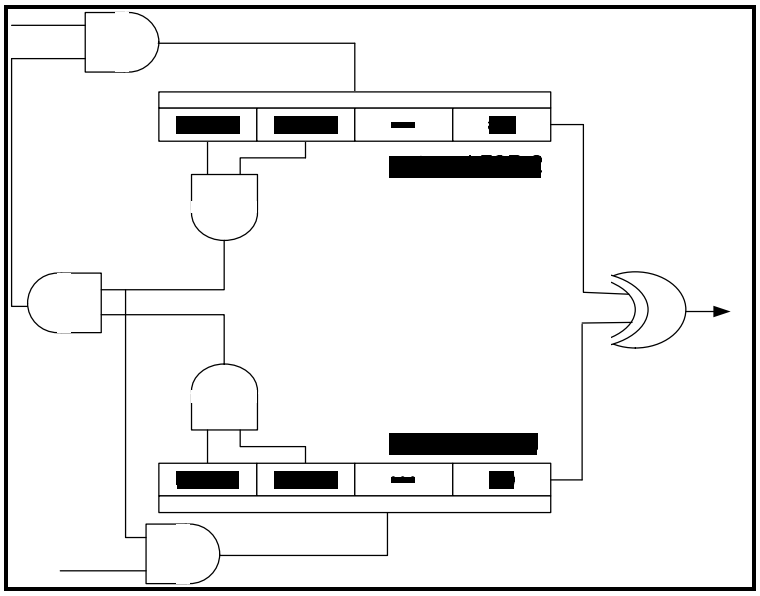


Gambar 2.9 Sketsa gambar Alternating Stop-and-Go Generator

2.4.8 Bilateral Stop-and-Go Generator

Generator ini menggunakan dua buah LFSR, keduanya dengan panjang n. Output dari generator adalah hasil XOR dari *output* dari setiap LFSR. Jika *output* dari LFSR-2 pada waktu t – 1 adalah 0 dan *output* pada waktu t – 2 adalah 1, maka LFSR-2 tidak di-*clock* pada waktu t. Kebalikannya, jika *output* dari LFSR-1 pada waktu t – 1 adalah 0 dan *output* pada t – 2 adalah 1 dan jika LFSR-1 di-*clock* pada waktu t, maka LFSR-2 tidak di-*clock* pada waktu t.

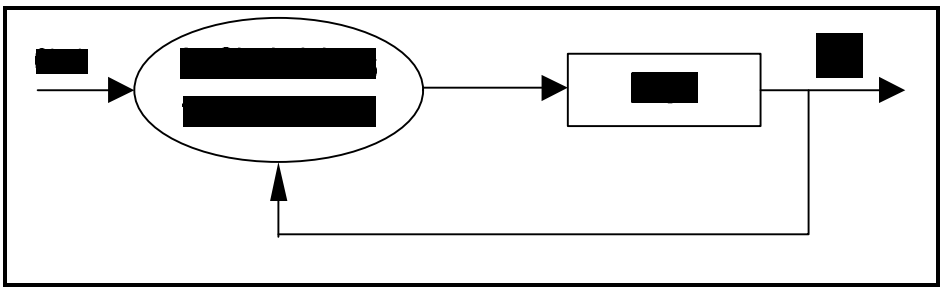
Kompleksitas linier dari generator ini jika ditinjau secara kasar sama dengan periodenya [8].



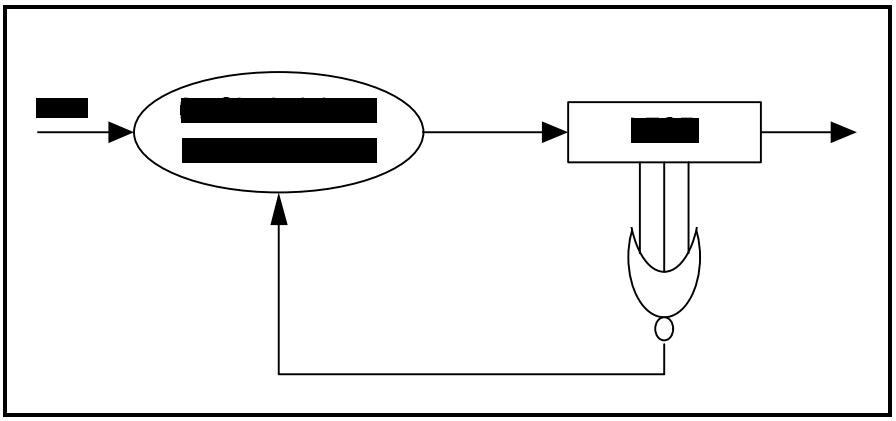
Gambar 2.10 Sketsa gambar Bilateral Stop-and-Go Generator

2.4.9 Self-Decimated Generators

Self-decimated generators adalah generator-generator yang mengontrol clock mereka sendiri. Generator ini telah dikemukakan dua versi yaitu oleh Rainer Rueppel dan oleh Bill Chambers dan Dieter Gollmann. Dalam *generator Rueppel*, ketika *output* dari LFSR adalah 0, maka LFSR di-*clock* d kali. Ketika *output* dari LFSR adalah 1, maka LFSR di-*clock* k kali. Generator Chambers dan Gollmann lebih rumit, tetapi idenya tetap sama. Namun, sayang sekali kedua generator tidak aman, walaupun beberapa modifikasi telah dilakukan yang diklaim dapat mengoreksi masalah [8].



Gambar 2.11 Sketsa gambar Rueppel self decimated Generator



Gambar 2.12 Sketsa gambar Chambers's and Gollmann's self decimated Generator

2.5 Operator Logika

Operasi-operasi logika dasar terdiri dari operasi *NOT*, operasi *AND*, dan operasi *OR*. Operasi logika lain merupakan kombinasi dari operasi - operasi logika dasar ini. Salah satu operasi logika hasil kombinasi operasi logika dasar adalah operasi *XOR* (*Exclusive-OR*).

2.5.1 Operasi NOT

Berbeda dengan operasi logika yang lain, operasi *NOT* hanya mempunyai satu *input* dan satu *output*. *Output* dari operasi logika *NOT* memiliki nilai logika yang berlawanan dengan nilai logika *input*. Apabila inputnya berlogika "1", maka outputnya berlogika "0", demikian juga sebaliknya. Oleh karena itu, *gate* dari operasi ini disebut sebagai *inverter*. Dalam sistem digital, operasi logika *NOT* dari suatu *input* A akan dilambangkan dengan A.

Tabel kebenaran dari operasi *NOT* dapat dinyatakan sebagai berikut,

Tabel 2.1 Tabel kebenaran operasi NOT

A	$\overline{\mathbf{A}}$
0	1
1	0

Keterangan :

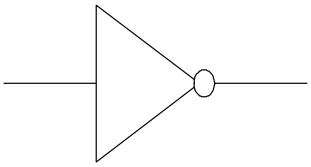
A : *input* operasi logika *NOT*.

$\overline{A}$: *output* operasi logika *NOT*.

1 : nilai logika benar (*true*).

0 : nilai logika salah (*false*).

Gambar *gate* logika dari operasi *NOT* adalah sebagai berikut,



Gambar 2.13 Gate logika dari operasi NOT

2.5.2 Operasi AND

Operasi *AND* mempunyai dua *input* dan satu *output*. Operasi *AND* dari dua *input* A dan B hanya akan bernilai logika “1” apabila kedua *input* A dan B bernilai logika “1”. Atau dengan kata lain output dari operasi *AND* akan memiliki nilai logika “0” apabila salah satu inputnya bernilai logika “0”. Dalam sistem digital, operasi logika *AND* dari *input* A dan B akan dilambangkan dengan A . B atau biasanya disingkat menjadi AB.

Tabel kebenaran dari operasi *AND* dapat dinyatakan sebagai berikut,

Tabel 2.2 Tabel kebenaran operasi AND

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

Keterangan :

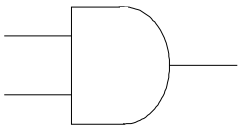
A, B : *input* operasi logika *AND*.

AB : *output* operasi logika *AND*.

1 : nilai logika benar (*true*).

0 : nilai logika salah (*false*).

Gambar *gate* logika dari operasi *AND* adalah sebagai berikut,



Gambar 2.14 Gate logika dari operasi AND

2.5.3 Operasi OR

Operasi *OR* mempunyai dua *input* dan satu *output*. Operasi *OR* dari dua *input* A dan B hanya akan bernilai logika “0” apabila kedua *input* A dan B bernilai logika “0”. Atau dengan kata lain output dari operasi *OR* akan memiliki nilai logika “1” apabila salah satu inputnya bernilai logika “1”. Dalam sistem digital, operasi logika *OR* dari *input* A dan B akan dilambangkan dengan A + B.

Tabel kebenaran dari operasi *OR* dapat dinyatakan sebagai berikut,

Tabel 2.3 Tabel kebenaran operasi OR

p	**q**	**p ∨ q**
0	0	0
0	1	1
1	0	1
1	1	1

Keterangan :

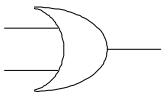
A, B : *input* operasi logika *OR*.

A + B : *output* operasi logika *OR*.

1 : nilai logika benar (*true*).

0 : nilai logika salah (*false*).

Gambar *gate* logika dari operasi *OR* adalah sebagai berikut,



Gambar 2.15 Gate logika dari operasi OR

2.5.4 Operasi XOR

Operasi *XOR* merupakan kombinasi dari operasi-operasi logika dasar. Operasi *XOR* mempunyai dua *input* dan satu *output*. Operasi *XOR* dari dua *input* A dan B hanya akan bernilai logika "0" apabila kedua *input* A dan B bernilai logika sama, yaitu "0" atau "1". Atau dengan kata lain output dari operasi *XOR* akan memiliki nilai logika "1" apabila kedua inputnya bernilai logika berbeda. Dalam sistem digital, operasi logika *XOR* dari *input* A dan B akan dilambangkan dengan $A \oplus B$. Operasi *XOR* dapat dinyatakan seperti berikut,

$$A \oplus B = \overline{A}B + A\overline{B}$$

Tabel kebenaran dari operasi *XOR* dapat dinyatakan sebagai berikut,

Tabel 2.4 Tabel kebenaran operasi XOR

A	**B**	**A ⊕ B**
0	0	0
0	1	1
1	0	1
1	1	0

Keterangan :

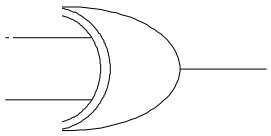
A, B : *input* operasi logika *XOR*.

$A \oplus B$: *output* operasi logika *XOR*.

1 : nilai logika benar (*true*).

0 : nilai logika salah (*false*).

Gambar *gate* logika dari operasi *XOR* adalah sebagai berikut,



Gambar 2.16 Gate logika dari operasi XOR

2.6 Landasan Matematis Kriptografi

2.6.1 Pergeseran Bit (Shift)

Pergeseran *bit* (*Shift*) adalah operasi pergeseran terhadap suatu barisan *bit* sebanyak yang diinginkan. *Bit* kosong yang telah tergeser akan diberikan nilai *bit* "0" (nol). Operasi pergeseran terbagi menjadi dua macam yaitu :

1. Operasi Geser Kiri (*Shift Left*) yaitu operasi yang menggeser (*shift*) sejumlah *bit* ke kiri (*left*) dengan nilai *bit* "0" (nol). Operasi *shift left* dilambangkan dengan "**<<**". Contoh operasi *shift left* :

   ```
   11000110 << 1 : 10001100
   11000110 << 2 : 00011000
   11000110 << 3 : 00110000
   ```

2. Operasi Geser Kanan (*Shift Right*) yaitu operasi yang menggeser (*shift*) sejumlah *bit* ke kanan (*right*) dengan nilai *bit* "0" (nol). Operasi *shift right* dilambangkan dengan "**>>**". Contoh operasi *shift right* :

   ```
   11000110 >> 1 : 01100011
   11000110 >> 2 : 00110001
   ```

2.6.2 Rotasi Bit (Rotate)

Rotasi *bit* (*Rotate*) adalah operasi perputaran terhadap suatu barisan *bit* sebanyak yang diinginkan. *Bit* yang tergeser akan dipindahkan ke sisi barisan *bit* yang berlawanan dengan arah putaran *bit*. Operasi rotasi terbagi atas dua macam yaitu :

1. Operasi Rotasi Kiri (*Rotate Left*) yaitu operasi memutar barisan *bit* ke kiri sebanyak nilai yang diberikan secara per *bit*, *bit* kosong yang telah tergeser di sebelah kanan akan digantikan dengan *bit* yang telah tergeser di sebelah kirinya. Operasi *rotate left* dilambangkan dengan "**<<<**". Contoh operasi *rotate left* :

   ```
   11000110 <<< 1 : 10001101
   11000110 <<< 2 : 00011011
   11000110 <<< 3 : 00110110
   ```

2. Operasi Rotasi Kanan (*Rotate Right*) yaitu operasi memutar barisan *bit* ke kanan sebanyak nilai yang diberikan secara per *bit*, *bit* kosong yang telah tergeser di sebelah kiri akan digantikan dengan *bit* yang telah tergeser di sebelah kanannya. Operasi *rotate right* dilambangkan dengan "**>>>**".Contoh operasi *rotate right* :

   ```
   11000110 >>> 1 : 01100011
   11000110 >>> 2 : 10110001
   11000110 >>> 3 : 11011000
   ```

2.7 Stream Cipher

Stream cipher adalah jenis algoritma enkripsi simetri dan dapat dibuat sangat cepat sekali, jauh lebih cepat dibandingkan dengan algoritma *block cipher* manapun. Algoritma *block cipher* secara umum digunakan untuk unit *plaintext* yang besar sedangkan *stream cipher* digunakan untuk blok data yang lebih kecil, biasanya ukuran bit. Proses enkripsi terhadap *plaintext* tertentu dengan algoritma *block cipher* akan menghasilkan *ciphertext* yang sama jika kunci yang sama digunakan. Dengan *stream cipher,* transformasi dari unit *plaintext* yang lebih kecil ini berbeda antara satu dengan lainnya, tergantung pada kapan unit tersebut ditemukan selama proses enkripsi. [3]

Satu *stream cipher* menghasilkan apa yang disebut suatu *keystream* (suatu barisan bit yang digunakan sebagai kunci). Proses enkripsi dicapai dengan menggabungkan *keystream* dengan *plaintext* biasanya dengan operasi bitwise XOR. Pembentukan *keystream* dapat dibuat independen terhadap *plaintext* dan *ciphertext,* menghasilkan apa disebut dengan *synchronous stream cipher*, atau dapat dibuat tergantung pada data dan enkripsinya, dalam hal mana *stream cipher* disebut sebagai *self-synchronizin*g. Kebanyakan bentuk *stream cipher* adalah *synchronous stream cipher*s.

Konsentrasi dalam *stream ciphers* pada umumnya berkaitan dengan sifat-sifat teoritis yang menarik dari *one-time pad*. Suatu *one-time pad,* kadang-kadang disebut *Vernam cipher,* menggunakan sebuah string dari bit yang dihasilkan murni secara random. *Keystream* memiliki panjang sama dengan pesan *plaintext;* string random digabungkan dengan menggunakan bitwise XOR dengan *plaintext* untuk

menghasilkan *ciphertext*. Karena *keystream* seluruhnya adalah random, walaupun dengan sumber daya komputasi tak terbatas seseorang hanya dapat menduga *plaintext* jika dia melihat *ciphertext*. Metode *cipher* seperti ini disebut memberikan kerahasiaan yang sempurna (*perfect secrecy*).

Metode *stream cipher* yang umum digunakan adalah RC4. Satu hal yang menarik bahwa mode operasi tertentu dari suatu *block cipher* dapat men-transformasikan secara efektif hasil operasi tersebut ke dalam satu *keystream generator* dan dalam hal ini, *block cipher* apa saja dapat digunakan sebagai suatu *stream cipher*; seperti dalam DES, CFB atau OFB. Akan tetapi, *stream ciphers* dengan desain khusus biasanya jauh lebih cepat.

2.8 Perangkat Lunak Pembelajaran

Seiring dengan perkembangan peradaban manusia dan kemajuan pesat di bidang teknologi, tanpa disadari komputer telah ikut berperan dalam dunia pendidikan terutama penggunaannya sebagai alat bantu pengajaran. Percobaan penggunaan komputer untuk proses belajar dimulai di Amerika Serikat pada akhir tahun 1950-an dan awal tahun 1960-an. Kemudian penelitian selanjutnya dilakukan oleh *Harvard University* bekerja sama dengan *IBM* pada tahun 1965. Setelah munculnya komputer mikro, sistem pengajaran dengan komputer menjadi semakin meluas pada pengembangan aplikasi perangkat lunak ajar yang dikenal dengan istilah perangkat lunak pembelajaran.

Banyak istilah yang dipakai untuk menyatakan perangkat lunak pembelajaran dengan komputer, seperti *Computer Assisted Instruction* (CAI), *Computer Based Instruction* (CBI), *Computer Based Education* (CBE), *Computer Assisted Learning* (CAL), atau *Computer Based Training* (CBT).

2.9 Pemograman Visual Basic

Visual basic adalah salah satu *development tool* untuk membangun aplikasi dalam lingkungan Windows. Dalam pengembangan aplikasi, visual basic menggunakan pendekatan visual dalam bentuk form, sedangkan bahasa pemograman untuk kodingnya adalah *basic* yang cenderung mudah untuk dipelajari [4]. Pada pemograman visual, pengembangan aplikasi dimulai dengan pembentukan *user interface*, yang kemudian kita atur properti dari objek - objek yang digunakan dalam user interface setelah itu baru dilakukan penulisan kode program untuk menangani kejadian - kejadian (event). Tahap pengembangan aplikasi ini dikenal dengan pendekatan *Bottom Up.*

2.9.1 IDE Visual Basic

Langkah awal dari belajar visual basic adalah mengenal IDE (*Integrated Development Environment*) Visual basic yang merupakan lingkungan pengembangan terpadu bagi programer dalam mengembangkan aplikasinya. Dengan menggunakan IDE programmer dapat membuat user *interface*, melakukan *coding*, melakukan *testing* dan *debuging* serta mengkompilasi program menjadi *executable* [4].

2.9.2 Jendela IDE pada Visul Basic

IDE visual basic 6 menggunakan model MDI (Multiple Document Interface), adapun jendela - jendela yang ada pada dan perlu kita perhatikan dalam menjalankan program visual basic ini secara umum adalah :

- Menu Baru, digunakan untuk memilih tugas – tugas tertentu seperti menyimpan project, membuka project, dll.
- Main Toolbar, dilakukan untuk melakukan tugas – tugas tertentu dengan cepat.
- Jendela Project, jendela yang berisikan gambaran semua modul yang terdapat dalam aplikasi yang kita kerjakan. Anda dapat menggunakan Icon Toggle Folder untuk menampilkan modul – modul dalam jendela tersebut secara group atau berurut berdasarkan nama.
- Jendela form designer Jendela yang berisi komponen – komponen yang dapat digunakan untuk mengembangkan user interface.
- Jendela Code, merupakan tempat bagi code untuk menuliskan koding dari aplikasi yang kita buat.
- Jendela properties, merupakan daftar properti – properti object yang sedang terpilih.
- Jendela form Layout, jendela bantu yang berfungsi menunjukan bagaimana form bersangkutan akan ditampilkan ketika *runtime*.