Penggunaan *Artificial Neural Network* pada Pembangkit Bilangan Acak Semu serta Perbandingannya dengan Algoritma lain

Novan Parmonangan Simanjuntak (13509034)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509034@stei.itb.ac.id

***Abstract*— Pembangkit bilangan acak semu merupakan salah satu komponen yang sangat penting di bidang kriptografi. Sudah banyak algoritma serta pendekatan pembangkitan bilangan acak semu yang dibuat. Rancangan dari algoritma pembangkit ini sangat menentukan kualitas bilangan acak semu yang dihasilkan. Pada makalah ini penulis akan membahas apa itu barisan bilangan acak dan bagaimana menentukan kualitas dari sebuah barisan bilangan acak. Selain itu penulis juga akan membahas mengenai pengujian bilangan acak semu yang standar. Pengetesan ini berguna untuk menentukan seberapa bagus sebuah barisan bilangan acak yang dihasilkan. Algoritma umum yang sudah digunakan saat ini menggunakan konsep teori bilangan seperti *linear congruential generator, linear feedback shift register, lagged fibonacci* dan lain-lain. Pada proposal ini penulis melakukan analisis algoritma pembangkit bilangan acak semu dengan pendekatan *ANN* atau Jaringan Saraf Buatan. Adapun pendekatan algoritma yang digunakan berdasarkan HNN (Hopfield Neural Network). Hal ini dikarenakan property unik dari HNN, yaitu perilaku yang tidak dapat diprediksi dalam kondisi tertentu. Penulis akan membandingkan barisan bilangan acak yang dihasilkan oleh HNN dengan barisan bilangan acak yang dihasilkan oleh algoritma lain yang standar atau tidak menggunakan *ANN*.**

Kata kunci — acak, pembangkit bilangan acak semu, pengujian bilangan acak semu, ANN, HNN, perbandingan barisan bilangan acak semu

1. PENDAHULUAN

Pembangkit bilangan acak mempunyai peran yang sangat penting di banyak bidang keilmuan, mulai dari sains, statistik, dan salah satunya adalah di bidang kriptografi. Dalam aplikasi di dunia nyata, digunakan pembangkit bilangan acak semua untuk menghasilkan data acak. Hal ini dikarenakan bilangan acak dihasilkan oleh komputer yang mempunyai sifat deterministik, yang berarti bahwa barisan bilangan yang dihasilkan diperoleh dari sebuah nilai awal atau disebut juga *seed*.

Berbagai macam metode digunakan untuk menghasilkan barisan bilangan acak semu seperti linear congruential generator, linear feedback shift register, lagged fibonacci. Metode ini menggunakan sifat teori bilangan dalam menentukan barisan bilangan acak semu. Barisan bilangan acak semua juga dapat dihasilkan dengan menggunakan jaringan saraf buatan. Hal ini dikarenakan sifat jaringan saraf buatan yang tidak linear.

Makalah ini dibagi menjadi enam bagian. Bagian pertama membahasa pendahuluan. Bagian kedua membahasa dasar teori. Adapun pada bagian kedua dijelaskan mengenai bilangan acak, pembangkit bilangan acak semu dan berbagai algoritma yang sudah ada, pembangkit bilangan acak semua dengan *ANN* serta pengujian dari barisan bilangan acak semu. Bagian ketiga adalah implementasi dari dasar teori pada bagian kedua. Bagian keempat adalah analisis. Bagian kelima adalah kesimpulan. Bagian keenam adalah saran.

2. DASAR TEORI

2.1 Definisi Bilangan Acak

Banyak filsuf yang mendiskusikan untuk mendefinisikan bilangan acak. Sebuah bilangan acak dapat diinterpretasikan sebagaik hasil dari pelemparan sebuah koin dengan label 0 dan 1. Setiap pelemparan koin mempunyai kemungkinan sekitar 0.5 untuk menghasilkan 0 atau 1. Di bawah ini merupakan definisi formal dari bilangan acak untuk distribusi biner.

Misalkan $X_1X_2...X_n$ adalah barisan variabel acak, di mana n=1, 2, 3, ...
X_i adalah variabel acak biner, artinya nilai yang mungkin dari X_i adalah 0 dan 1 dan $P(X_i = 1) = 0.5$ atau nilai distribusi dari semua barisan adalah 0.5^n, di mana terdapat 2^N kemungkinan.

Terdapat banyak distribusi bilangan acak, salah satunya biner. Pada makalah ini barisan bilangan acak yang dipakai adalah distribusi biner, artinya hanya terdapat angka 0 atau 1 pada barisan bilangan acak.

2.2 Aplikasi Bilangan Acak di bidang Kriptografi

Pembangkit bilangan acak memegang peran penting dalam bidang keamanan komputer. Dengan meningkatnya permintaan akan amannya komunikasi, teknik-teknik enkripsi data telah berkembang dengan sangat pesat. Data-data yang disimpan semakin sensitif dan berharga dan metode komunikasi sudah sangat bervariasi, seperti penggunakan jaringan nirkabel.

Proses menjadikan data biasa (*plainteks*) menjadi sesuatu yang tidak bisa dibaca (*cipherteks*) disebut eknripsi. Dekripsi adalah proses sebaliknya. Kunci adalah parameter rahasia yang digunakan oleh algoritma untuk mengontrol output dari algoritma enkripsi atau dekripsi. Selama tahun-tahun terakhir banyak algoritma kriptografi seperti DES dan AES. Algoritma ini bersifat publik dan telah dipelajari di seluruh dunia dan sangat sedikit sekali ada laporan mengenai serangan pada algoritma ini. Kunci yang digunakan pada algoritma ini direpresentasikan sebagai barisan bit. Ukuran kunci bervariansi dari ratusan ke ribuan tergantung pada tipe algoritma. Yang harus menjadi pusat perhatian di sini adalah pembangkitan dan pendistribusian kunci. Karena kunci sangat penting dan tidak akan ditebak oleh seseorang atau program komputer, kunci dipilih secara acak. Jika kunci yang dipilih tidak benar-benar acak, dengan melihat barisan bit yang membentuk kunci, pola dapatditemukan dan alhasil kunci akhir bisa ditebak. Dengan kata lain performansi dari algoritma kriptografi sangat terkait dengan performansi dari bilangan acak yang dihasilkan [4][5].

2.3 Pembangkit Bilangan Acak Semu

PRNG (*Pseudo Random Number Generator*) atau pembangkit bilangan acak semua adalah algoritma yang membangkitkan barisan bilangan. *PRNG* dibagi menjadi 2 jenis, yaitu :

1. *Deterministic PRNG*.
   Barisan bilangan acak dihitung dengan menggunakan vektor awal, disebut juga *seed* [5]. Beberapa algoritma PRNG memperoleh *seed* dari pengguna, ada juga yang memperoleh *seed* dari perangakat keras sistem, misalnya bit dari waktu sekarang.
2. *Nondeterministic PRNG*.
   Barisan bilangan acak ditentukan oleh sesuatu yang tidak bisa diprediksi, misalnya noise dari suatu signal akibat gangguan hardware.

Pada makalah ini, algoritma-algoritma yang digunakan adalah *Deterministic PRNG*, artinya ada *seed* yang diberikan ke algoritma. Digunakan juga tiga algoritma *Deterministic PRNG* standar yang sudah banyak digunakan, misalnya saja di fungsi random di suatu bahasa pemrograman. Tiga algoritma tersebut adalah *linear congruential generator*, *linear feedback shift register*.

Deterministic PRNG dirancang untuk sedekat mungkin menghasilkan barisan bilangan acak yang ideal. Tabel 1 menunjukan perbandingan antara fitur dari pembangkit bilangan acak ideal dan pembangkit bilangan acak semu.

Tabel 1
Perbandingan fitur Pembangkit bilangan acak ideal dan semu

RNG ideal	PRNG
tidak ada *periodicity*	ada *periodicity*
output tidak bisa diprediksi	output bisa diprediksi
tidak ada dependensi antar bilangan dalam barisan	ada dependensi antar bilangan dalam barisan
level keamanan tinggi	level keamanan tidak setinggi rng ideal
tidak berbasis algoritma dan *seed*	berbasis algoritma dan *seed*
cepat dan efisien	cepat dan efisien
cost rendah	cost rendah
reproducibility tidak ada	*reproducibility* ada

Pada Tabel 1, *periodicity* berarti terdapat pola yang berulang pada barisan bilangan acak, sedangkan *reproducibility* berarti barisan bilangan acak yang dihasilkan bisa muncul pada hasil pembangkitan barisan bilangan acak selanjutnya. Perlu diperhatikan perbedaan kriteria antara pembangkit bilangan acak ideal dan pembangkit bilangan acak semua.

2.3.1 Linear congruential generator

Algoritma ini merupakan salah satu dari algoritma tertua dan yang paling dikenal. Algoritmanya didefinisikan sebagai berikut [6] :

$$X_{n+1} \equiv (aX_n + c) \pmod{m}$$

Keterangan :

X adalah barisan bilangan acak semu
m, m > 0 , adalah modulus
a, 0 < a < m, adalah multiplier
c, 0 ≤ c < m, adalah increment atau shift
X_0, 0 ≤ X_0 < m, adalah *seed*

Nilai a, c, dan m ditentukan dari pengguna. Nilai a dipilih sedemikian sehingga algoritma pembangkit mempunyai periode maksimal. Pada komputer dengan ukuran integer sebesar 32 bit, dipilih $m = 2^{32}$ dan a = 65539. Nilai lain dari a yang telah terbukti memberikan hasil yang bagus adalah 1099087573, 2396548189, 2824527309, 3934873077, dan 230458073.

Selain itu nilai m diisi dengan bilangan prima. Pada unix, pembangkit bilangan acak semua drand48 menggunakan nilai a = 25214903917, c = 11, $m = 2^{48}$. Fungsi acak ini juga diimplementasikan sebagai fungsi gsl_rng_uniform di library GNU.

2.3.2 Linear feedback shift register

Pada algoritma ini, dihasilkan barisan bit acak. Bit ke-i dihitung dari p bit sebelumnya menggunakan relasi rekurens. Misalnya $b_i \equiv (a_i\ b_{i-1} + \ldots + a_d\ b_{i-p}) \bmod 2$, di mana nilai a_1, a_2, … a_d adalah 0 atau 1. Fungsi rekurens ini dapat dilakukan oleh hardware melalui shift-register. Barisan bits dapat dishift ke kiri atau ke kanan . b_i dihitung dengan menggunakan operasi XOR. Hasil akhir

fungsi rekurens adalah fungsi *feedback polynomial* dari bit sebelumnya. Berikut adalah persamaan akhir LFSR :

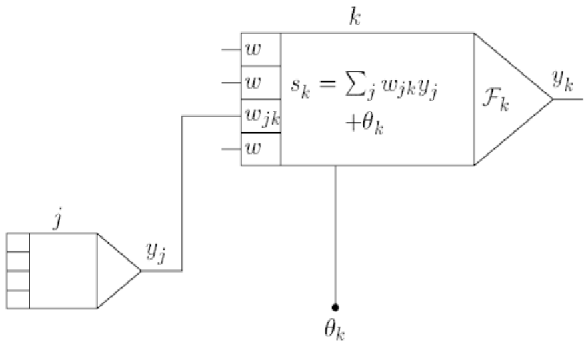
$$b_i = a_{i-1} b_{i-1}^{n} + a_{i-2} b_{i-1}^{n-1} + \ldots + a_1 1$$

Di mana a_{i-1} , a_{i-2} , .. , a_1 bernilai 0 atau 1. Untuk persamaan di atas feedback polynomialnya adalah :

$$a_{i-1} x^n + a_{i-2} x^{n-1} + \ldots + a_1$$

Fungsi random() pada Unix (c atau Fortran) diimplementaskan dengan menggunakan FSR dengan *feedback polynomial* berikut :

$$x^{81} + x^8 + 1$$



Gambar 1. Komponen dari Jaringan Saraf Tiruan

2.4 *Artificial Neural Network*

Artificial Neural Network merupakan model dari simulasi otak manusia, dalam hal ini neuron. Otak menggunakan *neuron* untuk melakukan perhitungan yang jauh lebih cepat daripada komputer digital tercepat saat ini. Selain itu otak sangatlah kompleks, bersifat *nonlinear* dan mampu memproses informasi secara pararel. Jaringan saraf tiruan dibuat untuk memodelkan otak melakukan suatu tugas.

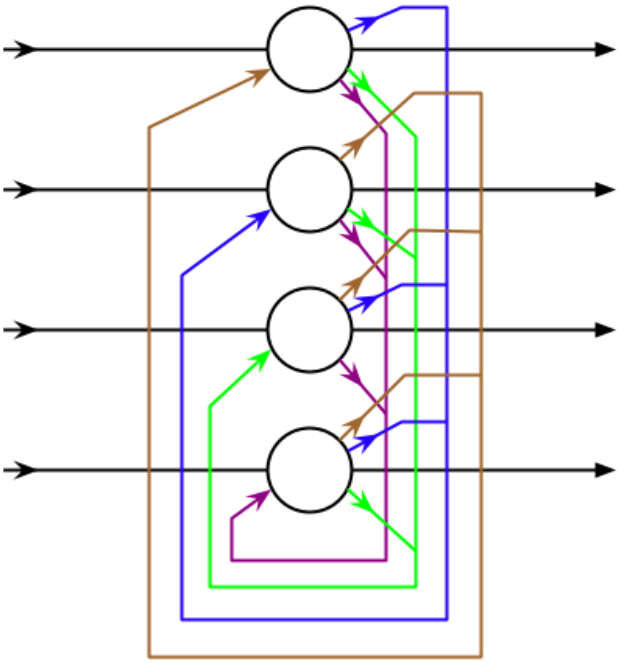
Jaringan saraf tiruan sendiri terdiri dari banyak *neuron* yang saling berkomunikasi dengan mengirimkan sinyal satu denga lainnya melalui *link*/koneksi. Komponen dari jaringan saraf tiruan antara lain:

1. Sekumpulan neuron.
2. Nilai aktivasi y_k untuk setiap neuron, y_k adalah output dari neuron.
3. Koneksi atau *link* antar unit. Setiap koneksi mempunyai nilai bobot w_{jk} yang menentukan efek dari sinyal yang dikirimkan neuron j ke neuron k.
4. *Propagation rule*, yang menentukan nilai input total s_k dari semua input eksternal.
5. Fungsi aktivasi F_k yang ditentukan berdasarkan nilai sk(t) dan yk(t).
6. Offset untuk input eksternal, θk untuk setiap neuron.
7. Aturan learning untuk setiap pasangan input dan output. Aturan ini digunakan untuk menentukan nilai bobot pada *link*.

Gambar 1 menjelaskan komponen dari jaringan saraf tiruan.

2.5 *HNN (Hopfield Neural Network)*

HNN adalah jaringan saraf buatan berulang yang ditemukan oleh John Hopfield. Jaringan saraf buatan berulang adalah jaringan saraf di mana setiap neuron menerima input atau *feedback* dari neuron yang lainnya. Gambar 2 menjelaskan topologi dari jaringan saraf buatan berulang.



Gambar 2. Topologi dari jaringan saraf buatan berulang

Properti penting pada *HNN* adalah ketika proses optimisasi(proses learning dilakukan untuk menghasilkan suatu nilai minimum atau optimum) dilakukan, keluaran dari network tidak dapat diprediksi ketika prosesnya tidak konvergen. Artinya di sini perlu dicari bagaimana agar ketika proses optimisasi atau iterasi dilakukan, output atau keluaran dari *HNN* dijamin tidak konvergen [1] [2].

Kemampuan untuk konvergen pada *HNN* dipengaruhi oleh beberapa hal, yaitu :

1. Topologi *HNN*.
2. Kondisi awal jaringan atau *seed*.
3. Aturan learning.

Cara yang bisa ditempuh agar keluaran *HNN* tidak konvergen adalah sebagai berikut :

1. Memberikan *seed* matriks bobot yang tidak simetris.
2. Memperbolehkan dua atau lebih neuron aktif secara bersamaan.
3. Menggunakan jaringan yang berukuran besar (jumlah neuronnya banyak dan koneksinya kompleks).
4. Menggunakan fungsi aktivasi yang tidak linear, pada umumnya digunakan fungsi tanh(x), di mana x adalah nilai input total s_k dari semua input eksternal.

2.6 Penerapan *HNN* dalam Pembangkit Bilangan Acak Semu

Dari hasil *output* tiap neuron, dapat digunakan menjadi sumber barisan bilangan acak. Misalkan hasilnya dinyatakan ke dalam barisan bit, maka dapat diambil bagian dari barisan bit tersebut untuk dijadikan bit pada barisan bit acak semu.

2.7 Pengujian Barisan Bilangan Acak Semu

Ada banyak metode pengujian barisan bilangan acak semu. Salah satu metode yang paling banyak digunakan adalah dengan menggunakan statistik. Di bawah ini beberapa jenis pengujian yang dilakukan pada makalah ini. Perlu diingatkan bahwa barisan bilangan acak yang dihasilkan adalah distribusi biner, artinya hanya ada bilangan 0 atau 1 pada barisan bilangan.

2.7.1 *Frequency Test*

Fokus utama dari pengujian ini adalah perbandingan antara 0 dan 1 pada keseluruhan barisan. Mula-mula jumlah setiap bilangan di mana 0 dihitung sebagai -1. Kemudian dibagi dengan akar dari panjang barisan. Perlu diperhatikan bahwa semakin mendekati 0 nilai *frequency test* dari sebuah pengujian, maka semakin acak barisan tersebut.

2.7.2 *Frequency Test within a block*

Pengujian ini seperti pada pengujian sebelumnya, tetapi dilakukan pada blok berukuran M-bit, di mana M adalah panjang dari blok. Pengujian ini menentukan apakah proporsi 1 memenuhi proporsi yang diharapkan atau dengan kata lain frekuensinya mendekati M/2. Perlu diperhatikan semakin kecil nilai pengujian, maka semakin acak barisan tersebut.

2.7.3 *Cumulative Sums (Cusum) Test*

Pengujian ini dilakukan dengan menjumlahkan tiap bilangan, di mana 0 dianggap sebagai -1. Semakin mendekati 0 maka semakin acak barisan tersebut. Hasil yang besar menunjukkan bahwa barisan tersebut tidak acak.

2.7.4 *Entropy*

Entropy menunjukkan kerapatan informasi di file, satuannya adalah bit per karakter. Pada barisan yang sangat rapat, maka dianggap semakin acak. Hal ini karena file yang tidak rapat dapat dikompresi untuk mengurangi ukurannya.

2.7.5 *Chi-Square Test*

Chi-Square Test adalah pengujian yang sangat sering digunakan untuk pembangkit bilangan acak, hal ini dikarenakan pengujian ini sangat sensitif terhadap kesalahan pada *PRNG*. Persentasi dari hasil *Chi-Square Test* menunjukkan derajat tidak acak.

2.7.6 *Arithmetic Mean*

Arithmetic Mean adalah pengujian untuk blok. Tiap bilangan interpretasi dari hasil satu blok (bukan bit) dijumlahkan dan dibagi dengan panjang file. Barisan semakin acak jika mendekati nilai rataan ini.

2.7.7 *Monte Carlo Value for Pi*

Pengujian dilakukan pada level byte, di mana barisan dibagi menjadi blok-blok berukuran delapan bit. Tiap enam byte yang berurutan dibuat menjadi 24 bit persegi X dan Y pada perhitungan Mont Carlo untuk nilai Pi. Jika hasilnya semakin mendekati nilai Pi, maka semakin acak. Begitu juga sebaliknya.

2.7.8 *Serial Correlation Coefficient*

Pengujian ini dilakukan pada level byte. Tujuan dari pengujian ini adalah mengukur seberapa tergantung sebuah byte pada suatu barisan terhadap byte sebelumnya. Jika nilai pengujian semakin mendekati nol, maka semakin acak barisan tersebut.

3. IMPLEMENTASI

Implementasi dilakukan dengan menggunakan bahasa Java untuk semua program yang dibuat. Digunakan platform Java SE-7 dan Netbeans sebagai IDE. Terdapat 2 modul utama, yaitu :

1. Modul *PRNG*.
   Modul ini adalah implementasi dari algoritma pembangkit bilangan acak semu yang sudah dijelaskan sebelumnya. Input untuk modul ini adalah parameter untuk setiap algoritma (seperti seed). Output dari modul ini adalah barisan 10 ribu bilangan acak pada distrbusi biner, dengan kata lain barisan 10 ribu bit acak
2. Modul Pengujian.
   Modul ini berisi implementasi dari setiap pengujian yang sudah dijelaskan. Input dari modul ini adalah barisan 10 ribu bit acak dari modul *PRNG*. Outputnya adalah nilai dari tiap pengujian.

3.1 Modul *PRNG*

Modul ini dibagi menjadi tiga bagian, yaitu modul untuk *linear congruential generator*, *linear feedback shift register*, dan *HNN*.

3.1.1 Implementasi *linear congruential generator*

Algoritma *linear congruential generator* diimplementasikan persis seperti penjelasan pada dasar teori dengan parameter input pada Tabel 2.

Tabel 2
Parameter algoritma *PRNG linear congruential generator*

m	**a**	**c**	***X_0* / *seed***
2^{48}	25214903917	11	1

Parameter di atas dipilih karena sudah terbukti memberikan barisan bilangan yang sangat acak.

Pseudo-code untuk algoritma ini adalah sebagai berikut :

```
Program linear congruential generator
a,c,m,i : integer
X : array of integer
R : array of bit
---
a = 25214903917, c = 11, m = 2^48;
X[0] = 1
for (int i = 1; i<=10000; i++)
   X[i] = (a*(X[i]-1) + b) % m;
   if X[i] mod 2 == 0
      R[i] = 0
   else
      R[i] = 1
```

3.1.2 Implementasi *linear feedback shift register*

Algoritma *linear feedback shift register* diimplementasikan persis seperti penjelasan pada dasar teori dengan parameter input pada Tabel 3.

Tabel 3
Parameter algoritma *PRNG linear feedback shift register*

Persamaan *feedback polynomial*	***X_0* / *seed***
$x^{32} + x^{22} + x^2 + 1$	1

Parameter di atas dipilih karena sudah terbukti memberikan barisan bilangan yang sangat acak.

Pseudo-code untuk algoritma ini adalah sebagai berikut :

```
Program linear feedback shift register
i : integer
X : array of integer
R : array of bit
---
X[0] = 1
for (int i = 1; i<=10000; i++)
   X[i] = X[i-1]^32 + X[i-1]^22 + X[i-1]^2 + 1
   if X[i] mod 2 == 0
      R[i] = 0
   else
      R[i] = 1
```

3.1.3 Implementasi *Hopfield Neural Network*

Algoritma *Hopfield Neural Network* diimplementasikan persis seperti penjelasan pada dasar teori dengan parameter input pada Tabel 4. Digunakan parameter yang mirip seperti [1].

Tabel 4
Parameter algoritma *PRNG linear feedback shift register*

Fungsi aktivasi	**Jumlah neuron**	***X_0* / *seed***
tanh(x)	100	Matriks bobot tidak simetris

Parameter di atas dipilih karena sudah terbukti memberikan barisan bilangan yang sangat acak [1]. Parameter yang berubah adalah *seed*, di sini *seed* dibuat sendiri oleh penulis, karena tidak dijelaskan pada [1]. Kriteria matriksnya adalah diagonal sangat positif, matriks segitiga atas mengandung bilangan positif, dan matriks segitiga bawah mengandung bilangan negatif.

3.2 Modul Pengujian

Modul ini dibagi menjadi delapan bagian, yaitu modul Frequency Test, modul *Frequency Test within a block*, modul *Cumulative Sums (Cusum) Test*, modul *Entropy*, *modul Chi-Square Test*, modul *Arithmetic Mean*, *Monte Carlo Value for Pi*, modul *Serial Correlation Coefficient*. Adapun penulis hanya mengimplementasikan tiga modul pertama. Lima modul sisanya diambil dari *library* pengujian barisan bilangan acak semu dari [8]. Modul yang akan dijelaskan juga merupakan tiga modul pertama.

3.2.1 Implementasi *frequency test*

Algoritma *frequency test* diimplementasikan persis seperti penjelasan pada dasar teori.

Pseudo-code untuk algoritma ini adalah sebagai berikut :

```
Program frequency test
i : integer
pnrg : array of bit
result : double
---
result = 0.0;
for(int i=0;i<prng.length();++i) {
if (prng.bitAt(i) == '0') {
    result += -1;
}else {
    result += 1;
}
}
result = abs(result);
result = result /
(sqrt(prng.length()));
```

3.2.2 Implementasi *frequency test within a block*

Algoritma *frequency test within a block* diimplementasikan persis seperti penjelasan pada dasar teor. Paramater input berupa panjang tiap blok adalah 8 bit. Artinya di sini satu blok berukuran satu byte.

Pseudo-code untuk algoritma ini adalah sebagai berikut :

```
Program frequency test within a block
i : integer
pnrg : array of bit
result : double
result = 0.0;
for(int i=0;i<prng.length();++i) {
   if (prng.bitAt(i) == '1') {
       result += 1;
   }
}
result = result / (prng.length()/2.0);
```

3.2.3 Implementasi *cumulative sums (cusum) test*

Algoritma *cumulative sums (cusum) test* diimplementasikan persis seperti penjelasan pada dasar teori.

Pseudo-code untuk algoritma ini adalah sebagai berikut :

```
Program cumulative sums (cusum) test
i : integer
pnrg : array of bit
result : double
result = 0.0;
for(int i=0;i<prng.length();++i) {
    if (prng.charAt(i) == '0') {
        result += -1;
    }else {
        result += 1;
    }
}
result = Math.abs(result);
```

4. PENGUJIAN DAN ANALISIS

Pengujian dilakukan di komputer penulis, di mana mesin komputernya mempunyai tujuh core prosesor. Implementasi code ditulis dalam bahasa Java dan C. Tabel 5 menggambarkan lingkungan pengujian yang digunakan.

Tabel 5
Lingkungan pengujian

Parameter	Nilai
Sistem Operasi	Windows 7 Home Premium
JDK Version	1.7
CPU	Core i7 1.79GHz
Memori	4GB, 7Mb 3 level cache

4.1 Data set barisan bilangan acak semu

Terdapat tiga buah data set barisan bilangan acak semu yang dari masing-masing algoritma yang diimplementasikan, berikut hasil barisan bilangan acak semu dari tiap algoritma. Tiap barisan terdiri dari 10000 bit.

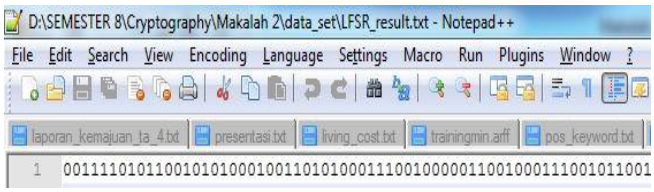
4.1.1 Hasil *linear congruential generator*

Gambar 3 menunjukkan hasil barisan bilangan acak semua oleh algoritma *linear congruential generator*.

Gambar 3. Hasil barisan bilangan acak semu algoritma *linear congruential*

4.1.2 Hasil *linear feedback shift register*

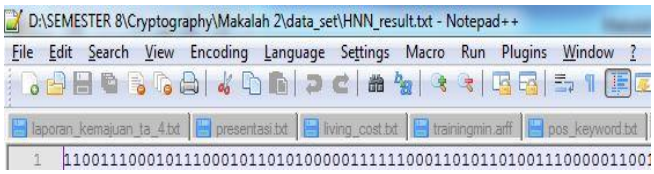
Gambar 4 menunjukkan hasil barisan bilangan acak semua oleh algoritma *linear feedback shift register*.



Gambar 4. Hasil barisan bilangan acak semu algoritma *linear feedback shift register*

4.1.3 Hasil *Hopfield Neural Network*

Gambar 5 menunjukkan hasil barisan bilangan acak semua oleh algoritma *Hopfield Neural Network*.



Gambar 5. Hasil barisan bilangan acak semu algoritma *Hopfield Neural Network*

4.2 Hasil pengujian pembangkit barisan bilangan acak semu

Tabel 6 menunjukkan hasil pengujian untuk ketiga barisan bilangan acak semu. Terdapat delapan buah pengujian untuk tiap barisan, sehingga terdapat 24 buah hasil pengujian.

Tabel 6
Hasil seluruh pengujian untuk tiap pembangkit barisan bilangan acak semu

prng test	Barisan hasil *linear congruential generator*	Barisan hasil *linear feedback shift register*	Barisan hasil *hopfield neural network*
frequency test	1.3198680197967005	1.9198080287952006	0.8799120131978003
frequency test within a block	0.9864027194561088	1.0187962407518496	0.9908018396320736
Cumulative Sums (Cusum) Test	132.0	192.0	88.0
Entropy	1.002616 bits per byte	1.002485	1.002687
Chi-Square Test	0.01 %	0.01%	0.01%
Arithmetic Mean	48.4859 (127.5 = random)	48.5021 (127.5 = random)	48.4881 (127.5 = random)
Monte Carlo Value for Pi	error 27.32 %	error 27.32 %	error 27.32 %
Serial Correlation Coefficient	0.250837	0.267900	0.253208

4.3 Analisis hasil pengujian

Pada tabel 6 dapat dilihat untuk *frequency test* didapat *HNN* memberikan hasil yang terlihat. Pada *frequency test within a block* didapat *linear congruential generator* memberikan hasil terbaik (paling kecil), sedangkan *HNN* berada pada posisi kedua. Pada *Cumulative Sums (Cusum) Test* didapat *HNN* memberikan hasil terbaik (paling kecil). Pada *Entropy* didapat *HNN* memberikan hasil terbaik (paling pada densitasnya). *Chi-Square Test* memberikan nilai galat yang sama untuk semua algoritma. Pada *Arithmetic Mean*, *linear feedback shift register* memberikan hasil terbaik (mendekati rataan ekspektasi 127.5), sedangkan *HNN* berada pada posisi kedua. *Monte Carlo Value for Pi* memberikan nilai galat yang sama untuk semua algoritma. Pada *Serial Correlation Coefficient*, *linear congruential generator* memberikan hasil terbaik (paling mendekati 0), sedangkan *HNN* berada pada posisi kedua.

Dari hasil pengujian, dapat dilihat *HNN* memberikan hasil yang lebih baik dibandingkan dua algoritma lainnya yang menggunakan teori bilangan. Hal ini dikarenakan sifat *HNN* yang lebih bisa menghasilkan nilai konvergensi dibandingkan dengan algoritma dengan teori bilangan.

5. KESIMPULAN

Dari hasil pengujian ditunjukkan bahwa *HNN* memberikan hasil terbaik, di mana dari delapan buah pengujian, *HNN* berada di peringkat satu pada tiga buah pengujian, peringkat dua pada tiga buah pengujian, dan seri pada dua buah pengujian. Kemudian disusul dengan *linear congruential generator* yang dua kali berada di peringkat satu. Peringkat terakhir adalah *linear feedback shift register*.

HNN dapat digunakan untuk membangkitkan barisan bilangan acak semu untuk aplikasi kriptografi. Hal ini terbukti dari hasil percobaan pada tiap pengujian barisan bilangan acak semu.

6. SARAN

Dalam menguji dan membandingkan algoritma pembangkit barisan bilangan acak semu dengan *HNN*, selain dengan hanya menguji kualitas kerandoman suatu barisan, perlu juga dipertimbangkan mengenai biaya yang dipakai dalam membangkitkan barisan bilangan acak semu. Ada 2 biaya, yaitu biaya ruang (jumlah memori yang dihabiskan) dan biaya waktu (waktu yang dihabiskan untuk membangkitkan suatu barisan bilangan acak semu). Untuk kedepannya perlu dilakukan pengetesan untuk kedua biaya ini.

REFERENSI

[1] Tirdad, K. (2010). Hopfield neural networks as pseudo random number generators. *Fuzzy Information Processing Society (NAFIPS), 2010 Annual Meeting of the North American*, (pp. 1-6).A. Klimov, “Analysis of Neural Cryptography”.

[2] Wang, Y.-H. (2006). Pseudo Random Number Generator Based on Hopfield Neural Network . *Machine Learning and Cybernetics, 2006 International* , (pp. 2810 - 2813).

[3] *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. (2013, May 18). Retrieved from http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf

[4] V. Gujral, “Cryptography using Artificial Neural Networks”. D. E. Eastlake, S. D. Crocker and J. I. Shiller, "RFC 1750: Randomness recommendation for security," *Internet Society Request for Comments, Internet Engineering Task Force*, December 1994.

[5] J. von Neumann, "Various Technique Used in Connection with Random Digits," *in Applied Math*

Series, notes by G. E. Forsythe, National Bureau of Standards, vol. 12, pp. 36–38.
[6] S.K. Park and K.W. Miller, "Random Number Generators: Good Ones Are Hard To Find,"*Communications of the ACM*, pp. 1192-1201, vol.31, 1998.
[7] *SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES*. (2013, May 19). Retrieved from http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
[8] *A Pseudorandom Number Sequence Test Program*. (2013, May 19). Retrieved from http://www.fourmilab.ch/random/

PERNYATAAN

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

Bandung, 23 Maret 2013

Novan Parmonangan Simanjuntak
13509034