Analisis Penerapan Berbagai Chaotic Map sebagai Pembangkit Bilangan Acak Semu

Danny Andrianto 13510011
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
danny.andrianto@students.itb.ac.id 13510011@std.stei.itb.ac.id

***Abstract*—Chaos theory sangat cocok untuk diterapkan sebagai pembangkit bilangan acak karena kesensitifannya terhadap input yang masuk. Mapping yang memiliki karakteristik chaos disebut juga dengan Chaotic Map. Pada makalah ini, penulis akan membuat algoritma pembangkit bilangan acak dengan berdasarkan kepada chaotic map-chaotic map yang sudah ada. Selain itu, penulis juga akan melakukan analisis terhadap tiap algoritma unutk menentukan apakah oembangkit bilangan acak yang dihasilkan aman secara kriptografi atau tidak.**

***Index Terms*—pembangkit bilangan acak semu, PRNG, CSPRNG, chaotic map, uji statistic, sawtooth map, gauss iterated map, logistic map, tent map, circle map**

I. PENDAHULUAN

A. *Latar Belakang*

Dewasa ini, sebagian besar komunikasi dilakukan melalui jaringan, baik itu jaringan telepon, radio, internet, dan lainnya. Jaringan-jaringan tersebut terekspos ke dunia luar sehingga dibutuhkan suatu mekanisme pengamanan agar informasi yang dipertukarkan tidak dapat disadap oleh pihak ketiga. Mekanisme tersebut disebut dengan kriptografi dimana setiap pesan yang dipertukarkan dienkripsi agar tidak dapat dimengerti oleh pihak yang menyadapnya.

Kriptografi sendiri dalam pelaksanaannya sangat membutuhkan perhitungan matematis. Di samping itu, terutama untuk kriptografi kunci public butuh untuk menghasilkan suatu bilangan acak semu yang akan digunakan dalam rumus pengenkripsi/dekripian.

Untuk membuat penghasil bilangan acak semu yang baik, perlu diperhatikan bahwa deretan bilangan yang dihasilkan tidak boleh dapat diprediksi. Hal ini untuk memastikan kunci dari algoritma kriptografi yang digunakan tidak terbongkar karena algoritma pembangkit bilangan acak semu yang kurang kuat.

Salah satu teori yang dapat diimpementasikan sebagai pembangkit bilangan acak semu adalah chaos theory. Dengan mengimplememntasikan chaos theory, akan didapatkan bilangan acak yang benar-benar kacau polanya dan sangat sensitive terhadap perubahan parameter sedikit apapun. Fungsi-fungsi yang memenuhi chaos theory disebut sebagai chaos map.

B. *Rumusan Masalah*

Sehubungan dengan latar belakang yang telah penulis kemukakan di atas, penulis merumuskan masalah sebagai berikut:

a. Chaotic map apa yang paling baik diimplementasi sebagai pembangkit bilangan acak semu
b. Chaotic map apa yang tidak baik untuk diimplementasi sebagai pembangkit bilangan acak semu

C. *Batasan Masalah*

Pada pengerjaan makalah ini, penulis akan:

a. Membuat program yang mengimplemntasikan berbagai chaotic map sebagai pembangkit bilangan acak semu
b. Menganalisis deret bilangin hasil pembangkitan tiap algoritma dari program

D. *Tujuan*

Berdasarkan latar belakang dan perumusan masalah di atas, penulis makalah ini bertujuan untuk:

a. Mengetahui algoritma chaotic map yang paling baik digunakan sebagai pembangkit bilangan acak semu
b. Mengetahui algoritma chaotic map yang harus dihindari untuk digunakan sebagai pembangkit bilangan acak semu

II. DASAR TEORI

A. *Pseudo Random Number Generator (PRNG)*

Pembangkit bilangan acak semu (PRNG – Pseudo Random Number Generator) adalah suatu algoritma yang dapat menghasilkan deretan angka yang tidak dapat diprediksi. Bilangan acak yang dihasilkan menggunakan suatu algoritma ini dinamakan bilangan acak semu karena pembangkitan bilangannya dapat diulang kembali.

Di antara bermacam-macam algoritma PRNG ada beberapa algoritma yang dikategorikan aman untuk kriptografi (CSPRNG – Cryptographically Secure Pseudo Random Number Generator). Agar suatu PRNG dapat dikategorikan sebagai CSPRNG ada beberapa persyaratan yang harus dipenuhi, yaitu:

1. Secara statistik, ia mempunyai sifat-sifat yang

bagus (dites melalui uji keacakan statistik)
2. Tahan terhadap serangan serius yang mencoba untuk memprediksi bilangan acak yang dihasilkan.

B. Chaos Theory

Teori chaos adalah teori yang menggambarkan perilaku system dinamis nirlanjar yang menunjukkan fenomena yang kacau. Salah satu teori system chaos adalah sangat peka terhadap nilai awal. Hal ini menyebabkan system chaos akan menunjukkan hasil yang sangat kacau jika nilai awal berbeda sedikit saja.

Chaotic map adalah map dari suatu nilai ke nilai tertentu yang polanya sangat sensitif terhadap perubahan. Ada banyak chaotic map yang telah ditemukan dan beberapa contohnya adalah sawtooth map, gauss iterated map, logistic map, tent map, dan circle map.

C. Statistical Test

Tes dilakukan dengan mengambil salah satu sample output dari suatu pembangkit bilangan acak unutk kemudian dilakukan berbagai uji statistik. Tiap tes akan menentukan apakah suatu bilangan acak semu memiliki beberapa karakteristik yang kemungkinan besar dimiliki bilangan acak sejati. Jika suatu output gagal melewati salah satu tes yang diberikan, maka dapat disimpulkan bilangan yang dihasilkan mugkin tidak random.

Ada 5 uji statistik yang sering digunakan yaitu frequency test, serial test, poker test, runs test, dan autocorrelation test. Frequency test bertujuan untuk menentukan apakah kemunculan 0 dan 1 relatif tidak jauh berbeda. Serial test bertujuan untuk menentukan apakah kemunculan 00, 01, 10, dan 11 relatif tidak jauh berbeda. Poker test menentukan apakah kemunculan tiap subderet sepanjang m relatif tidak jauh berbeda. Runs test menetukan apakah kemunculan 0 maupun 1 yang beruntun wajar ada pada deret acak. Autocorrelation test bertujuan untuk mennetukan hubungan suatu deret dan hasil pergeseran deret tesebut.

C.1 Frequency text

$$X_1 = \frac{(n_0 - n_1)^2}{n}$$

n_0: jumlah bit 0 pada deret
n_1: jumlah bit 1 pada deret
n: panjang deret
mengikuti χ^2 distribution dengan derajat kebebasan 1 jika $n \geq 10$.

C.2 Serial Test

$$X_2 = \frac{4}{n-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n}(n_0^2 + n_1^2) + 1$$

n_{00}: jumlah kemunculan 00 pada deret
n_{01}: jumlah kemunculan 01 pada deret
n_{10}: jumlah kemunculan 10 pada deret
n_{11}: jumlah kemunculan 11 pada deret
n_0: jumlah bit 0 pada deret
n_1: jumlah bit 1 pada deret
n: panjang deret
mengikuti χ^2 distribution dengan derajat kebebasan 2 jika $n \geq 21$.

C.3 Poker Test

$$X_3 = \frac{2^m}{k}\left(\sum_{i=1}^{2^m} n_i^2\right) - k$$

m: bilangan bulat yang memenuhi $\left\lfloor \frac{n}{m} \right\rfloor \geq 5.(2^m)$
k: $\left\lfloor \frac{n}{m} \right\rfloor$
n_i: kemunculan tipe ke-i dari k potongan deret asli
n: panjang deret
mengikuti χ^2 distribution dengan derajat kebebasan sebesar $2^m - 1$

C.4 Runs Text

$$X_4 = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(G_i - e_i)^2}{e_i}$$

e_i: $(n - i + 3) / 2^{i+2}$
k: bilangan buat terbesar yang memnuhi $e_i \geq 5$
B_i: jumlah block(bit 1 beruntun) sepanjang i
G_i: jumlah gap(bit 0 beruntun) sepanjang i
n: panjang deret
mengikuti χ^2 distribution dengan derajat kebebasan sebesar 2k - 2

C.5 Autocorrelation Test

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

d: bilangan bulat yang memenuhi $1 \leq d \leq \lfloor n/2 \rfloor$
A(d): jumlah bit pada deret yang tidak sesuai dengan bit pada deret yang telah digeser sejauh d bit
n: panjang deret
mengikuti N(0, 1) distribution jika $n - d \geq 10$

χ^2 dan N(0, 1) distribution yang disebutkan di atas akan digunakan untuk menentukan apakah hasil uji sautu tes dapat dinyatakan lolos atau tidak.

III. CHAOTIC MAPS

Berikut adalah chaotic map-chaotic map yang akan penulis gunakan untuk membuat algoritma pembangkitan bilangan acak semu. Alasan pemilihan chaotic map-chaotic map ini adalah kesemuanya menghasilkan nilai yang diskrit, menghasilkan bilangan real, dan komputasinya 1 dimensi.

A. Sawtooth Map

Sawtooth map ditentukan dengan menggunakan rumus:

$$x_{n+1} = 2x_n \ (mod\ 1)$$

dimana x (mod 1) adalah komponen decimal dari x..

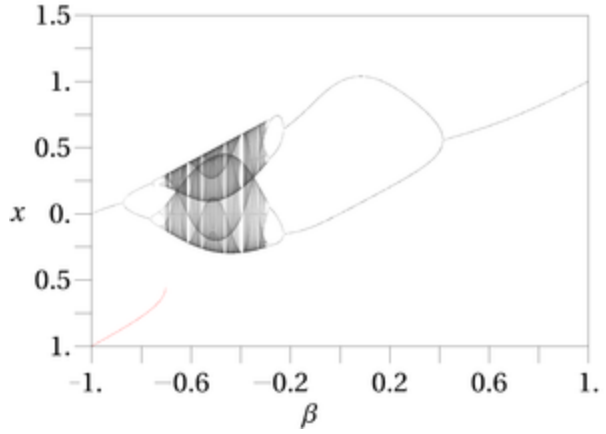
B. Gauss Iterated Map

Gauss map adalah map nonlinear yang memetakan bilangan real ke suatu interval bilangan real denagn menggunakan fungsi Gauss:

$$x_{n+1} = \exp(-\alpha x_n^2) + \beta,$$

dimana α dan β adalah parameter bernilai real.

Kekacauan Gauss Iterated Map dapat dilihat pada gambar berikut:



C. Logistic map

Persamaan logistic adalah model pertumbuhan populasi yang pertama akli dipublikasikan oleh Pierre Verhulst (1845, 1847). Model ini ditunjukkan dengan persamaan:

$$\frac{dN}{dt} = \frac{rN(K-N)}{K},$$

dimana r adalah parameter Malthusian (kecepatan pertumbuhan maksimum) dan K adalah populasi maksimum yang bisa ditunjang. Dengan membagi persamaan dengan K dan mendefinisikan $x \equiv N/K$, diapatkan rumus:

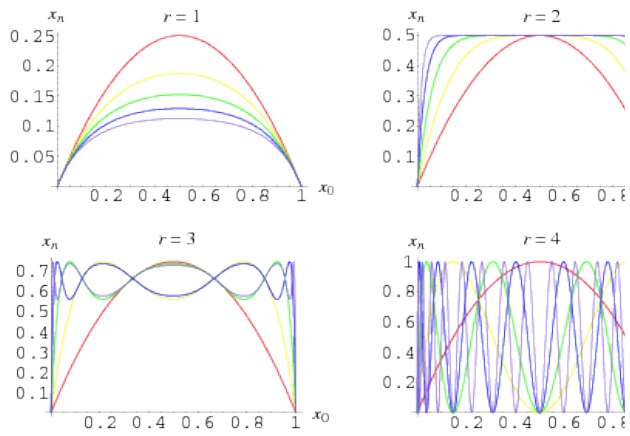
$$\frac{dx}{dt} = rx(1-x),$$

Persamaan di atas belum dapat digunakan sebagai penghasil bilangan acak semu, karena masih berupa persamaan kontinu. Logistic map sendiri merupakan versi diskrit dari persamaan tersebut yang didefiniskand engan rumus:

$$x_{n+1} = r x_n (1-x_n),$$

dimana r adalah sebuah konstanta positif.

Kekacauan logistic map dapat dilihat pada gambar berikut:



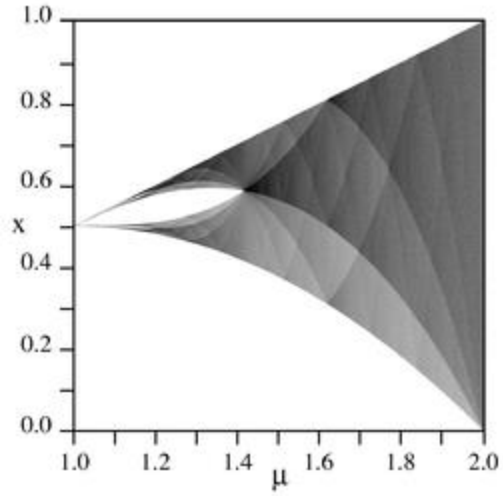
D. Tent Map

Tent map didefinsikan dengan rumus:

$$f_\mu := \mu \min\{x, 1-x\},$$

dengan μ adalah sebuah parameter berinlai real. Rumus diatas

Kekacauan tent map dapat dilihat pada gambar berikut:



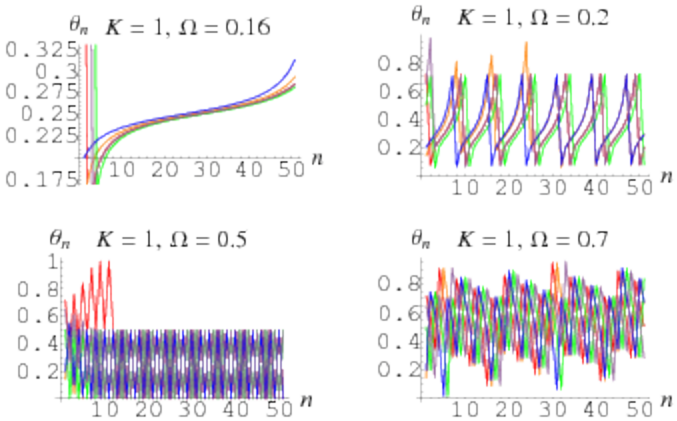
E. Circle Map

Circle map adalah map satu dimensi yang memtakan sebauh lingkaran ke dirinya sendiri menggunakan rumus:

$$\theta_{n+1} = \theta_n + \Omega - \frac{K}{2\pi}\sin(2\pi\theta_n),$$

Circle map memiliki 2 parameter, Ω dan K. Ω adalah frekuensi eksternal sedangkan K adalah ketinonlinearan.

Kekacauan circle map bsia dilihat pada gambar berikut:



IV. Implementasi

Implementasi dari ke 5 chaos map seperti yang telah penulis sebutkan membutuhkan sebuah umpan bilangan bulat bernilai lebih besar dari 0. Secara umum, berikut adalah gambaran algoritma CSPRNG yang digunakan penulis.

- Dapatkan nilai x_0
- Dapatkan barisan bilangan real acak dengan melakukan iterasi menggunakan chaos map:

  $$x_i = f(x_i - 1)$$

  i: iterasi ke / bilangan real acak ke
  f: fungsi chaos map
- Dapatkan bilangan bulat z_i dari nilai x_i

Algoritma CSPRNG akan menghasilkan barisan i buah

bilangan acak dengan tiap bilangan direpresentasikan dengan variabel z_i. Jadi untuk barisan 9 buah bilangan acak yang dihasilkan adalah $z_1, z_2, z_3, \ldots, z_9$.

Pertama untuk sebelum dapat menghitung z_1, sebelum menghitung z_1 dibutuhkan x_1, dan sebelumnya dibutuhkan x_0 yang bernilai antara 0 dan 1. Penentuan nilai x_0 unutk ke-5 chaos map sama yaitu dengan menggunakan umpan sebagai angka dibelakang koma. Jadi, jik umpan yang diberikan adalah 5824, x_0 bernilai 0,5824.

Karena beberapa chaos map memerlukan parameter tambahan, tahap berikutny akan dibagi unutk masing-masing map.

A. *Sawtooth Map*

Rumus sawtooth map tidak memiliki parameter tambahan, Map ini hanya membutuhkan umpan dan langsung siap dipakai. Hasilkan barisan bilangan real acak menggunakan rumus berikut:

$$x_i = 2x_{i-1} \; mod \; 1$$

B. *Gauss Iterated Map*

Gauss Iterated Map menggunakan fungsi Gauss yang membutuhkan 2 parameter tambahan yaitu α dan β. Tahap iterasi dibagi menjadi 2 tahap yaitu, menghasilkan nilai α dan β dan menghasilkan bilangan acak. Berikut adalah tahap-tahap tiap iterasi:

- Pilih bilangan real acak α yang bernilai antara 0 dan 10
- Pilih bilangan real acak β yang bernilai antara -10 dan 10
- Dapatkan bilangan real ke-i dengan rumus:

$$x_i = \exp(-\alpha x_{i-1}^2) + \beta$$

C. *Logistic Map*

Logistic map membutuhkan sebuah parameter tambahan yaitu bilanagm bulat r. Angka r ini akan berneda unutk tiap iterasi. Berikut adalah tahapan penghasil bilangan acak semu menggunakan logical map:

- Pilih bilangan bulat acak r antara 1 sampai 4
- Dapatkan bilangan real ke-I dengan rumus:

$$x_i = r x_{i-1}(1 - x_{i-1})$$

D. *Tent Map*

Tent map memerlukan sebuah bilangan real μ unutk melakukan mapping. Jadi, berikut adalah tahapan pembangkitan bilangan real acak dengan tent map:

- Pilih bilangan real acak μ yang bernilai antara 1 dan 3
- Dapatkan bilangan real ke-i dengan rumus:

$$x_i = \mu \min(x_{i-1}, 1 - x_{i-1})$$

E. *Circle Map*

Circle map memiliki 2 buah parameter yaitu Ω dan konstanta K. Pertama, tentukan dulu konstanta K yang ingin digunakan (biasanya 1). Ω sendiri adalah bilangan real yang akan berbeda unutk tiap iterasi. Lakukan iterasi seperti map-map sebelumnya:

- Pilih bilangan real acak Ω yang berniali antara 0 dan 1
- Dapatkan bilangan real ke-I dengan rumus:

$$x_i = x_{i-1} + \Omega - \frac{K}{2\pi}\sin(2\pi x_{i-1})$$

Setelah mendapatkan nilai x_i, kita bisa mendapatkan nilai z_i dengan menambahkan bilangan di kiri koma dengan bilanagn di kanan koma. Tetapi sebelum melakukan itu, ubahlah dahulu nilai x_i menjadi positif jika algoritma menghasilkan nilai negative. Contoh mendapatkan z_i dari x_i:

x_i = 1895166082.06145
z_i = 1895166082 + 6145 = 1895172227

x_i = -1388525910.59369
z_i = 1388525910 + 59369 = 1388585279

z_i juga bisa dibatasi dengan memodulokannay dengan batas maksimal yang kita inginkan. Misalnya jika z_i yang kita inginkan tidak boleh mencapai 100 maka bilangan yang tepat bsia didabatkan dengan z_i mod 100.

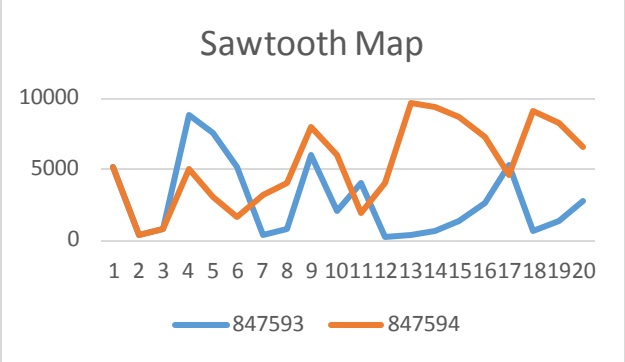
Untuk sawtooth map, sebelum z_i dapat digunakan perlu satu buah operasi lagi. Karena rumusnya yang sederhana, sering terjadi bilangan yang dimunculkan sawtooth map berbentuk seperti AB000000C atau D999999EFG. Ada kemungkinaan bilangan semacam itu dapat dilihat sebagai suatu pola, jadi tiap angka 0 atau 9 yang berulang harus diganti dengan 1 angka 0 atau 9. Jadi, AB000000C menjadi AB0C dan D999999EFG menjadi D9EFG.

Perlu diingat, kalau kita tidak perlu menggunakan nilai z_i seutuhnya. Kita tetap bsia menggunakan LSB dari z_i untuk membangkitkan bilangan acak per bitnya.

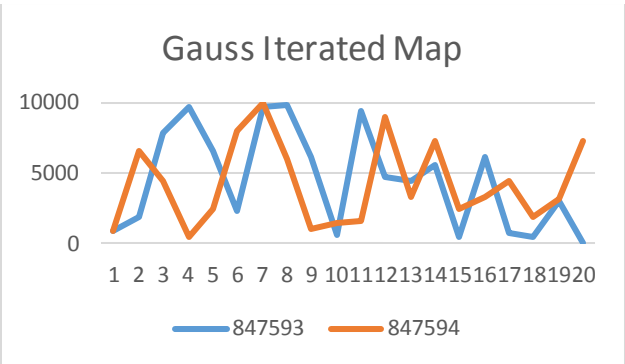
V. EKSPERIMEN

Eksperimen yang penulis lakukan adalah menguji kelima chaos map dengan mencoba menghasilkan sederet bilangan acak yang mana kelima algoritma diberikan umpan yang sama untuk melihat perbedaan perfroma masing-masing algoritma. Penulis membangkitkan bilangan acak dibatasi dengan range 0 hingga 10000 untuk melihat pola jika ada yang muncul pada tiap algoritma dan juga pembuktian chaos theory pada tiap algoritma. Penulis juga membangkitkan suatu deret bit sepanjang 128 buah untuk pada bagian berikutnya dapat dilakukan uji statistik.

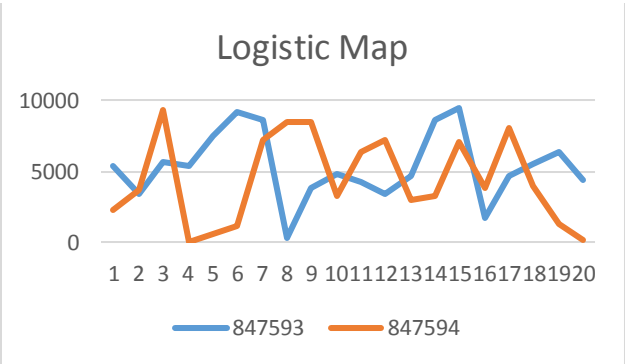
A. *Eksperimen I (umpan = 847593 dan 847594)*



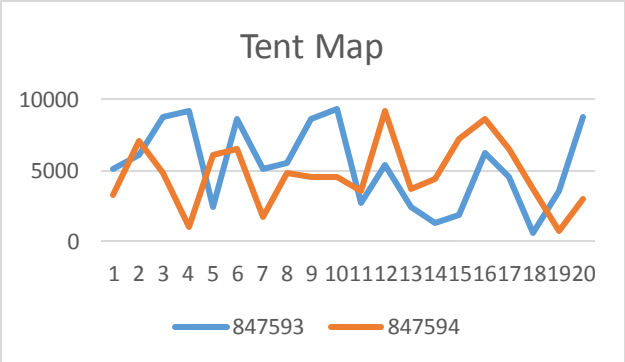
Lama eksekusi: 2.5 ms



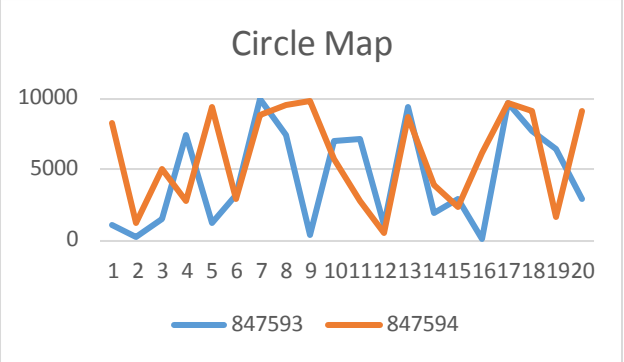
Lama eksekusi: 3 ms



Lama eksekusi: 3.5 ms



Lama eksekusi: 3 ms



Lama eksekusi: 4 ms

B. *Eksperimen II (deret bit)*

Umpan = 421

Hasil:

- Sawtooth map:
  00000100111000001011011101111001101000111
  11111111111100000000000000000000000000000
  00000000000000000000000000000000000000000
  00000
- Gauss Iterated Map:
  10100001000111001000001000011001110111111
  00110110000100011101000001111110000011100
  01110011011011100010111100010100001011100
  10000
- Logistic Map:
  10000111010001001110111111011001111100110
  10011111001111010001010111000101110101001
  11111111000001100011010111110111100111011
  10111
- Tent Map:
  10010101101000010110011010110101111110111
  10011111110000111011111011101111000100011
  10000111101100000001111011100010011000010
  00010
- Circle Map:
  01111110001000010001001100000110100110111
  00101101110001001010011010010001001000010
  11000010101000101111010001011111111111111
  01000

VI. ANALISIS

Hasil eksperimen-eksperimen di atas menunjukkan bahwa ke lima algoritma telah berhasil menunjukkan sifat chaosnya dimana dengan umpan yang selisihnya sedikit, dapat menghasilkan deretan bilangan acak yang sangat berbeda. Selain itu, waktu yang dibutuhkan unutk menghasilkan deretan tersebut juga tergolong sangat cepat unutk tiap algoritma dimana perbedaan yang terlihat hanya 1 hingga 3 mili detik. Berikutnya penulis akan membahas tiap algoritma. Pada bagian ini, penulis akan menganalisis tiap algoritma baik dari model dari algoritma tersebut dan juga uji statistik

A. *Analisis Sekilas*

Sawtooth map merupakan algoritma dengan rumus yang

yang paling sederhana. Akan tetapi, rumusnya yang sederhana juga menjadi kekurangannya. Kita bisa ada pola yang muncul pada eksperimen. Ada bagian grafik yang menunjukkan pola grafik kurva 2x. Hal ini bisa membahayakan jika dipakai unutk kriptografi karena hal ini memungkinkan kriptanalisis memprediksi deretan bilangan acak yang dihasilkan.Selain itu deret bit juga menunujukkan kalau setelah sampai pada iterasi tertentu bit yang dihasilkan akan selalu 0.

Gauss iterated map memiliki kelebihan dimana ia memiliki 2 parameter tambahan yaitu α dan β. Nilai α dan β bisa berbeda-beda sehingga algoritma ini bisa menghasilkan bilangan yang berbeda untuk umpan yang sama. Hal ini tentunya menyulitkan kriptanalisis untuk menbak deretan.

Logistic map menggunakan rumus yang cukup sederhana dengan menambah satu parameter tambahan yaitu r. Parameter ini dapat mempersulit predikisi. Namun, r adalah bilangan bulat. Jika dibandingkan dengan parameter algoritma lainnya yang merupakan bilangan real, bilangan bulat akan lebih mudah diprediksi terutama dengan range yang sangat terbatas.

Tent map menggunakan rumus sesederhana sawtooth map dengan menggunakan sebuah parameter real tambahan μ. Parameter ini dapat mempersulit prediksi baris. Namun, masih lebih baik Gauss iterated map yang memiliki 2 parameter bernilai real.

Circle Map menggunakan rumus trigonometri dengan dua parameter Ω (real) dan K (bulat). Sama seperti Gauss Iterated Map, dua buah parameter dapat mempersulit predikisi. Namun, parameter K bernilai bulat. Dengan parameter K yang nilainya terbatas, masih lebih baik menggunakan Gauss Iterated Map yang kedua parameternya bernilai real.

B. Uji Statistik

Pada bagian ini penulis akan melakukan uji statistic terhadap deret bit menggunakan frequency test, serial test, poker test, runs test, dan autocorrelation test.

Pertama-tama kita harus menentukan dahulu beberapa parameter yang akan digunakan pada beberapa uji statistik nanti. Poker test membutuhkan suatu nilai m yang akan digunakan untuk membagi deret bit ke beberapa potongan. Nilai m harus memenuhi kondisi $\left\lfloor \frac{n}{m} \right\rfloor \geq 5.(2^m)$. $5.2^m = 40$ dan n = 128. Nilai m terbesar yang masih memenuhi kondisi tersebut adalah 3. Jadi untuk poker test ktia gunakan m = 3. Runs test membutuhkan nilai k yang mana merupakan nilai i terbesar yang masih mememenuhi $e_i \geq 5$. Kita hitung satu persatu nilai e dan didapatkan $e_1 = 16.25$, $e_2 = 8.0625$, $e_3 = 4$. Karena nilai e_3 di bawah 5, nilai k yang kita gunakan adalah 2. Terakhir, autocorrelation test membutuhan nilai d yang berjarak antara 1 dan n / 2. Karena bebas, kita gunakan d = 8.

Untuk menentukan kelolosan tiap tes, kita menggunakan normal distributiin dan χ^2 distribution dengan nilai α = 0.05. Threshold dengan menggunakan nilai yang ada [ada tabel normal dan χ^2 distribution (lihat tabel pada lampiran). Jadi, tiap hasil test harus menghasilkan nilai dibawah threshold-threshold berikut untuk dinyatakan lolos:

- Frequency test: 3.8415
- Serial test: 5.9915
- Poker test: 14.0671
- Runs test: 5.9915
- Autocorrelation test: 1.6449

Deret Sawtooth map:
- Frequency test: $X_1 = 32$ => gagal
- Serial test: $X_2 = 95.7087$ => gagal
- Poker test: $X_3 = 97.8095$ => gagal
- Runs test: $X_4 = 29.1557$ => gagal
- Autocorrelation test: $X_5 = -5.2947$ => lolos

Deret Gauss Iterated Map:
- Frequency test: $X_1 = 0.7813$ => lolos
- Serial test: $X_2 = 4.2817$ => lolos
- Poker test: $X_3 = 3.3333$ => lolos
- Runs test: $X_4 = 6.6727$ => gagal
- Autocorrelation test: $X_5 = 0.5477$ => lolos

Deret Logistic Map:
- Frequency test: $X_1 = 6.125$ => gagal
- Serial test: $X_2 = 7.127$ => gagal
- Poker test: $X_3 = 9.8095$ => lolos
- Runs test: $X_4 = 3.9793$ => lolos
- Autocorrelation test: $X_5 = -1.0954$ => lolos

Deret Tent Map:
- Frequency test: $X_1 = 1.125$ => lolos
- Serial test: $X_2 = 3.4341$ => lolos
- Poker test: $X_3 = 5.2381$ => lolos
- Runs test: $X_4 = 5.0026$ => lolos
- Autocorrelation test: $X_5 = 1.4606$ => lolos

Deret Circle Map:
- Frequency test: $X_1 = 0.125$ => lolos
- Serial test: $X_2 = 0.0876$ => lolos
- Poker test: $X_3 = 3.3333$ => lolos
- Runs test: $X_4 = 1.8446$ => lolos
- Autocorrelation test: $X_5 = -1.0954$ => lolos

VII. KESIMPULAN

Menjawab rumusan masalah yang penulis paparkan pada pendahuluan, berikut kesimpulan-kesimpulan yang dapat penulis ambil:

1. Chaotic map yang baik untuk diimplementasi sebagai pembangkit bilangan acak semu adalah Tent Map dan Circle Map yang berhasil menghasilkan deret bit yang lolos semua uji statistik. Di antara kedua chaotic map tersebut, penulis berpendapat lebih baik menggunakan Circle Map karena memiliki 2 parameter yang bsia diatur sendiri oleh pengguna. Artinya circle map menggunakan 3 kunci yaitu umpan, Ω, dan K. 3 kunci akan lebih sulit untuk dibongkar dibandingkan Tent Map yang memilki 2 kunci, umpan dan μ.

2. Sawtooth map sangat tidak baik untuk digunakan sebagai pembangkit bilangan acak. Rumusnya yang terlalu sederhana masih dapat memunculkan pola pada deret bilangan acak yang dihasilkan. Deret bit yang dihasilkan pun akan terus menghasilkan bit 0 setelah melewati suatu iterasi tertentu. Tidak heran deret bit yang dihasilkan hanya lolos autocorrelation test dan gagal apda test lainnya. Gauss Iterated Map dan Logistic Map sekilas sudah menghasilkan deret bilangan acak yang bagus. Namun, setelah deret bit yang dihasilkan diuji statistic ternyata deret-deret tersebut gagal lolos pada beberapa tes. Deret Gauss Iterated Map gagal lolos dari Runs test dan deret Logistic Map gagal lolos dari frequency dan serial test. Deret bit yang gagal pada salah satu uji statistic yang diberikan masih dapat dipertanyakan sifat keacakaannya. Oleh karena itu,s ebaiknya penggunaan Gauss Iterated Map dan Logistic Map sebagai pembangkit bilangan acak semu dihindari.

Dari kesimpulan-kesimpulan di atas, didapatkan bahwa algoritma yang mengimplementasi Tent Map dan Circle Map tergolong sebagai CSPRNG karena lolos persyaratan 1 dan 2. Sedangkan algoritma yang mengimplementasi Sawtooth Map, Gauss Iterated Map, dan Logistic Map masih tergolong sebagai PRNG biasa karena belom lolos persyaratan pertama (persyaratan kedua juga tidak dipenuhi oleh sawtooth map).

REFERENCES

[1] Menezes, van Oorchot, dan Vanstone. 1996. Handbook of Applied Cryptography. CRC Press.
[2] Munir, Rinladi. 2011. Slide IF3058 Kriptografi – Pembangkit Bilangan Acak. STEI-ITB.
[3] http://www.ibiblio.org/e-notes/Chaos/saw.htm
[4] http://mathworld.wolfram.com/LogisticEquation.html
[5] http://mathworld.wolfram.com/LogisticMap.html
[6] http://mathworld.wolfram.com/CircleMap.html

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, 20 Mei 2013

ttd

Danny Andrianto 13510011

Lampiran 1: Tabel N(0, 1) distribution

α	0.1	0.05	0.025	0.01	0.005	0.0025	0.001	0.0005
x	1.2816	1.6449	1.9600	2.3263	2.5758	2.8070	3.0902	3.2905

Lampiran 2: Tabel χ^2 distribution

	α					
v	0.100	0.050	0.025	0.010	0.005	0.001
1	2.7055	3.8415	5.0239	6.6349	7.8794	10.8276
2	4.6052	5.9915	7.3778	9.2103	10.5966	13.8155
3	6.2514	7.8147	9.3484	11.3449	12.8382	16.2662
4	7.7794	9.4877	11.1433	13.2767	14.8603	18.4668
5	9.2364	11.0705	12.8325	15.0863	16.7496	20.5150
6	10.6446	12.5916	14.4494	16.8119	18.5476	22.4577
7	12.0170	14.0671	16.0128	18.4753	20.2777	24.3219
8	13.3616	15.5073	17.5345	20.0902	21.9550	26.1245
9	14.6837	16.9190	19.0228	21.6660	23.5894	27.8772
10	15.9872	18.3070	20.4832	23.2093	25.1882	29.5883
11	17.2750	19.6751	21.9200	24.7250	26.7568	31.2641
12	18.5493	21.0261	23.3367	26.2170	28.2995	32.9095
13	19.8119	22.3620	24.7356	27.6882	29.8195	34.5282
14	21.0641	23.6848	26.1189	29.1412	31.3193	36.1233
15	22.3071	24.9958	27.4884	30.5779	32.8013	37.6973
16	23.5418	26.2962	28.8454	31.9999	34.2672	39.2524
17	24.7690	27.5871	30.1910	33.4087	35.7185	40.7902
18	25.9894	28.8693	31.5264	34.8053	37.1565	42.3124
19	27.2036	30.1435	32.8523	36.1909	38.5823	43.8202
20	28.4120	31.4104	34.1696	37.5662	39.9968	45.3147
21	29.6151	32.6706	35.4789	38.9322	41.4011	46.7970
22	30.8133	33.9244	36.7807	40.2894	42.7957	48.2679
23	32.0069	35.1725	38.0756	41.6384	44.1813	49.7282
24	33.1962	36.4150	39.3641	42.9798	45.5585	51.1786
25	34.3816	37.6525	40.6465	44.3141	46.9279	52.6197
26	35.5632	38.8851	41.9232	45.6417	48.2899	54.0520
27	36.7412	40.1133	43.1945	46.9629	49.6449	55.4760
28	37.9159	41.3371	44.4608	48.2782	50.9934	56.8923
29	39.0875	42.5570	45.7223	49.5879	52.3356	58.3012
30	40.2560	43.7730	46.9792	50.8922	53.6720	59.7031
31	41.4217	44.9853	48.2319	52.1914	55.0027	61.0983
63	77.7454	82.5287	86.8296	92.0100	95.6493	103.4424
127	147.8048	154.3015	160.0858	166.9874	171.7961	181.9930
255	284.3359	293.2478	301.1250	310.4574	316.9194	330.5197
511	552.3739	564.6961	575.5298	588.2978	597.0978	615.5149
1023	1081.3794	1098.5208	1113.5334	1131.1587	1143.2653	1168.4972

Lampiran 3: Implementasi sawtooth map sebagai PRNG dalam bahasa C#

```
namespace ChaosTheory
{
    class Sawtooth
    {
        private double x;

        public Sawtooth(double x)
        {
            this.x = x;
        }

        public Sawtooth(int x)
        {
            this.x = 0 + Double.Parse("0." + x.ToString());
        }

        public long Next()
        {
            x = (2 * x) % 1;
            string[] parts = x.ToString().Split('.');
            //while (parts[1].IndexOf("00") != -1 || parts[1].IndexOf("99") != -1)
            //{
            //    parts[1] = parts[1].Replace("00", "0");
            //    parts[1] = parts[1].Replace("99", "9");
            //}
            //return Math.Abs(Int64.Parse(parts[0])) + Int64.Parse(parts[1]);
            if (parts.Length > 1)
            {
                return Int64.Parse("" + parts[0].ElementAt(parts[0].Length - 1)) +
Int64.Parse("" + parts[1].ElementAt(parts[1].Length - 1));
            }
            else
            {
                return Int64.Parse("" + parts[0].ElementAt(parts[0].Length - 1));
            }
        }

        public long Next(int maxValue)
        {
            return (Next() % maxValue);
        }

        public bool NextBit()
        {
            return (Next() % 2 != 0);
        }
    }
}
```

Lampiran 4: Implementasi gauss iterated map sebagai PRNG dalam bahasa C#

```
namespace ChaosTheory
{
    class Gauss
    {
        private double x, a, b;
        private Random rand;

        private void Step()
        {
            a = rand.Next(0, 10) + rand.NextDouble();
            b = rand.Next(-9, 10) + rand.NextDouble();
        }

        public Gauss(double x)
        {
            this.x = x;
            string[] parts = x.ToString().Split('.');
            rand = new Random(Int32.Parse(parts[1]));
            Step();
        }

        public Gauss(int x)
        {
            this.x = 0 + Double.Parse("0." + x.ToString());
            rand = new Random(x);
            Step();
        }

        public long Next()
        {
            x = Math.Exp(-a * Math.Pow(x, 2)) + b;
            Step();
            string[] parts = x.ToString().Split('.');
            //return Math.Abs(Int64.Parse(parts[0])) + Int64.Parse(parts[1]);
            if (parts.Length > 1)
            {
                return Int64.Parse("" + parts[0].ElementAt(parts[0].Length - 1)) +
Int64.Parse("" + parts[1].ElementAt(parts[1].Length - 1));
            }
            else
            {
                return Int64.Parse("" + parts[0].ElementAt(parts[0].Length - 1));
            }
        }

        public long Next(int maxValue)
        {
            return (Next() % maxValue);
        }

        public bool NextBit()
        {
            return (Next() % 2 != 0);
        }
    }
}
```

Lampiran 5: Implementasi logistic map sebagai PRNG dalam bahasa C#

```
namespace ChaosTheory
{
    class Logistic
    {
        private double x;
        private int r;
        private Random rand;

        private void Step()
        {
            r = rand.Next(1, 5);
        }

        public Logistic(double x)
        {
            this.x = x;
            string[] parts = x.ToString().Split('.');
            rand = new Random(Int32.Parse(parts[1]));
            Step();
        }

        public Logistic(int x)
        {
            this.x = 0 + Double.Parse("0." + x.ToString());
            rand = new Random(x);
            Step();
        }

        public long Next()
        {
            x = r * x * (1 - x);
            Step();
            string[] parts = x.ToString().Split('.');
            //return Math.Abs(Int64.Parse(parts[0])) + Int64.Parse(parts[1]);
            if (parts.Length > 1)
            {
                return Int64.Parse("" + parts[0].ElementAt(parts[0].Length - 1)) +
Int64.Parse("" + parts[1].ElementAt(parts[1].Length - 1));
            }
            else
            {
                return Int64.Parse("" + parts[0].ElementAt(parts[0].Length - 1));
            }
        }

        public long Next(int maxValue)
        {
            return (Next() % maxValue);
        }

        public bool NextBit()
        {
            return (Next() % 2 != 0);
        }
    }
}
```

Lampiran 6: Implementasi tent map sebagai PRNG dalam bahsa C#

```
namespace ChaosTheory
{
    class Tent
    {
        private double x, u;
        private Random rand;

        private void Step()
        {
            u = rand.Next(1, 3) + rand.NextDouble();
        }

        public Tent(double x)
        {
            this.x = x;
            string[] parts = x.ToString().Split('.');
            rand = new Random(Int32.Parse(parts[1]));
            Step();
        }

        public Tent(int x)
        {
            this.x = 0 + Double.Parse("0." + x.ToString());
            rand = new Random(x);
            Step();
        }

        public long Next()
        {
            x = u * Math.Min(x, 1 - x);
            Step();
            string[] parts = x.ToString().Split('.');
            //return Math.Abs(Int64.Parse(parts[0])) + Int64.Parse(parts[1]);
            if (parts.Length > 1)
            {
                return Int64.Parse("" + parts[0].ElementAt(parts[0].Length - 1)) +
Int64.Parse("" + parts[1].ElementAt(parts[1].Length - 1));
            }
            else
            {
                return Int64.Parse("" + parts[0].ElementAt(parts[0].Length - 1));
            }
        }

        public long Next(int maxValue)
        {
            return (Next() % maxValue);
        }

        public bool NextBit()
        {
            return (Next() % 2 != 0);
        }
    }
}
```

Lampiran 7: Implementasi circle map sebagai PRNG dalam bahasa C#

```
namespace ChaosTheory
{
    class Circle
    {
        private double x, k, o;
        private Random rand;

        private void Step()
        {
            o = rand.NextDouble();
        }

        public Circle(double x)
        {
            this.x = x;
            string[] parts = x.ToString().Split('.');
            rand = new Random(Int32.Parse(parts[1]));
            k = 1;
            Step();
        }

        public Circle(int x)
        {
            this.x = 0 + Double.Parse("0." + x.ToString());
            rand = new Random(x);
            k = 1;
            Step();
        }

        public long Next()
        {
            x = x + (o - (k * Math.Sin(2 * Math.PI * x) / 2 * Math.PI));
            Step();
            string[] parts = x.ToString().Split('.');
            //return Math.Abs(Int64.Parse(parts[0])) + Int64.Parse(parts[1]);
            if (parts.Length > 1)
            {
                return Int64.Parse("" + parts[0].ElementAt(parts[0].Length - 1)) +
Int64.Parse("" + parts[1].ElementAt(parts[1].Length - 1));
            }
            else
            {
                return Int64.Parse("" + parts[0].ElementAt(parts[0].Length - 1));
            }
        }

        public long Next(int maxValue)
        {
            return (Next() % maxValue);
        }

        public bool NextBit()
        {
            return (Next() % 2 != 0);
        }
    }
}
```