Pembangkitan Bilangan Acak Dengan Metode Lantai Dan Modulus Bertingkat

Kenji Prahyudi – 13508058
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
kenji_prahyudi@yahoo.com

***Abstrak*—Pembangkitan bilangan acak ini merupakan solusi dari umumnya rumus – rumus pembangkitan bilangan acak lain yang sudah sering digunakan. Dalam makalah ini, akan dibahas mengenai rumus pembangkitan bilangan acak baru, serta perbandingannya dengan pembangkitan bilangan acak yang sudah sering digunakan. Pembangkitan bilangan acak yang dibahas ini lebih dikhususkan untuk digunakan dalam kriptografi. Selanjutnya, ada pembahasan juga mengenai pengujiannya terhadap standar pembangkit bilangan acak yang aman untuk dipakai dalam kriptografi.**

***Index Terms*—Bilangan Acak, Chaos, Lantai, Modulus bertingkat.**

I. PENDAHULUAN

Pada dasarnya, rumus pembangkit bilangan acak sudah banyak ada. Alasan banyaknya rumus pembangkit bilangan acak adalah seringnya bilangan acak digunakan pada kriptografi. Tetapi, semakin sering rumus – rumus pembangkit bilangan acak yang umum itu dipakai, semakin rentan pula rumus – rumus tersebut dengan serangan – serangan kriptanalis untuk dapat memprediksi kemunculan angka berikutnya.

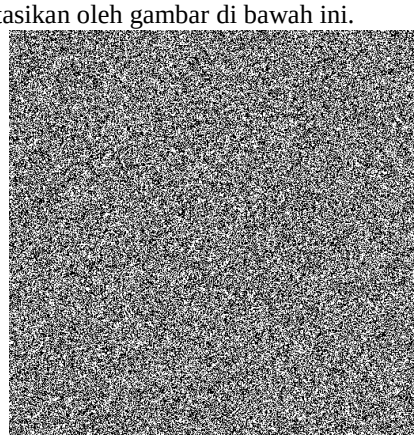
Dalam makalah ini, akan dibahas mengenai rumus pembangkit bilangan acak baru yang keamanannya diharapkan dapat bersaing dengan rumus – rumus yang lain yang sudah ada, terutama yang sudah umum digunakan.

II. BILANGAN ACAK DAN KAITANNYA DENGAN KRIPTOGRAFI

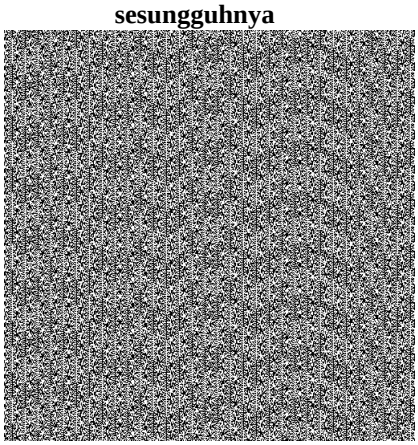
Bilangan acak adalah bilangan yang tidak dapat diprediksi. Misal, ada urutan bilangan sebagai berikut : 2839423, 23483247, 23085, 80458340, 53485, x. Maka kita tidak akan dapat menebak x. Karena tidak ada pola yang terjadi. Bahkan penulis pun tidak dapat menebaknya, karena urutan angka tersebut berasal dari masukan sembarang dari penulis, dimana penulis sendiri tidak memikirkan terlebih dahulu angka berapa yang akan dimasukkan.

Pada saat ini, kriptografi dibuat dengan komputasi. Maka dari itu, pembangkitan bilangan acak pun dilakukan oleh computer. Kenyataannya, sampai sekarang, tidak ada bilangan acak yang dapat dibangkitkan secara benar – benar acak, sehingga bilangan acak yang dibangkitkan dengan menggunakan pembangkitan bilangan acak dengan proses komputasi adalah bilangan acak semu (*pseudo)*, karena bilangan acak yang dihasilkan dapat diulang kembali. Pembangkit bilangan acak semua disebut *pseudo-random number generator (PRNG).*

Dari sebuah referensi di web, penulis menemukan sebuah contoh yang cukup menarik yang menunjukkan perbedaan antara bilangan acak yang sesungguhnya, dengan bilangan acak semu. Perbedaan itu direpresentasikan oleh gambar di bawah ini.



Gambar 1. Visualisasi bilangan acak yang sesungguhnya



Gambar 2. Visualisasi bilangan acak semu

Terlihat dengan jelas di gambar 1, tidak ada pola yang terjadi. Sedangkan pada gambar 2, ada pola yang terjadi pada gambar yang dihasilkan oleh bilangan acak semu.

Pembangkit bilangan acak yang cocok untuk kriptografi dinamakan *cryptographically secure pseudorandom generator* (*CSPRNG*). Persyaratan *CSPRNG* adalah:

1. Secara statistik ia mempunyai sifat-sifat yang bagus (yaitu lolos uji keacakan statistik).
2. Tahan terhadap serangan (*attack*) yang serius. Serangan ini bertujuan untuk memprediksi bilangan acak yang dihasilkan.

Dalam kriptografi, bilangan acak sering digunakan. Sebagai contoh, untuk pembangkitan parameter kunci pada algoritma kunci-publik, pembangkitan *initialization vector* (*IV*) pada algoritma kunci-simetri, dan sebagainya.

III. Pembangkitan Bilangan Acak dengan Teori Chaos

Teori Chaos adalah salah satu teori yang umum digunakan sebagai pembangkit bilangan acak. Teori *chaos* menggambarkan perilaku sistem dinamis nirlanjar yang menunjukkan fenomena *chaos*. Salah satu karakteristik sistem chaos: peka pada nilai awal (sensitive dependence on initial condition).

Sebagai hasil dari sensitifitas, kelakuan sistem yang memperlihatkan *chaos* muncul acak (*random*), meskipun sistem *chaos* sendiri deterministik (dapat didefinisikan dengan baik dan tidak punya parameter acak).

Contoh fungsi *chaos*: persamaan logistik (*logistic map*)

$$f(x) = r\,x(1-x)$$

Dalam bentuk persamaan iteratif:

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

Dimana r : laju pertumbuhan ($0 \le r \le 4$)

x : nilai-nilai chaos ($0 \le x \le 1$)

Misal $r = 4.0$ dan nilai awal $x_0 = 0.456$

$x_1 = 4.0x_0(1 - x_0) = 0.992256$

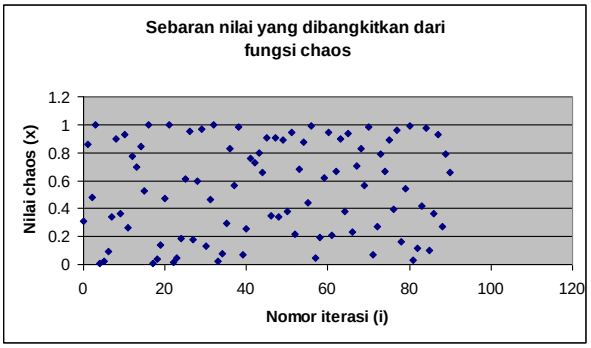
$x_2 = 4.0x_1(1 - x_1) = 0.030736$

…

$x_{99} = 4.0x_{98}(1 - x_{98}) = 0.914379$

$x_{100} = 4.0x_{99}(1 - x_{99}) = 0.313162$

dari fungsi tersebut, dapat disimpulkan bahwa bilangan acak dengan *chaos* tidak punya periode



Gambar 3. Grafik sebaran bilangan acak yang dibangkitkan fungsi chaos

III. Metode Lantai dan Modulus Bertingkat

Metode lantai dan modulus bertingkat hampir tidak pernah dipakai dalam pembangkitan bilangan acak. Terutama pembangkit bilangan acak umum seperti BBS, berbasis RSA, dan chaos. Padahal, metode lantai dan modulus bertingkat ini cukup menarik.

Metode lantai (*floor)* adalah metode sederhana dalam matematika yang membulatkan bilangan decimal ke bawah. Misalnya, nilai x adalah 184,2394. Dan nilai y adalah 48,9992. Nilai lantai dari x adalah 184. Begitu pula dengan y, nilai lantainya adalah 48, bukan 49. Karena pembulatan selalu dilakukan ke bawah, maka metode ini disebut dengan metode lantai, atau notasi simboliknya adalah sebagai berikut :

$$\lfloor 184{,}2394 \rfloor$$

Gambar 4. Notasi simbolik fungsi lantai

Metode modulus bertingkat yang dimaksud dalam makalah ini adalah operasi modulus yang dilakukan di setiap tingkat iterasi. Hal ini menyebabkan di bilangan di setiap iterasi tidak akan melebihi batas yang ditentukan.

IV. Pembangkitan Bilangan Acak dengan Metode Lantai dan Modulus Bertingkat

Pembangkitan bilangan acak dengan metode lantai dan modulus bertingkat adalah pembangkitan bilangan acak yang cukup rumit jika dirumuskan secara matematis, tetapi cukup sederhana jika divisualisasikan.

Angka pembangkit bilangan acak ini terdiri dari 3 buah bilangan, yaitu p, q, dan r, dimana r tidak boleh melebihi banyaknya digit p * q. Untuk amannya, ambil saja r secukupnya, sesuai dengan jumlah digit yang dibutuhkan untuk dipakai. Bilangan p dan q yang diambil pun sangat dianjurkan bilangan prima, agar tidak terjadi pengulangan nilai pada bilangan acak yang dibangkitkan.

Secara matematis, pembangkitan bilangan acak ini dapat dirumuskan sebagai berikut :

$$f(n) = \begin{cases} \left\lfloor \dfrac{p*q}{10^{j(p*q,0)} - r} \right\rfloor, n = 1 \\ \left\lfloor \dfrac{f(n-1)*p*q}{10^{j(p*q)} - r - (n \bmod (j(p*q,0) - r + 1) + 1} \right\rfloor, n > 1 \end{cases}$$

Dimana

$$j(x,y) = \begin{cases} y+1, & \left\lfloor \dfrac{x}{10} \right\rfloor \le 0 \\ j(x, y+1), & \left\lfloor \dfrac{x}{10} \right\rfloor > 0 \end{cases}$$

Fungsi *j(x, y)* dapat digunakan untuk menghitung jumlah digit, jika nilai y diinisialisasi dengan 0.

Sehingga jika p = 233, q = 177, dan r = 3, urutan bilangan acak yang dihasilkan adalah sebagai berikut :

f(1) = 412
f(2) = 699
f(3) = 827, dst..

Sangat rumit jika dilihat secara matematis. Sedangkan secara visual, rumus pembangkit bilangan acak ini adalah sebagai berikut :

p * q = 41241, f(1) = 412
f(1) * p * q = 16991292, f(2) = 699
f(2) * p * q = 28827459, f(3) = 827
dst..

Bagaimana bila angka yang dihasilkan tidak mencukupi? Fungsi akan kembali menelusuri mulai dari depan. Misalnya angka yang dihasilkan dari f(n-1) * p * q adalah sebagai berikut :

128033 = 1280
234823563 = 3482
184935825 = 4935
28358202 = 5820
218501 = ???

Jika terjadi kasus seperti di atas, maka angka yang didapatkan dari fungsi tersebut adalah 2185, yaitu pengambilan empat angka yang diinginkan kembali lagi ke depan.

Dan bagaimana pula kasusnya jika setelah angka yang tidak cukup jumlah digitnya, ternyata angka selanjutnya cukup lagi untuk jumlah iterasi ke-n? misalkan kasusnya sama seperti di atas, hanya saja ada angka selanjutnya yang mencukupi.

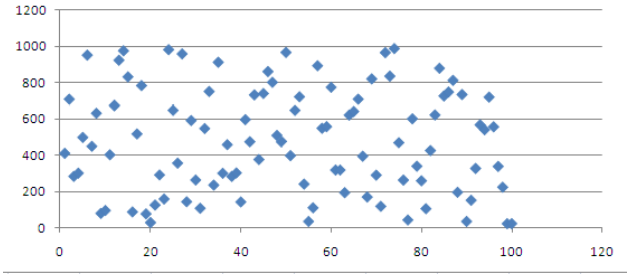
128033 = 1280
234823563 = 3482
184935825 = 4935
28358202 = 5820
218501 = 2185
182041283 = ???

Dengan menggunakan fungsi pembangkit bilangan acak ini, hasil yang didapat adalah lanjutan dari iterasi, karena pengambilan angka dilakukan menurut nilai iterasi dari fungsi itu sendiri, tidak ditentukan oleh pengambilan angka dari bilangan sebelumnya. Sehingga, angka yang diambil adalah 1283. Untuk lebih jelasnya, lihat table angka di bawah ini.

1	**2**	**8**	**0**	3	3			
2	**3**	**4**	**8**	**2**	3	5	6	3
1	8	**4**	**9**	**3**	**5**	8	2	5
2	8	3	**5**	**8**	**2**	**0**	2	
2	**1**	**8**	**5**	0	1			
1	8	2	0	4	**1**	**2**	**8**	**3**
1	**3**	**5**	**5**	8	6	2	7	
2	**5**	**2**	**3**	**4**	6	3	2	7

Tabel 1. Visualisasi pembangkitan bilangan acak dengan metode lantai dan modulus bertingkat

Fungsi pembangkitan bilangan acak ini, jika dilihat dari grafik, tidak memiliki periode. Untuk lebih jelasnya, akan diberikan contoh. Misalkan p yang diambil adalah 17713, dan q 23353. Maka, bilangan acak yang dihasilkan, jika ditampilkan dalam bentuk grafik (dari iterasi ke-1 hingga iterasi ke-100), adalah seperti gambar di bawah ini :



Gambar 5. Grafik sebaran bilangan acak yang dibangkitkan metode lantai dan modulus bertingkat

V. KEAMANAN PEMBANGKITAN BILANGAN ACAK DENGAN METODE LANTAI DAN MODULUS BERTINGKAT

Bilangan acak yang aman untuk kriptografi adalah bilangan acak yang memenuhi syarat CSPRNG. Dari grafik di atas (Gambar 5), dapat disimpulkan bahwa rumus ini memenuhi syarat pertama CSPRNG, yaitu acak secara statistik. Dalam bab ini, akan dicoba beberapa kemunculan angka dari metode pembangkitan bilangan acak ini, lalu dicoba proses penebakan dan pencarian pola dari angka – angka yang dihasilkan. Hal ini dilakukan untuk menguji syarat CSPRNG yang kedua, yaitu tahan terhadap serangan serius.

Misalkan urutan angka yang dihasilkan dari suatu pembangkit adalah 412, 699, 827, 63, 183. Dari urutan angka yang dihasilkan itu, kriptanalis dapat membuat catatan seperti berikut :

p * q = 412_____...
412 * p * q = _699_____...
699 * p * q = __827_____...
827 * p * q = ___063_____...
63 * p * q = ____183_____...

Dari catatan di atas, kriptanalis dapat mengetahui bahwa nilai r adalah 3, karena bilangan acak yang dihasilkan pertama memiliki 3 digit. Tetapi kenyataannya, diketahuinya informasi variable r tidak membantu banyak. Jika dilakukan substitusi p * q, didapatkan beberapa persamaan, yaitu :

- $\frac{\_699\_\_\_\_\ldots}{412} = 412\_\_\_\_\_\ldots$
- $\frac{\_827\_\_\_\_\ldots}{699} = 412\_\_\_\_\_\ldots$

Dari hasil analisis brute force (dicari satu per satu angka yang memungkinkan), angka – angka yang mungkin pada p * q untuk persamaan pertama adalah : untuk 4 digit adalah 4124 – 4126, untuk 5 digit adalah 41238 – 41262, dan seterusnya (sampai digit tak hingga jumlahnya).

Sedangkan pada persamaan kedua, p * q yang mungkin adalah : untuk 5 digit adalah 41241, untuk 6 digit adalah

412404 – 412417, dan seterusnya (sampai digit tak hingga jumlahnya).

Setelah dicari irisannya, ditemukanlah angka yang sama adalah 41241, dan masih banyak angka yang lainnya yang lebih besar, tetapi kriptanalis akan mencoba irisan yang paling pertama ditemukan dulu. Dan ternyata setelah angka 41241 disubstitusi ke dalam 5 persamaan yang dia dapatkan, hasilnya cocok. Berarti tebakan kriptanalisis benar!

Mengapa kriptanalis dapat memecahkan bilangan acak tersebut padahal baru menggunakan dua buah persamaan? Hal ini disebabkan oleh nnilai p dan q yang terlalu kecil, sehingga kemungkinan irisan himpunan nilai kemungkinan p * q antara persamaan satu dengan yang lain menjadi sedikit. Hal ini akan menjadi sangat sulit jika p dan q dibangkitkan dengan nilai 128 bit atau 256 bit.

Bayangkan saja, misalkan dengan nilai p = 27301273081912501259125095127512571 dan q = 17129412648612094126094126492182193, dengan r = 4. Betapa banyak irisan yang terjadi dalam lima bahkan lebih persamaan!

Untuk pembangkit bilangan acak :

p = 69437676016411973856886900154489513845522431527808932147285701492793072545813, dan

q = 10703877674199085428674520926485896105234394116458309599511667313909399986 4079, dengan

r = 3.

Maka n yang didapat adalah 743252390060341414048893240105948289519388670942839719850491593377962602694661837340338334193388972468056461339125398556504815751941244023438620850055122 7.

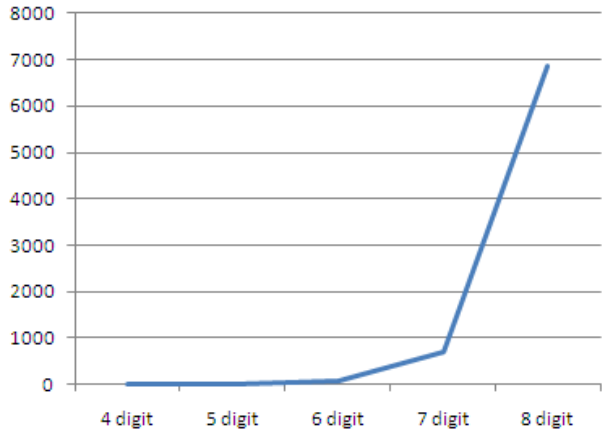
Dari nilai – nilai tersebut, angka – angka yang didapat adalah 743, 579, 250, 46, 98. Dari urutan angka yang dihasilkan itu, kriptanalis dapat membuat catatan seperti berikut :

p * q = 743_____...
743 * p * q = _579_____...
579 * p * q = __250_____...
250 * p * q = ___046_____...
46 * p * q = ____098_____...

Dari catatan di atas, kriptanalis dapat mengetahui bahwa nilai r adalah 3, karena bilangan acak yang dihasilkan pertama memiliki 3 digit. Tetapi, seperti yang telah disebutkan sebelumnya, informasi tersebut tidak membantu banyak. Jika dilakukan substitusi p * q, didapatkan beberapa persamaan, yaitu :

- $\frac{\_579\_\_\_\_\ldots}{743} = 743\_\_\_\_\_\ldots$.
- $\frac{\_\_250\_\_\_\_\_\ldots}{579} = 743\_\_\_\_\_\ldots$.

Dari hasil analisis brute force (dicari satu per satu angka yang memungkinkan), angka yang menjadi irisan antara dua persamaan tersebut sangat banyak, terutama di digit – digit yang besar. Hal ini terbukti dari statistic yang penulis buat di bawah ini.



Gambar 6. Grafik jumlah kemungkinan angka untuk n berjumlah 4-8 digit

Untuk penebakkan nilai n sebanyak 4 digit, angka yang memungkinkan ada 1 buah. 5 digit = 7 buah. 6 digit = 69 buah. 7 digit = 687 buah. 8 digit = 6865 buah. Dan seterusnya. Dari statistik tersebut, dapat dilihat perkembangan jumlah kesamaan yang ada adalah bersifat mendekati deret geometri dengan rasio 10. Sehingga, untuk nilai n yang disebutkan di atas (memiliki 154 digit), ada sekitar 0,7 x 10^{150} kemungkinan! Sehingga, dapat dirumuskan perkiraan jumlah kemungkinan untuk sekian buah persamaan yang diketahui.

$$jumlah\ kemungkinan = \lceil a \times 10^{j-b} \rceil$$

Dimana

- j adalah prediksi jumlah digit dari nilai n.
- a adalah sebuah nilai, dimana berbanding terbalik dengan jumlah persamaan yang diketahui, yaitu diambil dari jumlah kemungkinan pertama yang bernilai lebih dari satu dibagi 10.
- b adalah sebuah nilai yang mewakilkan jumlah digit terakhir sebelum jumlah kemungkinan mencapai lebih dari satu.

Misalnya, dalam kasus di atas :

- Nilai j = 154.
- Nilai a diambil dari jumlah kemungkinan pertama yang bernilai lebih dari satu, yang berasal dari percobaan n 5 digit, yaitu 7 (sebelumnya percobaan n 4 digit, jumlah kemungkinannya adalah 1), lalu dibagi 10, sehingga a = 0,7.
- Nilai b diambil dari jumlah digit terakhir sebelum jumlah kemungkinan mencapai lebih dari satu, yaitu 4, dimana percobaan selanjutnya akan menghasilkan jumlah kemungkinan lebih dari satu, yaitu 7.

Dari perhitungan di atas, didapatkanlah rumus yang tadi, yaitu 0,7 x 100^{154-4} = 0,7 x 100^{150}. Perlu diingat disini, jumlah kemungkinan yang didapat adalah hanya merupakan prediksi, sehingga bukan jumlah kemungkinan yang eksak.

Tetapi, pada kenyataannya, kriptanalis akan mencoba semua kemungkinan, karena kriptanalis tidak mengetahui jumlah digit dari n. Sehingga, jumlah percobaan yang

butuh dilakukan untuk menemukan pembangkit yang sebenarnya adalah jumlah dari semua kemungkinan digit sebelum jumlah digit yang sebenarnya.

Jumlah semua kemungkinan tersebut dapat dirumuskan dalam notasi sigma sebagai berikut :

$$\sum_{i=r}^{j} \lceil a \times 10^{i-b} \rceil$$

Dimana

- j adalah jumlah digit dari nilai n yang sebenarnya.
- a adalah sebuah nilai, dimana berbanding terbalik dengan jumlah persamaan yang diketahui, yaitu diambil dari jumlah kemungkinan pertama yang bernilai lebih dari satu dibagi 10.
- b adalah sebuah nilai yang mewakilkan jumlah digit terakhir sebelum jumlah kemungkinan mencapai lebih dari satu.

Sebenarnya, pembangkit bilangan p dan q tidak mempersulit kriptanalis untuk memprediksi kemunculan bilangan acak selanjutnya secara matematis. Tetapi, hal ini membuat kriptanalis akan mengalami kesulitan jika mencari dari mana asal p dan q muncul, karena bilangan yang dicari bukan satu buah, melainkan dua buah.

VI. Perbandingan dengan Bilangan Acak Teori Chaos

Pembangkitan bilangan acak dengan teori chaos persamaan logistic sudah dibahas di atas. Sekarang, bagaimana perbandingannya dengan pembangkitan bilangan acak metode lantai dan modulus bertingkat? Jika urutan bilangan acak pertama dan kedua diketahui, kunci pembangkit teori chaos akan sangat mudah ditebak dengan substitusi biasa.

Contoh :

Misalkan $x_1 = r * x_0(1 - x_0) = 0.992256$, dan

$x_2 = r * x_1(1 - x_1) = 0.030736$

Dari kedua persamaan di atas, nilai r dapat ditebak dari persamaan ke-2, yaitu dengan melakukan substitusi nilai x_1 ke dalam variable di persamaan ke-2. Jika r sudah dapat diketahui, maka x_0 pun dapat diketahui, sehingga bilangan acak – bilangan acak berikutnya dapat diprediksi.

Berbeda dengan pembangkitan bilangan acak dengan metode lantai dan modulus bertingkat, dimana walaupun diketahui beberapa urutan bilangan acak, prediksi bilangan acak berikutnya akan tetap sangat sulit untuk dilakukan.

Keamanan pembangkitan bilangan acak dengan metode lantai dan modulus bertingkat berada pada jumlah digit dari nilai p dan q, sedangkan bilangan acak dengan teori chaos sebaiknya dibangkitkan dengan angka yang dirahasiakan, dengan urutan yang tidak pasti (tidak dimulai dari satu, dan tidak selalu bertambah 1 di setiap iterasi pembangkitannya). Kendati demikian, cara mengamankan pembangkitan bilangan acak dengan teori chaos dapat pula dilakukan pada pembangkitan – pembangkitan bilangan acak lainnya.

Kesamaan dari kedua metode tersebut adalah, keduanya tidak memiliki periode. Hanya saja, metode pembangkitan yang penulis ajukan ini belum diuji benar – benar sampai dapat dipastikan ketidakadaan-periodenya. Hal ini berkaitan dengan kemampuan penulis dalam melakukan pengujian, dan program yang dibuat penulis belum dapat menampilkan jumlah angka yang memadai untuk membandingkannya satu sama lain, sehingga dapat diketahui periodenya jika ada.

VI. Kesimpulan dan Saran

Untuk mendapatkan keamanan yang maksimal dari rumus pembangkit bilangan acak dengan metode lantai dan modulus bertingkat, dianjurkan untuk menggunakan nilai pembangkit p dan q yang jumlah digitnya jauh lebih banyak dibandingkan nilai pembatas r.

Pembangkitan bilangan acak dengan metode lantai dan modulus bertingkat memenuhi kedua syarat CSPRNG, sehingga dapat digunakan dalam kriptografi.

Bilangan pembangkit p dan q pada pembangkit bilangan acak dengan metode lantai dan modulus bertingkat tidak berfungsi sebagai penambah kesulitan prediksi kemunculan bilangan acak dari segi matematis, tetapi akan mempermudah pengguna dalam menyembunyikan bilangan pembangkit, karena angka yang bersangkutan dengan pembangkitan bilangan acak adalah dua buah, bukan satu.

Saran untuk pengembangan selanjutnya adalah menelaah ulang sifat – sifat dari metode yang penulis ajukan, dan melakukan modifikasi lebih lanjut sehingga meningkatkan keamanan pada pembangkit bilangan acak ini.

VII. Ucapan Terima Kasih

Makalah ini dibuat sebagai tugas dari mata kuliah IF3058 Kriptografi – Sem. II Tahun 2010/2011. Penulis mengucapkan terima kasih sebesar – besarnya kepada Bpk. Ir. Rinaldi Munir, M.T. yang telah mengajar materi – materi kuliah di semester ini, sehingga makalah ini dapat dibuat dengan lancer.

Referensi

Munir, Rinaldi, IF3058 Kriptografi, Pembangkit Bilangan Acak, http://www.informatika.org/~rinaldi/Kriptografi/ 2010-2011/Pembangkit%20Bilangan%20Acak.ppt

Pseudorandom Number Generator, http://en.wikipedia.org/wiki/Random_number_generation#.22True.22_random_numbers_vs._pseudorandom_numbers, diakses tanggal : 5 Mei 2011.

Randomness, http://www.random.org/randomness/, diakses tanggal : 5 Mei 2011
Pseudo-random vs True-random, http://www.boallen.com/random-numbers.html

PERNYATAAN

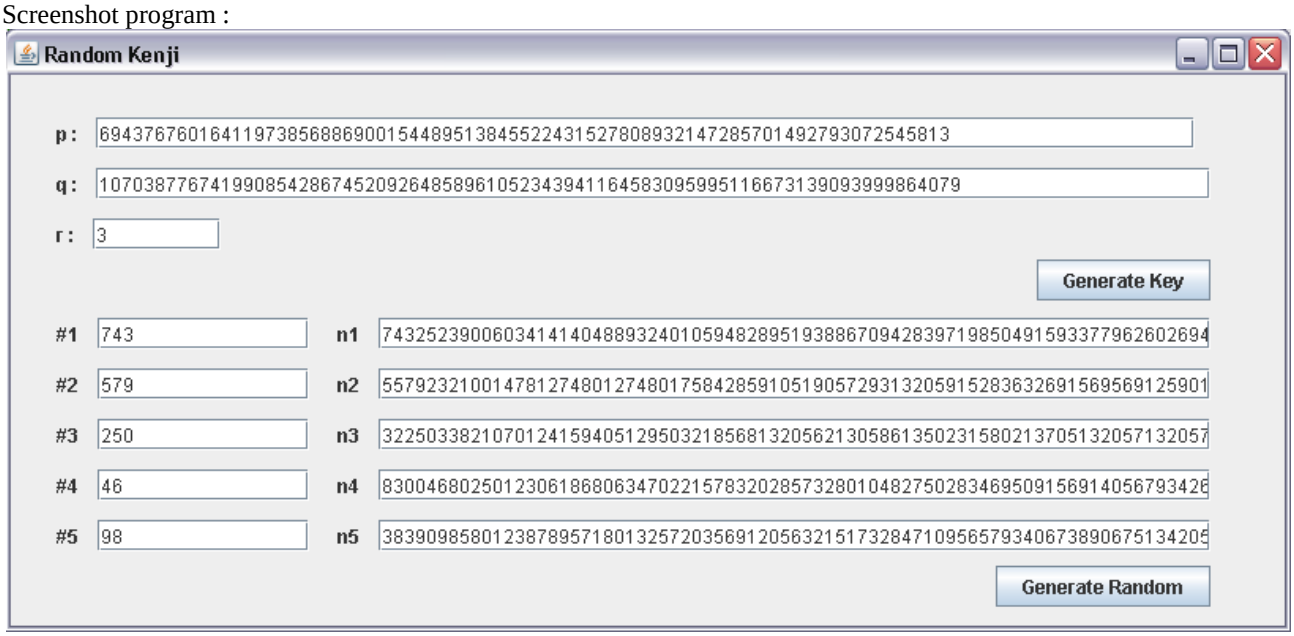
Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 6 Mei 2011

Kenji Prahyudi

LAMPIRAN

Screenshot program :



Gambar Lampiran 1. Contoh pembangkitan 5 bilangan acak dengan p dan q 256-bit

Source code dalam bahasa Java :

Pembangkitan 5 buah bilangan acak dengan p = 177, q = 233, dan r = 3

```
int p = 177, q = 233, r = 3;
long prevVal = 1;
for (int i=0; i<5; i++)
{
   prevVal =
   Integer.parseInt
   (
      String.valueOf
      (
         prevVal * p * q
      ).substring
      (
         i % (String.valueOf(prevVal * p * q).length() - r + 1),
         i % (String.valueOf(prevVal * p * q).length() - r + 1) + r
      )
   );
   System.out.println("nilai = " + prevVal);
}
```