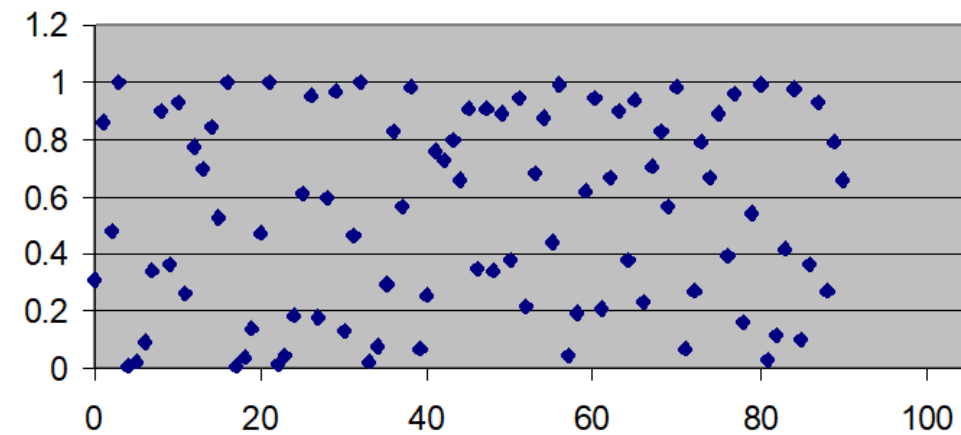


Bahan Kuliah IF4020 Kriptografi



Pembangkit Bilangan Acak



Oleh: Dr. Ir. Rinaldi, M.T

Program Studi Teknik Informatika
STEI-ITB
2025

Bilangan Acak

- Bilangan acak (*random number*): bilangan yang tidak dapat diprediksi nilainya dan kemunculannya
- Bilangan acak dapat berupa *integer*, bilangan riil antara 0-1, atau string biner
Integer: 2367, 987, 1092, 1762, 563, ...

Riil: 0.985, 0.00341, 0.6297, 0.00832, ...

Biner: 10011001010...

- Bilangan acak sangat penting di dalam kriptografi
- Contoh:
 1. Untuk pembangkitan nilai-nilai parameter kunci di dalam algoritma kriptografi kunci-publik,
Alice membangkitkan kunci privat a di dalam Diffie-Hellman Key Exchange
 2. Pembangkitan nilai acak k di dalam algoritma enkripsi ElGamal
Pilih bilangan acak k , yang dalam hal ini $1 \leq k \leq p - 1$
 3. Pembangkitan *initialization vector (IV)* di dalam *block-cipher*
 4. Pembangkitan string di dalam mekanisme *challenge and response* untuk otentikasi
 5. Pembangkitan kunci sesi oleh *client* di dalam SSL

- Tidak ada prosedur komputasi yang menghasilkan deret bilangan acak yang benar-benar sempurna (*truly random*).
- Bilangan acak yang dihasilkan dengan prosedur komputasi adalah **bilangan acak semu** (*pseudo-random*), karena pembangkitan bilangannya dapat diulang kembali.
- Pembangkit deret bilangan acak semacam itu disebut *pseudo-random number generator* (PRNG).
- PRNG bersifat deterministik, artinya bilangan acak bisa diulang kembali pembangkitannya asalkan kunci (umpan) yang digunakan sama.

Linear Congruential Generator (LCG)

- Pembangkit bilangan acak kongruen-lanjar adalah *PRNG* yang berbentuk:

$$x_{n+1} = (ax_n + b) \bmod m$$

x_{n+1} = bilangan acak sekarang

x_n = bilangan acak sebelumnya

a = faktor pengali ($0 < a < m$)

b = *increment* ($0 < b < m$)

m = modulus ($m > 0$)

x_0 = kunci pembangkit atau disebut **umpan** (*seed*), harus dirahasiakan.

- Nilai-nilai x_i terletak dari 0 sampai $m - 1$, yaitu $0 \leq x_i < m$

Contoh: $X_n = (7X_{n-1} + 11) \bmod 17$, dan $X_0 = 0$

Contoh 1: $x_{n+1} = (7x_n + 11) \bmod 17$ dan $x_0 = 0$

$$x_1 = (7 \cdot 0 + 11) \bmod 17 = 11$$

$$x_2 = (7 \cdot 11 + 11) \bmod 17 = 3$$

$$x_3 = (7 \cdot 3 + 11) \bmod 17 = 15$$

$$x_4 = (7 \cdot 15 + 11) \bmod 17 = 14$$

...

$$x_{16} = (7 \cdot 13 + 11) \bmod 17 = 0$$

$$x_{17} = (7 \cdot 0 + 11) \bmod 17 = 11$$

... (terulang kembali 11, 3, 15,)

n	X_n
0	0
1	11
2	3
3	15
4	14
5	7
6	9
7	6
8	2
9	8
10	16
11	4
12	5
13	12
14	10
15	13
16	0
17	11
18	3
19	15
20	14
21	7
22	9
23	6
24	2

- *LCG* mempunyai periode tidak lebih besar dari m , dan pada kebanyakan kasus periodenya kurang dari itu.
- *LCG* mempunyai periode penuh $(m - 1)$ jika memenuhi syarat berikut:
 1. b relatif prima terhadap m .
 2. $a - 1$ dapat dibagi dengan semua faktor prima dari m
 3. $a - 1$ adalah kelipatan 4 jika m adalah kelipatan 4
 4. $m > \max(a, b, x_0)$
 5. $a > 0, b > 0$

- Keunggulan *LCG* terletak pada kesederhanaannya dan komputasi yang relatif cepat.
- Sayangnya, *LCG* tidak dapat digunakan untuk kriptografi karena bilangan acaknya dapat diprediksi urutan kemunculannya.
- Oleh karena itu *LCG* tidak aman digunakan untuk kriptografi.
- Namun demikian, *LCG* tetap berguna untuk aplikasi non-kriptografi seperti simulasi, sebab *LCG* sangkil dan memperlihatkan sifat statistik yang bagus dan sangat tepat untuk uji-uji empirik

Pembangkit Bilangan Acak yang Aman Secara Kriptografi

- Pembangkit bilangan acak yang cocok untuk kriptografi dinamakan *cryptographically secure pseudorandom generator (CSPRNG)*.
- Persyaratan *CSPRNG* adalah:
 1. Secara statistik lolos uji keacakan (*randomness test*).
 2. Tahan terhadap serangan (*attack*) yang serius. Serangan ini bertujuan untuk memprediksi bilangan acak yang dihasilkan dari nilai-nilai sebelumnya.
- Meskipun demikian, CSPRNG tetap termasuk ke dalam pembangkit bilangan acak semu.

Blum Blum Shub (BBS)

- *BBS* adalah *CSPRNG* yang paling sederhana dan paling sangkil (secara kompleksitas teoritis).
- *BBS* dibuat pada tahun 1986 oleh Lenore Blum, Manuel Blum, dan Michael Shub.
- Berbasis teori bilangan (*number theory*)
- *BBS* menghasilkan barisan bilangan acak berupa bit-bit biner
- *BBS* berbentuk:

$$x_{n+1} = x_n^2 \bmod m$$

Algoritma BBS:

1. Pilih dua buah bilangan prima rahasia, p dan q , yang masing-masing kongruen dengan 3 (mod 4).
2. Kalikan keduanya menjadi $n = pq$. Bilangan n ini disebut **bilangan bulat Blum**
3. Pilih bilangan bulat acak lain, s , sebagai umpan sedemikian sehingga:
 - (i) $2 \leq s < n$
 - (ii) s dan n relatif primakemudian hitung $x_0 = s^2 \bmod n$
4. Barisan bit acak dihasilkan dengan melakukan iterasi berikut sepanjang yang diinginkan:
 - (i) Hitung $x_{i+1} = x_i^2 \bmod n$
 - (ii) $z_i = \text{bit } LSB \text{ (Least Significant Bit) dari } x_i$Barisan bit acak adalah $z_1, z_2, z_3, \dots$

Contoh 2. Misalkan kita memilih $p = 11$ dan $q = 23$ sehingga $n = pq = 253$.

Kita pilih $s = 3$ dan kita hitung $x_0 = 3^2 \bmod 253 = 9$.

Barisan bit acak kita hasilkan sebagai berikut:

$$x_1 = x_0^2 \bmod n = 9^2 \bmod 253 = 81 \rightarrow z_1 = 1 \text{ (karena 81 ganjil, bit } LSB\text{-nya pasti 1)}$$

$$x_2 = x_1^2 \bmod n = 81^2 \bmod 253 = 236 \rightarrow z_2 = 0 \text{ (karena 236 genap, bit } LSB\text{-nya pasti 0)}$$

$$x_3 = x_2^2 \bmod n = 236^2 \bmod 253 = 36 \rightarrow z_3 = 0 \text{ (36 genap)}$$

$$x_4 = x_3^2 \bmod n = 36^2 \bmod 253 = 31 \rightarrow z_4 = 1 \text{ (31 ganjil)}$$

$$x_5 = x_4^2 \bmod n = 31^2 \bmod 253 = 202 \rightarrow z_5 = 0 \text{ (202 genap)}$$

dst

Barisan bit acak yang dihasilkan 1 0 0 1 0...

- BBS adalah pembangkit bit acak yang aman secara kriptografi karena lulus uji bit berikutnya (*next-bit test*).
- Sebuah pembangkit bit acak dikatakan lulus uji bit berikutnya (*next-bit test*) jika diberikan barisan k bit, maka tidak dapat diprediksi bit berikutnya 0 atau 1 dengan peluang lebih besar dari $\frac{1}{2}$, sehingga dikatakan *unpredictable*.
- BBS adalah pembangkit bit acak yang *unpredictable*.
- Karakteristik BBS yang menarik adalah kita dapat menghitung x_i secara langsung dengan persamaan:

$$x_i = (x_0^{2^i \bmod \lambda(n)}) \bmod n$$

dalam hal ini, $\lambda(n) = \lambda(pq) = \text{KPK}(p-1, q-1)$, KPK = kelipatan persekutuan terkecil

- Bilangan acak tidak harus 1 bit *LSB* tetapi bisa juga j buah bit (j adalah bilangan bulat positif yang tidak melebihi $\log_2(\log_2 n)$).
- Perhatikan contoh berikut:

Contoh 3. Misalkan $p = 11351$ dan $q = 11987$ sehingga $n = pq = 136064437$.

Kita pilih $s = 80331757$ dan $j = 4$

(j tidak melebihi $\log_2(\log_2 136064437) = 4.75594$).

Hitung: $x_0 = 80331757^2 \bmod 136064437 = 1312737111$.

Barisan bit acak yang dihasilkan adalah sebagai berikut:

$$x_1 = x_0^2 \bmod n = 131273718^2 \bmod 136064437 = 47497112$$

$$z_1 = 47497112 \bmod 2^4 = 8 = 1000_{\text{basis } 2} \text{ (4 bit } \textit{LSB} \text{ dari 47497112)}$$

$$x_2 = x_1^2 \bmod n = 47497112^2 \bmod 136064437 = 69993144$$

$$z_2 = 69993144 \bmod 2^4 = 8 = 1000_{\text{basis } 2} \text{ (4 bit } \textit{LSB} \text{ dari 69993144)}$$

$$x_3 = x_2^2 \bmod n = 69993144^2 \bmod 136064437 = 13810821$$

$$z_3 = 13810821 \bmod 2^4 = 5 = 0101_{\text{basis } 2} \text{ (4 bit } \textit{LSB} \text{ dari 13810821)}$$

...

Barisan blok bit acak yang dihasilkan: 1000 1000 0101 ...

- Keamanan *BBS* terletak pada:
 1. Sulitnya membedakan bit-bit luaran dengan bit acak, paling sedikit sesulit memecahkan persoalan *quadratic residue problem*

Di dalam teori bilangan, bilangan bulat q disebut sisa kuadratik dalam modulus n jika q kongruen dengan akar kuadrat dari n , yaitu terdapat bilangan bulat x sedemikian sehingga $q \equiv x^2 \pmod{n}$ (atau ditulis $x^2 \equiv q \pmod{n}$ sama saja)

Contoh: $x^2 \equiv 4 \pmod{5}$, maka $x = 2$ atau 3 , tetapi $x^2 \equiv 3 \pmod{5}$ tidak ada solusinya

Perhatikan di dalam *BBS* terdapat perhitungan $x_0 = s^2 \bmod n$ dan $x_{i+1} = x_i^2 \bmod n$

2. Sulitnya memfaktorkan n untuk n yang besar. Nilai n tidak perlu rahasia dan dapat diumumkan kepada publik.
- *BBS* tidak dapat diprediksi dari arah kiri (*unpredictable to the left*) dan tidak dapat diprediksi dari arah kanan (*unpredictable to the kanan*), artinya jika diberikan barisan bit yang dihasilkan oleh *BBS*, kriptanalis tidak dapat memprediksi barisan bit sebelumnya dan barisan bit sesudahnya

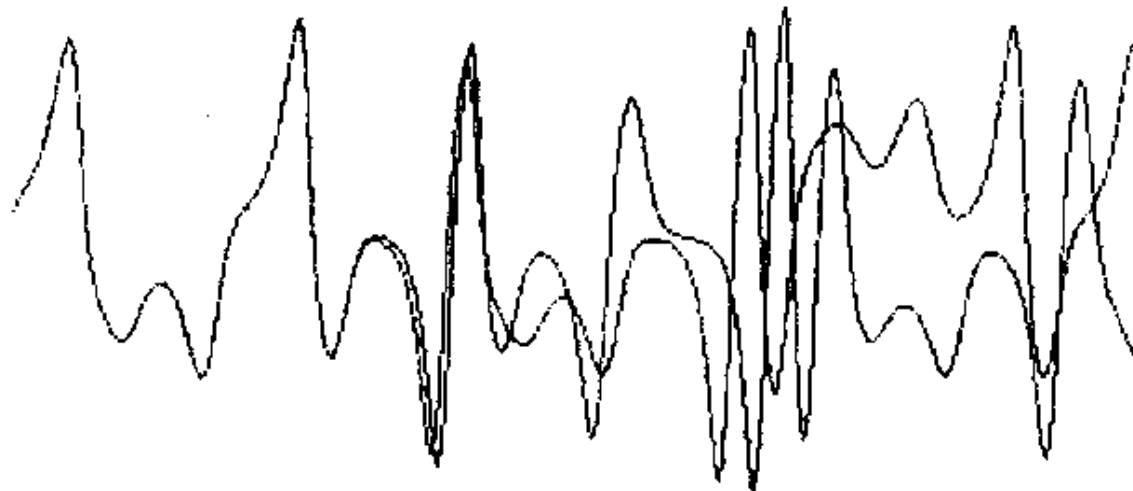
CSPRNG Berbasis RSA

Algoritma:

1. Pilih dua buah bilangan prima rahasia, p dan q , dan bilangan bulat e yang relatif prima dengan $(p - 1)(q - 1)$
2. Kalikan keduanya menjadi $n = pq$
3. Pilih bilangan bulat acak lain, s , sebagai x_0 yang dalam hal ini $2 \leq s \leq n$
4. Barisan bit acak dihasilkan dengan melakukan iterasi berikut sepanjang yang diinginkan:
 - (i) Hitung $x_{i+1} = x_i^e \bmod n$ dengan $x_0 = s$.
 - (ii) $z_i = \text{bit } LSB \text{ (Least Significant Bit) dari } x_i$
5. Barisan bit acak adalah $z_1, z_2, z_3, \dots$

Teori *Chaos*

- 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*).



Logistic Map

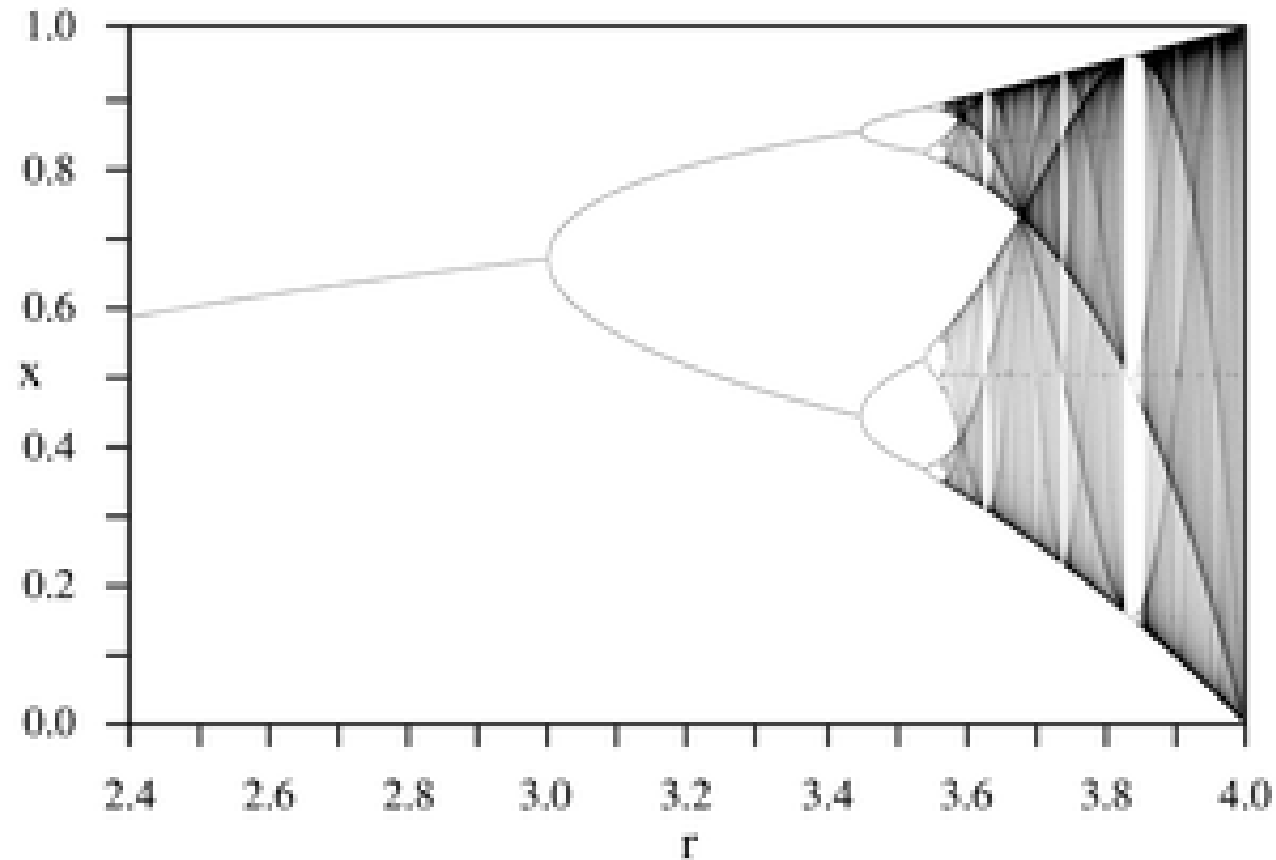
- Contoh fungsi *chaos* yang sederhana: persamaan logistik (*logistic map*)

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

r : laju pertumbuhan ($0 \leq r \leq 4$)

x : nilai-nilai *chaos* ($0 \leq x \leq 1$)

Logistic Map



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

Gambar 1 Diagram *bifurcation* untuk persamaan logistik $x_{i+1} = r x_i (1 - x_i)$

Logistic Map

- Fungsi *chaos* berguna untuk pembangkitan barisan bilangan acak .

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

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

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

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

...

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

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

... dst

- Menariknya bilangan acak dari persamaan *logistic map* tidak punya periode

- Jika kita ingin mendapatkan barisan bilangan bulat dari nilai-nilai *floating-point* yang dihasilkan oleh *logistic map*, maka kita dapat mengambil n digit signifikan dari setiap nilai-nilai *floating point* tersebut.

- Contoh: n = 4

$$x_1 = 4.0x_0(1 - x_0) = 0.992256 \rightarrow 9922$$

$$x_2 = 4.0x_1(1 - x_1) = 0.030736 \rightarrow 3073$$

...

$$x_{99} = 4.0x_{98}(1 - x_{98}) = 0.914379 \rightarrow 9143$$

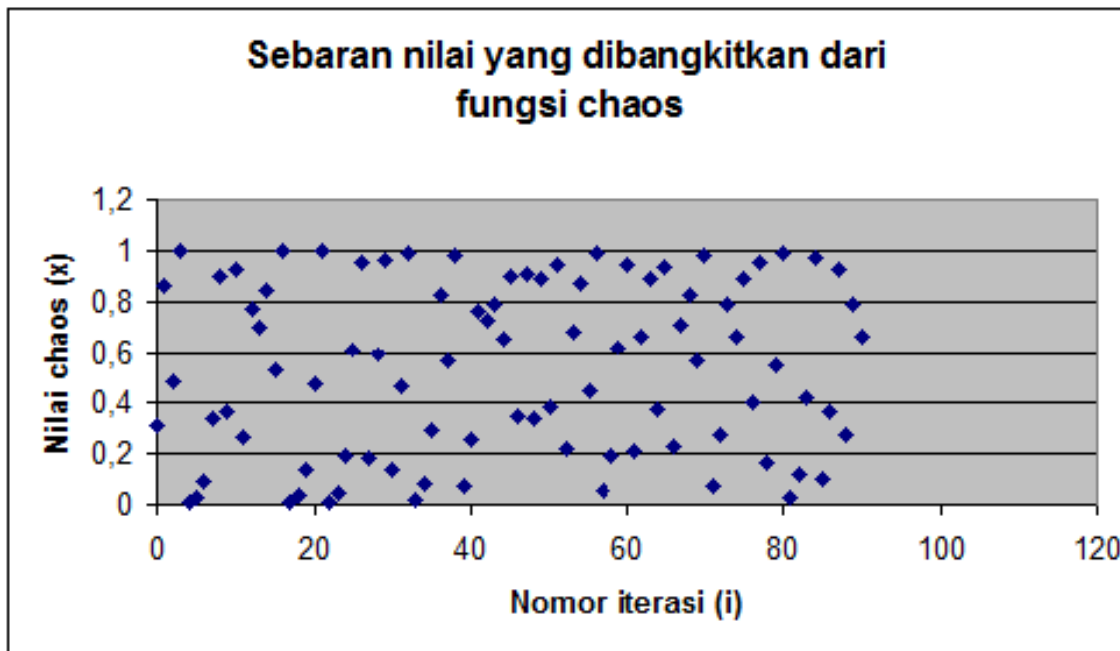
$$x_{100} = 4.0x_{99}(1 - x_{99}) = 0.313162 \rightarrow 3131$$

... dst

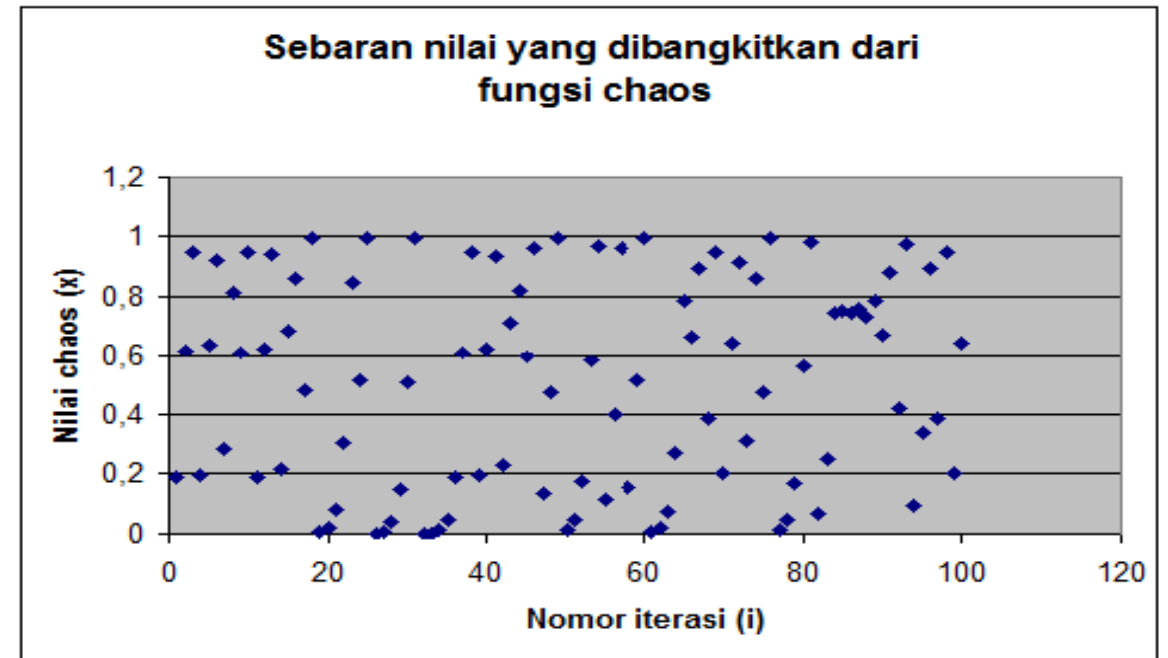
Logistic Map

```
main()
{
    int i, n
    double r, x;
    printf("Masukkan nilai awal (0 s/d 1) : "); scanf("%lf", &x);
    printf("Masukkan jumlah iterasi : "); scanf("%d", &n);
    r = 4.0;
    for (i = 1; i <= n; i++) {
        x = r * x * (1 - x);
        printf("x = %lf ", x);
    }
}
```

- Fungsi *chaos* sensitif terhadap perubahan kecil nilai-nilai awal. Jika x_0 diubah sedikit saja, maka nilai-nilai *chaos* yang dihasilkan berbeda signifikan.
- Perhatikan contoh berikut:



Gambar 1. Nilai-nilai *chaos* yang dibangkitkan dari fungsi *chaos* $x_{n+1} = 4.0 x_n (1 - x_n)$ dengan $x_0 = 0.456$



Gambar 2. Nilai-nilai *chaos* yang dibangkitkan dari fungsi *chaos* $x_{n+1} = 4.0 x_n (1 - x_n)$ dengan $x_0 = 0.456001$

Contoh *chaos map* lainnya:

1. *Henon map*

$$x_n = 1 + b(x_{n-2} - x_{n-3}) + cx_{n-2}^2$$

3. *Arnold's cat map*:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & b \\ c & bc+1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \bmod(N)$$

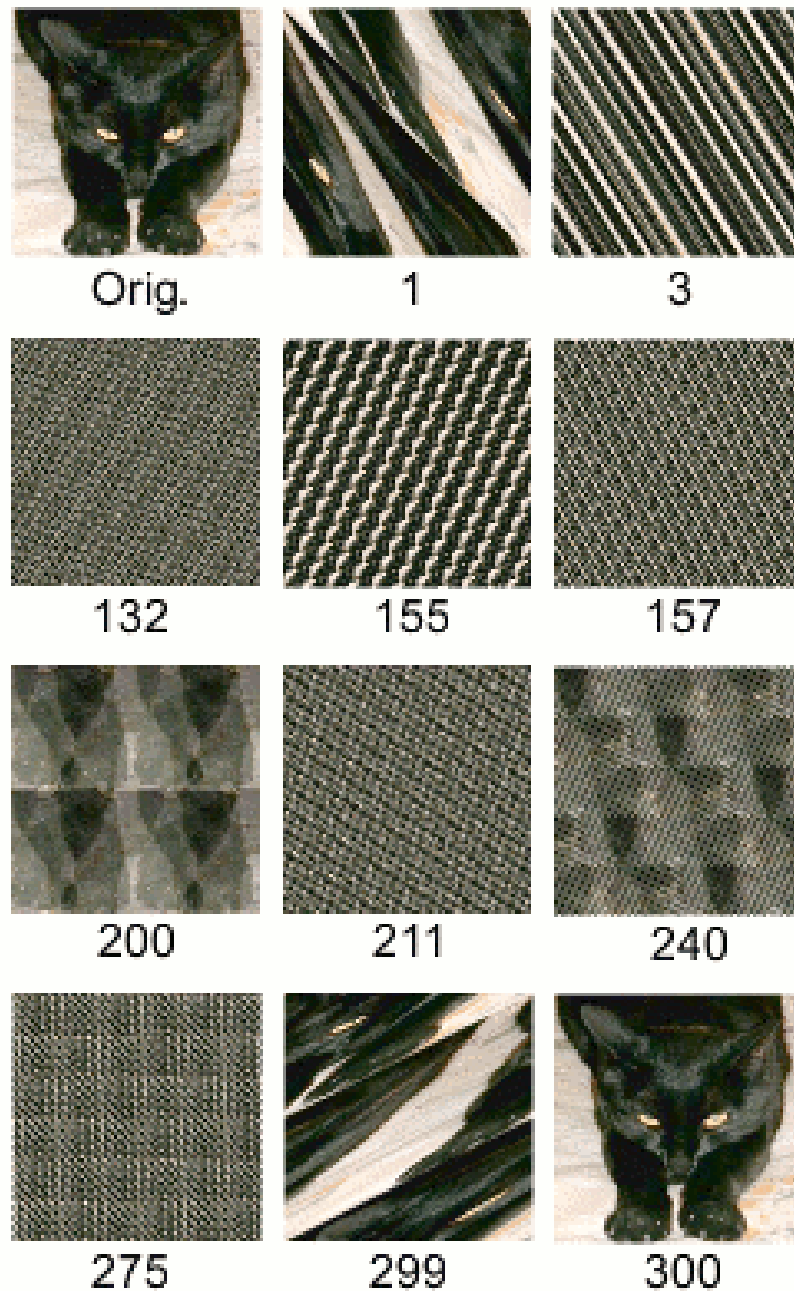
b dan c adalah *integer* positif sembarang. Determinan matriks $\begin{bmatrix} 1 & b \\ c & bc+1 \end{bmatrix}$ harus sama dengan 1

2. *Tent Map*

$$x_{i+1} = f_\mu(x_i) = \begin{cases} \mu x_i & , x_i < \frac{1}{2} \\ \mu(1 - x_i) & , x_i \geq \frac{1}{2} \end{cases}$$

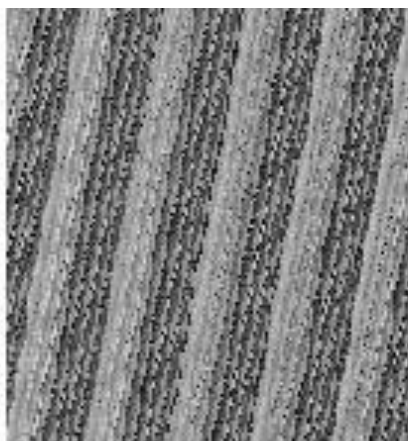
$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & b \\ c & bc+1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \bmod(N)$$

Hasil lelaran dengan
Arnold Cat Map

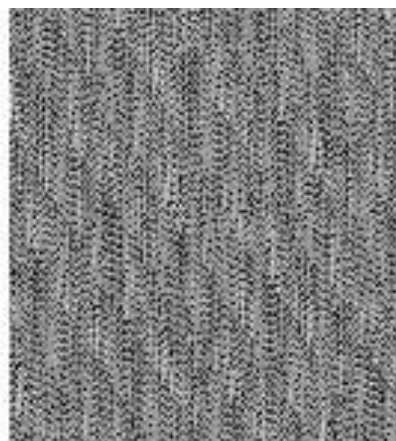




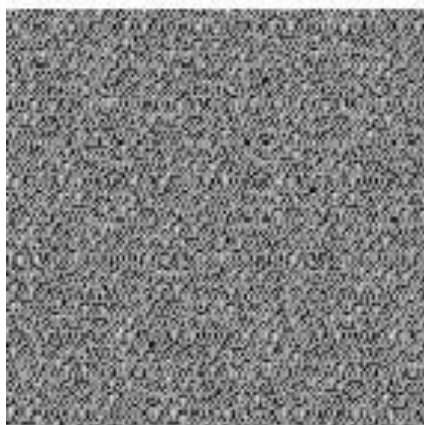
Citra awal



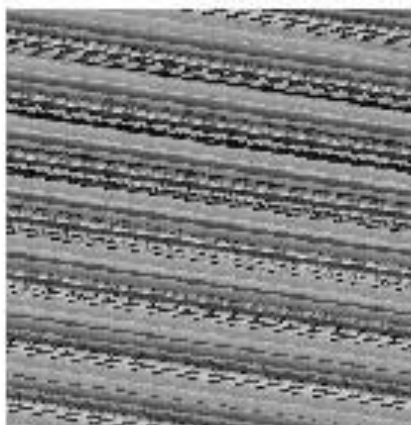
1



3



99



191



192