Pembangkit Bilangan Acak Semu

Bahan Kuliah IF4020 Kriptografi

- Bilangan acak: bilangan yang tidak dapat diprediksi

- Bilangan acak (*random*) banyak digunakan di dalam kriptografi

- Misalnya untuk pembangkitan parameter kunci pada algoritma kunci-publik, pembangkitan *initialization vector* (*IV*) pada algoritma kunci-simetri, dan sebagainya

- Tidak ada komputasi yang benar-benar menghasilkan deret bilangan acak secara sempurna.

- Bilangan acak yang dihasilkan dengan rumus-rumus matematika adalah bilangan acak semu (*pseudo*), karena pembangkitan bilangannya dapat diulang kembali.

- Pembangkit deret bilangan acak semacam itu disebut *pseudo-random number generator* (*PRNG*)

Linear Congruential Generator (*LCG*)

- Pembangkit bilangan acak kongruen-lanjar (*linear congruential generator* atau *LCG*) adalah *PRNG* yang berbentuk:

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

X_n = bilangan acak ke-n dari deretnya

X_{n-1} = bilangan acak sebelumnya

a = faktor pengali

b = *increment*

m = modulus

Kunci pembangkit adalah X_0 yang disebut **umpan** (*seed*).

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

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. $m >$ maks(a, b, x_0)
  5. $a > 0$, $b > 0$

- Keunggulan *LCG* terletak pada kecepatannya dan hanya membutuhkan sedikit operasi bit.

- 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* mangkus dan memperlihatkan sifat statistik yang bagus dan sangat tepat untuk uji-uji empirik

Pembangkit Bilangan Acak yang Aman untuk Kriptografi

- 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.

Blum Blum Shub (BBS)

- *CSPRNG* yang paling sederhana dan paling mangkus (secara kompleksitas teoritis).

- *BBS* dibuat pada tahun 1986 oleh Lenore Blum, Manuel Blum, dan Michael Shub.

- Berbasis teori bilangan

Algoritma:

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 prima

   kemudian hitung $x_0 = s^2 \bmod n$
4. Barisan bit acak dihasilkan dengan melakukan iterasi berikut sepanjang yang diinginkan:

   (i) Hitung $x_i = x_{i-1}^{\,2} \bmod n$

   (ii) z_i = bit *LSB* (*Least Significant Bit*) dari x_i

   Barisan bit acak adalah $z_1, z_2, z_3, \ldots$

Contoh. 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:

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

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

$x_3 = x_2^2 \bmod n = 236^2 \bmod 253 = 36$ ➔ $z_1 = 0$

$x_4 = x_3^2 \bmod n = 36^2 \bmod 253 = 31$ ➔ $z_1 = 1$

$x_5 = x_4^2 \bmod n = 31^2 \bmod 253 = 202$ ➔ $z_1 = 0$

dst

Barisan bit acak yang dihasilkan 10010..

The cryptographic security of the BBS generator assumes that the number theoretic problem of distinguishing a quadratic residue from a pseudo-square in Z mod n is computationally infeasible when n is the product of two primes p and q and the factorization of n is not known. Thus it also assumes that integer factorization is computationally infeasible. Recall the definitions of a quadratic residue and pseudo-square:

- 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 . [BIS03] Misalkan kita memilih p = 11351 dan q = 11987 sehingga n = pq = 136064437. Kita pilih s = 80331757 dan j = 4 (j tidak melebihi $\log_2(\log_2 136064437) = 4.75594$). Kita hitung $x_0 = 80331757^2 \bmod 136064437 = 1312737111$. Barisan bit acak kita hasilkan sebagai berikut:

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

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

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

$z_1 = 69993144 \equiv 8 \pmod{2^4} = 1000_{\text{basis 2}}$ (4 bit *LSB* dari 69993144)

…

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

$z_1 = 13810821 \equiv 5 \pmod{2^4} = 0101_{\text{basis 2}}$ (4 bit *LSB* dari 13810821)

…

Barisan blok bit acak yang dihasilkan: 1000 1000 0101 …

atau dalam basis 10: → 8 8 5 …

- Keamanan *BBS* terletak pada sulitnya memfaktorkan *n*. 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*

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:
   a. Hitung $x_i = x_{i-1}^{\,e} \bmod n$ degan $x_0 = s$.
   b. z_i = bit *LSB* (*Least Significant Bit*) dari x_i
5. Barisan bit acak adalah $z_1, z_2, z_3, \ldots$

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*).

Teori *Chaos*

- 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).

Teori *Chaos*

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

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

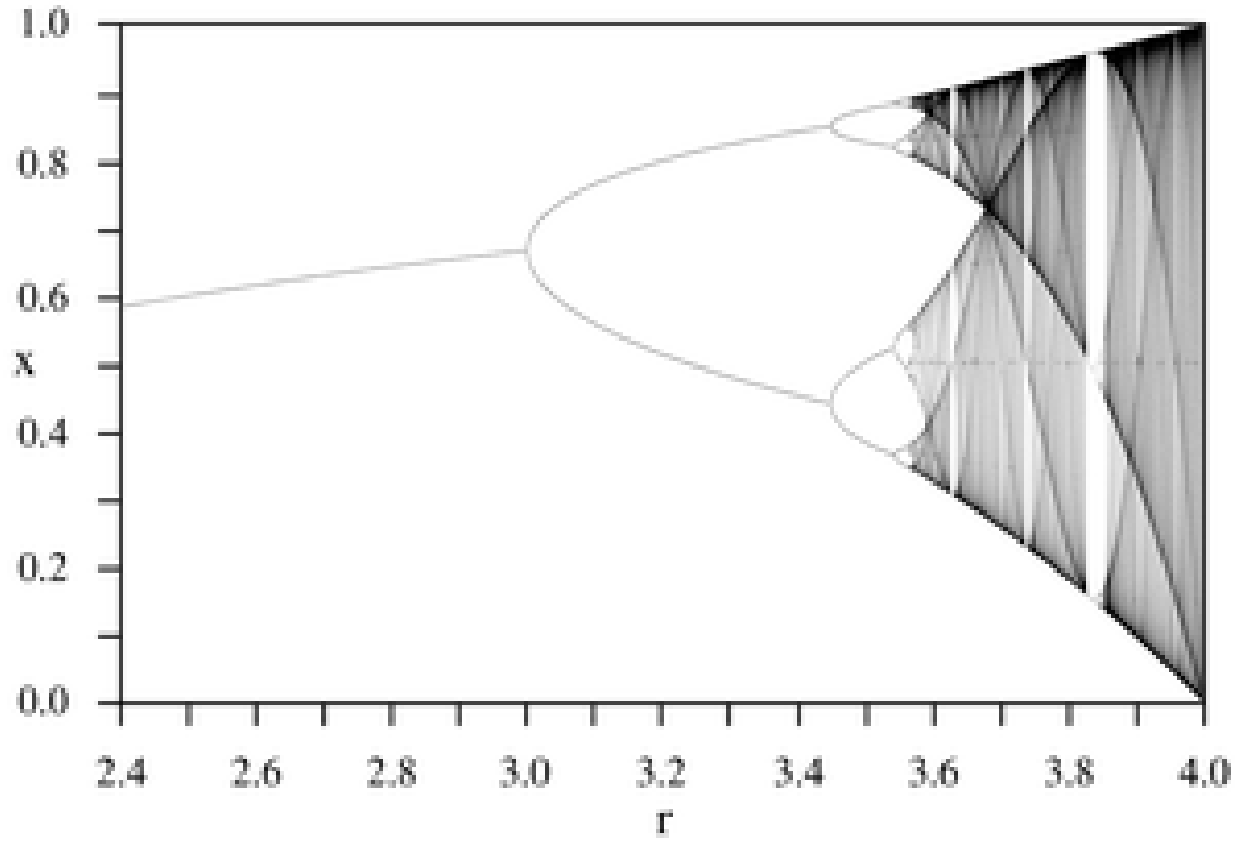
- Dalam bentuk persamaan iteratif:

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

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

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

Teori *Chaos*



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

Teori *Chaos*

- Sistem *chaos* berguna untuk pembangkitan bilangan acak

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

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

$$x_1 = 4.0x_0(1 - x_0) = 0.992256$$
$$x_2 = 4.0x_1(1 - x_1) = 0.030736$$
$$\ldots$$
$$x_{99} = 4.0x_{98}(1 - x_{98}) = 0.914379$$
$$x_{100} = 4.0x_{99}(1 - x_{99}) = 0.313162$$

- Bilangan acak dengan *chaos* tidak punya periode

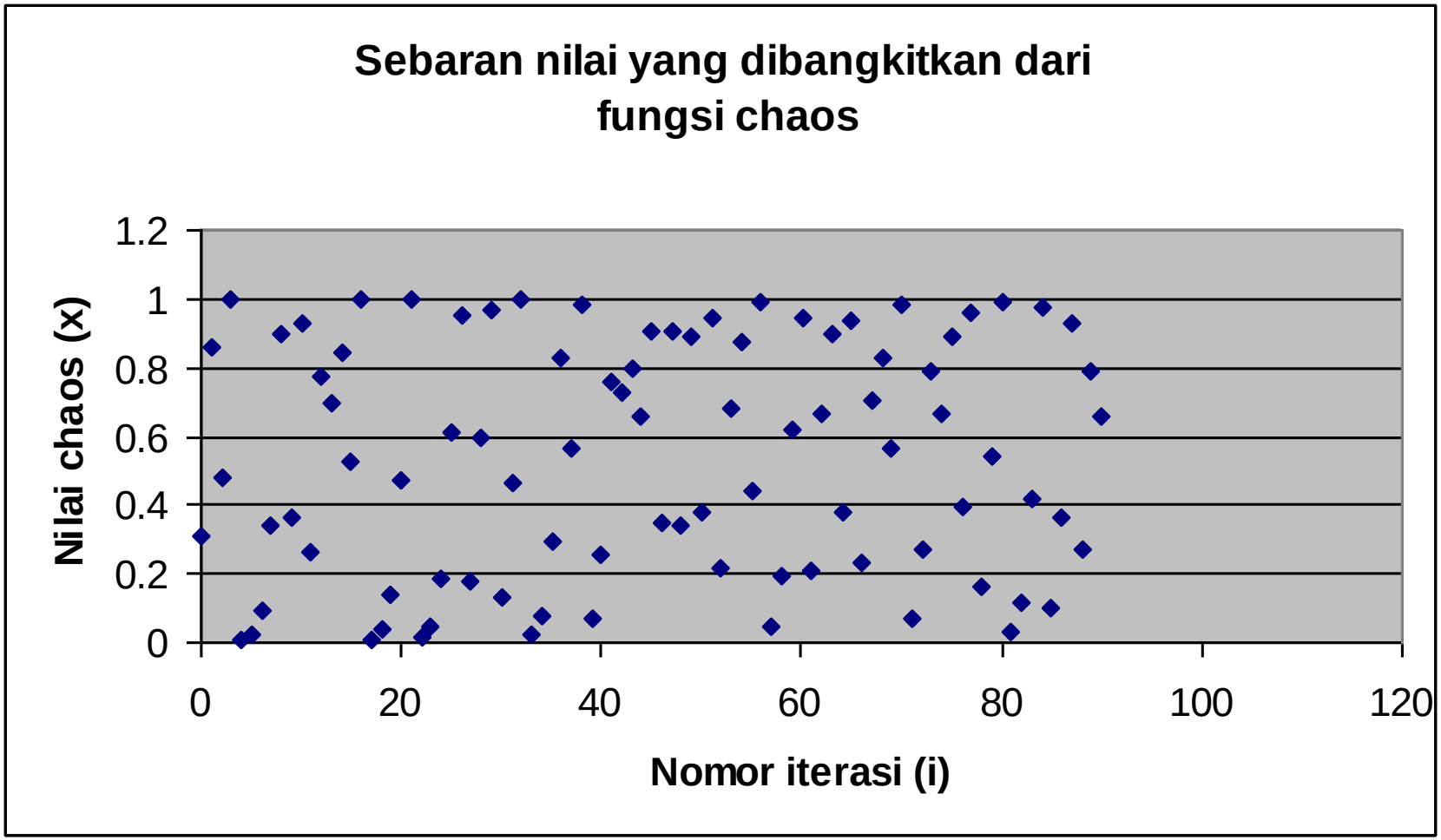
Teori *Chaos*

```
double f(double x, int iterasi)
/* menghitung barisan chaotik berikutnya */
{
 int i;

 for (i = 1; i <= iterasi; i++) {
   x = r * x * (1 - x);
 }
 return x;
}

printf("Ketikkan nilai awal (0 s/d 1) : ");
scanf("%lf", &x);
while ((p = getc(Fin)) != EOF)
{
  x = f(x, iterasi);       /* hitung nilai chaotik berikutnya */
  iterasi = iterasi + 10;  /* tentukan jumlah iterasi berikutnya */
}
```

Teori *Chaos*


Sebaran nilai yang dibangkitkan dari fungsi chaos
Nilai chaos (x)
1.2
1
0.8
0.6
0.4
0.2
0
0
20
40
60
80
100
120
Nomor iterasi (i)

Teori *Chaos*

- Contoh *chaos map* lainnya:

  1. *Henon map*

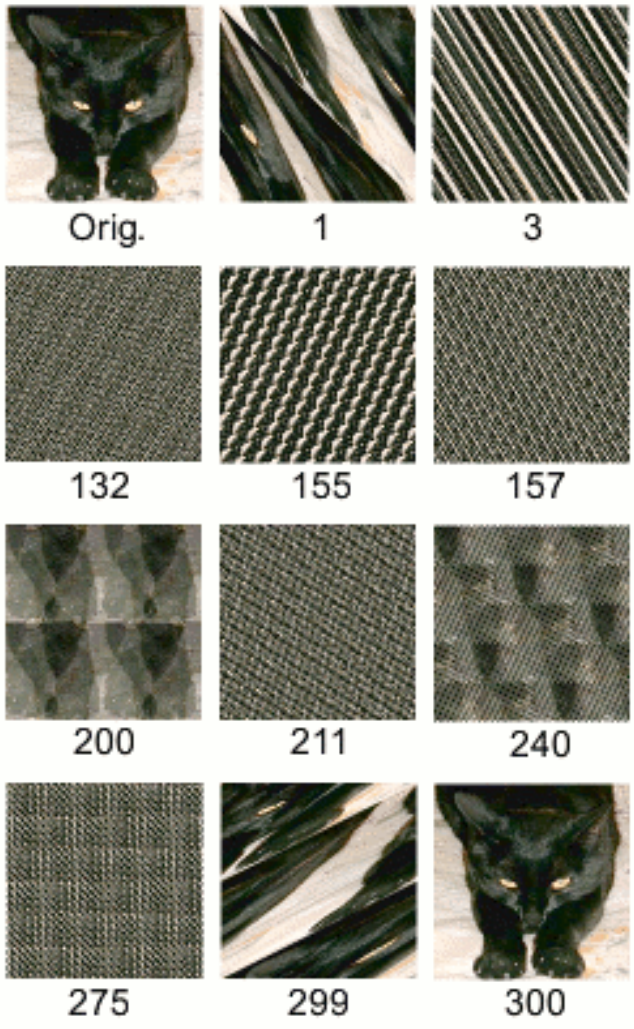
$$x_n = 1 + b(x_{n-2} - x_{n-3}) + cx^2{}_{n-2}$$

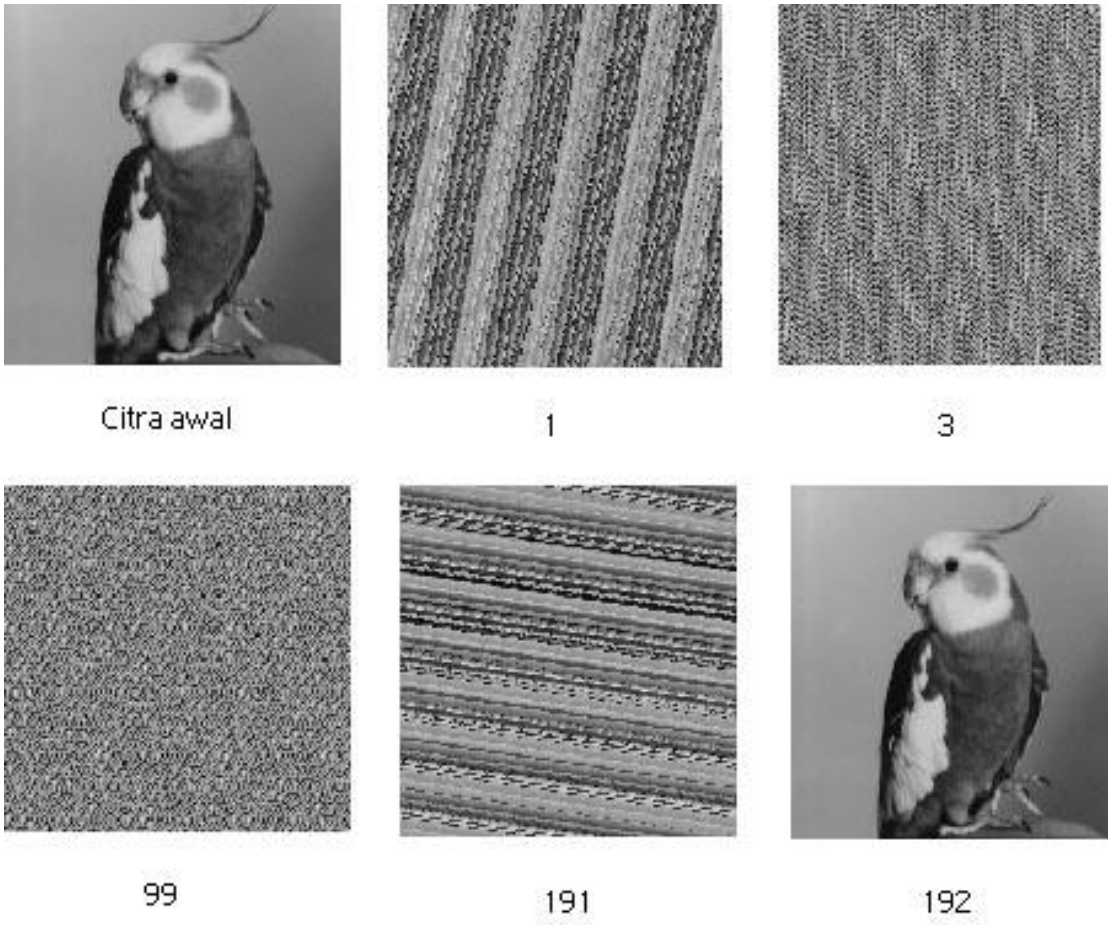
  2. *Arnold's cat map*:

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

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

Hasil lelaran dengan Arnold Cat Map




Citra awal
1
3
99
191
192