Generatori de numere pseudo - aleatoare

Prof. Dr. Adrian Atanasiu

Universitatea București

April 4, 2011

În general nu se poate vorbi de un singur număr aleator decât într-un context statistic.

Termenul corect este acela de *șir de numere aleatoare*.

În general nu se poate vorbi de un singur număr aleator decât într-un context statistic.
Termenul corect este acela de *șir de numere aleatoare*.

Folosirea calculatorului reduce termenul de *numere aleatoare* la un șir de biți generați aleator, grupați după o anumită regulă.

În general nu se poate vorbi de un singur număr aleator decât într-un context statistic.
Termenul corect este acela de *șir de numere aleatoare*.

Folosirea calculatorului reduce termenul de *numere aleatoare* la un șir de biți generați aleator, grupați după o anumită regulă.

Nu există o modalitate mai scurtă de a specifica șirul decât secvența însăși.

Definiție

Un generator de numere aleatoare, sau "generator real aleator" (GA) este un dispozitiv sau un algoritm care produce o secvență de numere independente și nepredictibile.

Definiție

Un generator de numere aleatoare, sau "generator real aleator" (GA) este un dispozitiv sau un algoritm care produce o secvență de numere independente și nepredictibile.

În majoritatea cazurilor, un *GA* generează secvențe binare, care ulterior sunt convertite în numere întregi.

În criptografie este esențial ca un număr aleator să nu poată fi aflat.

Un număr perfect aleator este acela pe care *Oscar* nu-l poate ghici decât prin forță brută.

În criptografie este esențial ca un număr aleator să nu poată fi aflat.
Un număr perfect aleator este acela pe care *Oscar* nu-l poate ghici decât prin forță brută.
Multe atacuri se bazează pe exploatarea imperfecțiunilor unor funcții care generează numere aleatoare.

În criptografie este esențial ca un număr aleator să nu poată fi aflat.
Un număr perfect aleator este acela pe care *Oscar* nu-l poate ghici decât prin forță brută.
Multe atacuri se bazează pe exploatarea imperfecțiunilor unor funcții care generează numere aleatoare.

O generare de numere pur aleatoare se realizează prin colectarea și procesarea de date obținute dintr-o sursă de entropie exterioară calculatorului.
Surse de entropie: variațiile mișcării mouse-ului; intervalul de timp dintre apăsarea a două taste; surse radioactive sau bazate pe zgomote din atmosferă.

Proprietatea de a fi aleator a fost introdusă în domeniul tehnicii de calcul cu ajutorul generatorilor de **numere pseudo-aleatoare**.

Proprietatea de a fi aleator a fost introdusă în domeniul tehnicii de calcul cu ajutorul generatorilor de **numere pseudo-aleatoare**.

Definiție

Fie m, k $(0 < k < m)$ numere întregi. Un (k, m) generator de numere pseudo-aleatoare este o aplicație recursivă

$$f : \mathbb{Z}_2^k \longleftarrow \mathbb{Z}_2^m$$

calculabilă în timp polinomial.

Proprietatea de a fi aleator a fost introdusă în domeniul tehnicii de calcul cu ajutorul generatorilor de **numere pseudo-aleatoare**.

Definiție

Fie m, k $(0 < k < m)$ numere întregi. Un (k, m) generator de numere pseudo-aleatoare este o aplicație recursivă

$$f : \mathbb{Z}_2^k \longleftarrow \mathbb{Z}_2^m$$

calculabilă în timp polinomial.

În aplicații, $m = h(k)$ unde h este o funcție polinomială.

Un generator de numere pseudo-aleatoare trebuie:

Un generator de numere pseudo-aleatoare trebuie:

- Să fie simplu și rapid.

Un generator de numere pseudo-aleatoare trebuie:

- Să fie simplu și rapid.
- Să producă șiruri de numere de lungime arbitrară care să nu conțină repetiții.

Un generator de numere pseudo-aleatoare trebuie:

- Să fie simplu și rapid.
- Să producă șiruri de numere de lungime arbitrară care să nu conțină repetiții.
  În calculator nu se poate construi un generator cu perioadă infinită. Generatorul trebuie să aibă totuși o perioadă de repetiție cât mai mare.

Un generator de numere pseudo-aleatoare trebuie:

- Să fie simplu și rapid.
- Să producă șiruri de numere de lungime arbitrară care să nu conțină repetiții.
  În calculator nu se poate construi un generator cu perioadă infinită. Generatorul trebuie să aibă totuși o perioadă de repetiție cât mai mare.
- Să producă numere independente unul de altul (sau cu o corelare cât mai vagă).

Un generator de numere pseudo-aleatoare trebuie:

- Să fie simplu și rapid.
- Să producă șiruri de numere de lungime arbitrară care să nu conțină repetiții.
  În calculator nu se poate construi un generator cu perioadă infinită. Generatorul trebuie să aibă totuși o perioadă de repetiție cât mai mare.
- Să producă numere independente unul de altul (sau cu o corelare cât mai vagă).
- Să genereze numere cu o repartiție uniformă în spațiul valorilor.

Generatori liniari congruențiali

Lehmer (1949). Este definit

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

Generatori liniari congruențiali

Lehmer (1949). Este definit

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

Valorile a (*multiplicatorul*), b (*incrementul*) și m (*modulul*) sunt constante.

Generatori liniari congruențiali

Lehmer (1949). Este definit

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

Valorile a (*multiplicatorul*), b (*incrementul*) și m (*modulul*) sunt constante.
Cheia de generare este valoarea inițială x_0.

Generatori liniari congruențiali

Lehmer (1949). Este definit

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

Valorile a (*multiplicatorul*), b (*incrementul*) și m (*modulul*) sunt constante.
Cheia de generare este valoarea inițială x_0.

Când $b = 0$, generatorul este *multiplicativ*.

Generatori liniari congruențiali

Lehmer (1949). Este definit

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

Valorile a (*multiplicatorul*), b (*incrementul*) și m (*modulul*) sunt constante.
Cheia de generare este valoarea inițială x_0.

Când $b = 0$, generatorul este *multiplicativ*.

Perioada maximă a unui generator liniar este m.
Ea poate fi atinsă pentru anumite valori ale perechii (a, b) (de exemplu dacă $cmmdc(b, m) = 1$).

Un generator liniar congruențial de perioadă m se numește *generator de perioadă maximală*.

Un generator liniar congruențial de perioadă m se numește *generator de perioadă maximală*.
Câțiva generatori de perioadă maximală.

a	b	m	a	b	m
105	1283	6075	1277	24749	117128
211	1663	7875	2311	25367	120050
421	1663	7875	3877	29573	139968
430	2531	11979	8121	28411	134456
171	11213	53125	9301	49297	233280
141	28411	134456	2416	374441	1771875
421	17117	81000	17221	107839	510300
1093	18257	86436	84589	45989	217728

Un generator liniar congruențial de perioadă m se numește *generator de perioadă maximală*.
Câțiva generatori de perioadă maximală.

a	b	m	a	b	m
105	1283	6075	1277	24749	117128
211	1663	7875	2311	25367	120050
421	1663	7875	3877	29573	139968
430	2531	11979	8121	28411	134456
171	11213	53125	9301	49297	233280
141	28411	134456	2416	374441	1771875
421	17117	81000	17221	107839	510300
1093	18257	86436	84589	45989	217728

Avantaj: viteză mare de calcul.

Securitate

Criptanaliza a fost realizată de Jim Reeds în 1977 și Joan Boyar în 1982.

Securitate

Criptanaliza a fost realizată de Jim Reeds în 1977 și Joan Boyar în 1982.
Ea a spart și generatorii pătratici

$$x_{n+1} = (ax_n^2 + bx_n + c) \pmod m$$

și cubici

$$x_{n+1} = (ax_n^3 + bx_n^2 + cx_n + d) \pmod m$$

Securitate

Criptanaliza a fost realizată de Jim Reeds în 1977 și Joan Boyar în 1982.
Ea a spart și generatorii pătratici

$$x_{n+1} = (ax_n^2 + bx_n + c) \pmod m$$

și cubici

$$x_{n+1} = (ax_n^3 + bx_n^2 + cx_n + d) \pmod m$$

Alți cercetători au extins metodele de atac pentru spargerea oricărui generator polinomial congruențial.

Securitate

Criptanaliza a fost realizată de Jim Reeds în 1977 și Joan Boyar în 1982.
Ea a spart și generatorii pătratici

$$x_{n+1} = (ax_n^2 + bx_n + c) \pmod m$$

și cubici

$$x_{n+1} = (ax_n^3 + bx_n^2 + cx_n + d) \pmod m$$

Alți cercetători au extins metodele de atac pentru spargerea oricărui generator polinomial congruențial.
Generatorii liniar congruențiali sunt folosiți cu predilecție în aplicații necriptografice; de exemplu, în simulare ei asigură o comportare statistică bună în majoritatea testelor.

Generatori *Ranrot*

Au fost definiți de Agner Fog în 1997, inițial pentru algoritmi de tip *Monte Carlo*.

Generatori *Ranrot*

Au fost definiți de Agner Fog în 1997, inițial pentru algoritmi de tip *Monte Carlo*.

Folosește generatori de numere Fibonacci, cu deplasare ciclică pe biți.

Generatori *Ranrot*

Au fost definiți de Agner Fog în 1997, inițial pentru algoritmi de tip *Monte Carlo*.
Folosește generatori de numere Fibonacci, cu deplasare ciclică pe biți.

- **Tip A**: $x_n = ((x_{n-j} + x_{n-k}) \pmod{2^b}) \gg r;$

Generatori *Ranrot*

Au fost definiți de Agner Fog în 1997, inițial pentru algoritmi de tip *Monte Carlo*.
Folosește generatori de numere Fibonacci, cu deplasare ciclică pe biți.

- **Tip A**: $x_n = ((x_{n-j} + x_{n-k}) \ (mod\ 2^b)) \gg r;$
- **Tip B**:

$$x_n = ((x_{n-j} \gg r_1) + (x_{n-k} \gg r_2)) \ (mod\ 2^b);$$

Generatori *Ranrot*

Au fost definiți de Agner Fog în 1997, inițial pentru algoritmi de tip *Monte Carlo*.
Folosește generatori de numere Fibonacci, cu deplasare ciclică pe biți.

- **Tip A**: $x_n = ((x_{n-j} + x_{n-k}) \ (mod\ 2^b)) \gg r;$
- **Tip B**:
  $$x_n = ((x_{n-j} \gg r_1) + (x_{n-k} \gg r_2)) \ (mod\ 2^b);$$
- **Tip B3**:
  $$x_n = ((x_{n-i} \gg r_1) + (x_{n-j} \gg r_2) + (x_{n-k} \gg r_3)) \ (mod\ 2^b);$$

Generatori *Ranrot*

Au fost definiți de Agner Fog în 1997, inițial pentru algoritmi de tip *Monte Carlo*.
Folosește generatori de numere Fibonacci, cu deplasare ciclică pe biți.

- **Tip A**: $x_n = ((x_{n-j} + x_{n-k}) \ (mod\ 2^b)) \gg r;$
- **Tip B**:
  $x_n = ((x_{n-j} \gg r_1) + (x_{n-k} \gg r_2)) \ (mod\ 2^b);$
- **Tip B3**:
  $x_n = ((x_{n-i} \gg r_1) + (x_{n-j} \gg r_2) + (x_{n-k} \gg r_3)) \ (mod\ 2^b);$
- **Tip W**:
  $z_n = ((y_{n-j} \gg r_3) + (y_{n-k} \gg r_1)) \ (mod\ 2^{b/2}),$
  $y_n = ((z_{n-j} \gg r_4) + (z_{n-k} \gg r_2)) \ (mod\ 2^{b/2}),$
  $x_n = y_n + z_n \cdot 2^{b/2}.$

Exemplu

Un generator *Ranrot* de tipul A cu $j = 1, k = 4, b = 7, r = 4$ are 24 cicluri de perioade
$1, 5, 9, 11, 14, 21, 129, 6576, 8854, 16124, 17689, 135756, 310417, 392239, 488483, 1126126, 1355840, 1965955, 4576377, 7402465, 8393724, 57549556, 184256986.$

Securitate

Generatorii *Ranrot* sunt ușor de spart, deoarece starea inițială se poate deduce ușor din k valori consecutive ale lui x.

Securitate

Generatorii *Ranrot* sunt ușor de spart, deoarece starea inițială se poate deduce ușor din k valori consecutive ale lui x.

Dacă însă parametrii nu se cunosc, generatorii *Ranrot* pot fi folosiți cu succes în criptografie, având o securitate sporită.

Securitate

Generatorii *Ranrot* sunt ușor de spart, deoarece starea inițială se poate deduce ușor din k valori consecutive ale lui x.

Dacă însă parametrii nu se cunosc, generatorii *Ranrot* pot fi folosiți cu succes în criptografie, având o securitate sporită.

Există un sistem de criptare (*Power Crypto*) bazat pe generatorul *Ranrot* de tip $B3$.

BBS

Cel mai simplu și – se pare – cel mai eficient generator de numere pseudo - aleatoare este: Blum-Blum-Shub (*BBS*), numit și *generator rezidual pătratic*.

BBS

Cel mai simplu și – se pare – cel mai eficient generator de numere pseudo - aleatoare este: Blum-Blum-Shub (*BBS*), numit și *generator rezidual pătratic*.

Definiție

Fie p, q două numere prime. Dacă

$$p \equiv 3 \ (mod\ 4), \qquad q \equiv 3 \ (mod\ 4)$$

atunci numărul $n = pq$ se numește întreg Blum.

Fie $n = pq$ un întreg Blum, unde p, q sunt numere prime pe $k/2$ biți.

Fie $n = pq$ un întreg Blum, unde p, q sunt numere prime pe $k/2$ biți.
Fie x_0 un reziduu pătratic modulo n.
Se definește secvența

$$x_{i+1} = x_i^2 \pmod{n}$$

Fie $n = pq$ un întreg Blum, unde p, q sunt numere prime pe $k/2$ biți.
Fie x_0 un reziduu pătratic modulo n.
Se definește secvența

$$x_{i+1} = x_i^2 \pmod n$$

Dacă $z_i = x_i \ (mod\ 2)$ pentru $1 \leq i \leq m$, atunci numărul aleator generat este

$$f(x_0) = z_1 z_2 \dots z_m.$$

Fie $n = pq$ un întreg Blum, unde p, q sunt numere prime pe $k/2$ biți.
Fie x_0 un reziduu pătratic modulo n.
Se definește secvența

$$x_{i+1} = x_i^2 \pmod n$$

Dacă $z_i = x_i \ (mod\ 2)$ pentru $1 \leq i \leq m$, atunci numărul aleator generat este

$$f(x_0) = z_1 z_2 \ldots z_m.$$

Biții z_i $(1 \leq i \leq m)$ se pot calcula direct:

$$z_i = x_0^{2^i \ (mod\ (p-1)(q-1))} \ (mod\ 2)$$

Exemplu

Fie $p = 383,\ q = 503$; deci $n = 192649$.

Exemplu

Fie $p = 383$, $q = 503$; deci $n = 192649$.
Alegând $x_0 = 101355^2 = 20749$ $(mod\ n)$, generatorul *BBS* va produce

$$11001110000100111 0$$

Exemplu

Fie $p = 383,\ q = 503$; deci $n = 192649$.
Alegând $x_0 = 101355^2 = 20749\ (mod\ n)$, generatorul *BBS* va produce

$$110011100001001110$$

i	0	1	2	3	4	5
x_i	20749	143135	177671	97048	89992	174051
z_i	–	1	1	0	0	1

i	6	7	8	9	10	11
x_i	80649	45663	69442	186894	177046	137922
z_i	1	1	0	0	0	0

i	12	13	14	15	16	17	18
x_i	123175	8630	114386	14863	133015	106065	45870
z_i	1	0	0	1	1	1	0

Securitate

Se bazează pe dificultatea factorizării lui n.

Securitate

Se bazează pe dificultatea factorizării lui n.
Mai mult, fiind dată o parte a secvenței, nu există nici o modalitate de a prezice bitul anterior sau cel ulterior secvenței.

Securitate

Se bazează pe dificultatea factorizării lui n.
Mai mult, fiind dată o parte a secvenței, nu există nici o modalitate de a prezice bitul anterior sau cel ulterior secvenței.

Algoritmul *BBS* este destul de lent, dar are unele implementări mai rapide.
Astfel, dacă n este lungimea lui x_i, pot fi păstrați ultimii $\lfloor log_2 x_i \rfloor$ biți.

Securitate

Se bazează pe dificultatea factorizării lui n.
Mai mult, fiind dată o parte a secvenței, nu există nici o modalitate de a prezice bitul anterior sau cel ulterior secvenței.

Algoritmul *BBS* este destul de lent, dar are unele implementări mai rapide.
Astfel, dacă n este lungimea lui x_i, pot fi păstrați ultimii $\lfloor log_2 x_i \rfloor$ biți.

În acest moment *BBS* este considerat cel mai bun generator de numere pseudo-aleatoare pentru protocoale de generare și distribuție a cheii.

LFSR

Circuitele liniare (Shift Registers) sunt folosite în teoria codurilor detectoare și corectoare de erori precum și în unele sisteme de criptare bloc.
Asigură viteză mare de calcul.

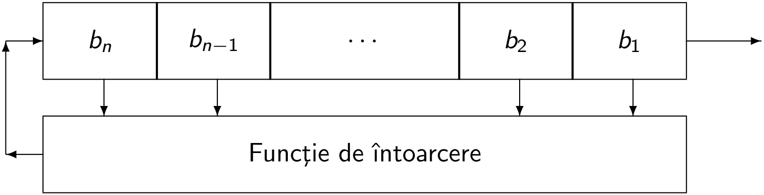
LFSR

Circuitele liniare (Shift Registers) sunt folosite în teoria codurilor detectoare și corectoare de erori precum și în unele sisteme de criptare bloc.
Asigură viteză mare de calcul.

Un *LFSR* (*Linear Feedback Shift Register*) este un circuit liniar format dintr-un registru serial și o funcție de întoarcere (*feedback*).

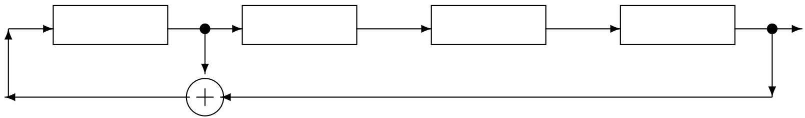
Dacă registrul este compus din n flip-flopuri de date $(DF - F)$, avem un $n - LFSR$.

Dacă registrul este compus din n flip-flopuri de date $(DF - F)$, avem un $n - LFSR$.

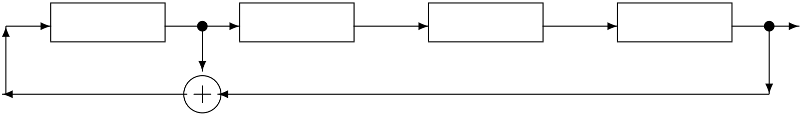


Funcția de întoarcere este un XOR între anumiți biți din registru.

Exemplu

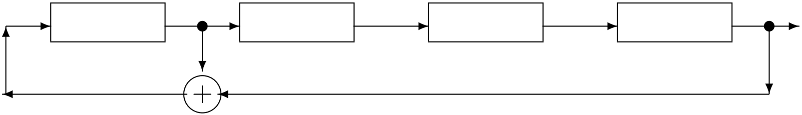


Exemplu



Să presupunem că inițial cei patru biți din registru sunt 1001.

Exemplu

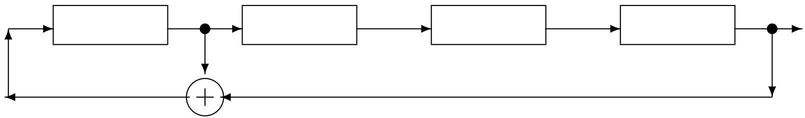


Să presupunem că inițial cei patru biți din registru sunt 1001.
La fiecare tact se obține o nouă configurație:

0100, 0010, 0001, 1000, 1100, 1110, 1111, 0111, 1011, 0101, 1010, 1101, 0110, 0011

după care apare din nou 1001.

Exemplu



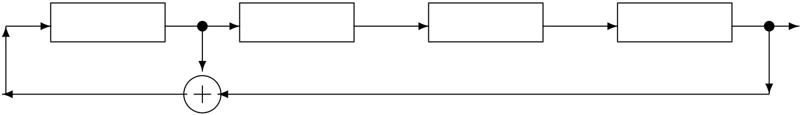
Să presupunem că inițial cei patru biți din registru sunt 1001.
La fiecare tact se obține o nouă configurație:

0100, 0010, 0001, 1000, 1100, 1110, 1111, 0111, 1011, 0101, 1010, 1101, 0110, 0011

după care apare din nou 1001.
La ieșire apare

$$100100011110101 1$$

Exemplu



Să presupunem că inițial cei patru biți din registru sunt 1001.
La fiecare tact se obține o nouă configurație:

0100, 0010, 0001, 1000, 1100, 1110, 1111, 0111, 1011, 0101, 1010, 1101, 0110, 0011

după care apare din nou 1001.
La ieșire apare

$$1001000111101011$$

Este un generator de perioadă 16.

Proprietăți

- Un $n - LFSR$ poate avea maxim $2^n - 1$ stări distincte.

Proprietăți

- Un $n - LFSR$ poate avea maxim $2^n - 1$ stări distincte.
- Fie $g(X) = 1 + g_1X + \cdots + g_nX^n \in \mathbb{Z}_2[X]$ polinomul asociat unui $n - LFSR$, unde $g_i = 1$ dacă și numai dacă bitul i participă la o adunare modulo 2.

Proprietăți

- Un $n - LFSR$ poate avea maxim $2^n - 1$ stări distincte.
- Fie $g(X) = 1 + g_1X + \cdots + g_nX^n \in \mathbb{Z}_2[X]$ polinomul asociat unui $n - LFSR$, unde $g_i = 1$ dacă și numai dacă bitul i participă la o adunare modulo 2.
- Fie polinomul $g(X) \in \mathbb{Z}_2[X]$, $grad(g(X)) = n$ și m cel mai mic număr astfel ca $g(X)|X^m + 1$.
  Atunci secvența binară generată de un $n - LFSR$ asociat lui $g(X)$ are perioada m.

Proprietăți

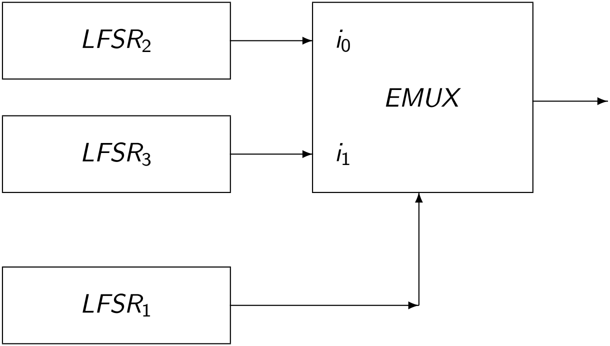
- Un $n - LFSR$ poate avea maxim $2^n - 1$ stări distincte.
- Fie $g(X) = 1 + g_1X + \cdots + g_nX^n \in \mathbb{Z}_2[X]$ polinomul asociat unui $n - LFSR$, unde $g_i = 1$ dacă și numai dacă bitul i participă la o adunare modulo 2.
- Fie polinomul $g(X) \in \mathbb{Z}_2[X]$, $grad(g(X)) = n$ și m cel mai mic număr astfel ca $g(X)|X^m + 1$.
  Atunci secvența binară generată de un $n - LFSR$ asociat lui $g(X)$ are perioada m.
- Dacă $g(X)$ este un polinom ireductibil peste $\mathbb{Z}_2$, atunci $m = 2^n - 1$ (maxim).

Geffe

Combină într-o formă neliniară trei *LFSR*:

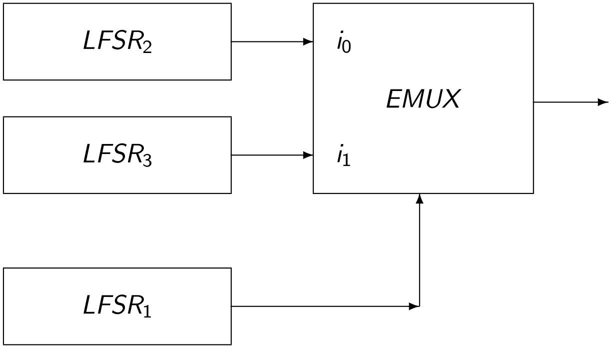
Geffe

Combină într-o formă neliniară trei *LFSR*:



Geffe

Combină într-o formă neliniară trei *LFSR*:



Dacă a_1, a_2, a_3 sunt ieșirile din *LFSR*-uri, ieșirea din generatorul Geffe este

$$b = (a_1 \wedge a_2) \oplus (\overline{a}_1 \wedge a_3)$$

Perioada generatorului este cel mai mic multiplu comun al perioadelor celor trei *LFSR*-uri.

Deci, dacă cele trei polinoame care definesc circuitele au grade prime între ele, perioada generatorului Geffe este produsul celor trei perioade.

Securitate

Generatorul *Geffe* nu rezistă unui atac *prin corelare*.

Securitate

Generatorul *Geffe* nu rezistă unui atac *prin corelare*.
Ieșirea din generator coincide cu ieșirea din $LFSR_2$ cam 75% din timp.

Securitate

Generatorul *Geffe* nu rezistă unui atac *prin corelare*.
Ieșirea din generator coincide cu ieșirea din $LFSR_2$ cam 75% din timp.
Deci, dacă definițiile polinomiale ale circuitelor sunt cunoscute se poate ghici valoarea inițială din $LFSR_2$ și genera secvența sa de ieșire.

Securitate

Generatorul *Geffe* nu rezistă unui atac *prin corelare*.

Ieșirea din generator coincide cu ieșirea din $LFSR_2$ cam 75% din timp.

Deci, dacă definițiile polinomiale ale circuitelor sunt cunoscute se poate ghici valoarea inițială din $LFSR_2$ și genera secvența sa de ieșire.

Apoi se numără de câte ori ieșirea din $LFSR_2$ coincide cu ieșirea din generator.

Securitate

Generatorul *Geffe* nu rezistă unui atac *prin corelare*.
Ieșirea din generator coincide cu ieșirea din $LFSR_2$ cam 75% din timp.
Deci, dacă definițiile polinomiale ale circuitelor sunt cunoscute se poate ghici valoarea inițială din $LFSR_2$ și genera secvența sa de ieșire.
Apoi se numără de câte ori ieșirea din $LFSR_2$ coincide cu ieșirea din generator.
Dacă nu s-a ghicit corect, cele două secvențe coincid cam 50%; dacă s-a ghicit corect, ele coincid cam 75%.

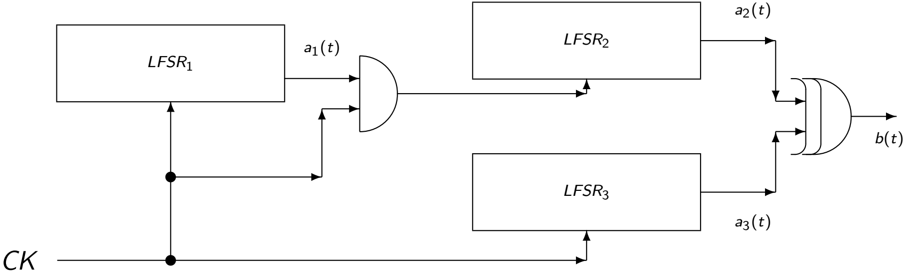
Securitate

Generatorul *Geffe* nu rezistă unui atac *prin corelare*.
Ieșirea din generator coincide cu ieșirea din $LFSR_2$ cam 75% din timp.
Deci, dacă definițiile polinomiale ale circuitelor sunt cunoscute se poate ghici valoarea inițială din $LFSR_2$ și genera secvența sa de ieșire.
Apoi se numără de câte ori ieșirea din $LFSR_2$ coincide cu ieșirea din generator.
Dacă nu s-a ghicit corect, cele două secvențe coincid cam 50%; dacă s-a ghicit corect, ele coincid cam 75%.
Similar, ieșirea generatorului coincide cu cea din $LFSR_3$ cam 75% din timp.

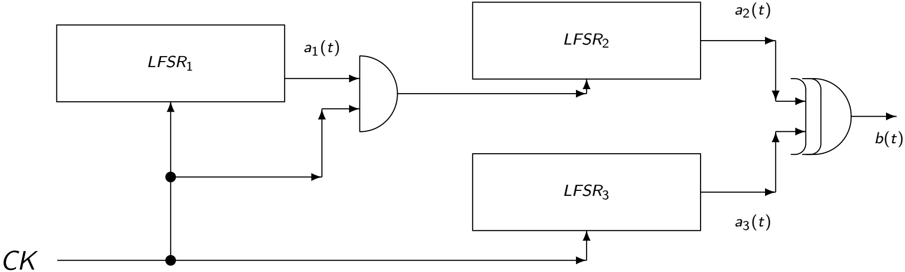
Securitate

Generatorul *Geffe* nu rezistă unui atac *prin corelare*.
Ieșirea din generator coincide cu ieșirea din $LFSR_2$ cam 75% din timp.
Deci, dacă definițiile polinomiale ale circuitelor sunt cunoscute se poate ghici valoarea inițială din $LFSR_2$ și genera secvența sa de ieșire.
Apoi se numără de câte ori ieșirea din $LFSR_2$ coincide cu ieșirea din generator.
Dacă nu s-a ghicit corect, cele două secvențe coincid cam 50%; dacă s-a ghicit corect, ele coincid cam 75%.
Similar, ieșirea generatorului coincide cu cea din $LFSR_3$ cam 75% din timp.
Cu aceste corelări secvența poate fi ghicită complet.

Beth - Piper

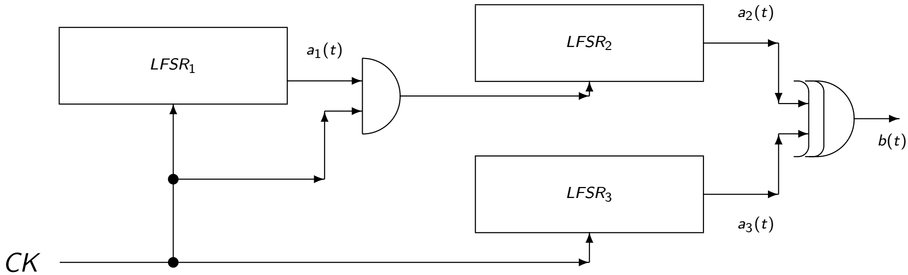


Beth - Piper



Generatorul controlează ceasurile celor trei circuite.
Ceasul de intrare în $LFSR_2$ este controlat de ieșirea din $LFSR_1$.
$LFSR_2$ schimbă starea la momentul t numai dacă ieșirea din $LFSR_1$ la momentul $t-1$ a fost 1.

Beth - Piper



Generatorul controlează ceasurile celor trei circuite.

Ceasul de intrare în $LFSR_2$ este controlat de ieșirea din $LFSR_1$.

$LFSR_2$ schimbă starea la momentul t numai dacă ieșirea din $LFSR_1$ la momentul $t-1$ a fost 1.

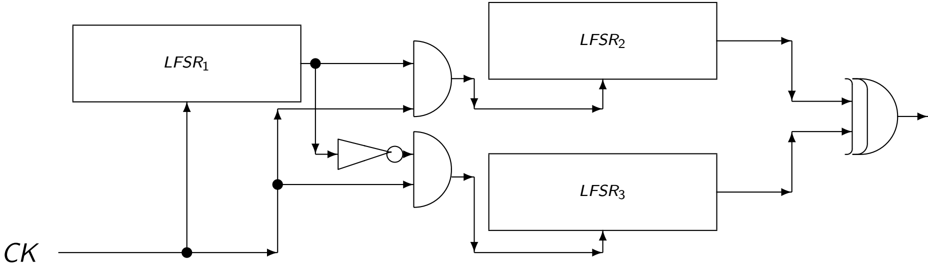
Nu rezistă atacurilor prin corelare.

Stop-and-Go alternativ

Folosește trei *LFSR* de lungimi diferite:

Stop-and-Go alternativ

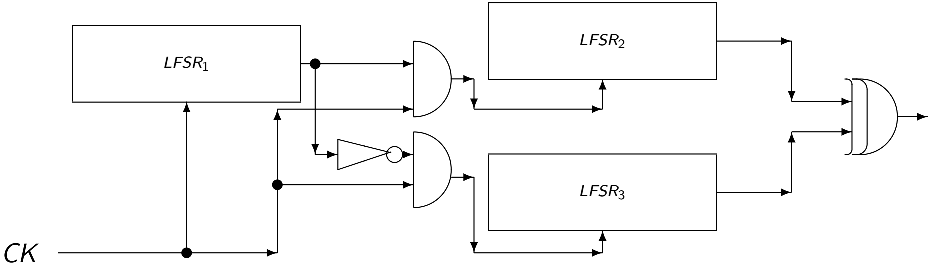
Folosește trei *LFSR* de lungimi diferite:



Când ieșirea din $LFSR_1$ este 1 ea activează $LFSR_2$; în caz contrar este activat $LFSR_3$.

Stop-and-Go alternativ

Folosește trei *LFSR* de lungimi diferite:



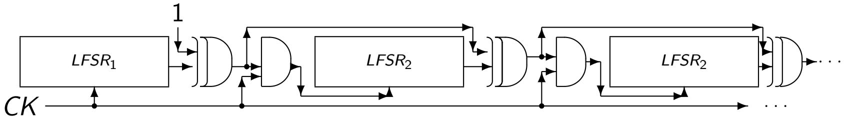
Când ieșirea din $LFSR_1$ este 1 ea activează $LFSR_2$; în caz contrar este activat $LFSR_3$.
Există un atac prin corelare asupra sa (mai precis asupra $LFSR_1$), dar acesta nu i-a slăbit substanțial securitatea.

Generatorul Gollmann

Extensie "în cascadă" a mai multor circuite *LFSR*, ceasul fiecărui *LFSR* fiind controlat de circuitul anterior.

Generatorul Gollmann

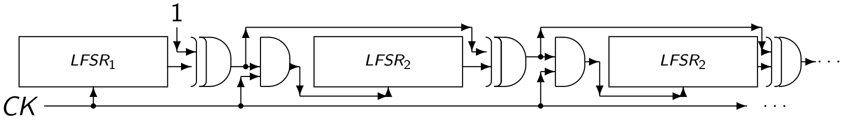
Extensie "în cascadă" a mai multor circuite *LFSR*, ceasul fiecărui *LFSR* fiind controlat de circuitul anterior.



Dacă la momentul $t-1$ ieșirea din $LFSR_i$ este 1, atunci la momentul t este activat $LFSR_{i+1}$.

Generatorul Gollmann

Extensie "în cascadă" a mai multor circuite *LFSR*, ceasul fiecărui *LFSR* fiind controlat de circuitul anterior.



Dacă la momentul $t-1$ ieșirea din $LFSR_i$ este 1, atunci la momentul t este activat $LFSR_{i+1}$.

Dacă toate circuitele liniare au aceiași lungime n, complexitatea unui generator *Gollmann* cu k *LFSR*-uri este $n \cdot (2^n - 1)^{k-1}$.

Blum - Micali

Fie g un număr prim, p un număr prim impar și x_0 o valoare inițială.

Blum - Micali

Fie g un număr prim, p un număr prim impar și x_0 o valoare inițială.
Se generează numerele

$$x_{i+1} = g^{x_i} \quad (mod\ p)$$

Blum - Micali

Fie g un număr prim, p un număr prim impar și x_0 o valoare inițială.
Se generează numerele

$$x_{i+1} = g^{x_i} \pmod p$$

Ieșirea din generator este 1 dacă $x_i < \frac{p-1}{2}$ și 0 altfel.

Blum - Micali

Fie g un număr prim, p un număr prim impar și x_0 o valoare inițială.
Se generează numerele

$$x_{i+1} = g^{x_i} \quad (mod\ p)$$

Ieșirea din generator este 1 dacă $x_i < \dfrac{p-1}{2}$ și 0 altfel.

Securitatea acestui sistem se bazează pe problema logaritmului discret.
Dacă p este suficient de mare astfel ca problema logaritmului discret să fie dificilă, generatorul este considerat sigur.

Generatorul *RSA*

Fie $n = pq$ un modul obținut prin produsul a două numere prime mari, un număr e astfel ca $cmmdc(e, (p-1)\cdot(q-1)) = 1$ și x_0 o valoare inițială ($x_0 < n$).

Generatorul *RSA*

Fie $n = pq$ un modul obținut prin produsul a două numere prime mari, un număr e astfel ca $cmmdc(e, (p-1) \cdot (q-1)) = 1$ și x_0 o valoare inițială ($x_0 < n$).
Se definește

$$x_{i+1} = x_i^e \pmod n$$

Generatorul *RSA*

Fie $n = pq$ un modul obținut prin produsul a două numere prime mari, un număr e astfel ca $cmmdc(e, (p-1) \cdot (q-1)) = 1$ și x_0 o valoare inițială ($x_0 < n$).
Se definește

$$x_{i+1} = x_i^e \pmod{n}$$

Ieșirea din generator este $z_i = x_i \ (mod\ 2)$.

Generatorul *RSA*

Fie $n = pq$ un modul obținut prin produsul a două numere prime mari, un număr e astfel ca $cmmdc(e, (p-1)\cdot(q-1)) = 1$ și x_0 o valoare inițială ($x_0 < n$).
Se definește

$$x_{i+1} = x_i^e \pmod{n}$$

Ieșirea din generator este $z_i = x_i \ (mod\ 2)$.

Securitatea generatorului se bazează pe dificultatea spargerii sistemului *RSA*.
Dacă n este suficient de mare, sistemul este considerat sigur.

Mother-of-all

Propus de *George Marsaglia*, fiind extrem de rapid în implementare pe limbaj de asamblare.

Mother-of-all

Propus de *George Marsaglia*, fiind extrem de rapid în implementare pe limbaj de asamblare.
Se aleg cinci numere întregi x_0, x_1, x_2, x_3, c (nu toate nule), fiecare de 32 biți.

Mother-of-all

Propus de *George Marsaglia*, fiind extrem de rapid în implementare pe limbaj de asamblare.
Se aleg cinci numere întregi x_0, x_1, x_2, x_3, c (nu toate nule), fiecare de 32 biți.

1. $n \longleftarrow 4$;

Mother-of-all

Propus de *George Marsaglia*, fiind extrem de rapid în implementare pe limbaj de asamblare.
Se aleg cinci numere întregi x_0, x_1, x_2, x_3, c (nu toate nule), fiecare de 32 biți.

1. $n \longleftarrow 4$;
2. **while** $n \leq MAX$ **do**
    1. $S \longleftarrow 2111111111 \cdot x_{n-4} + 1492 \cdot x_{n-3} + 1776 \cdot x_{n-2} + 5115 \cdot x_{n-1} + c$;

Mother-of-all

Propus de *George Marsaglia*, fiind extrem de rapid în implementare pe limbaj de asamblare.
Se aleg cinci numere întregi x_0, x_1, x_2, x_3, c (nu toate nule), fiecare de 32 biți.

1. $n \longleftarrow 4;$
2. **while** $n \leq MAX$ **do**
    1. $S \longleftarrow 2111111111 \cdot x_{n-4} + 1492 \cdot x_{n-3} + 1776 \cdot x_{n-2} + 5115 \cdot x_{n-1} + c;$
    2. $x_n \longleftarrow S \ (mod\ 2^{32}), \qquad c \longleftarrow \left\lfloor \frac{S}{2^{32}} \right\rfloor;$

Mother-of-all

Propus de *George Marsaglia*, fiind extrem de rapid în implementare pe limbaj de asamblare.
Se aleg cinci numere întregi x_0, x_1, x_2, x_3, c (nu toate nule), fiecare de 32 biți.

1. $n \longleftarrow 4;$
2. **while** $n \leq MAX$ **do**
   1. $S \longleftarrow 2111111111 \cdot x_{n-4} + 1492 \cdot x_{n-3} + 1776 \cdot x_{n-2} + 5115 \cdot x_{n-1} + c;$
   2. $x_n \longleftarrow S \ (mod\ 2^{32}), \qquad c \longleftarrow \left\lfloor \frac{S}{2^{32}} \right\rfloor;$
   3. $n \longleftarrow n + 1;$

Mother-of-all

Propus de *George Marsaglia*, fiind extrem de rapid în implementare pe limbaj de asamblare.
Se aleg cinci numere întregi x_0, x_1, x_2, x_3, c (nu toate nule), fiecare de 32 biți.

1. $n \longleftarrow 4$;
2. **while** $n \le MAX$ **do**
   1. $S \longleftarrow 2111111111 \cdot x_{n-4} + 1492 \cdot x_{n-3} + 1776 \cdot x_{n-2} + 5115 \cdot x_{n-1} + c$;
   2. $x_n \longleftarrow S \ (mod\ 2^{32}), \qquad c \longleftarrow \left\lfloor \frac{S}{2^{32}} \right\rfloor$;
   3. $n \longleftarrow n + 1$;

Suma intermediară S este stocată pe 64 biți.

Mother-of-all

Propus de *George Marsaglia*, fiind extrem de rapid în implementare pe limbaj de asamblare.
Se aleg cinci numere întregi x_0, x_1, x_2, x_3, c (nu toate nule), fiecare de 32 biți.

1. $n \longleftarrow 4$;
2. **while** $n \leq MAX$ **do**
    1. $S \longleftarrow 2111111111 \cdot x_{n-4} + 1492 \cdot x_{n-3} + 1776 \cdot x_{n-2} + 5115 \cdot x_{n-1} + c$;
    2. $x_n \longleftarrow S \ (mod\ 2^{32}), \qquad c \longleftarrow \left\lfloor \frac{S}{2^{32}} \right\rfloor$;
    3. $n \longleftarrow n + 1$;

Suma intermediară S este stocată pe 64 biți.
MAX este stabilit în funcție de lungimea secvenței generate.

$1/P$

Fie P un număr prim impar și b un generator al lui $\mathbb{Z}_P^*$.

$1/P$

Fie P un număr prim impar și b un generator al lui $\mathbb{Z}_P^*$.
Secvența produsă de generatorul $1/P$ cu intrarea (b, P) este șirul de cifre din reprezentarea numerică a fracției $1/P$ în baza b.

$1/P$

Fie P un număr prim impar și b un generator al lui $\mathbb{Z}_P^*$.
Secvența produsă de generatorul $1/P$ cu intrarea (b, P) este șirul de cifre din reprezentarea numerică a fracției $1/P$ în baza b.

Secvența obținută este periodică de perioadă $P-1$:

$$1/P = q_1 q_2 \ldots q_{P-1} q_P \ldots$$

Exemplu

Fie $b = 10$ și $P = 7$.

Exemplu

Fie $b = 10$ și $P = 7$.

Secvența produsă de $1/P$ cu intrarea $(10, 7)$ este

$$142857142\ldots$$

deoarece

$$1/7 = 0,142857142\ldots$$

Exemplu

Fie $b = 10$ și $P = 7$.

Secvența produsă de $1/P$ cu intrarea $(10, 7)$ este

$$142857142\ldots$$

deoarece

$$1/7 = 0,142857142\ldots$$

Evident, lungimea perioadei este $P - 1 = 6$.

Securitate

Deși generatorul $1/P$ are proprietăți care îl situează printre generatorii performanți din punct de vedere al distribuției datelor, criptografic este slab.

Securitate

Deși generatorul $1/P$ are proprietăți care îl situează printre generatorii performanți din punct de vedere al distribuției datelor, criptografic este slab.
Astfel, notând cu s numărul de cifre din reprezentarea binară a lui P:

Securitate

Deși generatorul $1/P$ are proprietăți care îl situează printre generatorii performanți din punct de vedere al distribuției datelor, criptografic este slab.
Astfel, notând cu s numărul de cifre din reprezentarea binară a lui P:

- Dacă se știe P și o secvență de caractere din șir egală cu s, aceasta se poate extinde la dreapta și la stânga.

Securitate

Deși generatorul $1/P$ are proprietăți care îl situează printre generatorii performanți din punct de vedere al distribuției datelor, criptografic este slab.
Astfel, notând cu s numărul de cifre din reprezentarea binară a lui P:

- Dacă se știe P și o secvență de caractere din șir egală cu s, aceasta se poate extinde la dreapta și la stânga.
- Dacă se știu orice $2s + 1$ elemente consecutive din șir, se poate reconstitui valoarea lui P.

*ANSI X*9.17

*ANSI X*9.17 este standard *FIPS* folosit pentru generarea de chei pseudo-aleatoare și vectori de inițializare (*VI*) din modurile de operare *DES*.

Folosește sistemul de criptare 3*DES*.

Intrare:

- s: secvență aleatoare secretă de 64 biți;
- K: cheia de criptare pentru $3DES$;
- m: număr întreg (lungimea secvenței generate).

Intrare:
- s: secvență aleatoare secretă de 64 biți;
- K: cheia de criptare pentru $3DES$;
- m: număr întreg (lungimea secvenței generate).

Ieșire: m secvențe de câte 64 biți.

Intrare:
- s: secvență aleatoare secretă de 64 biți;
- K: cheia de criptare pentru $3DES$;
- m: număr întreg (lungimea secvenței generate).

Ieșire: m secvențe de câte 64 biți.

Algoritm:

1. $I = e_K(s)$;

Intrare:
- s: secvență aleatoare secretă de 64 biți;
- K: cheia de criptare pentru $3DES$;
- m: număr întreg (lungimea secvenței generate).

Ieșire: m secvențe de câte 64 biți.

Algoritm:

1. $I = e_K(s)$;
2. **for** $i \longleftarrow 1$ **to** m **do**
    1. $x_i \longleftarrow e_K(s \oplus I)$;

Intrare:
- s: secvență aleatoare secretă de 64 biți;
- K: cheia de criptare pentru $3DES$;
- m: număr întreg (lungimea secvenței generate).

Ieșire: m secvențe de câte 64 biți.

Algoritm:

1. $I = e_K(s)$;
2. **for** $i \longleftarrow 1$ **to** m **do**
    1. $x_i \longleftarrow e_K(s \oplus I)$;
    2. $s \longleftarrow e_K(x_i \oplus I)$.

Intrare:
- s: secvență aleatoare secretă de 64 biți;
- K: cheia de criptare pentru $3DES$;
- m: număr întreg (lungimea secvenței generate).

Ieșire: m secvențe de câte 64 biți.

Algoritm:

1. $I = e_K(s)$;
2. **for** $i \longleftarrow 1$ **to** m **do**
   1. $x_i \longleftarrow e_K(s \oplus I)$;
   2. $s \longleftarrow e_K(x_i \oplus I)$.
3. **output**$(x_1, x_2, \ldots, x_m)$.

Securitate

Proprietatea de aleatorism a secvențelor se măsoară prin teste statistice.

Securitate

Proprietatea de aleatorism a secvențelor se măsoară prin teste statistice.

Un generator de secvențe pseudo-aleatoare trece testele statistice dacă se comportă similar sau identic cu un generator real aleator.

Securitate

Proprietatea de aleatorism a secvențelor se măsoară prin teste statistice.

Un generator de secvențe pseudo-aleatoare trece testele statistice dacă se comportă similar sau identic cu un generator real aleator.
Din punct de vedere criptografic, securitatea generatorului depinde de eficiența computațională a algoritmului folosit, precum și de posibilitatea ca un adversar – cunoscând secvența pseudo-aleatoare rezultată – să determine parametrii secreți ai algoritmului.

Securitate

Proprietatea de aleatorism a secvențelor se măsoară prin teste statistice.

Un generator de secvențe pseudo-aleatoare trece testele statistice dacă se comportă similar sau identic cu un generator real aleator.
Din punct de vedere criptografic, securitatea generatorului depinde de eficiența computațională a algoritmului folosit, precum și de posibilitatea ca un adversar – cunoscând secvența pseudo-aleatoare rezultată – să determine parametrii secreți ai algoritmului.

Un șir poate fi bun din punct de vedere statistic, dar să prezinte numeroase defecte de securitate criptografică.

Definiție

Un generator G este "nediferențiabil (în timp) polinomial" dacă nu există nici un text statistic eficient care să poată decide că G este diferit de un generator real aleator (GA).

Definiție

Un generator G este "nediferențiabil (în timp) polinomial" dacă nu există nici un text statistic eficient care să poată decide că G este diferit de un generator real aleator (GA).

Lemă

Dacă generatorul G este nediferențiabil polinomial, atunci nu există nici un algoritm de complexitate polinomială ("eficient") care să-l poată sparge.

Exemplu

Pentru generatorul $1/P$ se poate construi un test statistic care determină dacă – pentru un număr prim arbitrar de n cifre binare – o secvență de $3n$ numere a fost extrasă din $1/P$ sau a fost generată aleator.

Exemplu

Pentru generatorul $1/P$ se poate construi un test statistic care determină dacă – pentru un număr prim arbitrar de n cifre binare – o secvență de $3n$ numere a fost extrasă din $1/P$ sau a fost generată aleator.

Astfel, pentru generarea lui P se utilizează $2n+1$ elemente (din cele $3n$).
Apoi, folosind acest P se generează $3n$ cifre, care se compară cu secvența dată.

Exemplu

Pentru generatorul $1/P$ se poate construi un test statistic care determină dacă – pentru un număr prim arbitrar de n cifre binare – o secvență de $3n$ numere a fost extrasă din $1/P$ sau a fost generată aleator.

Astfel, pentru generarea lui P se utilizează $2n+1$ elemente (din cele $3n$).
Apoi, folosind acest P se generează $3n$ cifre, care se compară cu secvența dată.
Dacă cele două secvențe se potrivesc, atunci (cu o probabilitate de cel puțin $1 - \frac{1}{2^{n-1}}$) șirul a fost produs de generatorul $1/P$.

Teste

Pachete de teste utilizate ca standarde pentru evaluarea securității generatorilor de numere pseudo-aleatoare:

Teste

Pachete de teste utilizate ca standarde pentru evaluarea securității generatorilor de numere pseudo-aleatoare:

- *DIEHARD* (15 teste elaborate de George Marsaglia)

Teste

Pachete de teste utilizate ca standarde pentru evaluarea securității generatorilor de numere pseudo-aleatoare:

- *DIEHARD* (15 teste elaborate de George Marsaglia)
- *NIST* (16 teste elaborate de Institutul de Standarde din *SUA*).

Teste

Pachete de teste utilizate ca standarde pentru evaluarea securității generatorilor de numere pseudo-aleatoare:

- *DIEHARD* (15 teste elaborate de George Marsaglia)
- *NIST* (16 teste elaborate de Institutul de Standarde din *SUA*).

Testele au o valoare direct proporțională cu lungimea n a secvenței testate.

În general ordinul de mărime al lui n este în intervalul $[10^3, 10^7]$.

Exemple de teste

- Verifică dacă numărul de 0 și de 1 din secvență sunt aproximativ egale.

Exemple de teste

- Verifică dacă numărul de 0 și de 1 din secvență sunt aproximativ egale.
- Calculează și analizează frecvența bigramelor, trigramelor etc.

Exemple de teste

- Verifică dacă numărul de 0 și de 1 din secvență sunt aproximativ egale.
- Calculează și analizează frecvența bigramelor, trigramelor etc.
- Determină frecvența de apariție a bitului 1 în cadrul blocurilor de M caractere consecutive (M fixat).

Exemple de teste

- Verifică dacă numărul de 0 și de 1 din secvență sunt aproximativ egale.
- Calculează și analizează frecvența bigramelor, trigramelor etc.
- Determină frecvența de apariție a bitului 1 în cadrul blocurilor de M caractere consecutive (M fixat).
- Analizează numărul de iterații din șir ("iterație" $=$ o secvență neîntreruptă de biți identici).

Exemple de teste

- Verifică dacă numărul de 0 și de 1 din secvență sunt aproximativ egale.
- Calculează și analizează frecvența bigramelor, trigramelor etc.
- Determină frecvența de apariție a bitului 1 în cadrul blocurilor de M caractere consecutive (M fixat).
- Analizează numărul de iterații din șir ("iterație" = o secvență neîntreruptă de biți identici).
- Calculează rangul submatricilor create din secvența ce este testată; scopul este de a verifica independența liniară a subșirurilor de lungime fixată.
  Acest test apare atât în pachetul *DIEHARD* cât și în *NIST*.

Exemple de teste (II)

- Determină numărul de apariții ale unor secvențe țintă fixate.

Exemple de teste (II)

- Determină numărul de apariții ale unor secvențe țintă fixate.
- Calculează frecvența secvențelor de o anumită lungime.

Exemple de teste (II)

- Determină numărul de apariții ale unor secvențe țintă fixate.
- Calculează frecvența secvențelor de o anumită lungime.
- Cercetează dacă secvența poate fi comprimată semnificativ fără pierderi de informație; un șir care poate fi comprimat este considerat nealeator.

Exemple de teste (II)

- Determină numărul de apariții ale unor secvențe țintă fixate.
- Calculează frecvența secvențelor de o anumită lungime.
- Cercetează dacă secvența poate fi comprimată semnificativ fără pierderi de informație; un șir care poate fi comprimat este considerat nealeator.
- Calculează abaterea maximă de la 0 a sumelor parțiale ale secvenței (în care 0 este înlocuit cu -1), pentru a determina dacă suma cumulativă a unui subșir este prea mare sau prea mică în comparație cu cea corespunzătoare a unui șir aleator (unde aceste sume sunt apropiate de zero).

Exemple de teste (II)

- Determină numărul de apariții ale unor secvențe țintă fixate.
- Calculează frecvența secvențelor de o anumită lungime.
- Cercetează dacă secvența poate fi comprimată semnificativ fără pierderi de informație; un șir care poate fi comprimat este considerat nealeator.
- Calculează abaterea maximă de la 0 a sumelor parțiale ale secvenței (în care 0 este înlocuit cu -1), pentru a determina dacă suma cumulativă a unui subșir este prea mare sau prea mică în comparație cu cea corespunzătoare a unui șir aleator (unde aceste sume sunt apropiate de zero).
  Tot aici se urmărește de câte ori sumele parțiale au valoarea 0.

Sfârșit