Capitolul 12

Generatori de numere pseudo - aleatoare

12.1 Numere aleatoare şi numere pseudo-aleatoare

Aproape toate sistemele de criptare şi protocoalele folosite în criptografie au un punct central comun: alegerea unor numere arbitrare, necunoscute apriori, imprevizibile; denumirea standard este de "*numere aleatoare*" sau "*numere generate aleator*". În general nu se poate vorbi de un singur număr aleator decât într-un context statistic. Termenul corect este acela de *şir de numere aleatoare*.

Folosirea calculatorului reduce termenul de *numere aleatoare* la un şir de biţi generaţi aleator, grupaţi după o anumită regulă. Matematic, *nu există o modalitate mai scurtă de a specifica şirul decât secvenţa însaşi*.

Definiţia 12.1. *Un generator de numere aleatoare, sau "generator real aleator" (GA) este un dispozitiv sau un algoritm care produce o secvenţă de numere independente şi nepredictibile (care nu oferă informaţii asupra valorilor ulterioare).*

În majoritatea cazurilor, un GA produce secvenţe binare, care ulterior sunt convertite în numere întregi.

Statistica oferă destul de puţine informaţii despre biţii generaţi aleator. De exemplu, se ştie că 0 trebuie să apară la fel de frecvent ca 1, că 00 trebuie să apară de două ori mai rar decât 0 (sau 1) şi la fel de des ca 11, 10, 01. Există şi teste statistice (distribuţie normală standard, distribuţie χ^2 - Kolmogorov) care estimează cât de aleatoare sunt numerele dintr-un secvenţă.

În criptografie este esenţial ca un număr aleator să nu poată fi aflat. Un număr perfect aleator este acela pe care *Oscar* nu-l poate ghici decât prin forţă brută. O parte destul de importantă din criptanaliză se bazează pe exploatarea imperfecţiunilor unor funcţii care generează numere aleatoare.

O generare de numere pur aleatoare se realizează prin colectarea şi procesarea de date obţinute dintr-o sursă de entropie exterioară calculatorului. Sursa de entropie poate fi foarte simplă, ca de exemplu variaţiile mişcării mouse-ului, sau intervalul de timp dintre apăsarea a două taste. Surse foarte bune de entropie pot fi cele radioactive sau cele care folosesc zgomote din atmosferă.

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

Definiţia 12.2. *Fie* m, k $(0 < k < m)$ *numere întregi. Un* (k, m) *generator de numere pseudo-aleatoare este o aplicaţie recursivă*

$$f : Z_2^k \longleftarrow Z_2^m$$

calculabilă în timp polinomial.

În aplicaţii, $m = h(k)$ unde h este o funcţie polinomială.
Un generator de numere pseudo-aleatoare trebuie să satisfacă anumite cerinţe:

- Să fie simplu şi rapid.
- Să producă şiruri de numere de lungime arbitrară care să nu conţină repetiţii. Deoarece un calculator nu poate genera decât numere mai mici decât un număr dat, nu se poate construi un generator cu perioadă infinită. Generatorul trebuie să aibă totuşi o perioadă de repetiţie cât mai mare.
- Să producă numere independente unul de altul (sau cu o corelare cât mai vagă).
- Să genereze numere cu o repartiţie uniformă în spaţiul valorilor.

Putem defini un generator de numere pseudo-aleatoare folosind arhitectura calculatorului. Astfel, dacă $S = (Q, f)$ este un circuit secvenţial cu $m = card(Q)$ stări şi funcţie de tranziţie $f : Q \longrightarrow Q$, atunci secvenţa

$$x_n = f(x_{n-1}), \quad x_0 \text{ oarecare},$$

este o secvenţă de numere pseudo-aleatoare.

Un astfel de generator este cu atât mai eficient cu cât satisface mai bine cerinţele anterioare.

12.2 Generatori simpli de numere pseudo-aleatoare

12.2.1 Generatori liniari congruenţiali

Un astfel de generator (construit de Lehmer în 1949) este definit de formula

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

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

Când $b = 0$, generatorul se numeşte *multiplicativ*.

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

Un generator liniar congruenţial de perioadă m se numeşte *generator de perioadă maximală*. O analiză a strategiei de selecţie a valorilor pentru a asigura o perioadă maximală se poate găsi în [32] şi [64]. O analiză teoretică detaliată a generatorilor liniari congruenţiali se găseşte în [58].

În Tabelul 1 se află listaţi câţiva 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

Tabelul 1

Avantajul generatorilor liniari congruenţiali este viteza de calcul.

O generalizare a relaţiei de recurenţă este

$$x_{n+k} = (a^k x_n + (a^k - 1)c/b) \quad (mod\ m)$$

care dă un generator de perioadă maximală când:

- $cmmdc(c, m) = 1$;
- $b = a - 1$ este multiplu de p, pentru orice număr prim p care divide m;
- $b\ (mod\ 4) = 0$ dacă $m\ (mod\ 4) = 0$.

Dezavantajul generatorilor liniari congruenţiali este acela că ei nu mai pot fi folosiţi în criptografie; ştiind prima valoare, numerele pot fi găsite uşor. Criptanaliza a fost realizată de Jim Reeds în 1977 (cu completări în 1979) şi Joan Boyar în 1982. Ea a spart şi generatorii pătratici

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

şi cubici

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

Alţi cercetători au extins metodele de atac pentru spargerea oricărui generator polinomial congruenţial.

Acum, acest tip de generator de numere pseudo-aleatoare este folosit cu predilecţie în aplicaţii necriptografice; de exemplu, în simulare el asigură o comportare statistică bună în majoritatea testelor.

12.2.2 Generatori *Ranrot*

Clasa generatorilor *Ranrot* a fost definită de danezul Agner Fog în 1997 ([25]), iniţial pentru algoritmi de tip *Monte Carlo*. Ei se bazează pe generatoare de numere Fibonacci, completate cu operaţia de deplasare ciclică pe biţi. Sunt cunoscute şi studiate trei variante de generatoare *Ranrot*:

- **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**:
$$z_n = ((y_{n-j} \gg r_3) + (y_{n-k} \gg r_1))\ (mod\ 2^{b/2}),$$
$$y_n = ((z_{n-j} \gg r_4) + (z_{n-k} \gg r_2))\ (mod\ 2^{b/2}),$$
$$x_n = y_n + z_n \cdot 2^{b/2}.$$

S-au folosit următoarele convenţii:

1. Toate numerele x sunt întregi binare pe b biţi;
2. $0 < i < j < k \leq n$ numere întregi;
3. $\alpha \gg s$ este rotaţia secvenţei α spre dreapta cu s poziţii;
4. $0 \leq r_i \leq b-1$ pentru primele două tipuri, $0 \leq b_i \leq b/2$ pentru tipul W.

Valorile sunt calculate într-un vector (buffer) de k elemente, numit *stare* S_n.

Starea iniţială este

$$S_1 = (x_1, x_2, \ldots, x_k)$$

iar trecerea de la o stare la alta se realizează printr-o deplasare spre stânga de forma

$$(x_{n-k}, x_{n-k+1}, \ldots, x_{n-1}) \longleftarrow (x_{n-k+1}, \ldots, x_{n-1}, x_n)$$

unde x_n este calculat conform formulei specifice tipului său.

Fie $p = cmmdc(j,k)$. Dacă $p > 1$, atunci sistemul se poate descompune în p sisteme independente. Deci o primă condiţie de performanţă este $cmmdc(j,k) = 1$, ceea ce asigură inter-dependenţa tuturor numerelor din stare.

Din acelaşi motiv, la tipul W trebuie ca numărul $k - j$ să fie prim.

Din modul de implementare al adunării binare rezultă o scurgere de informaţie (prin bitul de transport - *carry*) de la biţii cei mai puţin semnificativi către cei mai semnificativi, informaţie care nu se transferă în sens contrar. Pentru eliminarea acestui neajuns s-au

încercat diverse variante, cum ar fi adunarea transportului la bitul cel mai puţin semnificativ (în loc de cel mai semnificativ) – operaţie care îmbunătăţeşte lungimea perioadei, dar nu şi caracterul aleator. Varianta considerată a fost aceea de deplasare ciclică a biţilor sumei rezultate. În plus, pentru ca toate elementele din S să rămână interdependente, trebuie ca cel puţin un r să fie nenul. Prin experimente s-a găsit că cele mai bune valori pentru numărul de poziţii rotite este aproape de $\frac{b}{2}$ pentru tipul A, de $\frac{b}{3}$ şi $\frac{2b}{3}$ pentru tipul B şi aproape de $\frac{b}{4}$, $\frac{b}{2}$, $\frac{3b}{4}$ pentru tipul $B3$.

Lungimea maximă a unei perioade la un generator *Ranrot* este $(2^k - 1) \cdot 2^{b-1}$.

Nu se cunoaşte încă un algoritm de obţinere de generatori *Ranrot* de perioadă maximală; cei cunoscuţi au fost găsiţi prin testări. Cel mai mare generator analizat are 2^{32} stări.

Exemplul 12.1. *Un generator Ranrot de tipul A cu* $j = 1, k = 4, b = 7, r = 4$ *are* 24 *cicluri de perioade* $1, 5, 9, 11, 14, 21, 129, 6576, 8854, 16124, 17689, 135756, 310417, 392239, 488483, 1126126, 1355840, 1965955, 4576377, 7402465, 8393724, 57549556, 184256986$.

Importanţa restricţiilor impuse diverselor tipuri de generatori *Ranrot* este listată în Tabelul 2.

Regula	A	B	$B3$	W
$cmmdc(j,k) = cmmdc(j,i) = cmmdc(k,i) = 1$	***	***	***	***
$1 < i < j < k - 1$	**	*	*	*
$k - j$ impar	-	-	-	***
un $r \neq 0$	***	***	***	**
toţi $r \neq 0$	***	**	*	-
r distincţi	-	**	**	**
$r > 1$	***	**	*	*
$cmmdc(r,b) = 1$	*	*	*	*
$cmmdc(b,k) = 1$	*	*	*	*

Tabelul 2

S-a notat cu: − o regulă fără importanţă, $*$ - importanţă minoră, $**$ - nerespectarea ei duce la apariţia unor cicluri de perioadă mică, $***$ - regulă importantă.

Şi aceşti generatori sunt uşor de spart, deoarece starea iniţială se poate deduce uşor din k valori consecutive ale lui x. Dacă însă parametrii nu se cunosc, generatorii *Ranrot* pot fi folosiţi cu succes în criptografie, având o securitate sporită.

Exemplul 12.2. *Să considerăm o variantă de generator Ranrot:*

$$x_n = ((x_{n-1} \gg r_1) + (x_{n-2} \gg r_2) + \ldots + (x_{n-k} \gg r_k) + h) \ (mod\ 2^b)$$

unde h *este un număr întreg arbitrar din* $[0, 2^b]$. *Pentru fiecare* r *sunt* b^k *valori posibile. Astfel, pentru* $k = 17,\ b = 32$ *numărul de variante distincte este* $1,6 \cdot 10^{35}$

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

12.2.3 Generatorul *Blum - Blum - Shub*

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

Definiţia 12.3. *Fie p, q două numere prime. Dacă*

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

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

Algoritmul *Blum - Blum - Shub* (BBS pe scurt) de generare de numere pseudo-aleatoare (prezentat ca un (k, m) generator) este:

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

$$x_{i+1} = x_i^2 \quad (mod\ n)$$

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

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

Generarea biţilor nu este de fapt recursivă, deoarece z_i $(1 \leq i \leq m)$ se poate calcula direct cu formula

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

Exemplul 12.3. *Fie $p = 383$, $q = 503$; deci $n = 192649$. Alegând $x_0 = 101355^2 \ (mod\ n) = 20749$, generatorul BBS va produce şirul pseudo-aleator* 1100111000010011101.

Detaliind

i	0	1	2	3	4	5	6	7	8	9
x_i	20749	143135	177671	97048	89992	174051	80649	45663	69442	186894
z_i	–	1	1	0	0	1	1	1	0	0

i	10	11	12	13	14	15	16	17	18	19
x_i	177046	137922	123175	8630	114386	14863	133015	106065	45870	137171
z_i	0	0	1	0	0	1	1	1	0	1

Securitatea acestui generator se bazează pe dificultatea factorizării lui n. n poate fi făcut public; oricine poate genera o secvenţă pseudo-aleatoare pe baza lui. Totuşi, dacă n nu se descompune în factori, nimeni nu poate prezice ieşirea; nici măcar o afirmaţie de genul: *Următorul bit este* 1 *cu probabilitate* 51%.

Mai mult, fiind dată o parte a secvenţei, nu există nici o modalitate de a prezice bitul anterior sau cel ulterior secvenţei.

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

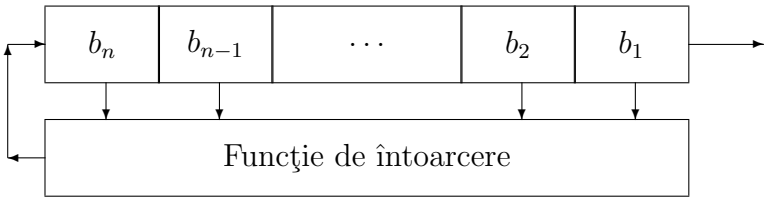
În acest moment BBS este considerat cel mai bun generator de numere pseudo-aleatoare pentru protocoale de generare şi distribuţie a cheii.

12.3 Circuite liniare

Circuitele liniare[1] sunt folosite pe scară largă în teoria codurilor detectoare şi corectoare de erori (codurile ciclice şi codurile convoluţionale) precum şi în unele sisteme de criptare liniare (AES de exemplu). Avantajul lor constă în modalitatea extrem de rapidă de calcul.

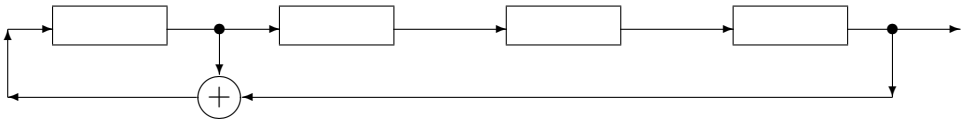
Teoria circuitelor liniare a fost stabilită în 1965 ([55]) de Ernst Selmer, şeful biroului de criptografie al guvernului norvegian. Pentru detalii şi rezultate teoretice poate fi consultat cartea de teoria codurilor [2].

Un $LFSR$ (*Linear Feedback Shift Register*) este un circuit liniar format dintr-un registru serial şi o funcţie de întoarcere (*feedback*). Dacă registrul este compus din n flip-flopuri de date ($DF - F$), vom avea un $n - LFSR$.



Funcţia de întoarcere este o adunare modulo 2 (XOR) a anumitor biţi din registru; uneori ea este numită *configuraţie Fibonacci* (vezi generatoarele *Ranrot*).

Exemplul 12.4. *Să considerăm un* $4 - LFSR$ *dat de schema:*



Funcţia de întoarcere este formată dintr-un singur XOR *între primul şi ultimul bit. Să presupunem că iniţial cei patru biţi din registru sunt* 1001. *La fiecare tact se va obţine o nouă configuraţie, anume:*

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

după care apare din nou 1001.

La ieşire va apare secvenţa 1001000111101011. *Acest circuit asigură un generator de perioadă* 16.

Un $n - LFSR$ poate avea maxim $2^n - 1$ stări distincte (starea $00\ldots0$ este exclusă deoarece ea formează un ciclu de lungime 1, neinteresant pentru generarea de numere pseudo-aleatoare).

Fie $g(X) \in Z_2[X]$, $g(X) = 1 + g_1X + \ldots + g_nX^n$ polinomul asociat unui $n - LFSR$, unde $g_i = 1$ dacă şi numai dacă bitul i participă la o adunare modulo 2. Astfel, în Exemplul 12.4 polinomul este $g(X) = 1 + X + X^4$.

[1] Termenul din engleză este *Shift Register*.

Să considerăm un polinom $g(X) \in Z_2[X]$, $grad(g(X)) = n$ şi fie m cel mai mic număr astfel ca $g(X)|X^m+1$. Atunci secvenţa binară generată de un $n-LFSR$ asociat lui $g(X)$ are perioada m (este o m - secvenţă).

Dacă $g(X)$ este un polinom ireductibil peste Z_2, atunci $m = 2^n - 1$, iar aceasta este valoarea maximă posibilă (egalează numărul de stări distincte posibile din $n - LFSR$). Toate detaliile teoretice care justifică aceste afirmaţii se găsesc în [2].

Lema 12.1. *Un polinom $g(X) \in Z_2[X]$ este ireductibil dacă şi numai dacă*

1. *Are un număr impar de termeni;*
2. *Cel puţin un termen este de forma X^{2p+1}.*

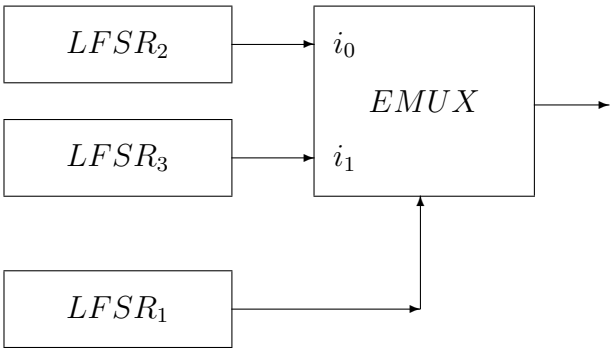
Demonstraţie: Exerciţiu.
În [51] pag. 376 este dat un tabel cu aproape 300 polinoame ireductibile.

Evident, un $n - LFSR$ este un generator de secvenţe, dar proprietatea lor pseudo-aleatoare este extrem de slabă; o stare internă oferă următorii n biţi din secvenţa de ieşire. Chiar dacă funcţia de întoarcere nu este cunoscută, ea poate fi determinată pe baza a $2n$ biţi de ieşire (Algoritmul de decodificare *Berlekamp - Massey*, [2]). Totuşi, prin combinarea mai multor circuite $LFSR$ se pot obţine generatori acceptabili de secvenţe pseudo - aleatoare.

12.4 Generatori bazaţi pe $LFSR$

12.4.1 Generatorul Geffe

Generatorul *Geffe* combină într-o formă neliniară trei $LFSR$, conform schemei următoare[2]:



$LFSR_1$ formează funcţia de selecţie a multiplexorului elementar, intrările fiind asigurate de celelalte două $LFSR$-uri. Dacă a_1, a_2, a_3 sunt ieşirile din cele trei $LFSR$-uri, ieşirea din generatorul Geffe este dată de relaţia

[2]Pentru noţiunile elementare de arhitectura calculatorului se poate folosi referinţa [3].

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

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.

Din punct de vedere criptografic generatorul nu rezistă unui atac *prin corelare*. Ieşirea din generator coincide cu ieşirea din $LFSR_2$ cam 75% din timp. Deci, dacă definiţiile polinomiale ale circuitelor sunt cunoscute se poate ghici valoarea iniţială din $LFSR_2$ şi genera secvenţa sa de ieşire. Apoi se numără de câte ori ieşirea din $LFSR_2$ coincide cu ieşirea din generator. Statistic, dacă nu s-a ghicit corect, cele două secvenţe coincid cam 50%; dacă s-a ghicit corect, ele coincid cam 75%.

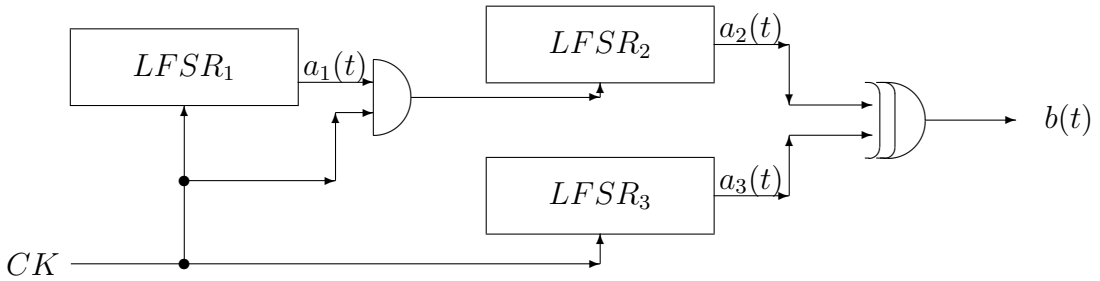
Similar, ieşirea generatorului coincide cu cea din $LFSR_3$ cam 75% din timp.

Cu aceste corelări secvenţa poate fi ghicită complet. Într-un articol din 1991, Zeng ş.a. ([61]) arată că dacă polinoamele ireductibile au câte trei termeni iar cel mai mare $LFSR$ este de lungime n, atunci o secvenţă de $37n$ biţi la ieşirea din generator este suficientă pentru determinarea stărilor interne din cele trei circuite $LFSR$.

Generatorul Geffe poate fi extins la $2^k + 1$ $LFSR$ legaţi printr-un MUX_k. Acest lucru nu va mări însă securitatea generatorului.

12.4.2 Generatori "Stop-and-Go"

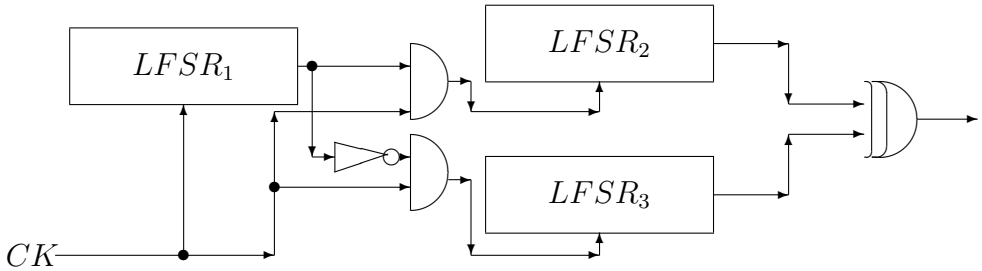
- Cel mai cunoscut este generatorul *Beth - Piper* (după numele autorilor); structura sa este următoarea:



  Acest generator controlează ceasurile celor trei circuite. Astfel, ceasul de intrare în $LFSR_2$ este controlat de ieşirea din $LFSR_1$; în acest fel, $LFSR_2$ şi schimbă starea la momentul t numai dacă ieşirea din $LFSR_1$ a fost 1 la momentul $t-1$.

  Nu se cunosc studii asupra complexităţii acestui generator. El totuşi nu a rezistat atacurilor corelate ([61]).

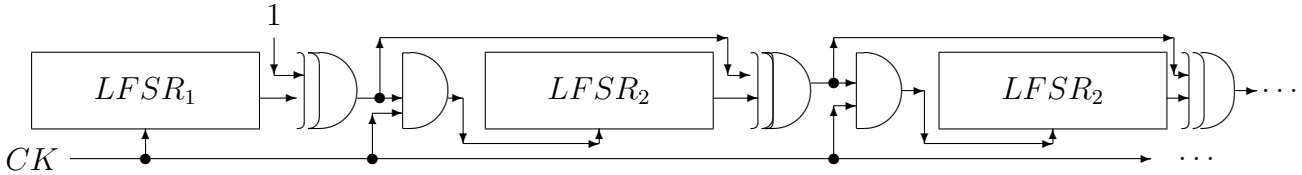
- *Stop-and-Go alternativ*: Această variantă foloseşte trei $LFSR$ de lungimi diferite, legate într-un circuit de forma:



Când ieşirea din $LFSR_1$ este 1 ea activează $LFSR_2$; în caz contrar este activat $LFSR_3$. Ieşirea din generator este un XOR dintre cele două ieşiri.

Acest generator are o perioadă mare. Există un atac prin corelare asupra sa (mai precis asupra $LFSR_1$), dar acesta nu i-a slăbit substanţial securitatea.

- *Generator Gollmann*: Este o legare serială "în cascadă" a mai multor circuite $LFSR$, ceasul fiecărui $LFSR$ fiind controlat de circuitul anterior.



Dacă la momentul $t-1$ ieşirea din $LFSR_i$ este 1, atunci la momentul t este activat $LFSR_{i+1}$. Ieşirea din generator este ieşirea din ultimul $LFSR$. Dacă toate circuitele liniare au aceiaşi lungime n, complexitatea unui generator *Gollmann* cu k $LFSR$-uri este $n \cdot (2^n - 1)^{k-1}$.

12.5 Alţi generatori de numere pseudo-aleatoare

12.5.1 Generatorul *Blum - Micali*

Fie g un număr prim, p un număr prim impar şi x_0 o valoare iniţială. Se generează numerele

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

Ieşirea din generator este 1 dacă $x_i < \dfrac{p-1}{2}$ şi 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 sigur.

12.5.2 Generatorul *RSA*

Sistemul de criptare *RSA* poate fi folosit şi pentru generare de numere aleatoare.

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

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

Ieşirea din generator este $z_i = x_i \pmod 2$.

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

12.5.3 Generatorul *Mother-of-all*

Este un generator propus de *George Marsaglia*.

Iniţial se aleg cinci numere întregi x_0, x_1, x_2, x_3, c (nu toate nule), stocate pe 32 biţi fiecare.

Algoritmul este:

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

Suma intermediară S este stocată pe 64 biţi. Valoarea MAX este stabilită în funcţie de lungimea secvenţei de numere pseudo-aleatoare generate.

Implementat în limbaj de asamblare, algoritmul este extrem de rapid, deoarece aici există o instrucţiune de înmulţire a două numere întregi pe 32 biţi, cu rezultatul pe 64 biţi. Scris într-un limbaj de nivel înalt, algoritmul foloseşte numere în virgulă mobilă cu o mantisă de 63 biţi.

12.5.4 Generatorul $1/P$

Fie P un număr prim impar şi b un generator al lui Z_P^*. Secvenţa pseudo-aleatoare produsă de generatorul $1/P$ cu intrarea (b, P) este şirul de cifre zecimale din reprezentarea numerică a fracţiei $1/P$ în baza b.

Secvenţa obţinută este periodică de perioadă $P - 1$: $1/P = q_1 q_2 \ldots q_{P-1} q_P \ldots$

Exemplul 12.5. *Fie $b = 10$ şi $P = 7$. Secvenţa pseudo-aleatoare produsă de generatorul $1/P$ cu intrarea $(10, 7)$ este $142857142\ldots$, deoarece $1/7 = 0,142857142\ldots$ Evident, lungimea perioadei este $P - 1 = 6$.*

Deşi generatorul $1/P$ are o serie de proprietăţi care îl situează printre generatorii performanţi din punct de vedere al distribuţiei datelor, criptografic este slab.

Astfel, notând cu s numărul de cifre din reprezentarea binară a lui P, se pot demonstra următoarele rezultate ([4]):

- Dacă se ştie P şi o secvenţă de caractere din şir egală cu s, aceasta se poate extinde la dreapta şi la stânga.

- Dacă se ştiu orice $2s + 1$ elemente consecutive din şir, se poate reconstitui valoarea lui P.

12.5.5 Generatorul $ANSI\ X9.17$

$ANSI\ X9.17$ este un standard $FIPS$ folosit pentru generarea de chei pseudo-aleatoare şi vectori de iniţializare (VI) din modurile de operare DES. El foloseşte sistemul de criptare $3DES$.

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

Ieşire: m secvenţe de câte 64 biţi.

Algoritm:

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

12.6 Securitatea generatorilor de numere pseudo-aleatoare

Proprietatea de aleatorism a secvenţelor se măsoară prin teste statistice. Un generator de secvenţe pseudo-aleatoare trece testele statistice dacă se comportă asemănător sau identic cu un generator real aleator.

Din punct de vedere criptografic, securitatea generatorului depinde de eficienţa computaţională a algoritmului folosit, precum şi de posibilitatea ca un adversar – cunoscând doar secvenţa pseudo-aleatoare rezultată – să determine parametrii secreţi ai algoritmului. Se poate spune că din punct de vedere criptografic este mai important să se pună în evidenţă slăbiciunile criptografice ale unei secvenţe pseudo-aleatoare decât proprietăţile sale statistice (un şir poate fi bun din punct de vedere statistic, dar să prezinte numeroase defecte de securitate criptografică).

Astfel, apare o relaţie de inter-dependenţă între existenţa unui test statistic eficient (care poate fi folosit la diferenţierea dintre un geneator şi un generator real aleator) şi existenţa unui posibil atac criptografic (de genul atacului prin corelare) care poate "sparge" generatorul.

Definiţia 12.4. *Un generator G este "nediferenţiabil (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).*

Lema 12.2. *Dacă generatorul G este nediferenţiabil polinomial, atunci nu există nici un algoritm de complexitate polinomială ("eficient") care să-l poată sparge.*

Demonstraţie. Presupunem prin absurd că există un algoritm A care poate sparge generatorul G. Atunci vom construi un test simplu pentru a diferenţia G de un generator real aleator, fapt care contrazice ipoteza.

Fie α o secvenţă suficient de lungă obţinută din G. Atunci $A(\alpha)$ va da parametrii necunoscuţi ai lui G; pe baza lor se calculează următoarea ieşire ipotetică din G. Dacă acest rezultat apriori diferă de valoarea reală produsă de G, putem trage concluzia că G este un GA; altfel, G nu va fi un generator real aleator. □

Exemplul 12.6. *Pentru generatorul $1/P$ se poate construi un test statistic simplu care determină dacă – pentru un număr prim arbitrar de n cifre binare – o secvenţă 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 secvenţa dată. Dacă cele două secvenţe se potrivesc, atunci (cu o probabilitate de cel puţin $1-\frac{1}{2^{n-1}}$) şirul a fost produs de generatorul $1/P$.*

12.6.1 Teste statistice

Există pachete de teste utilizate ca standarde pentru evaluarea securităţii generatorilor de numere pseudo-aleatoare. Cele mai cunoscute sunt $DIEHARD$ (15 teste elaborate de George Marsaglia) şi $NIST$ (16 teste elaborate de Institutul de Standarde din SUA). Testele propuse se concentrează pe o mare varietate de tipuri de non-aleatorism care pot exista într-o secvenţă. Astfel, câteca aspecte pe care testele încearcă să le pună în evidenţă sunt:

- Verifică dacă numărul de 0 şi de 1 din secvenţă sunt aproximativ egale.
- Calculează şi analizează frecvenţa bigramelor, trigramelor, etc (paternuri de lungime fixată).
- Determină frecvenţa de apariţie a bitului 1 în cadrul blocurilor de M caractere consecutive (M fixat).
- Analizează numărul de iteraţii din şir (prin "iteraţie" se înţelege o secvenţă neîntreruptă de biţi identici).
- Calculează rangul submatricilor create din secvenţa ce este testată; scopul este de a verifica independenţa liniară a subşirurilor de lungime fixată. Acest test apare atât în pachetul $DIEHARD$ cât şi în $NIST$.

- Determină numărul de apariţii ale unor secvenţe ţintă fixate, pentru a detecta generatorii care produc prea multe apariţii ale unor şabloane neperiodice.
- Calculează frecvenţa secvenţelor de o anumită lungime.
- Cercetează dacă secvenţa poate fi comprimată semnificativ fără pierderi de informaţie; un şir care poate fi comprimat este considerat nealeator.
- Calculează abaterea maximă de la 0 a sumelor parţiale ale secvenţei (în care 0 este înlocuit cu -1), pentru a determina dacă suma cumulativă a unui subşir este prea mare sau prea mică în comparaţie cu cea corespunzătoare a unui şir aleator (care are aceste sume apropiate de zero). Tot aici se urmăreşte de câte ori sumele parţiale au valoarea 0.

Testele au o valoare direct proporţională cu lungimea n a secvenţei testate. În general ordinul de mărime al lui n este în intervalul $[10^3, 10^7]$. Pentru secvenţe de lungime mai mică testele sunt neadecvate şi nu dau o estimare corectă de aleatorism.

12.7 Exerciţii

12.1. *Se dă generatorul liniar congruenţial*

$$x_{n+1} \equiv ax_n + b \pmod{15}$$

Pentru ce valori ale perechii $(a, b) \in Z_{15} \times Z_{15}$ *acest generator are perioadă maximă ?*

12.2. *Folosiţi generatorul BBS cu parametrii* $p = 7,\ q = 11,\ x_0 = 2$ *pentru a genera o secvenţă pseudo-aleatoare de lungime* 30.

12.3. *Folosiţi generatorul RSA cu parametrii* $p = 7,\ q = 11,\ x_0 = 2$ *şi* $e = 3$ *pentru a genera o secvenţă pseudo-aleatoare de lungime* 30. *Comparaţi secvenţa obţinută cu cea găsită în exerciţiul anterior.*

12.4. *Construiţi* $LFSR_i$ $(i = 1, 2, 3)$ *de polinoame generatoare* $1 + X + X^3,\ 1 + X + X^4$ *şi respectiv* $1 + X^2 + X^5$. *Pe baza lor construiţi generatori Geffe şi Stop-and-Go.*

Plecând de la secvenţele iniţiale 011, 1011 *respectiv* 00101, *generaţi secvenţe pseudo-aleatoare cu fiecare din aceşti generatori. Ce perioade au aceste secvenţe ?*

Bibliografie

[1] Anderson R. ş.a. - *Serpent: A proposal for the Advanced Encryption Standard*, http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/serpent.pdf

[2] Atanasiu A. - *Teoria codurilor corectoare de erori*, Editura Univ. Bucureşti, 2001;

[3] Atanasiu, A. - *Arhitectura calculatorului*, Editura Infodata, Cluj, 2006;

[4] L. Blum, M. Blum, M. Shub - *Comparision of two pseudo-random number generators*, Advanced in Cryptology, CRYPTO 82

[5] D. Bayer, S. Haber, W. Stornetta; Improving the efficiency and reliability of digital time-stamping. Sequences II, Methods in Communication, Security and Computer Science, Springer Verlag (1993), 329-334.

[6] E. Biham, A. Shamir, *Differential Cryptanalysis of DES - like Cryptosystems*, Journal of Cryptology, vol. 4, 1 (1991), pp. 3-72.

[7] E. Biham, A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.

[8] E. Biham, A. Shamir, *Differential Cryptanalysis of the Full 16-Round DES*, Proceedings of Crypto92, LNCS 740, Springer-Verlag.

[9] E. Biham, *On Matsuis Linear Cryptanalysis*, Advances in Cryptology - EUROCRYPT 94 (LNCS 950), Springer-Verlag, pp. 341-355, 1995.

[10] A. Biryukov, A. Shamir, D. Wagner, *Real Time Cryptanalysis of A5/1 on a PC*, Fast Software Encryption - FSE 2000, pp 118.

[11] A. Bruen, M. Forcinito, *Cryptography, Information Theory, and Error - Correction*, Wiley Interscience 2005.

[12] Bos J.N., Chaum D. - Provably unforgable signatures; Lecture Notes in Computer Science, $740(1993), 1 - 14$

[13] D. Chaum, H. van Antwerpen - Undeniable signatures; Lecture Notes in Computer Science, 435(1990), 212 − 216

[14] D. Chaum, E. van Heijst, B. Pfitzmann; Cryptographically strong undeniable signatures, unconditionally secure for the signer. Lecture Notes in Computer Science, 576 (1992), 470-484.

[15] Brigitte Collard - *Secret Language in Graeco-Roman antiquity* (teză de doctorat) *http* : *//bcs.fltr.ucl.ac.be/FE/*07*/CRYPT/Intro.html*

[16] Cook S., *http* : *//www.claymath.org/millennium/P_vs_NP/Official_Problem_Description.pdf*

[17] Coppersmith D. ş.a. - *MARS - a candidate cypher for AES*,
http://www.research.ibm.com/security/mars.pdf

[18] Daemen J., Rijmen V. - *The Rijndael Block Cipher Proposal*,
http://csrc.nist.gov/CryptoToolkit/aes/

[19] I.B. Damgard; A design principle for hash functions. Lecture Notes in Computer Science, 435 (1990), 516-427.

[20] Diffie D.W., Hellman M.E. - *New Directions in Cryptography*, IEEE Transactions on Information Theory, IT-22, 6 (1976), pp. 644-654

[21] W. Diffie, M.E. Hellman - Multiuser cryptographic techniques; AFIPS Conference Proceedings, 45(1976), 109 − 112

[22] L´ Ecuyer P. - *Random Numbers for Simulation*, Comm ACM 33, 10(1990), 742-749, 774.

[23] Enge A. - *Elliptic Curves and their applications to Cryptography*, Kluwer Academic Publ, 1999

[24] El Gamal T., *A public key cryptosystem and a signature scheme based on discrete algorithms*, IEEE Transactions on Information Theory, 31 (1985), 469-472

[25] Fog A. - *http://www.agner.org/random/theory*;

[26] Gibson J., *Discrete logarithm hash function that is collision free and one way.* IEEE Proceedings-E, 138 (1991), 407-410.

[27] S. Haber, W. Stornetta; How to timestamp a digital document. Journal of Cryptology, 3(1991), 99-111.

[28] H. M. Heyes, *A Tutorial on Linear and Differential Cryptanalysis.*

[29] van Heyst E., Petersen T.P. - How to make efficient fail-stop signatures; Lecture Notes in Computer Science, 658(1993), 366 – 377

[30] P. Junod, *On the complexity of Matsuiş attack*, in SAC 01: Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography, pp 199211, London, UK, 2001. Springer-Verlag.

[31] Kahn D. - *The Codebreakers*, MacMillan Publishing Co, New York, 1967

[32] Kelly T. - *The myth of the skytale*, Cryptologia, Iulie 1998, pp. 244 - 260.

[33] A. Konheim - *Computer Security and Cryptography*, Wiley Interscience, 2007.

[34] Knuth D. - *The art of computer Programming*, vol 2 (Seminumerical Algorithms)

[35] Lenstra, H.W. - *Factoring Integers with Eiipltic Curves*, Annals of Mathematics, vol. 126, pp. 649-673, 1987.

[36] Matsui, M, Yamagishi, A. - *A new method for known plaintext attack of FEAL cipher.* Advances in Cryptology - EUROCRYPT 1992.

[37] M. Matsui - *Linear Cryptanalysis Method for DES Cipher*, Advances in Cryptology - EUROCRYPT 93, LNCS 765, Springer-Verlag, pp. 386-397, 1994.

[38] M. Matsui - *The first experimental cryptanalysis of the Data Encryption Standard*, in Y.G. Desmedt, editor, Advances in Cryptology - Crypto 4, LNCS 839, SpringerVerlag (1994), 1- 11.

[39] M. Matsui - *New Structure of Block Ciphers with Provable Security against Differential and Linear Cryptalaysis*, Fast Software Encryption, LNCS 1039, Springer-Verlag, 1996, pp. 205-218.

[40] Merkle, R., Hellman, M. - *Hiding Information and Signatures in Trapdoor Knapsacks*, IEEE Trans. IT 24(5), Sept 1978, pp. 525530.

[41] Merkle R.C. - *A fast software one-way functions and DES*, Lecture Notes in Computer Science, 435 (1990), 428-446

[42] Mitchell C.J., Piper F., Wild, P. - Digital signatures; Contemporary Cryptology, The Science of Information Integrity, IEEE Press, (1992), 325 – 378

[43] Menezes A., Oorschot P., Vanstome S., *Handbook of Applied Cryptography*

[44] B. Preneel, R. Govaerts, J. Vandewalle; Hash functions based on block ciphers: a syntetic approach. Lecture Notes in Computer Science, 773 (1994), 368-378

[45] Rivest R. ş.a - *The RC6^{TM} Block Cipher*,
ftp://ftp.rsasecurity.com/pub/rsalabs/rc6/rc6v11.pdf

[46] R.L. Rivest; The **MD4** message digest algorithm. Lecture Notes in Computer Science, 537, (1991), 303-311

[47] Rivest R., Shamir A., Adleman A., *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, Vol. 21 (2), 1978, pages 120–126.

[48] Rosing, M - *Implementing Elliptic Curve Cryptography*, Manning, 1998

[49] D. Salmon - *Data Privacy and Security*, Springer Professional Computing, 2003

[50] Salomaa A. - *Criptografie cu chei publice*, Ed. Militară, Bucureşti 1994

[51] Schneier B. - *Applied Cryptography*, John Wiley and Sons, 1995

[52] Schneier B ş.a. - *Twofish*, http://www.counterpane.com/twofish.html

[53] Shamir, A. - *A polynomial time Algorithm for breaking the basic Merkle - Hellman cryptosystem*,
http://dsns.csie.nctu.edu.tw/research/crypto/HTML/PDF/C82/279.PDF

[54] Shoup, V. - *Lower bounds for discrete logarithm and related problems*, Advanced in Cryptology, EUROCRYPT 97, Springer - Verlag LNCS 1233, pp. 313-328, 1997.

[55] Selmer E.S. - *Linear Recurrence over Finite Field*, Univ. of Bergen, Norway, 1966;

[56] Sibley E.H. - *Random Number Generators: Good Ones are Hard to Find*, Comm ACM 31, 10(1988), 1192-1201.

[57] Smid M.E., Branstad, D.K. - *Response to comments on the NIST proposed digital signature standard*, Lecture Notes in Computer Science, $740(1993), 76 - 88$

[58] Stinton D., *Cryptography, Theory and Practice*, Chapman& Hall/CRC, 2002

[59] Wiener M.J. - *Cryptanalysis of short RSA secret exponents*, IEEE Trans on Information Theory, 36 (1990), 553-558

[60] Williams H.C. - *Some public-key criptofunctions as intractable as factorisation*, Cryptologia, 9 (1985), 224-237.

[61] Zeng K.G., Yang C.H., Wei D.Y., Rao T.R.N.- *Pseudorandom Bit Generators in Stream Cipher Cryptography*, IEEE Computer, 24 (1991), 8.17.

[62] Secure hash Standard. National Bureau of Standards, FIPS Publications 180, 1993

[63] Digital signature standard; National Bureau of Standards, FIPS Publications 186, 1994

[64] *http* : *//en.wikipedia.org/wiki/Enigma_machine*

[65] *http* : *//en.wikipedia.org/wiki/M* − 209

[66] *http://en.wikipedia.org/wiki/Caesar_ cipher# History_ and_ usage*

[67] *http://psychcentral.com/psypsych/Polybius_ square*

[68] *http://www.answers.com/topic/vigen-re-cipher*

[69] *http://en.wikipedia.org/wiki/Rosetta_ stone*

[70] *Serpent homepage*, http://www.cl.cam.ac.uk/ rja14/serpent.html

[71] *P versus NP homepage*, http://www.win.tue.nl/ gwoegi/P-versus-NP.htm

[72] http://www.win.tue.nl/ gwoegi/P-versus-NP.htm

[73] http://en.wikipedia.org/wiki/Complexity_ classes_ P_ and_ NP