Generatory liczb pseudolosowych

Tomasz Kubik

Definicja

- **Generator liczb pseudolosowych** (ang. *Pseudo-Random Number Generator*, **PRNG**)
  - program, który na podstawie niewielkiej ilości informacji (ziarna, ang. *seed*) generuje deterministycznie „potencjalnie nieskończony” ciąg bitów, który pod pewnymi względami jest podobny o ciągu uzyskanego z prawdziwie losowego źródła
  - generowane ciągi nie są całkiem losowe: jeśli ziarno jest reprezentowane przez k bitów informacji, to generator może wygenerować n-bitowy ciąg jedynie na 2^k sposobów spośród 2^n możliwych

Zastosowania

- obliczenia probabilistyczne (wystarczają „słabe” generatory)
  - całkowanie metodą Monte-Carlo
- kryptografia (wymagają „silnych” generatorów)
  - synchroniczne szyfrowanie strumieniowe:
    - tajnym kluczem jest ziarno. Znając klucz nadawca generuje ciąg bitów i składa je operacją XOR z bitami wiadomości. Odbiorca generuje ten sam ciąg bitów pseudolosowych i odzyskuje oryginalną wiadomość poprzez złożenie go operacją XOR z bitami wiadomości zaszyfrowanej
      Ciąg bitów klucza generowany jest niezależnie od szyfrowanej wiadomości i kryptogramu. Musi być zachowana synchronizacja pomiędzy nadawca i odbiorca.
    - zmiana bitu kryptogramu (przekłamanie) nie wpływa na możliwość deszyfrowania pozostałych bitów
    - dodanie lub usunięcie bitu powoduje utratę synchronizacji.
    - istnieje możliwość zmiany wybranych bitów kryptogramu, a co za tym idzie zmiany deszyfrowanej wiadomości.

Metoda Monte Carlo

- Definicja ogólna:
  - Technika Monte Carlo (MC) jest to dowolna technika używająca liczb losowych do rozwiązania problemu
- Definicja Haltona (1970):
  - Metoda Monte Carlo jest to metoda reprezentująca rozwiązanie problemu w postaci parametru pewnej hipotetycznej populacji i używająca sekwencji liczb losowych do skonstruowania próby losowej danej populacji, z której to statystyczne oszacowania tego parametru mogą być otrzymane.
- Metoda Monte Carlo jest stosowana do modelowania matematycznego procesów zbyt złożonych, aby można było przewidzieć ich wyniki za pomocą podejścia analitycznego
- Istotną rolę w metodzie MC odgrywa losowanie (wybór przypadkowy) wielkości charakteryzujących proces
- Metoda MC jest stosowana do:
  - obliczanie całek
  - łańcuchy procesów statystycznych
  - optymalizacja

Obliczanie liczby π metodą MC

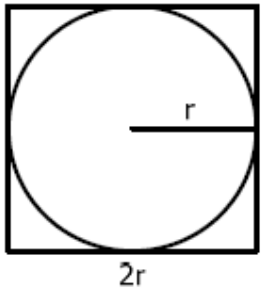
- Załóżmy, że chcemy obliczyć pole koła wpisanego w kwadrat o boku równym 2r, gdzie r - promień koła. Pola koła i kwadratu opisują wzory:

$$P_{koło} = \pi r^2$$

$$P_{kwadrat} (2r)^2 = 4 r^2$$

stąd:

$$\pi = 4 P_{koło} / P_{kwadrat}$$



Obliczanie liczby π metodą MC

- Wyznaczamy wewnątrz kwadratu bardzo dużo losowych punktów
- Zliczamy te punkty, które wpadają do wnętrza koła
- Stosunek liczby punktów zawierających się w kole do wszystkich wylosowanych punktów będzie dążył w nieskończoności do stosunku pola koła do pola kwadratu:

  $\pi = 4 P_{koło} / P_{kwadrat}$ $\pi \approx 4 n_{koło} / n_{kwadrat}$
- gdzie: $n_{koło}$ - liczba punktów w kole, $n_{kwadrat}$ - liczba wszystkich punktów

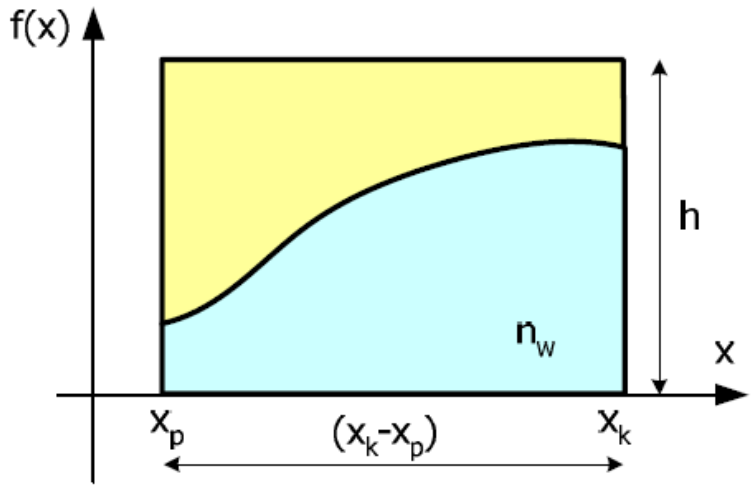
- Marquis Pierre-Simon de Laplace (1886)

Całkowanie numeryczne metodą MC

- obliczamy przybliżoną wartość całki oznaczonej

$$I = \int_{x_p}^{x_k} f(x)dx$$

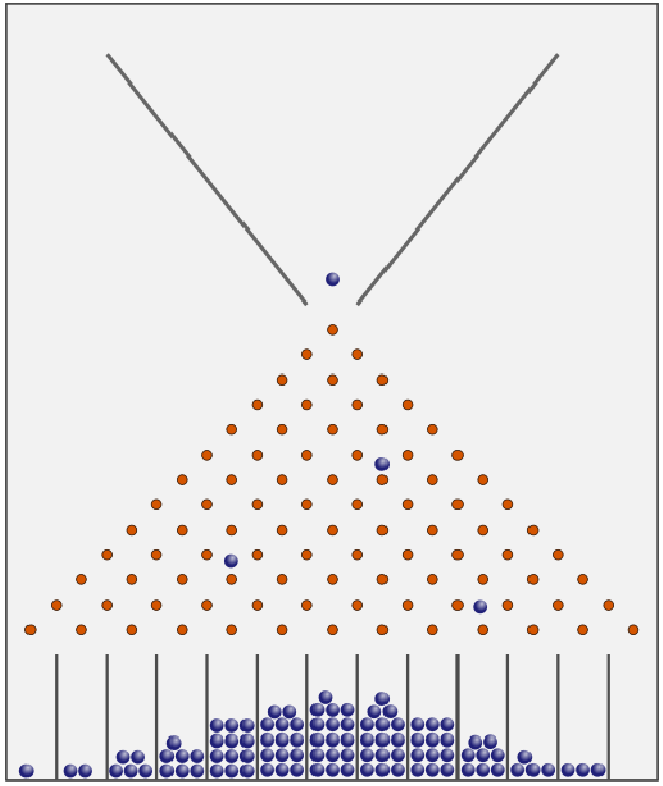
- dla funkcji f(x), której całkę chcemy obliczyć w przedziale [xp,xk] wyznaczamy prostokąt obejmujący pole pod wykresem tej funkcji o wysokości h i długości podstawy (xk-xp)
- losujemy n punktów i zliczamy te punkty nw, które wpadają w pole pod wykresem funkcji
- wartość całki obliczana jest na podstawie wzoru przybliżonego:



$$I = \int_{x_p}^{x_k} f(x)dx \approx \frac{n_w}{n} h(x_k - x_p)$$

Deska Galtona

- Pochylona deska z wbitymi gwoździami ułożonymi w trójkąt. Kulki wpadające do poszczególnych przegródek pod deską tworzą histogram rozkładu dwumianowego, prawie równego rozkładowi normalnemu (dokładnie byłyby one równe dla nieskończonej liczby nieskończenie małych kulek i nieskończenie dużej liczby przegródek).
- Deska Galtona ilustruje więc sposób powstawania w naturze rozkładu normalnego pod wpływem drobnych losowych odchyleń.
- Funkcja rozkładu prawdopodobieństwa $\binom{n}{k} p^k (1-p)^{n-k}$



- Załóżmy, że deska Galtona ma *n* przegródek. Prawdopodobieństwo, że kuleczka wpadnie do przegródki numer *k* $\left(\frac{1}{2}\right)^k \cdot \left(\frac{1}{2}\right)^{n-k}$

Pożądane cechy generatorów

- trudne do ustalenia ziarno, choć znany jest ciąg wygenerowanych bitów, trudne do ustalenia kolejno generowane bity, choć znany jest ciąg bitów dotychczas wygenerowanych

- 1. Jednorodność. W każdym punkcie generowanego ciągu bitów prawdopodobieństwo wystąpienia jedynki lub zera jest takie samo i wynosi 1/2, oczekiwana liczba zer w całym ciągu wynosi około n/2 dla ciągu *n* bitów.
- 2. Skalowalność. Każdy podciąg ciągu bitów, który uzyskał pozytywny wynik testu jakości, poddany temu samemu testowi powinien również uzyskać wynik pozytywny.
- 3. Zgodność. Zachowanie generatora musi dawać podobne rezultaty niezależnie od początkowej wartości lub fizycznego zjawiska będącego źródłem „losowości”.

Liniowy generator kongruencyjny (ang. *Linear Congruential Generator*)

- Algorytm:

$$x_{i+1} = (a \times x_i + c) \bmod m$$

$$bit_{i+1} = s_{i+1} \bmod 2$$

- gdzie:
  - a, c, i m to ustalone stałe (mnożnik, przyrost, moduł)
  - x_0 to ziarno (stan początkowy), kolejne x to liczby pseudolosowe
  - bit to kolejno generowany bit

Rodzaje generatorów LCG

- Addytywny LCG

  $x_{i+1} = (a \times x_i + c) \bmod m,$
- Multiplikatywny LCG

  $x_{i+1} = a \times x_i \bmod m$

- Generowane kolejno liczby są z zakresu od 0 do c-1, stąd po *m* cyklach obliczeniowych liczby pseudolosowe zaczynają się powtarzać
- Dla pewnych kombinacji parametrów generowany ciąg jest prawie losowy, dla innych bardzo szybko staje się okresowy

- Okres obu przedstawionych generatorów zależy od wartości parametrów równania i opisują twierdzenia:
  - Jeżeli $m = 2^k$, dla $m \geq 3$, to maksymalny okres generatora liniowego wynosi $N = 2^{k-2}$, gdy $a = 3 \bmod 8$ lub $a = 5 \bmod 8$.
  - Jeżeli m jest liczbą pierwszą, to generator liniowy posiada okres maksymalny równy m, gdy a jest pierwiastkiem pierwotnym m
  - Pierwiastek pierwotny modulo n to liczba z przedziału $< 1, n - 1 >$, której potęgi modulo n dają wszystkie liczby $z < 1, n - 1 >$.

Algorytm doboru współczynników dla generatora LCG

- Określamy zakres liczb pseudolosowych
  - $0..x_{max}$ dla LCG addytywnego
  - $1..x_{max}$ dla LCG multiplikatywnego
- Moduł m jest zawsze o 1 większy od maksymalnej liczby w zakresie, czyli:

  $m = x_{max} + 1$
- Przyrost c musi być względnie pierwszy z modułem m
  - moduł m można rozłożyć na czynniki pierwsze i jako c wybrane mogą być czynniki nie występujące w rozłożeniu
  - c może być generowane pseudolosowe, pod warunkiem, że spełnia warunek: gcd(c,m) = 1
- Mnożnik a dobierany jest tak, aby a - 1 było podzielne przez każdy czynnik pierwszy modułu m
  - jeśli moduł m dzieli się przez 4, to a - 1 również powinno być podzielne przez 4

Przykład generatora LCG (wg D. Knutha)

- $x_{i+1} = (a \times x_i + c) \bmod m$

  x_0 - ziarno

  $a = [\pi \times 10^9]$

  $c = [e \times 10^9]$

  $m = 34359738368 = 2^{35}$

  $e = 2{,}718281828459045235360287471352 7...$

  - podstawa logarytmów naturalnych

Przykładowa implementacja LCG

```
#include <iostream>
using namespace std;
int main()
{
  unsigned long long m,a,c,x,x0;
  cin >> m >> a >> c >> x0;
  x = x0;
  do
  {
    x = (a * x + c) % m;
    cout << x << " ";
  } while(x != x0);
  cout << endl;
  return 0;
}
```

Problemy generatorów LCG

- jeśli moduł jest potęgą liczby 2 ($m= 2^k$) to młodsze bity generowanych liczb pseudolosowych posiadają krótszy okres powtarzania niż cała liczba pseudolosowa
- problem występuje, gdyż wyliczanie operacji modulo zastępowane jest obcinaniem bitów:

$x_{i+1} = (a \times x_i + c) \bmod 2^k = (a \times x_{i-1} + c)$ obcięte do k bitów

Przykładowe implementacje LCG

Źródło	a	b	c	Wyjściowe bity ziarna w *rand()* lub *w Random(L)*
Numerical Recipes	1664525	1013904223	2^{32}	
Borland C/C++	22695477	1	2^{32}	bity 30..16 w *rand()*, 30..0 w *lrand()*
GNU Compiler Collection	69069	5	2^{32}	bity 30..16
ANSI C: Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++	1103515245	12345	2^{32}	bity 30..16
Borland Delphi, Virtual Pascal	134775813	1	2^{32}	bity 63..32 w *(seed * L)*
Microsoft Visual/Quick C/C++	214013	2531011	2^{32}	bity 30..16
ANSIC	1103515245	12345	2^{31}	
MINSTD	16807	0	$2^{31}-1$	
RANDU	65539	0	2^{31}	
SIMSCRIPT	630360016	0	$2^{31}-1$	
BCSLIB	5^{15}	7261067085	2^{35}	
BCPL	2147001325	715136305	2^{32}	
URN12	452807053	0	2^{31}	
APPLE	1220703125	0	2^{35}	
Super-Duper	69069	0	2^{32}	

LFSR — Linear Feedback Shift Register

- LFSR posiada rejestr przesuwający o długości n bitów, który na początku zawiera losowe bity.
- Niektóre bity rejestru są poddawane operacji XOR i wynik zastępuje najstarszy bit rejestru, jednocześnie pozostałe bity przesuwane są o jedna pozycje w prawo i najmłodszy bit staje się kolejnym bitem generowanego ciągu.

Przykład LFSR

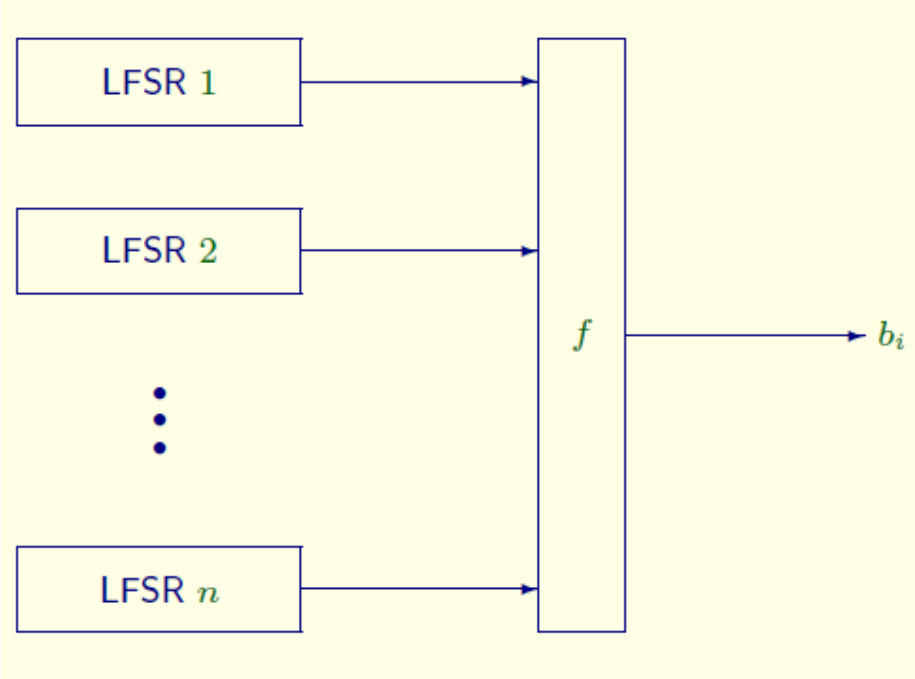
- Rejestr 4-bitowy, którego pierwszy i czwarty bit są poddawane operacji XOR
- Początkowo rejestr zawiera same jedynki
- Generowany ciąg to najmłodsze (prawe) bity kolejnych stanów rejestru
- n-bitowy rejestr może znaleźć się w jednym z $2^n - 1$ stanów, teoretycznie może więc generować ciąg o długości $2^n - 1$ bitów. Potem ciąg powtarza się.
- Rejestr zawierający same zera generuje niekończący się ciąg zer)

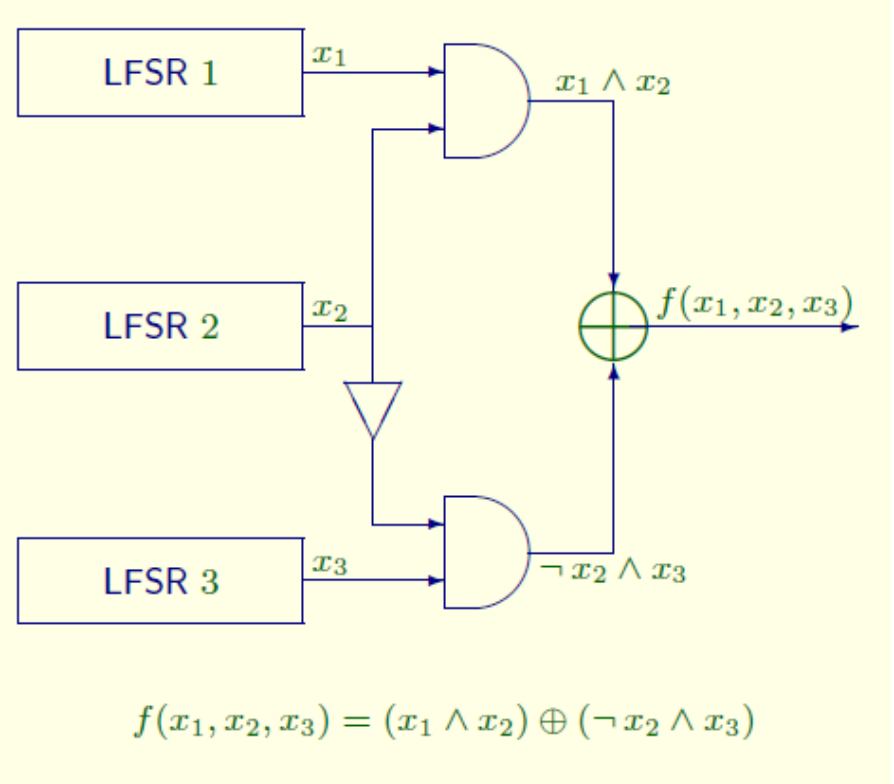
1 1 1 1
0 1 1 1
1 0 1 1
0 1 0 1
1 0 1 0
1 1 0 1
0 1 1 0
0 0 1 1
1 0 0 1
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0
1 1 0 0
1 1 1 0

Cechy LFSR

- LFSR ma słaba wartość kryptograficzna gdyż znajomość 2n kolejnych bitów ciągu pozwala na znalezienie wartości generowanych od tego miejsca.
- LFSR działa jednak bardzo szybko, zwłaszcza jeśli jest to układ sprzętowy, i stąd jest on bardzo atrakcyjny w praktycznych zastosowaniach.
- Można konstruować bardziej skomplikowane układy zawierające kilka LFSR i nieliniową funkcję f przekształcającą bity generowane przez poszczególne LFSR.

Generator Geffe





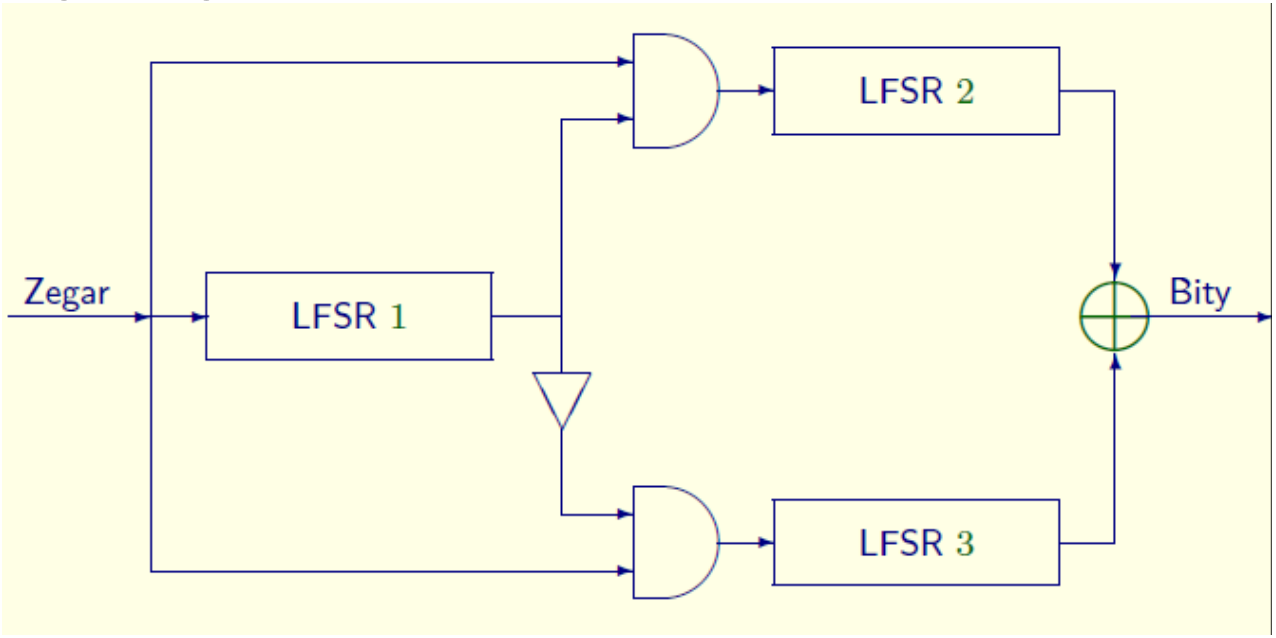
- Generator Geffe ma słabe własności kryptograficzne ze względu na korelacje pomiędzy generowanymi bitami i bitami LFSR 1 lub LFSR 2
- Niech $y(t) = f(x_1(t), x_2(t), x_3(t))$, wtedy

  $P(y(t) = x_1(t)) = P(x_2(t) = 1) + P(x_2(t) = 0) \cdot P(x_3(t) = x_1(t)) = 1/2 + 1/2 \cdot 1/2 = 3/4$,

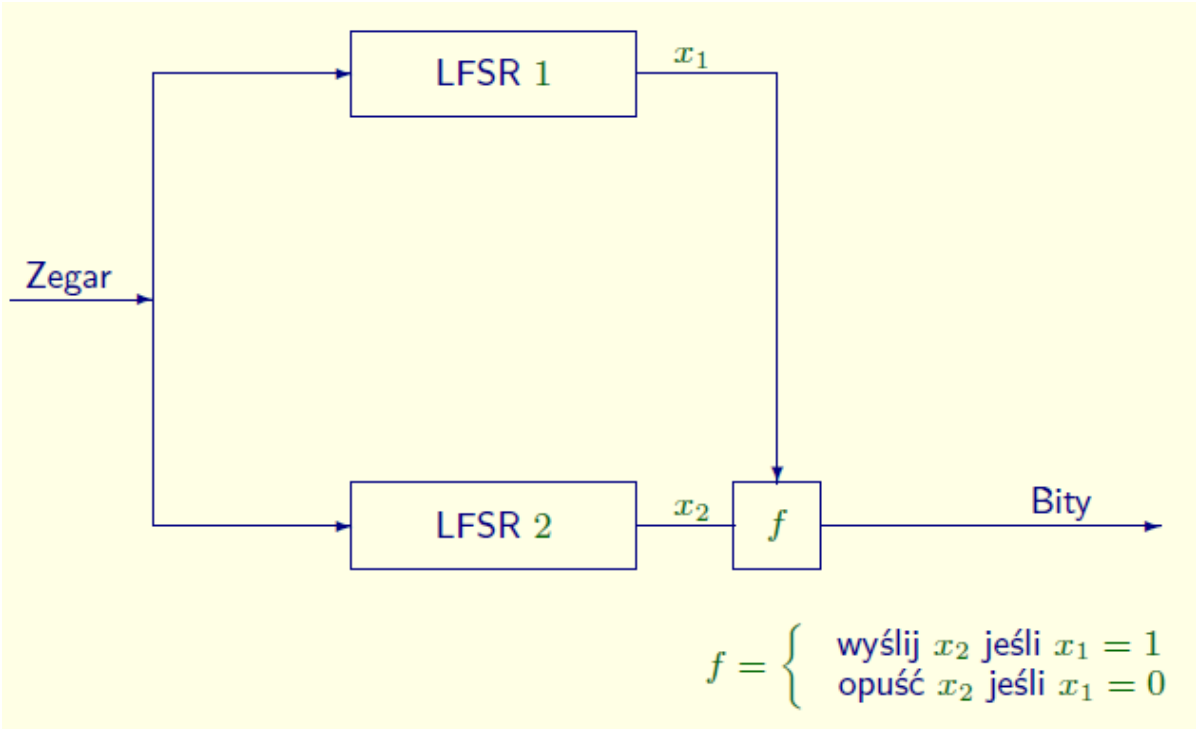
  i podobnie dla $x_3(t)$.

Generatory sterowane zegarem

- Generator o zmiennym kroku, przemienny Stop-and-Go (ang. alternating step generator, Stop-and-Go)
- LFSR 1 jest przesuwany w każdym takcie zegara.
- Jeśli na wyjściu LFSR 1 jest 1 to LFSR 2 jest przesuwany; LFSR 3 nie jest przesuwany (poprzedni bit jest powtarzany).
- Jeśli na wyjściu LFSR 1 jest 0 to LFSR 3 jest przesuwany; LFSR 2 nie jest przesuwany (poprzedni bit jest powtarzany).
- Wyjściowe bity LFSR 2 i LFSR 3 są dodawane modulo 2 () dając kolejny bit generowanego ciągu.



Generator obcinający (ang. shrinking generator)



Generatory, w których skorzystano z trudności obliczeniowych

- **Generator Blum-Micali**
- **Generator RSA**
- **Generator Blum-Blum-Shub — BBS**
- **Generator RC 4**

Generator Blum-Micali

- Wykorzystuje trudność w obliczaniu logarytmu dyskretnego
- Na początek wybierane są dwie liczby pierwsze a i p oraz ziarno x_0 (ziarno)
- Następnie obliczane jest

  $x_{i+1} = a\, x_i \pmod p$ dla i = 1, 2, 3, . . .
- Pseudolosowy ciąg bitów dostarcza wzór

  k_i =1 jeżeli $x_i < (p - 1)/2$,

  k_i = 0 w przeciwnym przypadku

Generator RSA

- Wykorzystuje trudność związaną z faktoryzacją liczb
- Na początek wybierane są dwie liczby pierwsze p i q (N = p q) oraz liczba e względnie pierwsza z (p − 1)(q − 1).
- Następnie wybierana jest losowe ziarno x_0 mniejsze od N, oraz obliczane jest

  $x_{i+1} = x_i^e \pmod N$
- generowanym bitem jest najmłodszy bit x_i

Generator Blum Blum Shub

- Algorytm

  $x_{i+1} = (x_i)^2 \bmod M$

- gdzie:

  M = p q,

  p, q to duże liczby pierwsze dających w dzieleniu przez 4 resztę 3 i mające możliwie największy wspólny dzielnik gcm(ϕ(p-1), ϕ(q-1))

  ϕ to funkcja Eulera dana wzorem:

  $\phi(n)= n (1 – 1/p_1) (1 – 1/p_2)…(1-1/p_k)$,

  gdzie p_1 do p_k są wszystkimi czynnikami pierwszymi liczby n liczonymi bez powtórzeń

  x0 obliczane jest dla x będącego losową liczbą względnie pierwszą z M

- Wynikiem generatora jest kilka ostatnich bitów (najmłodszy bit) x_k
- Generator ten jest dość powolny, za to jest bardzo bezpieczny. Przy odpowiednich założeniach, odróżnienie jego wyników od szumu jest równie trudne jak faktoryzacja M

Generator RC 4

- Opracowany przez Rona Rivesta w 1987 r. Przez kilka lat był tajny, aż do publikacji w 1994 r. programu realizującego ten algorytm w Internecie
- algorytm pracuje w trybie OFB (Output Feedback)
- Ciąg generowany przez RC 4 jest losowym ciągiem bajtów.

Algorytm RC 4

- Algorytm używa dwóch wskaźników i, j przyjmujących wartości 0, 1, 2, . . . , 255 oraz S-boksu z wartościami S_0, S_1, . . . , S_{255}, które tworzą permutacje liczb 0, 1, . . . , 255.
- Inicjalizacja:
  - Na początku i = j = 0, S_l = l dla l = 0, 1, . . . , 255, kolejna 256-bajtowa tablica wypełniana jest bajtami klucza, przy czym klucz jest używany wielokrotnie, aż do wypełnienia całej tablicy K0,K1, . . . ,K255.
- Następnie wykonujemy:

  for i = 0 to 255:

  j = (j + S_i + K_i) (mod 256)

  zamień S_i z S_j

Algorytm RC 4

- Generowanie kolejnego bajtu:

  $i = i + 1 \pmod{256}$
  $j = j + S_i \pmod{256}$
  zamien S_i z S_j
  $l = S_i + S_j \pmod{256}$
  $K = S_l$

- Otrzymany bajt K jest dodawany modulo 2 (xor) z kolejnym bajtem wiadomości dając kolejny bajt kryptogramu (przy deszyfrowaniu role tekstu jawnego i kryptogramu się zamieniają).
- Algorytm RC 4 jest używany w wielu programach komercyjnych.