Część 4

Generowanie ciągów losowych o „dobrych” właściwościach kryptograficznych

Generatory ciągów losowych w kryptografii

Wiele (o ile nie większość) algorytmów i mechanizmów kryptograficznych wymaga zastosowania losowych ciągów binarnych o odpowiednich właściwościach statystycznych.

Przykłady:

- **generowanie pary kluczy kryptografii asymetrycznej (ciąg losowy musi zazwyczaj spełniać dodatkowe warunki wynikające z logiki algorytmu);**
- **generowanie jednorazowych kluczy sesyjnych kryptografii symetrycznej (jednostronne lub wielostronne, jak np. D-H);**
- **generowanie losowych wyzwań w protokołach uwierzytelniania (tzw. nonce);**
- **generowanie strumienia klucza dla szyfratorów strumieniowych;**
- **generowanie losowych „wtrętów” (salts, seeds) w algorytmach kryptografii klucza publicznego w celu uniknięcia powtarzalności przetwarzania tych samych danych tym samym kluczem (prywatnym albo publicznym);**
- **itp., itd.**

Generatory ciągów losowych w kryptografii (cd.)

Generator losowych bitów to urządzenie lub algorytm, które generują ciąg statystycznie niezależnych i nieobciążonych bitów.

Generator może być niedeterministyczny („prawdziwie losowy") – tzw. TRNG (True Random Number Generator)/ NRNG (Non-Deterministic Random Number Generators)

albo

deterministyczny, czyli pseudolosowy – tzw. DRNG (Deterministic Random Number Generator)/ PRNG(Pseudorandom Number Generator).

Każdy bit idealnego ciągu losowego jest nieprzewidywalny i nieobciążony; jego wartość jest niezależna od wartości innych bitów tego ciągu, zaś prawdopodobieństwa tego, iż ta wartość wynosi 0 albo 1, są identyczne. Entropia idealnego ciągu losowego liczącego n bitów wynosi n bitów.

Ponadto idealny ciąg losowy jest nieokresowy. Wszystkie PRNG generują ciągi okresowe, zaś wymagania dotyczące długości tego okresu zależą od sposobu wykorzystania tego ciągu w algorytmie/mechanizmie kryptograficznym.

Generatory ciągów losowych w kryptografii (cd.)

Zasadą działania niedeterministycznych RNG (TRNG) jest pozyskiwanie ciągów binarnych z nieprzewidywalnego i nie kontrolowanego przez człowieka fizyc ko źródło sygnałów („wiadomości") o ok entropii.

Przykłady:

- **termiczne wahania v**
- **szumy termiczne di**
- **szumy „atmosferycz owy podłączony do komputera;**
- **licznik Geigera (cza podczas rozpadu promieniotwórczego**
- **turbulencje powietrz , które powodują przypadkowe wahan ra;**
- **dźwięki z mikrofonu**

TRNG są często wykorzystywane jako „źródła entropii" podczas inicjalizacji lub „odświeżania" PRNG.

Rozważa się także „niefizyczne niedeterministyczne TRNG", w których źródłem losowego ciągu o wymaganej entropii są zasoby systemowe (np. czas systemowy, zawartość RAM) i/lub interakcje z użytkownikiem (np.ruchy myszy, uderzenia w klawiaturę); patrz: BSI TR-02102-1 : 2019 „Cryptographic Mechanisms: Recommendations and Key Lengths".

Testy generatorów ciągów losowych

W 1992 roku Ueli Maurer zaproponował uniwersalny test statystyczny, którego podstawowa idea polega na sprawdzeniu stopnia bezstratnej kompresji badanego ciągu losowego. Ponadto wykorzystuje on fakt, że w przypadku statystycznie niezależnych zmiennych losowych wariancja sumy jest sumą wariancji.

Jego zaletą jest wykrywanie praktycznie wszelkich wad powodujących odstępstwa badanego ciągu od wymagań stawianych idealnemu ciągowi losowemu, zaś wadą to, że w porównaniu z innymi zalecanymi testami wymaga znacznie dłuższych próbek badanego ciągu.

Długość badanej próbki powinna wynosić $10 \bullet 2^L + 1000 \bullet 2^L$ bitów, gdzie $6 \le L \le 16$ (ogólnie: $(Q+K) \bullet L$ bitów).

$$f_{T_U}(s^N) = \frac{1}{K} \sum_{n=Q+1}^{Q+K} \log_2 A_n(s^N)$$

$$A_n(s^N) = \begin{cases} n & \text{if } \forall i < n, b_{n-i}(s^N) \neq b_n(s^N) \\ \min\{i : i \geq 1, b_n(s^N) = b_{n\ \ i}(s^N)\} & \text{otherwise.} \end{cases}$$

$$c(L,K) \cong c'(L,K) = 0.7 - \frac{0.8}{L} + \left(1.6 + \frac{12.8}{L}\right) K^{-4/L}$$

$$\mathrm{Var}[f_{T_U}(R^N)] = \sigma^2 = c(L,K)^2 \times \frac{\mathrm{Var}[\log_2 A_n(R^N)]}{K}$$

Testy generatorów ciągów losowych (cd.)

Testy jakości statystycznej wg. FIPS PUB 140-2 przed 2002 r.

(FIPS PUB 140-2 Security Requirements for Cryptographic Modules)

Wygenerować ciąg binarny o długości 20000 bitów

Test pojedynczych bitów (Monobit test)

1) Określić liczbę X wszystkich bitów o wartości 1 występujących w badanym ciągu.
2) Wynik testu jest pomyślny wtedy, gdy $9725 < X < 10275$.

Test „pokerowy" (Poker test)

1) Podzielić badany ciąg na 4-bitowe paczki obejmujące 4 kolejne bity (paczek tych jest 5000). Zliczyć częstości f (i) pojawiania się każdej z możliwych 16 sekwencji 4-bitowych ($0 \le i \le 15$).
2) Obliczyć wielkość X: $X = (16/5000)(\Sigma(f(i))^2)-5000$.
3) Wynik testu jest pomyślny wtedy, gdy $2.16 < X < 46.17$.

Test „długich podciągów identycznych ciągów" (Long runs test)

1) Jeżeli w badanym ciągu istnieje co najmniej jeden podciąg o długości > 26 bitów zawierający same bity o wartości 0 lub same bity o wartości 1, to wynik testu jest negatywny.

Testy generatorów ciągów losowych (cd.)

Testy jakości statystycznej wg. FIPS PUB 140-2 przed 2002 r.(c.d.)

Test „podciągów identycznych ciągów” (Runs test)

1) Zliczyć wszystkie podciągi składające się tylko z bitów o wartości 0, albo tylko z bitów o wartości 1.
   Podzielić je na sześć grup:
   pierwszą - zawierającą podciągi o długości 1 bita;
   drugą - zawierającą podciągi o długości 2 bitów, itd.;
   szóstą - podciągi o długości większej niż 5 bitów.
2) Jeżeli liczebność którejkolwiek z sześciu grup podciągów nie mieści się w zakresie podanym w poniższej tablicy, to wynik testu jest negatywny.

1	*2315-2685*
2	*1114-1386*
3	*527-723*
4	*240-384*
5	*103-209*
6+	*103-209*

Testy generatorów ciągów losowych (cd.)

FIPS 140-2 Change of Notice 3 Dec 2002

Ciągły test generatora liczb losowych (RNG).

Jeżeli elementami modułu kryptograficznego są zatwierdzone*) lub niezatwierdzone RNG, to moduł musi wykonywać następujące testy poprawności ich działania.

1. **Jeżeli każde odwołanie do RNG wytwarza bloki o długości n > 15 bitów, to pierwszy blok wygenerowany po podaniu zasilania, inicjalizacji lub zerowaniu (*reset*) nie może być wykorzystany, lecz musi być zachowany do porównania z następnym wygenerowanym blokiem. Każdy następny wygenerowany blok musi być porównany z blokiem wygenerowanym poprzednio. Wynik testu jest negatywny wtedy, gdy jakiekolwiek dwa porównywane bloki są identyczne.**
2. **Jeżeli każde odwołanie do RNG wytwarza ciągi o długości n < 16 bitów, to pierwsze n bitów wygenerowanych po podaniu zasilania, inicjalizacji lub zerowaniu (*reset*) nie może być wykorzystanych, lecz muszą być zachowane do porównania z następnym wygenerowanym ciągiem n-bitowym. Każdy następny wygenerowany ciąg n-bitowy musi być porównany z n-bitowym ciągiem wygenerowanym poprzednio. Wynik testu jest negatywny wtedy, gdy jakiekolwiek dwa porównywane ciągi n-bitowe są identyczne.**

W „drafcie” FIPS 140-3 z czerwca 2019 jawnie odsyła się do publikacji NIST serii 800-90, gdzie w części B (oprócz wymagań dotyczących samej wartości entropii) zaleca się min. testy określone w NIST SP 800-22.

Testy generatorów ciągów losowych (cd.)

O wiele bardziej wiarygodne rezultaty badania generatorów liczb losowych można uzyskać wykorzystując:

A STATISTICAL TEST SUITE FOR RANDOM AND PSEUDORANDOM NUMBER GENERATORS FOR CRYPTOGRAPHIC APPLICATIONS

NIST Special Publication 800-22 – rev. 1a – April 2010

oraz

Dieharder: A Random Number Test Suite

(http://webhome.phy.duke.edu/~rgb/General/dieharder.php)

„Dieharder" wyewoluował ze „słynnego" pakietu George'a Marsaglii: „DIEHARD Statistical Tests" i oprócz nowych testów objął także testy wskazane przez NIST.

Testy generatorów ciągów losowych (cd.)

NIST
National Institute of Standards and Technology
Technology Administration
U.S. Department of Commerce

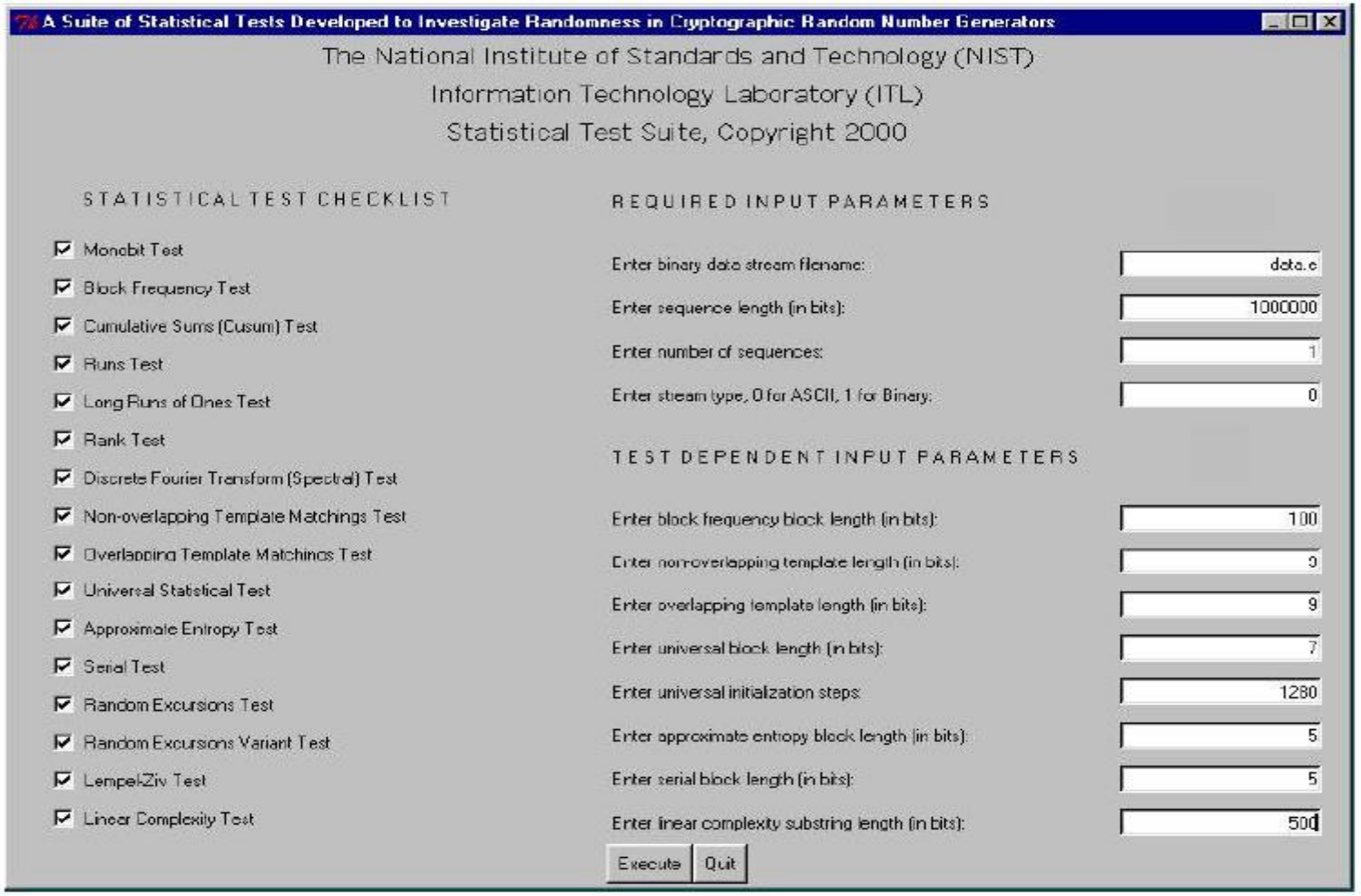
Special Publication 800-22
Revision 1a

A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications

1. The Frequency (Monobit) Test,
2. Frequency Test within a Block,
3. The Runs Test,
4. Tests for the Longest-Run-of-Ones in a Block,
5. The Binary Matrix Rank Test,
6. The Discrete Fourier Transform (Spectral) Test,
7. The Non-overlapping Template Matching Test,
8. The Overlapping Template Matching Test,
9. Maurer's "Universal Statistical" Test,
10. The Linear Complexity Test,
11. The Serial Test,
12. The Approximate Entropy Test,
13. The Cumulative Sums (Cusums) Test,
14. The Random Excursions Test, and
15. The Random Excursions Variant Test.

Testy generatorów ciągów losowych (cd.)

GUI dla NIST Statistical Test Suite (800-22)



Testy generatorów ciągów losowych (cd.)

Diehard battery of tests for randomness

Birthday spacings: Wybierz losowo punkty na dużym przedziale. Odstępy między tymi punktami powinny mieć rozkład Poissona (asymptotycznie).

Overlapping permutations: Przeanalizuj sekwencje pięciu kolejnych liczb losowych. 120 możliwych uporządkowań powinno mieć „statystycznie" równe prawdopodobieństwo.

Ranks of matrices: …

Monkey tests: …

Count the 1s: …

Parking lot test: …

Minimum distance test: …

Random spheres test: …

The squeeze test: …

Overlapping sums test: …

Runs test: …

The craps test: …

No. 1012. SERIES B. Chimpanzee at Typewriter. New York Zoological Park. Copyright 1907 by the New York Zoological Society.

J.S.Coron, D.Naccache – „Although one can easily conclude that a complex text ... has a negligible monkey probability, a simple word such as **cat** is expected to appear more frequently (each ≅ 17576 keystrokes) and could be used as a basic (yet very insufficient) randomness test."

Generatory liczb (bitów) losowych

D.E.Knuth – „Random numbers should not be generated with a method chosen at random”

American Scientist – 7/8 2016

PRNG oparte na funkcjach jednokierunkowych i blokowych algorytmach symetrycznych

Najbardziej obiecującym narzędziem do generowania takich pseudolosowych ciągów binarnych, które spełniałyby wymagania bezpieczeństwa dla algorytmów i mechanizmów kryptograficznych, wydają się być *funkcje jednokierunkowe*. Ciągi generowane z ich wykorzystaniem nazywane są ciągami *strukturalnie losowymi*.

Zasada generowania dowolnie długiego ciągu losowego za pomocą funkcji jednokierunkowej wyrażona jest zależnością:

$$k_n = f(k_{n-1})$$

$$k_0 = f(IV) \qquad (IV - Initial\ Value)$$

Jako funkcje jednokierunkowe można wykorzystać zarówno blokowe algorytmy kryptografii symetrycznej (np. AES), jak i funkcje skrótu (także funkcje skrótu z kluczem, np. HMAC)

PRNG oparte na funkcjach jednokierunkowych i blokowych algorytmach symetrycznych (cd.)

PRNG synchroniczny z algorytmem E_K jako funkcja stanu:

- **funkcja stanu generatora:** $x_i = E_K(x_{i-1})$
- **funkcja wyjścia generatora:** $k_i = g(x_i)$

PRNG synchroniczny z algorytmem E_K jako funkcja wyjścia:

- **funkcja stanu generatora:** $x_i = f(x_{i-1})$
- **funkcja wyjścia generatora:** $k_i = g(E_K(x_i))$

PRNG samosynchronizujący:

- **funkcja stanu generatora:** $x_i = f(x_{i-1}, k_{i-1})$
- **funkcja wyjścia generatora:** $k_i = g(E_K(x_i))$

PRNG oparte na funkcjach jednokierunkowych i blokowych algorytmach symetrycznych (cd.)

June 2015

NIST Special Publication 800-90A
Revision 1

Recommendation for Random Number Generation Using Deterministic Random Bit Generators

June 2006

NIST Special Publication 800-90

Recommendation for Random Number Generation Using Deterministic Random Bit Generators

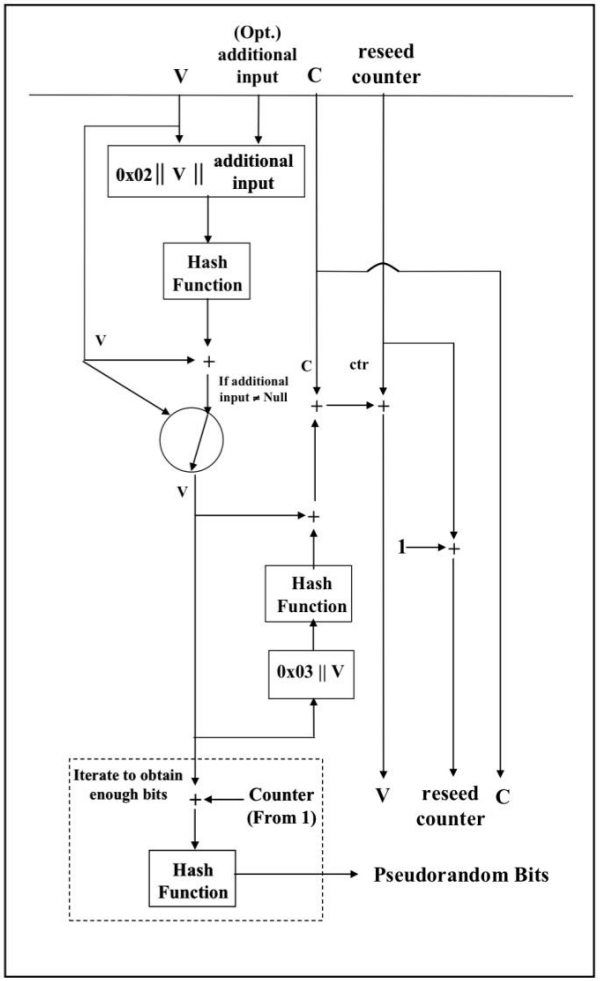
NIST SP 800-90A

The *internal_state* for **Hash_DRBG** consists of:

1. The *working_state*:
   a. A value (*V*) of *seedlen* bits that is updated during each call to the DRBG.
   b. A constant (*C*) of *seedlen* bits that depends on the *seed*.
   c. A counter (*reseed_counter*) that indicates the number of requests for pseudorandom bits since new *entropy_input* was obtained during instantiation or reseeding.

2. Administrative information:
   a. The *security_strength* of the DRBG instantiation.
   b. A *prediction_resistance_flag* that indicates whether or not a prediction resistance capability is available for the DRBG instantiation.

The values of *V* and *C* are the critical values of the internal state upon which the security of this DRBG mechanism depends (i.e., *V* and *C* are the "secret values" of the internal state).



NIST SP 800-90A

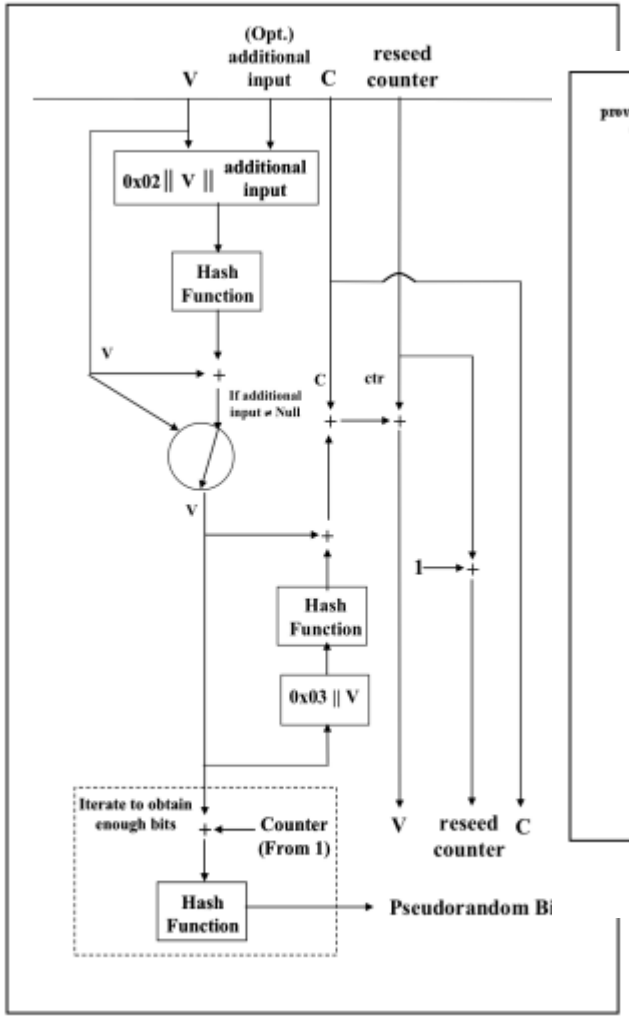


Figure 8: Hash_DRBG

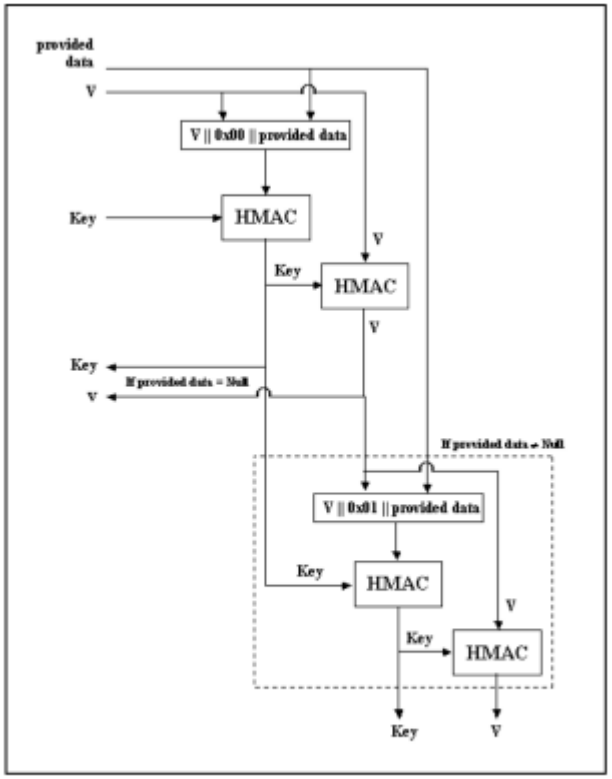


Figure 10: HMAC_DRBG_Update Function

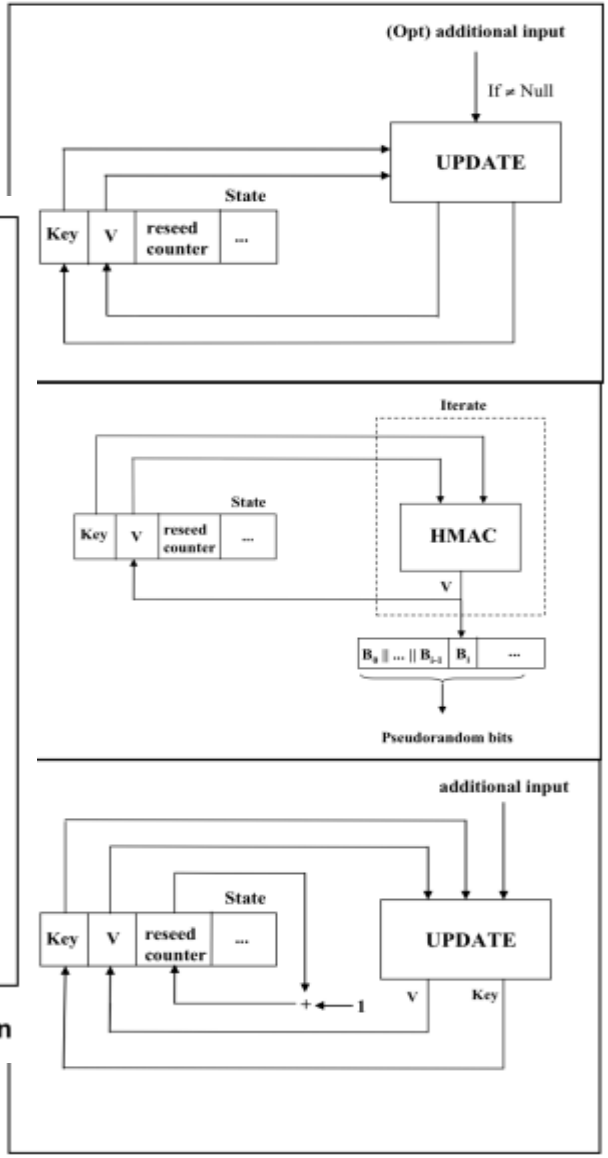


Figure 9: HMAC_DRBG Generate Function

NIST SP 800-90A

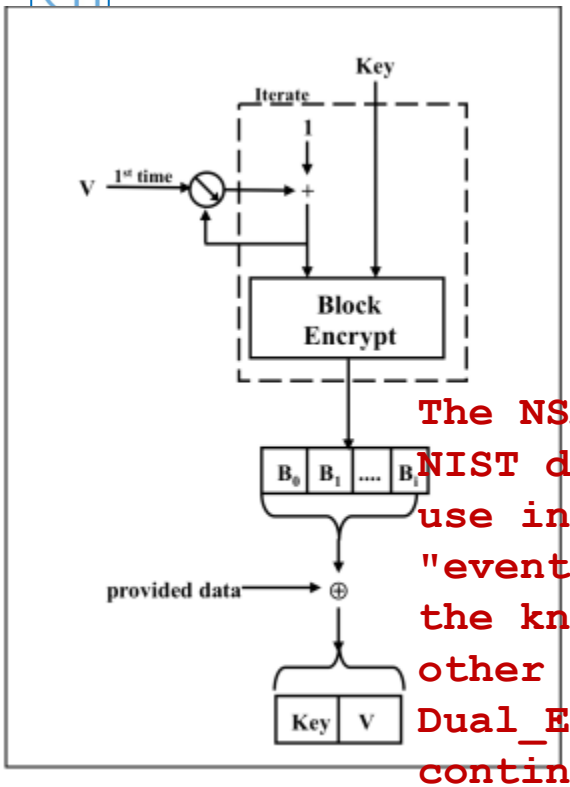


Figure 11: CTR_DRBG Update Function

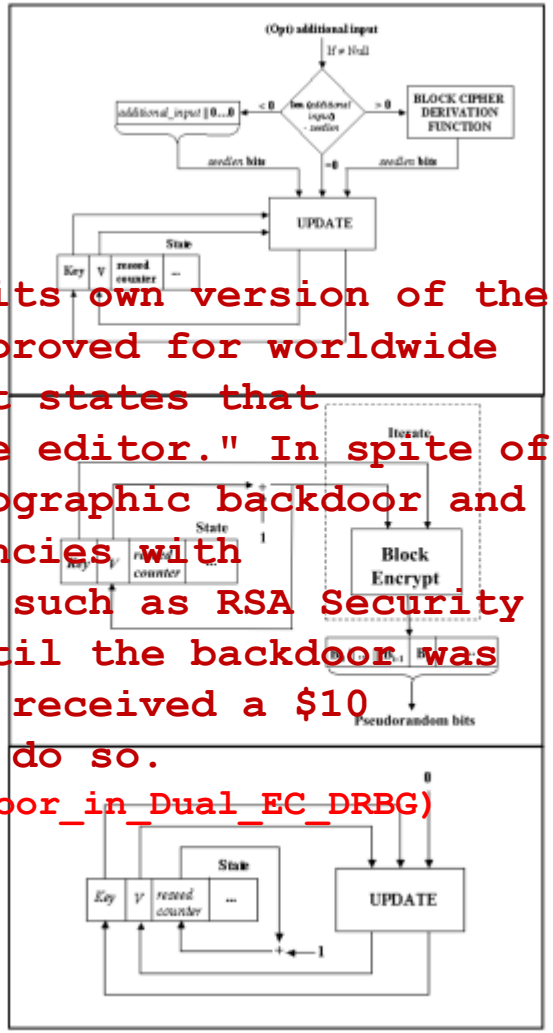


Figure 12: CTR-DRBG

The NSA worked covertly to get its own version of the NIST draft security standard approved for worldwide use in 2006. The leaked document states that "eventually, NSA became the sole editor." In spite of the known potential for a kleptographic backdoor and other known significant deficiencies with Dual_EC_DRBG, several companies such as RSA Security continued using Dual_EC_DRBG until the backdoor was confirmed in 2013. RSA Security received a $10 million payment from the NSA to do so.
(https://en.wikipedia.org/wiki/NIST_SP_800-90A#Backdoor_in_Dual_EC_DRBG)

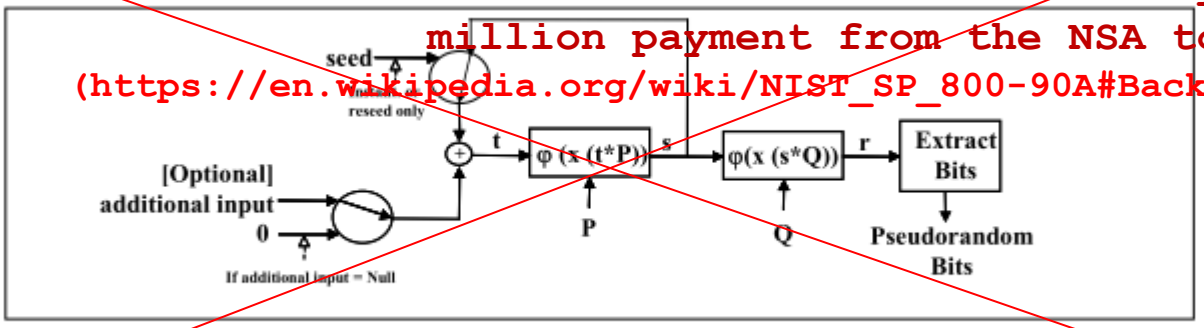


Figure 13: Dual_EC_DRBG

PRNG oparte na funkcjach jednokierunkowych i blokowych algorytmach symetrycznych (cd.)

Generator pseudolosowego ciągu binarnego ANSI X9.17

1. określić *64*-bitowe losowe (i tajne) ziarno *s*, liczbę naturalną *m* oraz *168*-bitowy klucz K algorytmu DES (E-D-E);
2. obliczyć wartość pośrednią $I = E_K(D)$, gdzie *D* jest *64*-bitową reprezentacją daty i godziny z najlepszą możliwą dokładnością;
3. obliczyć *m* pseudolosowych *64*-bitowych ciągów binarnych x_i zgodnie z poniższym pseudokodem:

```
for i = 1 until m do
        {
        x_i : = E_K(I ⊕ s);
        s : = E_K(x_i ⊕ I);
        }
```

4. zwrócić $(x_1 , x_2 , \ldots , x_m)$

Kryptograficznie bezpieczne generatory binarnych ciągów pseudolosowych

Przykład 1 - Algorytm BBS generowania pseudolosowych ciągów binarnych (Lenore Blum, Manuel Blum, Michael Shub - 1986 r.)

Wykorzystuje problemy faktoryzacji liczby złożonej i wyznaczania pierwiastka kwadratowego w algebrach modularnych.

1. *wybrać dużą liczbę Bluma n oraz parametr t określający długość klucza (w bitach);*

2. *wybrać losowo liczbę całkowitą (ziarno) s z przedziału [1,n-1] taką, że gcd(s,n) = 1, i obliczyć resztę kwadratową:*

$$x_0 = s^2 \bmod n;$$

3. *obliczyć t bitów x_i zgodnie z poniższym pseudokodem:*

```
for i = 1 until t do
{x_i : = x_{i-1}^2 mod n;
z_i : = najmniej znaczący bit x_i;}
```

4. *zwrócić $(z_1 , z_2 , \ldots , z_t)$*

Kryptograficznie bezpieczne generatory binarnych ciągów pseudolosowych (cd.)

Przykład 2 - Algorytm generowania pseudolosowych ciągów binarnych oparty na problemie RSA (Silvio Micali, Claus-Peter Schnorr – 1982 r.)

Wykorzystuje problem faktoryzacji liczby złożonej.

1.wybrać dwie duże liczby pierwsze p i q ; następnie obliczyć n = pq oraz $\Phi(n) = (p-1)(q-1)$;
niech $N = \lfloor \log_2 n \rfloor + 1$ (długość n w bitach); wybrać liczbę całkowitą e spełniającą warunek:

$$1 < e < \Phi(n),$$

oraz taką, że $\gcd(e, \Phi(n)) = 1$ i $80e \leq N$;

niech ponadto: $k = \lfloor N(1/2 - 1/e) \rfloor$

i $r = N - k$;

2.wybrać losowo binarną sekwencję bitów x_0 (ziarno) o długości r;

Kryptograficznie bezpieczne generatory binarnych ciągów pseudolosowych (cd.)

Przykład 2 (cd.)

3.obliczyć l sekwencji bitów z_i (każda o długości k bitów) zgodnie z poniższym pseudokodem:

```
for i = 1 until l do
        {
        y_i : = x_{i-1}^e mod n;
        x_i : = r najbardziej znaczących bitów y_i;
        z_i : = k najmniej znaczących bitów y_i;
        }
```

4. zwrócić $(z_1 , z_2 , \ldots , z_l)$, czyli konkatenację sekwencji $z_1 \| z_2 \| \ldots \| z_l$

Np. jeśli $e = 3$, zaś $N = 1024$, to wówczas podczas jednej „pętli" algorytmu generowanych jest $k = 341$ bitów, a ponadto jedna „pętla" wymaga tylko jednego mnożenia i jednego podnoszenia do kwadratu modulo n.

Generatory binarnych ciągów pseudolosowych efektywne ze względu na implementację programową

*Przykład – RC4 – szyfr strumieniowy (R.Rivest 1987 r.)**

Strumień klucza generowany jest jako ciąg bajtów (w każdej fazie generowany jest jeden pseudolosowy bajt).

Rolę kluczy wewnętrznych algorytmu spełniają uaktualniane w każdej fazie parametry i oraz j , interpretowane jako liczby całkowite ze zbioru $\{0,1,2,...,255\}$, a także permutacja π określona na tym samym podzbiorze zbioru liczb całkowitych.

Inicjacja algorytmu:

Po wygenerowaniu losowego ciągu m bajtów $(s_0,s_1,...,s_{m-1})$, gdzie $m \leq 256$ jest arbitralnie wybraną nieujemną liczbą całkowitą, utworzyć klucze wewnętrzne i, j i π zgodnie z podanym dalej pseudokodem.

Operacje wykonywane w fazie inicjalizacji mają za zadanie utworzenie przed pierwszą fazą pseudolosowej permutacji π.

*In 2001, a new and surprising discovery was made by Fluhrer, Mantin and Shamir: over all possible RC4 keys, the statistics for the first few bytes of output keystream are strongly non-random, leaking information about the key. (https://en.wikipedia.org/wiki/RC4).
Do 2015 formalnie dozwolony w WEP, WPA, SSL i TLS.

Generatory binarnych ciągów pseudolosowych efektywne ze względu na implementację programową (cd.)

Przykład (cd.)

Faza inicjalizacji:

$i := 0;$

$j := 0;$

$\text{for } n = 0 \text{ until } 255 \text{ do } (\pi(n) := n);$

$\text{for } n = 0 \text{ until } 255 \text{ do}$

$\{$

$j := j + \pi(i) + s_i \pmod{256};$

$tmp := \pi(i);$

$\pi(i) := \pi(j);$

$\pi(j) := tmp;$

$i := i + 1 \pmod{m};$

$\}$

Generatory binarnych ciągów pseudolosowych efektywne ze względu na implementację programową (cd.)

Przykład (cd.)

Pojedyncza faza algorytmu:

Kolejne 8-bitowe podciągi ciągu losowego (bajty k) generowane są zgodnie z poniższym pseudokodem:

$$i := i + 1 \pmod{256};$$

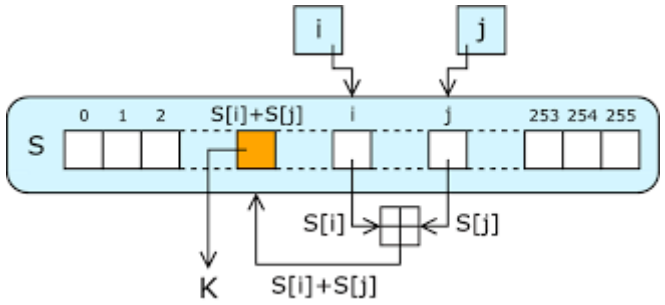
$$j := j + \pi(i) \pmod{256};$$

$$tmp := \pi(i);$$

$$\pi(i) := \pi(j);$$

$$\pi(j) := tmp;$$

$$k := \pi(\pi(i) + \pi(j));$$



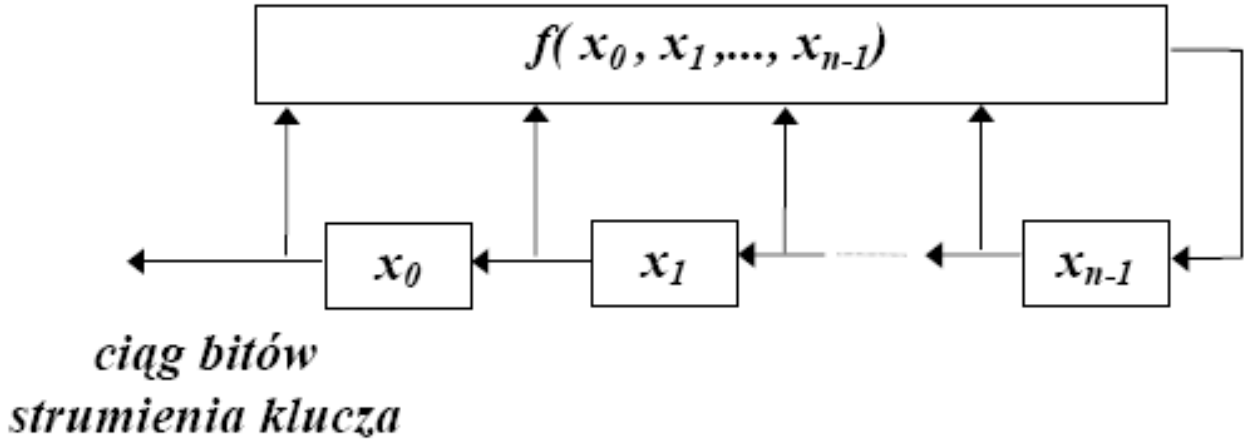
„Miejsce", z którego pobiera się w każdej fazie bajt k, zmienia się dynamicznie.

Generatory binarnych ciągów pseudolosowych wykorzystujące rejestry przesuwające (*FSR - Feedback Shift Registers*)

Rejestry przesuwające ze sprzężeniem zwrotnym (w szczególności liniowe rejestry przesuwające *LFSR - Linear Feedback Shift Registers*) są podstawowym elementem wielu współczesnych generatorów pseudolosowego binarnego strumienia klucza dla szyfratorów strumieniowych. Decydują o tym następujące zalety:

- ***LFSR* są łatwe do implementacji sprzętowej;**
- **mogą generować ciągi binarne o dużych okresach;**
- **mogą generować ciągi binarne o dobrych z punktu widzenia kryptografii cechach statystycznych;**
- **dzięki swojej strukturze mogą być skutecznie i łatwo analizowane za pomocą metod algebraicznych.**

Generatory binarnych ciągów pseudolosowych wykorzystujące rejestry przesuwające (*FSR - Feedback Shift Registers*) (cd.)



n*-stopniowy rejestr przesuwający z funkcją sprzężenia zwrotnego *f

W każdym cyklu zegarowym następuje przesunięcie bitów w stronę „wyjścia” oraz „wstawienie” do stopnia *(n-1)*-go wartości funkcji *f* .

Blok funkcyjny *f* realizuje pewna funkcie boolowską:

$$x_n = f(x_0, x_1, ...x_{n-1})$$

Generatory binarnych ciągów pseudolosowych wykorzystujące rejestry przesuwające (*FSR - Feedback Shift Registers*) (cd.)

Liniowe rejestry przesuwające (LSFR)

$$f(x_0, x_1, \ldots, x_{n-1}) = a_n x_0 \oplus a_{n-1} x_1 \oplus \ldots \oplus a_1 x_{n-1}$$

$$a_0, a_1, \ldots, a_{n-1} \in \{0, 1\}$$

Okres generowanego ciągu binarnego dla LSFR (zależny od warunków początkowych)*:*

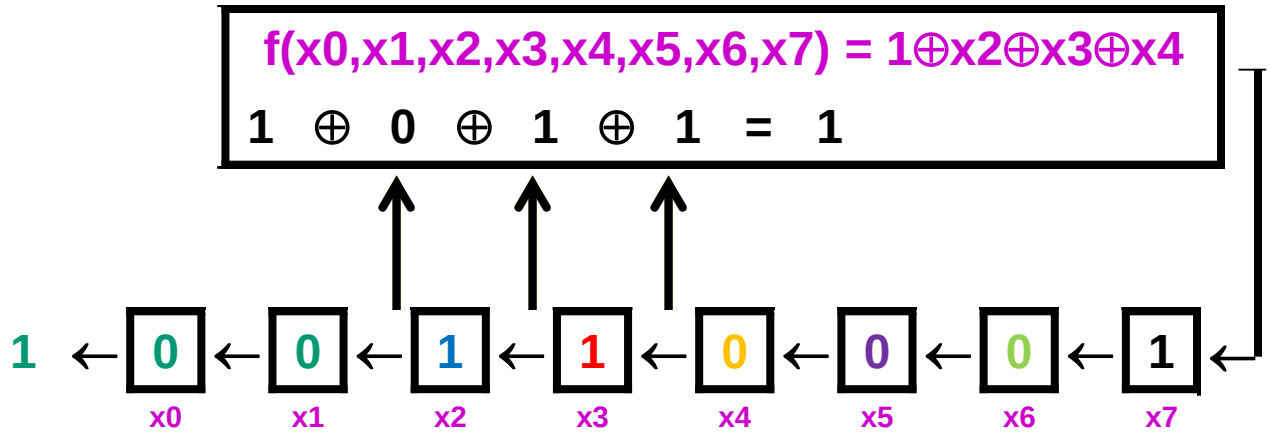
$$T \leq 2^n - 1,$$

zaś w przypadku nieliniowej funkcji *f* (NSFR) maksymalny okres:

$$T = 2^n.$$

Generatory binarnych ciągów pseudolosowych wykorzystujące rejestry przesuwające (*FSR - Feedback Shift Registers*) (cd.)

Ilustracja działania FSR



Strumień klucza: **101101** *itd..*

Generatory binarnych ciągów pseudolosowych wykorzystujące rejestry przesuwające (*FSR - Feedback Shift Registers*) (cd.)

Istnieją zasadniczo cztery podstawowe metody wykorzystywania LFSR do konstrukcji dobrych generatorów pseudolosowych ciągów binarnych:

- **wprowadzenie nieliniowości w funkcji logicznej w pętli sprzężenia zwrotnego pojedynczego LFSR (czyli utworzenie z niego NFSR);**
- **zastosowanie nieliniowej funkcji logicznej wyjść kilku LFSR;**
- **wykorzystanie wyjść jednego (lub więcej) LFSR do sterowania wejściem taktującym (wymuszającym „przesunięcie” zawartości stopni) innego (lub innych) LFSR.**
- **wykorzystanie wyjść jednego (lub więcej) LFSR do „kluczowania” wyjść innego (lub innych) LFSR (*shrinking generators* – generatory „obcinające”).**

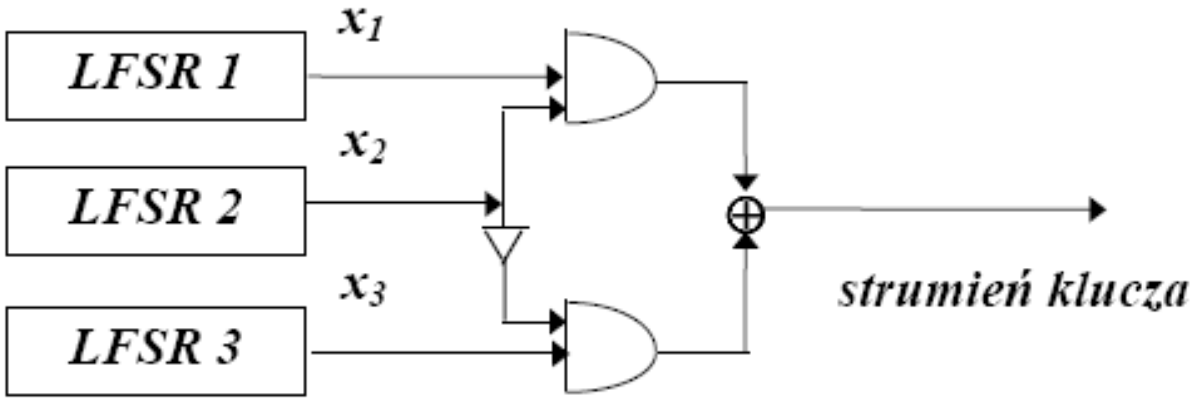
Generatory binarnych ciągów pseudolosowych wykorzystujące rejestry przesuwające (*FSR - Feedback Shift Registers*) (cd.)

Nieliniowe rejestry przesuwające (przykład):

Wykorzystanie LSFR (ograniczenie liczby możliwych wariantów)

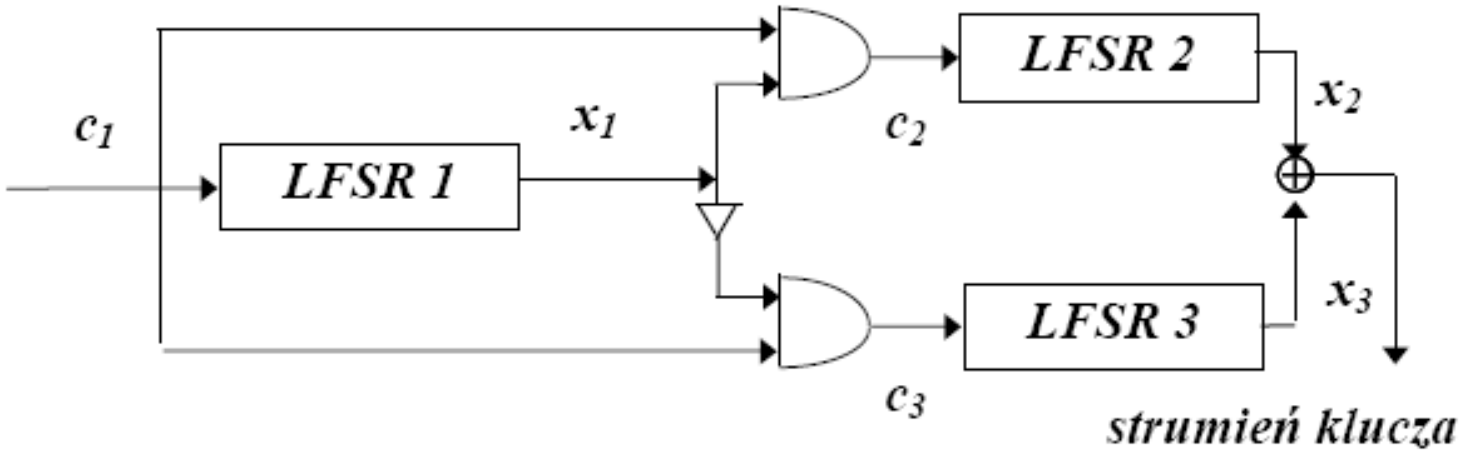
$$f^*(x_0, x_1, \ldots x_{n-1}) = f(x_0, x_1, \ldots x_{n-1}) \oplus \overline{x_0 x_1 \ldots x_{n-1}}$$

Nieliniowe generatory kombinacyjne (przykład – generator Geffe'go):



Generatory binarnych ciągów pseudolosowych wykorzystujące rejestry przesuwające (*FSR - Feedback Shift Registers*) (cd.)

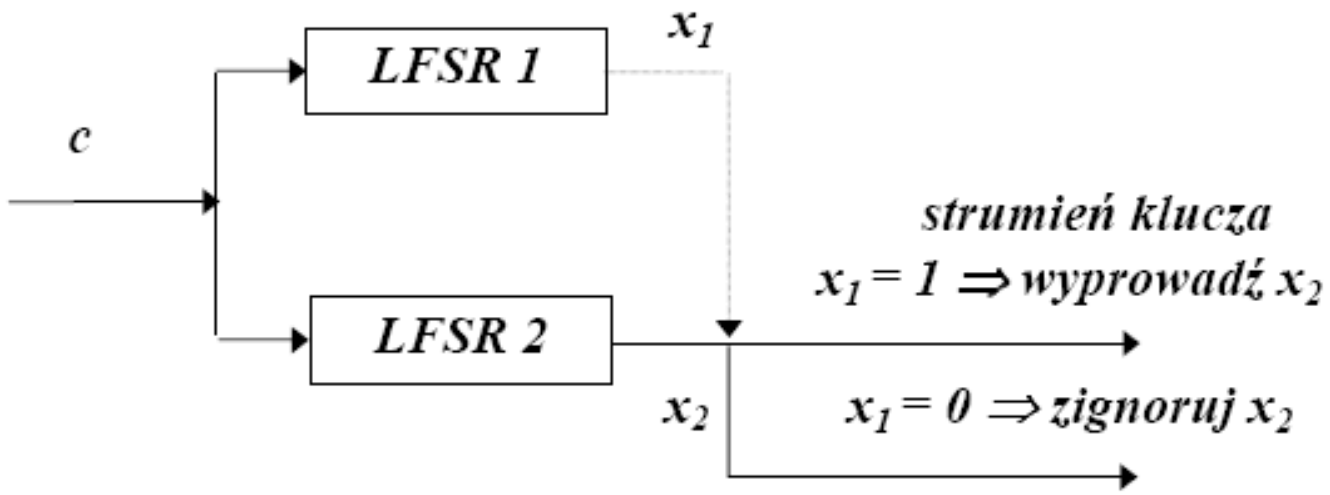
Generatory ze sterowanym wejściem taktującym (przykład):



c_i - sygnały taktujące *LFSR*

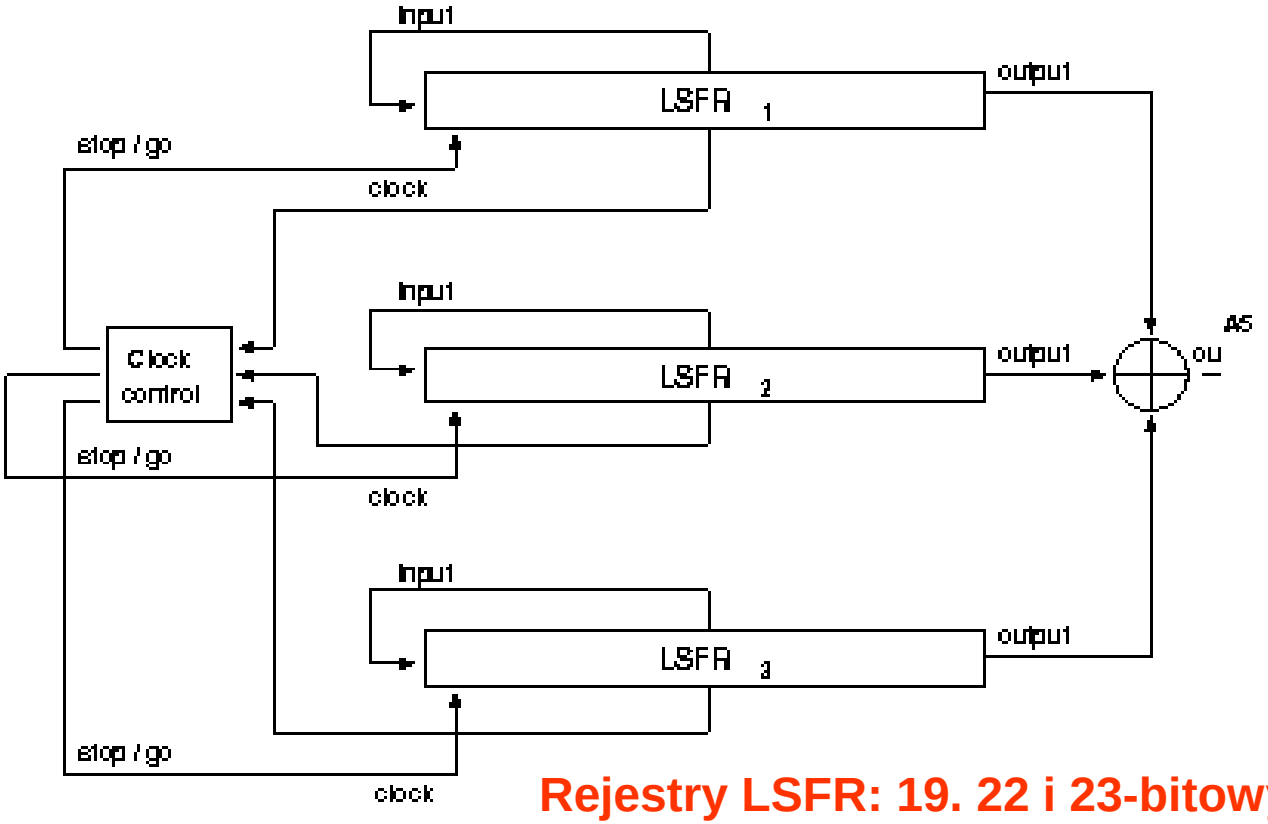
Generatory binarnych ciągów pseudolosowych wykorzystujące rejestry przesuwające (*FSR - Feedback Shift Registers*) (cd.)

Generatory z „kluczowanym” wyjściem – „shrinking generators” (przykład):



c - sygnał taktujący oba *LFSR*

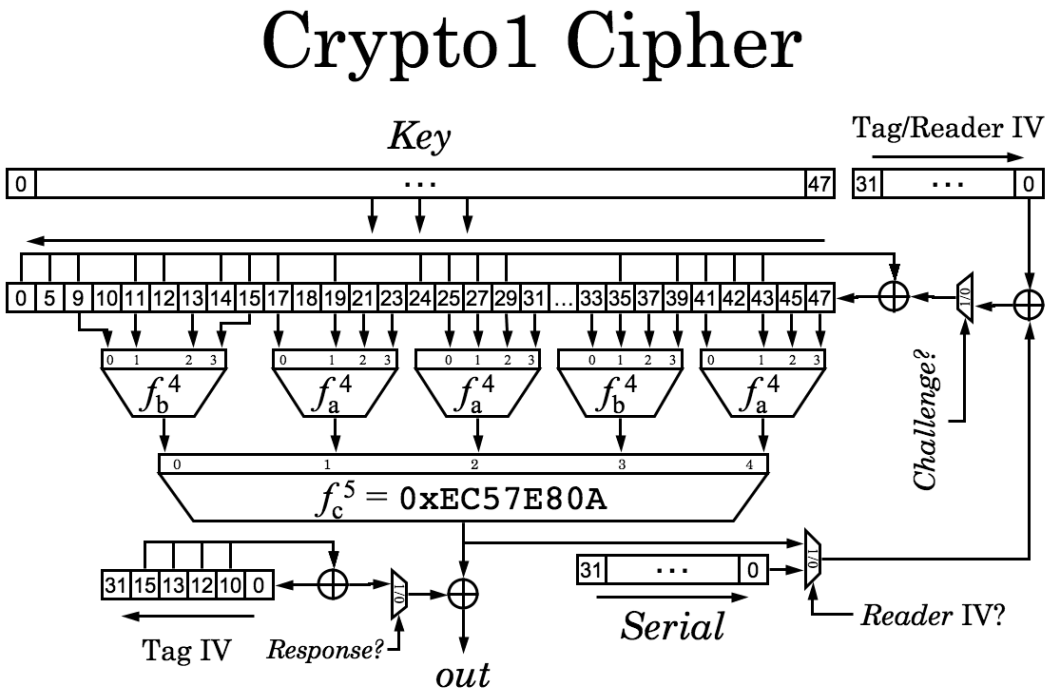
Algorytm szyfrujący A5 stosowany w telefonii GSM



Rejestry LSFR: 19. 22 i 23-bitowy

Zaprojektowany w 1987 r. i tajny.
1994 – „odkryto" architekturę A5/1 i A5/2.
1997 – „reverse engineering" ujawniający całą strukturę (J.D.Golic).
2021 – „still in use".

Algorytm szyfrujący CRYPTO1 stosowany w elektronicznych kartach zbliżeniowych z układem MIFARE®STANDARD



$f_a^4 = 0x9E98 = (a+b)(c+1)(a+d)+(b+1)c+a$

$f_b^4 = 0xB48E = (a+c)(a+b+d)+(a+b)cd+b$

Tag IV ⊕ Serial is loaded first, then Reader IV ⊕ NFSR

Zaprojektowany w 1993 r. i tajny.

2007 – „ Reverse-engineering a cryptographic RFID tag" (K.Nohl, H.Plötz, Starbug).
"the security of this cipher is ... close to zero„.

2021 – „still in use".

Koniec części 4