KRYPTOGRAFICZNE GENERATORY LICZB LOSOWYCH W ROZPROSZONYCH SYSTEMACH POMIAROWO--STERUJĄCYCH MAŁEJ MOCY

Paweł Czernik
Instytut Lotnictwa

Streszczenie

W pracy dokonano przeglądu i oceny generatorów liczb losowych pod kątem ich dopasowania do potrzeb i specyfiki Rozproszonych, Bezprzewodowych Systemów Pomiarowo – Sterujących (RBSPS), coraz powszechniej wykorzystywanych w awionice m. in. na pokładach samolotów bezzałogowych. W pracy rozważane są zasadnicze kategorie algorytmów generacji liczb losowych: generatory liczb „prawdziwie" losowych - fizyczne i programowe, generatory liczb pseudolosowych implementowane programowo jak i sprzętowo w układach FPGA. W ocenie wzięto pod uwagę poziom bezpieczeństwa algorytmu, efektywność obliczeniową i możliwości implementacji układowej w szybkich i oszczędnych pod względem zapotrzebowania na moc najnowszych systemach programowalnych.

1. WPROWADZENIE

Termin *Rozproszone, Bezprzewodowe Systemy Pomiarowo - Sterujące* (RBSPS) odnosi się do sieci tworzonych w trybie *ad hoc* przez nieduże, autonomiczne urządzenia nazywane węzłami pomiarowo-sterującymi, które oprócz funkcji komunikacyjnych realizują funkcje pomiarowe, przetwarzania, a często także funkcje wykonawcze. Komunikacja w sieci odbywa się za pomocą fal radiowych (lub innej bezprzewodowej techniki transmisyjnej). Węzy [1,2] poza elementami pomiarowymi, są wyposażone w niewielki mikrokontroler, pamięć, moduł radiowy oraz źródło zasilania. Ponieważ najczęściej są one zasilane bateryjnie [3], ich zasoby bywają znacznie ograniczone. Zastosowanie RBSPS nieustannie się poszerza [4]. Czujniki mogą być instalowane nie tylko na ziemi, ale też w wodzie i powietrzu. Bezpieczeństwo w większym lub mniejszym stopniu jest w zasadzie istotne we wszystkich rodzajach sieci. Ale biorąc pod uwagę, jak dużym ograniczeniom podlegają systemy RBSPS staje się oczywiste, że zapewnienie go w tego typu sieciach stanowi szczególnie trudne wyzwanie. Ograniczenia węzłów są konsekwencją wymagań co do ich rozmiarów, kosztów produkcji, lokalizacji, a przede wszystkim tego, aby stanowiły one autonomiczne terminale. Zasilanie bateryjne węzłów wiąże się z bardzo małymi zasobami energii. Od właściwego wykorzystania i gospodarowania energią zależy czas działania sieci, gdyż wymiana baterii często jest trudna lub w ogóle niemożliwa. Zarządzanie energią ma więc zasadnicze znaczenie. Z tego względu w większości rozwiązań rozproszone węzły charakteryzują się bardzo krótkim cyklem pracy. Terminale bezprzewodowe większą część czasu pozostają w stanie uśpienia zużywając o kilka rzędów mniej energii niż w czasie aktywności, kiedy są wykonywane pomiary oraz transmisja. W tego typu systemach występują również ograniczenia kluczowych parametrów węzłów jak: pamięć, moc obliczeniowa, przepływność łącza oraz zasięg przesyłu.

Krótki czas życia i mała wartość pojedynczej informacji mogłyby sugerować, że w systemach RBSPS nie ma potrzeby stosowania silnych zabezpieczeń. Jednak temat bezpieczeństwa w tych sieciach staje się coraz bardziej aktualny. Rosnąca liczba ataków szybko uświadomiła konieczność wprowadzania bardziej rygorystycznej polityki bezpieczeństwa. Do najbardziej spektakularnych można zaliczyć m. in. atak na system zabezpieczeń elektrowni Atomowej w Davise-Besse w USA w 2003 r, w tym samym roku został zaatakowany system sygnalizacji i sterowania kolei w USA, a rok później w Australii [5]. Zostało tez przeprowadzonych wiele mniejszych, skutecznych ataków. Jest więc oczywiste, że również sieci RBSPS w wielu przypadkach wymagają zachowania wysokiego poziomu bezpieczeństwa. Jednak ze względu na ich właściwości jest to szczególnie trudne. Atakujący może próbować zdobyć dostęp do informacji, do których nie jest uprawniony lub próbować zaburzyć, a nawet całkiem zablokować, komunikację w sieci.

Bezpieczeństwo dzisiejszych systemów kryptograficznych jest ściśle związane z generacją pewnych nieprzewidywalnych wartości. Przykładem tutaj może być generacja klucza do szyfru *„z kluczem jednorazowym" (ang. one-time pad*), klucza do algorytmu DES, AES lub jakiegokolwiek innego szyfru symetrycznego, liczb pierwszych do algorytmu RSA i innych losowych wartości używanych w szyfrach asymetrycznych oraz przy tworzeniu podpisów cyfrowych, czy uwierzytelnieniu typu *challenge-response.*

2. POJĘCIA PODSTAWOWE

W kontekście generowania liczb losowych niezwykle istotnym pojęciem jest tzw. *entropia* (ang. *entropy*), odnosząca się do nieodłącznej „niepoznawalności" danych wejściowych dla obserwatorów zewnętrznych. W takim wypadku, jeżeli bajt danych jest prawdziwie losowy, wówczas każda z 2^8 (256) możliwości jest równie prawdopodobna i można oczekiwać, że napastnik podejmie 2^7 prób jego odgadnięcia, nim uda mu się poprawnie zidentyfikować wartość. W takim przypadku mówimy, że bajt zawiera 8 bitów entropii. Z drugiej strony, jeżeli napastnik w pewien sposób odkryje, że bajt ma wartość parzystą, pozwoli mu to na zredukowanie liczby możliwych wartości do 2^7 (128) i w tym przypadku bajt posiada jedynie 7 bitów entropii. Możliwe jest wystąpienie entropii o wartościach ułamkowych. Jeżeli mamy jeden bit i istnieje 25% prawdopodobieństwo, że będzie on miał wartość 0 oraz 75%, że 1, napastnik ma większą szansę jej odgadnięcia niż w sytuacji, gdyby bit był całkowicie losowy. Jak ważną kwestią jest zapewnienie bezpieczeństwa systemu kryptograficznego opartego na produkcji entropii przez generator liczb losowych, powinien uświadomić następujący przykład. Stosujemy 256-bitowe klucze algorytmu AES, jednak są one wybierane za pomocą generatora PRNG zainicjowanego 56 bitami entropii. Wówczas wszelkie dane zaszyfrowane za pomocą 256-bitowego klucza AES byłyby równie mało bezpieczne, jak w przypadku ich zaszyfrowania przy użyciu 56-bitowego klucza algorytmu DES!

Niewystarczająca ilość entropii produkowanej przez generator liczb pseudolosowych jest częstym przypadkiem praktycznych nadużyć. Przykładem tego może być: w 1995r. przeglądarka Netscape – okazało się, że szyfrowanie tej przeglądarki może zostać łatwo złamane, ponieważ do szyfrowania kluczy 128-bitowych, w których rzeczywista entropia wynosiła jedynie 47 bitów. Algorytm „MIT-MAGIC-COOKIE" używał generatora kluczy, który był inicjowany jedną z 256 wartości. W kartach zbliżeniowych (bezdotykowych) standardu Mifare Classic firmy Philips (wykorzystywanych m. in. systemie Warszawskich Kart Miejskich), niewystarczająca ilość entropii generatora pozawala na atak przez „powtórzenie".

Generator liczb losowych (ang. *random number generator*) [6] jest to program komputerowy lub układ elektroniczny generujący liczby losowe. Ze względu na sposób generowania liczb losowych można wyróżnić dwa rodzaje stosowanych generatorów: sprzętowe (TRNG) działające na zasadzie ciągłego pomiaru procesu stochastycznego i programowe (PRNG).

2.1 Generator sprzętowy

Olbrzymią zaletą generatora sprzętowego, szczególnie ważną w kryptografii, jest jego nieprzewidywalność, wynikająca z nieprzewidywalności samego procesu fizycznego. Dlatego też często w literaturze są one określane jako „prawdziwe" generatory liczb losowych (ang. *True Random Number Generators*, w skrócie TRNG). Najczęściej wykorzystywane procesy losowe to szum termiczny oraz rozpad pierwiastka promieniotwórczego, choć możliwe są również inne rozwiązania. W [8] można znaleźć opis różnych metod generowania ciągów prawdziwie losowych opartych m. in. na: pomiarze czasu spoczynku klawiatury, szumu termicznego diody półprzewodnikowej, rozpadu radioaktywnego, niestałości częstotliwości własnej oscylatora, ładunku gromadzonego w kondensatorze polowym w ustalonym czasie i wiele innych …Oczywiście ciągi uzyskane z tych źródeł nie są dokładnie losowe: zazwyczaj obarczone są korelacją. Istnieją specjalne metody na usunięcie takich wad z ciągu. Przy ich wykorzystywaniu należy pamiętać o wpływie środowiska zewnętrznego na ich jakość. Odpowiednie urządzenia są produkowane przez różne firmy np. IBM, OMNISEC, SOPHOS.

2.2 Generator programowy liczb „prawdziwie losowych"

Generator takiego typu został zaimplementowany w jądrze systemu operacyjnego Linux (urządzenie */dev/random*). Algorytm pracy tego „urządzenia" przedstawia się następująco. Generator zbiera, dane losowe z otoczenia i gromadzi w „pojemniku losowości" (ang. *random pool*) o rozmiarze 512 bajtów. Danymi zbieranymi przez generator są np. kod naciśniętego klawisza lub odstęp czasu między kolejnymi przerwaniami. Generator szacuje w czasie dodawania danych, na ile dane te były losowe i na tej podstawie ustala ilość losowych danych w pojemniku (dalej zwana entropią). Kodu naciśniętego klawisza nie można uznać za losowy (więc jego dodanie do pojemnika nie zwiększa entropii). Także przerwania występujące w regularnych odstępach czasu nie dają losowych danych. W tym przypadku generator oblicza różnicę poprzedniego odstępu i obecnego i na tej podstawie określa ile losowych bitów zostało dodane (tak naprawdę, to tylko odstępy między przerwaniami mogą zwiększyć entropię). Przy dodawaniu do pojemnika 32-bitowego słowa, jest ono obracane cyklicznie o określoną ilość bitów i XOR-owane ze słowami na pozycjach i, $i + 7$, $i + 9$, $i + 31$, $i + 99$ (liczby to współczynniki pewnego wielomianu CRC; stosuje się też inne współczynniki), gdzie i to pozycja w pojemniku ostatnio wstawionego słowa. Otrzymana wartość jest znowu obracana cyklicznie (tym razem o jeden bit) i wstawiana na pozycję $i - 1$ (i jest liczone modulo rozmiar pojemnika). Przy dodawaniu słowa do pojemnika nie można wykorzystać kryptograficznych funkcji haszujących ze względu na wymaganą wydajność (słowa są dodawane przy prawie każdym przerwaniu). Przy pobieraniu bajtu z pojemnika (o ile entropia jest większa od zera), zwracana jest zawartość pojemnika przekształcona funkcją jednokierunkową (SHA lub MD5). Następnie cała zawartość pojemnika jest przekształcana jeszcze raz za pomocą SHA lub MD5. Oczywiście zmniejszana jest także entropia. Więcej informacji o generatorze zawartym w jądrze Linuksa jest w [9] oraz w źródłach jądra systemu [10]. Niestety wadą tego generatora jest jego mała wydajność, przez co jest stosowany jako jedynie zarodek (ang. *seed*), dla kryptograficznie bezpiecznych generatorów liczb pseudolosowych np. „urządzenia" */dev/urandom*.

2.3. Generator pseudolosowy

Generator pseudolosowy [11] (PRNG) to deterministyczny algorytm, który mając daną sekwencję binarną długości k, daje w wyniku sekwencję binarną długości $l >> k$, która „wydaje się" być losowa. Ciąg wejściowy zwany jest zarodkiem (ang. *seed*). Liczby pochodzące z tego generatora zwane są *liczbami pseudolosowymi*, ponieważ faktycznie nie są dziełem przypadku, lecz wynikiem skomplikowanych procedur matematycznych. Programowe generowanie liczb prawdziwie losowych wymaga posłużenia się ciągłym strumieniem próbek uzyskanych ze świata zewnętrznego.

Największą zaletą generatorów pseudolosowych jest ich szybkość, często też mają lepsze właściwości statystyczne niż generatory sprzętowe. Należy jednak zwrócić uwagę na fakt, że mając kontrolę lub znając wartości podawane na wejście generatora oraz jego stan wewnętrzny bez trudu można przewidzieć zwracane przezeń liczby.

Z tego powodu decydując się na zastosowanie w systemie kryptograficznym generatora liczb pseudolosowych należy zachować szczególną ostrożność przy doborze zarówno samego algorytmu, jak i sposobu inicjowania oraz rodzaju wartości podawanych na jego wejście.

2.4. Kryptograficznie bezpieczny generator pseudolosowy

Generatory używane w kryptografii poza tym, że powinny dobre w sensie statystycznym, muszą być także nieprzewidywalne, tak aby przeciwnik nawet znając poprzednie bity produkowane przez generator nie był w stanie przewidzieć kolejnych. Ta ważna cecha jest nazywana bezpieczeństwem kryptograficznym generatora (ang. *cryptographicaly secure*). Poniżej zostaną podane dokładniejsze definicje odnoszące się do tej własności:

•Definicja 1. *Mówimy, że generator pseudolosowy przechodzi wszystkie testy statystycznej o wielomianowym czasie, jeśli żaden test w czasie wielomianowym nie jest w stanie odróżnić sekwencji wyprodukowanej przez ten generator od prawdziwie losowej sekwencji z prawdopodobieństwem większym od* 12.

•Definicja 2. *Mówimy, że generator przechodzi test* next-bit; *jeśli nie istnieje żaden algorytm, który w wielomianowym czasie, znając pierwszych l bitów wyjściowej sekwencji s, może przewidzieć (l+*1) *bit sekwencji s z prawdopodobieństwem większym niż* $\frac{1}{2}$.

•Definicja 3. *Generator pseudolosowy, który przechodzi test* next-bit *(nawet pod pewnymi warunkami) jest nazywany* ***kryptograficznie bezpiecznym generatorem pseudolosowym*** **(CSPRNG)**. Powyższe definicje pochodzą z [11]. Można powiedzieć, że są one asymptotyczne-w każdej z nich mówimy o algorytmie działającym w czasie wielomianowym.

2.5. Generatory liczb w systemach RBSPS

W rozważanych systemach RBSPS wyłania się konieczność zastosowania programowej implementacji generatora liczb pseudolosowych o niewielkiej złożoności obliczeniowej. Taki sposób postępowania uzasadnia się uniwersalnością i przenośnością rozwiązania na niezliczoną gamę różnych typów węzłów.

W wielu przypadkach mamy do czynienia z brakiem możliwości dołączenia specjalizowanego układu (generatora fizycznego) ze względu na ograniczone możliwości zasilania, problematyczną kwestię zapewnienia optymalnych warunków pracy tego urządzenia, jak również dodatkowe koszy (dość wysokie) i komplikacja systemu, a przez to większa awaryjność.

W RBSPS nie mamy również możliwości zaimplementowana generatora na wzór Linuksowego urządzenia */dev/random*, ze względu prostotę architektury węzła, a przez to brak dostatecznej ilości źródła entropii.

3. PRZEGLĄD ISTNIEJĄCYCH REALIZACJI GENERATORÓW LICZB PSEUDOLOSOWYCH

3.1. Liniowe generatory kongruencyjne

Zdecydowanie najpopularniejszym i najlepiej poznanym typem generatorów pseudolosowych są opisywane za pomocą zależności liniowej [12]. Po raz pierwszy zostały zaprezentowane przez D.H. Lehmera w 1949 r. i od tej pory stanowią punkt wyjścia do konstruowania nowych generatorów. Generatory te pozwalają na uzyskanie ciągu liczb całkowitych z przedziału [0;M] w taki sposób, by wszystkie liczby występowały z jednakowym prawdopodobieństwem oraz by częstotliwość występowania wartości z każdego z podprzedziałów tego przedziału była w przybliżeniu jednakowa w czasie. Liniowy generator kongruencyjny możemy opisać wzorem:

$$x_{n+1} = (a * x_n + c) \bmod m,\ n \geq 0$$

Poszczególne parametry generatora są nazywane:
m - moduł, *m* > 0, *a* - mnożnik, $0 \le a \le m$, *c* - krok, $0 \le c \le m$, $0 \le x_0 < m$
x_0 - *wartość początkowa (ziarno)*,

Ze względów praktycznych będziemy dalej zakładać, że $a \ge 2$. Liniowe generatory kongruencyjne nazywamy w skrócie LCG (and. *linear congruential generators*) i oznacza się je *LCG(m,a,b,x*$_0$*)*. Nie wszystkie parametry dla LCG są dobre, a sposób ich wyznaczenia to dość obszerna teoria (odsyłam do [12]). Ideałem jest, by okres był możliwie długi; w przypadku *m* będącego liczbą pierwszą maksymalny okres to *m* – 1.

Ważną sprawą jest również aby moduł był liczbą pierwszą. Dla *m* będącego liczbą złożoną zachodzi bardzo niekorzystna właściwość, powodująca, że najmniej znaczące bity elementu ciągu są „mało" losowe.

Jeśli *d* jest dzielnikiem *m* i $y_n = x_n \bmod d$, to $y_{n+1} = \left(a_{y_n} + c\right) \bmod d$.

To sprawia, że tak wygodne w komputerowej implemetacji $m = 2^E$ mogą być używane, jeśli pominiemy najmłodszy bit. Generatory LCG są bardzo popularne i szeroko stosowane ze względu na szybkość działania i prostotę implementacji. W bibliotekach języka ANSI-C jest zaimplementowana funkcja *rand()* jako *LCG* (2^{31}, 1103515245, 12345, 12345), w bibliotekach *Fortranu* jest wykorzystana dla funkcji RAN *LCG* (2^{31}, 630360016, 0, 0).

Warta podkreślenia jest bezużyteczność prostych generatorów liniowych w kryptografii. Weźmy dla przykładu generator z ANSI-C. Przypuśćmy, że mamy algorytm generujący klucz dla jakiegoś systemu kryptograficznego, dla którego źródłem losowości jest funkcja *rand()*, a dodatkowo algorytm generowania klucza jest znany (dostępny kod źródłowy programu szyfrującego lub nawet plik wykonywalny, bo można go zdesalemblować). Aby wygenerować odpowiednio długi, losowy klucz (np. 1024 bity) algorytm korzysta wielokrotnie z funkcji *rand()*. Niestety wynik kolejnego wywołania rand jest jednoznacznie wyznaczony przez poprzednie. Możliwe ciągi losowe użyte w czasie działania algorytmu są jednoznacznie wyznaczone przez pierwszą wylosowaną liczbę. Widać więc, że różnych kluczy, które mogą być wygenerowane przez nasz algorytm jest co najwyżej 2^{31}. Złamanie klucza może polegać na sprawdzeniu jednego z 2^{31} kluczy, a nie jednego z 2^{1024}. W rzeczywistości jest jeszcze gorzej. Popularną metodą inicjowania generatora (funkcja *srand()*), czyli ustawianie wartości x_0, jest wykorzystanie zegara systemowego (zwykle podającego czas w sekundach). Przy takim rozwiązaniu, jeśli znamy przybliżony czas (np. dzień), kiedy klucz był generowany, możemy zmniejszyć ilość kluczy do parudziesięciu tysięcy. Wtedy szyfr może zostać złamany w ciągu paru sekund.

3.2. Uogólniony generator afiniczny

Uogólniony generator afiniczny [13], w skrócie MRG (ang. *multiple recursive generators*), to uogólniona wersja generatorów LCG, z tzw. opóźnieniem lub pamięcią.

Generator ten rzędu *k*można opisać równaniem:

$$x_n = \left(a_1 x_{n-1} + \ldots + a_k x_{n-k} + b\right) \bmod m,$$

gdzie: *m* i *k* są liczbami całkowitymi, a, b i $a_i \in Z_m$.

Ze względu na efektywność obliczeniową, jest zalecane używanie co najmniej dwóch niezerowych współczynników a_i oraz $b = 0$. To daje nam bardzo szybki generator, który przy odpowiednio dobranych współczynnikach może mieć okres równy $m^k - 1$. Jednak badania wykazały, iż zbyt mała liczba niezerowych współczynników w porównaniu do wartości *k* prowadzi do otrzymania generatora o defektach statystycznych, co bezwzględnie dyskwalifikuje go w zastosowaniach kryptograficznych.

3.3. Generator Fibonacciego i pokrewne

Wady generatorów LCG byty bodźcem do badań podobnego typu generatorów, opartych na „mniej" liniowych zależnościach.

Generator Fibonacciego [6], [7] opisany jest następująco: $x_{n-1} = (x_n + x_{n-1}) \text{modm}$. Daje on zazwyczaj dłuższy okres niż *m*, jednak jakość produkowanego ciągu jestniezadowalająca.

W wyniku dogłębnych dłuższych badań w 1958r. została zaproponowana jego modyfikacja opisana następującym wzorem: $x_n = (x_{n-24} + x_{n-55}) \text{modm}, \text{n} \geq 55$, gdzie *m* jest parzyste, zaś $x_0,\ldots,x_{54}$ są dowolnymi liczbami, z których przynajmniej jedna jest nieparzysta. Ten generator charakteryzuje się długim okresem i dużą efektywnością obliczeniową. Generator ten był uważany stosunkowo długo za jeden z „lepszych" generatorów losowych dopóki w latach 90 nie wykazano, że generowany ciąg ma duże wady: jeśli byśmy generowali pojedyncze bity (*m* = 2) to istnieje 51% prawdopodobieństwo, że w ciągu będą przeważały jedynki, co zdyskwalifikowało go z zastosowań w kryptografii.

3.4. Generator wykorzystujący odejmowanie-z-pożyczką (SWB)

Generator SWB jest oparty na następujących rekurencjach:

$$x_n = (x_{n-r} - x_{n-k} - c_{n-1}) \; mod \; m, \; cn = I \, [\, x_{n-r} - x_{n-k} - c_{n-1} < 0],$$

gdzie $k > r$. c_n nazywane jest pożyczką.

Zaletą tego typu generatora jest jego szybkość i prostota. Niestety nie przechodzi on niektórych nowszych testów statystycznych – niezależności, co dyskwalifikuje go z zastosowań w kryptografii.

3.5. GENERATOR WYKORZYSTUJĄCY *MNOŻENIE-Z-PRZENIESIENIEM* (MWC)

Generator MWC jest oparty na następujących rekurencjach:

$$x_n = \left(a_1 x_{n-1} + \ \ldots \ + a_k x_{n-k} + c_{n-1}\right) \; mod \; m$$

$$c_n = \left(a_1 x_{n-1} + \ \ldots \ + a_k x_{n-k} + c_{n-1}\right) \; div \; m,$$

gdzie *div* oznacza dzielenie liczb całkowitych

Generator ten charakteryzuje się w miarę prostą i szybką implementacją, długim okresem generowanego ciągu, zadawalającymi właściwościami statystycznymi wystarczającymi na potrzeby zaawansowanych symulacji zjawisk fizycznych.

W zastosowaniach kryptograficznych zaleca się korzystanie z niego tylko w ostateczności, nie nadaje się do zastosowań w układach małej mocy.

3.6. Generatory kongruencyjne nieliniowe

Generatory nieliniowe [7] są odpowiedzią na liniowe własności i przewidywalność generatorów kongruencyjnych. Generatory te są definiowane przy użyciu rekurencji pierwszego rzędu:

$$x_{n+1} = f\left(x_n\right), n \geq 0,$$

gdzie *f* jest przekształceniem różnowartościowym w ciele i *p* jest liczbą pierwszą

Ich podstawową i największą zaletą jest brak liniowości, obecnej w sekwencjach generowanych przez pozostałe klasy generatorów.

Przykładem tego typu generatorów są tzw. generatory kwadratowe, które się określa następującą zależnością: $x_{n+1} = (\text{dx}^2_{\text{n}} + ax_n + c) \; mod \; m$.

Przy odpowiednich parametrach *d, a, c* ta rodzina generatorów ma dużo lepsze właściwości niż rodzina LCG, a nakład kosztów związanych z możeniem nie jest dużo większy.

Jak można prosto zauważyć generatory QCG z powodu potęgowania (mnożenia) będą wolniejsze od generatorów LCG.

Przypadkiem szczególnym tego typu generatorów jest tzw. generator Blum-Blum-Shub (BBS), znany także jako x^2 *mod n,* został wyróżniony z powodu szczególnych właściwości.Jako jedyny z „klasycznych" generatorów jest *kryptograficznie bezpieczny* (ang. *cryptographically secure*). Generator ten został przedstawiony w pracy [16].

Definicja tego generatora jest następująca: niech *n* będzie iloczynem dwóch różnych liczb pierwszych *p i q,* takich, że $p \equiv 3 \bmod 4$ *i* $q \equiv 3 \bmod 4$.

Niech x_0 będzie dowolnym pierwiastkiem kwadratowym z Z_n. Sekwencję losową uzyskujemy poprzez obliczenie $x_{n+1} = x^2{}_n \ mod\ n$ i wybranie z uzyskanej liczby (x_{n+1}) najmniej znaczącego bitu.

Ważną cechą tego generatora jest fakt, że gdy należy obliczyć *i-ty* element sekwencji nie ma potrzeby koniecznie obliczać wszystkich poprzedzających go elementów: $x_i = x^2{}_{i-1} \ mod\ n = (x^2{}_{i-2} \ mod\ n)^2 \ mod\ n = x^2{}_0{}^i \ mod\ n = x^2{}_0 i \ mod\ \lambda(n) \ mod\ n,$ gdzie λ jest to funkcja *Carmichaela*, której wartością dla danej liczby *n* jest najmniejsza liczba, taka, że liczba względnie pierwsza z *n* podniesiona do potęgi przystaje do 1 *mod n.* Zatem mając mając x_0 i znając rozkład *n* możemy obliczyć dowolny element wygenerowanej sekwencji. Generator ten ma bardzo ważną cechę:jest kryptograficznie bezpieczny!

Podstawą tego twierdzenia jest założenie o problemie logarytmu dyskretnego i reszty kwadratowej, co jest na obecną chwilę zadaniem nierozwiązywalne w czasie wielomianowym (przy doborze odpowiednio dużych liczb).

Niestety dużą wadą tego algorytmu jest fakt, iż wygenerowanie każdego pseudolosowego bitu wymaga jednego podniesienia do kwadratu modulo, co powoduje złożonością obliczeniową oszacowaną jako $0\left(\left(\log 2\right)\left(\log^2 n\right)\right)$, drugim niekorzystnym aspektem tego generatora jest niska wydajność procesu generacji.

Właśnie z względu na te wady generator BBS jest nieakceptowalnym w generacji liczb pseudolosowych dla systemów RBSPS o małej mocy, mimo iż jego niepodważalną zaletą jest fakt, że przy odpowiednich założeniach można udowodnić, że odróżnienie jego wyników od szumów wymaga nakładu obliczeniowego porównywalnego z faktoryzacją liczby N.

W celu poprawy niskiej wydajności generatora BBS podjęte zostały badania w wyniku których powstał między innymi generator zaprezentowany w [17].

Ten generator opiera się tylko na założeniu trudności faktoryzacji. Jest bardziej efektywny od innych generatorów tej klasy, to jest np. [22], [23], [24].

Generator ten działa następująco:

Niech $y \in Z_n$ będzie losowym ciągiem wejściowym (ziarnem).

Dla *i* od 0 do $p(n)$ wykonaj następującą operację:

Podstaw $y = g^{y_{n...}y_{m+s}} * s^{y_{m+z}} * s^{y_{m+1}} \bmod N$. Do ciągu wyjściowego dołącz $y_m...y_1$. Na wejście $y \in Z_n$ ten generator produkuje *m* pseudolosowych bitów na jedno modularne potęgowanie z *c-bitowym* wykładnikiem, przy spełnionych następujących założeniach:

P i Q są silnymi liczbami pierwszymi tej samej długości; $N = P * Q$ i $ceil(log_2 N) = n$; *g* będzie losową resztą kwadratową *mod N*; *s* będą kwadratowymi nieresztami takimi, że: $J_N(s)=1$ i $J_N(s)=-1$; *m* jest funkcją *n*: $m=m(n)=ceil(n/2) + 0(log\ n)$; $c=c(n)=n - m$; p(*) będziewielomianem.

Na podstawie jego własności można udowodnić, że jest on o $ceil(n/2) + 0(log\ n)$ wydajniejszy niż BBS, który dla każdego *n* w jednej iteracji produkuje tylko 1 bit losowych danych, ale niestety wciąż jest on zbyt wymagający obliczeniowo dla systemów RBSPS małej mocy.

4. GENERATORY BAZUJĄCE NA SZYFRACH BLOKOWYCH

Pracując nad profesjonalnymi systemami kryptograficznymi, zazwyczaj mamy możliwość skorzystania z dobrego szyfru blokowego, takiego jak 3DES czy AES, działającego w trybie licznikowym (CTR). Przez co zyskujmy możliwość uzyskania kryptograficznie silnego generatora liczb pseudolosowych. CTR jest trybem strumieniowym, który polega na szyfrowaniu danych przez generowanie strumienia klucza i jego łączenie z jawnym tekstem za pomocą operacji XOR.

Tryb szyfrowania CTR generuje w jednym cyklu strumień klucza dla jednego bloku przez szyfrowanie takich samych (poza stale zmienianym licznikiem) jawnych tekstów. W ogólności wartość licznika rozpoczyna się od zera i jest zwiększana sekwencyjnie. Dla każdego klucza wykorzystanego podczas szyfrowania wartość licznika musi być unikalna. Generator liczb pseudolosowych typu AES działający w trybie CTR tworzy losowe dane wyjściowe po jednym bloku, szyfrując licznik, który jest zwiększany po każdej operacji szyfrującej. Ziarno (ang. *seed*) powinno być co najmniej tak duże, jak rozmiar klucza szyfru, ponieważ będzie używane w celu kluczowania szyfru blokowego.

Ponadto warto posiadać dodatkowe dane dla „drugiego" ziarna, które ustawią pierwszą wartość tekstu jawnego (licznika). Ten typ generatora charakteryzuje się na dzień dzisiejszy teoretycznie największym poziomem bezpieczeństwa kryptograficznego, niestety jest on stosunkowo powolny i cechuje go duża złożoność obliczeniową, co jest niestety faktem eleminującym go z zastosowaniach w systemach RBSPS małej mocy.

5. GENERATORY OPARTE NA REJESTRACH PRZESUWNYCH ZE SPRZĘŻENIEM ZWROTNYM

Większość generatorów kluczy szyfrujących (ang. *keystream generators*) dla szyfrów strumieniowych zbudowana jest z rejestrów przesuwnych ze sprzężeniem zwrotnym. Ich największą zaletą jest łatwość implementacji w strukturze sprzętowej. Także ich własności są dobrze zbadane i udokumentowane [7]. Zaproponowano także wiele generatorów zbudowanych z różnych kombinacji rejestrów, o bardzo ciekawych własnościach.

Podstawowy typ tego generatora jest oparty o liniowy rejestr przesuwający ze sprzężeniem zwrotnym (ang. *Linear Feedback Shift Register - LFSR*) o n stanach, który składa się z rejestru przesuwającego $R = (r_n, r_{n-1}, ..., r_1)$ i rejestru sprzężeń $T = (t_n, t_{n-1}, ..., t_1)$. Każdy element r_i i t_i reprezentuje jedną cyfrę dwójkową. W każdym kroku bit r_1 jest przyłączany do łańcucha klucza, bity $r_n, ..., r_2$ są przesuwane w prawo, a nowy bit obliczony z wartości T i R jest wprowadzony z lewej strony rejestru.

Następny stan rejestru $R' = (r'_n, r'_{n-1}, ..., r'_1)$ jest obliczany ze wzoru:

$$r_i^{'} = r_{i+1} \quad dla\ i = 1, \quad ..., n-1$$

$$\sum_i$$

$$r_n^{'} = TR = t_i r_{n-1} \bmod 2 = t_1 r_{n-1} \oplus t_2 r_{n-2} \oplus ... \oplus t_n r_1$$

Ziarnem tego generatora jest stan początkowy rejestru czyli n bitów.

Wielomian $T(x) = t_n x^n + t_{n-1}x^{n-1} + ... + t_1 x + 1$ utworzony z elementów wartości T zwiększonych o 1 nazywany wielomianem charakterystycznym rejestru. Rejestr LFSR o n stanach może wygenerować pseudolosowy, nieokresowy ciąg 2^n-1 bitów. Warunek taki jest spełniony, wtedy gdy wielomian charakterystyczny tego rejestru jest wielomianem pierwotnym.

Natomiast wielomianem pierwotnym stopnia n-tego nazywamy wielomian nierozkładalny, który dzieli wielomian $x^{2^n-1}+1$, ale nie dzieli wielomianów x^d+1 dla każdego d dzielącego 2^n-1

Wadą rejestrów LFSR jest to, że kolejne bity generowanego ciągu są ze sobą liniowo powiązane. Okazuje się, że do określenia ciągu odczepów T gałęzi sprzężenia zwrotnego wystarczy znajomość tylko $2n$ (n – długość rejestru) bitów wyjściowych generatora. Tak więc znalezienie struktury generatora opartego na pojedynczym rejestrze LFSR, czyli „złamanie" go

jest jak najbardziej możliwe wystarczy tylko znajomość 2n bitów z wygenerowanego ciągu.

Dlatego powstały modyfikacje tego typu generatorów w znaczny sposób poprawiające ich właściwości. Do tego typu generatorów należą m. in.: generator Geffe, generator *alternating step,* czy generator *shrinking*.

5.1. Generator Geffe

Wykorzystuje trzy rejestry LFSR powiązane ze sobą nieliniowo przez multiplekser. Działanie tego generatora można opisać wzorem $k=a_3a_1 + (NOT\ a_1)\ a_2$. Liniowa złożoność tego generatora jest równa $\Lambda=(L_1+1)\ L_2+L_1L_3$ gdzie L_i oznacza długość rejestru i. Okres ciągu wyjściowego z generatora jest równy najmniejszej wspólnej wielokrotności trzech okresów generatorów składowych. W najlepszym przypadku, gdy długości rejestrów składowych są względnie pierwsze jest on równy iloczynowi okresów generatorów składowych $T=T_1T_2T_3$.

Generator ten jest kryptograficznie słaby. Okazuje się, że ciąg wyjściowy jest w 75% czasu równy ciągowi wyjściowemu LFSR-1 i w 25% czasu ciągowi wyjściowemu LFSR-2. Przy tego rodzaju skorelowaniu ten generator może być łatwo złamany.

Jeżeli na przykład wszystkie trzy wielomiany charakterystyczne są trójmianami i wielomian najwyższego stopnia ma stopień n, to wystarcza fragment ciągu wyjściowego o długości 37n bitów do zrekonstruowania wewnętrznych stanów wszystkich trzech rejestrów, co dyskwalifikuje go z zastosowań w kryptograficznie bezpiecznych systemach małej mocy.

5.2. Generator alternating step

Zbudowany [14] z trzech rejestrów LFSR o różnej długości. LFSR-1 taktuje rejestr LFSR-2 jedynką, a rejestr LFSR-3 zerem. Ciąg wyjściowy jest sumą modulo 2 ciągów wyjściowych LFSR-2 i LFSR-3.

Jeśli długości wszystkich trzech rejestrów są względnie pierwsze, a także okresy generatorów LFSR-2 i LFSR-3 to okres generatora jest równy iloczynowi okresów wszystkich rejestrów $T = T_1T_2T_3$, a złożoność liniowa jest ograniczona z dołu zgodnie ze wzorem:

$$\Lambda \geq (L_2+L_3-2)T_1.$$

Znaleziono metodę ataku korelacyjnego na LFSR-1, ale nie zmniejsza to wysiłku wymaganego do złamania generatora. Aby zapewnić maksymalne bezpieczeństwo tego generatora składowe rejestry powinny być *maksymalnej długości,* zaś ich długości być parami względnie pierwsze, wtedy nadaje się on do zastosowań w systemach kryptograficznych.

Bardzo dużą zaletą tego generatora jest jego prostota i dobra wydajność. W każdym cyklu zegara produkuje on losowy bit danych, co gwarantuje stałą w czasie przepływność generowanego ciągu losowego i dodatkowo zapewnia odporność na ataki typu „timing attacks".

Wadą jego jest potrzeba zastosowania aż trzech rejestrów LFSR, co znacznie powiększa jego złożoność pamięciową projektowanego systemu. Złożoność obliczeniową tego algorytmu można oszacować (przyjmując za operacje dominujące XOR i iloczyn iloczyn) sumę operacji XOR równą 4 i 3 operacje dodawania, w sumie 7 operacji dominujących potrzebnych do wygenerowania bloku danych, a więc złożoność określić można jako 0(n). Natomiast złożoność pamięciową można oszacować jako: trzy N-*elementowe* rejestry LFSR.

Wydaje się to być na dzień dzisiejszy akceptowalny generator licz pseudolosowych do zastosowań w systemach RBSPS małej mocy, zapewniający wystarczające bezpieczeństwo kryptograficzne, stałą w czasie wydajność generowanego ciągu i niskie zapotrzebowanie na moc obliczeniową węzła.

5.3. Generator shrinking

Zbudowany [15] z dwóch rejestrów LFSR. Generator ten używa bitu wyjściowego LFSR-1 aby dodać do sekwencji wyjściowej bit wyjściowy LFSR-2 lub odrzucić. Tak więc sekwencja produkowana przez ten generator składa się z wybranych bitów rejestru LFSR-2. Jeśli LFSR-1

i LFSR-2 mają maksymalny okres: odpowiednio T_1 i T_2 to okres sekwencji tego generatora wynosi $\left(2^{T_2}-1\right)*2^{T_1-1}$.

Złożoność liniowa tego generatora (za cały okres) jest ograniczona następująco:

$$T_2*2^{T_1-2} < T\left(x\right) \le T_2*2^{T_1-1}$$

Rozważając kwestię bezpieczeństwa tego generatora należy przeanalizować przypadek, gdy składowe LFSR generatora mają długość L_1 i L_2. Jeśli wielomiany obu LFSR-ów są znane, zaś warunki początkowe nie, to najlepszy atak prowadzący do odkrycia warunków początkowych ma $0\left(2^{L_1}*L_2^3\right)$ kroków.

Jeśli jednak wielomiany charakterystyczne obu rejestrów są tajne i uzależnione od zmiennych najlepszy atak potrzebuje kroków $0\left(2^{2L_1}*L_1*L_2\right)$.

Istnieje również atak wykorzystujący złożoność liniową, który wymaga $0\left(2^{L_1}*L_2^2\right)$ kroków. Nie jest wtedy istotne czy wielomian charakterystyczny jest jawny, czy tajny, ale ten atak wymaga $2^{L_1}*L_2$ kolejnych bitów sekwencji wyjściowej, co oznacza, że jest dosyć trudny do przeprowadzenia dla dużych LFSR. Aby *shrinking generator* był bezpieczny należy tak dobrać LFSR aby były maksymalnej długości, ich długości powinny być względnie pierwsze, a wielomiany charakterystyczne tajne. Jeśli założymy, że $L_1 = l$ i $L_2 = l$, to poziom bezpieczeństwa takiego generatora wynosi około 2^{2l}.

Dobór odpowiednio dużych rejestrów gwarantuje odpowiednie bezpieczeństwo. Niestety potrzeba wykorzystania dwóch długich rejestrów LFSR, powoduje duże zapotrzebowanie na pamięć węzła, co może czasem okazać się kwestią wykluczają zastosowanie tego generatora w niskomocowych systemach RBSPS.

6. GENERATORY BAZUJĄCE NA SZYFRACH STRUMIENIOWYCH

Szyfry strumieniowe [11], same w sobie stanowią generatory liczb pseudolosowych, w których klucz (oraz, jeśli ma zastosowanie, wektor inicjalizacji) stanowią ziarno.

Najczęściej takiego typu generatory liczb pseudolosowych implementuje się w oparciu o dobrze znany szyfr RC4. RC4 generuje pseudolosowy strumień bitów (strumień szyfrujący). W celu zaszyfrowania wykonywana jest operacja XOR na tekście oryginalnym i strumieniu szyfrującym. Niestety mając na uwadze dobrze znane wady tego szyfru zalecane jest zastąpienie go przez SNOW 2.0. Ze względu na popularność szyfru RC4, jest on obecnie bardzo często praktycznie wykorzystywany w rzeczywistych systemach, chociaż nie jest on równie dobry jak SNOW [18]. Mimo wszystko szyfr strumieniowy RC4 stanowi akceptowalny generator liczb pseudolosowych i jest dość szybki, jeśli zmiana klucza jest dokonywana niezbyt często (jest to szczególnie przydatne w przypadku, gdy potrzebnych jest bardzo wiele liczb). Jeżeli zmiana klucza następuje często w celu umknięcia ataków tropienia, użycie szyfru blokowego może zapewnić większą szybkość działania.

Jeśli zależy nam na możliwie najszybszym rozwiązaniu w zakresie generacji liczb pseudolosowych, powinniśmy rozważyć zastosowanie algorytmu SNOW 2.0, który jest obecnie uważany za bardzo dobry mechanizm kryptograficzny [19]. Mechanizm ten posiada znacznie większy margines bezpieczeństwa niż bardzo popularny RC4, a poza tym jest od niego szybszy. Przed dokonaniem ostatecznego wyboru należy zdać sobie sprawę, że algorytm SNOW 2.0 jest stosunkowo nowy. Zastosowano w nim wiele znaczących udoskonaleń istniejących reguł. Poza faktem, że szyfr ten gwarantuje wyższy poziom bezpieczeństwa niż wspomniany przed chwilą szyfr RC4, SNOW jest także szybszy - jego zoptymalizowana wersja napisana w języku C działa około dwa razy szybciej od dobrze zoptymalizowanej implementacji szyfru RC4 napisanej w asemblerze. Nowy szyfr strumieniowy był co prawda poddawany dosyć dokładnym analizom, podczas których nie znaleziono w algorytmie SNOW żadnych poważnych uchybień, a kilka mniej

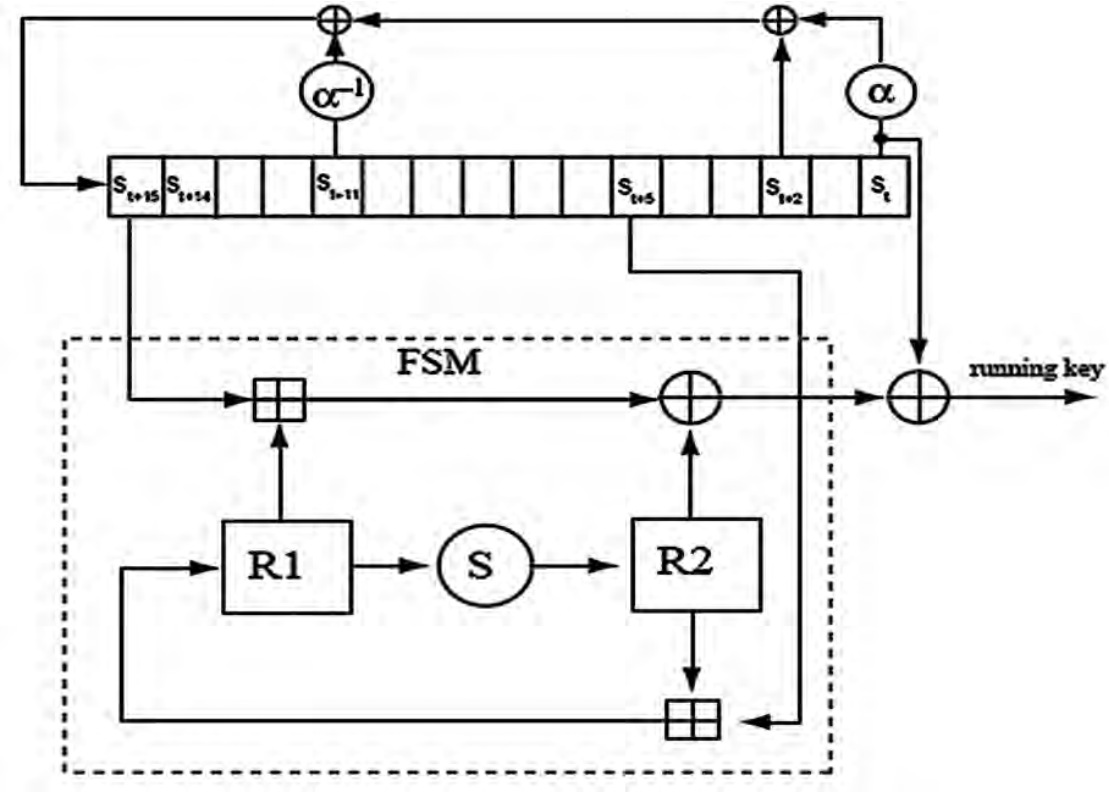
istotnych niedociągnięć teoretycznych z pierwszej wersji zostało szybko naprawionych (w ten sposób powstał szyfr SNOW 2.0).

6.1. SNOW 2.0

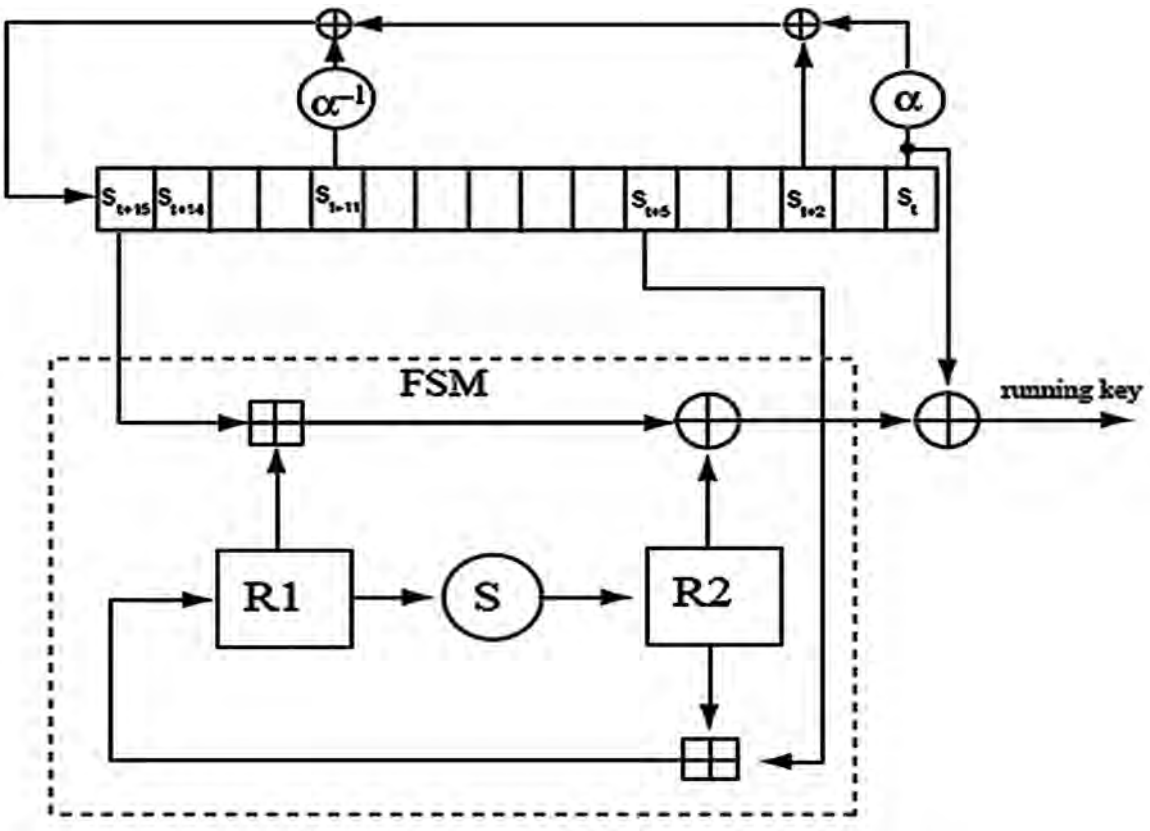
Algorytm SNOW 2.0 bazuje na 32 bitowym słowie. Umożliwia generację ciągu losowego o długości 128, lub 256 bitów. W procesie kodowania wykorzystuje rejestr LFSR o długości 16-tu słów 32 bitowych i *automat stanów* – FSM (ang. *Finite State Machine*) z dwoma 32 bitowymi rejestrami, gdzie LFSR „zasila" następny stan automatu FSM. Głównym zadaniem automatu FSM jest wprowadzenie do algorytmu przekształcenia nieliniowego, bazującego na transformacie *MixColumns* Rijndael-a. Algorytm ten cechuje krótka fraza inicjująca i niezwykle wysoka wydajność zarówno w implementacji na procesorach 32 bitowych, jak i w sprzęcie [25]. Na poniższym rysunku (Rys. 1) został zamieszczony schemat pracy tego algorytmu. Złożoność obliczeniową tego algorytmu można oszacować (przyjmując za operacje dominujące XOR i dodawanie) sumę operacji XOR równą 7 (w tym 3 operacje XOR w transformacie *MixColumns*) i dwie operacje dodawania, w sumie 9 operacji dominujących potrzebnych do wygenerowania bloku danych, a więc złożoność określić można jako 0(*n*).

Natomiast złożoność pamięciową można oszacować jako: 16-to elementowy rejestr LFSR o 32 bitowym słowie i dwa 32 bitowe rejestry automatu FSM, co daje razem 72 bajty niezbędnej pamięci operacyjnej.

Wydaje się że na dzień dzisiejszy najlepszy jest to najlepszy generator liczb pseudolosowych do zastosowań w systemach RBSPS małej mocy, zapewniający zarówno duże bezpieczeństwo kryptograficzne, wysoką wydajność generowanego ciągu, niezbyt wygórowane zapotrzebowanie na moc obliczeniową, jak i zajętość pamięci operacyjnej węzła.



Rys. 1. Schemat pracy algorytmu SNOW 2.0



Rys. 2. Schemat pracy algorytmu A5/1

6.2. A5/1

Częstym praktycznym rozwiązaniem dla klasy algorytmów szyfrów strumieniowych wykorzystywanych głównie w systemach GSM jest algorytm A5/1. Zbudowany jest on na bazie trzech rejestrów LFSR o różnej długości, gdzie wielomiany charakterystyczne to wielomiany pierwotne. Generuje on 228 bitów strumienia klucza. Na powyższym rysunku (Rys. 2) został zamieszczony schemat pracy tego algorytmu.

Jego główną zaletą jest większa wydajność w stosunku do SNOW 2.0 [25]. Niestety jest on podatny na znane ataki kryptoanalizy opisane w [26], [27], [28], [29], co dyskwalifikuje go w rozważaniach w kryptograficznie bezpiecznych systemach RBSPS.

7. GENERATORY BAZUJĄCE NA KRYPTOGRAFICZNEJ FUNKCJI SKRÓTU

Generatory te produkują pseudolosowe liczby na podstawie ziarna (zarodka), z którego jest obliczana funkcja skrótu. Jednym z bardzo bezpiecznych sposobów użycia kryptograficznej funkcji skrótu w kryptograficznych generatorach liczb pseudolosowych jest wykorzystanie algorytmu HMAC (ang. *Keyed-Hash Message Authentication Code*) [20] w trybie licznikowym. HMAC jest to kod MAC (ang. *Message Authentication Code*) z wmieszanym kluczem tajnym zapewniający zarówno ochronę integralności jak i autentyczności danych. Implementacje HMAC są oparte o standardowe kryptograficzne funkcje skrótu takie jak SHA-1 czy MD5.

Najczęściej popełnianym błędem w przypadku podejmowania prób użycia funkcji skrótu jako kryptograficznego generatora liczb pseudolosowych jest ciągłe tworzenie skrótu na podstawie tej samej porcji danych. Takie podejście ujawnia wewnętrzny stan generatora przy każdym przekazaniu na wyjście danych. Załóżmy, na przykład, że stanem wewnętrznym jest pewna wartość X oraz że generujemy porcję danych wyjściowych Y poprzez utworzenie skrótu z X. Kolejnym razem, gdy będą potrzebne losowe dane, ponowne utworzenie skrótu z X da ten sam rezultat i każdy napastnik, który będzie znał ostatnie dane wyjściowe generatora, będzie mógł określić kolejne porcje danych, jeżeli są one generowane poprzez tworzenie skrótu z Y.

Kolejną ważną kwestią dla tego typu generatora jest fakt, że jeżeli stosowana funkcja skrótu tworzy *n*-bitowe porcje danych wyjściowych oraz nie posiada żadnych praktycznych słabości, nie należy używać generatora po uruchomieniu algorytmu MAC więcej niż $2^{n/2}$ razy. Przykładowo, w przypadku algorytmu SHA-1 generator taki nie powinien sprawiać żadnych problemów dla co najmniej $2^{80} * 20$ bajtów.

W artykule [30] została opisana niskomocowa implementacja (język VHDL przy wykorzystaniu niskomocowej biblioteki ASIC) algorytmu HMAC bazującego na funkcji SHA-1 w systemach TCG (ang. *Trusted Computing Group*). Jak twierdzą autorzy układ pracował z częstotliwością 25MHz, potrzebował 1.750 cykli zegarowych na generację 512 bitów klucza, zajął 16.320 bramek i pobierał prąd rzędu 2,58mA przy zasilaniu napięciem 2,5V w systemach 0.25μm CMOS.

Generatory tego typu wydają się być akceptowalnym rozwiązaniem dla systemów RBSPS, choć cechują się mniejszą wydajnością produkowanego ciągu liczb pseudolosowych i większą złożonością obliczeniową jak i pamięciową niż przedstawione uprzednio generatory strumieniowe.

8. GENERATOR RSA

Generator RSA korzysta ze znanego problemu, który jest podstawą algorytmu RSA (w [11] opis algorytmu RSA) tj. problemu faktoryzacji (mając odpowiednio dużą liczbę złożoną nie jesteśmy w stanie w czasie wielomianowo zależnym od liczby bitów rozłożyć ją na czynniki pierwsze, jeśli składa się z iloczynu dwóch dużych liczb pierwszych). Niestety jego główną wadą jest fakt podnoszenia liczby reprezentującej ziarno (ang. seed) do potęgi *e mod n*. To właśnie one niekorzystnie wpływają na efektywność obliczeniową generatora.

W [21] można znaleźć również ulepszoną wersję algorytmu: algorytm *Schnorra-Micaliego*, która jest efektywniejszy, gdyż tylko $floor\left[n\left(1-\frac{2}{\epsilon}\right)\right]$ bitów jest generowanych przez potęgowanie.

Co więcej, w każdym potęgowaniu (mnożeniu modulo) biorą udział mniejsze liczby niż poprzednio, więc praktycznie nie jest potrzebna redukcja modularna, tylko samo mnożenie. Inną zaletą ulepszonej wersji algorytmu jest fakt, że łatwo za jej pomocą równolegle produkować wiele pseudolosowych sekwencji i następnie je łączyć, co przyśpiesza działanie generatora.

Generatory bazujące na RSA i ich modyfikacje wydają się być zbyt wymagające dla systemów małej mocy, a wspomniana powyżej modyfikacja (algorytm *Schnorra-Micaliego*), która daje możliwość zrównolegnienia, na dzień dzisiejszy nie jest możliwa do implementacji w systemach RBSPS opartych na prostym mikrokontrolerze.

9. PODSUMOWANIE

W pierwszej części tego artykułu przedstawione zostały różne możliwości generacji liczb „prawdziwie" losowych zarówno przez rzeczywiste generatory bazujących na entropii źródła, pochodzącej z zjawisk fizycznych typu szum termiczny diody, rozpad radioaktywny, czy generatorów programowych. Kiedy to stworzone w tym celu oprogramowanie jest odpowiedzialne za gromadzenie entropii na podstawie różnych prawie całkowicie losowych zjawisk (różnych odstępów czasu między kolejnymi przerwaniami pochodzącymi od fizycznych urządzeń, czasu spoczynku klawiatury itd.), jakie są „odnotowywane" przez system operacyjny w dzisiejszych systemach komputerowych (urządzenie */dev/random* w Linuksie). Generatory tego typu są na pierwszy rzut oka dobrym rozwiązaniem, ale niestety mają poważne wady uniemożliwiające ich zastosowanie w RBSPS.

Dla przykładu, rozwiązanie programowe na styl jądra Linuksa odpada ze względu na prostotę bezprzewodowych węzłów tego systemu. Natomiast zastosowanie generatora fizycznego wiązałoby się koniecznością zapewnienia mu zasilania, z czym w RBSPS bywa duży problem, jak również skutkowałoby dodatkową komplikacją układu, a przez to większą potencjalną awaryjnością i dodatkowym kosztem całkowitego rozwiązania. W tych warunkach najkorzystniejszym rozwiązaniem wydaje się być wybór generatora liczb pseudolosowych, charakteryzującego

się niską złożonością obliczeniową, wysoką wydajnością, niskimi wymaganiami na pamięć operacyjną i dużym bezpieczeństwem kryptograficznym.

Przy tak ostro sprecyzowanych wymaganiach najlepszym algorytmem produkcji liczb pseudolosowych wydaje się być generator oparty na szyfrach strumieniowych, bazujący na algorytmie SNOW 2.0 lub ewentualnie *alternating step*. Dla przypadku postawienia na pierwszym miejscu potrzeby bezwzględnego bezpieczeństwa kryptograficznego, należałoby zastosować generator bazujący na szyfrach blokowych, pracujący w oparciu o algorytm AES czy 3DES, lub zmodyfikowany generator Blum-a, Blum-a i Shub-a.

Jak zostało wspomniane w tym artykule, bardzo ważną kwestią jest pełna losowość zarodka. W warunkach RBSPS dla przypadku najprostszego rozwiązania można skorzystać z aktualnej wartości czasu systemowego węzła zaszyfrowanej za pomocą funkcji skrótu np. SHA-1.

W systemach bardziej rozbudowanych, gdzie do bezpieczeństwa przyjmujemy wysoką wagę, zaleca się wykorzystanie entropii pochodzącej z niestałości częstotliwości własnej oscylatora węzła RBSPS.

LITERATURA:

[1] **W. Nawrocki:** *Rozproszone systemy pomiarowe,* WKŁ, Warszawa 2006

[2] **I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci:** *A Survey on Sensor Networks,* IEEE Communication Magazine, 2002

[3] **B. Holt, L. Doherty, E. Brewer:** *Flexible Power Scheduling for Sensors Networks*, Information Processing in Sensor Networks (IPSN), Berkeley, CA, 2004

[4] **I. Dunajewski, Z. Kotulski:** *Optimal Wireless Sensors Location For Widespread Structures Monitoring.* 36th Solid Mechanics Conferen-ce, Gdańsk 2008

[5] **E. Micha**: *Tendencje rozwojowe w obszarze systemów pomiarowo-sterujących,* Systemy Pomiarowe w Badaniach Naukowych i w Przemyśle, Łagowo, 2006

[6] **R. Zieliński:** *Generatory liczb losowych*, WNT, Warszawa 1972

[7] **R. Wieczorkowski, R. Zieliński:** *Komputerowe generatory liczb losowych*, WNT 1997

[8] **B. Schneier:** *Kryptografta dla praktyków,* WNT, 1995

[9] **P. Hoffman:** *Kryptografia a generator liczb losowych w systemie Linux*,

[10] **Dokumentacja jądra Linuksa:** http://www.kernel.org/

[11] **A. Menezes, P. van Oorschot, S. Yanstone:** *Handbook oj Applied Cryptography,* WNP, 2005

[12] **D. Knuth:** *Sztuka programowania,* WNT, 2002

[13] **P. L'Ecuyer:** *Combined Multiple Recursive Random Number Generators*, Operations research 44 (5), 1996

[14] **A. A. Kanso:** *Clock-Controlled Alternating Step Generator*, Cryptology ePrint Ar-chive (IACR), 2002 ()

[15] **D. Coppersmith, H. Krawczyk, Y. Mansour :** *The Shrinking Generator*, Crypto'93, Advances in Cryptology 1981 - 1997, Springer, 1997

[16] **L. Blum, M. Blum, M. Shub:** *Comparison of two pseudorandom generators*, Crypto'82, Advances in Cryptology 1981 - 1997, Sprmger, 1997

[17] **N. Dedić, L. Reyzin, S. Vadham:** *An Improved Pseudorandom Generator Based on Hardness of Factoring*, Cryptology ePrint Archive (IACR), 2002 ()

[18] **SNOW 2.0:** http://www.it.lth.se/cryptology/snow/

[19] **J. Viega, M. Messier:** *Secure Programming Cookbook for C and C++,* O'Reilly 2003

[20] **HMAC :** http://tools.ietf.org/html/frc2104

[21] **S. Micali, C.P. Schnorr:** *Efficient, perfect random number generators*, *Crytpo'88, Adyances in Cryptology 1981 - 1997,* Springer, 1997

[22] **W. Alexi, B. Chor, O. Goldreich, and C. Schnorr:** *RSA and Rabin functions: Certain parts are as hard as the whole.* SIAM Journal on Computing, April 1988.

[23] **J. H˚astad, A. W. Schrift, and A. Shamir:** *The discrete logarithm modulo a composite hides O*(*n*) *bits.* Journal of Computer and System Sciences, 1993.
[24] **Oded Goldreich and Vered Rosen:** *On the security of modular exponentiation with application to the construction of pseudorandom generators.* Technical Report 2000/064, Cryptology e-print archive, http://eprint.iacr.org, 2000.
[25] **P. Leglise, F. X. Standaert, G. Rouvroy, J. J. Quisquater :** *EFFICIENT IMPLEMENTATION OF RECENT STREAM CIPHERS ON RECONFIGURABLE HARDWARE DEVICES*
[26] **Jovan Dj. Golic:** *Cryptanalysis of Alleged A5 Stream Cipher* EUROCRYPT 1997
[27] **Biryukov, Alex; Adi Shamir; David Wagner:** *Real Time Cryptanalysis of A5/1 on a PC*, Fast Software Encryption—FSE 2000
[28] **Ross Anderson (1994-06-17):** *A5* (*Was: HACKING DIGITAL PHONES*). uk.telecom
[29] **Biham, Eli; Orr Dunkelman:** *Cryptanalysis of the A5/1 GSM Stream Cipher*. Indocrypt 2000
[30] **M. Kim, Y. Kim, J. Ryou, S. Jun:** *Efficient Implementation of the Keyed-Hash Message Authentication Code Based on SHA-1 Algorithm for Mobile Trusted*

Paweł Czernik

CRYPTOGRAPHICALLY SECURE PSEUDORANDOM NUMBER GENERATORS IN LOW POWER DISTRIBUTED MEASUREMENT AND CONTROL SYSTEMS

Summary

This paper focuses on the study reviewed evaluation and random number generators to determine whether they fit demands and characteristics Low Power Distributed Measurement and Control Systems (*LPDMCS*)*, becoming more widely used in avionics, among others on boards Unmanned Aerial Vehicles* (*UAV*)*. In this work are considered basic categories of random number generation algorithms: hardware random number generators* (*TRNG*) *and software pseudo-random number generators implemented software and hardware systems in FPGAs. The evaluation took into account the level of security algorithms, computational efficiency and the possibility of systemic implementation of rapid and efficient in terms of demand on the power of the latest programmable systems.*

Keywords: *random number generators, private key cryptography, public-key cryptography, digital signatures, distributed measurement and control systems, sensor networks, unmanned flying machines*