

Liczby losowe, a kryptografia

Paweł Dąbrowski

18 czerwca 2001

<http://ving.edunet.pl/projects>

1 Liczby losowe

1.1 Co to są generatory liczb losowych ?

Istota generatorów liczb losowych jest dość prosta. Poprzez losowy ciąg możemy, na przykład, uważać ciąg bitów generowanych poprzez rzucanie monetą: wyrzucenie reszki daje 1, orła zaś 0. Taki pojedynczy rzut monetą jest niezależny od pozostałych rzutów. Dodatkowo musimy założyć, że moneta jest dobrze wyważona co oznacza, że otrzymanie orła w jednym rzucie jest tak samo prawdopodobne jak wyrzucenie reszki.

Liczby losowe są niezbędnym elementem wielu algorytmów. Wykorzystuje się je między innymi w algorytmach symulacyjnych czy randomizowanych oraz w najbardziej nas interesującej - kryptografii. Sławny Donald E. Knuth pisze: "There are reports that many executives make their decisions by flipping a coin or by throwing darts, etc. It is also rumored that some college professors prepare their grades on such a basis. Sometimes it is important to make a completely "unbiased" decision; this ability is occasionally useful in computer algorithms, for example in situations where a fixed decision made each time would cause the algorithm to run more slowly".

W praktyce generatory liczb losowych, próbuje się tworzyć wykorzystując zdarzenia losowe w komputerze. Zdarzenia te pochodzą z jego urządzeń zewnętrznych, a w szczególności z klawiatury oraz myszy itp. Zdarzenia takie są gromadzone, w celu uzyskania coraz większej liczby losowych bitów. Niestety, tylko bardzo niewielką część otrzymanych od urządzeń danych można uznać za losową. Np. naciśnięcie klawisza o danym kodzie jest bardzo mało losowym zdarzeniem: zwykle są to kody ASCII małych liter, a korelacja między nimi jest bardzo duża. Tworząc generator liczb losowych buduje się go tak, żeby dodanie całkowicie deterministycznych ciągów, np. 100000 bajtów o tej samej wartości, nie zmniejszyło losowości zawartej w generatorze. Generator liczb losowych może utrzymywać specjalny licznik, który liczy, ile jest jeszcze losowych bitów w generatorze. Implementacja takiego licznika jest jednak bardzo trudna, dlatego stosuje się raczej liczenie "na oko" i trudno powiedzieć, czy z tych obliczeń

otrzymuje się sensowne wyniki. Dodatkową cechą generatora służącego do celów kryptograficznych powinno być, że nawet znając kod generatora i odczytując z niego kolejne tworzone przez niego liczby losowe, nawet jeżeli nie dostaje on żadnych kolejnych losowych bitów, nie powinno się dać przewidzieć, jaka jest następna liczba.

1.2 Liczby pseudolosowe

Dotychczas nie ma dostępnych na rynku tanich i efektywnych urządzeń generujących losowe ciągi. Decydującym powodem jest to, iż tanio można uzyskać tzw. *ciągi pseudolosowe* za pomocą specjalnych algorytmów. Ciągi pseudolosowe mają wszystkie własności ciągów losowych, jakie efektywnie można przetestować, ale jednocześnie mogą być wygenerowane w deterministyczny sposób. Jedynym losowym elementem takiego ciągu jest zarodek. Ciąg pseudolosowy jest tworzony tak, że i -ty element s_i jest wyznaczony przez pewien deterministyczny algorytm z zarodka x , indeksu i oraz wartości s_1, \dots, s_{j-1} . Zarodek jest zwykle stosunkowo krótkim ciągiem, więc można go wygenerować takimi metodami, jak poprzez mierzenie odstępów czasu pomiędzy uderzeniami palców w klawiaturę. Z drugiej strony, zarodek musi być na tyle długi, by wykluczyć możliwość odtworzenia zarodka poprzez przeglądanie wszystkich zarodków i ciągów przez nich generowanych.

Ciągi pseudolosowe są wykorzystywane, na przykład, w dziedzinie algorytmów zrandomizowanych, symulacjach procesów stochastycznych itp. Metody generowania ciągów pseudolosowych używane w tych dziedzinach nie mogą być stosowane do celów kryptograficznych, gdyż w kryptografii żądamy od nich dodatkowo aby nie były rozróżnialne od prawdziwych losowych ciągów przy użyciu żadnych praktycznych metod.

1.3 Własności ciągów pseudolosowych

1.3.1 Nierozróżnialność

Jeżeli ciąg bitów długości l wygenerujemy w sposób naprawdę losowy, to każdy ciąg jest generowany z tym samym prawdopodobieństwem $\frac{1}{2^l}$. Natomiast jeżeli ciągi długości l generujemy za pomocą zarodków długości k ($k < l$), to możemy otrzymać co najwyżej 2^k różnych ciągów długości l . Tym samym, przy założeniu, że różne zarodki dają różne ciągi pseudolosowe oraz każdy zarodek jest jednakowo prawdopodobny, niektóre ciągi otrzymujemy z prawdopodobieństwem 0, a niektóre z prawdopodobieństwem $\frac{1}{2^l}$. Wynika stąd, iż rozkłady prawdopodobieństwa ciągów losowych i pseudolosowych o długości l różnią się między sobą. Pytanie jednak, czy na podstawie kilku wygenerowanych ciągów l bitów jesteśmy w stanie stwierdzić, czy mamy do czynienia z generatorem liczb losowych czy pseudolosowych. W przypadku ciągu przedstawionych przy pomocy generatora pseudolosowego nie możemy zdobyć stuprocentowej pewności, gdyż każdy ciąg może być wygenerowany przy pomocy generatora liczb losowych. Dlatego też rozróżnianie generatorów jest procedurą, w której można mówić jedynie o

prawdopodobieństwie prawidłowej odpowiedzi.

Formalnie, niech g_1 i g_2 będą dwoma generatorami ciągów o długości l . Dla ciągu x długości l i $i \in \{0, 1\}$, niech $p_i(x)$ będzie prawdopodobieństwem wygenerowania x przez g_i . Niech A będzie algorytmem. Wtedy definiujemy

$$E_A(p_i) = \sum_{x \in \{0,1\}^l} p_i(x) * Pr(A(x) = 1)$$

Mówimy, że g_1 i g_2 są ϵ -rozdzielalne za pomocą algorytmu A jeśli $|E_A(p_1) - E_A(p_2)| > \epsilon$. Własność taką chcielibyśmy osiągnąć aby móc wykorzystać generator pseudolosowych ciągów g_1 w kryptografii, to jego nierozdzielalność od generatora losowego g_2 : dla każdego wystarczającego dużego ϵ .

1.3.2 Nieprzewidywalność

Jest to własność mówiąca, że bez znajomości zarodka x nie da się obliczyć w praktyce s_i , i -tego bitu ciągu pseudolosowego, z bitów s_1, \dots, s_{i-1} . W rzeczywistości pojęcia nierozdzielalności i nieprzewidywalności są ze sobą powiązane.

2 Kryptografia

Kryptografia to jedno z ważniejszych zastosowań liczb losowych. W Polsce istnieje jej prawna definicja:

„Kryptografia to dziedzina wiedzy zajmująca się zasadami, narzędziami i metodami przekształcania danych w celu ukrycia zawartych w nich informacji, zapobiegania możliwości tajnego ich modyfikowania lub eliminacji dostępu do nich osobom niepowołanym. 'Kryptografia' ogranicza się do przekształcania informacji za pomocą jednego lub więcej „tajnych parametrów” (np. szyfrów) i/lub związanego z tym zarządzania kluczami. Uwaga: 'tajny parametr': wartość stała albo klucz trzymany w tajemnicy przed osobami postronnymi albo znany wyłącznie pewnej grupie osób” (Dz.U. Nr 129, poz.598)

Jej zastosowanie:

- a) ochrona danych przed niepowołanym dostępem.
- b) uwierzytelnianie dokumentów.
- c) ochrona prywatności poczty elektronicznej.

Na swoje potrzeby kryptografia potrzebuje silnych generatorów liczb losowych,

2.1 Historia kryptografii

Początki kryptografii sięgają czasów starożytności. Około 4000 lat temu starożytni Egipcjanie potrafili szyfrować swoje hieroglify. Także starożytni Hebrajczycy szyfrowali niektóre słowa w swoich skryptach. Co ciekawe stopień skomplikowania tych metod był znacząco niższy niż stan wiedzy matematycznej w każdej

właściwie epoce. Stosowane wówczas metody były zazwyczaj dość prymitywne i pozwalały na złamanie szyfrów dostatecznie zdeterminowanemu przeciwnikowi. Sytuacja uległa zmianie już w pierwszej połowie dwudziestego wieku, Zbudowano wtedy wiele urządzeń mechanicznych i wykorzystywano je powszechnie podczas drugiej wojny światowej. Część tych systemów została skutecznie złamana np. Enigma. Dopiero prawdziwą rewolucję pod względem projektowania systemów szyfrujących przyniósł rozwój elektroniki, dający olbrzymie możliwości operacji obliczeniowych niskim kosztem. Rozwój kryptografii od swych początków był bardzo silnie związany z celami wojskowymi. W związku z tym wszelkie prace z dziedziny kryptografii miały charakter tajny i ich rezultaty nie były publikowane. Przełom nastąpił w latach siedemdziesiątych wraz z odkryciem asymetrycznych algorytmów szyfrujących.

2.2 Algorytmy asymetryczne

Algorytm asymetryczny to algorytm, w którym:

- klucze występują w parach: jeden klucz do szyfrowania i jeden do deszyfrowania.
- opublikowanie jednego z kluczy występującego w parze praktycznie nie zdradza drugiego z kluczy, nawet gdy można w tym celu wykonać dość złożone obliczenia.
- zwykle jeden z kluczy w parze jest powszechnie dostępny - może to być klucz szyfrujący lub deszyfrujący, w zależności od zamierzanych zastosowań (klucz publiczny). Drugi jest kluczem trzymanym w tajemnicy przez jego posiadacza (klucz prywatny).

2.2.1 Algorytm RSA

RSA jest jednym z najważniejszych dla praktyki algorytmów kryptograficznych. Nazwa RSA pochodzi od jego autorów: Rivesta, Shamira i Adlemana. Liczby losowe stosują się tu do generowania kluczy. Od momentu przedstawiania RSA dokonywane były liczne próby złamania szyfrów generowanych tą metodą. Tylko ograniczone sukcesy osiągnięto na tym polu. Ich konsekwencją było zwiększenie długości kluczy używanych przez RSA. Algorytm RSA przedstawia się następująco:

1. Losowo wybieramy dwie duże liczby pierwsze p, q .
2. Losowo wybieramy liczbę e tak, aby e i $(p - 1)(q - 1)$ były względnie pierwsze (wybieramy e i przy użyciu algorytmu Euklidesa obliczamy $NWD(e, (p - 1)(q - 1))$; jeśli liczba e została źle wybrana to powtarzamy próby, aż znajdziemy właściwe e).
3. Za pomocą algorytmu Euklidesa znajdujemy d takie, że:

$$e * d = 1 \text{ mod } (p - 1)(q - 1)$$

4. Obliczamy $n := p * q$ i kasujemy liczby p, q tak aby nie pozostał po nich żaden ślad.
5. $[e, n]$ jest wygenerowanym kluczem publicznym, $[d, n]$ jest kluczem prywatnym.

Szyfrowanie (szyfrowane mogą być liczby $m < n$) odbywa się następująco:

$$E_{[e,n]}(m) = m^e \bmod n$$

Deszyfrujemy podobnie:

$$D_{[d,n]}(m) = m^d \bmod n$$

Jak widać pojawia się w algorytmie problem generowania losowych dużych liczb pierwszych. Wydaje się to trudnym zadaniem. Na szczęście istnieje dużo liczb pierwszych: dla każdego x ilość liczb pierwszych w przedziale między $x/2$ a x wynosi około $x/\ln x$. Nawet, gdy przypadkowe trafienie na liczbę pierwszą jest dość prawdopodobne, stoimy wobec tego zagadnienia, jak odróżnić liczby pierwsze od liczb złożonych. Najprostszą metodą byłoby dokonać rozkładu na czynniki pierwsze każdej wylosowanej liczby. Jednak dla liczb rozważanego rozmiaru jest to praktycznie nie wykonalne ze względu na zakres bezpieczeństwa, jaki mają gwarantować szyfry RSA konstruowane za ich pomocą. Wyjściem jest stosowanie testów pierwszości liczb. Wszystkie znane testy są testami probabilistycznymi, tzn. dla zadanego (dowolnego) argumentu x mogą się pomylić z pewnym małym prawdopodobieństwem.

2.2.2 Algorytmem ElGamala

Algorytm ElGamala bazuje na trudności obliczenia tzw. dyskretnych logarytmów. Szyfrowanie za każdym razem wykorzystuje losowo wybraną liczbę, co powoduje, że ten sam tekst jawny może być w różny sposób zaszyfrowana. Ten algorytm określa jednoznacznie który z pary kluczy do czego ma służyć. Klucz publiczny służy do szyfrowania, natomiast prywatny do deszyfrowania danych. Nie istnieje tu zamienność kluczy tak jak było to w przypadku algorytmu RSA. Cechą charakterystyczną i niekiedy wadą jest to, że kryptogramy uzyskiwane w wyniku algorytmu ElGamala są dwukrotnie dłuższe od tekstu jawnego.

Na początek przyda się wyjaśnienie pojęcia dyskretnego logarytmu, a więc w zasadzie najważniejszego elementu całego algorytmu.

Przeprowadźmy następujące założenia:

Niech p będzie liczbą pierwszą, natomiast α generatorem Z_p (dla każdej liczby x większej od 0 i mniejszej od p istnieje taka liczba i dla której zachodzi zależność $x = \alpha^i \bmod p$). Trudność dyskretnego logarytmu polega na znalezieniu dla danego β większego od 0 i mniejszego od p liczby i takiej, że $a^i = \beta \bmod p$.

Musi być spełniony dodatkowy warunek, mianowicie liczba p musi być wystarczająco duża, w praktyce co najmniej 150 cyfrowa oraz to aby liczba $p - 1$ miała co najmniej jeden duży czynnik pierwszy.

Elementami algorytmu ElGamala są liczba pierwsza p przy czym liczba ta obarczona jest warunkami wymienionymi powyżej, a obliczenie dla niej logarytmu modulo p jest praktycznie niemożliwe, następnie generator α dla Z_p oraz liczby t i β , przy czym t musi być mniejsze od $p - 1$, natomiast $b = a^t$. Na klucz publiczny składają się liczby p , α oraz β . Kluczem prywatnym jest liczba t .

Szyfrowanie algorytmem ELGamala odbywa się w oparciu o powyższe dane. Powiedzmy, że chcemy zaszyfrować tekst jawny T . Długość tego tekstu musi być mniejsza od liczby p . Wybieramy losowo liczbę $l < p$, następnie obliczamy $s_1 = \alpha^l \bmod p$ oraz $s_2 = T * \beta^l \bmod p$. Otrzymana para (s_1, s_2) jest naszym kryptogramem tekstu jawnego T .

Deszyfrowanie odbywa się w następujący sposób: zwróćmy uwagę na zależność: $s_2 * (s_1^t)^{-1} = T * \beta^l * (\alpha^{lt})^{-1} = T * \alpha^{lt} * (\beta^{lt})^{-1} = T \bmod p$. Oczywiście osoba, która deszyfruje dane zna liczbę t i jest w związku z tym w stanie wyznaczyć liczę $s_2 * (s_1^t)^{-1} \bmod p$, a tym samym liczę T , czyli nasz tekst jawny.

Algorytm powyższy stosuje się nie tylko do szyfrowania ale w nieco zmienionej postaci do generowania podpisów cyfrowych.

2.3 Metoda One-time pad

Jest to metoda szyfrowania, w której używany jest losowo wybrany klucz $K = k_1, \dots, k_n$. Samo szyfrowanie odbywa się za pomocą operacji XOR. Załóżmy, że tekst jawny T jest ciągiem bitów: $T = t_1, \dots, t_n$. Wówczas kryptogramem C jest ciąg: $C = c_1, \dots, c_n$ taki, że: $c_i = t_i \text{ XOR } k_i$ dla $i = 1, \dots, n$. Natomiast deszyfrowanie polega na użyciu tych samych kluczy K . Tekst jawny otrzymuje się tak samo jak kryptogram: $t_i = c_i \text{ XOR } k_i$ dla $i = 1, \dots, n$. Tajemnicza funkcja XOR (eXclusive OR) definiuje się:

$$0 \text{ XOR } 0 = 1 \text{ XOR } 1 = 0 \quad 0 \text{ XOR } 1 = 1 \text{ XOR } 0 = 1$$

Jeżeli dla każdego szyfrowania będziemy używać innego, wygenerowanego niezależnie klucza to bez jego znajomości i bez względu na zastosowane moce obliczeniowe żadna informacja dotycząca tekstu jawnego nie może być wydedukowana z kryptogramu. Własność ta często jest nazywana bezpieczeństwem doskonałym. Oprócz losowego klucza, także kryptogram jest ciągiem losowym n bitów.

Jedną z dziedzin zastosowań one-time pad jest szyfrowanie stosunkowo krótkich, ale bardzo ważnych informacji, jak rozkazy wojskowe o strategicznym znaczeniu. Niestety przy stosowaniu powyższej metody napotykamy na szereg trudności. Klucz musi zostać uzgodniony przed każdą transmisją przez osoby komunikujące się oraz musi być naprawdę losowy, co z kolei technicznie nie jest łatwe.

Ideę metody one-time pad wykorzystuje się także w szyfrowaniu strumieniowym.

2.4 Szyfrowanie strumieniowe RC4

Algorytm szyfrowania RC4 jest przykładem szyfrowania strumieniowego. Był sekretem handlowym zanim ktoś umieścił kod źródłowy algorytmu w Usenecie, mówiąc, że to RC4. Algorytm ten jest bardzo szybki. Jego bezpieczeństwo jest nieznane, ale jego łamanie nie wydaje się również proste. Z powodu jego szybkości może mieć zastosowanie w specyficznych aplikacjach. Może również akceptować klucze o przypadkowej długości. RC4 jest najkrócej mówiąc pseudo generatorem liczb losowych i wynik generowania jest xorowany ze strumieniem danych. Z tego powodu jest bardzo ważne by ten sam klucz RC4 nie był nigdy używany do szyfrowania dwóch różnych strumieni danych. Co najważniejsze jest zaskakująco prosty do implementacji. Poniżej przykładowa jego implementacja w paru liniijkach Perla [4]:

```
#!/usr/bin/perl -0777 @k=unpack('C*',pack('H*',shift)); for(@t=@s=0..255)
{ $y= ($k[$_ % @k] + $s[$x=$_] + $y) % 256; $S{$_} = $y; $x = $y = 0;
for(unpack('C*',<>)) { $x++; $y = ($s[$x % 256] + $y) % 256; $S{$_} = $y; print
pack('C',$_ ^ ($s[$x] + $s[$y]) % 256)} sub S{@s[$x,$y] = @s[$y,$x]}
```

Algorytm przedstawia się następująco:

1. Załóżmy, że tekst jawny to ciąg bitów $A = a_1, \dots, a_n$, a w wyniku szyfrowania otrzymujemy ciąg bitów kryptogramu $C = c_1, \dots, c_n$. Klucz jest także ciągiem bitów $K = s_0, \dots, s_{m-1}$ o długości m .
2. Na początku inicjalizujemy parametry $i, j < 256$ oraz permutację π na zbiorze liczb $\{1, \dots, 256\}$. Parametry są nazwane wewnętrznymi kluczami:

$$i := 0, j := 0 \quad \pi(h) = h \text{ dla } h \leq 256$$

Następnie 256 razy wykonujemy pętle:

$$j := j + \pi(i) + s_i \text{ mod } 256$$

$$\text{swap}(\pi(i), \pi(j))$$

$$i := i + 1 \text{ mod } 256$$

3. Każda faza algorytmu generuje jeden bajt ciągu pseudolosowego i jednocześnie uaktualniając parametry i, j oraz permutację π . Wykonywane są następujące czynności:

$$i := i + 1 \text{ mod } 256$$

$$j := j + \pi(i) \text{ mod } 256$$

$$\text{swap}(\pi(i), \pi(j))$$

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

Liczba k jest kolejnym bajtem ciągu pseudolosowego generowanym w tej fazie.

4. Zgodnie z zasadą szyfrowania strumieniowego generowany jest i -ty bit kryptogramu $c_i := s_i XOR a_i$.

2.5 Szyfrowanie probabilistyczne

Szyfrowanie probabilistyczne uzyskuje się poprzez użycie niedeterministycznego algorytmu szyfrującego. Oznacza to, że tekst jawny może odpowiadać wielu kryptogramom przy zastosowaniu tego samego klucza. Przykładem takiego algorytmu jest SZYFROWANIE BLUM-GOLDWASSERA.

Algorytm ten oparty jest o własności generatora liczb pseudolosowych BBS, który bazuje w uproszczeniu problemu trudności ostatniego bitu kryptogramów RSA. Kolejna liczba pseudolosowa generatora równa jest: $x_{i+1} = x_i^2 \bmod n$, mając dane: $n = p * q$, p i q są względnie pierwsze i $p = q = 3 \bmod 4$.

Liczby p i q stanowią jednocześnie tajny klucz służący do deszyfrowania, szyfrowanie dokonywane jest za pomocą publicznego klucza n .

Szyfrowanie wykorzystuje ideę szyfrowania strumieniowego:

1. Wybieramy losowo liczbę s_0 , będący jednocześnie zarodkiem generatora BBS.

2. Dla l -bitowego tekstu jawnego generujemy s_1, \dots, s_{l+1} , że dla $i = 0, \dots, l$ zachodzi:

$$s_{i+1} = s_i^2 \bmod n$$

3. Dla tekstu jawnego a_1, \dots, a_l obliczamy c_1, \dots, c_l , gdzie dla $i \leq l$:

$$c_i = a_i \bmod (s_i \bmod 2)$$

4. Kryptogramem podanego tekstu jest $c_1, c_2, \dots, c_l, s_{l+1}$.

Deszyfrowanie:

1. Na podstawie s_{l+1} , p i q obliczamy s_1 .
2. Na podstawie s_1 wyliczamy pozostałe s_i dla $i \leq l$
3. Mając s_i wyliczamy kolejne bity tekstu jawnego: $a_i = c_i XOR z_i$.

2.6 Uwierzytelnianie

Uwierzytelnianie jest jednym z kluczowych zadań w zakresie zapewnienia bezpieczeństwa w systemach komputerowych. Chodzi o to, aby system komputerowy mógł stwierdzić czy osobą podającą się za określonego użytkownika jest nim faktycznie. Także użytkownik może żądać potwierdzenia że znalazł się (połączył) z właściwym systemem. Liczby losowe stosowane są tu w przeróżnych protokołach uwierzytelniania.

2.6.1 Protokół challenge and response

Jest to jeden z najprostszych protokołów uwierzytelniania. Wymaga wpięrcz ustalnia pewnej jednokierunkowej funkcji f oraz tajnego klucza k .

Załóżmy, iż mamy do czynienia z chęcią dokonania operacji finansowej przez Alicję w jakimś Banku. Alicja oraz Bank zna funkcję f oraz Bank zna prywatny klucz Alicji k . Przykładowa sesja uwierzytelniania wyglądała by następująco:

1. Alicja nawiązuje połączenie z bankiem podając swoje dane identyfikacyjne.
2. Bank generuje losowy ciąg r i przesyła go Alicji.
3. Alicja oblicza $f(k, r)$ i przesyła wynik do Banku.
4. Bank oblicza $f(k, r)$. Jeśli wynik zgodzi się z tym, który dostał od Alicji wówczas przyjmuje, że Alicja jest tą osobą, za którą się podaje.

Jak widać do czynienia mamy tu z liczbą losową r . Niepowtarzalność tej liczby zapewnia nam bezpieczeństwo pomimo potencjalnej możliwości podsłuchem transmiji. Tym samym ważna jest tu odpowiednia duże generowane liczby r .

3 Podsumowanie

Liczby losowe na stałe zadomowiły się w kryptografii. Można by nawet powiedzieć, iż są jej solidnym fundamentem. W algorytmie RSA losowe liczby wykorzystuje się w generowaniu kluczy. Natomiast w algorytmie ElGamala generator liczb losowych wykorzystywany jest w każdej sesji szyfrowania. Podobnie w metodzie One-time pad. RC4 samo w sobie jest generatorem pseudolosowym, a w protokole challenge and response wygenerowana liczba losowa jest wykorzystywana do dalszej transformacji. Tym samym jeżeli generator ciągów losowych nie jest wystarczająco silny i istnieje możliwość odgadnięcia lub drastycznego zawężenia zbioru liczb generowanych kryptograficzna siła powyższych algorytmów maleje. Przykład: przypuśćmy, że na pewnej maszynie wieloużytkownikowej potrafimy przewidzieć jaka liczba zostanie wygenerowana w procesie generowania kluczy RSA. Znając tą liczbę atakujący może poznać klucz prywatny ofiary.

Bibliografia

- [1] Mirosław Kutylowski, Willy-B. Strohmann "Kryptografia - teoria i praktyka zabezpieczeń systemów komputerowych"
- [2] http://www.micronet.com.pl/~biv/doc/mirror/lew.tu.koszalin.pl/_abernat/polish/kryptgr.htm- Wykłady z Kryptografii
- [3] http://rainbow.mimuw.edu.pl/S0/LabLinux/WEJSCIE-WYJSCIE/PODTEMAT_1/crypto.html- algorytmy kryptograficzne w Linuxie
- [4] <http://www.cipherspace.org/~adam/rsa/rc4.html> - o RC4, źródła
- [5] <http://eagles.asd.wednet.edu/LibWebPg/Cryptography/default.htm> - historia kryptografii