JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 4 (2017) 82-86
Wysłane: 2017-01-07
Przyjęte: 2017-01-18

Porównanie szybkości działania wybranych funkcji skrótu i algorytmów szyfrowania

Dawid Górniak*, Piotr Kopniak

Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Dane to często najcenniejsza rzecz, jaką gromadzimy na naszych komputerach. Bez odpowiedniego zabezpieczenia danych algorytmami szyfrującymi, cenne dla nas informacje w przypadku dostania się w niepowołane ręce mogą zostać bezproblemowo wykorzystane. Artykuł przedstawia wybrane metody szyfrujące i funkcje skrótu dostępne w bibliotece Boucy Castle dla środowiska Java. Przedstawiona analiza dotyczy pomiaru prędkości generowania i weryfikacji 240 bitowej sygnatury dla algorytmów szyfrujących, natomiast dla funkcji skrótu analiza dotyczy szybkości działania funkcji. Z pośród badanych algorytmów szyfrowania i funkcji skrótu najszybsze okazały się AES i SHA1.

Słowa kluczowe: algorytmy szyfrujące; funkcje skrótu; Bouncy Castle

*Autor do korespondencji.

Adres/adresy e-mail: dawid.gorniak@pollub.edu.pl

Comparing the speed of the selected hash and encryption algorithms

Dawid Górniak *, Piotr Kopniak

Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The data is often the most valuable thing that we collect on our computers. Without proper data security with encryption our valuable information may be illegally used by an unauthorised person. The article presents selected encryption methods and hash functions available in Boucy Castle library for Java programming language. The presented analysis applies to measurement of the speed of signature generation and verification. The signatures are for 240 bit encryption algorithms. In case of a hash function, the analysis refers to the speed of such functions. The fastest encryption algorithm and hash function from the research group were AES and SHA1.

Keywords: encryption algorithms; hash functions; Bouncy Castle

*Corresponding author.

E-mail address/addresses: dawid.gorniak@pollub.edu.pl

1. Wstęp

Postęp techniczny w dużej mierze wpływa na bezpieczeństwo prywatnych danych. Niezależnie od tego, do jakich celów wykorzystujemy komputer, na dysku twardym zazwyczaj znajduje się mnóstwo ważnych danych, które mogą dostać się w niepowołane ręce. Bez odpowiedniego szyfrowania, przechwycone dane przez osoby postronne mogą być bezproblemowo odczytane i wykorzystane. W związku z tym należy dbać o bezpieczeństwo informacji. Prywatne dane, chronione przez algorytmy szyfrujące, są w dużej mierze bezpieczne. Szyfrowanie to proces przekształcania informacji w taki sposób, aby osoba nieupoważniona nie mogła jej odczytać – zaufana osoba może odszyfrować daną wiadomość i przeczytać ją w pierwotnej formie. Istnieje wiele popularnych metod szyfrowania, deszyfrowania, jednak kluczem do bezpieczeństwa wcale nie jest odpowiednio zastrzeżony algorytm. Najważniejszą rzeczą jest utrzymanie w tajemnicy klucza szyfrowania[1].

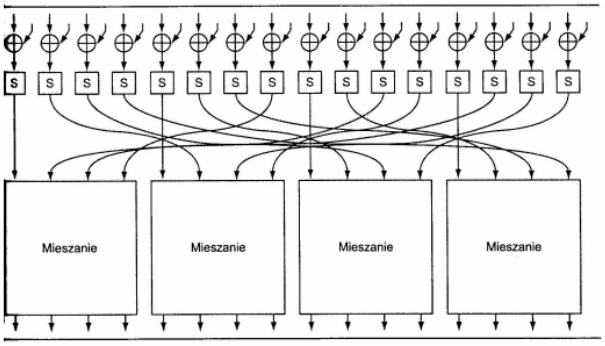
Celem artykułu jest analiza wybranych algorytmów szyfrujących i funkcji skrótu pod względem szybkości. Do badań wykorzystano implementacje dostępne w bibliotece Bouncy Castle dla środowiska programistycznego Java. Dzięki odpowiedniemu skryptowi napisanemu w Javie przeprowadzono testy pomiaru szybkości dla komunikatu o długości 240 bitów. Dla algorytmów szyfrujących przeprowadzono pomiar generowania i weryfikacji sygnatury. Natomiast dla funkcji skrótu zbadano szybkość działania funkcji haszującej. Uzyskane wyniki zostały przedstawione za pomocą wykresów, z których wysnuto wnioski.

2. Algorytmy szyfrujące

Do przeprowadzonych badań użyto wybranych algorytmów używanych do elektronicznego podpisu dokumentów oraz zaprezentowane zostaną schematy ich działania.

2.1. AES

Szyfr blokowy AES został przyjęty w 1997 roku przez Narodowy Instytut Samorządu Terytorialnego. Algorytm ten jest standardem USA używającym kluczy o rozmiarach 128, 192, 256 bitów i operuje na 128 bitowych blokach danych [2]. Na poniższym schemacie (Rys. 1) przedstawiona jest pojedyncza tura tego algorytmu.



Rys. 1. Struktura pojedynczej tury AES [2].

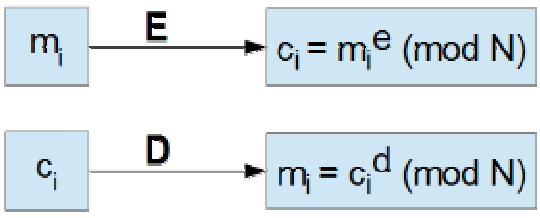
Na wejściu otrzymywany jest 16 bitowy tekst jawny, który poddawany jest metodzie XOR z 16 bitowym kluczem tury. Każdy z 16 bajtów wyjściowych składa się na tablicę S-Boksów. Istotą działania S-boksa jest podstawianie określonego bitu innym bitem zależnym od następnego i poprzedniego bitu. Następnie bity są grupowane dzięki funkcji mieszającej. Przebieg pojedynczej tury polega na przeprowadzeniu operacji XOR na kilku bitach wejściowych. Liczba tur szyfrowania zależna jest od rozmiaru klucza.

2.2. RSA

Nazwa algorytmu RSA wywodzi się od nazwisk trzech projektantów: Ron'a Rivest'a, Adi Shamir'a oraz Leonard'a Adleman'a. Asymetryczny blokowy algorytm wykorzystuje dwa powiązane klucze: publiczny i prywatny o długości od 1000 do 4000 bitów. Do zastosowania systemu RSA wymagane jest wygenerowanie pary kluczy, z którymi wiążą się poniższe zadania[3]:

- wybór dwóch dużych losowych liczb pierwszych p i q
- obliczenie wartości n= pq
- obliczenie dla n wartości funkcji Eulera: φ(n)=(p-1)(q-1)
- wybranie liczby e *(1<e< φ(n)) względnie pierwszej z φ(n)
- wyszukanie liczby d: φ(n): d≡ e^{-1}(mod φ(n))

Głównym problemem jest wyszukanie liczb p i q. Liczby te należą do zbioru liczb pierwszych. W dzisiejszych czasach nie ma efektownej metody do poszukiwania dużych liczb pierwszych, polegającej na wybraniu z dużego zbioru liczby p i q w celu uniemożliwienia ich odkrycia podczas kolejnych prób wyliczanych z wzoru n=pq. Jednak do tego celu stosuje się funkcję, która polega na wylosowaniu żądanej wielkości liczby nieparzystej i sprawdzeniu czy liczba zalicza się do zbioru liczb pierwszych.

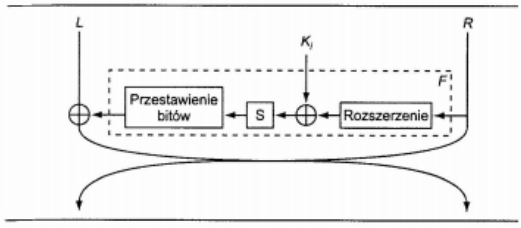


Rys. 2. Schemat szyfrowania i deszyfrowania w algorytmie RSA [3].

Klucz publiczny używany jest do szyfrowania wiadomości, która dzielona jest na fragmenty zamieniane na liczbę bitów. Takie liczby podnoszone są modulo n do potęgi e, co widoczne jest na powyższym rysunku (Rys. 2). Klucz prywatny wykorzystywany jest do deszyfrowania wiadomości składającej się z liczb mniejszych niż n. Liczby te mnożone są przez modulo n i potęgowane przez liczbę d (Rys. 2).

2.3. 3DES

Algorytm 3DES jest rozszerzeniem algorytmu DES. Zbudowany jest z trzech kolejnych rund DES. 3DES posiada dłuższy klucz, ale dziedziczy po DES właściwości dopełnienia oraz słabe klucze. 64 bitowy rozmiar bloku jest ograniczeniem tego szyfru, ponieważ wiąże się to z ograniczeniem szyfrowanych danych pojedynczym kluczem [4].



Rys. 3. Struktura pojedynczej tury algorytmu DES [4].

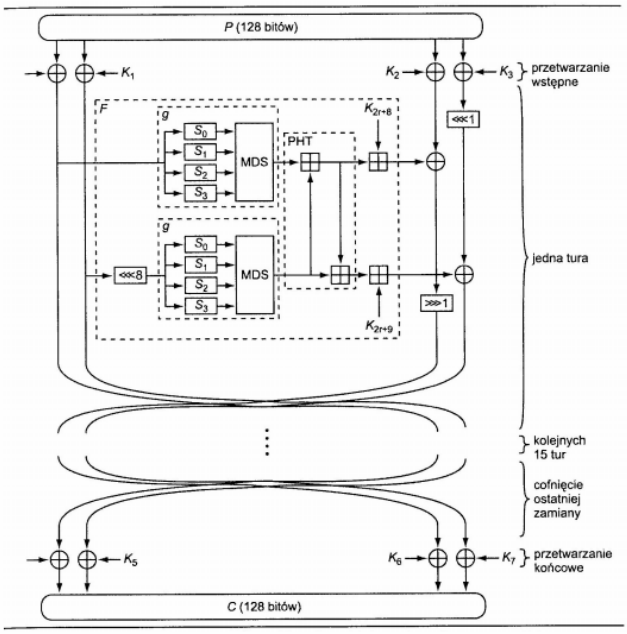
DES wykorzystuje 64 bitowy jawny tekst, który jest dzielony na dwie 32 bitowe części L i R. Szesnaście ponumerowanych od 1 do 6 tur, które tworzą nową parę zgodnie z Ki kluczem tury. Funkcja tury F wykonująca większość pracy oznaczona jest na rysunku za pomocą prostokąta narysowanego przerywaną linią. Początkowo za pomocą funkcji rozszerzającej zwiększana jest liczba bitów do 48 dla wartości R. Następnie przekształcany jest za pomocą metody XOR otrzymany ciąg z Ki 48 bitowym kluczem tury. Otrzymany wynik jest wejściem do S-boksu, składającego się z ośmiu tablic, z czego każda 6 bitowa tablica przekształcana jest na 4 bity. Dzięki temu ponownie wraca do 32 bitowej wielkości, która jest przekształcana przez funkcje mieszającą, a następnie poddawana z wartością L operacji XOR. Ostatecznie zmieniana jest wartość L i R.

2.4. Serpent

Algorytm Serpent pozwala jednocześnie szyfrować albo deszyfrować dane. Zalicza się do algorytmów bezpiecznych, skomplikowanych i wymagających dużej ilości zasobów sprzętowych przez jego blok deszyfrujący i szyfrujący. Działanie algorytmu serpent opiera się na 256 bitowym kluczu, który jest dzielony na osiem bloków 32 bitowych. Bloki te otrzymywane są rekurencyjnie z wartości k_i. Poprzez S-Boxy transformowane są wartości, które w dalszej części są scalane w wektory K_i stanowiące klucze dla pojedynczych rund szyfrowania i deszyfrowania. W skład rundy wchodzi operacja XOR danych z kluczem, przejście przez S-Box i transformacja liniowa. Proces deszyfrowania opiera się na bloku z kluczem K_i. , który podlega operacji XOR. Pozostałe 32 rundy podają dane na wejściu S-Box-1 następnie stosują klucz K_i oraz odwrotną transformacje linearną [5].

2.5. Twofish

Algorytm Twofish jest blokowym symetrycznym algorytmem szyfrującym, który operuje na 128 bitowych blokach danych i korzysta z kluczy z zakresu od 128 do 256 bitów. Poniższy schemat (Rys. 4) przedstawia działanie tego algorytmu.

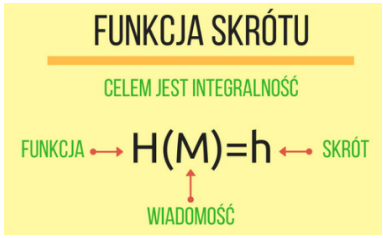


Rys. 4. Struktura Twofish [6].

Szyfrując, algorytm Twofish na wejściu otrzymuje 128 bitową wiadomość, którą dzieli na cztery 32 bitowe wartości. Funkcja F jest turą na którą składa się podwójne wywołanie funkcji g, funkcji PHT oraz dodania klucza. Rezultat funkcji F i pozostałe części danych poddawane są operacji XOR. Prostokąty z symbolami „<<<i", „>>>i" oznaczają przesunięcia 32 bitowych wartości o zadaną liczbę bitów. Funkcja g zawiera cztery zależne od klucza S-boxy i funkcję liniową podobną do mieszającej funkcji AES. Na podstawie klucza wyliczana jest specjalnym algorytmem tablica S-boksów. Za mieszanie dwóch wyników funkcji g z wykorzystaniem 32 bitowej operacji odpowiada funkcja PHT. Algorytm Twofish posiada dodatkowe wstępne i końcowe przetwarzanie, które polega na dołączeniu informacji z klucza [6].

3. Funkcje haszujące

Podstawowym elementem systemów uwierzytelniania oraz sprawdzania integralności wiadomości są funkcje skrótu (ang. hash functions). Funkcja skrótu tworzy z wiadomości skrót o stałej długości nie zależnej od długości wejściowej wiadomości (Rys. 5). Występuje tutaj sytuacja zwana kolizją, która oznacza że istnieją wiadomości zwracające identyczne wartości funkcji skrótu. Takie sytuacje są wysoce niepożądane w zastosowaniach kryptograficznych [7].



Rys. 5. Funkcja skrótu [7].

3.1. MD5

Algorytm haszujący MD5 został opublikowany przez R. Rivesta bez podania argumentów, uzasadniających matematycznie, że może on pełnić funkcję dobrej jednokierunkowej funkcji skrótu. Wejściem algorytmu jest komunikat M o praktycznie dowolnej długości, a wyjściem 128 bitowy skrót, czyli wyciąg tego komunikatu. Przetwarzanie komunikatu na skrót odbywa się w 4 krokach, w których dane są reprezentowane za pomocą bloków 512 bitowych. Przetwarzanie komunikatu [8]:

1) gdy warunek K ≡ 448 (mod 512) jest spełniony, wiadomość dopełniana jest dodatkowymi bitami. K oznacza pierwotną długość komunikatu. W takiej sytuacji dopełnienie ma długość od 1 do 512 bitów.
2) do wiadomości dodawana jest liczba K, reprezentująca pierwotną długość komunikatu M. W przypadku gdy początkowa wartość komunikatu przekracza 264, wtedy poddana operacji modulo 264 zostaje liczba, która określa pierwotną długość komunikatu. W rezultacie tej czynności otrzyma się L 512-bitowych bloków Y_0, Y_1, . . . , Y_{L-1}, które można podzielić na 16 podbloki 32-bitowe, ponieważ 512=16 · 32.
3) operacje, realizowane przez algorytm MD5, polegają na zmianie wartości zmiennych D,C,B i A, zapamiętanych w 4 rejestrach 32 bitowych. Zmienne otrzymują wartości początkowe zapisane w systemie szesnastkowym: D = 76543210, C = fedcba98, B = 89abcdef, A = 01234567.
4) w tym kroku używa się czterech funkcji pomocniczych, których argumentami są trzy 32 bitowe słowa, a wyjściem jedno słowo 32 bitowe. Są to następujące funkcje:
   F(X, Y, Z) = (X and Y) or ((not X) and Z), (1)
   G(X, Y, Z) = (X and Z) or (Y and (not Z)), (2)
   H(X, Y, Z) = X xor Y xor Z, (3)
   I(X, Y, Z) = Y xor (X or (not Z)). (4)
   W tym kroku stosuje się również 64 elementową tablicę T[1..64], której elementami są liczby całkowite zapisane w formie szesnastkowej. Niech M[0, 1, . . . , N −1] oznacza 32 bitowe słowa uzupełnionego komunikatu, składającego się z L 512 bitowych bloków. Wobec tego N = 16 ·L, czyli każdy blok Y_q składa się z 16 słów M[j]. Proces przetwarzania w tym kroku obejmuje każdy blok 16 słowowy Y_q. Wyjściem algorytmu jest 128 bitowy skrót komunikatu M.

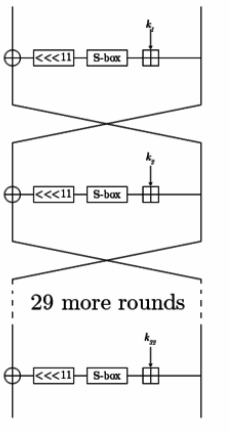
3.2. GOST

Algorytm GOST to szyfr blokowy opracowany w Związku Radzieckim. Operuje na 64 bitowych blokach,

a do szyfrowania wykorzystuje 256 bitowy klucz. Jego funkcja cykliczna składa się z dwóch kroków:

- dodawany jest 32 bitowy pod klucz modulo 232.
- wynik jest przetwarzany przez warstwę S-boxy i rotowany o 11 bitów w lewo.

Na poniższym schemacie (Rys. 6) jedna linia reprezentuje 32 bity [9].

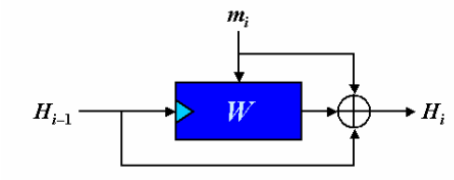


Rys. 6. Diagram GOST [9].

Podklucze są z góry wybierane w określonej kolejności. Plan kluczy jest prosty: dzieli klucz 256 bitowy na osiem 32 bitowych pod kluczy, a każdy pod klucz używany jest czterokrotnie w algorytmie. Pierwsze 24 rund używane jest w naturalnym porządku, a ostatnie osiem rund w odwrotnej kolejności.

3.3. Whirlpool

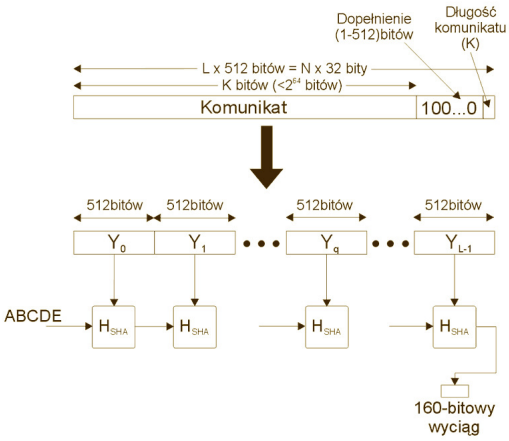
Funkcja skrótu Whirlpool przyjmuje wiadomość o dowolnej długości mniejszej niż 2^{256} bity i zwraca 512 bitowy skrót. Wiadomość jest dzielona na 512 bitowe bloki. Jeżeli wiadomość nie zostanie podzielona równomiernie, wtedy do ostatniego bloku jest dołączana jedynka i liczba zer, a na końcu dołączana jest długość oryginalnej wiadomości jako 256 bitowa liczba, tak aby ostatni blok uzyskał 512 bitów. Bloki te są kompresowane schematem Miyaguchi Preneel, a następnie poddawane są 10 rundom szyfru Whirlpool „W", gdzie W wykorzystuje wynik poprzedniej operacji i obecnego bloku jako klucza. Wynikiem jest operacja XOR wyjścia W i dwóch wejść [9].



Rys. 7. Struktura Whirlpool. H_{i-1} jest wyjściem poprzedniego bloku, m_i jest wejściem bloku [9].

3.4. SHA

Secure Hash Algorithm to funkcja skrótu zaprojektowana przez National Security Agency i publikowana przez National Institute of Standards and Technology. Algorytm z informacji o rozmiarze 264 bitów generuje 160 bitowy skrót. SHA wykonuje podstawowe operacje bitowe na 32 bitowych zmiennych podczas działania.



Rys. 8. Schemat działania algorytmu SHA [9].

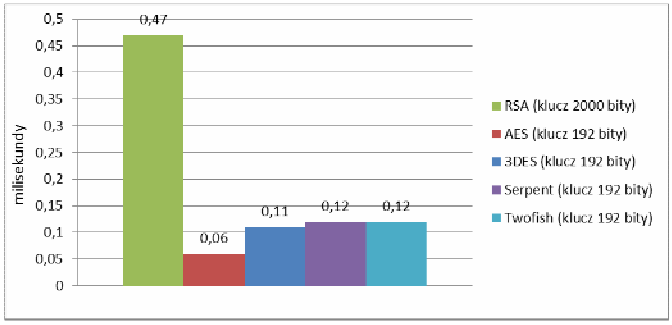
Działanie algorytmu SHA [9]:

- komunikat jest dopełniany w taki sposób, aby jego długość w bitach wynosiła do 448 mod 512.
- do długości wiadomości dodawana jest dodatnia 64 bitowa liczba.
- rezultatem tych kroków jest wiadomość o długości równej wielokrotności 512 bitów, którą można przedstawić jako sekwencję 512 bitowych bloków.
- stosowany jest bufor 160 bitowy, który można przedstawić jako pięć 32 bitowe rejestry. Bufor służy do przetrzymywania pośrednich i końcowych wartości funkcji haszującej.
- głównym etapem algorytmu SHA jest przetwarzanie komunikatu w 512 bitowych blokach.
- wynikiem L-tego przetworzenia 512 bitowego bloku jest 160 bitowy wyciąg.

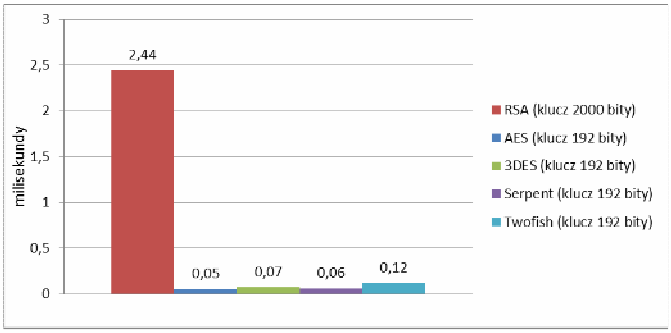
4. Wyniki

4.1. Algorytmy kryptograficzne

Wyniki szybkości generowania i weryfikacji sygnatury o długości 240 bitów dla algorytmów kryptograficznych: RSA, AES, 3DES, Serpent, Twofish są przedstawione na rysunkach 9,10.



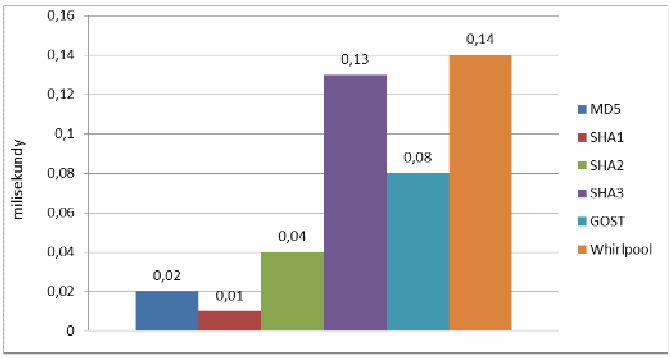
Rys. 9. Wynik szybkości generowania sygnatury.



Rys. 10. Wynik szybkości weryfikacji sygnatury.

4.2. Funkcje haszujące

Wyniki pomiarów szybkości działania funkcji skrótu dla MD5, SHA1, SHA2, SHA3, GOST, Whirlpool dla komunikatu 240 bitowego zostały przedstawione na poniższym rysunku 11.



Rys. 11. Wynik szybkości działania funkcji haszującej.

5. Wnioski

Analizując powyższe wykresy algorytmów szyfrujących można wywnioskować, że wygenerowanie 240 bitowej sygnatury najszybciej zachodzi w algorytmie AES generowanie trwało 0,06 milisekund. Natomiast RSA okazał się prawie ośmiokrotnie wolniejszy od najszybszego algorytmu, osiągając wynik 0,47 milisekund. Pozostałe algorytmy wypadły nie znacznie gorzej osiągając czas od 0,11 do 0,12 milisekund. W przypadku weryfikacji szybkość sprawdzania sygnatury dla algorytmu AES jest około czterdzieści osiem razy szybsza od metody RSA z czasem 2,44 milisekund. Nieco gorsze od AES z wynikiem 0,05 milisekundy okazały się pozostałe algorytmy, gdzie ich czasy to: 3DES 0,07 milisekund, Serpent 0,06 oraz Twofish 0,12 milisekund. Z pośród analizowanych algorytmów szyfrujących algorytm AES okazał się najszybszym algorytmem w generowaniu i weryfikacji 240 bitowej sygnatury. Pamiętając o tym, że w dzisiejszych czasach dużą rolę odrywa bezpieczeństwo danych najbezpieczniejszym algorytmem z pośród badanych szyfrów jest szyfr RSA z kluczem 2000 bitowym, gdyż jest on szyfrem asymetrycznym wymagającym dwa klucze: klucza publicznego do wygenerowania sygnatury oraz klucza prywatnego do dekodowania danych.

Odnosząc się do analizy wykresu funkcji haszujących algorytm SHA1 okazał się najszybszą funkcją haszującą z czasem 0,01 milisekund, o 0,01 milisekund wolniejszym algorytmem jest algorytm MD5, o 0,02 milisekund wolniejszym od MD5 okazał się SHA2. Natomiast GOST uzyskał wynik 0,08 milisekund kwalifikując się na 4 miejsce co do prędkości generowania skrótu dla 240 bitowej wiadomości. Pozostałe algorytmy SHA3 oraz Whirlpool wypadły najgorzej osiągając kolejno czas 0,13 i 0,14 milisekund. W kwestii bezpieczeństwa funkcji haszujących, najodporniejszym na różnego rodzaju fałszerstwa jest funkcja mieszająca SHA1, gdyż jest ona tak zaprojektowana, aby obliczeniowo uniemożliwić uzyskanie wiadomości z danego skrótu wiadomości lub wystąpienia kolizji. Dzięki temu jakakolwiek zmiana dokonana w wiadomości utworzy inny skrót wiadomości.

Literatura

[1] W. Stallings, Kryptografia i bezpieczeństwo sieci komputerowych, Helion, 2011.

[2] Kryptografia i bezpieczeństwo sieci komputerowych, Helion, 2012.

[3] Marcin Karbowski, Podstawy kryptografii RSA. Wydanie III, Helion, 2014.

[4] A. Menezes, Kryptografia stosowana, WTN, 2005.

[5] Serpent, www.pl.wikipedia.org/wiki/Serpent_(kryptografia)

[6] Twofish , www.pl.wikipedia.org/wiki/Twofish.

[7] I. Damgard, A design principle for hash functions, Springer-Verlag, 1990.

[8] MD5, www.pl.wikipedia.org/wiki/MD5

[9] M. Jakobsson, Applied Cryptography and Network Security, Springer Verlag gmbh, 2014.