|  |  |
| --- | --- |
| **#**  implementacja funkcji skrótu sha-3  Projekt z przedmiotu Zaawansowane metody kryptografii i ochrony informacji [MKOI] | Temat projektu  Implementacja funkcji skrótu SHA-3 (wszystkie warianty) oraz porównanie jej wydajności z istniejącą implementacją  Wykonujący projekt  inż. Paweł Kamiński  inż. Jan Muczyński  PROWADZĄCY PROJEKT  mgr inż. Albert Sitek  Semestr: 17Z |

1. Wprowadzenie

Tematem projektu jest implementacja funkcji skrótu SHA-3. Jest to najnowsza funkcja w rodzinie standardów *Secure Hash Algorithm*. Została wyłoniona w wyniku konkursu przeprowadzonego przez NIST w 2015 roku [1]. Algorytmem, który został wybrany był algorytm *Keccak.*

Funkcje skrótu są powszechnie używane na świecie w wielu dziedzinach informatyki. Służą przekształceniu wiadomości cyfrowej na krótki „skrót” wiadomości. Wykorzystuje się je między innymi przy tworzeniu podpisów cyfrowych. Za ich pomocą można również np. porównać dwa pliki i jeśli oba zwracają taką samą wartość skrótu to z bardzo dużym prawdopodobieństwem można stwierdzić, że są one takie same i nie zostały zmodyfikowane (integralność). Wykorzystywane są również często do uwierzytelnienia wiadomości. Innym przykładem zastosowania takich funkcji jest to, że w bazach danych systemów informatycznych zazwyczaj zapamiętywane są skróty z haseł użytkowników, zamiast konkretnych ciągów znaków. W przypadku wykradzenia takiej bazy danych, atakujący nadal nie ma możliwości odczytać danych służących do uwierzytelnienia przez użytkownika.

Dobre funkcje skrótu powinny posiadać trzy cechy:

* Odporność na kolizje – brak możliwości wygenerowania dwóch dowolnych wiadomości o tym samym skrócie
* Odporność na kolizje konkretnych wiadomości – brak możliwości wygenerowania wiadomości o takim samym skrócie jak wskazana wiadomość
* Jednokierunkowość – brak możliwości wnioskowania o wiadomości na podstawie jej skrótu. Zmiana pojedynczego bita w wiadomości powinna znacząco zmieniać wartość skrótu.

SHA-3 powstało w 4 wariantach, każdy różniący się długością skrótu. Są to długości: 224 bity, 256 bitów, 384 bity, 512 bitów.

1. Narzędzia i struktura implementacji

Implementacja funkcji skrótu SHA-3 została zrealizowana w języku C++ jako biblioteka współdzielona DLL. Podczas tworzenia oprogramowania zostało użyte środowisko programistyczne Visual  Studio 2017. W ramach projektu została również napisana aplikacja konsolowa w języku C++, umożliwiająca wyliczanie skrótów SHA-3.

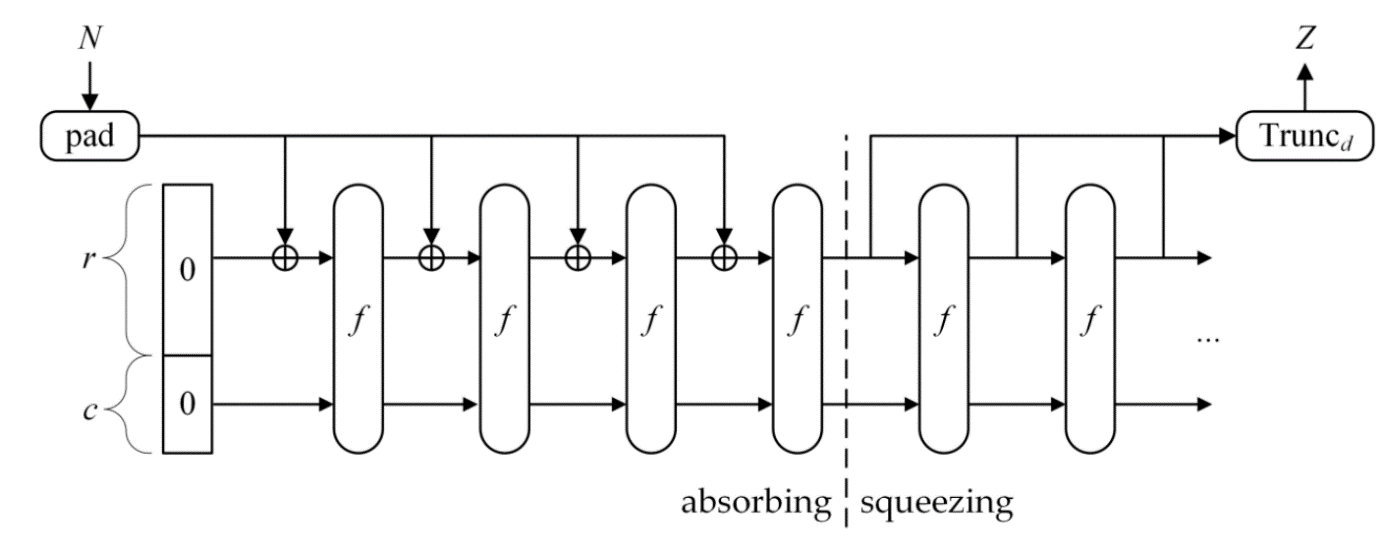
Aplikacja okienkowa, która wywołuje funkcje SHA-3 z biblioteki współdzielonej, została napisana w języku Python. Na okienku użytkownik ma opcję wyboru pliku, rodzaju algorytmu oraz przycisk uruchamiający procedurę obliczania skrótu SHA-3.

Biblioteka współdzielona generuje plik z logami, do którego zapisywane są informacje o przebiegu działania algorytmu.

Do przetestowania wydajności działania SHA-3 został napisany program w języku Python, który oblicza funkcje skrótu z wielu plików. Korzysta w tym celu z zaimplementowanej przez nas biblioteki oraz z biblioteki libcrypto++. Program oblicza wszystkie warianty długościowe SHA-3 oraz generuje wykresy porównujące wydajność obu bibliotek.

1. Opis algorytmu

Szczegółowy opis algorytmu SHA-3 został zamieszczony w dokumencie FIPS-202 [2]. Nazwa algorytmu implementującego SHA-3 to *Keccak*. Ma on strukturę gąbki (*sponge)*. Analogia do gąbki polega na tym, że bity wejściowe są „wchłaniane” do stanu funkcji, po czym pewna ilość bitów wyjściowych jest „wyciskana” ze stanu funkcji.



Rysunek 1 Konstrukcja gąbki

Funkcja sponge przyjmuje jako argumenty: ciąg bitów *N* oraz długość wyjściowego ciągu *d* (w bitach). Ciąg *N* jest uzupełniany o padding. Funkcja *f* to funkcja mieszająca stanu. Składa się z 24 rund, podczas których wykonywane jest 5 funkcji:

* Θ (theta) – miesza odpowiednie bity ze sobą
* ρ (rho) – stosuje rotacje bitowe
* π (Pi) – przestawienie bitów
* χ (Chi) – mieszanie bitów
* ι (Jota) – przekształcenie niektórych bitów leżących w Lane[0,0] o wartość zależącą od rundy

Ciało funkcji *f* wygląda w pseudokodzie następująco:

// m – ciąg na podstawie którego obliczany będzie skrót

// A – aktualna tablica stanu

keccakPermutation(String m, StateArray A)

{

convertStringToStateArray(m, A);

for (i = 0; i < 24; i++)

{

keccakTheta(A);

keccakRho(A);

keccakPi(A);

keccakChi(A);

keccakJota(A, round);

}

return convertStateArrayToString(A);

}

**Algorytm 1: θ(A)**  
*Input*:  
tablica stanu **A**.  
*Output*:  
tablica stanu **A′**.  
*Kroki*:  
1. Dla wszystkich par (*x*, *z*) takich że 0≤*x*<5 i 0≤*z*<*w*, niech  
 *C*[*x*, *z*]=**A**[*x*, 0, *z*] ⊕ **A**[*x*, 1, *z*] ⊕ **A**[*x*, 2, *z*] ⊕ **A**[*x*, 3, *z*] ⊕ **A**[*x*, 4, *z*].  
2. Dla wszystkich par (*x, z*) takich że 0≤*x*<5 i 0≤*z*<*w* niech  
 *D*[*x*, *z*]=*C*[(*x*−1) mod 5, *z*] ⊕ *C*[(*x*+1) mod 5, (*z* –1) mod *w*].  
3. Dla wszystkich (*x*, *y*, *z*) takich że 0≤*x*<5, 0≤*y*<5, i 0≤*z*<*w*, niech  
 **A′**[*x*, *y*, *z*] = **A**[*x*, *y*, *z*] ⊕ *D*[*x*, *z*].

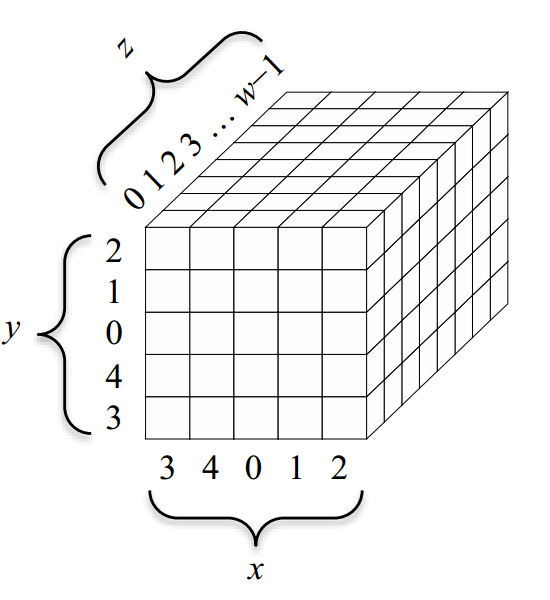
**Algorytm 2: ρ(A)***Input*:  
tablica stanu **A**.  
*Output*:  
tablica stanu **A′**.  
*Kroki*:  
1. Dla wszystkich *z* takich że 0≤*z*<*w*, niech **A′** [0, 0, *z*] = **A**[0, 0, *z*].  
2. Niech (*x*, *y*) = (1, 0).  
3. Dla *t* od 0 do 23:  
 a. dla wszystkich *z* takich że 0≤*z*<*w*, niech **A′**[*x*, *y*, *z*] = **A**[*x*, *y*, (*z–*(*t*+1)(*t*+2)/2) mod *w*];  
 b. niech (*x*, *y*) = (*y*, (2*x*+3*y*) mod 5).  
4. Zwróć **A′.**

**Algorithm 3: π(A)**  
*Input*:  
tablica stanu **A**.  
*Output*:  
tablica stanu **A′**.  
*Kroki*:  
1. Dla wszystkich (*x*, *y*, *z*) takich że 0≤*x*<5, 0≤*y*<5, i 0≤*z*<*w*, niech  
 **A′**[*x*, *y*, *z*]=**A**[(*x* + 3*y*) mod 5, *x*, *z*].  
2. Zwróć **A′**.

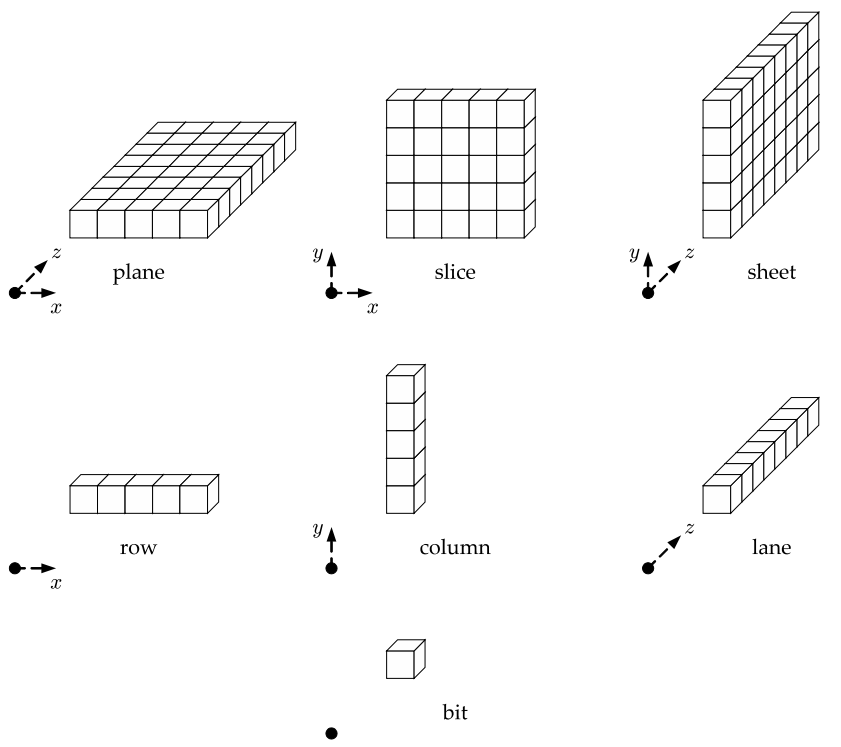
**Algorytm 4: χ(A)***Input*:  
tablica stanu **A**.  
*Output*:  
tablica stanu **A′**.  
*Kroki*:  
1. Dla wszystkich (*x*, *y*, *z*) takich że 0≤*x*<5, 0≤*y*<5, i 0≤*z*<*w*, niech  
 **A′**[*x*, *y*, *z*] = **A**[*x*, *y*, *z*] ⊕ ((**A**[(*x*+1) mod 5, *y*, *z*] ⊕ 1) ⋅ **A**[(*x*+2) mod 5, *y*, *z*]).  
2. Zwróć **A′**

**Algorytm 5: ι(A, *ir*)***Input*:  
tablica stanu **A**;  
indeks rundy *ir*.  
*Output*:  
tablica stanu **A′**.  
*Kroki*:  
1. Dla wszystkich (*x*, *y*, *z*) takich że 0≤*x*<5, 0≤*y*<5, i 0≤*z*<*w*, niech **A′**[*x*, *y*, *z*] = **A**[*x*, *y*, *z*].  
2. Niech *RC*=0w.  
3. Dla *j* od 0 do *l*, niech *RC*[2*j*–1]=*rc*(*j*+7*ir*).  
4. Dla wszystkich *z* takich że 0≤*z*<*w*, niech **A′**[0, 0, *z*]=**A′**[0, 0, *z*] ⊕ *RC*[*z*].  
5. Zwróć **A′.**

Tablica stanu:

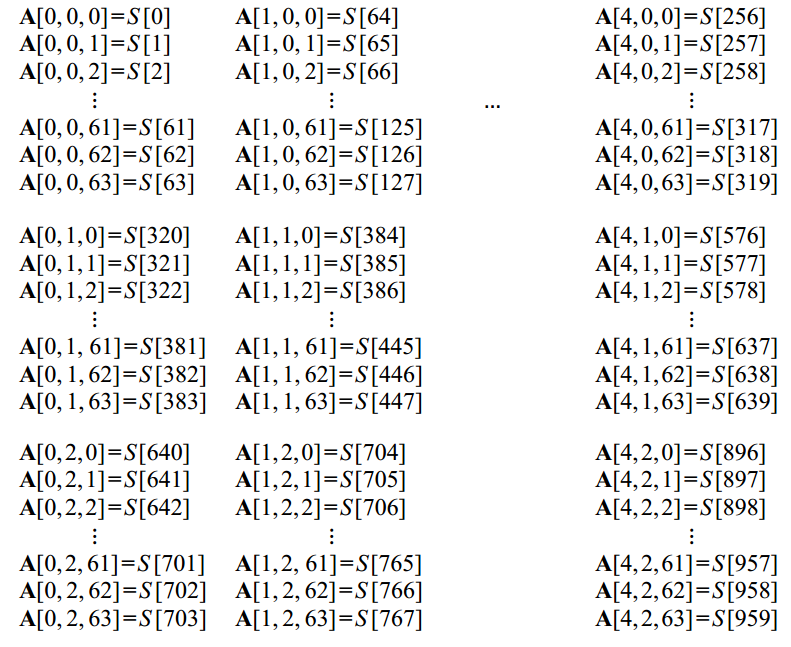


Rysunek 2 Tablica stanu wraz z opisem współrzędnych



Rysunek 3 Części tablicy stanu pod względem wymiaru

Sposób tworzenia tablicy stanu A z ciągu bitów S został przedstawiony poniżej:



1. Zastosowania praktyczne implementowanego zagadnienia

Funkcje skrótu są powszechnie stosowane w dzisiejszej informatyce. Są bardzo ważnym komponentem podpisu elektronicznego – to właśnie skrót dokumentu jest podpisywany przez użytkownika. Dzięki funkcjom skrótu możemy w łatwy sposób sprawdzić czy dane w obu plikach są takie same. Jeśli skróty się różnią, to znaczy, że pliki różnią się zawartością. Bardzo często skróty są wyliczane z haseł i zapamiętywane w bazach danych, aby nie zapisywać hasła czystym tekstem. Utrudnia to bardzo mocno kradzież danych uwierzytelniających do kont takiego systemu.

SHA-3 jest nowym standardem, który jest następcą SHA-2. Cechuje go między innymi wyższa wydajność. NIST nie planuje jednak wycofywać SHA-2. Celem stworzenia algorytmu SHA-3 było to, aby w razie potrzeby można było nim zastąpić bezpośrednio SHA-2 i aby poprawić odporność ogólnego zestawu narzędzi NIST do obliczania funkcji skrótu.

W ramach projektu zostały stworzone 3 komponenty:

* Aplikacja konsolowa – umożliwia na szybkie policzenie skrótu SHA-3 z poziomu konsoli
* Biblioteka DLL – pozwala na użycie zaimplementowanych funkcji SHA-3 w innych programach
* Aplikacja okienkowa – pozwala na policzenie skrótu w łatwy, intuicyjny sposób z poziomu okienka

Zaletą aplikacji okienkowej jest również to, że wyświetlany w niej jest przebieg działania programu. Zyskuje dzięki temu charakter dydaktyczny, pozwalający każdemu na prześledzenie poszczególnych kroków algorytmu.

1. Instrukcja użytkowania programu

Istnieją dwie wersje programu: aplikacja konsolowa oraz aplikacja okienkowa.

* 1. Aplikacja konsolowa

Aplikacja konsolowa służy do wyliczania skrótów z plików z poziomu konsoli. Aplikacja posiada następujące przełączniki:

Compute SHA3 hash of file:

-h [ --help ] Help screen

--224 file Program computes SHA3-224 of file

--256 file Program computes SHA3-256 of file

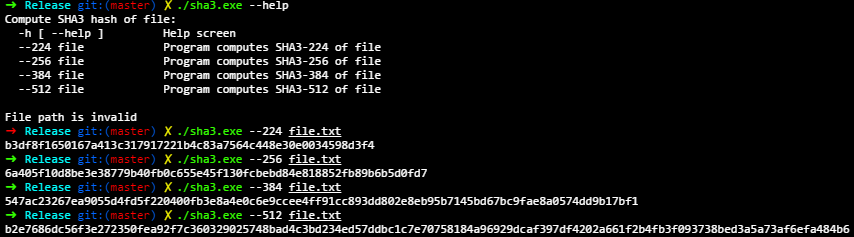
--384 file Program computes SHA3-384 of file

--512 file Program computes SHA3-512 of file

Przykładowo w celu wyliczenia skrótu SHA3-224 z pliku *file.txt* należy użyć komendy

*sha3.exe --224 file.txt*

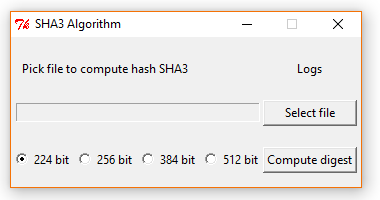
Na poniższym rysunku znajduje się zrzut ekranu z przykładowego korzystania z aplikacji konsolowej.



Rysunek 4 Korzystanie z aplikacji konsolowej

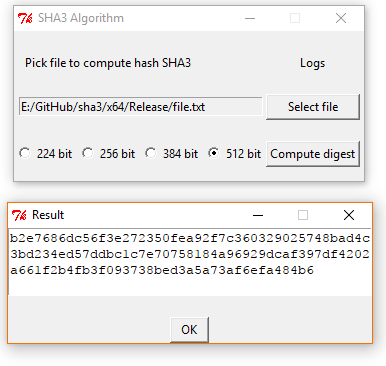
* 1. Aplikacja okienkowa

Aplikacja okienkowa umożliwia wyliczanie skrótów SHA3 wybranych plików. Po uruchomieniu aplikacji, użytkownikowi ukazuje się główne okno, które zostało przedstawione na Rysunku 5. Użytkownik ma z jego poziomu możliwość wybrania pliku, z którego chce policzyć skrót SHA3. Okno umożliwia również wybranie interesującej użytkownika długości skrótu, który ma zostać policzony.



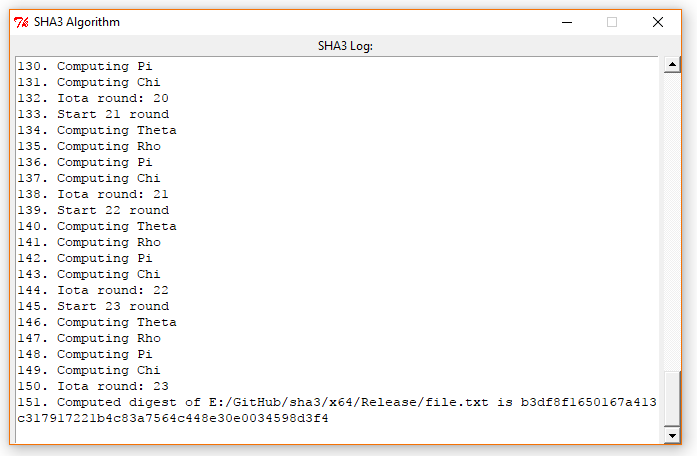
Rysunek 5 Główne okno aplikacji

Po wybraniu pliku i uruchomieniu obliczania skrótu użytkownikowi ukazuje się nowe okienko, w którym znajduje się wynik działania funkcji. Zrzut ekranu z takiej sytuacji został zaprezentowany na Rysunku 6.



Rysunek 6 Rezultat funkcji

Użytkownik ma również możliwość włączenia logów aplikacji w celu prześledzenia działania algorytmu. Po kliknięciu przycisku *Logs* w głównym oknie aplikacji użytkownikowi pokazuje się okno, w którym znajduje się cały przebieg działania algorytmu. Zostało ono przedstawione na Rysunku 7.



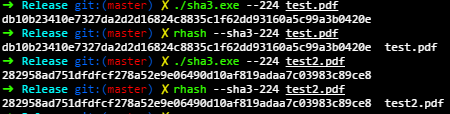
Rysunek 7 Okno z przebiegiem algorytmu

1. Raport z testów aplikacji
   1. Weryfikacja poprawności działania aplikacji

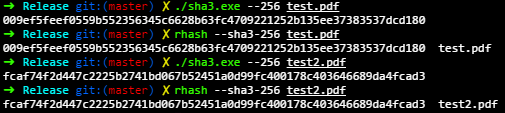
Weryfikacja poprawności działania aplikacji polegała na porównaniu obliczonych skrótów SHA3, ze skrótami wygenerowanymi przez konsolową aplikację RHash[[1]](#footnote-1), którą można znaleźć w wielu dystrybucjach Linuxa.

Podczas weryfikacji została użyta konsolowa aplikacja napisana w języku C++ o nazwie *sha3.exe*.

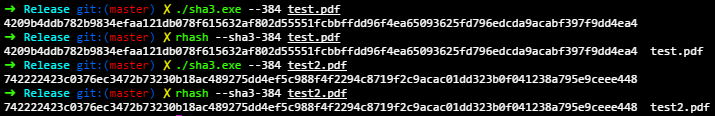
Na poniższych zrzutach ekranu znajdują się dowody poprawności działania zaimplementowanej aplikacji dla wszystkich długości skrótu SHA3.



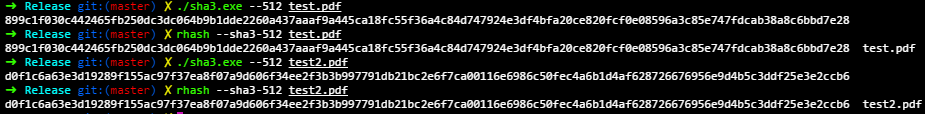
Rysunek 8 Weryfikacja skrótu SHA3-224



Rysunek 9 Weryfikacja skrótu SHA-256



Rysunek 10 Weryfikacja skrótu SHA3-384



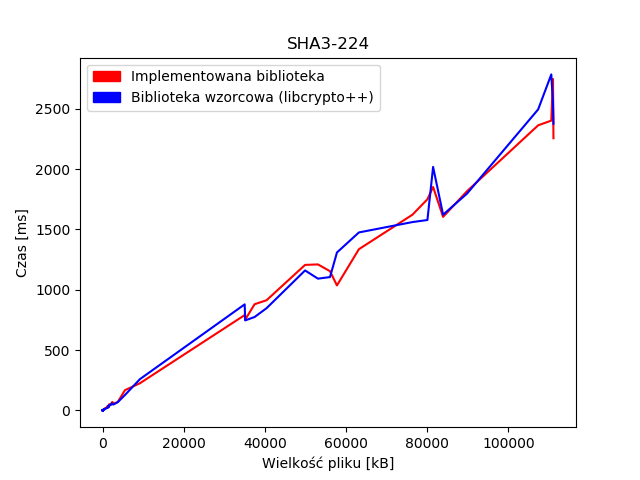
Rysunek 11 Weryfikacja skrótu SHA3-512

* 1. Testy wydajności aplikacji

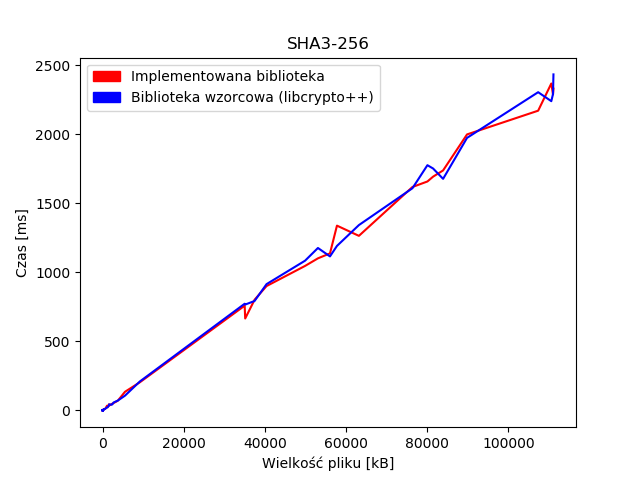
Testom wydajności podlegała napisana w języku C++ biblioteka współdzielona DLL. Wyniki były porównywane z implementacją z biblioteki libcrypto++[[2]](#footnote-2). Libcrypto++ to darmowa biblioteka kryptograficzna, napisana w języku C++.

W celu automatyzacji procesu testów obu bibliotek, został napisany skrypt w języku Python, który wywoływał z nich funkcje liczenia skrótów SHA3 dla kilkudziesięciu różnych plików i gromadził czasy ich wykonania. Na sam koniec skrypt generował wykresy zestawiające ze sobą otrzymane wyniki dla obu bibliotek, w zależności czasu wykonania od długości pliku.

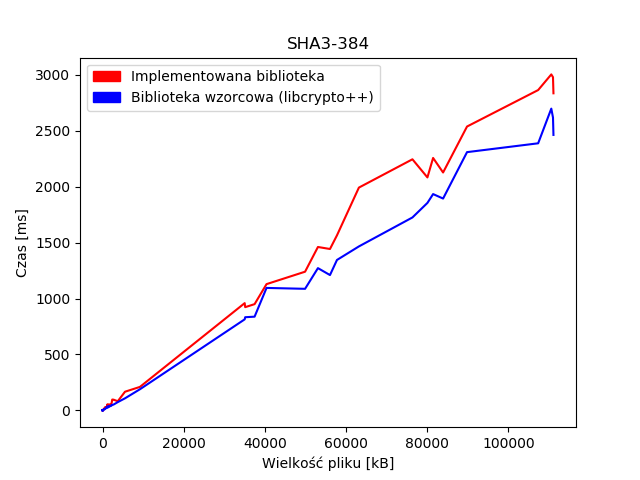
Testy zostały przeprowadzone na 81 plikach różnych formatów (dokumenty pdf, pliki tekstowe, pliki binarne, dokumenty docx, programy wykonywalne exe), w rozmiarach od 1 kB do około 100 MB.



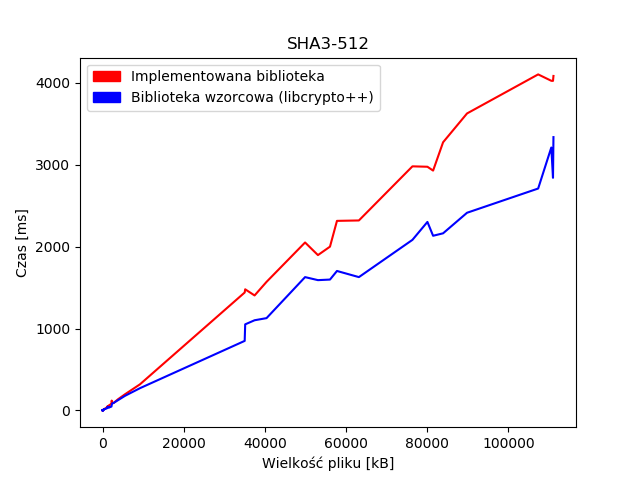
Rysunek 12 Porównanie wydajności SHA3-224



Rysunek 13 Porównanie wydajności SHA3-256



Rysunek 14 Porównanie wydajności SHA3-384



Rysunek 15 Porównanie wydajności SHA3-512

Podczas analizy wykresów można zauważyć, że mają one w dobrym przybliżeniu charakter liniowy. Czas wyliczania skrótu SHA3 przyrasta liniowo w stosunku wzrostu długości pliku. Czas wyliczania skrótów o długościach 224 oraz 256 bitów był w obu bibliotekach bardzo zbliżony. Większe rozbieżności można zauważyć przy dłuższych funkcjach skrótu tj. 384 oraz 512 bitów. Tutaj lepszą wydajnością wykazała się biblioteka libcrypto++. Jej wydajność była lepsza o około kilka do kilkunastu procent.

# Bibliografia

|  |  |
| --- | --- |
| [1] | NIST, „NIST,” 2015. [Online]. Available: https://www.nist.gov/news-events/news/2015/08/nist-releases-sha-3-cryptographic-hash-standard. |
| [2] | NIST, „FIPS PUB 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions,” Gaithersburg, MD 20899-8900, 2015. |

1. http://rhash.anz.ru/ [↑](#footnote-ref-1)
2. https://www.cryptopp.com/ [↑](#footnote-ref-2)