Security: Funkcje skrótu - hash functions

Do realizacji zadania wykorzystano WSL1 na systemie Windows 2019 server.
Dystrybucja uruchomiona pod systemem WSL to Ubuntu 22.04.4 LTS
Do włączenia legacy providera w OpenSSL wykorzystano poniższy tutorial:
https://lindevs.com/enable-openssl-legacy-provider-on-ubuntu

Zadanie 1

Treść Zadania:
Informacje dotyczące używania programu openssl dostępne są po wydaniu polecenia man openssl lub na stronie pod adresem : https://wiki.openssl.org/index.php/Manual:Dgst(1)
Krok1 : Stwórz plik tekstowy z wiadomością np. „This is a trial message to test digest functions!” i zapisz w pliku msg
Krok 2: Zapisz polecenie obliczające skrót tej wiadomości używając kolejno funkcji: MD5, SHA1, SHA-256.
Krok 3: Wykonaj obliczenia skrótu a wyniki zapisz w plikach MD5dgst, SHA1dgst, SHA256dgst.
Krok 4: Zmodyfikuj wiadomość msg poprzez skasowanie jednego znaku np. znaku !
Krok 5: Ponownie oblicz, wykorzystując w/w funkcje, skrót wiadomości msg po modyfikacji
Krok 6: Porównaj skróty wiadomości msg przed i po modyfikacji.

Realizacja

```
root@WSL:lab5_pliki> ls
'Lab5 - PI zaoczne - funkcje skrótu.pdf'  'Lab5 - funkcja skrótu.pdf'   iha1
iha1.c
root@WSL:lab5_pliki> echo "This is a trial message to test digest
functions!" > msg
root@WSL:lab5_pliki> openssl dgst -md5 msg
MD5(msg)= 5f554df44e51b0eea071ae49a854223e
root@WSL:lab5_pliki> openssl dgst -sha1 msg
SHA1(msg)= fdc68fd28db0c992b376a3edf36727ea66d2d391
root@WSL:lab5_pliki> openssl dgst -sha256 msg
SHA2-256(msg)=
a2636d4da000f8b9174e3e1135c6ccd70a098b1e7e3843c3505e54361a572b44
root@WSL:lab5_pliki> openssl dgst -md5 msg > MD5dgst
root@WSL:lab5_pliki> openssl dgst -sha1 msg > SHA1dgst
root@WSL:lab5_pliki> openssl dgst -sha256 msg > SHA256dgst
root@WSL:lab5_pliki> sed 's/!//' msg > msg_mod
root@WSL:lab5_pliki> mv msg_mod msg
root@WSL:lab5_pliki> openssl dgst -md5 msg > MD5dgst_mod
root@WSL:lab5_pliki> openssl dgst -sha1 msg > SHA1dgst_mod
root@WSL:lab5_pliki> openssl dgst -sha256 msg > SHA256dgst_mod
root@WSL:lab5_pliki> diff MD5dgst MD5dgst_mod
1c1
```

```
< MD5(msg)= 5f554df44e51b0eea071ae49a854223e
---
> MD5(msg)= 946e2d2c199b85da32acfd9177bf562c
root@WSL:lab5_pliki> diff SHA1dgst SHA1dgst_mod
1c1
< SHA1(msg)= fdc68fd28db0c992b376a3edf36727ea66d2d391
---
> SHA1(msg)= 5fcb355462f783d439712044e720091c2b6c8dbb
root@WSL:lab5_pliki> diff SHA256dgst SHA256dgst_mod
1c1
< SHA2-256(msg)=
a2636d4da000f8b9174e3e1135c6ccd70a098b1e7e3843c3505e54361a572b44
---
> SHA2-256(msg)=
cea38c49da18b7b1c2fee0cccbb51f48355b39cb3cd68588eba81a9f3f30ed97
root@WSL:lab5_pliki>
```

Pytania

Co można powiedzieć po porównaniu tych skrótów?

Po porównaniu skrótów wiadomości przed i po modyfikacji można zauważyć, że nawet drobna zmiana (np. usunięcie pojedynczego znaku) prowadzi do całkowicie innego wyniku funkcji skrótu. Jest to cecha charakterystyczna funkcji skrótu zwana efektem lawiny, która zapewnia, że zmiana choćby jednego bitu danych wejściowych znacznie zmienia wartość skrótu.

Jakiej długości (wyrażonej w bitach) są poszczególne skróty?

- MD5: 128 bitów (16 bajtów)
- SHA1: 160 bitów (20 bajtów)
- SHA-256: 256 bitów (32 bajty)

Każda funkcja skrótu zawsze generuje wynik o ustalonej długości niezależnie od długości danych wejściowych.

Zadanie 2

Treść Zadania:
W tym zadaniu wykorzystywany jest przykładowy algorytm skrótu zaimplementowany w języku C (program iha1.c) Algorytm nazywa się IHA1 (Insecure Hash Agorithm) i działa na zasadzie funkcji XOR wykonywanej bitowo, a wynikiem jest liczba reprezentowana na 4 bitach (jedna cyfra w zapisie hexadecymalnym).
Krok 1: Dokonaj kompilacji pliku iha1.c poprzez wydanie polecenia:

```
gcc iha1.c -o iha1
```

Krok 2: Przygotuj plik z tekstem jawnym np. „This is a test message”.
Krok 3: Użyj funkcji iha1 do obliczenia skrótu wiadomości zapisanej w w/w pliku.
Składnia wywołania jest następująca:

```
iha1 nazwa_pliku
```

Przykładowy wynik działania programu iha1:

```
root@kali:~/lab11/support> ./iha1 message
    iha1(message) = A
```

Krok 4: Spróbuj znaleźć kolizję (stwórz inną wiadomość jawną która będzie miała taki sam skrót co wiadomość stworzona w tym zadaniu). W tym celu wykonaj 160 prób, w każdej próbie podając na wejście funkcji skrótu inną wartość.

Realizacja

```
root@WSL:lab5_pliki> ls
'Lab5 - PI zaoczne - funkcje skrótu.pdf'  'Lab5 - funkcja skrótu.pdf'
MD5dgst   MD5dgst_mod   SHA1dgst   SHA1dgst_mod   SHA256dgst
SHA256dgst_mod   iha1.c   msg
root@WSL:lab5_pliki> gcc iha1.c -o iha1
root@WSL:lab5_pliki> echo "This is a test message" > message
root@WSL:lab5_pliki> ./iha1 message
iha1(message) = A
root@WSL:lab5_pliki> cat > script.sh
mkdir collisions
cd collisions

for i in $(seq 1 160); do
echo "Test message number $i" > test_$i.txt
../iha1 test_$i.txt >> results.txt
done

^C
root@WSL:lab5_pliki> chmod +x script.sh
root@WSL:lab5_pliki> ./script.sh
root@WSL:lab5_pliki> ls
'Lab5 - PI zaoczne - funkcje skrótu.pdf'   MD5dgst       SHA1dgst
SHA256dgst       collisions   iha1.c    msg
'Lab5 - funkcja skrótu.pdf'                MD5dgst_mod   SHA1dgst_mod
SHA256dgst_mod   iha1         message   script.sh
root@WSL:lab5_pliki> cd collisions/
root@WSL:collisions> ls
results.txt  test_11.txt  test_121.txt test_133.txt test_145.txt
test_157.txt
```

```
[...]
test_35.txt  test_47.txt  test_59.txt  test_70.txt  test_82.txt  test_94.txt
root@WSL:collisions> cat results.txt | grep '= A'
iha1(test_68.txt) = A
iha1(test_79.txt) = A
iha1(test_86.txt) = A
iha1(test_97.txt) = A
root@WSL:collisions>
```

Pytania

Ile średnio potrzebnych jest operacji, żeby znaleźć kolizję dla wyżej opisanego przypadku? Odpowiedź uzasadnij.

Algorytm IHA1 zwraca wynik skrótu reprezentowany jako jedna cyfra szesnastkowa, co oznacza że wynik może przyjąć jedną z 16 wartości (od 0 do F, czyli $2^4 = 16$ możliwych wyników).

Zgodnie z teorią prawdopodobieństwa oraz tzw. paradoksem urodzin, średnia liczba operacji potrzebna do znalezienia kolizji dla funkcji skrótu o n możliwych wartościach wynosi około: $$\sqrt{\frac{\pi n}{2}} \approx 1.253 \cdot \sqrt{n}$$ Dla $n = 16$: $$\sqrt{\frac{\pi \cdot 16}{2}} \approx \sqrt{8\pi} \approx 5.01$$

Zatem średnio potrzeba około 5 różnych wiadomości (czyli 4–5 prób), aby z dużym prawdopodobieństwem znaleźć kolizję dla funkcji skrótu zwracającej 4-bitowy wynik. W najgorszym przypadku może to być do 16 prób, jednak statystycznie znacznie szybciej znajdziemy kolizję.

Zadanie 3

Treść Zadania:
Krok 1: Na maszynie A przygotuj wiadomość zapisaną w pliku np. „This is a test message used to calculate a keyed digest”.
Krok 2: Na maszynie A przygotuj klucz i zapisz go w pliku.
Krok 3: Zapisz i wykonaj polecenie openssl, które pozwoli na obliczenie funkcji HMAC na przygotowanej wiadomości z kluczem zapisanym w pliku z użyciem algorytmu MD5.
Krok 4: Zapisz i wykonaj polecenie openssl, które pozwoli na weryfikację poprawności obliczonej funkcji HMAC na podstawie znanej wiadomości oraz znanego klucza.

Realizacja

```
root@WSL:lab5_pliki> ls
'Lab5 - PI zaoczne - funkcje skrótu.pdf'   MD5dgst       SHA1dgst
SHA256dgst       collisions   iha1.c    msg
'Lab5 - funkcja skrótu.pdf'                MD5dgst_mod   SHA1dgst_mod
SHA256dgst_mod   iha1         message   script.sh
```

```
root@WSL:lab5_pliki> echo "This is a test message used to calculate a keyed
digest" > message.txt
root@WSL:lab5_pliki> echo "secretkey123" > key.txt
root@WSL:lab5_pliki> chmod 600 key.txt
root@WSL:lab5_pliki> openssl dgst -md5 -mac HMAC -macopt key:file:key.txt
message.txt > hmac.md5
root@WSL:lab5_pliki> openssl dgst -md5 -mac HMAC -macopt key:file:key.txt
message.txt
HMAC-MD5(message.txt)= b6e3401fc3288777c285655aabd5d6d6
root@WSL:lab5_pliki> cat hmac.md5
HMAC-MD5(message.txt)= b6e3401fc3288777c285655aabd5d6d6
root@WSL:lab5_pliki>
```

Pytania

Co musi wiedzieć użytkownik maszyny B jeżeli chce zweryfikować prawdziwość skrótu HMAC z kluczem? Odpowiedź uzasadnij.

Użytkownik maszyny B musi znać dokładnie ten sam klucz symetryczny, który został użyty przez maszynę A do obliczenia funkcji HMAC. Jest to niezbędne, ponieważ funkcja HMAC wykorzystuje klucz nie tylko do skrócenia wiadomości, ale do generowania unikalnego podpisu kryptograficznego. Bez znajomości tego klucza niemożliwe jest odtworzenie tego samego HMAC i tym samym jego weryfikacja.

Jakie operacje musi wykonać użytkownik maszyny B jeżeli chce zweryfikować prawdziwość wiadomości i skrótu otrzymanych od użytkownika maszyny A? Odpowiedź uzasadnij.

Aby zweryfikować prawdziwość wiadomości i skrótu HMAC, użytkownik maszyny B musi:

1. Otrzymać wiadomość i oryginalny HMAC od użytkownika maszyny A.
2. Upewnić się, że posiada ten sam klucz, który został użyty do wygenerowania HMAC.
3. Obliczyć lokalnie nowy HMAC z otrzymanej wiadomości, używając tego samego algorytmu (MD5) i klucza.
4. Porównać wynik z otrzymanym HMAC.

Jeśli oba skróty są identyczne, oznacza to, że wiadomość nie została zmodyfikowana oraz że HMAC został wygenerowany z właściwego klucza. W przeciwnym razie wiadomość mogła zostać zmieniona lub klucz jest niepoprawny.

Zadanie 4

Treść Zadania: Dokonaj oceny wydajności algorytmów obliczających skrót wiadomości. Wypełnij Tabelę 1 wpisując wartość średnią czasu obliczania funkcji skrótu obliczoną z 12 pomiarów. Stwórz

pliki o rozmiarach: 100kB, 1MB, 10MB, 100MB. Do tego celu można użyć poniższej komendy:

```
openssl rand -out nazwa_pliku.txt rozmiar_pliku_w_bajtach
```

Do sprawdzenia czasu wykonywania operacji obliczania skrótu można wykorzystać komendę 'time':

```
time polecenie_obliczające_skrót
```

Przykładowy wynik polecenia: time

```
real 0m5.126s
user 0m0.004s
sys 0m0.012s
```

Wpisz przedziały ufności obliczone dla poziomu ufności równego 95%. Wyniki przedstaw również w postaci wykresu. W sprawozdaniu umieść analizę wyników wydajności obliczania funkcji skrótu (sprawdź wyniki – uzyskane czasy dla różnych rozmiarów wiadomości wejściowych oraz dla różnych rozmiarów funkcji skrótu).

Realizacja

```
root@WSL:lab5_pliki> ls
'Lab5 - PI zaoczne - funkcje skrótu.pdf'   MD5dgst       SHA1dgst
SHA256dgst       collisions   iha1     key.txt   message.txt   script.sh
'Lab5 - funkcja skrótu.pdf'                MD5dgst_mod   SHA1dgst_mod
SHA256dgst_mod   hmac.md5     iha1.c   message   msg
root@WSL:lab5_pliki> openssl rand -out file_100KB.txt 102400
root@WSL:lab5_pliki> openssl rand -out file_1MB.txt 1048576
root@WSL:lab5_pliki> openssl rand -out file_10MB.txt 10485760
root@WSL:lab5_pliki> openssl rand -out file_100MB.txt 104857600
root@WSL:lab5_pliki> ls -lah file_*
-rwxrwxrwx 1 root root 100K May  4 12:01 file_100KB.txt
-rwxrwxrwx 1 root root 100M May  4 12:02 file_100MB.txt
-rwxrwxrwx 1 root root  10M May  4 12:02 file_10MB.txt
-rwxrwxrwx 1 root root 1.0M May  4 12:01 file_1MB.txt
root@WSL:lab5_pliki> for i in {1..12}; do time openssl dgst -md5
file_100KB.txt; done
MD5(file_100KB.txt)= 4f03d8f7e57ee31b3c17495e0ce06af5

real    0m0.024s
user    0m0.000s
sys     0m0.016s
MD5(file_100KB.txt)= 4f03d8f7e57ee31b3c17495e0ce06af5

[...]

SHA2-512(file_100MB.txt)=
a36f6a029dcf6195926179cdd260e6133e8638a300f83eeaf74589ff639b267837f846108dcf
fb85734da58037dd274013f5a649c1d0107b13f8a0872f909a1e
```

```
real    0m0.429s
user    0m0.297s
sys     0m0.141s
root@WSL:lab5_pliki>
```

Analiza wydajności dla każdego algorytmu

Poniżej zestawiono średnie czasy „real” (w sekundach) i 95% przedziały ufności dla 12 pomiarów każdego algorytmu i rozmiaru pliku. Wspólny współczynnik t-Studenta dla 11 stopni swobody: $t_{0.975,11}\approx2.201$.

Wzory obliczeniowe

Średnia arytmetyczna

$$\bar{x} \;=\;\frac{1}{n}\sum_{i=1}^{n}x_i$$

- **x_i** wartość i-tego pomiaru czasu,
- **n** liczba pomiarów (w naszym przypadku $n=12$),
- **$\bar{x}$** średnia arytmetyczna zestawu pomiarów.

Odchylenie standardowe próbki

$$s \;=\;\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

- **x_i** wartość i-tego pomiaru,
- **$\bar{x}$** średnia arytmetyczna pomiarów,
- **n** liczba pomiarów,
- **s** odchylenie standardowe obliczone dla próby.

Przedział ufności 95%

$$\bar{x} \;\pm\; t_{\alpha/2,\;n-1}\,\frac{s}{\sqrt{n}}$$

- **$\bar{x}$** średnia arytmetyczna pomiarów,
- **s** odchylenie standardowe próby,
- **n** liczba pomiarów,
- **α** poziom istotności (tutaj $\alpha=0{,}05$ dla 95% ufności),
- **$t_{\alpha/2,\;n-1}$** wartość statystyki t-Studenta dla rozkładu o $n-1$ stopniach swobody (tutaj $t_{0{,}975,11}\approx2{,}201$),
- **$\displaystyle\frac{s}{\sqrt{n}}$** błąd standardowy średniej.

MD5

100 kB

- Czasy (s): 0.024, 0.019, 0.015, 0.013, 0.015, 0.016, 0.013, 0.013, 0.013, 0.015, 0.013, 0.022
- Średnia: $\bar{x}=0.01592$, odchylenie standardowe: $s=0.00378$
- Przedział ufności 95%: $[0.01352,\;0.01832]$

1 MB

- Czasy (s): 0.030, 0.027, 0.020, 0.015, 0.018, 0.017, 0.019, 0.018, 0.019, 0.017, 0.017, 0.018
- Średnia: $\bar{x}=0.01958$, odchylenie standardowe: $s=0.00440$
- Przedział ufności 95%: $[0.01679,\;0.02238]$

10 MB

- Czasy (s): 0.064, 0.074, 0.063, 0.046, 0.054, 0.047, 0.046, 0.048, 0.049, 0.052, 0.053, 0.049
- Średnia: $\bar{x}=0.05375$, odchylenie standardowe: $s=0.00878$
- Przedział ufności 95%: $[0.04817,\;0.05933]$

100 MB

- Czasy (s): 0.377, 0.329, 0.328, 0.344, 0.320, 0.327, 0.378, 0.459, 0.383, 0.361, 0.325, 0.331
- Średnia: $\bar{x}=0.35517$, odchylenie standardowe: $s=0.04007$
- Przedział ufności 95%: $[0.32971,\;0.38063]$

SHA-1

100 kB

- Czasy (s): 0.024, 0.022, 0.016, 0.019, 0.017, 0.020, 0.037, 0.025, 0.025, 0.023, 0.021, 0.024
- Średnia: $\bar{x}=0.02275$, odchylenie standardowe: $s=0.00540$
- Przedział ufności 95%: $[0.01932,\;0.02618]$

1 MB

- Czasy (s): 0.032, 0.025, 0.017, 0.018, 0.017, 0.019, 0.020, 0.016, 0.019, 0.018, 0.019, 0.019
- Średnia: $\bar{x}=0.01992$, odchylenie standardowe: $s=0.00442$
- Przedział ufności 95%: $[0.01711,\;0.02273]$

10 MB

- Czasy (s): 0.065, 0.048, 0.055, 0.051, 0.048, 0.050, 0.049, 0.046, 0.047, 0.051, 0.053, 0.059

- Średnia: $\bar{x}=0.05183$, odchylenie standardowe: $s=0.00552$
- Przedział ufności 95%: $[0.04832,\;0.05534]$

100 MB

- Czasy (s): 0.357, 0.313, 0.312, 0.344, 0.313, 0.306, 0.306, 0.343, 0.318, 0.463, 0.438, 0.370
- Średnia: $\bar{x}=0.34858$, odchylenie standardowe: $s=0.05231$
- Przedział ufności 95%: $[0.31535,\;0.38182]$

RIPEMD-160

100 kB

- Czasy (s): 0.029, 0.027, 0.021, 0.013, 0.016, 0.017, 0.015, 0.017, 0.016, 0.016, 0.016, 0.014
- Średnia: $\bar{x}=0.01808$, odchylenie standardowe: $s=0.00504$
- Przedział ufności 95%: $[0.01488,\;0.02128]$

1 MB

- Czasy (s): 0.033, 0.027, 0.022, 0.022, 0.021, 0.019, 0.021, 0.022, 0.023, 0.021, 0.024, 0.023
- Średnia: $\bar{x}=0.02317$, odchylenie standardowe: $s=0.00419$
- Przedział ufności 95%: $[0.02045,\;0.02589]$

10 MB

- Czasy (s): 0.091, 0.072, 0.084, 0.074, 0.083, 0.101, 0.082, 0.078, 0.075, 0.081, 0.074, 0.077
- Średnia: $\bar{x}=0.08158$, odchylenie standardowe: $s=0.00870$
- Przedział ufności 95%: $[0.07639,\;0.08678]$

100 MB

- Czasy (s): 0.590, 0.602, 0.598, 0.579, 0.599, 0.578, 0.613, 0.738, 0.642, 0.598, 0.583, 0.599
- Średnia: $\bar{x}=0.61450$, odchylenie standardowe: $s=0.04909$
- Przedział ufności 95%: $[0.58439,\;0.64461]$

SHA-256

100 kB

- Czasy (s): 0.028, 0.020, 0.019, 0.014, 0.014, 0.014, 0.013, 0.020, 0.015, 0.017, 0.016, 0.018
- Średnia: $\bar{x}=0.01775$, odchylenie standardowe: $s=0.00425$
- Przedział ufności 95%: $[0.01511,\;0.02039]$

1 MB

- Czasy (s): 0.066, 0.022, 0.018, 0.025, 0.023, 0.023, 0.025, 0.025, 0.026, 0.027, 0.025, 0.026
- Średnia: $\bar{x}=0.02483$, odchylenie standardowe: $s=0.01327$
- Przedział ufności 95%: $[0.01893,\;0.03074]$

10 MB

- Czasy (s): 0.118, 0.067, 0.070, 0.066, 0.088, 0.092, 0.211, 0.101, 0.084, 0.096, 0.101, 0.119
- Średnia: $\bar{x}=0.10167$, odchylenie standardowe: $s=0.03507$
- Przedział ufności 95%: $[0.07639,\;0.12695]$

100 MB

- Czasy (s): 0.547, 0.498, 0.664, 0.596, 0.528, 0.493, 0.531, 0.502, 0.517, 0.499, 0.516, 0.535
- Średnia: $\bar{x}=0.53550$, odchylenie standardowe: $s=0.04933$
- Przedział ufności 95%: $[0.50416,\;0.56684]$

SHA-512

100 kB

- Czasy (s): 0.350, 0.030, 0.022, 0.022, 0.018, 0.018, 0.016, 0.016, 0.016, 0.016, 0.015, 0.016
- Średnia: $\bar{x}=0.04625$, odchylenie standardowe: $s=0.09575$

1 MB

- Czasy (s): 0.021, 0.028, 0.024, 0.023, 0.023, 0.019, 0.018, 0.020, 0.022, 0.024, 0.017, 0.021
- Średnia: $\bar{x}=0.02167$, odchylenie standardowe: $s=0.00303$
- Przedział ufności 95%: $[0.01974,\;0.02359]$

10 MB

- Czasy (s): 0.083, 0.057, 0.062, 0.055, 0.059, 0.086, 0.112, 0.214, 0.085, 0.081, 0.084, 0.085
- Średnia: $\bar{x}=0.08858$, odchylenie standardowe: $s=0.04278$
- Przedział ufności 95%: $[0.06141,\;0.11576]$

100 MB

- Czasy (s): 0.416, 0.390, 0.433, 0.663, 0.587, 0.452, 0.425, 0.394, 0.411, 0.422, 0.398, 0.429
- Średnia: $\bar{x}=0.45167$, odchylenie standardowe: $s=0.08440$
- Przedział ufności 95%: $[0.39804,\;0.50529]$

Wykres

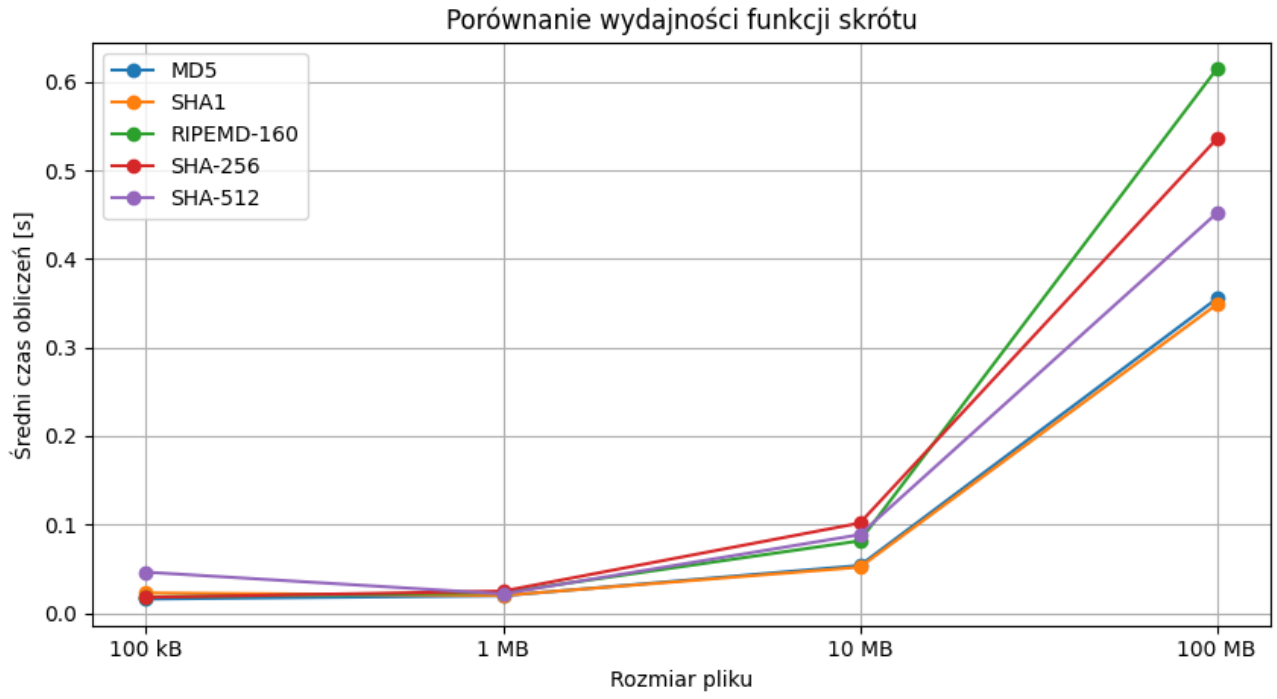
graph_hash.py

```python
import matplotlib.pyplot as plt

# Average times (in seconds) for each algorithm and file size
file_sizes = ["100 kB", "1 MB", "10 MB", "100 MB"]
md5_times = [0.01592, 0.01958, 0.05375, 0.35517]
sha1_times = [0.02275, 0.01992, 0.05183, 0.34858]
ripemd160_times = [0.01808, 0.02317, 0.08158, 0.61450]
sha256_times = [0.01775, 0.02483, 0.10167, 0.53550]
sha512_times = [0.04625, 0.02167, 0.08858, 0.45167]

plt.figure()
plt.plot(file_sizes, md5_times, marker='o', label='MD5')
plt.plot(file_sizes, sha1_times, marker='o', label='SHA1')
plt.plot(file_sizes, ripemd160_times, marker='o', label='RIPEMD-160')
plt.plot(file_sizes, sha256_times, marker='o', label='SHA-256')
plt.plot(file_sizes, sha512_times, marker='o', label='SHA-512')

plt.xlabel("Rozmiar pliku")
plt.ylabel("Sredni czas obliczen [s]")
plt.title("Porównanie wydajności funkcji skrótu")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Wykres porównujący wydajność funkcji skrótu

Tabela

Wydajność obliczania funkcji skrótu (średni czas „real” w sekundach z 12 pomiarów)

	100 kB	**1 MB**	**10 MB**	**100 MB**
MD5	0.0159	0.0196	0.0538	0.3552
SHA1	0.0228	0.0199	0.0518	0.3486
RMD160	0.0181	0.0232	0.0816	0.6145
SHA256	0.0178	0.0248	0.1017	0.5355
SHA512	0.0463	0.0217	0.0886	0.4517