Security: hash functions

WSL1 on Windows 2019 server was used to complete the task.
The distribution running under WSL is Ubuntu 22.04.4 LTS
The following tutorial was used to enable the legacy provider in OpenSSL:
https://lindevs.com/enable-openssl-legacy-provider-on-ubuntu

Task 1

Task content:
Information on how to use openssl is available by issuing the openssl man command or on the website at : https://wiki.openssl.org/index.php/Manual:Dgst(1)
Step1 : Create a text file with a message e.g. „This is a trial message to test digest functions!” and save it in a msg file
Step2 : Write the command that calculates the digest of this message using the functions in turn: MD5, SHA1, SHA-256.
Step 3: Perform the digest calculation and save the results in MD5dgst, SHA1dgst, SHA256dgst files.
Step 4: Modify the msg message by deleting one character, e.g. the !
Step 5: Recalculate, using the above functions, the digest of the msg message after modification
Step 6: Compare the msg message digests before and after modification.

Implementation

```
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>
```

Questions

What can you tell from comparing these abbreviations?

After comparing the message digests before and after modification, it can be seen that even a small change (e.g. deleting a single character) leads to a completely different result of the hash function. This is a characteristic of the hash function called the avalanche effect, which ensures that a change of even one bit of the input data significantly changes the hash value.

What is the length (expressed in bits) of each hash function?

- MD5: 128 bits (16 bytes)
- SHA1: 160 bits (20 bytes)
- SHA-256: 256 bits (32 bytes)

Each hash function always generates a result of a fixed length regardless of the length of the input data.

Task 2

Task content:
This task uses an example of a hash algorithm implemented in C (program iha1.c) The algorithm is called IHA1 (Insecure Hash Agorithm) and works on the principle of a bitwise XOR function, with the result being a number represented on 4 bits (one digit in hexadecimal notation).
Step 1: Compile the iha1.c file by issuing the command:

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

Step 2: Prepare a file with plain text, e.g. „This is a test message”.
Step 3: Use the iha1 function to calculate the hash of the message stored in the above file.
The syntax of the call is as follows:

```
iha1 nazwa_pliku
```

Example result of the iha1 programme:

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

Step 4: Try to find a collision (create another plaintext message that has the same hash as the message created in this task). To do this, make 160 attempts, giving a different value to the input of the hash function in each attempt.

Implementation

```
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>
```

Questions

On average, how many operations are needed to find a collision for the case described above? Justify your answer.

The IHA1 algorithm returns a hash result represented as one hexadecimal digit, which means that the result can take one of 16 values (from 0 to F, i.e. $2^4 = 16$ possible results).

According to probability theory and the so-called birth paradox, the average number of operations needed to find a collision for a hash function with n possible values is approximately: $$sqrt{frac{n}{2}} \$$approx 1.253 $$dot $$sqrt{n}$$ For $n = 16$: $$sqrt{frac{n}{2}} \$$approx $sqrt{8}} \$$approx 5.01 So, on average, it takes about 5 different messages (i.e. 4-5 attempts) to find a collision with high probability for a hash function returning a 4-bit result. In the worst case it could be up to 16 attempts, but statistically it is much faster to find a collision. ====== Task 3 ====== Task content:\\ Step 1: On machine A, prepare a message stored in a file, e.g. "This is a test message used to calculate a keyed digest".\\ Step 2: On machine A, prepare a key and save it in a file.\\ Step 3: Save and execute the openssl command to calculate the HMAC function on the prepared message with the key stored in the file using the MD5 algorithm.\\ Step 4: Save and execute the openssl command, which will allow you to verify the correctness of the calculated HMAC function based on a known message and a known key.\\ ===== Implementation ===== <code numberLines> 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> </code> ===== Questions ===== ==== What does the user of machine B need to know if he wants to verify the veracity of the HMAC digest with the key? Justify your answer. ==== The user of machine B needs to know exactly the same symmetric key that was used by machine A to calculate the HMAC function. This is necessary because the HMAC function uses the key not only to truncate the message, but to generate a unique cryptographic signature. Without knowledge of this key, it is impossible to reproduce the same HMAC and thus verify it. ==== What operations must the user of machine B perform if he wants to verify the veracity of the message and digest received from the user of machine A? Justify your answer. ==== To verify the veracity of the message and the hash of the HMAC, the user of machine B must: - Receive the message and original HMAC from machine user A. - Ensure that it has the same key that was used to generate the HMAC. - Calculate locally the new HMAC from the received message, using the same algorithm (MD5) and key. - Compare the result with the HMAC received. If both hashes are identical, it means that the message has not been modified and that the HMAC was generated from the correct key. Otherwise, the message may have been modified or the key is incorrect. ======

Task 4 ====== Task content: Evaluate the performance of algorithms that compute message digests. Complete Table 1 by entering the average value of the hash function calculation time calculated from 12 measurements. Create file sizes of: 100kB, 1MB, 10MB, 100MB. You can use the following command to do this: <code Bash> openssl rand -out nazwa_pliku.txt rozmiar_pliku_w_bajtach </code> The 'time' command can be used to check the execution time of the hash calculation operation: <code Bash> time polecenie_obliczające_skrót </code> Example result of the command: time <code Bash> real 0m5.126s user 0m0.004s sys 0m0.012s </code> Enter the confidence intervals calculated for a confidence level of 95%. Also present the results in the form of a graph. In your report, include an analysis of the performance results of the hash function calculation (check the results - times obtained for different sizes of input messages and for different sizes of the hash function). ===== Implementation ===== <code numberLines> root@WSL:lab5_pliki> ls 'Lab5 - Pl 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)= a36f6a029dcf6195926179cdd260e6133e8638a300f83eeaf74589ff639b267837f846108dcffb85734da58037dd274013f5a649c1d0107b13f8a0872f909a1e real 0m0.429s user 0m0.297s sys 0m0.141s root@WSL:lab5_pliki> </code> ===== Performance analysis for each algorithm ===== The average 'real' times (in seconds) and 95% confidence intervals for the 12 measurements of each algorithm and file size are summarised below. Common Student's t-ratio for the 11 degrees of freedom: $t_{0.975,11}approx2.201$.

Computational formulae

Arithmetic mean

$$bar{x}=\=}^{n}x_i$$.

- **x_i** The i-value of this time measurement,
- **n** the number of measurements (in our case $n=12$),
- **$bar{x}$** the arithmetic mean of the set of measurements.

The standard deviation of the sample

$$s };=}; $$sqrt{frac{1}{n-1} sum_{i=1}^{n}(x_i - bar{x})^2}$$. * **x_i** The value of the ith measurement,

- **$bar{x}$** the arithmetic mean of the measurements,
- **n** number of measurements,
- **s** standard deviation calculated for the sample.

Confidence interval 95%.

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

- **$\bar{x}$** arithmetic mean of measurements,
- **s** standard deviation of the sample,
- **n** number of measurements,
- **α** significance level (here $\alpha=0{,}05$ for 95% confidence),
- **$t_{\alpha/2,}$.** the value of the Student's t-statistic for a distribution with $n-1$ degrees of freedom (here $t_{0{,}975,11}approx2{,}201$),
- **$\displaystyle\frac{s}{\sqrt{,}201}$$.** standard error of the mean.

MD5

100 kB

- Times (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
- Average: $\bar{x}=0.01592$, standard deviation: $s=0.00378$
- 95% confidence interval: $[0.01352,\;0.01832]$

1 MB

- Times (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
- Average: $\bar{x}=0.01958$, standard deviation: $s=0.00440$
- 95% confidence interval: $[0.01679,\;0.02238]$

10 MB

- Times (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
- Average: $\bar{x}=0.05375$, standard deviation: $s=0.00878$
- 95% confidence interval: $[0.04817,\;0.05933]$

100 MB

- Times (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
- Average: $\bar{x}=0.35517$, standard deviation: $s=0.04007$
- 95% confidence interval: $[0.32971,\;0.38063]$

SHA-1

100 kB

- Times (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
- Average: $\bar{x}=0.02275$, standard deviation: $s=0.00540$

- 95% confidence interval: $[0.01932,\;0.02618]$

1 MB

- Times (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
- Average: $bar{x}=0.01992$, standard deviation: $s=0.00442$
- 95% confidence interval: $[0.01711,\;0.02273]$

10 MB

- Times (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
- Average: $bar{x}=0.05183$, standard deviation: $s=0.00552$
- 95% confidence interval: $[0.04832,\;0.05534]$

100 MB

- Times (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
- Average: $bar{x}=0.34858$, standard deviation: $s=0.05231$
- 95% confidence interval: $[0.31535,\;0.38182]$

RIPEMD-160

100 kB

- Times (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
- Average: $bar{x}=0.01808$, standard deviation: $s=0.00504$
- 95% confidence interval: $[0.01488,\;0.02128]$

1 MB

- Times (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
- Average: $bar{x}=0.02317$, standard deviation: $s=0.00419$
- 95% confidence interval: $[0.02045,\;0.02589]$

10 MB

- Times (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
- Average: $bar{x}=0.08158$, standard deviation: $s=0.00870$
- 95% confidence interval: $[0.07639,\;0.08678]$

100 MB

- Times (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
- Average: $bar{x}=0.61450$, standard deviation: $s=0.04909$

- 95% confidence interval: $[0.58439,\;0.64461]$

SHA-256

100 kB

- Times (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
- Average: $bar{x}=0.01775$, standard deviation: $s=0.00425$
- 95% confidence interval: $[0.01511,\;0.02039]$

1 MB

- Times (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
- Average: $bar{x}=0.02483$, standard deviation: $s=0.01327$
- 95% confidence interval: $[0.01893,\;0.03074]$

10 MB

- Times (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
- Average: $bar{x}=0.10167$, standard deviation: $s=0.03507$
- 95% confidence interval: $[0.07639,\;0.12695]$

100 MB

- Times (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
- Average: $bar{x}=0.53550$, standard deviation: $s=0.04933$
- 95% confidence interval: $[0.50416,\;0.56684]$

SHA-512

100 kB

- Times (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
- Average: $bar{x}=0.04625$, standard deviation: $s=0.09575$

1 MB

- Times (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
- Average: $bar{x}=0.02167$, standard deviation: $s=0.00303$
- 95% confidence interval: $[0.01974,\;0.02359]$

10 MB

- Times (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
- Average: $bar{x}=0.08858$, standard deviation: $s=0.04278$
- 95% confidence interval: $[0.06141,\;0.11576]$

100 MB

- Times (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
- Average: $bar{x}=0.45167$, standard deviation: $s=0.08440$
- 95% confidence interval: $[0.39804,\;0.50529]$

Graph

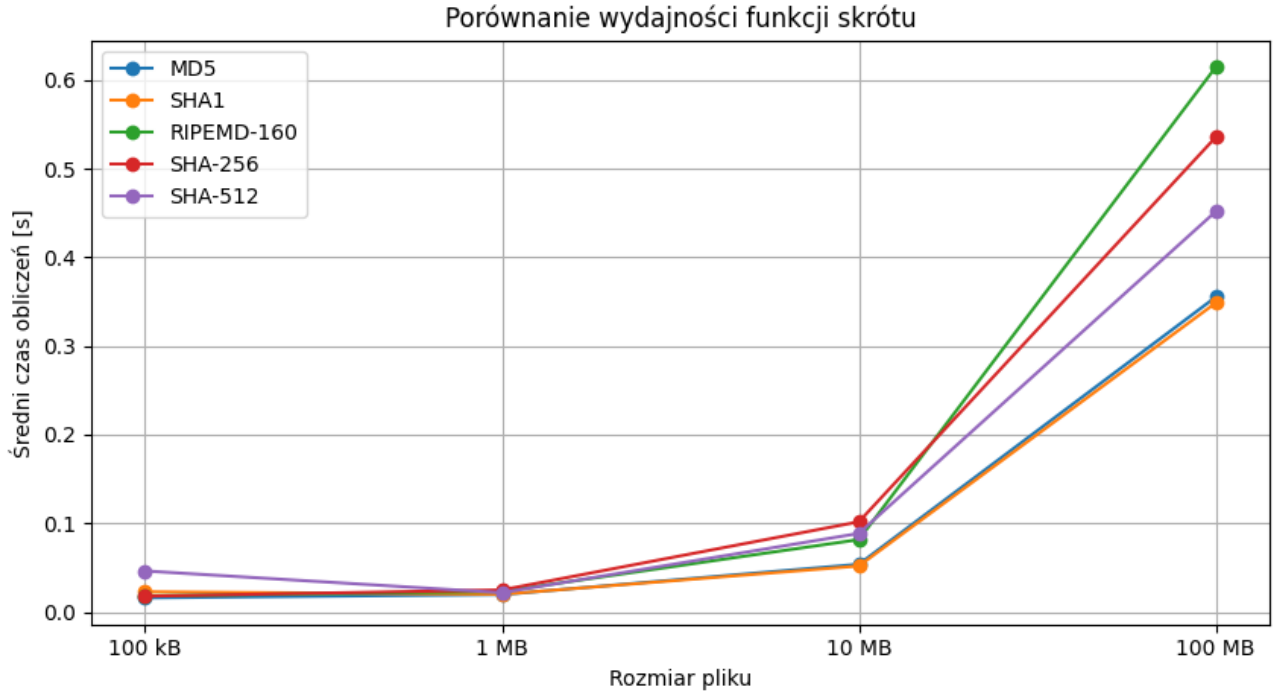
graph_hash.py

```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()
```



Graph comparing the performance of the shortcut function

Table

Performance of shortcut function calculation (average „real” time in seconds from 12 measurements)

	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