

```
(.python_env) [admin@parrot]~]
└─ $openssl version
OpenSSL 3.0.17 1 Jul 2025 (Library: OpenSSL 3.0.17 1 Jul 2025)
(.python_env) [admin@parrot]~]
└─ $openssl list -cipher-algorithms
Legacy:
AES-128-CBC
AES-128-CBC-HMAC-SHA1
AES-128-CBC-HMAC-SHA256
id-aes128-CCM
AES_128_CFB
```

Lista dostępnych algorytmów /\

```
└─ $podman run -dp 10001:10001 --name hashingmd5ex1 docker.io/mazurkatarzyna/hashing-md5-ex1:latest
Trying to pull docker.io/mazurkatarzyna/hashing-md5-ex1:latest...
Getting image source signatures
Copying blob 0ebcc011f0ec skipped: already exists
Copying blob 64b78282ca88 skipped: already exists
Copying blob 92d63ec5cbeb skipped: already exists
Copying blob 8f9bf61351d4 done
Copying blob 457ede85efe6 skipped: already exists
Copying blob 308b4f5d7cef done
Copying blob 2a9e35fdb1ce done
Copying blob 7955333bbf14 done
Copying blob 3da95a905ed5 skipped: already exists
Copying config 7a92f39f1b done
Writing manifest to image destination
Storing signatures
b63ad14ec21765139d04a9a8a16008f734083e4c206a86dd9dab8bd821e4be09
(.python_env) [admin@parrot]~]
└─ $podman run -dp 10001:10001 --name hashingmd5ex1 docker.io/mazurkatarzyna/hashing-md5-ex1:latest
Error: creating container storage: the container name "hashingmd5ex1" is already in use by b63ad14ec21765139d04a9a8a16008f734083e4c206a86dd9dab8bd821e4be09. You have to remove that container to be able to reuse that name: that name is already in use
(.python_env) [x]-[admin@parrot]~]
└─ $^C
(.python_env) [x]-[admin@parrot]~]
└─ $podman rm b63ad14ec21765139d04a9a8a16008f734083e4c206a86dd9dab8bd821e4be09
Error: cannot remove container b63ad14ec21765139d04a9a8a16008f734083e4c206a86dd9dab8bd821e4be09 as it is running - running or paused containers cannot be removed without force: container state improper
```

```
└─ $openssl list -digest-algorithms
Legacy:
RSA-MD4 => MD4
RSA-MD5 => MD5
RSA-RIPEMD160 => RIPEMD160
RSA-SHA1 => SHA1
RSA-SHA1-2 => RSA-SHA1
RSA-SHA224 => SHA224
RSA-SHA256 => SHA256
RSA-SHA3-224 => SHA3-224
RSA-SHA3-256 => SHA3-256
RSA-SHA3-384 => SHA3-384
RSA-SHA3-512 => SHA3-512
RSA-SHA384 => SHA384
RSA-SHA512 => SHA512
RSA-SHA512/224 => SHA512-224
RSA-SHA512/256 => SHA512-256
RSA-SM3 => SM3
BLAKE2b512
BLAKE2s256
id-rsassa-pkcs1-v1_5-with-sha3-224 => SHA3-224
id-rsassa-pkcs1-v1_5-with-sha3-256 => SHA3-256
id-rsassa-pkcs1-v1_5-with-sha3-384 => SHA3-384
id-rsassa-pkcs1-v1_5-with-sha3-512 => SHA3-512
SHA4
```

```
└─ $echo -n "hello" | openssl dgst -md5
MD5(stdin)= 5d41402abc4b2a76b9719d911017c592
(.python_env) └[admin@parrot]~]
```

Save Copy Collapse All Expand All Filter JSON

session_id: 476f3c2aedd2487b
woId: banana20

Parrot Terminal

```

{ KECCAK-KMAC-256, KECCAK-KMAC256 } @ default
NULL @ default
{ 1.2.840.113549.2.4, MD4 } @ legacy
{ 1.0.10118.3.0.55, whirlpool } @ legacy
{ 1.3.36.3.2.1, RIPEMD, RIPEMD-160, RIPEMD160, RMD160 } @ legacy
(.python_env) [admin@parrot] ~
└── $echo -n "hello" | openssl dgst -md5
MD5(stdin)= 5d41402abc4b2a76b9719d911017c592
(.python_env) [admin@parrot] ~
└── $curl -X POST http://127.0.0.1:10001/submit -d '{"session_id": "476f3c2aedd2487b", "hash_hex": "5d41402abc4b2a76b9719d911017c592"}' -H "Content-Type: application/json"
{"result": "Incorrect MD5 hash"}
(.python_env) [admin@parrot] ~
└── $echo -n "banana20" | openssl dgst -md5
MD5(stdin)= 0267a70c40f938981570853ecc47b648
(.python_env) [admin@parrot] ~
└── $curl -X POST http://127.0.0.1:10001/submit -d '{"session_id": "476f3c2aedd2487b", "hash_hex": "0267a70c40f938981570853ecc47b648"}' -H "Content-Type: application/json"
<!doctype html>
<html lang=en>
<title>400 Bad Request</title>
<h1>Bad Request</h1>
<p>The browser (or proxy) sent a request that this server could not understand.</p>
(.python_env) [admin@parrot] ~
└── $curl -X POST http://127.0.0.1:10001/submit -d '{"session_id": "476f3c2aedd2487b", "hash_hex": "0267a70c40f938981570853ecc47b648"}' -H "Content-Type: application/json"
{"result": "Correct MD5 hash!"}
(.python_env) [admin@parrot] ~
└── $

```

(.python_env) [admin@parrot] ~
└── \$echo -n "hello" | openssl dgst -md5
MD5(stdin)= 5d41402abc4b2a76b9719d911017c592
(.python_env) [admin@parrot] ~
└── \$curl -X POST http://127.0.0.1:10001/submit -d '{"session_id": "476f3c2aedd2487b", "hash_hex": "5d41402abc4b2a76b9719d911017c592"}' -H "Content-Type: application/json"
{"result": "Incorrect MD5 hash"}
(.python_env) [admin@parrot] ~
└── \$echo -n "banana20" | openssl dgst -md5
MD5(stdin)= 0267a70c40f938981570853ecc47b648
(.python_env) [admin@parrot] ~
└── \$curl -X POST http://127.0.0.1:10001/submit -d '{"session_id": "476f3c2aedd2487b", "hash_hex": "0267a70c40f938981570853ecc47b648"}' -H "Content-Type: application/json"
<!doctype html>
<html lang=en>
<title>400 Bad Request</title>
<h1>Bad Request</h1>
<p>The browser (or proxy) sent a request that this server could not understand.</p>
(.python_env) [admin@parrot] ~
└── \$curl -X POST http://127.0.0.1:10001/submit -d '{"session_id": "476f3c2aedd2487b", "hash_hex": "0267a70c40f938981570853ecc47b648"}' -H "Content-Type: application/json"
{"result": "Correct MD5 hash!"}

```
(.python_env) [admin@parrot]~
└─$ nano payload.json
(.python_env) [admin@parrot]~
└─$ cat payload.json
{"session_id": "476f3c2aedd2487b", "hash_hex": "0267a70c40f938981570853ecc47b648"}
(.python_env) [admin@parrot]~
└─$ curl -X POST http://127.0.0.1:10001/submit -H "Content-Type: application/json" --data @payload.json
{"result": "Correct MD5 hash!"}
(.python_env) [admin@parrot]~
└─$ curl -s -X GET http://127.0.0.1:10001/hash -o response.json
(.python_env) [admin@parrot]~
└─$ if=$(jq -r '.session_id' response.json)
(.python_env) [admin@parrot]~
└─$ id=$(jq -r '.session_id' response.json)
(.python_env) [admin@parrot]~
└─$ echo $id
bbe630bfffeaf299e
(.python_env) [admin@parrot]~
└─$ word=$(jq -r '.word' response.json)
(.python_env) [admin@parrot]~
└─$ echo $word
anhar
(.python_env) [admin@parrot]~
└─$ echo -n $word
anhar(.python_env) [admin@parrot]~
└─$ echo -n $word | openssl dgst -md5
MD5(stdin)= 68aacc5d77146a2398ca64c5c3c596f6
(.python_env) [admin@parrot]~
└─$ echo -n $word | openssl dgst -md5 | awk '{print $2}'
68aacc5d77146a2398ca64c5c3c596f6
(.python_env) [admin@parrot]~
└─$ sol=$(echo -n $word | openssl dgst -md5 | awk '{print $2}')
(.python_env) [admin@parrot]~
└─$ echo $sol
68aacc5d77146a2398ca64c5c3c596f6
(.python_env) [admin@parrot]~
└─$ curl -X POST http://127.0.0.1:10001/submit -H "Content-Type: application/json" -d '{"session_id": "$id", "hash_hex": "$sol"}'
{"result": "Invalid session ID"}
(.python_env) [admin@parrot]~
└─$
```

Nie działa bo jest w hashu i id znak końca linii (spróbować pozbyć się ich) 45:00

```
$ podman run -dp 10002:10002 --name hashingsha256ex1 docker.io/mazurkatarzyna/hashing-sha256-ex1:latest
Trying to pull docker.io/mazurkatarzyna/hashing-sha256-ex1:latest...
Getting image source signatures
Copying blob 4639fa84df42 done
Copying blob 92d63ec5cbeb skipped: already exists
Copying blob 457ede85efe6 skipped: already exists
Copying blob 0ebcc011f0ec skipped: already exists
Copying blob 64b78282ca88 skipped: already exists
Copying blob 45b848213d5c done
Copying blob 3da95a905ed5 skipped: already exists
Copying blob e9479c062bd6 done
Copying blob 89d165e47fe8 done
Copying config a300e15771 done
Writing manifest to image destination
Storing signatures
aed0773f19a8aee6880220d697b6aa0d0a18f0480762bdc5d4b4a1300186ac38
(.python_env) └─[admin@parrot]─[~]
└─ $
```

1.2 Pod adresem <http://127.0.0.1:10002> działa prosty serwer HTTP udostępniający dwa endpointy: <http://127.0.0.1:10002/hash> oraz <http://127.0.0.1:10002/submit>.

Twoim zadaniem jest obliczenie skrótu SHA-256 podanego przez serwer losowego słowa. Wynikowy hash powinien być w formacie szesnastkowym (hex). Aby otrzymać od serwera słowo do zahashowania oraz identyfikator sesji, wyslij zapytanie HTTP GET do endpointa <http://127.0.0.1:10002/hash>. Otrzymasz w odpowiedzi unikalny identyfikator sesji oraz losowe słowo do zahashowania.

Następnie powinieneś obliczyć skrót SHA-256 otrzymanego słowa, kodując wynik w formacie szesnastkowym (hex).

Po obliczeniu hash'a wyslij go do serwera metodą HTTP POST na endpoint <http://127.0.0.1:10002/submit>, przekazując w formacie JSON zarówno identyfikator sesji, jak i obliczony skrót MD5 (hex). Serwer zweryfikuje poprawność przesłanego hasha i zwróci informację o sukcesie lub błędzie.

- (a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 10002:10002 mazurkatarzyna/hashing-sha256-ex1:latest
```

- (b) Do obliczenia skrótu SHA-256 użyj narzędzia OpenSSL.
(c) Aby wysłać odpowiedź do serwera, użyj narzędzia cURL. Pamiętaj o dołączeniu nagłówka HTTP "Content-Type: application/json".

```
(.python_env) [admin@parrot]~
└─ $curl -X GET http://127.0.0.1:10002/hash
{"session_id":"a6c77fc025f9f7ad","word":"mary3mary"}
(.python_env) [admin@parrot]~
└─ $echo -n "mary3mary" | openssl dgst -SHA-256
SHA2-256(stdin)= d50f1652146876f2b5303e36d5d54f29dd4e74e7b046adf242653807eda62107
(.python_env) [admin@parrot]~
└─ $curl -X POST http://127.0.0.1:10002/submit -d '{"session_id":"a6c77fc025f9f7ad", "hash_hex":"d50f1652146876f2b5303e36d5d54f29dd4e74e7b046adf242653807eda62107"}' -H "Content-Type: application/json"
{"result":"Correct SHA-256 hash!"}
(.python_env) [admin@parrot]~
```

- 1.3 Pod adresem <http://127.0.0.1:10003> działa prosty serwer HTTP udostępniający dwa endpointy: <http://127.0.0.1:10003/hash> oraz <http://127.0.0.1:10003/submit>.

Twoim zadaniem jest obliczenie skrótu SHA-512 podanego przez serwer losowego słowa. Wynikowy hash powinien być w formacie szesnastkowym (hex). Aby otrzymać od serwera słowo do zahashowania oraz identyfikator sesji, wyślij zapytanie HTTP GET do endpointa <http://127.0.0.1:10003/hash>. Otrzymasz w odpowiedzi unikalny identyfikator sesji oraz losowe słowo do zahashowania.

Następnie powinieneś obliczyć skrót SHA-512 otrzymanego słowa, kodując wynik w formacie szesnastkowym (hex).

Po obliczeniu hash'a wyślij go do serwera metodą HTTP POST na endpoint <http://127.0.0.1:10003/submit>, przekazując w formacie JSON zarówno identyfikator sesji, jak i obliczony skrót SHA-512 (hex). Serwer zweryfikuje poprawność przesłanego hasha i wróci informację o sukcesie lub błędzie.

- (a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 10003:10003 mazurkatarzyna/hashing-sha-512-ex1:latest
```

- (b) Do obliczenia skrótu SHA-512 użyj narzędzia OpenSSL.

- (c) Aby wysłać odpowiedź do serwera, użyj narzędzia cURL. Pamiętaj o dołączeniu nagłówka HTTP "Content-Type: application/json".

```
└─ $podman run -dp 10003:10003 --name hashingsha512ex1 docker.io/mazurkatarzyna/hashing-sha-512-ex1:latest
Trying to pull docker.io/mazurkatarzyna/hashing-sha-512-ex1:latest...
Getting image source signatures
Copying blob 2a840ba0f58b done
Copying blob 92d63ec5cbeb skipped: already exists
Copying blob 0ebcc011f0ec skipped: already exists
Copying blob 64b78282ca88 skipped: already exists
Copying blob 457ede85efe6 skipped: already exists
Copying blob f4ef97ffc00b done
Copying blob 042bbfa4b12b done
Copying blob 9515231cffc8 done
Copying blob 3da95a905ed5 skipped: already exists
Copying config 62feb974f9 done
Writing manifest to image destination
Storing signatures
a17aedb1402edb39f5855aabcb0b154956a13e914ea8fe0764989c37b5ba9f113
(.python_env) [admin@parrot]~
```

1.4 Pod adresem `http://127.0.0.1:10004` działa prosty serwer HTTP udostępniający dwa endpointy: `http://127.0.0.1:10004/hash` oraz `http://127.0.0.1:10004/submit`.

Twoim zadaniem jest obliczenie skrótu **Argon** podanego przez serwer losowego słowa. Wynikowy hash powinien być w formacie szesnastkowym (hex). Aby otrzymać od serwera słowo do zahashowania, parametry użyte do haszowania oraz identyfikator sesji, wyslij zapytanie HTTP GET do endpointa `http://127.0.0.1:10004/hash`. Otrzymasz w odpowiedzi unikalny identyfikator sesji, losowe słowo do zahashowania i niezbędne parametry.

Następnie powinieneś obliczyć skrót **Argon** otrzymanego słowa, kodując wynik w formacie szesnastkowym (hex).

Po obliczeniu hash'a wyslij go do serwera metodą HTTP POST na endpoint `http://127.0.0.1:10004/submit`, przekazując w formacie JSON zarówno identyfikator sesji, jak i obliczony skrót **Argon** (hex). Serwer zweryfkuje poprawność przesłanego hasha i zwróci informację o sukcesie lub błędzie.

- (a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 10004:10004 mazurkatarzyna/hashing-argon-ex1:latest
```

- (b) Do obliczenia skrótu Argon narzędzia OpenSSL dostępnego jako kontener Dockerowy. W tym celu uruchom kontener, i wejdź do jego środka. Haszowanie za pomocą algorytmu Argon dostępne jest pod nazwą `kdf`.

```
docker run -it mazurkatarzyna/openssl-332-ubuntu:latest
openssl kdf -help
```

- (c) Aby wysłać odpowiedź do serwera, użyj narzędzia cURL. Pamiętaj o dołączeniu nagłówka HTTP "Content-Type: application/json".

```
$podman run -dp 10004:10004 --name hashingargonex1 docker.io/mazurkatarzyna/hashing-argon-ex1:latest
Trying to pull docker.io/mazurkatarzyna/hashing-argon-ex1:latest...
Getting image source signatures
Copying blob 03d759b6e759 done
Copying blob 40225ac59a78 done
Copying blob 3ed75cb01584 done
Copying blob e735f3a6b701 done
Copying blob 431e32a943f2 done
Copying blob 88a74f644769 done
Copying blob 071520bfe1ac done
Copying blob ac7b2aff7fc4 done
Copying blob fc04d2c8f0d3 done
Copying blob 3ec45aa4aab done
Copying blob 67333c556853 done
Copying blob b363bc0d9ca4 done
Copying blob 7e0b4f91e0f6 done
Copying blob 4d598ae73b77 done
Copying config a407c1ba33 done
Writing manifest to image destination
Storing signatures
1b04fe04f2082f751fd8b4538b43882df9aff0d5a7871679e9d989de30a9e698
```

```
[root@153e86f925f3: /app] $ curl -X GET http://127.0.0.1:10004/hash
{
  "kdf_options": {
    "iter": 1,
    "keylen": 24,
    "memcost": 8192,
    "salt": "NaCl2024"
  },
  "session_id": "3ce4f9363d8fb42c",
  "word": "wypelznieci"
}
```

```
(.python_env) [x]-[admin@parrot]-[~]
└─ $ podman run -it docker.io/mazurkatarzyna/openssl-332-ubuntu:latest
Trying to pull docker.io/mazurkatarzyna/openssl-332-ubuntu:latest...
Getting image source signatures
Copying blob f24d33ed7339 done
Copying blob 7d4bbb6a9eae done
Copying blob 1d387567261e done
Copying blob 038e7ff2d571 done
Copying blob 3f34f8369f71 done
Copying blob f186222ac064 done
Copying blob 965acd2b4f69 done
Copying blob f6d24776cde2 done
Copying blob e610e957fc29 done
Copying blob 03b997b92b54 done
Copying blob b72ad92f63fc done
Copying blob a68e8327e7dc done
Copying blob cfcf5b07973c done
Copying blob 0ac11a9723c2 done
Copying blob 40225ac59a78 skipped: already exists
Copying blob 7dc94baab4d3 done
Copying blob eebae4277690 done
Copying blob 4f4fb700ef54 done
Copying blob c695ceb5f249 done
Copying blob 438765a91c7f done
Copying config 2ab1cd8112 done
Writing manifest to image destination
Storing signatures
root@153e86f925f3:/app# openssl kdf -keylen 24 -kdfopt pass:wypelznieci -kdfopt salt:NaCl2024
-kdfopt iter:1 -kdfopt memcost:8192 ARGON2D | tr -d ':' | tr '[[:upper]]' '[[:lower]]'
3F842B4F2A7EE1AF117531840DCB3E3ABCA1665A9C405C2A
```

exit żeby wyjść kontenera

```
[root@153e86f925f3: /app] $ curl -X POST http://127.0.0.1:10004/submit -d '{"session_id": "3ce4f9363d8fb42c", "hash_hex": "3F842B4F2A7EE1AF117531840DCB3E3ABCA1665A9C405C2A"}' -H "Content-Type: application/json"
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>
```

1.5 Pod adresem `http://127.0.0.1:10005` działa prosty serwer HTTP udostępniający dwa endpointy: `http://127.0.0.1:10005/hash` oraz `http://127.0.0.1:10005/submit`.

Twoim zadaniem jest obliczenie skrótu `bcrypt` podanego przez serwer losowego słowa. Wynikowy hash powinienny być w formacie szesnastkowym (hex). Aby otrzymać od serwera słowo do zahashowania oraz identyfikator sesji, wyslij zapytanie HTTP `GET` do endpointa `http://127.0.0.1:10005/hash`. Otrzymasz w odpowiedzi unikalny identyfikator sesji i losowe słowo do zahashowania.

Następnie powinieneś obliczyć skrót `brypt` otrzymanego słowa, kodując wynik w formacie szesnastkowym (hex).

Po obliczeniu hash'a wyslij go do serwera metodą HTTP `POST` na endpoint `http://127.0.0.1:10005/submit`, przekazując w formacie JSON zarówno identyfikator sesji, jak i obliczony skrót `brypt` (hex). Serwer zweryfikuje poprawność przesłanego hasha i zwróci informację o sukcesie lub błędzie.

- (a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 10005:10005 mazurkatarzyna/hashing-bcrypt-ex1:latest
```

- (b) Do obliczenia skrótu `brypt` użyj narzędzia `htpasswd` dostępnego jako kontener Dockerowy.

```
docker run mazurkatarzyna/htpasswd:latest --help
```

- (c) Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dodaniu nagłówka HTTP "Content-Type: application/json".

```
$ podman run -dp 10005:10005 --name hashingbcryptex1 docker.io/mazurkatarzyna/hashing-bcrypt-ex1:latest
Trying to pull docker.io/mazurkatarzyna/hashing-bcrypt-ex1:latest...
Getting image source signatures
Copying blob ef25d0cf19e1 done
Copying blob ee8ba591bc6f done
Copying blob abcec076ac35 done
Copying blob edf70a019d4a done
Copying blob 1d387567261e skipped: already exists
Copying blob b16fa1b36b3f done
Copying blob 7d2be8a47718 done
Copying blob 5c2bf2837ad6 done
Copying config 72ee7c0c3a done
Writing manifest to image destination
Storing signatures
fc07e2b067a6c2b598f00ecb50107094d9c835f76584fe430603ed9036e6b3d1
(.python_env) └─[admin@parrot]─[~]
```

1.10 Mając dany początkowy ciąg znaków `helloworld`, który następnie został zahashowany, określ, jaka funkcja skrótu została wykorzystana do utworzenia hasha: `6adfb183a4a2c94a2f92dab5ade762a47889a5a1`. Możesz wykorzystać narzędzie `hash-identifier`:

```
docker run -it mazurkatarzyna/hash-identifier:latest
```

Czy `hash-identifier` odnalazł nazwę hasha? Jeśli nie, zaproponuj inny sposób na rozwiązanie zadania.

- 1.11 Mając dany początkowy ciąg znaków `helloworld`, który następnie został zahashowany, określ, jaka funkcja skrótu została wykorzystana do utworzenia hasha:
`$2y$10$xbyAv5a46CQYPay5UISCNefWpVdx2qvhCBEOZ/YtfxoVXh0GrVKqa`. Możesz wykorzystać narzędzie `hash-identifier`:

```
docker run -it mazurkatarzyna/hash-identifier:latest
```

Czy **hash-identifier** odnalazł nazwę hasha? Jeśli nie, zaproponuj inny sposób na rozwiązanie zadania.

- 1.12** Mając dany początkowy ciąg znaków `helloworld`, który następnie został zahashowany, określ, jaka funkcja skrótu została wykorzystana do utworzenia hasha: `6adfb183a4a2c94a2f92dab5ade762a47889a5a1`. Możesz wykorzystać narzędzie `hashid`:

```
docker run mazurkatarzyna/hashid:latest yourhash
```

Czy hashid odnalazł nazwę hasha? Jeśli nie, zaproponuj inny sposób na rozwiązanie zadania.