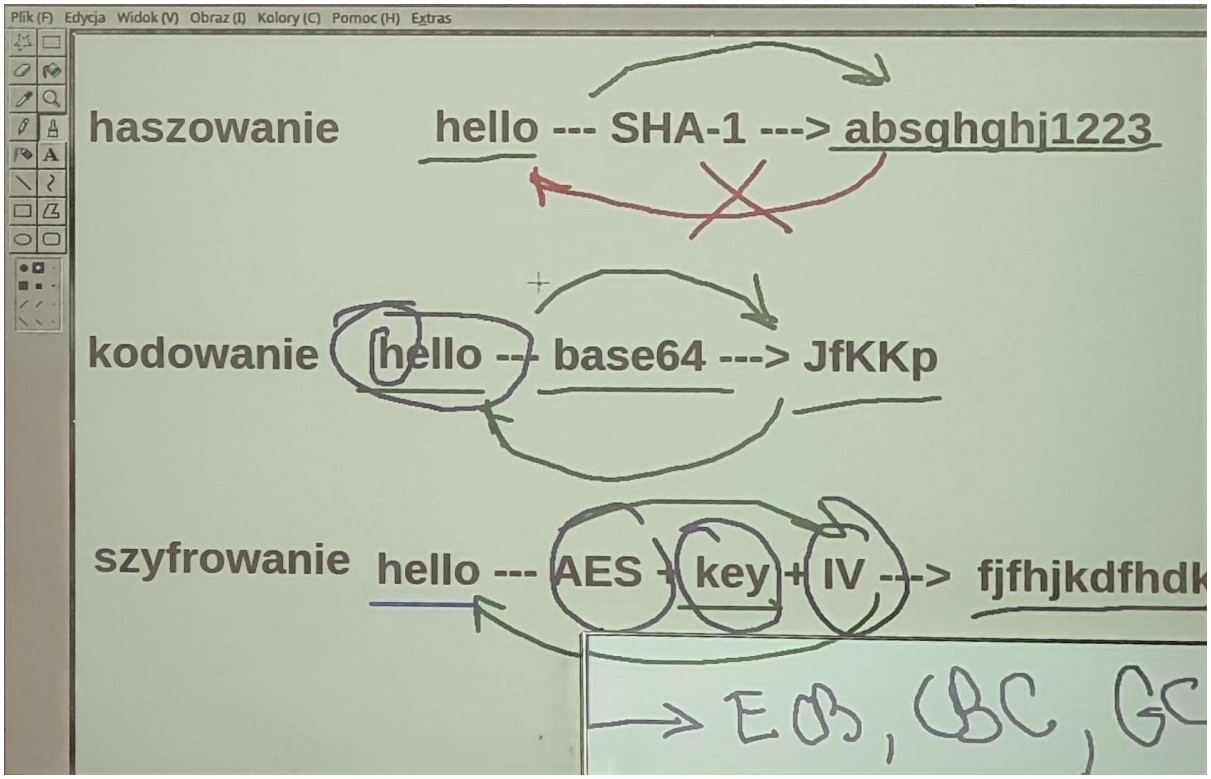


2.1 *Pod adresem* `http://127.0.0.1:2001` *działa prosty serwer HTTP udostępniający 2 endpointy:* `http://127.0.0.1:2001/encrypt` *oraz* `http://127.0.0.1:2001/submit`.

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 2001:2001 --name ex1 docker.io/mazurkatarzyna/symmetric-enc-ex1:latest
podman run -p 2001:2001 --name ex1 docker.io/mazurkatarzyna/symmetric-enc-ex1:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:2001/encrypt` używając meody HTTP `GET`. Otrzymasz w odpowiedzi unikalny identyfikator sesji (`session_id`), losowe słowo do zaszyfrowania (`word`) oraz klucz szyfrujący podany jako 32-bajtowy ciąg szesnastkowy (`key_hex`).

(c) Zaszyfruj słowo odebrane od serwera użyciu algorytmu `AES-256` w trybie `ECB` wykorzystując odebrany od serwera klucz szyfrujący.

(d) Po wykonaniu szyfrowania, zaszyfrowany ciąg znaków zakoduj w formacie `base64`.

(e) Po uzyskaniu zaszyfrowanego i zakodowanego wyniku wyślij go do serwera poprzez endpoint `http://127.0.0.1:2001/submit` używając metody HTTP `POST`, w którym przekażesz zarówno identyfikator sesji (jako `session_id`), jak i zaszyfrowany ciąg w formacie `base64` (jako `encrypted_b64`). Serwer w odpowiedzi zwróci odpowiednią informację o sukcesie lub błędzie.

UWAGI:

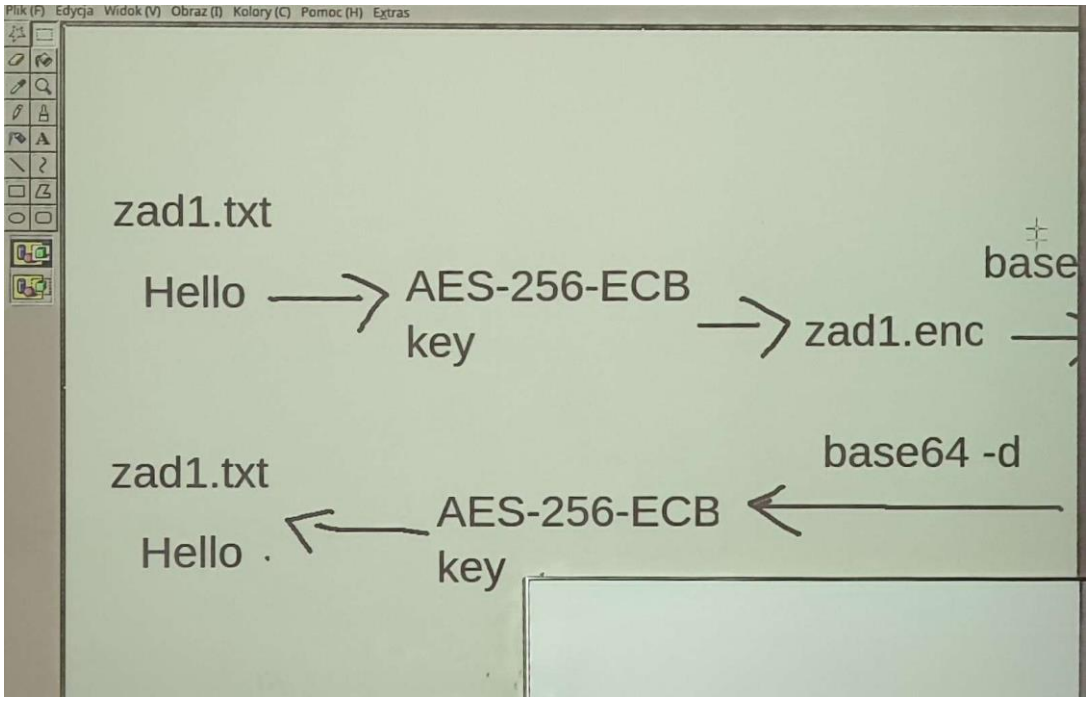
- Aby zaszyfrować wylosowane przez serwer słowo, wykorzystaj narzędzie `OpenSSL`.
- Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dodaniu nagłówka protokołu HTTP `Content-Type: application/json`.

Od c)

```
(.python_env) ┌─[✗]─[admin@parrot]─[~]
└──╼ $echo -n "Hello" > zad1.txt
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad1.txt
Hello(.python_env) ┌─[admin@parrot]─[~]
```

```
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl rand -hex 32 > key
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat key
747edda744d8cbb28dfa724cc6cdd871a7eedfab4b46fdf5cf57890e6c74cbfc
```

```
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl enc -aes-256-ecb -in zad1.txt -K $(cat key) -out zad1.enc
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad1.enc
t4xb
     mX
└──╼ $openssl enc -aes-256-ecb -in zad1.txt -K $(cat key) -out zad1.enc -base64
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad1.enc
xaU0eGLyDOmvbdClpVp3BA==
```



```
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl enc -d -aes-256-ecb -in zad1.enc -out zad1.dec -K $(cat key) -base64
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad1.dec
Hello(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad1.txt
```

2.3 *Pod adresem* `http://127.0.0.1:2003` *działa prosty serwer HTTP udostępniający 2 endpointy:* `http://127.0.0.1:2003/encrypt` *oraz* `http://127.0.0.1:2003/submit`.

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 2003:2003 --name ex3 docker.io/mazurkatarzyna/symmetric-enc-ex3:latest
podman run -p 2003:2003 --name ex3 docker.io/mazurkatarzyna/symmetric-enc-ex3:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:2003/encrypt` używając meody HTTP `GET`. Otrzymasz w odpowiedzi unikalny identyfikator sesji (`session_id`), losowe słowo do zaszyfrowania (`word`) oraz klucz szyfrujący podany jako 32-bajtowy ciąg szesnastkowy (`key_hex`).

(c) Zaszyfruj otrzymane od serwera słowo, stosując algorytm `CAMELLIA-128` w trybie `ECB` wykorzystując odebrany od serwera klucz szyfrujący.

(d) Po wykonaniu szyfrowania, zaszyfrowany ciąg znaków zakoduj w formacie `base64`.

(e) Po uzyskaniu zaszyfrowanego i zakodowanego wyniku wyślij go do serwera poprzez endpoint `http://127.0.0.1:2003/submit` używając metody HTTP `POST`, w którym przekażesz zarówno identyfikator sesji (jako `session_id`), jak i zaszyfrowany ciąg w formacie `base64` (jako `encrypted_b64`). Serwer w odpowiedzi zwróci odpowiednią informację o sukcesie lub błędzie.

UWAGI:

- Aby zaszyfrować wylosowane przez serwer słowo, wykorzystaj narzędzie `OpenSSL`.
- Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dodaniu nagłówka protokołu HTTP `Content-Type: application/json`.

```
Hello(.python_env) ┌─[admin@parrot]─[~]
└──╼ $echo -n "slowo" > zad3.txt
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad3.txt
slowo(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl rand -hex 32 > key3
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat key3
da3a812d999c4072d5632318791990b12a60820721adef9127e15af00d1d432f
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl enc -camellia-128-ecb -in zad3.txt -K $(cat key3) -out zad3.enc
hex string is too long, ignoring excess
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl rand -hex 16 > key3
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl enc -camellia-128-ecb -in zad3.txt -K $(cat key3) -out zad3.enc
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad3.enc
-[illegible]}(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl enc -camellia-128-ecb -in zad3.txt -K $(cat key3) -out zad3.enc -base64
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad3.enc
LZ2Dmv7JXPICoYfahp42fQ==
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl enc -camellia-128-ecb -in zad3.txt -K $(cat key3) -out zad3.enc -a
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad3.enc
LZ2Dmv7JXPICoYfahp42fQ==
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl enc -d -camellia-128-ecb -in zad3.enc -out zad3.dec -K $(cat key3) -a
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad3.dec
slowo(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad3.txt
slowo(.python_env) ┌─[admin@parrot]─[~]
```

```
(.python_env) ┌─[✗]─[admin@parrot]─[~]
└──╼ $openssl list -cipher-commands
aes-128-cbc       aes-128-ecb       aes-192-cbc       aes-192-ecb
aes-256-cbc       aes-256-ecb       aria-128-cbc      aria-128-cfb
aria-128-cfb1     aria-128-cfb8     aria-128-ctr      aria-128-ecb
aria-128-ofb      aria-192-cbc      aria-192-cfb      aria-192-cfb1
aria-192-cfb8     aria-192-ctr      aria-192-ecb      aria-192-ofb
aria-256-cbc      aria-256-cfb      aria-256-cfb1     aria-256-cfb8
aria-256-ctr      aria-256-ecb      aria-256-ofb      bf
bf-cbc            bf-cfb            bf-ecb            bf-ofb
camellia-128-cbc  camellia-128-ecb  camellia-192-cbc  camellia-192-ecb
camellia-256-cbc  camellia-256-ecb  cast              cast-cbc
cast5-cbc         cast5-cfb         cast5-ecb         cast5-ofb
des               des-cbc           des-cfb           des-ecb
des-ede           des-ede-cbc       des-ede-cfb       des-ede-ofb
des-ede3          des-ede3-cbc      des-ede3-cfb      des-ede3-ofb
des-ofb           des3              desx              rc2
rc2-40-cbc        rc2-64-cbc        rc2-cbc           rc2-cfb
rc2-ecb           rc2-ofb           rc4               rc4-40
seed              seed-cbc          seed-cfb          seed-ecb
seed-ofb          sm4-cbc           sm4-cfb           sm4-ctr
sm4-ecb           sm4-ofb
```

2.2 *Pod adresem* `http://127.0.0.1:2002` *działa prosty serwer HTTP udostępniający 2 endpointy:* `http://127.0.0.1:2002/decrypt` *oraz* `http://127.0.0.1:2002/submit`.

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 2002:2002 --name ex2 docker.io/mazurkatarzyna/symmetric-enc-ex2:latest
podman run -p 2002:2002 --name ex2 docker.io/mazurkatarzyna/symmetric-enc-ex2:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:2002/decrypt` używając meody HTTP `GET`. Otrzymasz w odpowiedzi unikalny identyfikator sesji (`session_id`), zakodowane i zaszyfrowane słowo (`encrypted_b64`) oraz klucz deszyfrujący podany jako 32-bajtowy ciąg szesnastkowy (`key_hex`).

(c) Odkoduj otrzymane słowo, stosując kodowanie `base64`. Następnie odszyfruj odkodowane słowo algorytmem `AES-256` w trybie `ECB` z użyciem podanego klucza deszyfrującego.

(d) Po uzyskaniu odszyfrowanego i odkodowanego wyniku wyślij go do serwera poprzez endpoint `http://127.0.0.1:2002/submit` używając metody HTTP `POST`, w którym przekażesz zarówno identyfikator sesji (jako `session_id`), jak i odkodowane i odszyfrowane słowo (jako `decrypted_word`). Serwer w odpowiedzi zwróci odpowiednią informację o sukcesie lub błędzie.

UWAGI:

- Aby odszyfrować zaszyfrowane przez serwer słowo, wykorzystaj narzędzie `OpenSSL`.
- Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dodaniu nagłówka protokołu HTTP `Content-Type: application/json`.

```
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $nano key2
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $nano zad2.enc
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl enc -d -aes-256-ecb -in zad2.enc -out zad2.dec -K $(cat key2) -a
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad2.dec
UMCS(.python_env) ┌─[admin@parrot]─[~]
```

2.5 *Pod adresem* `http://127.0.0.1:2005` *działa prosty serwer HTTP udostępniający 2 endpointy:* `http://127.0.0.1:2005/encrypt` *oraz* `http://127.0.0.1:2005/submit`.

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 2005:2005 --name ex5 docker.io/mazurkatarzyna/symmetric-enc-ex5:latest
podman run -p 2005:2005 --name ex5 docker.io/mazurkatarzyna/symmetric-enc-ex5:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:2005/encrypt` używając meody HTTP `GET`. Otrzymasz w odpowiedzi unikalny identyfikator sesji (`session_id`) oraz losowe słowo do zaszyfrowania (`word`).

(c) Wygeneruj klucz szyfrujący (klucz musi mieć 192 bity).

(d) Zaszyfruj otrzymane od serwera słowo, stosując algorytm `ARIA-192` w trybie `ECB` wykorzystując wygenerowany przez Ciebie (w punkcie **2.5**c) klucz szyfrujący.

(e) Po wykonaniu szyfrowania zaszyfrowany ciąg znaków zakoduj w formacie `base64`.

(f) Po uzyskaniu zaszyfrowanego i zakodowanego wyniku wyślij go do serwera poprzez endpoint `http://127.0.0.1:2005/submit` używając metody HTTP `POST`, w którym przekażesz: identyfikator sesji (`session_id`), zaszyfrowany ciąg w formacie `base64` (`encrypted_b64`), wygenerowany przez Ciebie klucz szyfrujący w formacie szesnastkowym (`key_hex`, 192 bity). Serwer w odpowiedzi zwróci odpowiednią informację o sukcesie lub błędzie.

UWAGI:

- Aby zaszyfrować wylosowane przez serwer słowo i wygenerować klucz szyfrujący wykorzystaj narzędzie `OpenSSL`.
- Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dodaniu nagłówka protokołu HTTP `Content-Type: application/json`.

```
UMCS(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl rand -hex 24 > key5
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat 5
cat: 5: Nie ma takiego pliku ani katalogu
(.python_env) ┌─[✗]─[admin@parrot]─[~]
└──╼ $cat key5
bf1c73dd6124df73e7c34eec46d5cef6c6efb7ff2e696d26
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $echo -n "zdanie5" > zad5.txt
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl end -aria-192-ecb -in zad5.txt -out zad5.enc -K $(cat key5) -base64
Invalid command 'end'; type "help" for a list.
(.python_env) ┌─[✗]─[admin@parrot]─[~]
└──╼ $openssl enc -aria-192-ecb -in zad5.txt -out zad5.enc -K $(cat key5) -base64
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad5.enc
vEFaOZ172DhCGCbUAhXWwQ==
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl enc -d -aria-192-ecb -in zad5.enc -out zad5.dec -K $(cat key5) -base64
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad5.dec
zdanie5(.python_env) ┌─[admin@parrot]─[~]
```

2.6 *Pod adresem* `http://127.0.0.1:2006` *działa prosty serwer HTTP udostępniający 2 endpointy:* `http://127.0.0.1:2006/encrypt` *oraz* `http://127.0.0.1:2006/submit`.

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 2006:2006 --name ex6 docker.io/mazurkatarzyna/symmetric-enc-ex6:latest
podman run -p 2006:2006 --name ex6 docker.io/mazurkatarzyna/symmetric-enc-ex6:latest
```

(b) Wyślij żądanie HTTP `GET` do endpointa `http://127.0.0.1:2006/encrypt`. Otrzymasz w odpowiedzi unikalny identyfikator sesji (`session_id`) oraz losowe słowo do zaszyfrowania (`word`).

(c) Wygeneruj klucz szyfrujący (klucz musi mieć 128 bitów).

(d) Wygeneruj wektor inicjalizacyjny (IV) (klucz musi mieć 128 bitów).

(e) Zaszyfruj pobrane od serwera słowo rzy użyciu algorytmu `AES-128` w trybie `CBC`, z kluczem szyfrującym oraz wektorem inicjalizującym (IV) wygenerowanymi przez Ciebie (w punktach **2.6**c i **2.6**d).

(f) Po wykonaniu szyfrowania zakoduj zaszyfrowany ciąg do formatu `base64`.

(g) Po uzyskaniu zaszyfrowanego i zakodowanego wyniku wyślij go do serwera poprzez endpoint `http://127.0.0.1:2006/submit` używając metody HTTP `POST`, przekazując następujące dane: identyfikator sesji (`session_id`), zaszyfrowany ciąg w formacie `base64` (`encrypted_b64`), klucz szyfrujący w formacie szesnastkowym (`key_hex`, 128 bitów) oraz IV w formacie szesnastkowym (`iv_hex`, 128 bitów). Serwer w odpowiedzi zwróci odpowiednią informację o sukcesie lub błędzie.

UWAGI:

- Aby zaszyfrować wylosowane przez serwer słowo, wygenerować klucz szyfrujący oraz IV, wykorzystaj narzędzie `OpenSSL`.
- Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dodaniu nagłówka protokołu HTTP `Content-Type: application/json`.

```
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $echo -n "word" > zad6.txt
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl rand -hex 16 > key6
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl rand -hex 16 > IV
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl enc -aes-128-cbc -in zad6.txt -out zad6.enc -K $(cat key6) -iv $(cat IV) -nosalt -base64
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad6.enc
E8T3H6N5ZKWUZULKP+BV2Q==
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl enc -d -aes-128-cbc -in zad6.enc -out zad6.dec -K $(cat key6) -iv $(cat IV) -nosalt -base64
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad6.dec
word(.python_env) ┌─[admin@parrot]─[~]
└──╼ $
```

2.7 *Pod adresem* `http://127.0.0.1:2007` *działa prosty serwer HTTP udostępniający 2 endpointy:* `http://127.0.0.1:2007/decrypt` *oraz* `http://127.0.0.1:2007/submit`.

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 2007:2007 --name ex7 docker.io/mazurkatarzyna/symmetric-enc-ex7:latest
podman run -p 2007:2007 --name ex7 docker.io/mazurkatarzyna/symmetric-enc-ex7:latest
```

(b) Wyślij żądanie HTTP `GET` do endpointa `http://127.0.0.1:2007/decrypt`. W odpowiedzi otrzymasz identyfikator sesji (`session_id`), zakodowany w formacie `base64` ciąg zaszyfrowanego tekstu (`encrypted_b64`), użyte hasło (`password`) oraz IV w formacie szesnastkowym (`iv_hex`). Serwer zaszyfrował losowo wybrane słowo przy użyciu algorytmu `AES-256-CBC`, z kluczem wygenerowanym na podstawie hasła przy użyciu funkcji `PBKDF2`, oraz losowego wektora inicjalizującego (IV).

(c) Odkoduj otrzymane słowo, stosując kodowanie `base64`. Następnie odszyfruj odkodowane słowo algorytmem `AES-256-CBC`, z użyciem kluczem wygenerowanym na podstawie hasła przy użyciu funkcji `PBKDF2`, oraz losowego wektora inicjalizującego (IV).

(d) Pamiętaj, że do szyfrowania NIE użyto soli, więc w deszyfrowaniu również NIE JEST potrzebna.

(e) Wyślij do serwera odpowiedź poprzez endpoint `http://127.0.0.1:2007/submit`, używając metody HTTP `POST`, przekazując dane w formacie `JSON`: identyfikator sesji (`session_id`) oraz odszyfrowane słowo (`decrypted_word`). Serwer w odpowiedzi zwróci odpowiednią informację o sukcesie lub błędzie.

UWAGI:

- Aby odszyfrować zaszyfrowane przez serwer słowo, wykorzystaj narzędzie `OpenSSL`.
- Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dodaniu nagłówka protokołu HTTP `Content-Type: application/json`.

```
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $nano IV
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $nano haslo
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl enc -d aes-256-cbc ^C
(.python_env) ┌─[✗]─[admin@parrot]─[~]
└──╼ $openssl enc -help
Usage: enc [options]

General options:
 -help               Display this summary
 -list               List ciphers
 -ciphers            Alias for -list
 -e                  Encrypt
 -d                  Decrypt
 -p                  Print the iv/key
 -P                  Print the iv/key and exit
 -engine val         Use engine, possibly a hardware device

Input options:
 -in infile          Input file
 -k val              Passphrase
 -kfile infile       Read passphrase from file

Output options:
 -out outfile        Output file
 -pass val           Passphrase source
 -v                  Verbose output
 -a                  Base64 encode/decode, depending on encryption flag
 -base64             Same as option -a
 -A                  Used with -[base64|a] to specify base64 buffer as a single line
```

```
Encryption options:
 -nopad              Disable standard block padding
 -salt               Use salt in the KDF (default)
 -nosalt             Do not use salt in the KDF
 -debug              Print debug info
 -bufsize val        Buffer size
 -K val              Raw key, in hex
 -S val              Salt, in hex
 -iv val             IV in hex
 -md val             Use specified digest to create a key from the passphrase
 -iter +int          Specify the iteration count and force the use of PBKDF2
                     Default: 10000
 -pbkdf2             Use password-based key derivation function 2 (PBKDF2)
                     Use -iter to change the iteration count from 10000
 -none               Don't encrypt
 -*                  Any supported cipher

Random state options:
 -rand val           Load the given file(s) into the random number generator
 -writerand outfile  Write random data to the specified file

Provider options:
 -provider-path val  Provider load path (must be before 'provider' argument if required)
 -provider val       Provider to load (can be specified multiple times)
 -propquery val      Property query used when fetching algorithms
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl enc -d -aes-256-cbc -pbkdf2 pass:$(cat haslo) -in zad7.enc -out zad7.dec -base64 -iv $(cat
IV) -nosalt
enc: Use -help for summary.
(.python_env) ┌─[✗]─[admin@parrot]─[~]
└──╼ $openssl enc -d -aes-256-cbc -pbkdf2 -pass pass:$(cat haslo) -in zad7.enc -out zad7.dec -base64 -iv
$(cat IV) -nosalt
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad7.dec
Crypto(.python_env) ┌─[admin@parrot]─[~]
└──╼ $
```