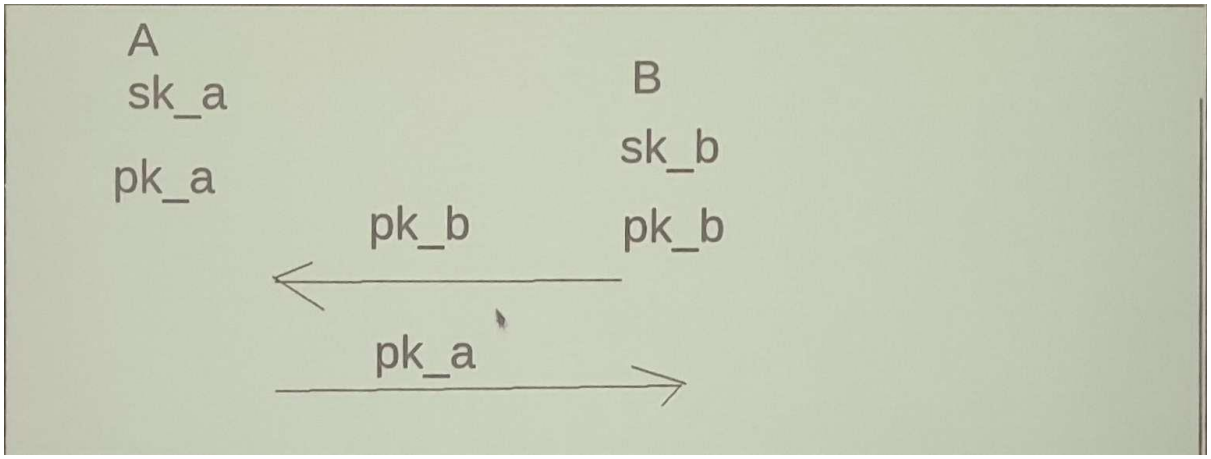
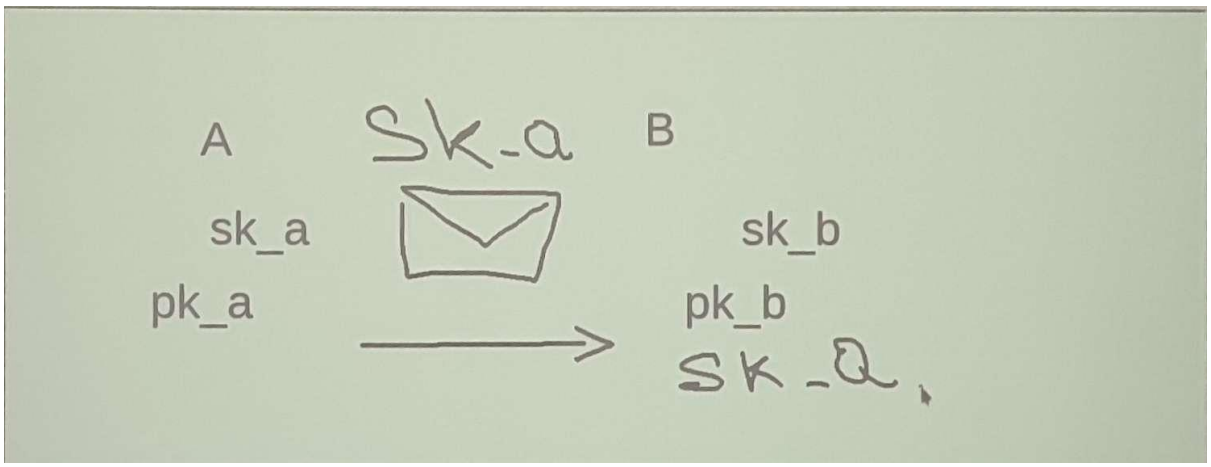




3.1 *Pod adresem* `http://127.0.0.1:3001` *działa prosty serwer HTTP udostępniający dwa endpointy:* `http://127.0.0.1:3001/pubkey` *oraz* `http://127.0.0.1:3001/privkey`.

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 3001:3001 --name ex1 docker.io/mazurkatarzyna/asymmetric-enc-ex1:latest
podman run -p 3001:3001 --name ex1 docker.io/mazurkatarzyna/asymmetric-enc-ex1:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 3001:3001 --name ex1 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex1:latest
podman run -p 3001:3001 --name ex1 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex1:latest
```

(b) Wygeneruj parę kluczy RSA (publiczny i prywatny). Wyeksportuj oba klucze do plików.

(c) Używając metody HTTP `POST`, wyślij parę kluczy RSA do serwera poprzez endpointy `http://127.0.0.1:3001/upload/public` (jako `file`) oraz `http://127.0.0.1:3001/upload/private` (również jako `file`). W odpowiedzi serwer zwróci odpowiednią informację (szczegóły klucza).

UWAGI:

- Aby wygenerować parę kluczy RSA, wykorzystaj narzędzie `OpenSSL`. Użyj argumentu `genpkey` oraz `pkey`.
- Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dodaniu nagłówków protokołu HTTP: `Content-Type: application/json`, `Content-Type: multipart/form-data`.

```
(.python_env) ┌─[✗]─[admin@parrot]─[~]
└──╼ $openssl genpkey -algorithm RSA -out priv.pem -pkeyopt rsa_keygen_bits:1024
......+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
..............+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $ls
Desktop    juice-shop  key5         Obrazy     snap            zad1.dec  zad2.enc  zad3.txt  zad6.dec  zad7.enc
Dokumenty  key         key6         Pobrane    Templates       zad1.enc  zad2.txt  zad5.dec  zad6.enc
haslo      key2        Muzyka       priv.pem   Untitled.ipynb  zad1.txt  zad3.dec  zad5.enc  zad6.txt
IV         key3        obrazek.png  Publiczny  Wideo           zad2.dec  zad3.enc  zad5.txt  zad7.dec
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat priv.pem
-----BEGIN PRIVATE KEY-----
MIICdAIBADANBgkqhkiG9w0BAQEFAASCAl4wggJaAgEAAoGBALL/bKTk7OHAMvsy
S1I9niTZ3lmPtvg2AtpAkYNi15zHhMsvl35PtByo5VE53SB8RFJduIxkeuOIkGOl
MvlDGGa5VpcoyA5gQsMIy5FqNIpTSnXawUa7SDN9Bxq8/1c5b1CrA/2zQOvc5t3q
lGY3I4kL3TE1r4O9CV9oikQHAHS1AgMBAAECgYBNlieKhewjokK23UMGv8ynnoQh
gAjIJ+kunyQztglNXLIzvEpHJtckQRU9Ev7FUPfwV81hlUArFgu4FWg14R5W1gc+
TNl44caBpUfhHzlbEdueoBlAYxSOZsHIFoaWcYTGCmmZby+WHwE4sZGP9HsgZjfA
BChSqtlqBXOZJ3LFtQJBAOr9cF/WFDHGj80wi/bvDqz94AV1RAfdxyjfmx77jrFA
1oJpeB5jMHNQcXb3R9zxPmylsF9g11g1wOG8AW+OYesCQQDDAGiiQBXkDXxP3Ad8
bFSbuNcnWd3NpAga34yXcGUDU6MWvH924iwSzN31OqQmrMXWig8QLa68Uk3ahTRa
ODvfAkBgkjj9WahA8H4mcD4FBCd+1nKbK58CsJTpakOAjxcMJz5AqAFyU4cUzMlk
f6kr1oS1972s5Mp68kD7Q90/MD9zAkB9KLX/7cIfLcisOT1UzvM/P9RYnYWPh8jr
9q/2Q12SVQxIe5QUafZ/2p5b19ekOyCAIvyoFaohlXIC35xKi+1hAj8ogZ+Rj25C
ot9CYCWOV2I6dAN2qAVvki7tNM+Rcpt6Lo3PrbhwutfR11fYl1V1qvaH14+wXh6l
X67/HAoIf6U=
-----END PRIVATE KEY-----
```

```
└──╼ $openssl pkeyutl -help
Usage: pkeyutl [options]

General options:
 -help                       Display this summary
 -engine val                 Use engine, possibly a hardware device
 -engine_impl                Also use engine given by -engine for crypto operations
 -sign                       Sign input data with private key
 -verify                     Verify with public key
 -encrypt                    Encrypt input data with public key
```

```
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl pkey -help
Usage: pkey [options]

General options:
 -help                Display this summary
 -engine val          Use engine, possibly a hardware device

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
 -check               Check key consistency
 -pubcheck            Check public key consistency

Input options:
 -in val              Input key
 -inform format       Key input format (ENGINE, other values ignored)
 -passin val          Key input pass phrase source
 -pubin               Read only public components from key input

Output options:
 -out outfile         Output file for encoded and/or text output
 -outform PEM|DER     Output encoding format (DER or PEM)
 -*                   Any supported cipher to be used for encryption
 -passout val         Output PEM file pass phrase source
 -traditional         Use traditional format for private key PEM output
 -pubout              Restrict encoded output to public components
 -noout               Do not output the key in encoded form
 -text                Output key components in plaintext
 -text_pub            Output only public key components in text form
 -ec_conv_form val    Specifies the EC point conversion form in the encoding
 -ec_param_enc val    Specifies the way the EC parameters are encoded
(.python_env) ┌─[admin@parrot]─[~]
```

To jest dorma tekstowa /\ to na dole. Na górze base64

```
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl pkey -in priv.pem -pubout -out pubkey.pem
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat pubkey.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCy/2yk5OzhwDL7MktSPZ4k2d5Z
j7b4NgLaQJGDYtecx4TLL5d+T7QcqOVROd0gfERSXbiMZHrjiJBjpTL5QxhmuVaX
KMgOYELDCMuRajSKU0p12sFGu0gzfQcavP9XOW9QqwP9s0Dr3Obd6pRmNyOJC90x
Na+DvQlfaIpEBwB0tQIDAQAB
-----END PUBLIC KEY-----
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl pkey -in priv.pem -text -noout > privkey.pem
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat privkey.pem
Private-Key: (1024 bit, 2 primes)
modulus:
    00:b2:ff:6c:a4:e4:ec:e1:c0:32:fb:32:4b:52:3d:
    9e:24:d9:de:59:8f:b6:f8:36:02:da:40:91:83:62:
    d7:9c:c7:84:cb:2f:97:7e:4f:b4:1c:a8:e5:51:39:
    dd:20:7c:44:52:5d:b8:8c:64:7a:e3:88:90:63:a5:
    32:f9:43:18:66:b9:56:97:28:c8:0e:60:42:c3:08:
    cb:91:6a:34:8a:53:4a:75:da:c1:46:bb:48:33:7d:
    07:1a:bc:ff:57:39:6f:50:ab:03:fd:b3:40:eb:dc:
    e6:dd:ea:94:66:37:23:89:0b:dd:31:35:af:83:bd:
    09:5f:68:8a:44:07:00:74:b5
publicExponent: 65537 (0x10001)
privateExponent:
    4d:96:27:8a:85:ec:23:a2:42:b6:dd:43:06:bf:cc:
```

```
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl pkey -in priv.pem -text_pub -noout
Public-Key: (1024 bit)
Modulus:
    00:b2:ff:6c:a4:e4:ec:e1:c0:32:fb:32:4b:52:3d:
    9e:24:d9:de:59:8f:b6:f8:36:02:da:40:91:83:62:
    d7:9c:c7:84:cb:2f:97:7e:4f:b4:1c:a8:e5:51:39:
    dd:20:7c:44:52:5d:b8:8c:64:7a:e3:88:90:63:a5:
    32:f9:43:18:66:b9:56:97:28:c8:0e:60:42:c3:08:
    cb:91:6a:34:8a:53:4a:75:da:c1:46:bb:48:33:7d:
    07:1a:bc:ff:57:39:6f:50:ab:03:fd:b3:40:eb:dc:
    e6:dd:ea:94:66:37:23:89:0b:dd:31:35:af:83:bd:
    09:5f:68:8a:44:07:00:74:b5
Exponent: 65537 (0x10001)
(.python_env) ┌─[admin@parrot]─[~]
```

```
(.python_env) ┌─[✗]─[admin@parrot]─[~]
└──╼ $curl -X 'POST'   'http://127.0.0.1:3001/upload/public'   -H 'accept: application/json'
 -H 'Content-Type: multipart/form-data'   -F 'file=@pubkey.pem'
{
  "bits": 1024,
  "bytes": 272,
  "error": null,
  "is_rsa": true,
  "type": "public"
}
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $curl -X 'POST'   'http://127.0.0.1:3001/upload/private'   -H 'accept: application/json'
  -H 'Content-Type: multipart/form-data'   -F 'file=@privkey.pem'
{
  "bits": null,
  "bytes": 2085,
  "error": "RSA key format is not supported",
  "is_rsa": false,
  "type": null
}
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl pkey -in priv.pem -text_pub -noout > privkey.pem
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $curl -X 'POST'   'http://127.0.0.1:3001/upload/private'   -H 'accept: application/json'
  -H 'Content-Type: multipart/form-data'   -F 'file=@privkey.pem'
{
  "bits": null,
  "bytes": 489,
  "error": "RSA key format is not supported",
  "is_rsa": false,
  "type": null
}
(.python_env) ┌─[admin@parrot]─[~]
```

3.2 *Pod adresem* `http://127.0.0.1:3002` *działa prosty serwer HTTP udostępniający dwa endpointy:* `http://127.0.0.1:3002/upload/ec/public` *oraz* `http://127.0.0.1:3002/upload/ec/private`.

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 3002:3002 --name ex2 docker.io/mazurkatarzyna/asymmetric-enc-ex2:latest
podman run -p 3002:3002 --name ex2 docker.io/mazurkatarzyna/asymmetric-enc-ex2:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 3002:3002 --name ex2 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex2:latest
podman run -p 3002:3002 --name ex2 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex2:latest
```

(b) Wygeneruj parę kluczy EC - krzywych eliptycznych (publiczny i prywatny). Wykorzystaj krzywą `prime256v1`. Wyeksportuj oba klucze do plików.

(c) Używając metody HTTP `POST`, wyślij parę kluczy EC do serwera poprzez endpointy `http://127.0.0.1:3002/upload/ec/public` (jako `file`) oraz `http://127.0.0.1:3002/upload/ec/private` (również jako `file`). W odpowiedzi serwer zwróci odpowiednią informację (szczegóły klucza).

UWAGI:

- Aby wygenerować parę kluczy EC, wykorzystaj narzędzie `OpenSSL`. Użyj argumentu genpkey oraz ec.
- Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dodaniu nagłówków protokołu HTTP: `Content-Type: application/json`, `Content-Type: multipart/form-data`.

```
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl genpkey -algorithm EC -out ecpriv.pem -pkeyopt ec_paramgen_curve:prime256v1
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl ec -in ecpriv.pem -pubout -out ecpub.pem
read EC key
writing EC key
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $curl -X 'POST'   'http://127.0.0.1:3002/upload/ec/public'   -H 'accept: */*'   -H 'Conte
nt-Type: multipart/form-data'   -F 'file=@ecpub.pem'
{
  "bytes": 178,
  "curve": "NIST P-256",
  "error": null,
  "is_ec": true,
  "type": "public"
}
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $
```

3.4 *Pod adresem* `http://127.0.0.1:3004` *działa prosty serwer HTTP udostępniający dwa endpointy:* `http://127.0.0.1:3004/getprivkey` *oraz* `http://127.0.0.1:3004/submit`.

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 3004:3004 --name ex4 docker.io/mazurkatarzyna/asymmetric-enc-ex4:latest
podman run -p 3004:3004 --name ex4 docker.io/mazurkatarzyna/asymmetric-enc-ex4:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 3004:3004 --name ex4 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex4:latest
podman run -p 3004:3004 --name ex4 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex4:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:3004/getprivkey` używając metody HTTP `GET`. Otrzymasz w odpowiedzi identyfikator sesji (`session_id`), oraz klucz prywatny RSA (`private_key_pem`)

(c) Na podstawie otrzymanego klucza prywatnego, wygeneruj odpowiadający mu klucz publiczny. Ilu bitowy jest klucz?

(d) Za pomocą metody HTTP `POST`, wyślij pod endpoint `http://127.0.0.1:3004/submit` wygenerowany klucz publiczny (`public_key_pem`) oraz identyfikator sesji (`session_id`).

UWAGI:

- Aby wygenerować klucz publiczny, wykorzystaj narzędzie `OpenSSL`. Użyj argumentu `pkey`.
- Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dodaniu nagłówka protokołu HTTP `Content-Type: application/json`.

```
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $curl -X 'GET' 'http://127.0.0.1:3004/getprivkey'
{"private_key_pem":"-----BEGIN RSA PRIVATE KEY-----\nMIIEogIBAAKCAQEAwHNsbmWNPLMpCC6j0ScT6RzO9
0lh0D4lr1+oMfRSP0s3Ykjw\nthUnicp7aQmmEq2Iy5Y/0IIBC92VbL+qG95FW174nnohZULoq2dPDNzdiqdyPFBh\nky+
/9xk0av2iCclWce1o/cCiogKkxO+ctJQoq7WJSvKZDDWGABgfNJkIYZu1c7Kr\nmLA9B28KdAn146C3dTqmBHC5IWnqlPQ
MPAqnEMhyZl9YbERnhGNd2FIj7t5cColi\n85rAib92abulOEN5kkkMm4rfjpW2HO0ihndQjsdO82kYwFP18wJiCzLvdkW
w91a4\nB2j6155ENAYps37fJDGDHiZhH0jzhVJAwHsUEQIDAQABAoIBADZeRVM40DpGOQDN\nQE1dl1PjZhrZOc9dR0Mwq
HEYX0XvR2G2Tn56InXcZJnrdmXPbZZqR3PtzkLeYhDb\nYsDFvd1Oyrjmstm LGfUWVfBB0zqRDIY6hR5SPo6dqb/SEH1oS
kocMm2BhtBzI75Z\nHPGs64J64iLwv0kCM+2B4wr2crY0DBiG5VJQ8l2+jsx2/GqUBLpdnNgvqw08ixGS\naebyDp2CbM5
jBgCe1LlC8bVzOUevm7X0ZZTE1mFacWo+kUY4PSidh5CDqnKbd8Ca\n7S40qLrSAht3RPF04EW+z45hQuwvcGWAW6ITA2L
duXz1iSfR1PJsUXfNRTP3UkzF\ngeoXNZsCgYEA3RPYOBV3XpOskLiwt32PUMVHLCWkMbUppp7XapgNMaQIaccdllDF\ny
4MRvuH0jUYLAIWb6upYvgWHAQja+3NmQgIbMkQmPwzDrtAv67+yDbhfamZV+4qo\nY7V9Eq0tKV5yXrG5YPCxY7FHXJWzr
0JJ919i4nFZrzqZHHLeKJyzYy8CgYEA3tny\nI5NlYejlam6TwRyCXQ0ujIvkXtB0OR6R5ELSRrBJnWj+wtMM4KepiI6i6
DEOxEIG\nMOopfE8Zgo2PTgllwOGTeNGl8j3cpIaMHgRytvwPdM77lrYXSY6YHhQLMJbgdLV6\nev+W52kGi1PdC0Lkry9
nGXdMtIfvXVZ/sxHoLL8CgYAgCw9M2beqiqb5dvuOLPHH\n2EA/otkQp1x5HQ7GEXHgHxFlG4orTlvM6pt42lFS5LMViUg
XHY5tRGnT+Y+b4JcS\nvDFq8LvGabg3L0+y0N8i4/VYe6q8wdU0VorhWR5xn5HeLc5bwadEdt6MCdq4bVhB\nNZ8kiR9hq
ZGlsPSIWGYb2wKBgDMsJs8GhKxD95JGDAW3DzRg14SMhwdCpL2NCIE6\n2uBCkNTljEKehgF8XpCt+DiLM1Hbu2I2Ewqpe
zkgWnzX6Zat9dgzhACpNeZNwxUN\nfm4IWWBAO0NzUsugQ8v8XElvC+Gg3pwp3rQKyObrfgGc/bhAkVMmsuVtpYhwWWh8\
nPy0vAoGAK/NiMkYjA5KC6bP7sLbJyE7fJhbKcBcUcRyExZwQQazg1o1r1y+pZcGo\nGUR7f1t9enfI9cfryvl7JcohlMo
DVASdHoDlEML/9r6mc4mwf4M+sQfDWEDJ+0Uc\npEp14Y7GOkkaIVWPC5GRSn/bUewRhZHbrDepkCxmnWJvQki/AKI=\n-
----END RSA PRIVATE KEY-----","session_id":"df06693d92c22191"}
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $nano zad3_4.priv
```

```
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl pkey -in zad3_4.priv -pubout -out zad3_4.pub
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat zad3_4.pub
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwHNsbmWNPLMpCC6j0ScT
6RzO90lh0D4lr1+oMfRSP0s3YkjwthUnicp7aQmmEq2Iy5Y/0IIBC92VbL+qG95F
W174nnohZULoq2dPDNzdiqdyPFBhky+/9xk0av2iCclWce1o/cCiogKkxO+ctJQo
q7WJSvKZDDWGABgfNJkIYZu1c7KrmLA9B28KdAn146C3dTqmBHC5IWnqlPQMPAqn
EMhyZl9YbERnhGNd2FIj7t5cColi85rAib92abulOEN5kkkMm4rfjpW2HO0ihndQ
jsdO82kYwFP18wJiCzLvdkWw91a4B2j6155ENAYps37fJDGDHiZhH0jzhVJAwHsU
EQIDAQAB
-----END PUBLIC KEY-----
(.python_env) ┌─[admin@parrot]─[~]
```

```
└──╼ $curl -X 'POST' 'http://127.0.0.1:3004/submit' -H 'accept: */*' -H 'Content-Type: applica
tion/json' -d "$(jq -n --arg session_id "df06693d92c22191" --arg pub "$(cat zad3_4.pub)" '{ses
sion_id: $session_id, public_key_pem: &pub}')"
jq: error: syntax error, unexpected INVALID_CHARACTER (Unix shell quoting issues?) at <top-lev
el>, line 1:
{session_id: $session_id, public_key_pem: &pub}
jq: 1 compile error
<!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]─[~]
```

3.5 *Pod adresem* `http://127.0.0.1:3005` *działa prosty serwer HTTP udostępniający dwa endpointy:* `http://127.0.0.1:3005/getprivkey` *oraz* `http://127.0.0.1:3005/submit`.

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 3005:3005 --name ex5 docker.io/mazurkatarzyna/asymmetric-enc-ex5:latest
podman run -p 3005:3005 --name ex5 docker.io/mazurkatarzyna/asymmetric-enc-ex5:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 3005:3005 --name ex5 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex5:latest
podman run -p 3005:3005 --name ex5 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex5:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:3005/getprivkey` używając metody HTTP `GET`. Otrzymasz w odpowiedzi identyfikator sesji (`session_id`), oraz klucz prywatny EC (`private_key_pem`).

(c) Na podstawie otrzymanego klucza prywatnego, wygeneruj odpowiadający mu klucz publiczny. Ilu bitowy jest klucz?

(d) Za pomocą metody HTTP `POST`, wyślij pod endpoint `http://127.0.0.1:3005/submit` wygenerowany klucz publiczny (`public_key_pem`) oraz identyfikator sesji (`session_id`).

UWAGI:

- Aby wygenerować klucz publiczny, wykorzystaj narzędzie `OpenSSL`. Użyj argumentu `ec`.
- Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dodaniu nagłówka protokołu HTTP `Content-Type: application/json`.

```
└──╼ $echo -n "Hello" > slowo.txt
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl genpkey -algorithm RSA -out keys.pem -pkeyopt rsa_keygen_bits:1024
.....................+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.........+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl pkey -in key
key       key2      key3      key5      key6      keys.pem
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl pkey -in keys.pem -pubout -out pub.pem
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl pkeyutl -encrypt -pubin -inkey pub.pem -in slowo.txt -pkeyopt rsa_padding_mode:o
aep -out ciphertext.enc
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat ciphertext.enc
ηZ[illegible]
&<-[illegible]
                    [illegible]
[illegible].python_env) ┌─[admin@parrot]─[~]
└──╼ $cat ciphertext.enc | base64 > ciphertext.txt
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat ciphertext.txt
HtWyWswl5UYBhWb1EO1RMNKyocJvBQ5mso+JN8jfh4lVmgomPC3duNtDK+4IYQLlX8DjAPD8+EPm
bYMoouk5C8pJST4j4bsUTNexwIPLnBps9V4jv3XoOosPWzdj66Qo97fiUgG+qSbVdTHOPzgwvUiV
TQrE0mSWSEZXF63/HdY=
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl pkeyutl -decrypt -inkey priv.pem -in ciphertext.enc -pkey opt rsa_padding_mode:o
aep -out decrypted.txt
pkeyutl: Unknown option: -pkey
pkeyutl: Use -help for summary.
(.python_env) ┌─[✗]─[admin@parrot]─[~]
└──╼ $openssl pkeyutl -encrypt -pubin -inkey pub.pem -in slowo.txt -pkeyopt rsa_padding_mode:o
aep -out ciphertext.enc
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl pkeyutl -decrypt -inkey keys.pem -in ciphertext.enc -pkeyopt rsa_padding_mode:oa
ep -out decrypted.txt
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $
```

```
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat slowo.txt
Hello(.python_env) ┌─[admin@parrot]─[~]
└──╼ $cat decrypted.txt
Hello(.python_env) ┌─[admin@parrot]─[~]
└──╼ $
```

3.9 *Pod adresem* `http://127.0.0.1:3009` *działa prosty serwer HTTP udostępniający jeden endpoint:* `http://127.0.0.1:3009/sign`.

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 3009:3009 --name ex9 docker.io/mazurkatarzyna/asymmetric-enc-ex9:latest
podman run -p 3009:3009 --name ex9 docker.io/mazurkatarzyna/asymmetric-enc-ex9:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 3009:3009 --name ex9 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex9:latest
podman run -p 3009:3009 --name ex9 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex9:latest
```

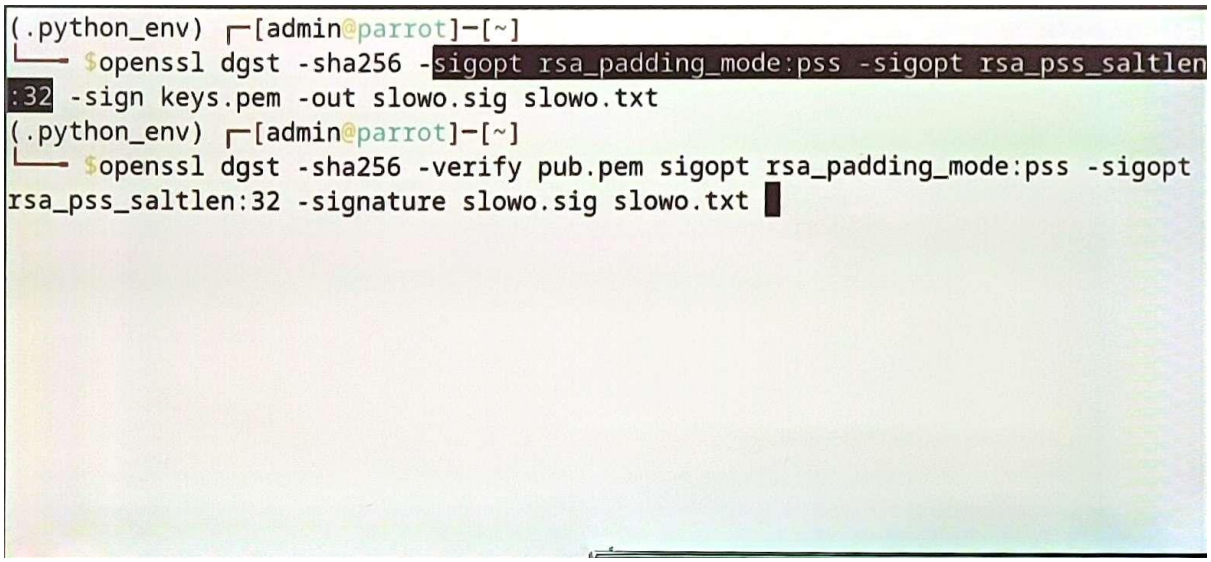
(b) Wyślij request do endpointa `http://127.0.0.1:3009/sign` używając metody HTTP `GET`. Otrzymasz w odpowiedzi unikalny identyfikator sesji (`session_id`), klucz prywatny RSA (`private_key_pem`) oraz słowo (`word`).

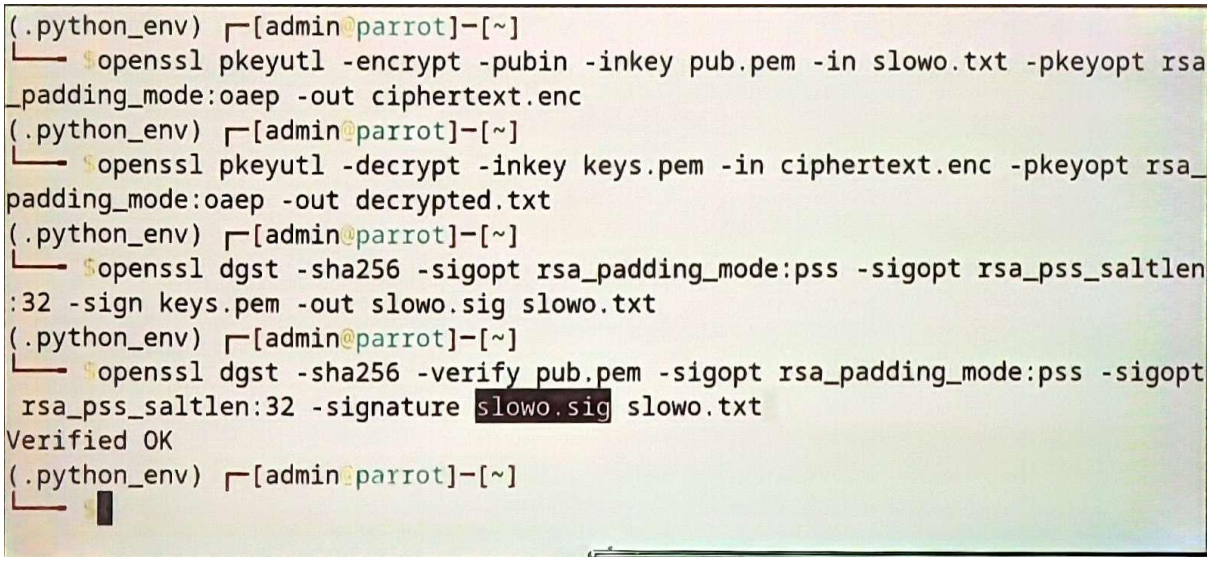
(c) Podpisz otrzymane słowo (`word`) kluczem prywatnym (`private_key_pem`), a następnie zakoduj wynik do formatu `base64`. Przy podpisywaniu należy zwrócić uwagę na parametry paddingu PSS:

- Używana funkcja skrótu: `SHA-256`
- `Salt length`: długość soli powinna odpowiadać długości skrótu (32 bajty dla `SHA-256`), aby podpis był zgodny z weryfikacją po stronie serwera.

(d) Używając metody HTTP `POST`, wyślij do serwera poprzez endpoint `http://127.0.0.1:3009/submit` podpisane słowo (jako `signature_b64`) oraz identyfikator sesji (`session_id`).

(e) Serwer zweryfikuje podpis przy użyciu klucza publicznego i zwróci odpowiednią informację o sukcesie lub błędzie.




(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl pkeyutl -encrypt -pubin -inkey pub.pem -in slowo.txt -pkeyopt rsa
_padding_mode:oaep -out ciphertext.enc
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl pkeyutl -decrypt -inkey keys.pem -in ciphertext.enc -pkeyopt rsa_
padding_mode:oaep -out decrypted.txt
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl dgst -sha256 -sigopt rsa_padding_mode:pss -sigopt rsa_pss_saltlen
:32 -sign keys.pem -out slowo.sig slowo.txt
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $openssl dgst -sha256 -verify pub.pem -sigopt rsa_padding_mode:pss -sigopt
rsa_pss_saltlen:32 -signature slowo.sig slowo.txt
Verified OK
(.python_env) ┌─[admin@parrot]─[~]
└──╼ $