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`.

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`.

3.3 *Pod adresem http://127.0.0.1:3003 działa prosty serwer HTTP udostępniający jeden endpoint: http://127.0.0.1:3003/checkkeys.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 3003:3003 --name ex3 docker.io/mazurkatarzyna/asymmetric-enc-ex3:latest
podman run -p 3003:3003 --name ex3 docker.io/mazurkatarzyna/asymmetric-enc-ex3:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 3003:3003 --name ex3 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex3:latest
podman run -p 3003:3003 --name ex3 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex3:latest
```

(b) Wygeneruj parę kluczy RSA (publiczny i prywatny) o długości 1024 bitów. Wyeksportuj oba klucze do plików.

(c) Używając metody HTTP `POST`, wyślij parę kluczy RSA do serwera poprzez endpoint `http://127.0.0.1:3003/checkkeys` (jako pliki `private_key_pem` oraz `public_key_pem` w jednym requeście). W odpowiedzi serwer zwraci informację o poprawności i dopasowaniu kluczy (czy klucz publiczny odpowiada przesłanemu kluczowi prywatnemu).

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 nagłówka protokołu HTTP `Content-Type: application/json`.

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 (w nagłówku `X-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 (jako plik, `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`.

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 (w nagłówku `X-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 (jako plik, `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`.

3.6 *Pod adresem http://127.0.0.1:3006 działa prosty serwer HTTP udostępniający 2 endpointy: http://127.0.0.1:3006/encrypt oraz http://127.0.0.1:3006/submit.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 3006:3006 --name ex6 docker.io/mazurkatarzyna/asymmetric-enc-ex6:latest
podman run -p 3006:3006 --name ex6 docker.io/mazurkatarzyna/asymmetric-enc-ex6:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 3006:3006 --name ex6 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex6:latest
podman run -p 3006:3006 --name ex6 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex6:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:3006/encrypt` używając metody HTTP `GET`. Otrzymasz w odpowiedzi unikalny identyfikator sesji (w nagłówku `X-Session-ID`), losowe słowo do zaszyfrowania (w nagłówku `X-Word`) oraz klucz publiczny RSA w formacie `PEM` (jako `public_key_pem`).

(c) Zaszyfruj otrzymane słowo, stosując algorytm `RSA-2048` z użyciem klucza publicznego oraz wybranego trybu paddingu: `OAEP`.

(d) Wyślij request do serwera poprzez endpoint `http://127.0.0.1:3006/submit` używając metody HTTP `POST`, identyfikator sesji (`session_id`), jak i zaszyfrowane słowo (jako plik, `encrypted_file`).

(e) W odpowiedzi serwer zwróci odpowiednią informację o sukcesie lub błędzie - zweryfikuje poprawność przesłanego zaszyfrowanego i zakodowanego słowa.

UWAGI:

- Aby zaszyfrować odebrane od serwera słowo, wykorzystaj narzędzie `OpenSSL`. Użyj argumentu `pkeyutl`.
- Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dodaniu nagłówka protokołu HTTP `Content-Type: application/json`.

3.7 *Pod adresem `http://127.0.0.1:3007` działa prosty serwer HTTP udostępniający dwa endpointy: `http://127.0.0.1:3007/decrypt` oraz `http://127.0.0.1:3007/submit`.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 3007:3007 --name ex7 docker.io/mazurkatarzyna/asymmetric-enc-ex7:latest
podman run -p 3007:3007 --name ex7 docker.io/mazurkatarzyna/asymmetric-enc-ex7:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 3007:3007 --name ex7 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex7:latest
podman run -p 3007:3007 --name ex7 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex7:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:3007/decrypt` używając metody HTTP `GET`. Otrzymasz w odpowiedzi unikalny identyfikator sesji (w nagłówku `X-Session-ID`), klucz prywatny oraz zakodowane i zaszyfrowane słowo, oba zapakowane w pliku `*.zip`. (Serwer wygenerował losowe słowo, zaszyfrował je algorytmem `RSA` (klucz 4096-bit) i zakodował w formacie `base64`.)

(c) Odkoduj podane słowo, następnie odszyfruj je przy użyciu klucza prywatnego i algorytmu `RSA-4096` z paddingiem `OAEP`.

(d) Wyślij request do serwera poprzez endpoint `http://127.0.0.1:3007/submit` używając metody HTTP `POST`, identyfikator sesji (`session_id`) oraz odszyfrowane słowo (`decrypted_word`).

(e) W odpowiedzi serwer zweryfikuje poprawność odszyfrowanego słowa i zwróci informację o sukcesie lub błędzie.

UWAGI:

- Aby odszyfrować odebrane od serwera słowo, wykorzystaj narzędzie `OpenSSL`. Użyj argumentu `pkeyutl`.
- Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dodaniu nagłówka protokołu HTTP `Content-Type: application/json`.

3.8 *Pod adresem `http://127.0.0.1:3008` działa prosty serwer HTTP udostępniający jeden endpoint: `http://127.0.0.1:3008/decrypt`.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 3008:3008 --name ex8 docker.io/mazurkatarzyna/asymmetric-enc-ex8:latest
podman run -p 3008:3008 --name ex8 docker.io/mazurkatarzyna/asymmetric-enc-ex8:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 3008:3008 --name ex8 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex8:latest
podman run -p 3008:3008 --name ex8 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex8:latest
```

(b) Wygeneruj parę kluczy RSA (jako `keys_pem`).

(c) Wyeksporuj klucz publiczny z pary kluczy do pliku (jako `public_key.pem`).

(d) Wybierz słowo i zapisz je do pliku (jako `plaintext.txt`).

(e) Zaszyfruj wybrane przez siebie słowo (`plaintext.txt`) wygenerowanym kluczem publicznym (`public_key.pem`). Wykorzystaj padding `OAEP`.

(f) Zakoduj zaszyfrowane słowo i zapisz je do pliku (`ciphertext.txt`).

(g) Wyślij request do endpointa `http://127.0.0.1:3008/decrypt` używając metody HTTP `POST` i prześlij do serwera:

- wygenerowany klucz prywatny (jako plik `private_key.pem`),
- wygenerowany klucz publiczny (jako plik `public_key.pem`),
- wybrane przez siebie słowo (jako plik `plaintext`),
- wybrane przez siebie słowo zaszyfrowane kluczem publicznym (`public_key.pem`) z paddingiem `OAEP` i zakodowane w `base64` (jako plik `ciphertext.txt`).

(h) Zadaniem serwera jest sprawdzenie, czy zaszyfrowane słowo zostało zaszyfrowane podanym kluczem, oraz, czy słowa zgadzają się po odszyfrowaniu.

UWAGI:

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

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.

3.10 *Pod adresem `http://127.0.0.1:3010` działa prosty serwer HTTP udostępniający dwa endpointy: `http://127.0.0.1:3010/verify` oraz `http://127.0.0.1:3010/submit`.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 3010:3010 --name ex10 docker.io/mazurkatarzyna/asymmetric-enc-ex10:latest
podman run -p 3010:3010 --name ex10 docker.io/mazurkatarzyna/asymmetric-enc-ex10:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 3010:3010 --name ex10 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex10:latest
podman run -p 3010:3010 --name ex10 ghcr.io/mazurkatarzynaumcs/asymmetric-enc-ex10:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:3010/verify` używając metody HTTP `GET`. Otrzymasz w odpowiedzi klucz publiczny RSA (`public_key_pem`), podpis (`signature_b64`) oraz słowo (`word`).

(c) Zweryfikuj podpis przy użyciu algorytmu `RSA-2048` oraz trybu paddingu `PSS`. Przy weryfikacji 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 weryfikacja lokalna była zgodna z serwerem.

(d) Używając metody HTTP `POST`, wyślij do serwera poprzez endpoint `http://127.0.0.1:3009/submit` słowo (jako `word`), klucz publiczny (jako `public_key_pem`), podpis w formacie `base64` (jako `signature_b64`) oraz informację, czy weryfikacja lokalna zakończyła się sukcesem (true/false) (jako `user_verified`).

(e) Serwer sprawdzi, czy Twoja lokalna weryfikacja jest zgodna z faktycznym podpisem i zwróci odpowiednią informację.