5.1 *Pod adresem http://127.0.0.1:5001 działa prosty serwer HTTP udostępniający 2 endpointy: http://127.0.0.1:5001/submit/public oraz http://127.0.0.1:5001/submit/private.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 5001:5001 --name ex1 docker.io/mazurkatarzyna/gpg-ex1:latest
podman run -p 5001:5001 --name ex1 docker.io/mazurkatarzyna/gpg-ex1:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 5001:5001 --name ex1 ghcr.io/mazurkatarzynaumcs/gpg-ex1:latest
podman run -p 5001:5001 --name ex1 ghcr.io/mazurkatarzynaumcs/gpg-ex1:latest
```

(b) Wygeneruj parę kluczy GPG używając algorytmu RSA.

(c) Wyeksportuj klucz publiczny i klucz prywatny do plików, odpowiednio `pub.key` oraz `priv.key`.

(d) Wyślij request do endpointa `http://127.0.0.1:5001/submit/public` używając meody HTTP `POST` i prześlij na serwer wyeksportowany klucz publiczny GPG. Serwer w odpowiedzi zwróci informacje o kluczu.

(e) Wyślij request do endpointa `http://127.0.0.1:5001/submit/private` używając meody HTTP `POST` i prześlij na serwer wyeksportowany klucz prywatny GPG. Serwer w odpowiedzi zwróci informacje o kluczu.

UWAGI:

- Aby za pomocą narzędzia `cURL` wysłać plik do serwera, użyj składni: `-F "plik=@hello.txt"`

5.2 *Pod adresem http://127.0.0.1:5002 działa prosty serwer HTTP udostępniający 2 endpointy: http://127.0.0.1:5002/submit/public oraz http://127.0.0.1:5002/submit/private.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 5002:5002 --name ex2 docker.io/mazurkatarzyna/gpg-ex2:latest
podman run -p 5002:5002 --name ex2 docker.io/mazurkatarzyna/gpg-ex2:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 5002:5002 --name ex2 ghcr.io/mazurkatarzynaumcs/gpg-ex2:latest
podman run -p 5002:5002 --name ex2 ghcr.io/mazurkatarzynaumcs/gpg-ex2:latest
```

(b) Wygeneruj parę kluczy GPG używając algorytmu DSA.

(c) Wyeksportuj klucz publiczny i klucz prywatny do plików, odpowiednio `pub.key` oraz `priv.key`.

(d) Wyślij request do endpointa `http://127.0.0.1:5002/submit/public` używając meody HTTP `POST` i prześlij na serwer wyeksportowany klucz publiczny GPG. Serwer w odpowiedzi zwróci informacje o kluczu.

(e) Wyślij request do endpointa `http://127.0.0.1:5002/submit/private` używając meody HTTP `POST` i prześlij na serwer wyeksportowany klucz prywatny GPG. Serwer w odpowiedzi zwróci informacje o kluczu.

UWAGI:

- Aby za pomocą narzędzia `cURL` wysłać plik do serwera, użyj składni: `-F "plik=@hello.txt"`

5.3 *Pod adresem http://127.0.0.1:5003 działa prosty serwer HTTP udostępniający 2 endpointy: http://127.0.0.1:5003/submit/public oraz http://127.0.0.1:5003/submit/private.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 5003:5003 --name ex3 docker.io/mazurkatarzyna/gpg-ex3:latest
podman run -p 5003:5003 --name ex3 docker.io/mazurkatarzyna/gpg-ex3:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 5003:5003 --name ex3 ghcr.io/mazurkatarzynaumcs/gpg-ex3:latest
podman run -p 5003:5003 --name ex3 ghcr.io/mazurkatarzynaumcs/gpg-ex3:latest
```

(b) Wygeneruj parę kluczy GPG używając algorytmu RSA. Klucze powinny mieć `1024` bity długości, mają być ważne rok, oraz być wygenerowane dla adresu `student@uczelnia.pl`.

(c) Wyeksportuj klucz publiczny i klucz prywatny do plików, odpowiednio `pub.key` oraz `priv.key`.

(d) Wyślij request do endpointa `http://127.0.0.1:5003/submit/public` używając meody HTTP `POST` i prześlij na serwer wyeksportowany klucz publiczny GPG. Serwer w odpowiedzi zwróci informacje o kluczu.

(e) Wyślij request do endpointa `http://127.0.0.1:5003/submit/private` używając meody HTTP `POST` i prześlij na serwer wyeksportowany klucz prywatny GPG. Serwer w odpowiedzi zwróci informacje o kluczu.

5.4 *Pod adresem http://127.0.0.1:5004 działa prosty serwer HTTP udostępniający 2 endpointy: http://127.0.0.1:5004/decrypt oraz http://127.0.0.1:5004/submit.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 5004:5004 --name ex4 docker.io/mazurkatarzyna/gpg-ex4:latest
podman run -p 5004:5004 --name ex4 docker.io/mazurkatarzyna/gpg-ex4:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 5004:5004 --name ex4 ghcr.io/mazurkatarzynaumcs/gpg-ex4:latest
podman run -p 5004:5004 --name ex4 ghcr.io/mazurkatarzynaumcs/gpg-ex4:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:5004/decrypt` używając meody HTTP `GET` i zapisz odpowiedź do pliku z rozszerzeniem `*.zip`.

(c) Rozpakuj pobrane archiwum.

(d) Odszyfruj plik `encrypted.txt` używając algorytmu `CAMELLIA128` i hasła znajdującego się w pliku `passphrase.txt`.

(e) Wyślij request do endpointa `http://127.0.0.1:5004/submit/` używając meody HTTP `POST`, i prześlij do serwera ID sesji (jako `session_id`, które znajdziesz w pliku `session_id.txt`) oraz odszyfrowane słowo (jako `decrypted_text`). Serwer w odpowiedzi zwróci informacje o poprawnym lub nieprawidłowym deszyfrowaniu.

UWAGI:

- Aby zapisać odpowiedź do pliku z rozszerzeniem `*.zip`, użyj opcji `-o`.
- Podczas deszyfrowania symetrycznego sprawdź opcje:
  `--cipher-algo`
  `--batch --yes`
  `--passphrase`

5.5 *Pod adresem http://127.0.0.1:5005 działa prosty serwer HTTP udostępniający 2 endpointy: http://127.0.0.1:5005/encrypt oraz http://127.0.0.1:5005/submit.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 5005:5005 --name ex5 docker.io/mazurkatarzyna/gpg-ex5:latest
podman run -p 5005:5005 --name ex5 docker.io/mazurkatarzyna/gpg-ex5:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 5005:5005 --name ex5 ghcr.io/mazurkatarzynaumcs/gpg-ex5:latest
podman run -p 5005:5005 --name ex5 ghcr.io/mazurkatarzynaumcs/gpg-ex5:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:5005/encrypt` używając meody HTTP `GET`. W odpowiedzi od serwera otrzymasz słowo do zaszyfrowania (jako `word_to_encrypt`), hasło potrzebne do szyfrowania (jako `passphrase`) oraz id sesji (jako `session_id`).

(c) Zaszyfruj odebrane od serwera słowo (`word_to_encrypt`) za pomocą algorytmu `AES128` oraz odebranego od serwera hasła (`passphrase`). Zaszyfrowane słowo ma być również zakodowane przy pomocy kodowania base64.

(d) Wyślij request do endpointa `http://127.0.0.1:5005/submit` używając meody HTTP `POST` i prześlij do serwera id sesji (jako `session_id`) oraz zaszyfrowany i zakodowany plik (jako plik, `encrypted_file`).

UWAGI:

- Aby wykorzystać kodowanie base64 podczas szyfrowania, użyj opcji `--armor`,
- Aby za pomocą narzędzia `cURL` wysłać plik do serwera, użyj składni: `-F "plik=@hello.txt"`

5.6 *Pod adresem http://127.0.0.1:5006 działa prosty serwer HTTP udostępniający 2 endpointy: http://127.0.0.1:5006/encrypt oraz http://127.0.0.1:5006/submit.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 5006:5006 --name ex6 docker.io/mazurkatarzyna/gpg-ex6:latest
podman run -p 5006:5006 --name ex6 docker.io/mazurkatarzyna/gpg-ex6:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 5006:5006 --name ex6 ghcr.io/mazurkatarzynaumcs/gpg-ex6:latest
podman run -p 5006:5006 --name ex6 ghcr.io/mazurkatarzynaumcs/gpg-ex6:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:5005/encrypt` używając meody HTTP `GET` i zapisz odpowiedź do pliku z rozszerzeniem `*.zip`.

(c) Rozpakuj pobrane archiwum. W odpowiedzi od serwera otrzymałeś id sesji (jako `session_id`), klucz publiczny GPG (`public_key.asc`), klucz prywatny GPG (`private_key.asc`), oraz słowo do zaszyfrowania (`word.txt`).

(d) Zaimportuj do swojego zbioru kluczy klucz publiczny pobrany od serwera.

(e) Wykorzsytując narzędzie `gpg` oraz pobrany klucz publiczny (z pliku `public_key.asc`), zaszyfruj odebrane od serwera słowo (podczas szyfrowania nie używaj hasła).

(f) Zaszyfrowane słowo zakoduj w base64 i zapisz do pliku.

(g) Wyślij request do endpointa `http://127.0.0.1:5006/submit` używając meody HTTP `POST` i prześlij do serwera id sesji (jako `session_id`) oraz zaszyfrowane i zakodowane słowo (jako plik, `encrypted_file`).

UWAGI:

- Aby zapisać odpowiedź do pliku z rozszerzeniem `*.zip`, użyj opcji `-o`.
- Aby wykorzystać kodowanie base64 podczas szyfrowania, użyj opcji `--armor`,
- Aby za pomocą narzędzia `cURL` wysłać plik do serwera, użyj składni: `-F "plik=@hello.txt"`

5.7 *Pod adresem http://127.0.0.1:5007 działa prosty serwer HTTP udostępniający 2 endpointy: http://127.0.0.1:5007/decrypt oraz http://127.0.0.1:5007/submit.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 5007:5007 --name ex7 docker.io/mazurkatarzyna/gpg-ex7:latest
podman run -p 5007:5007 --name ex7 docker.io/mazurkatarzyna/gpg-ex7:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 5007:5007 --name ex7 ghcr.io/mazurkatarzynaumcs/gpg-ex7:latest
podman run -p 5007:5007 --name ex7 ghcr.io/mazurkatarzynaumcs/gpg-ex7:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:5007/decrypt` używając meody HTTP `GET` i zapisz odpowiedź do pliku z rozszerzeniem `*.zip`.

(c) Rozpakuj pobrane archiwum. W odpowiedzi od serwera otrzymałeś id sesji (jako `session_id`), klucz publiczny GPG (`public_key.asc`), klucz prywatny GPG (`private_key.asc`), oraz słowo do odszyfrowania (`encrypted_word.asc`).

(d) Odszyfruj odebrane od serwera słowo (`encrypted_word.asc`) za pomocą pobranego klucza prywatnego (`private_key.asc`).

(e) Wyślij request do endpointa `http://127.0.0.1:5007/submit` używając meody HTTP `POST` i prześlij do serwera id sesji (jako `session_id`) oraz odszyfrowane słowo (jako tekst, `decrypted_word`).

UWAGI:

- Aby zapisać odpowiedź do pliku z rozszerzeniem `*.zip`, użyj opcji `-o`

5.8 *Pod adresem http://127.0.0.1:5008 działa prosty serwer HTTP udostępniający 2 endpointy: http://127.0.0.1:5008/sign oraz http://127.0.0.1:5008/submit.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 5008:5008 --name ex8 docker.io/mazurkatarzyna/gpg-ex8:latest
podman run -p 5008:5008 --name ex8 docker.io/mazurkatarzyna/gpg-ex8:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 5008:5008 --name ex8 ghcr.io/mazurkatarzynaumcs/gpg-ex8:latest
podman run -p 5008:5008 --name ex8 ghcr.io/mazurkatarzynaumcs/gpg-ex8:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:5008/sign` używając meody HTTP `GET` i zapisz odpowiedź do pliku z rozszerzeniem `*.zip`.

(c) Rozpakuj pobrane archiwum. W odpowiedzi od serwera otrzymałeś id sesji (jako `session_id`), klucz publiczny GPG (`public_key.asc`), klucz prywatny GPG (`private_key.asc`), oraz słowo do podpisania (w pliku `word.txt`).

(d) Podpisz plik (`word.txt`) za pomocą pobranego klucza publicznego (`public_key.asc`), tworząc plik `word.txt.gpg`.

(e) Wyślij request do endpointa `http://127.0.0.1:5008/submit` używając meody HTTP `POST` i prześlij do serwera id sesji (jako `session_id`) oraz podpisane słowo (jako plik, `signed_file`).

UWAGI:

- Aby zapisać odpowiedź do pliku z rozszerzeniem `*.zip`, użyj opcji `-o`,
- Aby utworzyć podpis zintegrowany, użyj opcji `--sign`,
- Aby wskazać klucz, którego będziemy używać do podpisywania, sprawdź opcję `-u`,
- Aby za pomocą narzędzia `cURL` wysłać plik do serwera, użyj składni: `-F "plik=@hello.txt"`

5.9 *Pod adresem `http://127.0.0.1:5009` działa prosty serwer HTTP udostępniający 2 endpointy: `http://127.0.0.1:5009/detach-sign` oraz `http://127.0.0.1:5009/submit`.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 5009:5009 --name ex9 docker.io/mazurkatarzyna/gpg-ex9:latest
podman run -p 5009:5009 --name ex9 docker.io/mazurkatarzyna/gpg-ex9:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 5009:5009 --name ex9 ghcr.io/mazurkatarzynaumcs/gpg-ex9:latest
podman run -p 5009:5009 --name ex9 ghcr.io/mazurkatarzynaumcs/gpg-ex9:latest
```

(b) Wyślij request do endpointa `http://127.0.0.1:5009/detach-sign` używając meody HTTP `GET` i zapisz odpowiedź do pliku z rozszerzeniem `*.zip`.

(c) Rozpakuj pobrane archiwum. W odpowiedzi od serwera otrzymałeś id sesji (jako `session_id`), klucz publiczny GPG (`public_key.asc`), klucz prywatny GPG (`private_key.asc`), oraz słowo do podpisania (w pliku `word.txt`).

(d) Podpisz plik (`word.txt`) za pomocą pobranego klucza publicznego (`public_key.asc`), tworząc plik `word.txt.sig`.

(e) Wyślij request do endpointa `http://127.0.0.1:5009/submit` używając meody HTTP `POST` i prześlij do serwera id sesji (jako `session_id`), plik ze słowem (jako plik, `original_file`) oraz podpisane słowo (jako plik, `signature_file`).

UWAGI:

- Aby zapisać odpowiedź do pliku z rozszerzeniem `*.zip`, użyj opcji `-o`,
- Aby utworzyć oddzielny podpis, użyj opcji `--detach-sign`,
- Aby wskazać klucz, którego będziemy używać do podpisywania, sprawdź opcję `-u`,
- Aby za pomocą narzędzia `cURL` wysłać plik do serwera, użyj składni: `-F "plik=@hello.txt"`

5.10 *Pod adresem `http://127.0.0.1:5010` działa prosty serwer HTTP udostępniający 2 endpointy: `http://127.0.0.1:5010/clearsign` oraz `http://127.0.0.1:5010/submit`.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 5010:5010 --name ex10 docker.io/mazurkatarzyna/gpg-ex10:latest
podman run -p 5010:5010 --name ex10 docker.io/mazurkatarzyna/gpg-ex10:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 5010:5010 --name ex10 ghcr.io/mazurkatarzynaumcs/gpg-ex10:latest
podman run -p 5010:5010 --name ex10 ghcr.io/mazurkatarzynaumcs/gpg-ex10:latest
```

(b) Wyślij request do endpointa http://127.0.0.1:5010/clearsign używając meody HTTP `GET` i zapisz odpowiedź do pliku z rozszerzeniem `*.zip`.

(c) Rozpakuj pobrane archiwum. W odpowiedzi od serwera otrzymałeś id sesji (jako `session_id`), klucz publiczny GPG (`public_key.asc`), klucz prywatny GPG (`private_key.asc`), oraz słowo do podpisania (w pliku `word.txt`).

(d) Podpisz plik (`word.txt`) za pomocą pobranego klucza publicznego (`public_key.asc`), tworząc plik `word.txt.asc`.

(e) Wyślij request do endpointa http://127.0.0.1:5010/submit używając meody HTTP `POST` i prześlij do serwera id sesji (jako `session_id`) oraz podpisane słowo (jako plik, `clearsigned_file`).

UWAGI:

- Aby zapisać odpowiedź do pliku z rozszerzeniem `*.zip`, użyj opcji `-o`,
- Aby utworzyć czytelny podpis, użyj opcji `--clearsign`,
- Aby wskazać klucz, którego będziemy używać do podpisywania, sprawdź opcję `-u`,
- Aby za pomocą narzędzia `cURL` wysłać plik do serwera, użyj składni: `-F "plik=@hello.txt"`

5.11 *Pod adresem http://127.0.0.1:5011 działa prosty serwer HTTP udostępniający 2 endpointy: http://127.0.0.1:5011/sign oraz http://127.0.0.1:5011/verify.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 5011:5011 --name ex11 docker.io/mazurkatarzyna/gpg-ex11:latest
podman run -p 5011:5011 --name ex11 docker.io/mazurkatarzyna/gpg-ex11:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 5011:5011 --name ex11 ghcr.io/mazurkatarzynaumcs/gpg-ex11:latest
podman run -p 5011:5011 --name ex11 ghcr.io/mazurkatarzynaumcs/gpg-ex11:latest
```

(b) Wyślij request do endpointa http://127.0.0.1:5011/sign używając meody HTTP `GET` i zapisz odpowiedź do pliku z rozszerzeniem `*.zip`.

(c) Rozpakuj pobrane archiwum. W odpowiedzi od serwera otrzymałeś id sesji (jako `session_id`), klucz publiczny GPG (`public_key.asc`), oraz podpisane słowo (w pliku `word.txt.gpg`).

(d) Zweryfikuj podpis (plik `word.txt.gpg`) za pomocą pobranego klucza publicznego (`public_key.asc`). W przypadku, gdy weryfikacja się powiedzie, powinieneś zobaczyć adres e-mail osoby, która podpisała plik.

(e) Wyślij request do endpointa http://127.0.0.1:5011/verify używając meody HTTP `POST` i prześlij do serwera id sesji (jako `session_id`) oraz adres e-mail podpisującego (jako tekst, `signer_email`).

UWAGI:

- Aby zapisać odpowiedź do pliku z rozszerzeniem `*.zip`, użyj opcji `-o`

5.12 *Pod adresem http://127.0.0.1:5012 działa prosty serwer HTTP udostępniający 2 endpointy: http://127.0.0.1:5012/detach-sign oraz http://127.0.0.1:5012/verify.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 5012:5012 --name ex12 docker.io/mazurkatarzyna/gpg-ex12:latest
podman run -p 5012:5012 --name ex12 docker.io/mazurkatarzyna/gpg-ex12:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 5012:5012 --name ex12 ghcr.io/mazurkatarzynaumcs/gpg-ex12:latest
podman run -p 5012:5012 --name ex12 ghcr.io/mazurkatarzynaumcs/gpg-ex12:latest
```

(b) Wyślij request do endpointa http://127.0.0.1:5012/sign używając meody HTTP `GET` i zapisz odpowiedź do pliku z rozszerzeniem `*.zip`.

(c) Rozpakuj pobrane archiwum. W odpowiedzi od serwera otrzymałeś id sesji (jako `session_id`), klucz publiczny GPG (`public_key.asc`), wylosowane słowo (w pliku `word.txt`) oraz podpisane słowo (w pliku `word.txt.sig`).

(d) Zweryfikuj podpis (plik `word.txt.sig`) za pomocą pobranego klucza publicznego (`public_key.asc`). W przypadku, gdy weryfikacja się powiedzie, powinieneś zobaczyć adres e-mail osoby, która podpisała plik.

(e) Wyślij request do endpointa http://127.0.0.1:5012/verify używając meody HTTP `POST` i prześlij do serwera id sesji (jako `session_id`) oraz adres e-mail podpisującego (jako tekst, `signer_email`).

UWAGI:

- Aby zapisać odpowiedź do pliku z rozszerzeniem `*.zip`, użyj opcji `-o`

5.13 *Pod adresem http://127.0.0.1:5013 działa prosty serwer HTTP udostępniający 2 endpointy: http://127.0.0.1:5013/detach-sign oraz http://127.0.0.1:5012/verify.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 5013:5013 --name ex13 docker.io/mazurkatarzyna/gpg-ex13:latest
podman run -p 5013:5013 --name ex13 docker.io/mazurkatarzyna/gpg-ex13:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 5013:5013 --name ex13 ghcr.io/mazurkatarzynaumcs/gpg-ex13:latest
podman run -p 5013:5013 --name ex13 ghcr.io/mazurkatarzynaumcs/gpg-ex13:latest
```

(b) Wyślij request do endpointa http://127.0.0.1:5013/clearsign używając meody HTTP `GET` i zapisz odpowiedź do pliku z rozszerzeniem `*.zip`.

(c) Rozpakuj pobrane archiwum. W odpowiedzi od serwera otrzymałeś id sesji (jako `session_id`), klucz publiczny GPG (`public_key.asc`) oraz podpisane słowo (w pliku `word.txt.asc`).

(d) Zweryfikuj podpis (plik `word.txt.asc`) za pomocą pobranego klucza publicznego (`public_key.asc`). W przypadku, gdy weryfikacja się powiedzie, powinieneś zobaczyć adres e-mail osoby, która podpisała plik.

(e) Wyślij request do endpointa http://127.0.0.1:5013/verify używając meody HTTP `POST` i prześlij do serwera id sesji (jako `session_id`) oraz adres e-mail podpisującego (jako tekst, `signer_email`).

UWAGI:

- Aby zapisać odpowiedź do pliku z rozszerzeniem `*.zip`, użyj opcji `-o`

5.14 *Pod adresem http://127.0.0.1:5014 działa prosty serwer HTTP udostępniający 2 endpointy: http://127.0.0.1:5014/getkey oraz http://127.0.0.1:5014/submit.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 5014:5014 --name ex14 docker.io/mazurkatarzyna/gpg-ex14:latest
podman run -p 5014:5014 --name ex14 docker.io/mazurkatarzyna/gpg-ex14:latest
```

Jeśli Docker Hub nie odpowiada, użyj obrazu zapasowego:

```
docker run -p 5014:5014 --name ex14 ghcr.io/mazurkatarzynaumcs/gpg-ex14:latest
podman run -p 5014:5014 --name ex14 ghcr.io/mazurkatarzynaumcs/gpg-ex14:latest
```

(b) Wyślij request do endpointa http://127.0.0.1:5014/getkey używając meody HTTP `GET` i zapisz odpowiedź do pliku z rozszerzeniem `*.zip`.

(c) Rozpakuj pobrane archiwum. W odpowiedzi od serwera otrzymałeś id sesji (jako `session_id`) oraz klucz publiczny serwera (`server_key.asc`).

(d) Podpisz klucz publiczny serwera swoim kluczem prywatnym. Wyeksportuj podpisany klucz serwera do pliku `signed_key.asc`.

(e) Wyślij request do endpointa http://127.0.0.1:5014/submit używając meody HTTP `POST` i prześlij do serwera id sesji (jako `session_id`), swój klucz publiczny (jako plik, `your_pubkey`) oraz podpisany swoim kluczem prywatnym klucz publiczny serwera (jako plik, `signed_key`).

5.15 Wyświetl wszystkie klucze publiczne i prywatne z Twojego keyring'a (keyring inaczej nazywany jest repozytorium kluczy programu `gpg`).

5.16 Utwórz parę kluczy GPG przy użyciu algorytmu RSA. Następnie usuń parę kluczy (publiczny i prywatny) z Twojego keyringa.

5.17 Użytkownicy systemów linuksowych mają możliwość pobrania oprogramowania z repozytoriów przygotowanych dla danej dystrybucji systemu. Niekiedy jednak dodatkowe oprogramowanie można pobrać jedynie ze strony WWW. W takim przypadku, jaka jest pewność, że plik znajdujący się na stronie, został na niej w rzeczywistości umieszczony przez dewelopera aplikacji, a nie hakera? Niektórzy programiści podpisują swoje oprogramowanie przy pomocy rozwiązań PGP (takich jak np. GPG). Dzięki temu, jako użytkownicy, mamy możliwość weryfikacji integralności oprogramowania. Proces weryfikacji jest prosty, należy:

(a) Pobrać klucz publiczny autora oprogramowania
(b) Sprawdzić fingerprint klucza
(c) Zaimportować klucz do własnego keyringa
(d) Pobrać sygnaturę dla ściąganego oprogramowania
(e) Użyć klucza publicznego do zweryfikowania podpisu

Pobierz oprogramowanie **VeraCrypt** i zweryfikuj jego integralność.

5.18 Pobierz najnowszą **wersję serwera Apache** i zweryfikuj integralność pobranego pliku. Potrzebny serwer kluczy to `pgpkeys.mit.edu`. Podpisz **klucz publiczny Apache** dwoma kluczami z własnego keyringu.

Bardzo istotna jest pewność, że klucz publiczny, który właśnie w ten czy inny sposób pozyskaliśmy, należy naprawdę do osoby, do której wydaje się należeć. Zapewnieniu tego służą certyfikaty kluczy. Certyfikowanie (podpisywanie) czyjegoś klucza to nic innego, jak sygnowanie go swoim własnym - ma to na celu potwierdzenie jego autentyczności. Jeśli użytkownik A otrzyma klucz publiczny użytkownika B bezpośrednio od niego, to zapewne jest to naprawdę klucz użytkownika B. Ale jeśli otrzymuje go od użytkownika C, któremu niekoniecznie ufa, i podejrzewa, że w rzeczywistości klucz ten może być sfałszowany przez C? Cóż, jeśli klucz ten jest certyfikowany przez pewnego innego użytkownika D (któremu A ufa, i którego klucz publiczny już ma), i sygnatura się zgadza, to A zyskuje pewność, że klucz przekazany przez C jest zgodny z oryginałem. Oczywiście A zakłada w tym momencie, że D w chwili certyfikowania tego klucza miał 100% pewności, że klucz ten należy do B - albo dlatego, że otrzymał go od B bezpośrednio, albo dlatego, że klucz był certyfikowany przez kolejną osobę, do której D miał zaufanie. Należy z tego wysnuć jeden wniosek: NIGDY nie certyfikuj cudzego klucza, jeśli nie masz pewności, że nie jest on fałszywy. W przeciwnym razie osoby, które następnie ten klucz od Ciebie otrzymają, będą sądzić, że taką pewność miałeś i używać (być może fałszywego) klucza z powodu zaufania do Ciebie.