Lab 4: Notes

```
import datetime
import re
from cryptography.hazmat.primitives import hashes, hmac
from cryptography.exceptions import InvalidSignature
from pathlib import Path

def generate_MAC(key, message):
    if not isinstance(message, bytes):
        message = message.encode()

    h = hmac.HMAC(key, hashes.SHA256())
    h.update(message)
    signature = h.finalize()
    return signature

def verify_MAC(key, signature, message):
    if not isinstance(message, bytes):
        message = message.encode()

    h = hmac.HMAC(key, hashes.SHA256())
    h.update(message)
    try:
        h.verify(signature)
    except InvalidSignature:
        return False
    else:
        return True

if __name__ == "__main__":

    """     # 1) Sign the file content.    # koristimo izraz "sign" iako ne koristimo public encryption (za koji smo vezali pojam potpisiva

    # 1.1 Read the file content.
    with open("message.txt", "rb") as file:
        content = file.read()

    # 1.2 Sign the content.
    key = "my super secure secret".encode()
    signature = generate_MAC(key=key, message=content)

    # 1.3 Save the signature into a file.
    with open("message.sig", "wb") as file:
        file.write(signature) """

    """     # 2) Verify message authenticity.

    # 2.1 Read received file content.
    with open("message.txt", "rb") as file:
        content = file.read()

    # 2.2 Read the received signature.
    with open("message.sig", "rb") as file:
        signature = file.read()

    # 2.3.1 Sign the received file.
    # 2.3.2 Compare locally generated signature with the received one.
    key = "my super secure secret".encode()
    is_authentic = verify_MAC(key=key, signature=signature, message=content)
    print(f"Message is {'OK' if is_authentic else 'NOK'}") """

    PATH = "challenges/g2/pupacic_karla/mac_challenge/"
    KEY = "pupacic_karla".encode()
    authentic_messages = []

    for ctr in range(1, 11):
        msg_filename = f"order_{ctr}.txt"
        sig_filename = f"order_{ctr}.sig"

        msg_file_path = Path(PATH + msg_filename)
        with open(msg_file_path, "rb") as file:
            message = file.read()
```

```
        sig_file_path = Path(PATH + sig_filename)
        with open(sig_file_path, "rb") as file:
            signature = file.read()

        is_authentic = verify_MAC(key=KEY, signature=signature, message=message)

        # print(f'Message {message.decode():>45} {"OK" if is_authentic else "NOK":<6}')
        if is_authentic:
            authentic_messages.append(message.decode())

        authentic_messages.sort(key=lambda m: datetime.datetime.fromisoformat(re.findall(r'\(.*?\)', m)[0][1:-1]))

        for m in authentic_messages:
            print(f'Message {m:>45} {"OK":<6}')
```

- demonstracija korištenja MAC mehanizma (sa simetričnim ključevima) za zaštitu integriteta poruka i autentikaciju

1. zadatak:

- stvaramo txt file s porukom koju želimo sakriti
- unutar python datoteke čitamo sadržaj tajne poruke
- hashiramo tu poruku MAC funkcijom (generate_MAC) —> potpis
- povezujemo potpis s porukom
- zatim trebamo verificirati integritet poruke, pa:
- ponovno čitamo sadržaj tajne poruke
- čitamo sadržaj datoteke koja sadrži potpis
- hashiramo važno poruku još jednom
- uspoređujemo dva potpisa (verify_MAC) koja smo generirali ovim postupkom —> ako su isti, integirtet je u redu

2. zadatak:

- svakome je dodjeljen njegov osobni zadatak
- provjeravamo ispravan vremenski slijed dobivenih poruka o kupnji/prodaji dionica
- provjeravamo svaku poruku slično kao u 1. zadatku
- sortiramo po timestampu