Sigurnost računala i podataka - Lab 4.

Zadatak 1.

```
from cryptography.hazmat.primitives import hashes, hmac
from cryptography.hazmat.primitives import hashes, hmac
from cryptography.exceptions import InvalidSignature

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
#     # 1.1 Read the file content

#     # Reading from a file
#     with open("message.txt", "rb") as file:
#         content = file.read()

#     print(content)

#     # 1.2 Sign the content

#     key = "my super secure secret".encode() #pretvara niz u bajtove tj enodira
#     signature = generate_MAC(key = key, message = content) #ako vise puta pozovemo, uvik ce se isti generirat jer je ista hash funkcija i
#     print(signature)

#     # 1.3 Save the signatur into a file

#     with open("message.sig", "wb") as file:
#         file.write(signature)

    # 2. Verify message authenticity
    # 2.1 Read the received file

    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'}")
```

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

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
#     # 1.1 Read the file content

#     # Reading from a file
#     with open("message.txt", "rb") as file:
#         content = file.read()

#     print(content)

#     # 1.2 Sign the content

#     key = "my super secure secret".encode() #pretvara niz u bajtove tj enodira
#     signature = generate_MAC(key = key, message = content) #ako vise puta pozovemo, uvik ce se isti generirat jer je ista hash funkcija i
#     print(signature)

#     # 1.3 Save the signatur into a file

#     with open("message.sig", "wb") as file:
#         file.write(signature)

    # 2. Verify message authenticity
    # 2.1 Read the received file

    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'}")


    # 2. Zadatak

    # wget -r -nH -np --reject "index.html*" http://challenges.local/challenges/g2/bonic_paula/mac_challenge/
    # s prethodnom linijom smo skinuli datoteke

    PATH = "challenges/g2/bonic_paula/mac_challenge/"
    KEY = "bonic_paula".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}')
```

Sažetak:

Da bi dokazali autentičnost poruke možemo je potpisati. Prvo trebamo pročitati file, napraviti potpis te ga spremiti u taj file. Zatim ga pošsaljemo na odredište. Druga strana ga pročita, generira lokalni potpis te uspoređuje dani i generirani potpis. Ako su potpisi isti onda je poruka autentična.

Proces:

Zaštita poruke pomoću MAC-a, Koristili HMAC iz Python biblioteke cryptography. U novoj datoteci spremili poruku, a u novom python sriptu kod našeg programa.

Učitali sadržaj datoteke s porukom, pomoću funkcije dobili potpis i spremili ga u odvojenu datoteku.

Pročitali file, uz pomoć iste funkcije generirali potpis te usporedili s fileom gdje je signature-

Bitno je utvrditi vremenski redoslijed poslanih poruka, a to možemo označavanjem jer ako se poruka izgubi/kasnije dođe gubi se značenje.