SRP Vježba 3

Cilj vježbe: praktična primjena teorijskih znanja o autentikaciji poruka i integritetu koristeći simetričnu kriptografiju.

Zadatak 1: implementirati zaštitu integriteta poruke koristeći simetričnu kriptografiju tj. *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, message, mac):
    if not isinstance(message, bytes):
        message = message.encode()
    h = hmac.HMAC(key, hashes.SHA256())
    h.update(message)
    try:
        h.verify(mac)
    except InvalidSignature:
        return False
    else:
        return True

if __name__ == "__main__":
  key = b"my secret key"
  msg_filename = "message.txt"
  mac_filename = "message.mac"
  with open(msg_filename, "rb") as file:
    content = file.read()
  with open(mac_filename, "rb") as file:
    mac = file.read()
  is_valid = verfiy_MAC(key, content, mac)
  print(is_valid)
  # mac = generate_MAC(key, content)
  # with open(mac_filename, "wb") as file:
  # file.write(mac)
```

Objašnjenje: prvo smo napravili datoteku **message.txt** u kojoj je naša poruka, koja u ovom slučaju nije tajna. Nakon toga deklarirali smo naš tajni ključ s kojim generiramo M*essage Authentication Code* uz pomoć funkcije *generate_MAC* koja prima ključ (simetrični) i samu poruku. Taj *MAC* se spremi u datoteku. Kasnije samo uz pomoć funckije *verify_MAC* provjerimo jesu li MAC-ovi isti tako da generiramo jedan lokalno i usporedimo s primljenim. Ako je napadač nešto promjenio dobivamo grešku i poruka se odbacuje.

Zadatak 2: utvrditi vremensku ispravnost poruka o prodaji dionica.

```
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, message, mac):
    if not isinstance(message, bytes):
        message = message.encode()
    h = hmac.HMAC(key, hashes.SHA256())
    h.update(message)
    try:
        h.verify(mac)
    except InvalidSignature:
        return False
    else:
        return True

if __name__ == "__main__":
  key = b"my secret key"
  msg_filename = "message.txt"
  mac_filename = "message.mac"
  with open(msg_filename, "rb") as file:
    content = file.read()
  with open(mac_filename, "rb") as file:
    mac = file.read()
  is_valid = verfiy_MAC(key, content, mac)
  print(is_valid)
  # mac = generate_MAC(key, content)
  # with open(mac_filename, "wb") as file:
  # file.write(mac)
  challenge_key = "prezime_ime".encode()
```

```
print("Security key for challenge:", challenge_key)
for ctr in range(1, 11):
  chg_msg_filename = f"challenges\prezime_ime\mac_challenge\order_{ctr}.txt"
  chg_sig_filename = f"challenges\prezime_ime\mac_challenge\order_{ctr}.sig"
  #print(chg_msg_filename)
  #print(chg_sig_filename)
  with open(chg_msg_filename, "rb") as file:
    challenge_content = file.read()
  with open(chg_sig_filename, "rb") as file:
    challenge_mac = file.read()
  is_authentic = verify_MAC(challenge_key, challenge_content, challenge_mac)
  print(f'Message {challenge_content.decode():>45} {"OK" if is_authentic else "NOK":<6}')
```

Objašnjenje: u folderu *mac_challenge* nalazi se set od 10 deset poruka i potpisa. Nekima od poruka promjenjen je sadržaj vremenski. Naš kod otkriva koje su to poruke tako da provjerava lokalni MAC i s naše strane generirani.

Vidimo da je simetrična kriptografija iznimno sigurna jer svaka promjena u poslanoj poruci ili *MAC-u* rezultira odbacivanjem same poruke.