Laboratorijska vježba 4 - Message authentication and integrity

U ovoj laboratorijskoj vježbi smo primjenili teoretske spoznaje o osnovim kriptografskim mehanizmima za autentikaciju i zaštitu integriteta poruka. Koristili smo simetrične i asimetrične kriptografske mehanizme: *message authentication code* (MAC) koje se temelje na podijeljenom simetričnom ključu i *digitalne potpise* zasnovane na javnom ključu.

Zadatak 1 - Message authentication code (MAC)

U prvom zadatku smo koristili kriptografski mehanizam message auth. code (MAC) kako bi implementirali zaštitu integriteta sadržaja poruke primjenom odgovarajućeg MAC algoritma. Kako bi to implementirali, trebali smo napraviti 4 koraka;
1.1 Pročitati file

1.2 Generirati tajnu, tj. tajni ključ

1.3 Zapravo potpisati poruku

1.4 Spremiti potpis/MAC/autentikacijski tag u poseban file.

Ispod se nalazi odgovarajući kod koji smo koristili sa svim koracima:

```
# 1. Zastititi poruku(potpisivanje poruke)- sign the message
   # 1.1 Read the file content
       # Reading from a file

      with open("message.txt", "rb") as file:
          message = file.read()

      print(message)
   # 1.2 Generate signing key (secret)

      key = "my super secret".encode()

   # 1.3 Actually sign the message

      signature = generate_MAC(key=key, message=message)
```

```
        print(signature)

    # 1.4 Save the signature/MAC/auth_tag into a separate file

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

Učitali smo poruku čiji integritet želimo zaštititi. Izračunali smo odgovarajuću MAC vrijednost za zadani file funkcijom generate_MAC i spremili tu vrijednost u poseban file. Kako bi provjerili integritet poruke, učitali smo poruku i potpis. Za učitanu poruku smo ponovno izračunali MAC vrijednost te smo tu vrijednost usporedili s učitanim potpisom. Ako su vrijednosti jednake, integritet poruke je očuvan.

Zadatak 2

U drugom zadatku nam je bio cilj utvrditi vremenski ispravnu/autentičnu sekvencu transakcija dionica. Sa servera smo preuzeli 10 transakcija i njihove autentikacijske kodove (MAC tagove) kojima smo trebali provjeriti autentičnost. Postupak je sličan kao u prvom zadatku, samo što smo ovdje imali 10 transakcija, tj. fileova pa je taj postupak trebalo napraviti 10 puta. Učitavali smo redom transakciju i njen MAC tag i uspoređivali ih. Na kraju smo sve autentične poruke pohranili u niz messages koji smo još dodatno sortirali po timestamp-u.

Cijeli kod:

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

from pathlib import Path
import re # regular expression regularni izrazi
import datetime



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. Zastititi poruku(potpisivanje poruke)- sign the message
    # # 1.1 Read the file content
    # # Reading from a file

     with open("message.txt", "rb") as file:
         message = file.read()

     #print(message)
    # # 1.2 Generate signing key (secret)

     key = "my super secret".encode()

    # # 1.3 Actually sign the message

     signature = generate_MAC(key=key, message=message)
    # print(signature)

    # # 1.4 Save the signature/MAC/auth_tag into a separate file

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

    # 2. Verify message authenticity
    # 2.1 Read the message file content and the signature

     with open("message.txt", "rb") as file:
         message = file.read()

     with open("message.sig", "rb") as file:
         signature = file.read()

    # # 2.2 Get the signing key

     key = "my super secret".encode()

    # # 2.3 Sign the message and compare locally generated MAC with the received one

     is_authentic = verify_MAC(key=key, signature=signature, message=message)
     print(f"Message is ('OK' if is_authentic else 'NOK')")

    # procitali imena fileova

    PATH = "challenges/g3/granic_andjela/mac_challenge/"
    KEY = "granic_andjela".encode()

    messages = []
```

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

        msg_file_path = Path(PATH + msg_filename)
        sig_file_path = Path(PATH + sig_filename)

        with open(msg_file_path, "rb") as file:
            message = file.read()

        with open(sig_file_path, "rb") as file:
            signature = file.read()

        #print(message)

        #is_authentic = ...

        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:
            messages.append(message.decode())

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

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

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