3 lab. vjezba

MAC

1 izazov

Pokusat cemo zastiti integritet dane poruke pomocu MAC-a.

Koristili smo HMAC mehanizam iz python biblioteka criptography

U lokalnom direktoriju kreiramo datoteku ciji sadrzaj zelimo zastititi i nazovemo je message.txt i datoteku message_integrity.py

Izracunavamo MAC za danu poruku:

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

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
```

MAC izracunavamo tako sto generiramo tajni kljuc kroz funkciju generate_MAC(key, message) koja prima kljuc i poruku.

Dobiveni MAC upisujemo u datoteku message.sig.

Funkcija za provjeru ispravnosti MAC-a

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


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
```

Kako bi provjerili jeli poruka autenticna koristimo funkciju verify_MAC(key, signature, message). Argumenti (key, signature,message) su nam potrebni da bi mogli lokalno generirati MAC i usporediti s onim prije. Ako poruka nije mijenjana trebali bi dobiti True, a ako jeste False

if __name__ == "**main**":
  key = b"my secret key"

  with open("message.txt", "rb") as file: # ucitavamo datoteku
   content = file.read()
                                # mac = generate_MAC(key, content)
                                # with open("message.sig", "wb") as file:
                                # file.write(mac)
  with open("message.sig", "rb") as file:
  mac = file.read()
  is_authentic = verify_MAC(key, mac, content)
  print(is_authentic)

2 izazov

Želimo ustvrditi ispravni vremenski redoslijed transakcija dionica. Digitalno potpisane naloge preuzimamo preko servera

Kljuc koji koristimo je key="vulevic_natasa".encode()

Želimo napraviti kod koji ce vrtiti petlju koja ce izlistavati prijedloge za kupovinu kronoloski. Otvaramo sve datoteke te preko verify_MAC provjeravamo autenticnost parova .txt i .sig datoteka.

One koje su autenticne cemo dodati u listu koju poredamo vremenski.

Kod:

```
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__":

    for ctr in range(1,11):
        msg_filename = f"order_{ctr}.txt"
        sig_filename = f"order_{ctr}.sig"
        with open(msg_filename, "rb") as file:
            content = file.read()
        with open(sig_filename, "rb") as file:
            signature = file.read()

        key = "vulevic_natasa".encode()
        is_authentic = verify_MAC(key, signature, content)

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

Digitalni potpisi koristenjem javnog kljuca

U ovom izazovu treba odrediti autenticnu sliku izmedu dvije ponudene koje su potpisane privatnim kljucem. Mi cemo iskoristit javni kljuc tako sto cemo s njim dekriptirati .sig datoteku. Dobit cemo hash vrijednost koja je identicna ili ne onoj koju dobijemo kada hashiramo odgovarajucu sliku.

Kod:

```
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
from cryptography.exceptions import InvalidSignature

def load_public_key():                               # ucitavamo javni kljuc iz datoteke
    with open("public.pem", "rb") as f:
        PUBLIC_KEY = serialization.load_pem_public_key(
            f.read(),
            backend=default_backend()
        )
    return PUBLIC_KEY

def verify_signature_rsa(signature, message):   #provjeravamo ispravnost dig potpisa
    PUBLIC_KEY = load_public_key()
    try:
        PUBLIC_KEY.verify(
            signature,
            message,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )

    except InvalidSignature:
        return False
    else:
        return True

if __name__ == "__main__":

    public = load_public_key();

    print(public)

if __name__ == "__main__":

    with open("image_1.png", "rb") as file:
        image = file.read()
    with open("image_1.sig", "rb") as file:
        signature = file.read()

    isAuthentic = verify_signature_rsa(signature, image)
    print(isAuthentic)

    with open("image_2.png", "rb") as file:
        image2 = file.read()
    with open("image_2.sig", "rb") as file:
        signature2 = file.read()

    isAuthentic2 = verify_signature_rsa(signature2, image2)

    print(isAuthentic2)
```