Treći izvještaj

Message authentication and integrity

Zadatak 1

Na početku vježbe kreiramo virtualno okruženje u Pythonu, unutar kojeg koristimo biblioteku cryptography. Trebamo zaštititi integritet poruke primjenom MAC algoritma.

```
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)
    mac = h.finalize()
    return mac

def verify_MAC(key, mac, message):
    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"Ooooooogromna tajna"

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

    # mac = generate_MAC(key, message)

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

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

```
    isAuthentic = verify_MAC(key, mac, message)

    print(isAuthentic)
```

2.zadatak

U ovom izazovu se utvrđuje ispravan vremenski redoslijed transakcija.

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

def generate_MAC(key, message):

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

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

  return mac

def verify_MAC(key, mac, message):

  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

def extractOrderDateTime(order):

  first = order.index('(')
  second = order.index(')')
  orderDate = order[first+1: second]

  return datetime.strptime(orderDate, "%Y-%m-%dT%H:%M")
```

```
def checkIfAuthentic(path, key):

  valid = []

  for ctr in range(1,11):

    msgFile = os.path.join(path, f"order_{ctr}.txt")
    sigFile = os.path.join(path, f"order_{ctr}.sig")

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

    with open(sigFile, "rb") as file:
      mac = file.read()

    is_authentic = verify_MAC(key, mac , message)

    if is_authentic:
      valid.append(message.decode("utf-8"))
      valid.sort(key = extractOrderDateTime)

  print(valid)

if __name__ == "__main__":

  key = "kevric_laura".encode()
  path = os.path.join("challenges","kevric_laura", "mac_challenge")
  checkIfAuthentic(path,key)
```

Digital signatures using public-key cryptography

U ovom izazovu trebamo odrediti koja je od dvije slike autentična, koju je profesor potpisao svojim privatnim ključem. Iz repozitorija smo preuzeli slike, odgovarajuće digitalne potpise i javni ključ.

Za rješavanje smo koristili **RSA** kriptosustav.

Ključ učitavamo, pa ga *deserijaliziramo*:

```
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.backends import default_backend
```

```
def load_public_key():
    with open(PUBLIC_KEY_FILE, "rb") as f:
        PUBLIC_KEY = serialization.load_pem_public_key(
            f.read(),
            backend=default_backend()
        )
    return PUBLIC_KEY
```

Ispravnost javnog ključa provjeravamo naredbom:

```
print(load_public_key())
```

Zatim provjeravamo ispravnost digitalnog potpisa:

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


def verify_signature_rsa(signature, message):
    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
```

Učitavamo slike i potpise:

```
with open("image1.png", "rb") as file:
    image = file.read()
with open("image1.sig", "rb") as file:
    signature = file.read()
```

Provjeravamo autentičnost slike:

```
is_authentic = verify_signature_rsa(signature, image)
print(is_authentic)
```