SRP Vjezbe3

Cilj vjezbe je prakticna primjena teorijskih znanja o mehanizmima za autentikaciju i zaštitu integriteta poruka koristeći simetričnu kriptografiju

Zadatak 1

Implementirati zaštitu integriteta sadržaja poruke primjenom odgovarajućeg *message authentication code (MAC)* algoritma. Koristite pri tome HMAC mehanizam iz Python biblioteka `cryptography`.

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

U datoteku message.txt upisujemo poruku kojoj zelimo zastiti integritet. Poruka u ovom slucaju nije sifrirana, moze biti javna poruka. Nakon toga deklariramo tajni kljuc s kojim generiramo Message Authentication Code pomocu generate_MAC funkcije. Funkcija generate_MAC prima tajni kljuc i poruku koju zelimo zastititi i generira MAC. Pomocu funkcije verify_MAC provjeravamo jesu li MAC-ovi isti tako da generiramo jedan lokalno i usporedimo s primljenim. Ako je napravljena promjena, biti ce detektirana i MAC nece biti prihvacen tj. poruka se odbacuje

Zadatak 2

Utvrditi vremenski ispravnu/autentičnu skevencu transakcija (ispravan redosljed transakcija) dionicama.

Autenticirani nalozi transakcija (primjenom MAC-a) nalaze se na lokalnom poslužitelju: http://challenges.local koje preuzimamo pomocu naredbe

```
#wget.exe -r -nH -np --reject "index.html*"http://challenges.local/challenges/<id_grupe>/<prezime_ime>/mac_challenge/
wget.exe -r -nH -np --reject "index.html*" http://challenges.local/challenges/g3/maretic_josip/mac_challenge/
```

Preuzete naloge transakcija zatim provjeravamo slicno kao u prethodnom zadatku. Na kraju koda autenticne poruke se sortiraju po datumu.

```
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}')

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

      for x in messages:
          print(x)
```