Izvještaj laboratorijske vježbe

3. Message authentication and integrity

Zadatak

Koristili smo konceptualne spoznaje o osnovnim kriptografskim mehanizmima za autentikaciju i zaštitu integriteta poruka u praktičnim primjerima. Riješili smo 2 izazova vezana uz simetrični kriptografski mehanizam **"Message authentication code" (MAC)** te 1 izazov vezan uz asimetrični kriptografski mehanizam **"Digital signature"**, zasnovan na primjeni javnog ključa.

Izazov 1

Implementirali smo zaštitu integriteta sadržaja poruke primjenom MAC algoritma koristeći HMAC mehanizam iz biblioteke **"cryptography"**.

- Prvo smo kreirali našu poruku i pohranili je u direktorij gdje se nalazi i python datoteka

```
Secret message of mine!
```

- Zatim smo za danu poruku izračunali MAC vrijednost funkcijom **"generate_MAC"** koja za ulaz uzima tajni ključ i sadržaj poruke. Funkcija vraća MAC odnosno **autentikacijski tag** dobiven iz **SHA256** hash funkcije za dane parametere. Tajni ključ korišten u primjeru je **kratak** i **male entropije**!

```
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

if __name__ == "__main__":
    key = b"Secret_key"

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

    mac = generate_MAC(key, content)

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

- Na kraju smo varijablu mac iz gornjeg primjera spremili u datoteku "message.sig" sljedećeg formata

```
B�z�M�q����
�@��T
Y���[#�F��
```

- U drugom dijelu izazova provjeravali smo validnost generiranog MAC-a funkcijom **"verify_MAC"** koja za ulaz prima tajni ključ, autentikacijski tag(MAC) te sadržaj poruke. Za tajni ključ i sadržaj poruke izračuna se **lokalni MAC** i metodom "verify" uspoređuje se sa onim **primljenim MAC-om** (u ovome slučaju to je varijabla **signature**). Funkcija vraća False, ukoliko je integritet narušen i True, ukoliko smo primili originalnu poruku.

```
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

if __name__ == "__main__":
    key = b"secret_key"

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

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

    is_valid = verify_MAC(key, signature, content)
    print(is_valid)
```

- Gore geneirani kod u python virtualnom okruženju ispisao je **"True"**. Stoga smo radi provjere poruku prvo otvorili u heksadecimalnom formatu i modificirali je, tj. promijenili njen originalni sadržaj (**m** smo pretvorili **D**).

```
53 65 63 72 65 74 20 44 65 73 73 61 67 65 20 6F 66 20 6D 69 6E 65 21
S  e  c  r  e  t     D  e  s  s  a  g  e     o  f     m  i  n  e  !
```

Naravno sada smo za ispis očekivano dobili **"False"**.

Izazov 2

U ovome izazovu utvrdili smo vremenski ispravnu sekvencu transakcija sa odgovarajućim dionicama "Tesle". Transakcije kao i njihove digitalne potpise (potpisane primjenom MAC-a) preuzeli smo sa lokalnog servera, gdje je pojedinim transakcijama bio narušen integritet.

- Programski smo ovo riješili slično kao izazov 1. Tajni ključ zadan kao b"prezime_ime" nije siguran ni u ovom primjeru. Provjera validnosti digitalnih potpisa ista je kao kod provjere autentikacijskih tagova u izazovu 1. Ovdje smo samo upotrijebili for petlju kako ne bismo ručno provjeravali svaku transakciju zasebno i pojedinačno otvarali potrebne datoteke. Dakle, jedna iteracija formatira imena datoteka poruka i potpisa te ih sprema u zasebne varijable **"msg_filename"** i **"sig_filename"**. Nakon toga, otvaramo datoteke koristeći te varijable i spremamo sadržaje poruke i potpisa u varijable **"content"** i **"signature"** koje uz ključ šaljemo u funkciju **"verify_MAC"** koja vraća True ili False, ovisno o integritetu poruke i rezultat sprema u varijablu **"is_authentic"**. Na kraju samo formatiramo ispis tako što ispišemo sadržaj transakcije i poruku OK, ako su sačuvani integritet i autentičnost te NOK, ako su isti ugroženi.

```
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

if __name__ == "__main__":
    key = b"pijuk_mario"

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

        with open(msg_filename, "rb") as file:
            content = file.read()

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

        is_authentic = verify_MAC(key, signature, content)

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

- Kada u terminalu pokrenemo gore navedeni kod za rezultat dobijemo sljedeći sadržaj. Sada "broker na Wall Street-u" zna kako postupiti sa dionicama i koje transakcije su djelo zlonamjernog napadača.

```
(lab3) C:\Users\MARIO\Desktop\SRP\lab3>python message_integrity.py
order_1.txt
order_1.sig
Message    Sell 97 shares of Tesla (2021-11-08T09:50) NOK
order_2.txt
order_2.sig
Message     Buy 15 shares of Tesla (2021-11-05T07:30) NOK
order_3.txt
order_3.sig
Message    Sell 51 shares of Tesla (2021-11-04T21:05) NOK
order_4.txt
order_4.sig
Message    Sell 42 shares of Tesla (2021-11-10T06:35) NOK
order_5.txt
order_5.sig
Message    Sell 21 shares of Tesla (2021-11-06T17:52) OK
order_6.txt
order_6.sig
Message    Sell 81 shares of Tesla (2021-11-06T12:43) OK
order_7.txt
order_7.sig
Message    Sell 59 shares of Tesla (2021-11-08T11:58) OK
```

```
order_8.txt
order_8.sig
Message    Sell 77 shares of Tesla (2021-11-09T19:45) NOK
order_9.txt
order_9.sig
Message    Sell 58 shares of Tesla (2021-11-08T00:05) OK
order_10.txt
order_10.sig
Message     Buy 87 shares of Tesla (2021-11-06T18:49) OK
```

Izazov 3

U ovome izazovu odredili smo autentičnu sliku od dvije ponuđene koristeći asimetrični kriptografski mehanizam, točnije **RSA** sustav iz biblioteke cryptography. Autentična slika potpisana je od strane profesora njegovim **privatnim ključem** za kojeg smo znali **javni ključ** i isti iskoristili za utvrđivanje validnosti potpisa odnosno autentičnosti slike. Po ovome principu **Certificate Authority** stranke privatnim ključem potpisuju certifikate u kojima jednoznačno spajaju javni ključ i njegovog vlasnika, dok stranka koja verificira certifikat koristi javni ključ CA-ja i tako provjerava njegovu autentičnost.

- Digitalne potpise verificirali smo funkcijom **"verify_signature_rsa"** koja za ulaz ima digitalni potpis i sadržaj poruke (u ovome slučaju je to slika). Prvo se u funkciji serijalizira javni ključ, zatim se metodom **"verify"** validira autentičnost poruke, ali ovoga puta koristeći RSA sustav. Funkcija vraća **True** ili **False**.

```
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():
    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):
    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

print(load_public_key())

with open("image_2.png", "rb") as file:
    image = file.read()

with open("image_2.sig", "rb") as file:
    signature = file.read()

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

- Gore navedeni program u terminalu za sliku potpisanu od strane profesora vraća True, dok za drugu sliku vraća False. Također, za bilo koju drugu kombinaciju **slika-potpis** vraća se False.

```
(lab3) C:\Users\MARIO\Desktop\SRP\lab3>python digital_signature.py
<cryptography.hazmat.backends.openssl.rsa._RSAPublicKey object at 0x00000193A6C19A60>
True

(lab3) C:\Users\MARIO\Desktop\SRP\lab3>python digital_signature.py
<cryptography.hazmat.backends.openssl.rsa._RSAPublicKey object at 0x000002A94C9D9A60>
False
```

Zaključak

Cilj vježbe bio je na primjerima pokazati kako simetrični i asimetrični kriptografski sustavi čuvaju autentičnost i integritet poruke. Kada koristimo MAC koncept potrebno je

voditi računa o tome da tajni ključ bude visoke entropije i distribuiran na siguran način. S druge strane kod koncepta digitalnih potpisa bitno je da imamo treću stranku kojoj doista možemo vjerovati. Stoga je bitno odabrati kriptografski mehanizam koji će najbolje odgovarati postavljenim zahtjevima sigurnosti te isti implementirati na pravi način.