IZVJEŠTAJ 4.LABORATORIJSKE VJEŽBE

Na 4. laboratorijskoj vježbi smo se dotakli teme autentifikacije poruke i integriteta iste te kako ih zaštititi koristeći MAC (Message Authentication Code). U ovoj vježbi, naš je cilj primjeniti teorijsku spoznaju o kriptografiji tj. kako funkcionira MAC algoritam, kako se generira ključ, generiraranje potpisa, *timestampova* i slično. U vježbi smo obradili ukupno 2 zadatka.

1.ZADATAK

U ovome zadatku, primjenom MAC algoritma smo htjeli zaštititi integritet same poruke. Integritet štitimo na način da pomoću MAC algoritma generiramo digitali potpis (odnosno MAC vrijednost) kojim potvrđujemo da je poruka poslana od pravog *usera*, sa nama prihvatljivim kašnjenjem te da sama poruka nije mijenjana. MAC algoritam to rješava na način da pomoću odabranog ključa i poruke generira taj potpis. Taj potpis 'zakačimo' na našu originalnu poruku i ta poruka ide na javni kanal (mi smo to 'kačenje' banalizirali na način da smo pohranili MAC vrijednost u drugi file). Kada primetalj primi tu istu poruku, primjeljuje MAC algoritam sa ključem (istim kod simetrične kriptografije, a *public keyem* od pošiljatelja kod asimetrične kriptografije) te uspoređuje potpis koji je primio sa onim koji je on generirao. Ako se poklapaju, poruci je zaštićen integritet, a ako je nešo mijenjano na kanalu, potpisi će se razlikovati.

Primjenu ovoga smo napravili na slijedeći način: lokalno smo kreirali vlastiti file u kojega smo spremili poruku čiji smo integritet htjeli zaštititi te smo sadražaj tog filea učitali u memoriju. Nakon toga, iz sadržaja tog filea te našeg odabranog ključa koji smo *hardcodirali* u *sorce code*, kreiramo potpis kojega smo pohranili u drugi file. Mala digrasija, NIKADA nije dobro *hardcodirati* neku vrstu ključa ili passworda unutar samog *sorce coda* jer ako poznajemo algoritam kojim je nešto ili enkriptirano ili *hashirano* ili ako poznajemo MAC algoritam, tada je naša prethodna komunikacija i sve buduće komunikacije ugrožene.

Nadalje, kako bismo testirali našu zaštitu, obavili smo verifikaciju tog generiranog potpisa na način da bi ili promijenili file ili sam potpis. Kada bismo promijenili file, MAC bi bio drukčiji jer MAC algoritam koristi sadržaj samog *filea* prilikom generacije MAC vrijednost tj. potpisa, a kada bismo promjenili sam popis tj. MAC vrijednost, vrijednost se nebi poklapale iz očitih razloga. Kôd koji smo koristili za ovaj zadatak je slijedeći:

CODE

```
# GOAL: Protect file content authenticty using MAC primitive

# Signing Proces
# 1. Load content from file     DONE
# 2. Sign the content of the given file using MAC primitive     DONE
# 3. Make separate file with signature      DONE

# Verification
# 1. Read from file     DONE
# 2. Read signature     DONE
# 3. Sign message conten using MAC primitive        DONE
# 4. Compare gemerated signature with the one we recived        DONE


from cryptography.hazmat.primitives import hashes, hmac
from cryptography.exceptions import InvalidSignature
from pathlib import Path
import re
import datetime

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)     #veryfies on safe way, do NOT use hmac == etc
    except InvalidSignature:
        return False
    else:
        return True

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__":

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

    key = "my super secure secret".encode()

    sig = generate_MAC(key, message)

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

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

```
    message = file.read()

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

key = "my super secure secret".encode()

is_authentic = verify_MAC(key, sig, message)

print(f"Message is {'OK' if is_authentic else 'NOT OK'}")
```

2.ZADATAK

U 2.zadatku smo se bavili vremenskim ispravnim/autentičnim sekvencama transakcija dionica. Svaki od studenata je dobio svoju sekvencu MAC autenticiranih transakcija za koje smo provjeravaili da li su same transakcije autentične. Na ne baš siguran način smo generirali ključ kojim ćemo generirati MAC vrijednost (koristili smo enkodirano ime i prezime) te smo kroz petlju generirali MAC vrijednosti koje smo uspoređivali sa MAC vrijednostima koje smo preuzeli sa lokalnog poslužitelja.

Ako smo naišli na neke nepravilnosti, npr. mjenjanje sadržaja, *timestampa* ili bilo čaga drugog bi rezlultiralo nepravilnom MAC vrijednošću te ta poruka nebi bila autentična. Na kraju smo sve te poruke (autentične) pohranili i sortirali prema *timestampu*. Kôd koji smo koristili za ovaj zadatak je slijedeći:

CODE

```
# GOAL: Protect file content authenticty using MAC primitive


# Signing Proces
# 1. Load content from file     DONE
# 2. Sign the content of the given file using MAC primitive     DONE
# 3. Make separate file with signature      DONE

# Verification
# 1. Read from file     DONE
# 2. Read signature     DONE
# 3. Sign message conten using MAC primitive        DONE
# 4. Compare gemerated signature with the one we recived        DONE


from cryptography.hazmat.primitives import hashes, hmac
from cryptography.exceptions import InvalidSignature
from pathlib import Path
import re
import datetime

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)     #veryfies on safe way, do NOT use hmac == etc
    except InvalidSignature:
        return False
    else:
        return True

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 = "juric-pesic_mijo".encode()

    PATH = "challenges/g1/juric-pesic_mijo/mac_challenge/"

    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:
            sig = file.read()

        is_authentic = verify_MAC(key, sig, 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}'
```