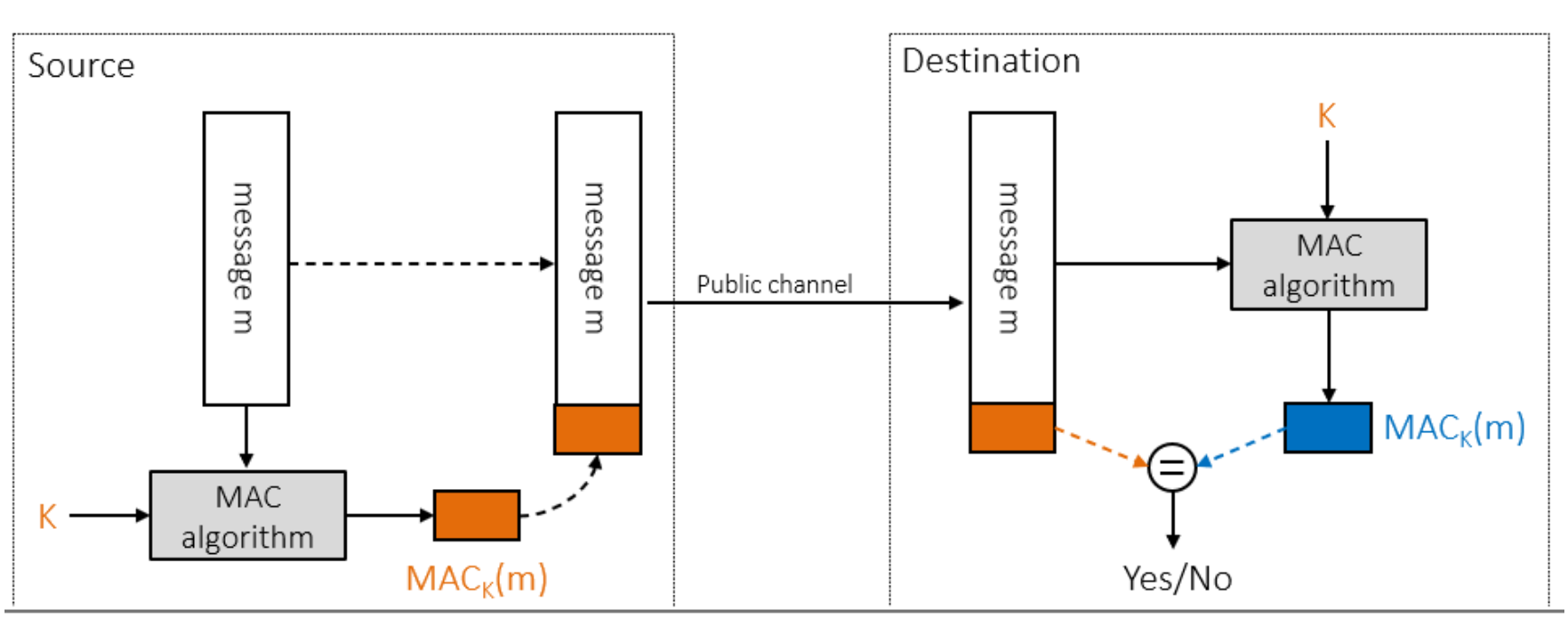
Lab 4: Message authentication and integrity

Cilj vježbe je primjeniti teoretske spoznaje o osnovnim kritografskim mehanizmima za autentikaciju i zaštitu integriteta poruka u praktičnim primjerima. Pri tome ćemo koristiti simetrične i asimetrične kriptografske mehanizme: *message authentication code (MAC)* zasnovane na simetričnim ključevima i *digitalne potpise* zasnovane na javnim ključevima.

Zadatak 1

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

Prvo što smo napravili u ovom zadatku je tekstualna datoteka koju smo nazvali “message.txt”. Sadržaj ove datoteke je poruka čiji integritet želimo zaštititi. To ćemo postići primjenom MAC algoritma kojim štitimo poruku od aktivnih napada. Kako bismo zaštitili poruku koristit ćemo digitalni potpis kojim primatelj može potvrditi da je primljena poruka autentična te da je izvor poruke autentičan, odnosno da je poruka nepromjenjenog sadržaja i da je došla od ispravnog pošiljatelja (od onog od kojeg očekujemo poruku). Na slici ispod vidimo generalnu ideju kako bi naš postupak trebao izgledati.



▼ Glavni elementi:

- **message m** - originalna poruka koju šalje pošiljatelj
- **K** - tajni ključ kojeg znaju i pošiljatelj i primatelj poruke
- **MAC algorithm** - algoritam koji prima originalnu poruku i tajni ključ i na temelju toga generira potpis
- **MACk(m)** - potpis

Zadatak ćemo podijeliti u dva dijela:

- **Potpisivanje poruke (Signing process)** - lijevi dio sa slike
- **Provjera autentičnosti poruke (Verification process)** - desni dio sa slike

Za rješavanje zadatka koristit ćemo funkciju za izračun MAC vrijednosti za danu poruku i funkciju za provjeru validnosti MAC-a 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
```

```
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
```

Potpisivanje poruke

Kako bismo potpisali poruku prvo trebamo pročitati datoteku u kojoj se nalazi poruka, a to je u našem slučaju tekstualna datoteka “message.txt”. Nakon toga trebamo potpisati poruku korištenjem MAC algoritma. Pritom koristimo ključ K koji je poznat i pošiljatelju i primatelju poruke (shared key). Kao rezultat dobijemo potpis. Naposljetku spremimo potpis u zasebnu datoteku koju smo nazvali “message.sig”.

Kod za postupak potpisivanja poruke:

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

# Signing process
# 1. read file
# 2. sign the message content using MAC primitive
# 3. store the signature in a separate file

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__":
    # 1.
    with open("message.txt", "rb") as file:
        message = file.read()

    # 2.
    key = "my super secure secret".encode()
    sig = generate_MAC(key, message)

    # 3.
    with open("message.sig", "wb") as file:
        file.write(sig)
```

Način na koji smo napisali ključ u kodu (key = "my super secure secret".encode( )) nije siguran ali poslužit će za svrhe ovog zadatka. Jedan od sigurnih načina na koji bi mogli generirat ključ je da zatražimo od korisnika da upiše neki passphrase koji ćemo onda koristiti.

Provjera autentičnosti poruke

Kako bismo provjerili autentičnost poruke prvo moramo pročitati datoteku u kojoj se nalazi poruka (”message.txt”) te datoteku u kojoj se nalazi njen potpis (”message.sig”), a koje bi putem javnog kanala dobili od pošiljatelja poruke. Nakon toga treba generirati potpis za poruku koju smo primili korištenjem MAC algoritma kojem prosljeđujemo tu poruku i zajednički ključ K. Naposljetku treba usporediti potpis koji smo upravo generirali s potpisom kojeg smo primili od pošiljatelja. Ukoliko su potpisi isti to znači da je poruka autentična, a ako su različiti onda poruka nije autentična.

Kod za postupak provjere autentičnosti poruke:

```
# Verification process
# 1. read file
# 2. read signature
# 3. sign the message content using MAC primitive
# 4. compare generated signature with received

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

if __name__ == "__main__":

    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 'NOK'}")
```

Funkcija verify_MAC interno na siguran način provjeri potpis.

Zadatak smo provjerili tako što isprva nismo promijenili orginalnu poruku ni njen potpis te kao rezultat dobili “Message is OK”. Nakon toga smo promijenili sadržaj poruke i ponovo pokrenili skriptu u kojoj se nalazi naš kod i dobili “Message is NOK”. Na kraju smo isprobali i slučaj u kojem smo koristili originalnu poruku ali smo promjenili njen potpis te također kao rezultat dobili “Message is NOK”.

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 (Da bi pristupili serveru **trebamo** biti dio lokalne mreže.)

Na lokalnom serveru se nalaze direktoriji sa našim imenima unutar kojih se nalazi po 20 tekstualnih datoteka. 10 od tih datoteka su transakcije, a 10 njihovi potpisi. Naš zadatak je da od tih 10 transakcija nađemo one koje su autentične te da ih ispravno vremenski poredamo. Ovaj zadatak se dosta oslanja na prethodni. Kako bismo ga riješili koristili smo for petlju koja se izvršavala 10 puta, po jednom za svaku transakciju. Autentičnost svake transakcije smo provjerili funkcijom verify_MAC.

Kod za drugi zadatak:

```
import re
import datetime
from pathlib import Path

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__":

    key = "ivic_sara".encode()

    PATH = "challenges/g1/ivic_sara/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}')
```

Moje rješenje prije sortiranja ispravnih poruka:

Message Sell 61 shares of Tesla (2022-12-05 21:22) NOK
Message Sell 65 shares of Tesla (2022-12-06 08:11) OK
Message Sell 32 shares of Tesla (2022-11-29 21:31) NOK
Message Sell 20 shares of Tesla (2022-12-03 13:20) OK
Message   Buy 5 shares of Tesla (2022-11-29 23:06) OK
Message Buy 60 shares of Tesla (2022-11-30 17:42) NOK
Message Sell 63 shares of Tesla (2022-12-05 21:29) NOK
Message Sell 64 shares of Tesla (2022-12-02 12:20) NOK
Message Sell 55 shares of Tesla (2022-12-02 13:05) OK
Message Sell 27 shares of Tesla (2022-11-29 17:10) NOK

Konačno rješenje (nakon sortiranja):

Message   Buy 5 shares of Tesla (2022-11-29 23:06) OK
Message Sell 55 shares of Tesla (2022-12-02 13:05) OK
Message Sell 20 shares of Tesla (2022-12-03 13:20) OK
Message Sell 65 shares of Tesla (2022-12-06 08:11) OK