**sigurnost lab5**

prvo smo napisali neku važnu poruku i želimo osigurat njezinu autentičnost

kako cemo pristupit problemu?

imamo 2 dijela problema,prvo moramo kreirati, otpisati ili osigurati poruku a drugi dio problema je sta moramo osigurati autenticnost te poruke

ako zelimo potpisat sadrzaj file-a, prvo moramo otvorit file(read file contetnt),nakon toga ga moramo ga potpisat te spremiti taj potpis u separate file

onaj koji prima tu poruku treba treba pročitati sadrzaj(reada recived file) treba pročitati potpis i mora samo potpisat taj file koristeči MAC pirimitive i usporediti tu lokalni generirani potpis sa onim kojeg je primia

funkcija za provjeru i funkcija za generiranje se samo razlikuju u verify dijelu

ako slucajno nakon sta smo prvi put procitali poruku i poslali je primatelju prominimo javllja da poruka nije uredu

zadtak 2

imamo 10 fileova sa porukon i 10 potpisa

nas je zadatak prepoznat koji su autenticni i povezat ih sa svojin potpison te ih kronloski rasporedi

kod:

import datetime

import re

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\_\_":

    # # 1. Sign the file content

    # # 1.1 Read the file content

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

    # message = file.read()

    # # print(content)

    # # 1.2 Sign the content

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

    # signature = generate\_MAC(key=key, message=message)

    # # print(signature)

    # # 1.3 Save the signature into a file

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

        # file.write(signature)

    # # 2. verify message authenticity

    # # 2.1 Read the recived file

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

        # content = file.read()

    # # 2.2 Read the recived signature

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

    # signature = file.read()

    # # 2.3.1 Sign the recived file

    # # 2.3.2 Compare locally generated signature with the received one

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

    # is\_authentic = verify\_MAC(key=key, signature=signature, message=content)

    # print(f"Message is {'OK'if is\_authentic else 'NOK'}")

PATH = "challenges/g2/belamaric\_gina\_aurora/mac\_challenge/"

KEY = "belamaric\_gina\_aurora".encode()

authentic\_messages = []

for ctr in range(1, 11):

    msg\_filename = f"order\_{ctr}.txt"

    sig\_filename = f"order\_{ctr}.sig"

    msg\_file\_path = Path(PATH + msg\_filename)

    with open(msg\_file\_path, "rb") as file:

        message = file.read()

    sig\_file\_path = Path(PATH + sig\_filename)

    with open(sig\_file\_path, "rb") as file:

        signature = file.read()

    is\_authentic = verify\_MAC(

        key=KEY, signature=signature, message=message)

    # print(f'Message {message.decode():>45} {"OK" if is\_authentic else "NOK":<6}')

    # sort po vrimenu

    if is\_authentic:

        authentic\_messages.append(message.decode())

    authentic\_messages.sort(

        key=lambda m: datetime.datetime.fromisoformat(

            re.findall(r"\(.\*?\)", m)[0][1:-1]

            )

    )

for m in authentic\_messages:

    print(f'Message {m:>45} {"OK":<6}')