3. laboratorijska vježba

Izazov 1

U ovom izazovu ćemo zaštititi integritet poruke (tekstualne datoteke) koristeći message authentication code (MAC) algoritam

1. Napravimo tekstualnu datoteku koju želimo zaštiti i ispišimo je na standardni izlaz

💻
```
from cryptography.hazmat.primitives import hashes

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

if __name__ == "__main__":
    main()
```

2. Napravimo funkciju za izračun MAC-a.

💻
```
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
```

3. Napravimo funkciju za validaciju MAC-a.

💻
```
def is_mac_valid(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
```

4. main funkcija

💻
```
def main():
    secret = b"My super secret"

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

    mac = generate_MAC(secret, content)
    print(mac.hex())
```

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

💻
```
with open("./message.sig", "rb") as file:
mac = file.read()
```

```
💻 if is_mac_valid(secret, content, mac):
       print("MAC je validan.")
   else:
       print("MAC nije validan.")
```

Izazov 2

U ovom izazovu ćemo utvrditi vremensku ispravnu sekvencu transakcija sa odgovarajućim dionicama. Na raspolaganju nam je 10 tekstualnih datoteka koje čine naloge za kupnju dionica. Svaka datoteka ima kreirani MAC. Zadatak je odrediti ispravan redoslijed transakcija autentičnih poruka.

💻
```
import os
from pprint import pprint
from datetime import datetime

from cryptography.exceptions import InvalidSignature
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives import hashes, hmac

def is_mac_valid(key, message, mac):
    if not isinstance(message, bytes):
        message = message.encode()

    h = hmac.HMAC(key, hashes.SHA256(), None)
    h.update(message)
    try:
        h.verify(mac)
    except InvalidSignature:
        return False
    else:
        return True

def read_orders_and_macs(directory):
    orders_and_macs = {}

    for index in range(1, 10):
        order_text_file_route = os.path.join(directory, f"order_{index}.txt")
        order_mac_file_route = os.path.join(directory, f"order_{index}.sig")

        order_text_file = open(order_text_file_route, "rb")
        order_mac_file = open(order_mac_file_route, "rb")

        orders_and_macs[order_text_file.read()] = order_mac_file.read()

        order_text_file.close()
        order_mac_file.close()
    return orders_and_macs

def extract_order_datetime(order):
    open_bracket_pos = order.index("(") + 1
    close_bracket_pos = order.index(")")

    date_string = order[open_bracket_pos:close_bracket_pos]
    return datetime.strptime(date_string, "%Y-%m-%dT%H:%M")

def main():
    key = "firic_filip".encode()

    orders_and_macs = read_orders_and_macs("mac_challenge")
    valid_orders = []

    for order, mac in orders_and_macs.items():
        if is_mac_valid(key, order, mac):
            valid_orders.append(order.decode("utf-8"))

    sorted_valid_orders = sorted(
        valid_orders,
        key=lambda order: extract_order_datetime(order),
    )

    pprint(sorted_valid_orders)

if __name__ == "__main__":
    main()
```