Lab 3: Symmetric key cryptography (a crypto challenge)

U sklopu vježbe student će riješiti odgovarajući *crypto* izazov, odnosno dešifrirati odgovarajući *ciphertext*. Izazov proizlazi iz činjenice da student nema pristup enkripcijskom ključu.

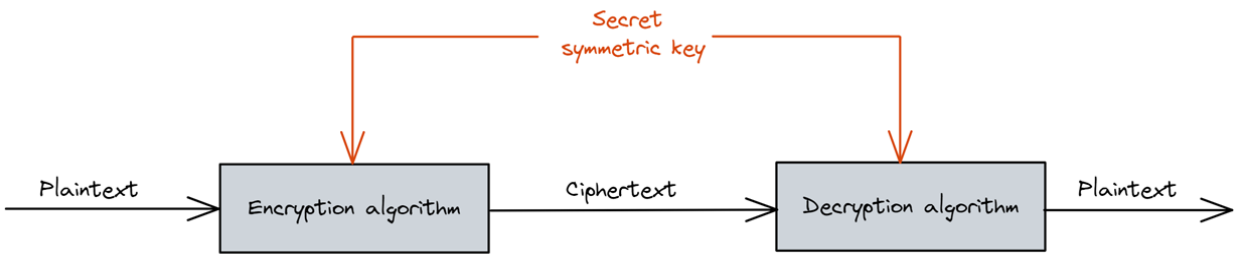
Zadatak

Za pripremu *crypto* izazova, odnosno enkripciju korištena je Python biblioteka `cryptography`. *Plaintext* koji student treba otkriti enkriptiran je korištenjem *high-level* sustava za simetričnu enkripciju iz navedene biblioteke - Fernet.

Fernet koristi sljedeće *low-level* kriptografske mehanizme:

- AES šifru sa 128 bitnim ključem u CBC enkripcijskom načinu rada
- HMAC (*hash MAC*) sa 256 bitnim ključem za zaštitu integriteta poruka
- Timestamp za osiguravanje svježine (*freshness*) poruka

Slika koja prikazuje glavne elemente simetričnog enkripcijskog sustava:



Pronalazak personaliziranog crypto izazova

Prvi zadatak koji smo trebali rješiti u ovoj vježbi bio je da pronađemo svoj personalizirani crypto izazov kojeg ćemo onda trebat dekriptirati. Naš izazov se nalazi među izazovima

koji su putem internog servera dostupni na sljedećoj adresi: http://challenges.local. Pritom je naš izazov pohranjen u datoteci čiji je naziv generiran na sljedeći način:

```
from cryptography.hazmat.primitives import hashes

def hash(input):
    if not isinstance(input, bytes):
        input = input.encode()

    digest = hashes.Hash(hashes.SHA256())
    digest.update(input)
    hash = digest.finalize()

    return hash.hex()

filename = hash('prezime_ime') + ".encrypted"
```

Ja sam svoj izazov pronašla koristeći sljedeći kod:

```
if __name__ == "__main__":
    filename = hash('ivic_sara') + ".encrypted"
    print(filename)
```

Kao rezultat sam dobila:
dbab177e15777581b7272afa1f77dc153128992d7c3cdaedf95a6e6375ebda91.encrypted

Sada znam u kojoj je datoteci pohranjen moj izazov te ju samo moram pronaći po imenu. Nakon što sam pronašla svoj izazov preuzela sam ga na lokalno računalo i spremila u datoteku s istim imenom.

Dekriptiranje osobnog izazova

Izazovi su enkriptirani koristeći ključ u kojeg samo zadnjih 22 bita pridonose njegovoj entropiji, tj. imamo ključ ograničene entropije koja iznosi 22 bita. Naš sljedeći zadatak je da dekriptiramo naš osobni izazov i da ga pohranimo u odgovarajuću datoteku. Pritom ćemo koristiti brute-force metodu, a ne nasumično pogađanje ključeva kako ne bismo isti ključ prolazili više puta. Prvo što smo napisali u našoj brute_force funkciji je iteriranje kroz ključeve (*enumerating all possible keys*)

```
def brute_force(ciphertext):
  ctr = 0
```

```
   while True:
     key_bytes = ctr.to_bytes(32, "big")
     key = base64.urlsafe_b64encode(key_bytes)

     ctr += 1
```

Zasad nam ova funkcija generira ključeve.

Dakle mi znamo ciphertext i iz njega nekako moramo doći do ključa te do samog plaintexta. Osim toga smo dobili informaciju da je naš ciphertext enkriptirana slika u .png formatu. Kako bismo došli do rješenja koristit ćemo sljedeći pseudokod unutar funkcije brute_force_attack(ciphertext) :

```
if plaintext ima smisla:
    print(key, plaintext)
    save(plaintext)
    break
ctr += 1
```

Najveći problem na koji sada nailazimo je kako ćemo provjeriti ima li plaintext smisla. To možemo napraviti tako da gledamo neke njegove karakteristike. Npr. ako znamo da je enkriptiran tekst o nekoj određenoj temi možemo tražit neku riječ specifičnu za tu temu. Nama je enkriptirana slika u png formatu, a svaka slika ima neke metapodatke. Nakon malo istraživanja slika u png formatu došli smo do sljedećeg podatka:

PNG file signature

The first eight bytes of a PNG file always contain the following (decimal) values:

```
   137 80 78 71 13 10 26 10
```

This signature indicates that the remainder of the file contains a single PNG image.

Sada znamo da će prvih 8 byteova našeg plaintexta biti 137 80 78 71 13 10 26 10. Kako znamo da se ovi brojevi nalaze na početku plaintexta kada gledamo ima li naš plaintext smisla nećemo gledati cijeli plaintext nego samo njegov header (Mi smo gledali prvih 32 byteova). Umjesto pseudokoda ćemo unutar brute_force funkcije napisati sljedeći kod:

```
header = plaintext[:32]
    if test_png(header):
```

```
        print(f"Kex FOUND: {key}")
        with open("BINGO.png", "wb") as file:
            file.write(plaintext)
        break
    #kontrola
    ctr+=1
    if not ctr % 1000:
        print(f"[*] Keys tested: {ctr:,}", end="\r")
```

Pritom je test_png(header) funkcija sljedećeg oblika:

```
def test_png(header):
    if header.startswith(b"\211PNG\r\n\032\n"):
        return True
    return False
```

Također smo dodali da nam brute_force funkcija ispisuje kada generira svaki 1000-ti ključ kako bismo mogli pratiti koliko smo ključeva isprobali. Ne ispisujemo kada generiramo svaki ključ kako ne bismo potrošili previše vremena.

Sljedeći problem na koji nailazimo je da nam kod ne radi jer pokušavamo dekriptirat ciphertext s pogrešnim ključem i library fernet se buni. Narušen je integritet podataka. Da bismo riješili ovaj problem koristimo exception

```
 except InvalidToken:
            pass
```

S obzirom da po redu ispitujemo sve moguće ključeve brute-force metoda će nekima trajat duže a nekima kraće. S obzirom da imamo ključ entropije 22 bita što znači da nam samo tih 22 bita utječe na ključeve koje isprobajemo za dekiptirati ciphertext ima smisla pretraživati dok ne ispitamo 2^22 ključeva. To iznosi 4194304. Meni se ciphertext uspješno dekriptirao, tj. pronašao ispravan ključ nakon što se isprobalo otprilike 3 milijuna ključeva.

Cijeli kod korišten u rješavanju ove vježbe:

```
import base64
from os import path
from pydoc import plain
import re
from xmlrpc.client import TRANSPORT_ERROR
```

```
from cryptography.hazmat.primitives import hashes
from cryptography.fernet import Fernet, InvalidToken


def hash(input):
    if not isinstance(input, bytes):
        input = input.encode()

    digest = hashes.Hash(hashes.SHA256())
    digest.update(input)
    hash = digest.finalize()

    return hash.hex()

def test_png(header):
    if header.startswith(b"\211PNG\r\n\032\n"):
        return True
    return False

def brute_force_attack(ciphertext):
    #print(ciphertext)
    ctr= 0
    while True:
        key_bytes = ctr.to_bytes(32, "big")
        key = base64.urlsafe_b64encode(key_bytes)

        try:
            # Try to use this key somehow
            plaintext = Fernet(key).decrypt(ciphertext)

            header = plaintext[:32]
            if test_png(header):
                print(f"Key FOUND: {key}")
                with open("BINGO.png", "wb") as file:
                    file.write(plaintext)
                break

        except InvalidToken:
            pass
        #kontrola
        ctr+=1
        if not ctr % 1000:
            print(f"[*] Keys tested: {ctr:,}", end="\r")


if __name__ == "__main__":
    filename = hash('ivic_sara') + ".encrypted"
    print(filename)

    if not path.exists(filename):
        with open(filename, "wb") as file:
            file.write(b"")
```

```
with open(filename, "rb") as file:
    encrypted_challenge = file.read()

brute_force_attack(encrypted_challenge)
```

Moj ključ i slika:
Key FOUND: b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAtZBY='