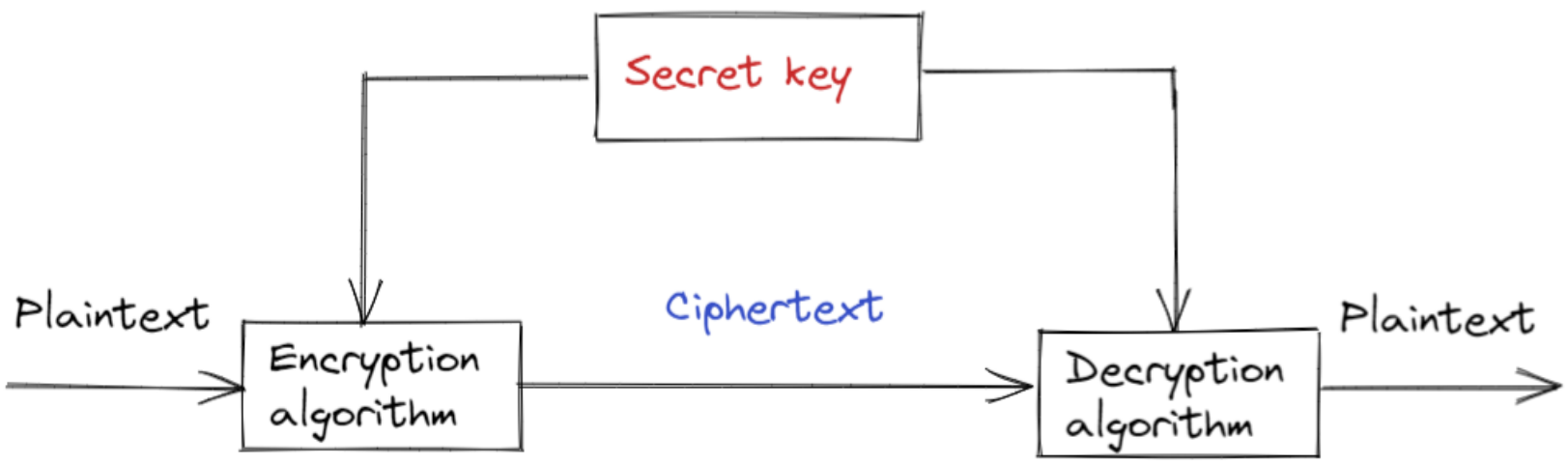
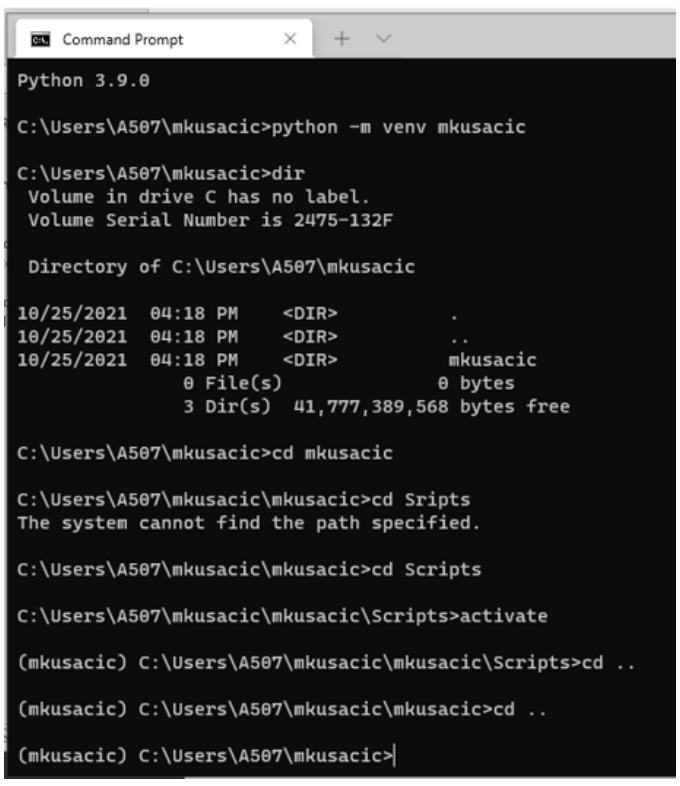
Vježba 2: Symmetric key cryptography

Simetrična kriptografija

Simetrična kiptografija je proces šifriranja polazne poruke (Plaintexta) pri kojem odašiljatelj i primatelj poruke koriste isti ključ pri njenom šifriranju i dešifriranju.





Napravili smo virtualno okruženje u Pythonu kako nove biblioteke i nova verzija Pythona koje koristimo u ovom projektu ne bi interferirali s ostalim projektima.

- Pomoću metoda iz biblioteke cryptography.fernet generirali smo slučajni ključ i upisali je u varijablu key. Ključ:

```
b'BWNkxoRz_LJ5LT-2GgmcBv1LTA4yiybXWxRtbj5X2F8='
```

- Nakon toga smo stvorili objekt f klase Fernet koji prima parametar key
- Inicijaliziramo plaintext s tim da varijabla mora biti formatirana u bitovima (b" ")
- Na objekt f koristimo metode .encrypt(plaintext) za enkriptiranje i .decrypt(plaintext) za decryptiranje plaintexta.

Cyphertext tj. enkriptirani plaintext:

```
b'gAAAAABhdsAqNVBMk4KM2z-Ihf2Ekj-Z_8nfDYfATL2c4BrLGSNeL9cWW8Wmp6UQkcnu9H6NF5kYYRgf-qz638k-3GqfneTyZw=='
```

```
>>> from cryptography.fernet import Fernet
>>> key = Fernet.generate_key()
>>> Fernet.generate_key()
b'BWNkxoRz_LJ5LT-2GgmcBv1LTA4yiybXWxRtbj5X2F8='
>>> f = Fernet(key)
>>> plainttext=b"mydeepsecret"
>>> plaintext=b"mydeepsecret"
>>> f.encrypt(plaintext)
b'gAAAAABhdsAqNVBMk4KM2z-Ihf2Ekj-Z_8nfDYfATL2c4BrLGSNeL9cWW8Wmp6UQkcnu9H6NF5kYYRgf-qz638k-3GqfneTyZw=='
>>> ciphertext=f.encrypt(plaintext)
>>> f.decrypt(ciphertext)
b'mydeepsecret'
```

Pri generiranju novoga ključa te pokušaja dekripcije ciphertexta, program izbacuje Exception.

```
>>> f.encrypt(plaintext)
b'gAAAAABhdsAqNVBMk4KM2z-Ihf2Ekj-Z_8nfDYfATL2c4BrLGSNeL9cWW8Wmp6UQkcnu9H6NF5kYYRgf-qz638k-3GqfneTyZw=='
>>> ciphertext=f.encrypt(plaintext)
>>> f.decrypt(ciphertext)
b'mydeepsecret'
>>> key = Fernet.generate_key()
>>> f=Fernet(key)
>>> f.decrypt(ciphertext)
Traceback (most recent call last):
  File "C:\Users\A507\mkusacic\mkusacic\lib\site-packages\cryptography\fernet.py", line 124, in _verify_signature
    h.verify(data[-32:])
  File "C:\Users\A507\mkusacic\mkusacic\lib\site-packages\cryptography\hazmat\primitives\hmac.py", line 78, in verify
    ctx.verify(signature)
  File "C:\Users\A507\mkusacic\mkusacic\lib\site-packages\cryptography\hazmat\backends\openssl\hmac.py", line 76, in ver
ify
    raise InvalidSignature("Signature did not match digest.")
cryptography.exceptions.InvalidSignature: Signature did not match digest.

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\A507\mkusacic\mkusacic\lib\site-packages\cryptography\fernet.py", line 85, in decrypt
    return self._decrypt_data(data, timestamp, time_info)
  File "C:\Users\A507\mkusacic\mkusacic\lib\site-packages\cryptography\fernet.py", line 142, in _decrypt_data
    self._verify_signature(data)
  File "C:\Users\A507\mkusacic\mkusacic\lib\site-packages\cryptography\fernet.py", line 126, in _verify_signature
    raise InvalidToken
cryptography.fernet.InvalidToken
```

Vježba - crypto challange

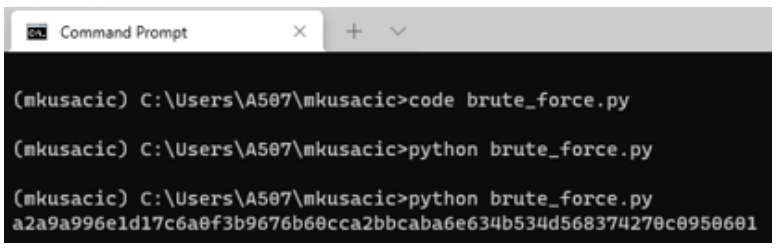
U sklopu vježbe treba dešifrirati odgovarajući ciphertext u kontekstu simetrične kriptografije.
Izazov počiva na činjenici da student nema pristup enkripcijskom ključu.

brute_force.py

C: > Users > A507 > mkusacic > brute_force.py

```python
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()

if __name__ == "__main__":
    print(hash("kusacic_marko"))
```

Kod preko kojeg smo dobili hash value našeg imena. Koristili smo hash funkciju SHA256 koja generira 256-bitni
(32 byteni) value.
Hash value je uvijek isti za iste plaintextove.



Dobili smo hash funkciju našega imena.
Sada možemo dekriptirati file kojeg skidamo s lokalnog servera.

Iteracija kroz ključeve:

```python

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

        # Now initialize the Fernet system with the given key
        # and try to decrypt your challenge.
        # Think, how do you know that the key tested is the correct key
        # (i.e., how do you break out of this infinite loop)?

        ctr += 1


if __name__ == "__main__":
    brute_force()

```

Brute force attack → pokušavamo naći naš ključ prolaženjem kroz sve ključeve.

U ovom slučaju riječ je infinite loopu iz razloga što nismo uveli uvjet izlaska iz petlje.

Rezultat beskonačne iteracije. Na slici su prikazani ključevi.

Dodali smo brojač ključeva kroz koju petlja prolazi:

```python
while True:
    key_bytes = ctr.to_bytes(32, "big")
    key = base64.urlsafe_b64encode(key_bytes)
    print(f"[*] Keys tested; {ctr +1:,}", end="\r")
```

```
(mkusacic) C:\Users\A507\mkusacic>python brute_force.py
[*] Keys tested; 119,232
```

```
39          try:
40              plaintext = Fernet(key).decrypt(ciphertext)
41              header = plaintext[:32]
42
43              if test_png(header):
44                  print(f"[+] KEY FOUND: {key}")
45                  with open("BINGO.png", "wb") as file:
46                      file.write(plaintext)
47
48                  print(plaintext)
49                  break
50
51          except Exception:
52              pass
53
54          ctr += 1
55
```

Kako bi izašli iz petlje moramo uvesti try-except block.

Pošto znamo da se u plaintextu nalazi slika, iskoristit ćemo znanje da svaka slika ima header.

Varijabla header poprima prvih 32 vrijednosti liste plaintext.

Ako header pripada slici, stvaramo novi file "BINGO.png" i u njega upisujemo plaintext.

Rezultat: