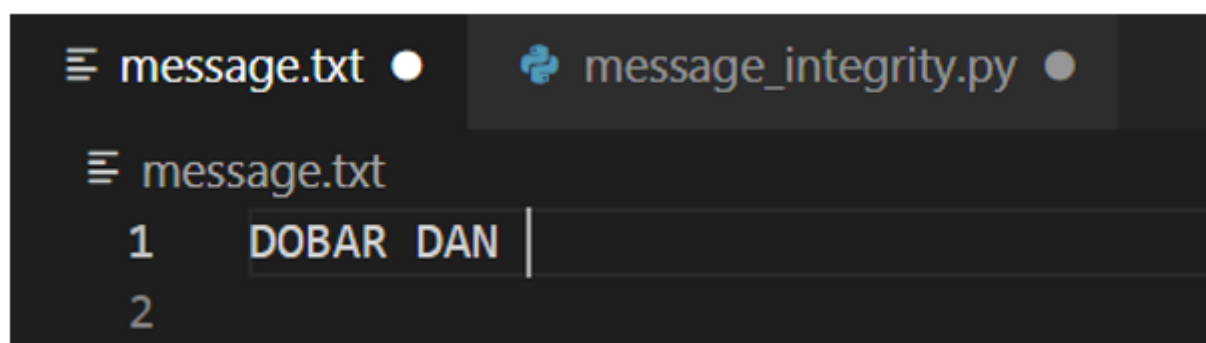


# Vježba 3: Message Authentication Code (MAC)

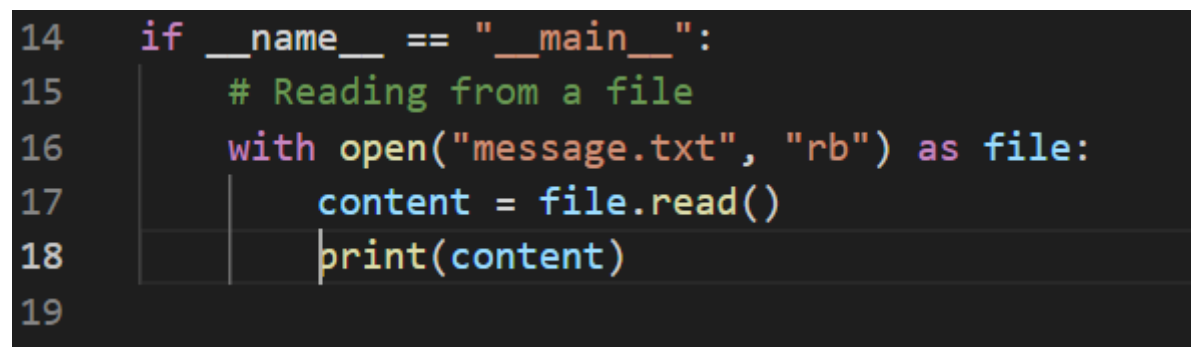
## Izazov 1

Implementirajte zaštitu integriteta sadržaja dane poruke primjenom odgovarajućeg *message authentication code (MAC)* algoritma. Koristiti pri tome HMAC mehanizam iz Python biblioteka *cryptography*.

Napravili smo datoteku *message.txt* u kojem smo upisali poruku kojoj želimo zaštititi integritet.

A screenshot of a code editor with a dark background. At the top, there are two tabs: 'message.txt' (active) and 'message\_integrity.py'. The 'message.txt' tab is selected, and its content is displayed below. The content consists of two lines: '1 DOBAR DAN' and '2'. The text is in a light-colored font, likely white or light blue, against the dark background.

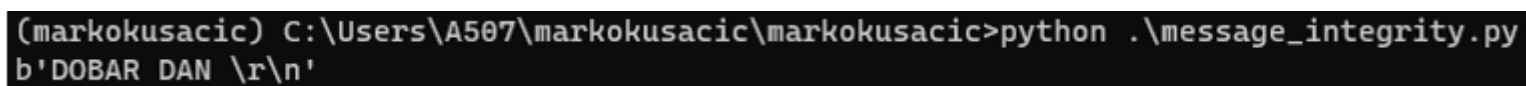
Na sljedeći način učitavamo sadržaj datoteke, upisujemo ga u varijablu *content* i ispisujemo pomoću *print()*.

A screenshot of a code editor showing Python code. The code is as follows:

```
14 if __name__ == "__main__":  
15     # Reading from a file  
16     with open("message.txt", "rb") as file:  
17         content = file.read()  
18         print(content)  
19
```

The code is written in a light-colored font on a dark background. The line numbers 14 through 19 are visible on the left side of the code block.

Rezultat:

A screenshot of a terminal window showing the execution of a Python script. The prompt is '(markokusacic) C:\Users\A507\markokusacic\markokusacic>'. The command executed is 'python .\message\_integrity.py'. The output is 'b'DOBAR DAN \r\n'', which is displayed in a light-colored font on a dark background.

```
message_integrity.py ● message.txt ●
message_integrity.py > ...
1  from cryptography.hazmat.primitives import hashes, hmac
2
3  def generate_MAC(key, message):
4      if not isinstance(message, bytes):
5          message = message.encode()
6
7      h = hmac.HMAC(key, hashes.SHA256())
8      h.update(message)
9      signature = h.finalize()
10     return signature
```

```
if __name__ == "__main__":
    key = b"skrivena tajna"
    # Reading from a file
    with open("message.txt", "rb") as file:
        content = file.read()
        print(content)

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

Pozivamo funkciju **generate\_MAC** s parametrima: skriveni ključ (**key**) i sadržaj teksta (**content**).

Povratna vrijednost funkcije se upisuje u varijablu **mac** i ispisuje u heksidecimalnom obliku.

Objašnjenje funkcije **generate\_MAC**:

- funkcija prima dva parametra: key i content
- `message.encode()` se izvrši u slučaju da sadržaj teksta ima članove koje nespadaju pod UTF-8 standard
- objekt `h` uzima key i algoritam (**SHA256**) i stvara HMAC koji se pridodaje sadržaju sa `h.update(message)`
- funkcija vraća HMAC

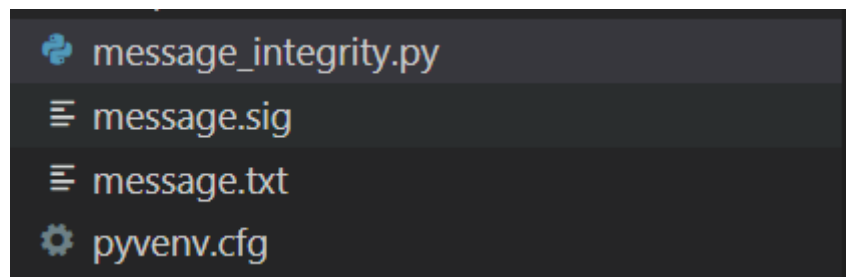
Rezultat:

```
(markokusacic) C:\Users\A507\markokusacic\markokusacic>python .\message_integrity.py
b'DOBAR DAN \r\n'
f7580bd806f9a036ecb562776f93de094768f67f0e571d92742a33fd9a910c97
```

Kreiramo novu datoteku *message.sig* i upisujemo sadržaj varijable **mac** u nju.

```
22     mac = generate_MAC(key, content)
23     print(mac.hex())
24
25     with open("message.sig", "wb") as file:
26         content = file.write(mac)
```

Rezultat:



```
30 if __name__ == "__main__":
31     key = b"skrivena tajna"
32     # Reading from a file
33     with open("message.txt", "rb") as file:
34         content = file.read()
35
36     print(content)
37
38     # mac = generate_MAC(key, content)
39     # print(mac.hex())
40
41     with open("message.sig", "rb") as file:
42         mac = file.read()
43
44     is_authentic = verify_MAC(key, mac, content)
45     print(is_authentic)
```

Nakon toga, sadržaj iz datoteke *message.sig* šaljemo kao parametar u funkciju **verify\_MAC**.

```
6 def verify_MAC(key, signature, message):
7     if not isinstance(message, bytes):
8         message = message.encode()
9
10    h = hmac.HMAC(key, hashes.SHA256())
11    h.update(message)
12    try:
13        h.verify(signature)
14    except InvalidSignature:
15        return False
16    else:
17        return True
```

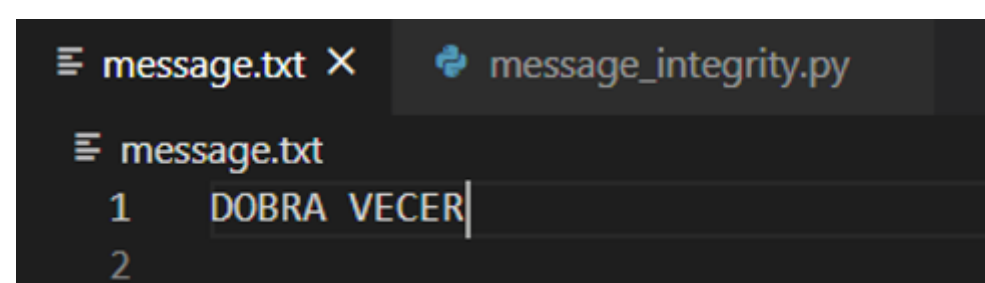
Funkcija **verify\_MAC** je slična prošloj funkciji samo što još prima MAC (iz kanala) kao parametar i uspoređuje ga s lokalnim MACom ( `h.verify(signature)` ). Ako try block izbaci exception, MAC iz kanala i lokalni MAC nisu isti ,te funkcija **verify\_MAC** vraća *False*. U protivnome vraća *True*, te je integritet poruke očuvan..

Rezultat kada je integritet poruke očuvan, dobivamo *True*:

```
(markokusacic) C:\Users\A507\markokusacic\markokusacic>python .\message_integrity.py
b'DOBAR DAN \r\n'
True
```

Dva načina na koji možemo urušiti integritet poruke:

1. Promijeniti sadržaj poruke : DOBAR DAN → DOBRA VECER



2. Promijeniti HMAC naše poruke: A4 → C0

message.sig																	
	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	DECODED TEXT
00000000	F7	58	0B	D8	06	F9	A0	36	EC	B5	62	77	6F	93	DE	09	÷ X . ø . ù . 6 ì μ b w o . þ .
00000010	47	68	F6	7F	0E	57	C0	92	74	2A	33	FD	9A	91	0C	97	G h ö . . W Ä . t * 3 ý . . . .
00000020	+																+

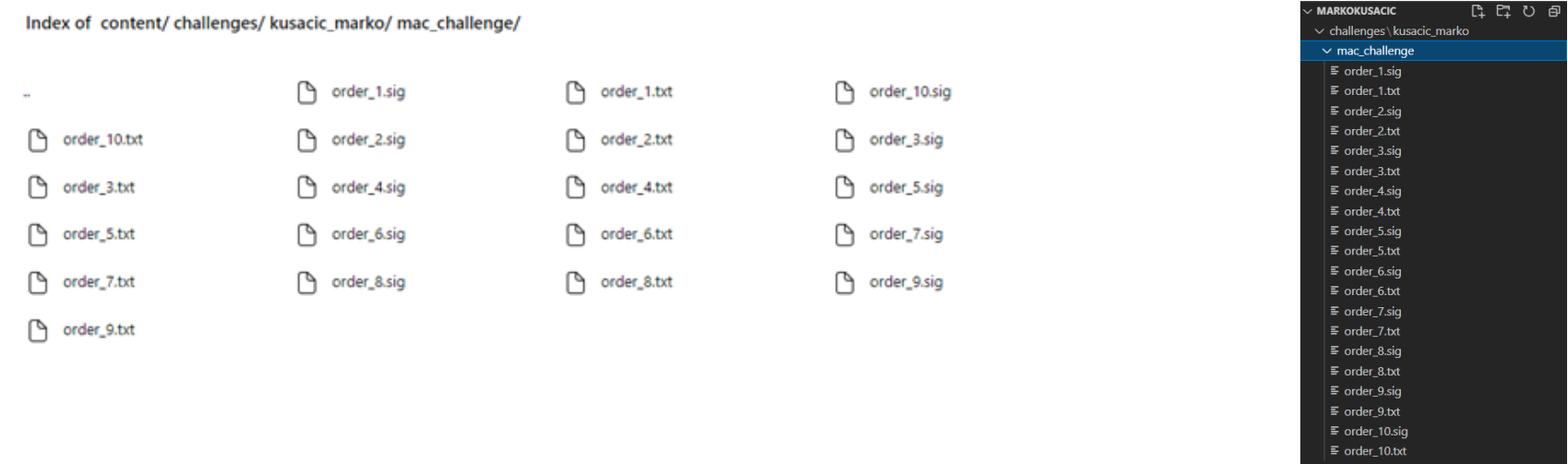
Rezultat kada je integritet poruke urušen, dobivamo *False*:

```
(markokusac) C:\Users\A507\markokusac\markokusac>python .\message_integrity.py
b'DOBAR DAN \r\n'
False
```

## Izazov 2

U ovom izazovu želimo utvrditi vremenski ispravnu skevencu transakcija (ispravan redosljed transakcija) sa odgovarajućim dionicama.

Pomoću wget.exe programa skinili smo .txt i .sig datoteke te ih spremili mapu gdje se nalazi naš python program.



Cilj nam je usporediti .txt i .sig datoteke istih imena, odbaciti one s urušenim integritetom i poredati ih prema vremenu slanja koji se nalazi u svakom sadržaju .txt datoteka.

Ključ koji se koristio pri kreiranju MAC-a je naše "prezime\_ime" ,tako da se koristi i pri dekriptiranju u **verify\_MAC** funkciji.

```
30 if __name__ == "__main__":
31     key = "kusac_marko"
32     key = key.encode()
33
34     for ctr in range(1, 11):
35         msg_filename = f"order_{ctr}.txt"
36         sig_filename = f"order_{ctr}.sig"
37
38         with open(msg_filename, "rb") as file:
39             content = file.read()
40
41         with open(sig_filename, "rb") as file:
42             mac = file.read()
43
44         print(msg_filename)
45         print(sig_filename)
46
47         is_authentic = verify_MAC(key, mac, content)
48         print(
49             f'Message {content.decode():>45} {"OK" if is_authentic else "NOK":<6}')
```

Kako bi iterirali kroz sve datoteke koristimo **for** petlju.  
Čitamo .txt i .sig datoteke istih indeksa (**ctr**) te uzimamo sadržaj **content** i **mac** koje ubacujemo u funkciju **verify\_MAC**.  
Nakon toga ispisujemo **"OK"** uz poruku ako funkcija vraća *True* ,te **"NOK"** ako vraća *False*.

Rezultat :

```
Command Prompt
Message      Buy 50 shares of Tesla (2021-11-11T05:25) NOK
order_2.txt
order_2.sig
Message      Sell 60 shares of Tesla (2021-11-11T15:08) OK
order_3.txt
order_3.sig
Message      Buy 73 shares of Tesla (2021-11-10T08:37) OK
order_4.txt
order_4.sig
Message      Buy 69 shares of Tesla (2021-11-13T16:12) OK
order_5.txt
order_5.sig
Message      Buy 57 shares of Tesla (2021-11-11T09:36) OK
order_6.txt
order_6.sig
Message      Sell 5 shares of Tesla (2021-11-14T02:37) OK
order_7.txt
order_7.sig
Message      Sell 91 shares of Tesla (2021-11-09T03:55) OK
order_8.txt
order_8.sig
Message      Buy 10 shares of Tesla (2021-11-10T20:49) OK
order_9.txt
order_9.sig
Message      Buy 84 shares of Tesla (2021-11-13T21:58) NOK
order_10.txt
order_10.sig
Message      Buy 59 shares of Tesla (2021-11-13T12:36) OK

(markokusacic) C:\Users\A507\markokusacic\markokusacic>
```

Način na koji bi poredali datoteke prema vremenu u kojem su poslani (**na svome primjeru**) :

OPEN FILES

×

BZVZ.py

×

poruka1.txt

×

poruka2.txt

×

poruka3.txt

×

poruka4.txt

×

poruka5.txt

Napravili smo 5 poruka sa različitim dužinama stringa, te različitim vremenima u kojima su poslani.

Ispis sadržaja poruka:

```
b'ccccccccc (2021-05-06T11:30) '
b'bbbbbbbbbbbbbb (2021-12-13T12:36) '
b'cccccccccccccccc (2021-11-13T12:37) '
b'ddddddddddddddddddddddddddddddddddd (2021-11-12T12:36) '
b'aweqesda (2021-12-12T00:00) '
[Finished in 60ms]
```

Pošto vidimo da je na kraju stringa dio poruke koji nam pokazuje vrijeme slanja poruke uvijek iste veličine, za svaku poruku možemo koristiti slice string i izvaditi taj podatak :

```
4 for i in range(1,6):
5
6     with open(f"poruka{i}.txt","rb") as file:
7         content = file.read()
8
9
10    print(content[-18:])
```

```
b'(2021-05-06T11:30)'
b'(2021-12-13T12:36)'
b'(2021-11-13T12:37)'
b'(2021-11-12T12:36)'
b'(2021-12-12T00:00)'
[Finished in 70ms]
```

Kako bi izvukli konvertirali dio stringa u vrijeme, koristimo library **datetime**, pomoću koje možemo napraviti objekte vremena koji se sastoje od više varijabla: godine, mjeseca ,dana ,sata i minute.

Te objekte smo pohranili u listu lista[ ] .

```
1 import time
2 from datetime import datetime
3
4 lista=[]
5 for i in range(1,6):
6
7     with open(f"poruka{i}.txt","r") as file:
8         content = file.read()
9
10
11
12     datetime_object=datetime.strptime(content[-18:], '%Y-%m-%dT%H:%M')
13     lista.append(datetime_object)
14
15
16
17
18 print(lista[1])
19 print(lista[2])
20 print(lista[1]>lista[2])
```

```
2021-12-13 12:36:00
2021-11-13 12:37:00
True
[Finished in 100ms]
```

Te objekte možemo uspoređivati. Uspoređivanje radi na način ako je jedan veći od drugoga  
( npr. lista[1] > lista[2] ) znači da za usporedbu noviji > stariji dobivamo True.



```

16 i=0
17 for i in range(i,len(lista)):
18
19     for j in range(i+1,len(lista)):
20
21         if lista[i]<lista[j]:
22
23             with open(f"poruka{i+1}.txt","r") as file:
24                 content = file.read()
25
26             with open(f"temp_poruka.txt","w") as file:
27                 file.write(content)
28
29             os.remove(f"poruka{i+1}.txt")
30
31             ime_sad=f'poruka{j+1}.txt'
32             ime_nakon=f'poruka{i+1}.txt'
33             os.rename(ime_sad,ime_nakon)
34
35             ime_sad=f'temp_poruka.txt'
36             ime_nakon=f'poruka{j+1}.txt'
37             os.rename(ime_sad,ime_nakon)
38

```

S obzirom da znamo da je povezanost između indeksa naziva datoteke i mjesta datuma u listi[i] i+1 možemo izvesti iteraciju kroz dvije for petlje gdje se datumi iz liste međusobno uspoređuju, te se datoteke raspoređuju na način da je najnovija datoteka s indeksom 1, a najstarija 5.

Ovaj proces sličan je raspoređivanju članova niza od najvećeg prema najmanjem.

**Pri svakoj iteraciji ( ako if grana vraća *True*) dolazi do zamjene imena datoteka.**

Kako bi preimenovali datoteku koristimo os.rename() s parametrima: trenutno ime i ime u koje želimo datoteku preimenovat.

Problem na koji nailazimo je da ne možemo datoteku preimenovat u ime koje već postoji.

To smo riješili na način da stvorimo privremenu datoteku *temp\_poruka.txt* u koju upisujemo sadržaj prve datoteke (koju brišemo) i kasnije tu privremenu datoteku preimenujemo u ime druge datoteke.

**Rezultat :**

```

2021-12-13 12:36:00
2021-12-12 00:00:00
2021-11-13 12:37:00
2021-11-12 12:36:00
2021-05-06 11:30:00
[Finished in 148ms]

```

Na taj način rasporedili smo sve datoteke na način da je najnovija datoteka s najmanjim indeksom, a najstarija s najvećim.

```

39 for i in range(1,6):
40     with open(f"poruka{i}.txt","rb") as file:
41         content = file.read()
42         print (content)

```

```

b'bbbbbbbbbbbbbb (2021-12-13T12:36)'
b'aweqesda (2021-12-12T00:00)'
b'cccccccccccccccc (2021-11-13T12:37)'
b'ddddddddddddddddddddddddddddddddddd (2021-11-12T12:36)'
b'cccccccccc (2021-05-06T11:30)'
[Finished in 114ms]

```

Cijeli kod:

```

1  import time
2  from datetime import datetime
3  import os
4
5  lista=[]
6  for i in range(1,6):
7
8      with open(f"poruka{i}.txt","r") as file:
9          content = file.read()
10
11
12
13      datetime_object=datetime.strptime(content[-18:], '%Y-%m-%dT%H:%M')
14      lista.append(datetime_object)
15
16  i=0
17  for i in range(i,len(lista)):
18
19      for j in range(i+1,len(lista)):
20
21          if lista[i]<lista[j]:
22
23              with open(f"poruka{i+1}.txt","r") as file:
24                  content = file.read()
25
26              with open(f"temp_poruka.txt","w") as file:
27                  file.write(content)
28
29              os.remove(f"poruka{i+1}.txt")
30
31              ime_sad=f'poruka{j+1}.txt'
32              ime_nakon=f'poruka{i+1}.txt'
33              os.rename(ime_sad,ime_nakon)
34
35              ime_sad=f'temp_poruka.txt'
36              ime_nakon=f'poruka{j+1}.txt'
37              os.rename(ime_sad,ime_nakon)
38
39  for i in range(1,6):
40      with open(f"poruka{i}.txt","rb") as file:
41          content = file.read()
42      print (content)

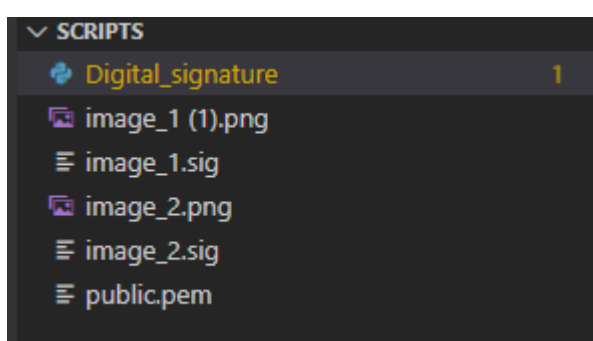
```

## Digital signatures using public-key cryptography

U ovom izazovu trebamo odrediti koja je slika (između njih dvije) autentična.

Slike su potpisane privatnim ključem, što znači da javnim ključem provjeravamo autentičnost slike.

Javni ključ je dostupan na lokalnom servera.



Učitane slike, digitalni potpisi i javni ključ.



```

Digital_signature > ...
1  from cryptography.hazmat.primitives import serialization
2  from cryptography.hazmat.backends import default_backend
3
4
5  def load_public_key():
6      with open("public.pem", "rb") as f:
7          PUBLIC_KEY = serialization.load_pem_public_key(
8              f.read(),
9              backend=default_backend()
10         )
11     return PUBLIC_KEY
12
13
14 public_key = load_public_key()
15
16 print(public_key)
17

```

Funkcijom **load\_public\_key** učitavamo javni ključ iz **public.pem** datoteke koje smo skinuli sa servera. U funkciji se vrši deserializacija javnog ključ kojeg upisujemo u varijablu **public\_key**.

**Rezultat:**

```

(marko_kusacic) C:\Users\A507\marko_kusacic\marko_kusacic>python .\Digital_signature.py
<cryptography.hazmat.backends.openssl.rsa._RSAPublicKey object at 0x0000016EFC940D30>

```

Provjeramo autentičnost slika funkcijom **verify\_signature\_rsa** koja kao argumente prima **signature** i **image** gdje je :

- **signature** → varijabla u koju je učitana sadržaj .sig datoteke (potpisa)
- **image** → varijabla u koju je učitana sadržaj .png datoteke (slike koje provjeravamo)

Također koristi već spomenuti javni ključ.

Pri kreiranju potpisa se koristio **padding** (proces dodavanja random sadržaja tj. soli sadržaju poruke koje potpisujemo), te hashiranje te cjeline algoritmom SHA256. Tako je dobiven **potpis**.

Proces verifikacije je isti, gdje se cijeli proces ponovi na sliku koju provjeravamo, te dobijemo njeni hash vrijednost. Nakon toga uspoređujemo dva hasha.

U slučaju da su **message** i **signature** isti, funkcija vraća *True*, u protivnome *False*.

```

17 def verify_signature_rsa(signature, message):
18     PUBLIC_KEY = load_public_key()
19     try:
20         PUBLIC_KEY.verify(
21             signature,
22             message,
23             padding.PSS(
24                 mgf=padding.MGF1(hashes.SHA256()),
25                 salt_length=padding.PSS.MAX_LENGTH
26             ),
27             hashes.SHA256()
28         )
29     except InvalidSignature:
30         return False
31     else:
32         return True
33
34
35 with open("image_1.sig", "rb") as file:
36     signature = file.read()
37
38 with open("image_1.png", "rb") as file:
39     image = file.read()
40
41
42 is_authentic = verify_signature_rsa(signature, image)
43 print(is_authentic)

```

Rezultat: **Slika 1** → **Autentična**

```

(marko_kusacic) C:\Users\A507\marko_kusacic\marko_kusacic\Scripts>python Digital_signature.py
True

```

Rezultat: **Slika 2** → **Nije Autentična**

```

(marko_kusacic) C:\Users\A507\marko_kusacic\marko_kusacic\Scripts>python Digital_signature.py
False

```