Usando i diritti da super user, installo “openssl” che è un implementazione open source dei protocolli SSL e TLS:

```
┌──(root㉿kali)-[/home/kali]
└─# sudo apt install openssl_
```

Poi scarico” pip”, il gestore di pacchetti per Python:

```
┌──(root㉿kali)-[/home/kali]
└─# apt install python3-pip_
```

E successivamente il pacchetto “cryptography” :

```
┌──(root㉿kali)-[/home/kali]
└─# pip3 install cryptography
Requirement already satisfied: cryptography in /usr/li
b/python3/dist-packages (42.0.5)
WARNING: Running pip as the 'root' user can result in
broken permissions and conflicting behaviour with the
system package manager, possibly rendering your system
 unusable.It is recommended to use a virtual environme
nt instead: https://pip.pypa.io/warnings/venv. Use the
 --root-user-action option if you know what you are do
ing and want to suppress this warning.
```

Ho ricevuto questo Messaggio di errore perché non è consigliato essere super user perché potrebbero crearsi conflitti con i permessi dei pacchetti che si installano e quindi ho ripetuto diventando utente normale:

```
┌──(root㉿kali)-[/home/kali]
└─# exit

┌──(kali㉿kali)-[~]
└─$ pip3 install cryptography
Defaulting to user installation because normal site-pa
ckages is not writeable
Requirement already satisfied: cryptography in /usr/li
b/python3/dist-packages (42.0.5)
```

Adesso posso andare a generare le chiavi con “openssl” .
Per generare la chiave privata (genpkey) si utilizza l’algoritmo RSA e si salva col nome “private_key.pem” (-out) e si indica 2040 come dimensione di bit per la chiave RSA (rsa_keygen_bits:2040)

```
┌──(kali㉿kali)-[~]
└─$ openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2040
.....................................................................+++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++
.........+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Per la chiave pubblica, si usa sempre l’algoritmo RSA e si indica che si desidera estrarre la chiave pubblica dalla chiave privata (-pubout) presa in input dal file “private_key.pem” (-in) e successivamente si scrive la chiave pubblica nel file “public_key.pem” (-out)

```
┌──(kali㉿kali)-[~]
└─$ openssl rsa -pubout -in private_key.pem -out public_key.pem
writing RSA key
```

Il formato .pem (Privacy-Enhanced Mail) è un formato di file testuale codificato in Base64 che generalmente è racchiuso da due delimitatori specifici:

```
-----BEGIN PRIVATE KEY-----
<contenuto codificato>
-----END PRIVATE KEY-----
```

Successivamente ho creato il file “endec.py” per criptare il messaggio “Ciao, Epicode spacca!”:

```python
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import serialization
import base64
import os
os.system('clear')

# Carica la chiave privata
with open('private_key.pem', 'rb') as key_file:
    private_key = serialization.load_pem_private_key(
    key_file.read(),
    password=None)
# Carica la chiave pubblica
with open('public_key.pem', 'rb') as key_file:
    public_key = serialization.load_pem_public_key(key_file.read())

message = 'Ciao, Epicode spacca!'

#Criptazione con la chiave pubblica
encrypted = public_key.encrypt(message.encode(), padding.PKCS1v15())

#Decriptazione con la chiave privata
decrypted = private_key.decrypt(encrypted, padding.PKCS1v15())

print("\nMessaggio originale:", message)
print("\nMessaggio criptato:", base64.b64encode(encrypted).decode('utf-8'))
print("\nMessaggio decriptato:", decrypted.decode('utf-8'))
```

E successivamente il file dirma.py:

```python
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives import serialization
import base64
import os
os.system('clear')

# Carica la chiave privata
with open('private_key.pem', 'rb') as key_file:
    private_key = serialization.load_pem_private_key(
    key_file.read(),
    password=None)

# Carica la chiave pubblica
with open('public_key.pem', 'rb') as key_file:
    public_key = serialization.load_pem_public_key(key_file.read())

message = 'Ciao, Epicode spacca!'

#Firma con la chiave privata
signed = private_key.sign(message.encode(), padding.PKCS1v15(), hashes.SHA256())

# Verifica della firma con la chiave pubblica
try:
    encrypted_b64 = base64.b64encode(signed).decode('utf-8')
    public_key.verify(signed, message.encode(), padding.PKCS1v15(), hashes.SHA256())
    print("\nBase64 della firma:", encrypted_b64)
    print("\nMessaggio originale da confrontare:", message)
    print("\nLa firma è valida.")
except Exception as e:
    print("\nLa firma non è valida.", str(e))

```