

## REPORT S3/L3

---

### TRACCIA

Dato il seguente messaggio trovare la chiave e decifrarlo:

HSNFRGH

QWJhIHZ6b2VidHl2bmdyIHb1ciB6ciBhciBucHBiZXRi

Se uso il cifrario di cesare con chiave 3 per il primo messaggio, spostando di 3 lettere a sinistra otterrò:

### EPICODE

Per quanto riguarda il secondo messaggio lo converto in base64 e utilizzo un famoso cifrario di cesare che è il rot13:

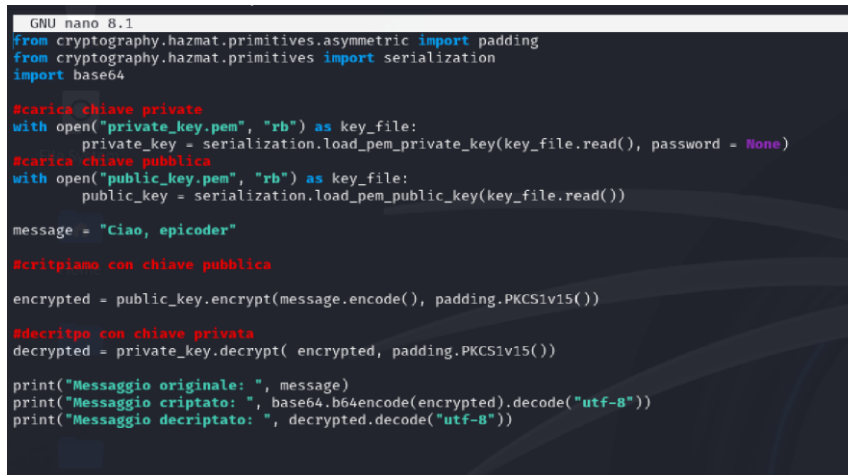
*Non imbrogliate che me ne accorgo*

### ESERCIZIO BONUS

---

Per prima cosa andremo a generare le chiavi pubblica e privata direttamente da terminale, generando due file, rispettivamente *private\_key.pem* e *public\_key.pem*.

Andiamo a creare il file *encdec.py* con il comando *touch* e andremo ad editarlo con *nano* come segue.



```
GNU nano 8.1
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import serialization
import base64

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

message = "Ciao, epicoder"

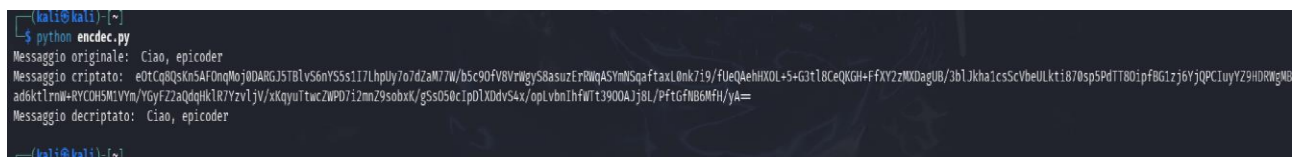
#crittiamo con chiave pubblica
encrypted = public_key.encrypt(message.encode(), padding.PKCS1v15())

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

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

Il modulo *padding* mi permette di aggiungere dati ad un messaggio in maniera da renderlo della lunghezza tale per essere crittografato, il modulo *serialization* mi permettere di leggere chiavi pubbliche e private ed infine *base64* mi permette di leggere in base 64. Con open invece leggeremo dai file le variabili trasferendole in *private\_key* e *public\_key*.

Dopo aver scritto il programma con python lo andremo ad eseguire per everificare che effettivamente gli output siano il messaggio originale, il messaggio criptato e la sua versione decriptata ovviamente uguale al messaggio originale.



```
(kali@kali)~$ python encdec.py
Messaggio originale: Ciao, epicoder
Messaggio criptato: e0tCp0QsKn5AF0nqMoj0DARGJ5TBlvS6nYS5s1I7Lhplu7o7d2aM77W/b5c90Fv8VrMgyS8asuzErRwqASymNSqaftaxL0mk7i9/fUeQ4ehHXDL+5+G3t18ceQKGH+FFxY2zMXDagUB/3b1Jkha1csScVbeULkt1870sp5PdtT80ipf0G1zj6YjQPC1uyYZ9H0R0gW8
adbkt1rni+RYCOH5MLVYm/YcyF22aQdHk1R7Yzvljv/xQyutTwcZWPD7i2mnZ9sobxk/gS050cIpD1XDdvS4x/opLvbniHfuit3900AJj0L/PftGfNB6MfU/yA=
Messaggio decriptato: Ciao, epicoder
(kali@kali)~$
```