Report S3/L4

Matteo Congiu

Esercizio di oggi: Crittografia.

Dato un messaggio cifrato cercare di trovare il testo in chiaro:

1) Messaggio cifrato: **"HSNFRGH"**

Svolgimento

Utilizzando il cifrario di Cesare, ogni lettera viene sostituita con la lettera di 3 posti dopo.

H=E;

S=P;

N=I;

F=C;

R=O;

G=D;

H=E.

HSNFRGH=EPICODE

2) Esercizio di oggi Criptazione e Firmatura con OpenSSLe Python:

Obiettivi dell'esercizio:

- Generare chiavi RSA.
- Estrarre la chiave pubblica da chiave privata.
- Criptare e decriptare messaggi.
- Firmare e verificare messaggi.

Strumenti utilizzati:

- OpenSSLper la generazione delle chiavi.
- Libreria cryptography in Python.

Svolgimento:

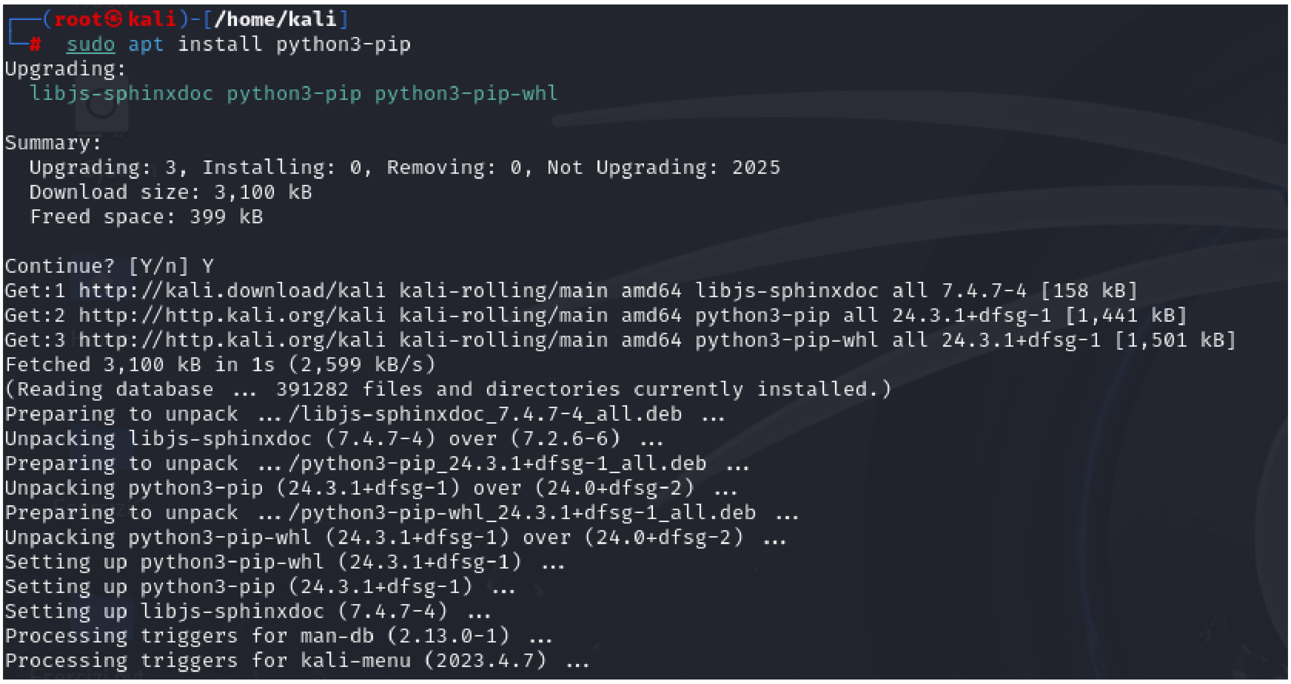
Per iniziare sono entrato nella modalità root e poi ho installato OpenSSL con i comandi:

-sudo aptupdate

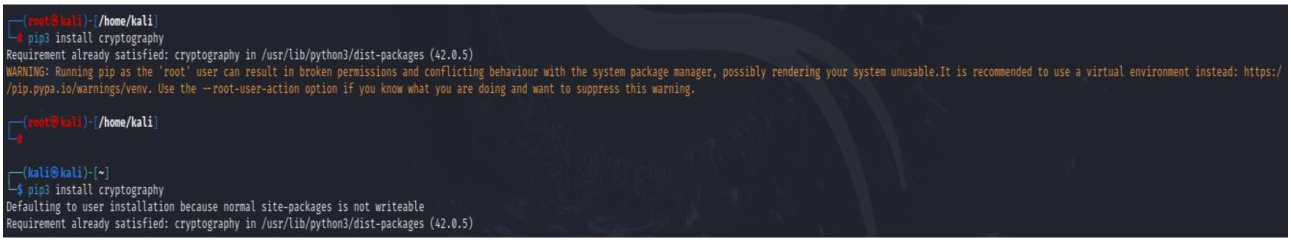
-sudo apt installopenssl

Ho installato la libreria Python con i comandi:

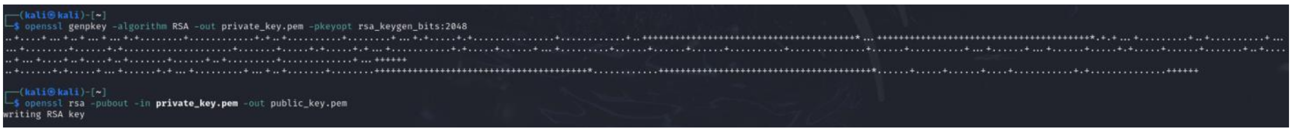
-sudo apt installpython3-pip



-pip3 installcryptography -→ Con questo comando la libreria mi risultava gia presente



Ho digitato per primo il comando per generare la chiave privata RSA e successivamente il comando per estrarre la chiave pubblica:



Con il comando -touch ho creato il file Python con nome encdec.py e ho iniziato a scriverlo:

-**from cryptography.hazmat.primitives.asymmetric import padding**

Padding è un processo che aggiunge dati extra a un messaggio per far sì che abbia una lunghezza adeguata per essere crittografato.

-**from cryptography.hazmat.primitives import serialization**

Serialization per leggere le chiavi private e pubbliche

-**import base64**

base64 per visualizzare il criptato in base64

Con **open** leggiamo le chiavi e le trasferiamo nelle variabili: ***private_key***; ***public_key***.

-with open('private_key.pem', 'rb') as key_file: Apre il file private_key.pem in modalità lettura binaria '*rb*'.

-private_key= serialization.load_pem_private_key(Carica la chiave privata dal file pem.

-key_file.read(),

-password=None)

-With open('public_key.pem', 'rb') as key_file: Apre il file public_key.pem in modalità lettura binaria (rb).

-public_key= serialization.load_pem_public_key(key_file.read()) Carica la chiave pubblica dal file pem.

-message ='Ciao, Epicode spacca!' Messaggio che si vuole criptare.

-encrypted= public_key.encrypt(message.encode(), padding.PKCS1v15())

Utilizza la chiave pubblica per criptare il messaggio.

-decrypted= private_key.decrypt(encrypted, padding.PKCS1v15())

Utilizza la chiave privata per decriptare il messaggio.

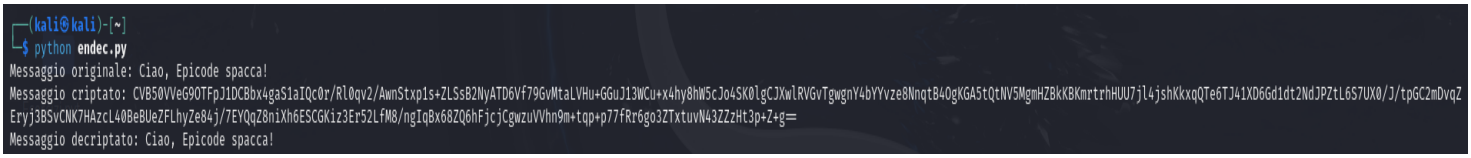
-print("Messaggio originale:", message) Stampa il messaggio originale

-print("Messaggio criptato:", base64.b64encode(encrypted).decode('utf-8')) Stampa il messaggio criptato in base64

-print("Messaggio decriptato:", decrypted.decode('utf-8')) Stampa il messaggio decriptato

```python
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import serialization
import base64
# 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 ("Messaggio originale:", message)
print ("Messaggio criptato:", base64.b64encode(encrypted).decode('utf-8'))
print ("Messaggio decriptato:", decrypted.decode('utf-8'))
```

Ho avviato il programma con il commando: **-python endec.py**



Successivamente ho creato il file **firma.py** attraverso il comando touch e l'ho aperto con nano, ho scritto il programma come quello precedente apportando alcune modifiche che evidenzierò:

-from cryptography.hazmat.primitives.asymmetric import padding

-from cryptography.hazmat.primitives import hashes Serve a creare l'hash del messaggio

-from cryptography.hazmat.primitives import serialization

-import base64

-with open('private_key.pem', 'rb') as key_file:

-private_key= serialization.load_pem_private_key(

-key_file.read(),

-password=None)

-With open('public_key.pem', 'rb') as key_file:

-public_key= serialization.load_pem_public_key(key_file.read())

-message ='Ciao, Epicode spacca!'

-signed = private_key.sign(message.encode(), padding.PKCS1v15(), hashes.SHA256())

Viene utilizzata la chiave privata per firmare il messaggio e viene usato l'hash SHA256

-try:

-encrypted_b64 = base64.b64encode(signed).decode('utf-8')

-public_key.verify(signed, message.encode(), padding.PKCS1v15(), hashes.SHA256())

Utilizza la chiave pubblica per verificare la firma digitale

- print("Base64 della firma:", encrypted_b64) Stampa la firma in formato base64

-print("Messaggio originale da confrontare:", message) Stampa il messaggio originale

-print("La firma è valida.") Stampa la convalida della firma

-except Exception as e:

-print("La firma non è valida.", str(e)) Se ci dovessero essere difformità viene stampata la scritta: non è valida

```python
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives import serialization
import base64
# 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 ("Base64 della firma:", encrypted_b64)
    print ("Messaggio originale da confrontare:", message)
    print ("La firma è valida.")
except Exception as e:
    print ("La firma non è valida.", str(e))
```

Ho avviato il programma con –**python firma.py**