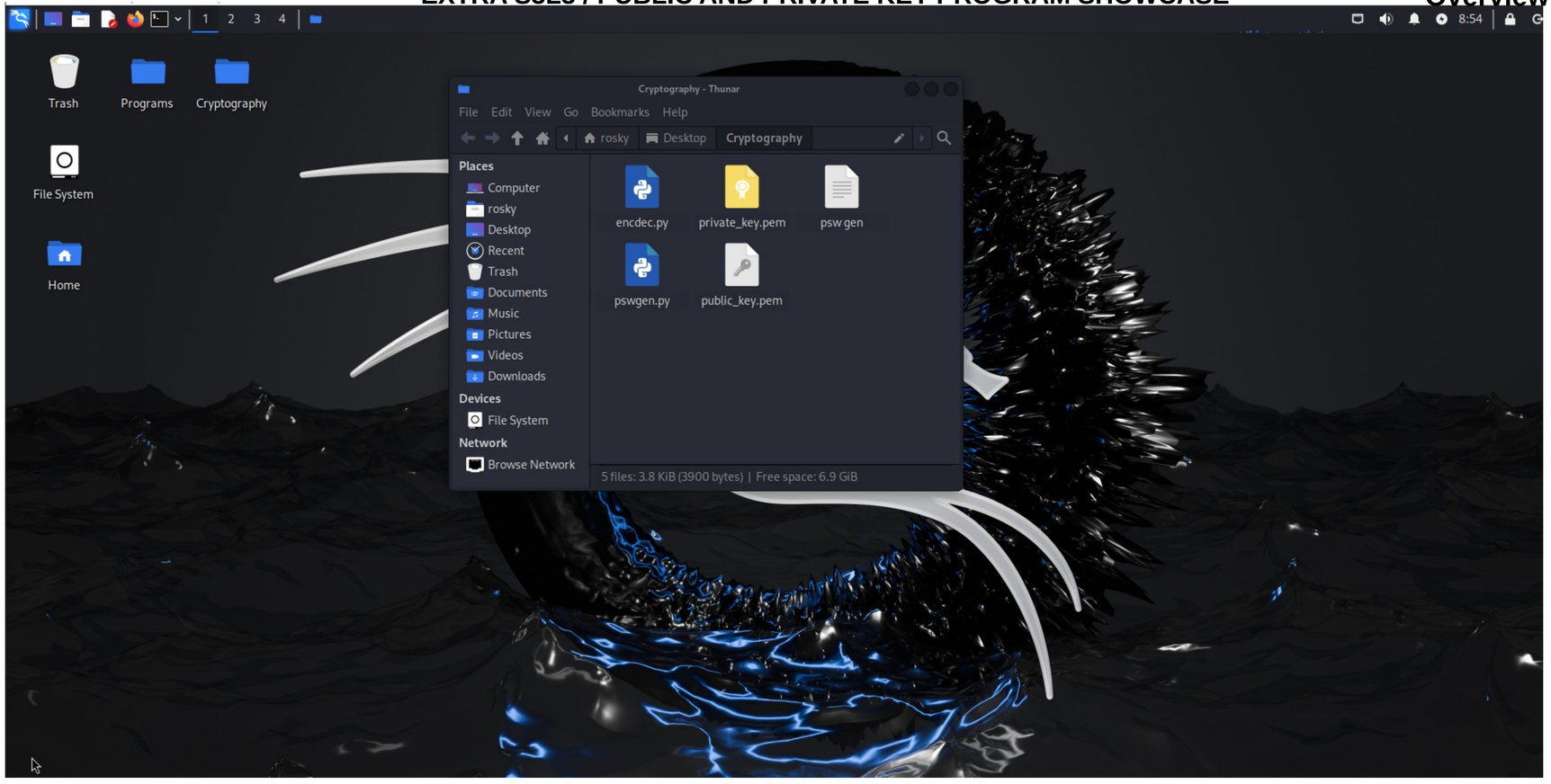
EXTRA S3L3 / PUBLIC AND PRIVATE KEY PROGRAM SHOWCASE

Overview



Comandi usati: (without “) not in order
“su -”
“sudo Apt install python3-cryptography”
“”Pip 3 install cryptography (tried)”
“openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048”
“openssl rsa -pubout -in private_key.pem -out public_key.pem”
“sudo apt install openssl”
“sudo passwd”
“sudo apt update”
“sudo apt full-upgrade -y”
“python encdec.py”
“python firma.py”

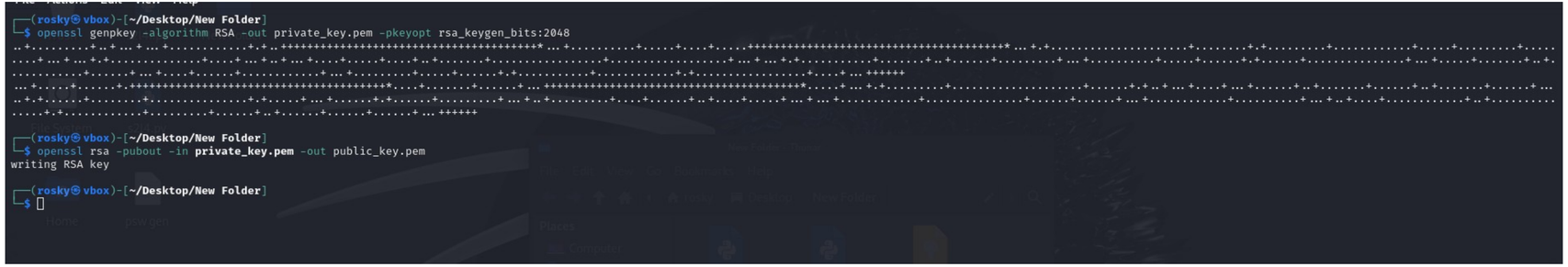
Revisited version! 2.0! With suggestion to improve the program (without a showcase just ipothetical)

In questo esercizio prima bisogna installare con pip3 install cryptography (dependencies) ma non funzionava, quindi ho usato apt install python3-cryptography dopo aver ottenuto I permessi di root con su - .

```
┌──(rosky㉿vbox)-[~]
└─$ pip3 install cryptography
error: externally-managed-environment

× This environment is externally managed
╰─> To install Python packages system-wide, try apt install
    python3-xyz, where xyz is the package you are trying to
    install.

    If you wish to install a non-Kali-packaged Python package,
    create a virtual environment using python3 -m venv path/to/venv.
    Then use path/to/venv/bin/python and path/to/venv/bin/pip. Make
    sure you have pypy3-venv installed.

    If you wish to install a non-Kali-packaged Python application,
    it may be easiest to use pipx install xyz, which will manage a
    virtual environment for you. Make sure you have pipx installed.

    For more information, refer to the following:
    * https://www.kali.org/blog/python-externally-managed/
    * /usr/share/doc/python3.12/README.venv

note: If you believe this is a mistake, please contact your Python installation or OS distribution provider. You can override this, at the risk of breaking your Python installation or OS, by passing --break-system-packages.
hint: See PEP 668 for the detailed specification.

┌──(rosky㉿vbox)-[~]
└─$ apt install python3-cryptography
Error: Could not open lock file /var/lib/dpkg/lock-frontend - open (13: Permission denied)
Error: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), are you root?
```

“Per poi scoprire che era già installato, che mi serva di lezione.”

```
hint: See PEP 668 for the detailed specification.

┌──(root㉿vbox)-[~]
└─# apt install python3-cryptography
python3-cryptography is already the newest version (43.0.0-1).
python3-cryptography set to manually installed.
The following packages were automatically installed and are no longe
  fonts-liberation2  libblosc2-3              libgdal34t64    libgt
  freerdp2-x11       libboost-iostreams1.83.0  libgeos3.12.2   libgt
```



Procediamo nel creare nella stessa cartella dove vengono utilizzati gli script per la cifratura/decifratura del messaggio, altrimenti il programma non troverebbe le chiavi da usare (la cartella new folder è stata rinominata cifratura in seguito).

```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())
# Input
message = input("Inserisci il messaggio da criptare: ")

# 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'))
```

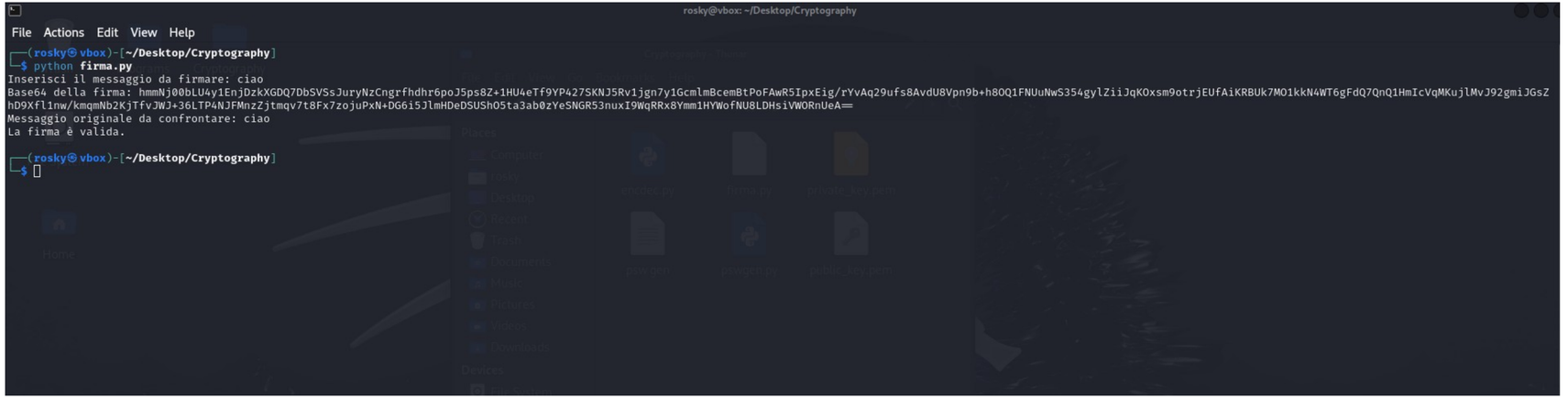
Una volta create le chiavi, si procede con la creazione del primo programma per la criptazione del messaggio.
Importiamo due librerie, padding,serialization e base 64.
Il padding fornisce schemi necessari per l'algoritmo RSA garantendo che I messaggi abbiano una lunghezza adeguata.
Serialization è utilizzato per caricare o salvare le chiavi pubblice e private da file.
Base64 serve per convertire dati Binari in ASCII (stringhe)
Altrimenti non potrebbe visualizzarlo.
Con open leggiamo le chiavi e le trasferiamo nelle variabili private_key e public_key.

```
┌──(rosky㉿vbox)-[~/Desktop/Cryptography]
└─$ python encdec.py
Inserisci il messaggio da criptare: adoro python
Messaggio originale: adoro python
Messaggio criptato: PKAqPJ2FZ38S9T4aXZDq1M4U4nCSNjsoRP65AifvDuCcMi8TzQYsg48ib1F0JMvIh2WyV/bY3ovdQu7k3+KPkklF7KgZcyWjfoygSrU7iS4TjegE/g7YM1jTFpmgYvjYxzmtL+NfDVSHcCRAojSJc0DcCUok/W/REcL0GBQMzkf411Jf2ntkxGWtCFhi1Gcf/4w/WPYnA908Frch7L0eGwVQ1YBvUX+u3Tovg/byoDnRhdPAN5oGe7GDrx6lv/jlWpDyhLCs7E6S/HrCbMyljYUSpUtkyDKYl1Vc2n5DdF+mKPafLPUiC4VhDrBN9TJ13gptSVmP0fAIVohKl3AqNQ==
Messaggio decriptato: adoro python

┌──(rosky㉿vbox)-[~/Desktop/Cryptography]
└─$
```

Facciamo partire il primo programma e chiederà input da parte dell'User.
Cifrando in questo caso il messaggio inserito dall'User con la chiave pubblica e lo decritta con la chiave privata ritornando a video il risultato della corretta cifratura e della sua decrittazione.

```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())

# Input
message = input("Inserisci il messaggio da firmare: ")

# 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))

```

Qua abbiamo invece un programma per "firmare". C'è comunque da dire che entrambi I programmi potrebbero usare tkinter per la finestra di selezione di file e askopenfilename() per far aprire la finestra di dialogo e permettere all'User di scegliere un file e firmarlo/criptarlo/decriptarlo a scelta (non ho tempo per fare un improving i'm sorry)
Utilizzeremo le stesse chiavi pubbliche e private create in precedenza.
C'è solo una libreria in più che è stata utilizzata qui ed è hashes che serve per fornire accesso agli algoritmi di hashing come SHA-256 (standard) o SHA-512.
Open svolge lo stesso ruolo, e verrà ugualmente chiesto all'User tramite video di inserire un messaggio da firmare.



Una volta inserito il messaggio da firmare lo confronterà tramite hashing e se l'hash risulta identico ed il file non è stato compromesso allora confermerà la firma (o l'integrità del messaggio/file/programma).

"Overview Finale"