Prabin Shrestha

T00664996

Comp 3260

Assignment 1

October 2, 2024

1) **Introduction**

The purpose of this assignment was to implement the RSA encryption algorithm. It involved generating public and private keys, encrypting file content with public keys and using private keys to decrypt them. Encryption is a crucial aspect of modern computing that helps ensure confidentiality and integrity of data. RSA(Rivest-Shamir-Adleman) was first publicly described in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman of the Massachusetts Institute of Technology, through the 1973 creation of a public key algorithm by British mathematician Clifford Cocks. RSA is widely used and popular due to its reliance on the mathematical difficulty of factoring large prime numbers. Multiplying two prime numbers is easy, but determining the original prime numbers from the total or factoring is considered infeasible due to the time it would take using even today's computers.

2) **Setup**

Libraries Used:

- Cryptography library:
  → Install cryptography library with ***pip install cryptography***
  The following library to be exact from cryptography library.

```
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization
```

- Easygui: for file selection through dialog boxes. To install it just use the ***pip install easygui*** command on the terminal.
- OS library: Provides functions to interact with the operating system. It's used for verifying file paths and checking whether a file exists.

3) **Implementation Details:**

i) RSA Key pair generation

→ The RSA key pair is generated using the cryptography library. One key insight during the implementation was the importance of key length and its impact on security. I chose 2048 key length, if say 4096 was chosen, it would offer stronger encryption but would also increase processing time for key generation, encryption and decryption.
Below is the code snippet of this key generation.

```
def PS_generate_rsa_key_pair():
    PS_private_key = rsa.generate_private_key(public_exponent=65537,
key_size=2048)
    PS_public_key = PS_private_key.public_key()
    with open("private_key.pem", "wb") as f:
        f.write(PS_private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption()
        ))
    with open("public_key.pem", "wb") as f:
        f.write(PS_public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
```

```
        ))
```

A key realization here was that the private key can be password-protected. I opted for no encryption during file storage to keep it simple

ii) File Encryption and Decryption

→ The encryption and decryption processes both use the RSA key pair and a straightforward encryption/ decryption method with the public and private keys. The encryption function takes file path and public key as argument and reads the file content as binary data. Then encrypt using RSA, then saves the encrypted message to same file path with .enc extension.

```
def PS_encrypt_file(PS_file_path, PS_public_key):
    with open(PS_file_path, 'rb') as file:
        PS_file_data = file.read()
    print(f"Original message from {PS_file_path}:")
    print(PS_file_data.decode())

    PS_encrypted_data = PS_public_key.encrypt(
        PS_file_data,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None

    PS_encrypted_file_path = PS_file_path + ".enc"
    with open(PS_encrypted_file_path, 'wb') as file:
        file.write(PS_encrypted_data)

    return PS_encrypted_file_path
```

Decryption is done in the same way. Decryption function takes file path of encrypted file and private key as argument, then decrypted file using RSA and then save decrypted message to same file path with .dec extension

```
ef PS_decrypt_file(PS_encrypted_file_path, PS_private_key):
    with open(PS_encrypted_file_path, 'rb') as file:
        PS_encrypted_data = file.read()
    PS_decrypted_data = PS_private_key.decrypt(
        PS_encrypted_data,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )

    PS_decrypted_file_path = PS_encrypted_file_path.replace(".enc", ".dec")
    with open(PS_decrypted_file_path, 'wb') as file:
        file.write(PS_decrypted_data)

    return PS_decrypted_file_path
```

One interesting aspect I encountered during implementation of this project was padding. RSA encryption requires padding to make sure the data being encrypted is randomized and secure.

4) Results

→ The program successfully generated RSA key pair and saved them as .pem files.

→ Encrypted files of various formats including numbers, symbols and text. Save the files with the .enc extension.

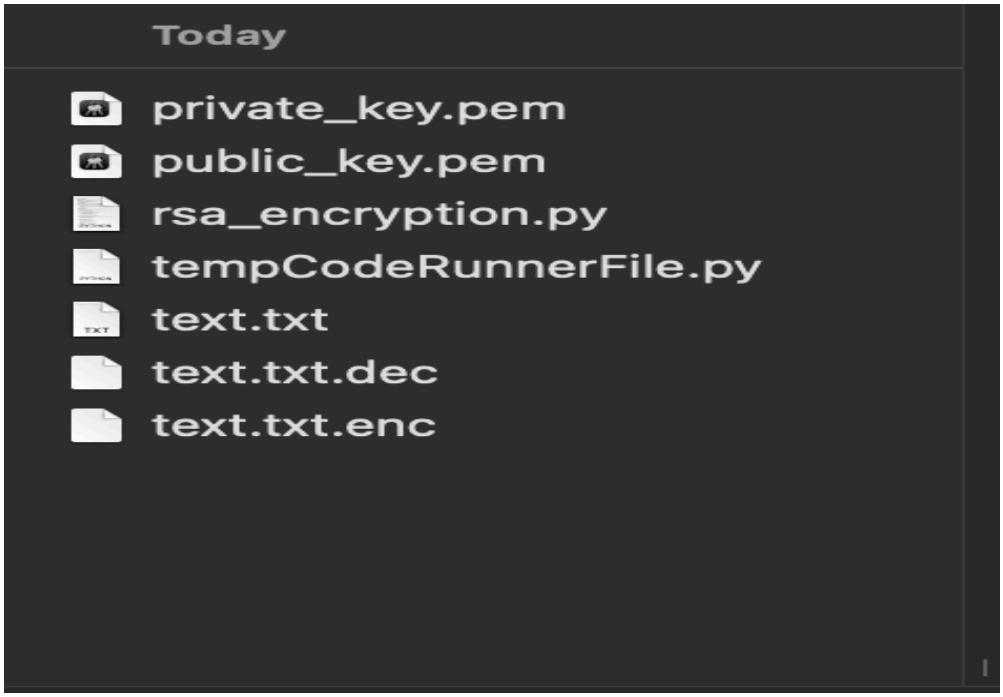
→ Decrypt the encrypted file, and display the decoded contents as well as save it as .dec file

Demo 1:

So, I selected the text.txt file. It reads the content of it and displays it on the console. Then encrypts that text, outputs encrypted message in console as well as save it as text.txt.enc. Then it also decrypts the encrypted message, displays it in the console as well as saves it in a text.txt.dec file.

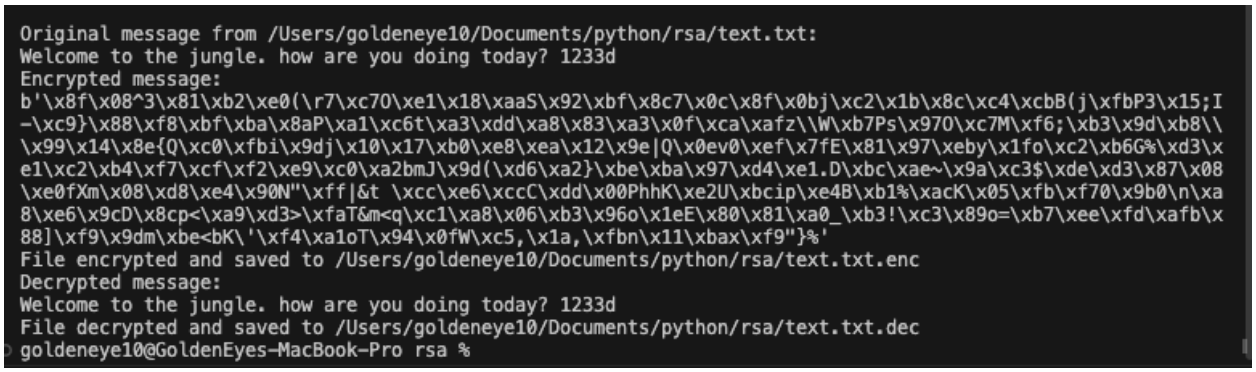
```
Original message from /Users/goldeneye10/Documents/python/rsa/text.txt:
how are you doing today in this beautifucl lsdflf sldkfw weather
Encrypted message:
b'v\xfd\xa639\xf0\x00>j\xcd>\xbb\xb2)\x81j-\x8a3\x05\x93K(\x82\x9c\xb3\\\x8b\x82k\xf81P\xed2\xd6x\xd2k=\x17
\xe5\xbbc\xea\xa5\x0c]\xf0\xf2Gj\xb9\xf8\xb6\x87\x18U\xe80m\xd9\xbd>\xf3\xb6x|]\xc1B\x95\x8c^\x18j<a1\xddMl
\t\x16\x0fV\xf3\x95\xe3\xce`\xf4\x03\xd5\x839\xc1\x0c:\x93S\xf4\x8fD\xf8\xdf\x0f\xf7:n\x83\xd9a3b\xef\xf2\x
cd\x1d\xda\xa8$:"\xb3\xda`\xb9\x04\xddm}\x1e\x81R\xeb\xb9\xdbP\xa5H\x98:\x03\xab3\xaa\xb7\xc6gY\xc4v\x04F\x
18\x80\xf4\xe8$\x97\xe3\xa0z\xbb\xd9\xa0\xf5\xf2\x8d\xbf\xc2\x9b?\x9aZX\xe2\x90+\xa3\xc9\xed0\x83\n\xbb\xa1
s#Lz\'ro\x81\x15\x1b\xbe\xdf\xd9\x89\xb5\xa6*\x99C\x7fxn\x15o\xeawE\xbd-\xabc\x0e\xb6c\xb7\xec\xb6\x19\x18\
xc9\xd5nXC\xf3u@5\xda\xfe\xaa\xa3\xcf\x8c\xf1e\xc4\x17\xffM\x881~\x15\x894UC'
File encrypted and saved to /Users/goldeneye10/Documents/python/rsa/text.txt.enc
Decrypted message:
how are you doing today in this beautifucl lsdflf sldkfw weather
File decrypted and saved to /Users/goldeneye10/Documents/python/rsa/text.txt.dec
goldeneye10@GoldenEyes-MacBook-Pro rsa %
```

This is the result on console



These are the files created by the application.

Demo 2:



5) Conclusion

This project leveled up my understanding of cryptography, specially about how asymmetric encryption works in practical scenarios. I learned that while RSA is a powerful tool, it's mainly suited for encrypting small data. Padding schemes, key sizes, and secure key management are all critical to ensure the overall security of an RSA encryption. By implementing this program, I became more familiar with the cryptography library, key handling, and the importance of choosing the right encryption parameters.

6) References

https://dev.to/aaronktberry/generating-encrypted-key-pairs-in-python-69b

https://cryptography.io/en/latest/development/custom-vectors/rsa-oaep-sha2/#rsa-oaep-sha2-vector-creation

Stack overflow