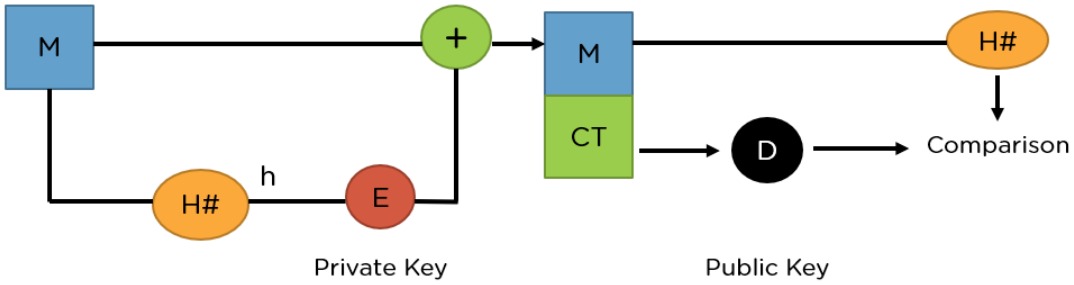
INTRODUCTION

Data privacy is important since it can contain sensitive and valuable information. The best way to ensure data security and prevent it from being exposed to hackers or unauthorized users who could fall victim to cyber threats or attacks is to alter the data so that, even if someone were to gain access to it, it would be impossible for them to extract plain text from it. This is where the RSA algorithm comes in handy.

WHAT IS RSA? https://www.encryptionconsulting.com/education-center/what-is-rsa/

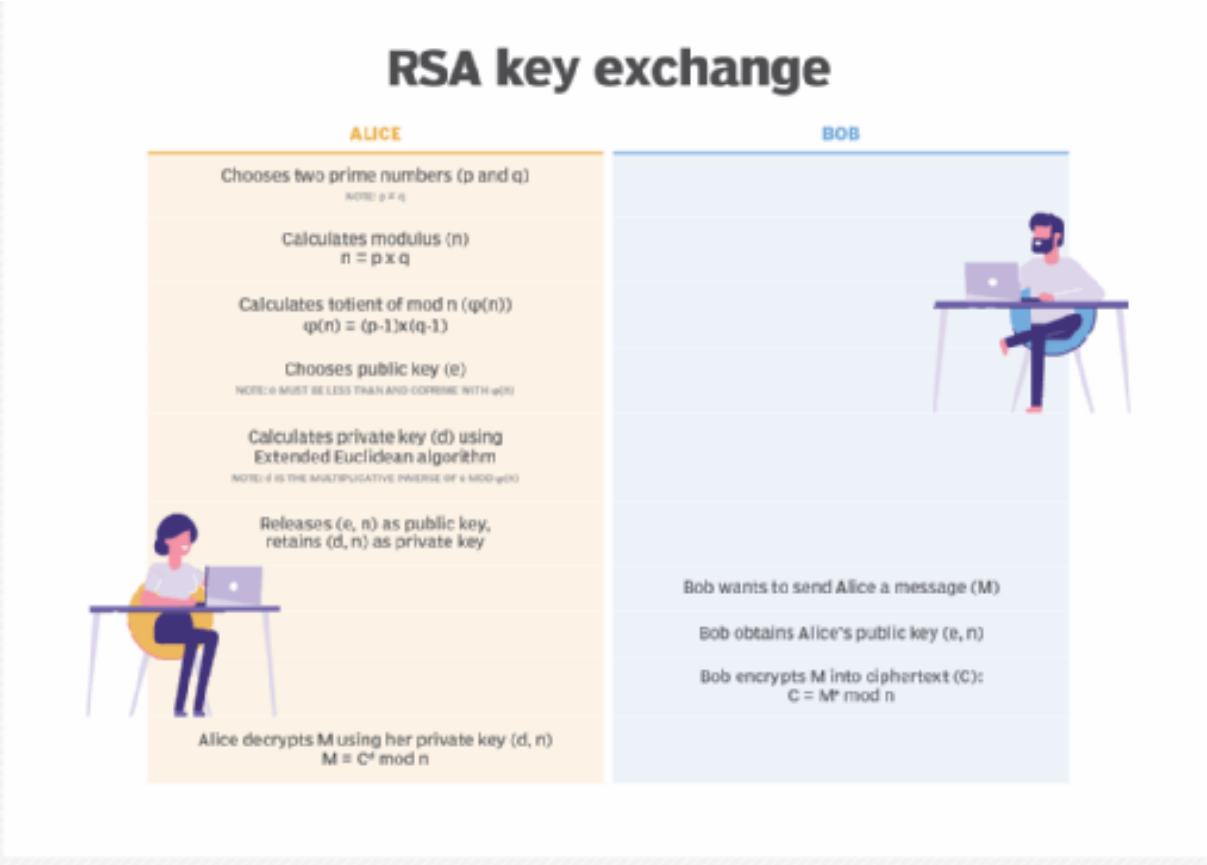
The Rivest-Shamir-Adleman (RSA) encryption algorithm is an asymmetric encryption algorithm that is widely used in many products and services. Asymmetric encryption uses a key pair that is mathematically linked to encrypt and decrypt data. A private and public key is created, with the public key being accessible to anyone and the private key being a secret known only by the key pair creator. With RSA, either the private or public key can encrypt the data, while the other key decrypts it. This is one of the reasons RSA is the most used asymmetric encryption algorithm.



HOW DOES RSA WORK?

Numerous benefits are available to RSA users whether they choose to encrypt using the private or public key. Data encrypted with the public key requires decryption with the private key. If the recipient of the data provides the data sender their public key, this is ideal for delivering sensitive data across a network or Internet connection. The private information is then sent to the recipient after the sender encrypts it using the public key. Only the owner of the private key may decode sensitive data since it was encrypted using the public key. So, even if the data were captured during transmission, only the intended receiver may decode it.

The technical details of RSA work on the idea that it is easy to generate a number by multiplying two sufficiently large numbers together (p and q) but factorizing that number back into the original prime numbers is extremely difficult. The public and private key are created with two numbers, one of which is a product of two large prime numbers. Both use the same two prime numbers to compute their value. RSA keys tend to be 1024 or 2048 bits in length, making them extremely difficult to factorize, though 1024-bit keys are believed to be breakable soon.



ALGORITHM FOR ENCRYTION

A message m (number) is encrypted with the public key (n,e) by calculating:

$C = m^e \pmod n$

ALGORITHM FOR DECRYPTION

Decrypting with the private key (n,d) is done with:

$M = c^d \pmod n$

GUI (Graphical User Interface) https://www.britannica.com/technology/graphical-user-interface

A graphical user interface (GUI), is a computer program that enables a person to communicate with a computer through the use of symbols, visual metaphors, and pointing devices. Best known for its implementation in Apple Inc.'s Macintosh and Microsoft Corporation's Windows operating system, the GUI has replaced the arcane and difficult textual interfaces of earlier computing with a relatively intuitive system that has made computer operation not only easier to learn but more pleasant and natural. The GUI is now the standard computer interface and its components have themselves become unmistakable cultural artifacts.

IMPLEMENTATION

1) Importing:

```
import tkinter as tk
from tkinter import filedialog, messagebox
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.asymmetric import rsa, padding
```

This part of the code imports specific modules and functions required for building a graphical user interface (GUI) and implementing encryption/decryption functionality using the RSA algorithm from the cryptography library.

Here's a breakdown:
"import tkinter as tk:" Imports the Tkinter library and assigns it an alias tk for ease of use.

"from tkinter import filedialog, messagebox": Specifically imports filedialog and messagebox modules from Tkinter. These modules are used to prompt users to select files and display messages/alerts in the GUI.

"from cryptography.hazmat.backends import default_backend:" Imports the default backend module from the cryptography library. This module provides default cryptographic functionality, allowing access to underlying cryptographic primitives.

"from cryptography.hazmat.primitives import serialization, hashes:" Imports modules related to serialization and hash functions from cryptography.hazmat.primitives. These modules are essential for serializing cryptographic keys and performing cryptographic hash operations.

”from cryptography.hazmat.primitives.asymmetric import rsa, padding”: Specifically imports modules related to asymmetric encryption (RSA) and padding mechanisms from cryptography.hazmat.primitives.asymmetric. These modules are crucial for implementing RSA encryption and padding methods for secure data transmission.

2.

```
    # Key generation (moved here for accessibility)
    self.private_key, self.public_key = self.generate_key_pair()

def generate_keys(self):
    private_key, public_key = self.generate_key_pair()
    self.save_key_to_file(private_key, 'private_key.pem')
    self.save_key_to_file(public_key, 'public_key.pem')
    messagebox.showinfo("Keys Generated", "RSA key pair generated and saved.")

def generate_key_pair(self):
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )
    public_key = private_key.public_key()

    private_pem = private_key.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.PKCS8,
        encryption_algorithm=serialization.NoEncryption()
    )
    public_pem = public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    )

    return private_pem, public_pem

def save_key_to_file(self, key, filename):
    with open(filename, 'wb') as key_file:
        key_file.write(key)
```

This code is a Python script for generating RSA key pairs and saving them as PEM (Privacy Enhanced Mail) files. Here's a simple breakdown:

- The generate_key_pair method creates an RSA private key and its corresponding public key with a key size of 2048 bits and a public exponent of 65537.
- The private and public keys are then converted to PEM format using the private_bytes and public_bytes methods, respectively.
- The save_key_to_file method takes a key and a filename as parameters and writes the key to a file with the specified filename in binary mode.
- The generate_keys method uses the generate_key_pair method to obtain a key pair, saves both keys to separate files ('private_key.pem' and 'public_key.pem'), and displays a messagebox indicating successful key generation.

3.

```python
# Encryption methods
def encrypt_file(self):
    file_path = filedialog.askopenfilename(title="Select File to Encrypt", filetypes=[("Text files", "*.txt")])
    if file_path:
        public_key_path = filedialog.askopenfilename(title="Select Public Key", filetypes=[("PEM files", "*.pem")])
        if public_key_path:
            with open(file_path, 'rb') as file:
                plaintext = file.read()

            public_key_obj = serialization.load_pem_public_key(open(public_key_path, 'rb').read(), backend=default_backend())
            ciphertext = public_key_obj.encrypt(
                plaintext,
                padding.OAEP(
                    mgf=padding.MGF1(algorithm=hashes.SHA256()),
                    algorithm=hashes.SHA256(),
                    label=None
                )
            )

            encrypted_file_path = filedialog.asksaveasfilename(defaultextension=".enc", filetypes=[("Encrypted files", "*.enc")])
            with open(encrypted_file_path, 'wb') as encrypted_file:
                encrypted_file.write(ciphertext)

            messagebox.showinfo("Encryption Successful", f"File {file_path} encrypted and saved as {encrypted_file_path}.")
```

This code defines a method for encrypting a file using RSA public key encryption:

- The user is prompted to select a file for encryption through a file dialog.
- If a file is selected, the user is then prompted to choose a public key file (in PEM format) using another file dialog.
- If both the file to encrypt and the public key file are selected, the code reads the contents of the chosen file (plaintext), loads the chosen public key from the PEM file, and encrypts the plaintext using the public key.
- The encrypted data (ciphertext) is then saved to a new file, and the user is informed about the successful encryption through a message box.

4.

```
# Decryption methods
def decrypt_file(self):
    file_path = filedialog.askopenfilename(title="Select File to Decrypt", filetypes=[("Encrypted files", "*.enc")])
    if file_path:
       private_key_path = filedialog.askopenfilename(title="Select Private Key", filetypes=[("PEM files", "*.pem")])
       if private_key_path:
           with open(file_path, 'rb') as file:
               ciphertext = file.read()

           private_key_obj = serialization.load_pem_private_key(open(private_key_path, 'rb').read(), password=None, backend=default_backend())
           plaintext = private_key_obj.decrypt(
               ciphertext,
               padding.OAEP(
                   mgf=padding.MGF1(algorithm=hashes.SHA256()),
                   algorithm=hashes.SHA256(),
                   label=None
               )
           )

           decrypted_file_path = filedialog.asksaveasfilename(defaultextension=".dec", filetypes=[("Decrypted files", "*.dec")])
           with open(decrypted_file_path, 'wb') as decrypted_file:
               decrypted_file.write(plaintext)

           messagebox.showinfo("Decryption Successful", f"File {file_path} decrypted and saved as {decrypted_file_path}.")
```

This code defines a method for decrypting a file that has been previously encrypted using RSA public key encryption:

- The user is prompted to select an encrypted file through a file dialog.
- If an encrypted file is selected, the user is then prompted to choose a private key file (in PEM format) using another file dialog.
- If both the encrypted file and the private key file are selected, the code reads the contents of the encrypted file (ciphertext), loads the chosen private key from the PEM file, and decrypts the ciphertext using the private key.
- The decrypted data (plaintext) is then saved to a new file, and the user is informed about the successful decryption through a message box.

5.

```
# File signing methods

def sign_file(self):
    file_path = filedialog.askopenfilename(title="Select File to Sign")
    if file_path:
        try:
            with open(file_path, "rb") as f:
                message = f.read()

            private_key_obj = serialization.load_pem_private_key(self.private_key, password=None, backend=default_backend())

            signature = private_key_obj.sign(
                message,
                padding.PSS(
                    mgf=padding.MGF1(hashes.SHA256()),
                    salt_length=padding.PSS.MAX_LENGTH
                ),
                hashes.SHA256()
            )

            with open(file_path + "_signature", "wb") as f:
                f.write(signature)

            messagebox.showinfo("Signing", "File Signed Successfully.")
        except Exception as e:
            messagebox.showerror("Signing", f"Error Occurred: {e}")
```

This code defines a method for signing a file using RSA digital signature:

- The user is prompted to select a file to sign through a file dialog.
- If a file is selected, the code reads the contents of the file (message).
- The private key is loaded from the instance variable (self.private_key) and is used to sign the message using the PSS (Probabilistic Signature Scheme) padding with SHA256 hashing.
- The signature is then saved to a new file with the original file's name appended with "_signature".
- A message box is displayed indicating successful signing, or an error message is shown if an exception occurs during the signing process.

6.

```
# verification methods
def verify_file(self):
    file_path = filedialog.askopenfilename(title="Select File to Verify")
    if file_path:
        try:
            with open(file_path, "rb") as f:
                message = f.read()

            public_key_obj = serialization.load_pem_public_key(self.public_key, backend=default_backend())

            signature_path = file_path + "_signature"
            with open(signature_path, "rb") as f:
                signature = f.read()

            public_key_obj.verify(
                signature,
                message,
                padding.PSS(
                    mgf=padding.MGF1(hashes.SHA256()),
                    salt_length=padding.PSS.MAX_LENGTH
                ),
                hashes.SHA256()
            )

            messagebox.showinfo("Verification", "Signature is Valid.")
        except Exception as e:
            messagebox.showerror("Verification", f"Verification Error: {e}")
```

This code defines a method for verifying the digital signature of a file using RSA:

- The user is prompted to select a file to verify through a file dialog.
- If a file is selected, the code reads the contents of the file (message) that was presumably signed.
- The public key is loaded from the instance variable (self.public_key), and the corresponding signature file (_signature) is read.
- The public key is then used to verify the signature against the message using the PSS padding with SHA256 hashing.
- If the verification is successful, a messagebox is displayed indicating that the signature is valid; otherwise, an error message is shown if an exception occurs during the verification process.

7.

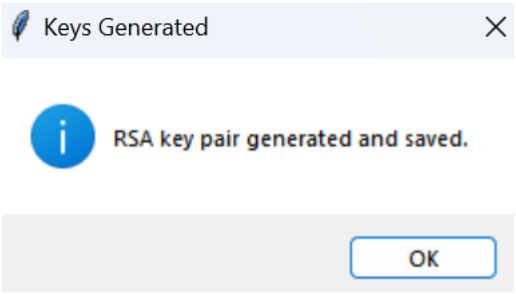
```
def main():
    root = tk.Tk()
    app = RSAEncryptionApp(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

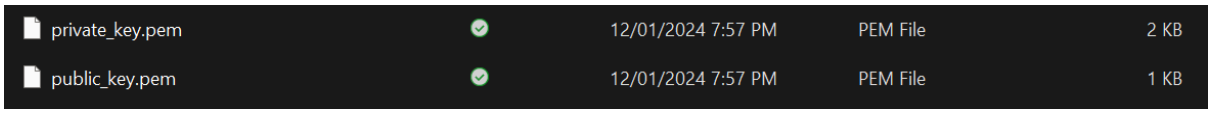
This script serves as an entry point for a Tkinter-based application for RSA encryption. The main function initiates a Tkinter root window and creates an instance of the RSAEncryptionApp class, which encapsulates the application's functionality. The Tkinter main event loop is then invoked with root.mainloop(), enabling the application to handle user interactions. The conditional if _name_ == "_main_": ensures that the main function is executed only when the script is run independently, not when imported as a module.

8.

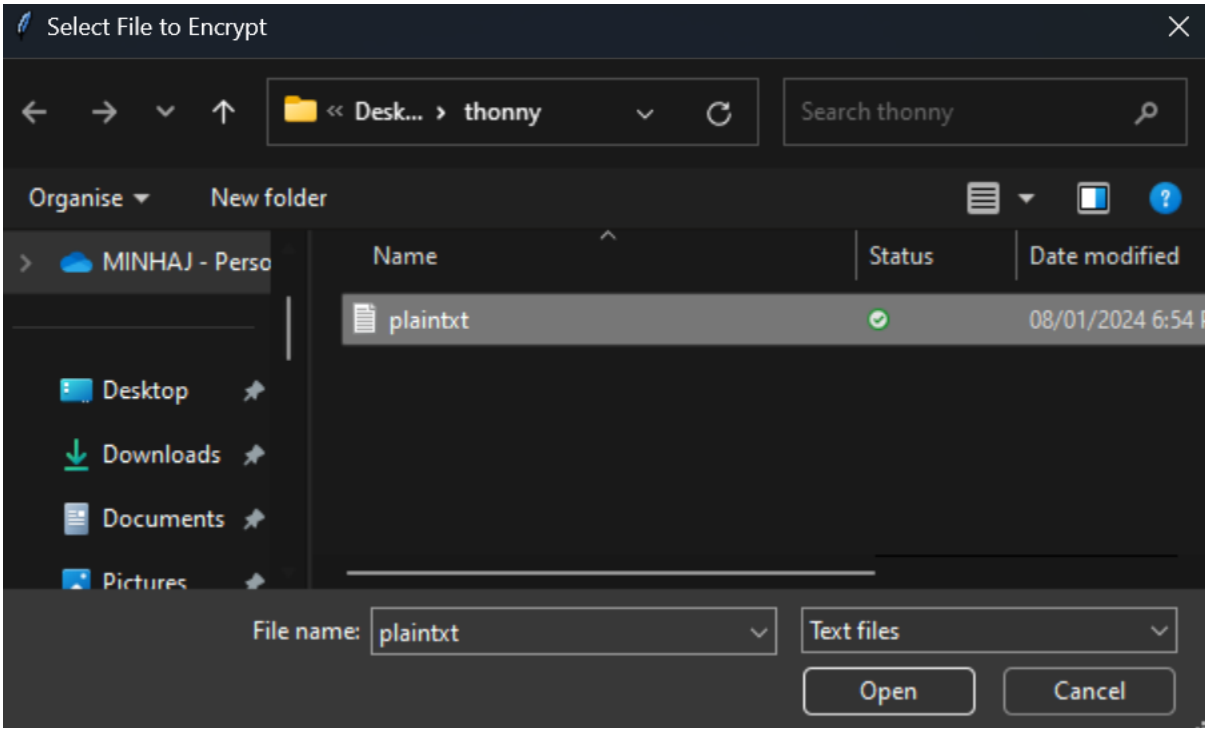
RSA tool with easy application that will allow the user to encrypt files using RSA encryption. The utility features a straightforward terminal with options for the user to sign, decrypt, encrypt, sign and verify files. By clicking on the button, a user is free to choose any.txt file and utilise any of the following.

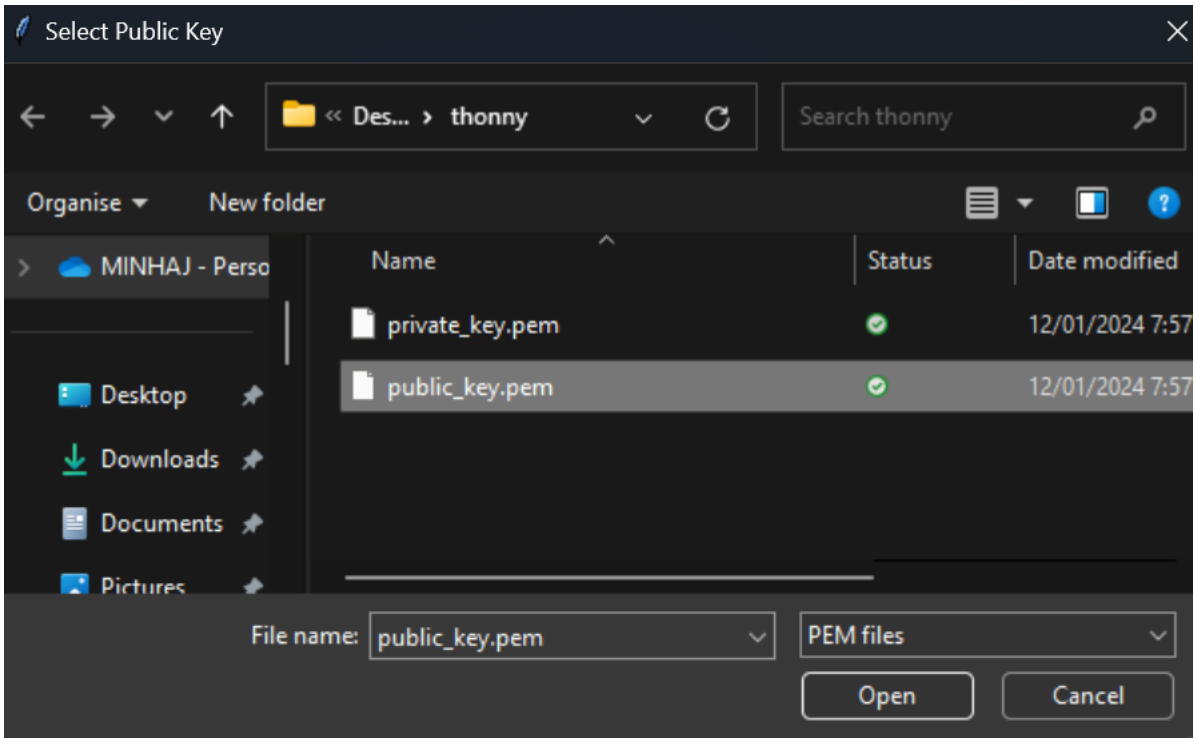
9.

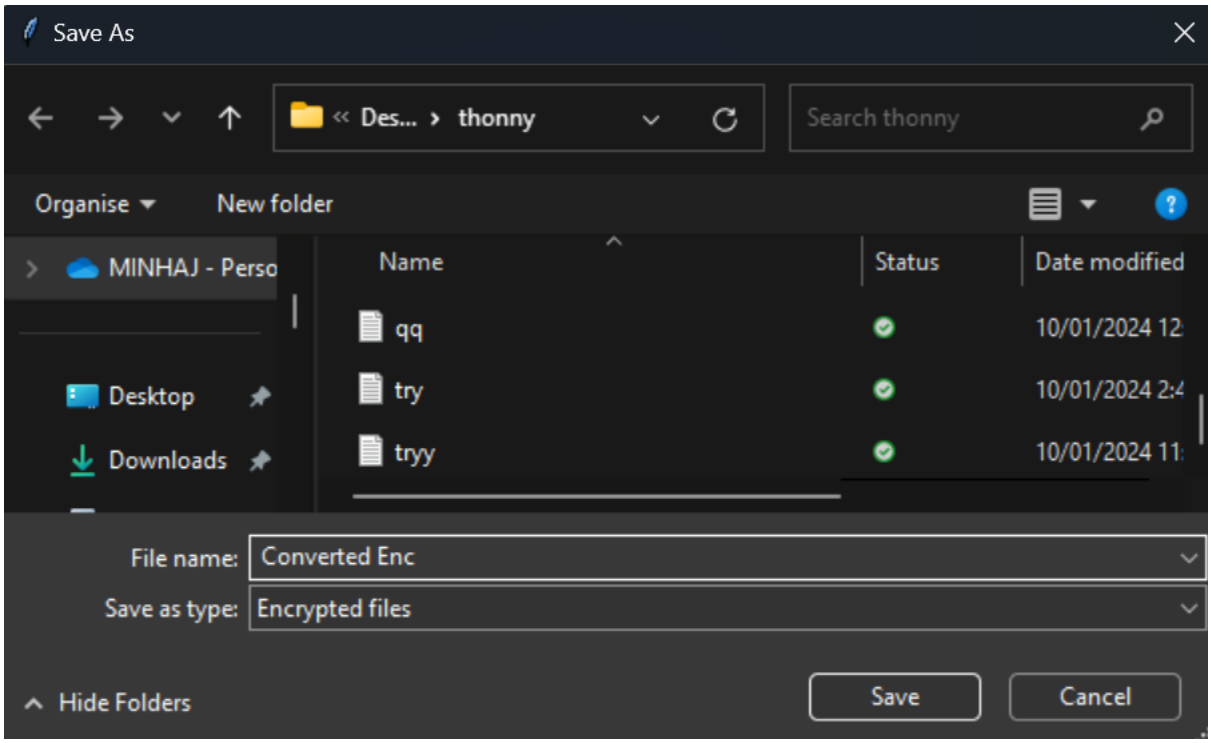


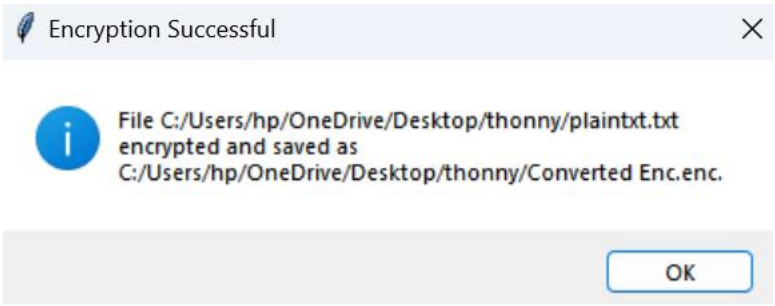


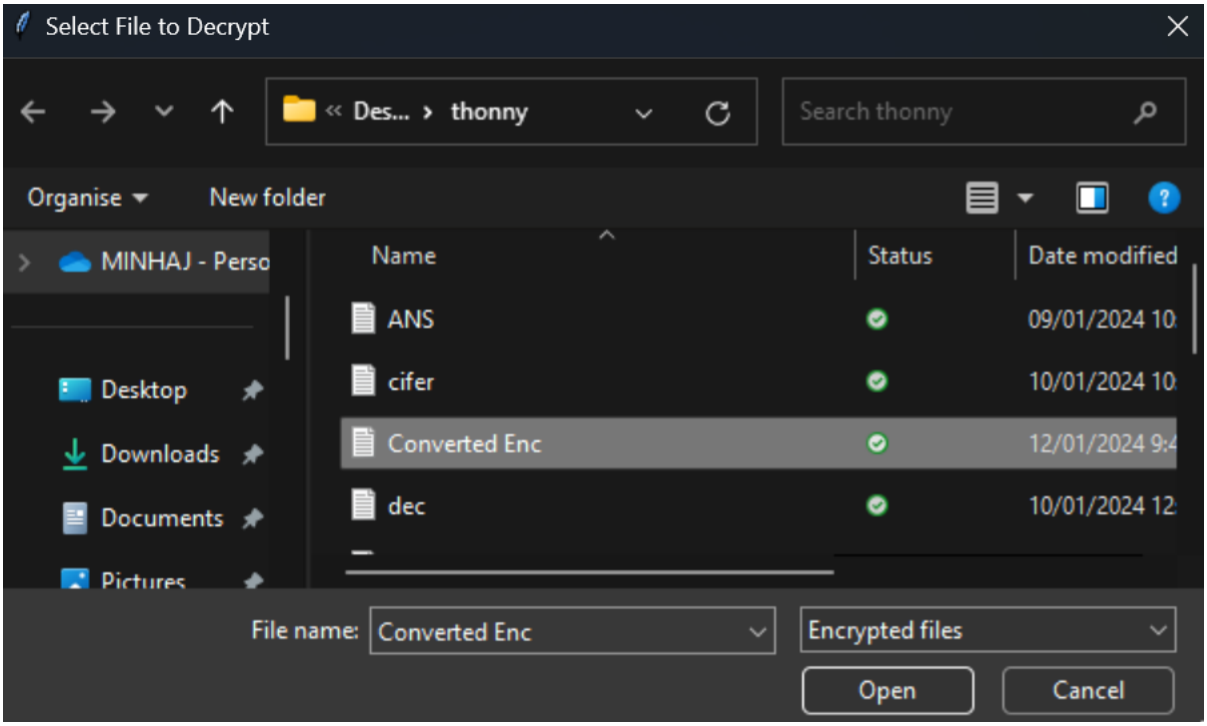
Generated Public and Private Key are Saved in the folder

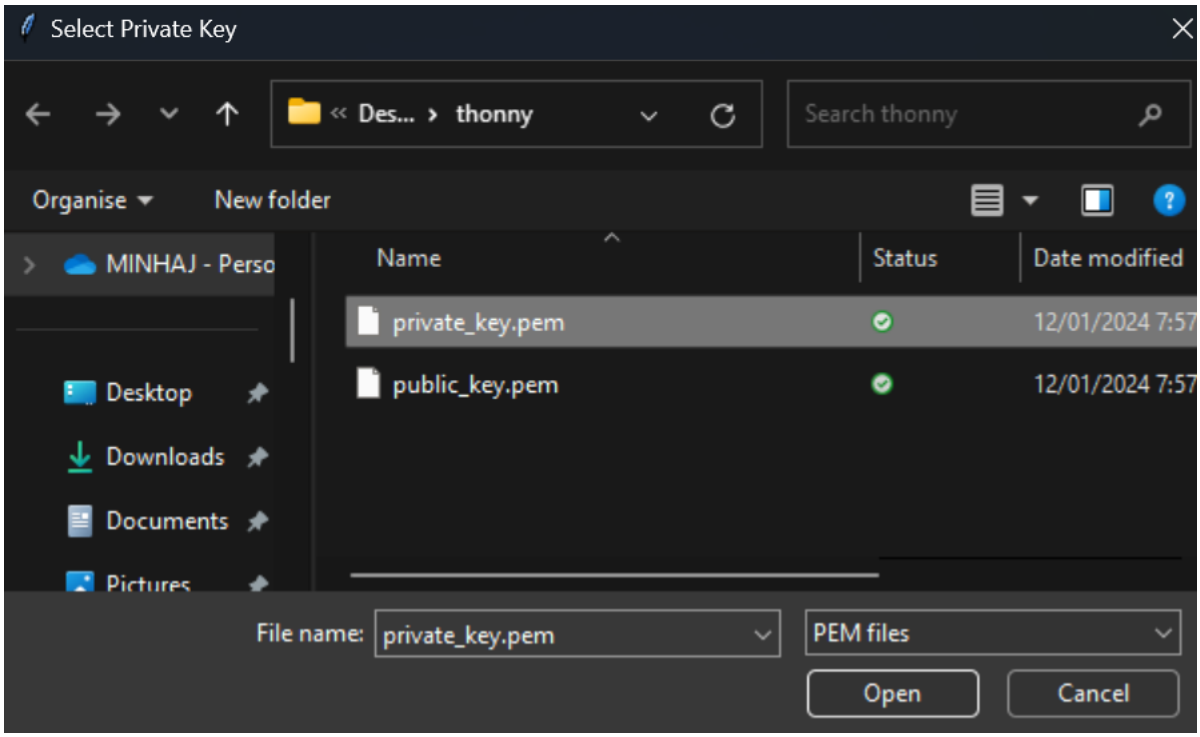


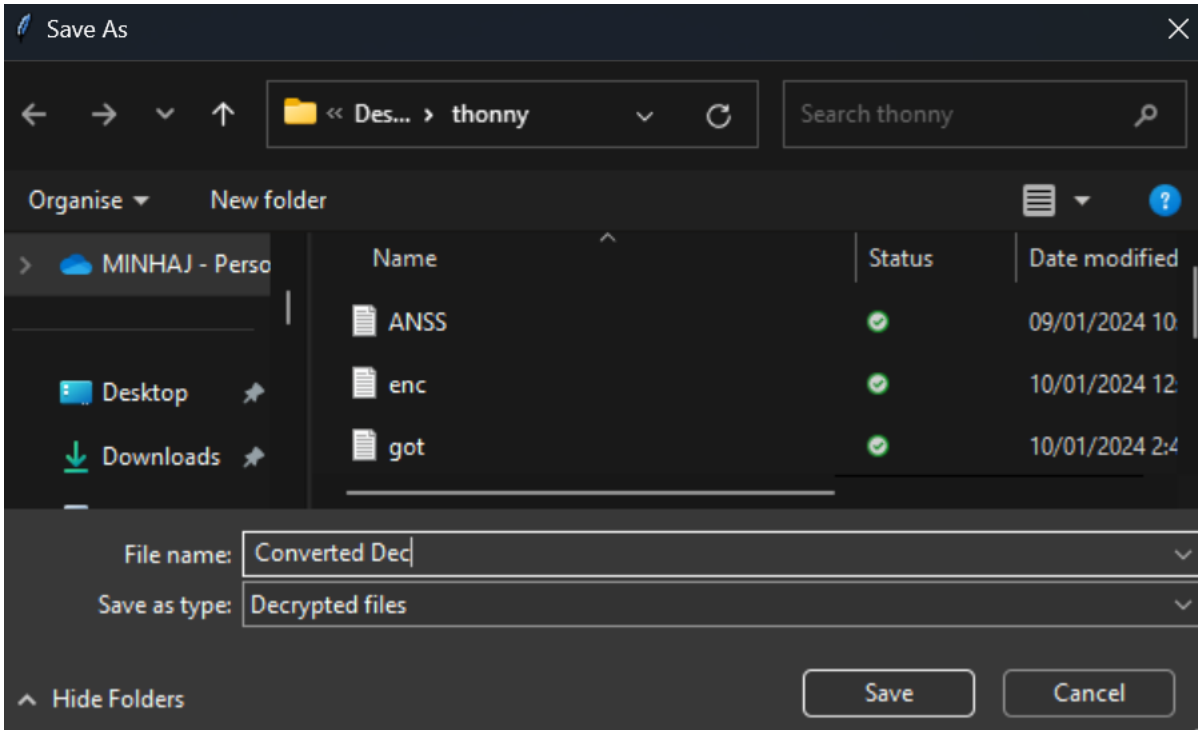


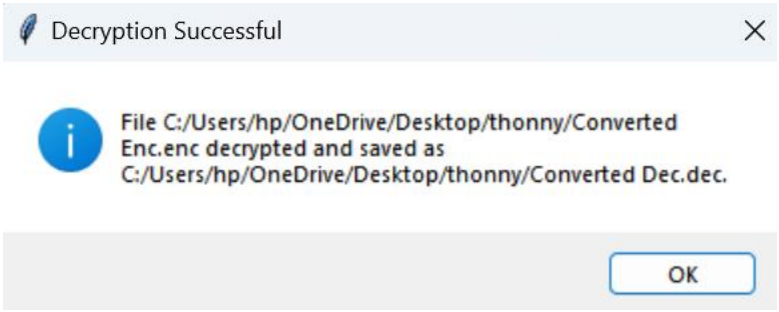

Save As
Des... › thonny
Search thonny
Organise
New folder
MINHAJ - Perso
Desktop
Downloads
Name
Status
Date modified
qq
10/01/2024 12
try
10/01/2024 2:4
tryy
10/01/2024 11
File name:
Converted Enc
Save as type:
Encrypted files
Hide Folders
Save
Cancel

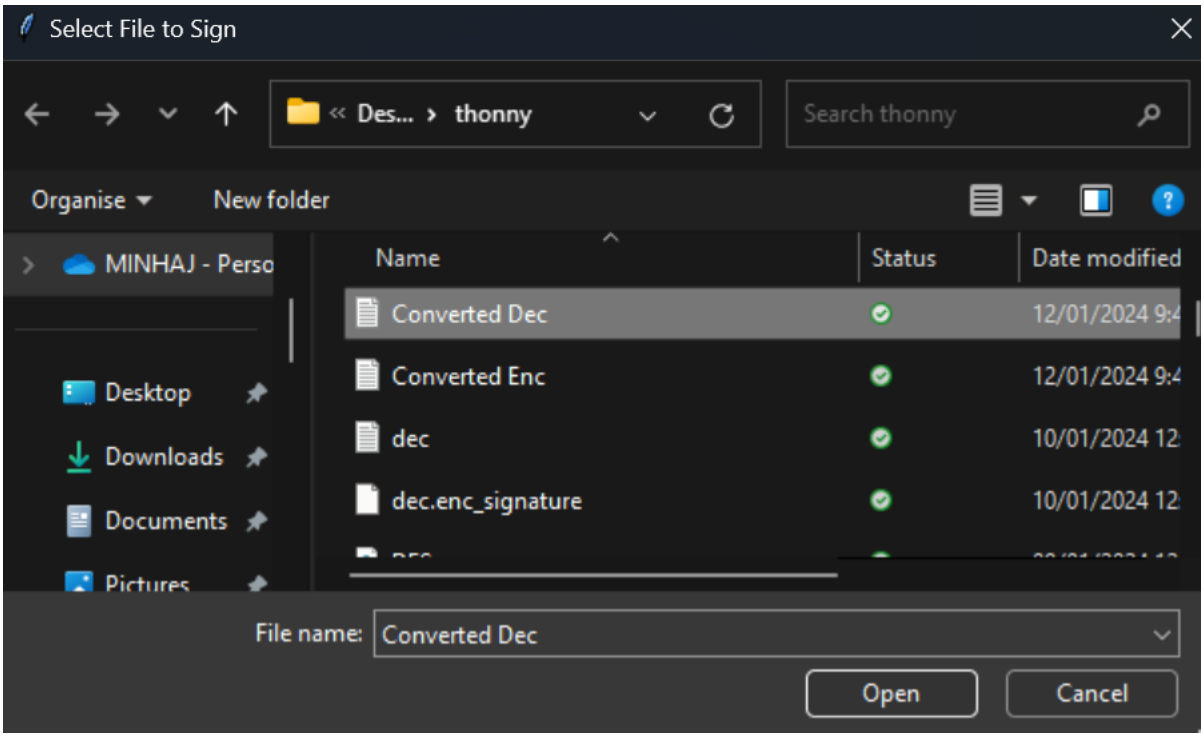

Encryption Successful
File C:/Users/hp/OneDrive/Desktop/thonny/plaintxt.txt
encrypted and saved as
C:/Users/hp/OneDrive/Desktop/thonny/Converted Enc.enc.
OK

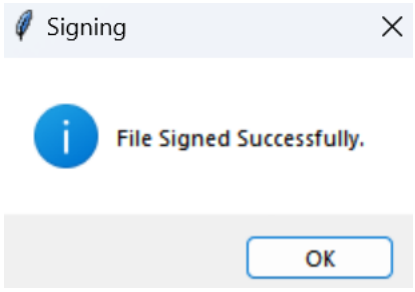

Select File to Decrypt
Des... › thonny
Search thonny
Organise
New folder
MINHAJ - Perso
Desktop
Downloads
Documents
Pictures
Name
Status
Date modified
ANS
09/01/2024 10
cifer
10/01/2024 10
Converted Enc
12/01/2024 9:4
dec
10/01/2024 12
File name:
Converted Enc
Encrypted files
Open
Cancel

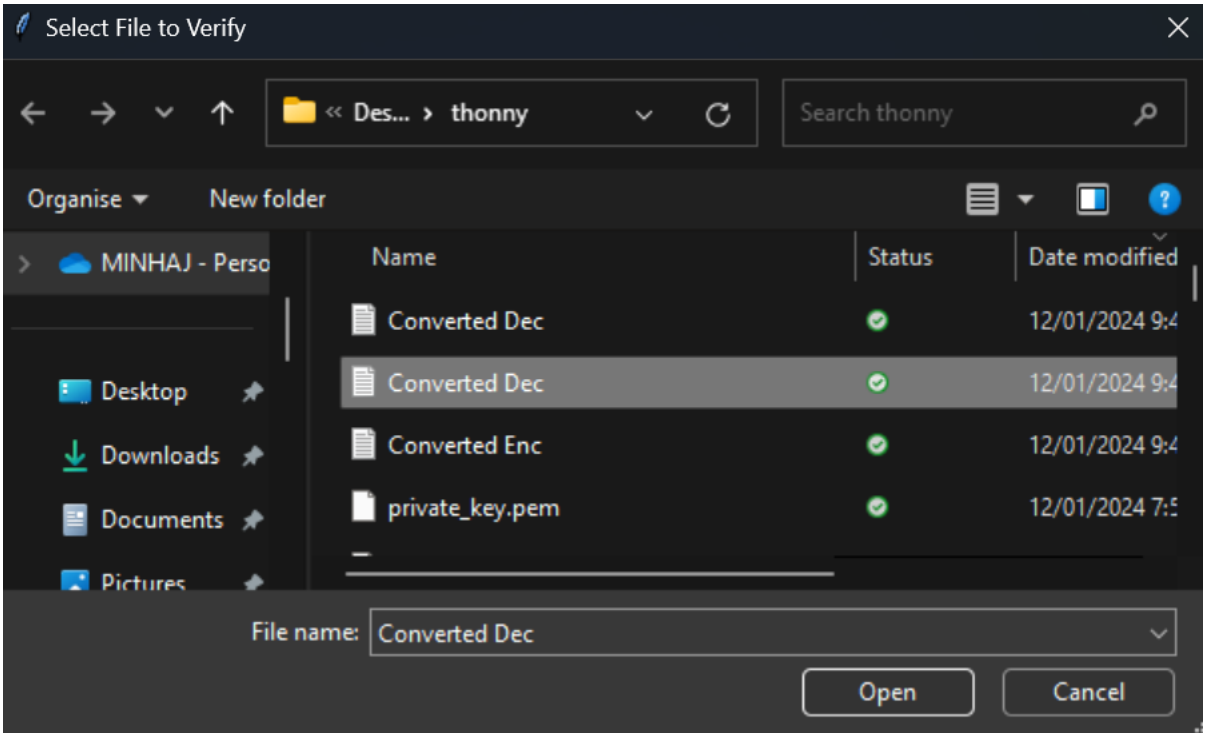

Select Private Key
Des... › thonny
Search thonny
Organise
New folder
MINHAJ - Perso
Desktop
Downloads
Documents
Pictures
Name
Status
Date modified
private_key.pem
12/01/2024 7:57
public_key.pem
12/01/2024 7:57
File name:
private_key.pem
PEM files
Open
Cancel

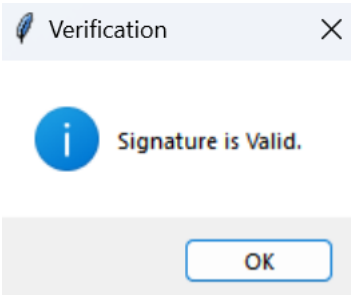

Save As
Des... › thonny
Search thonny
Organise
New folder
MINHAJ - Perso
Desktop
Downloads
Name
Status
Date modified
ANSS
09/01/2024 10
enc
10/01/2024 12
got
10/01/2024 2:4
File name:
Converted Dec
Save as type:
Decrypted files
Hide Folders
Save
Cancel


Decryption Successful
File C:/Users/hp/OneDrive/Desktop/thonny/Converted Enc.enc decrypted and saved as C:/Users/hp/OneDrive/Desktop/thonny/Converted Dec.dec.
OK


Select File to Sign
Des... › thonny
Search thonny
Organise
New folder
MINHAJ - Perso
Desktop
Downloads
Documents
Pictures
Name
Status
Date modified
Converted Dec
Converted Enc
dec
dec.enc_signature
File name:
Converted Dec
Open
Cancel


Signing
File Signed Successfully.
OK


Select File to Verify
Des... › thonny
Search thonny
Organise
New folder
MINHAJ - Perso
Desktop
Downloads
Documents
Pictures
Name
Status
Date modified
Converted Dec
Converted Dec
Converted Enc
private_key.pem
File name:
Converted Dec
Open
Cancel



CONCLUSION

BIBLIOGRAPHY

[1] Kalpana P., Singaraju S.: Data Security in Cloud Computing using RSA Algorithm, International Journal of Research in Computer and Communication Technology, 1(4) (2012) [Accessed 20 Dec. 2023].

[2] Burnett, S. and Paine, S. (2001). RSA Security's Official Guide to Cryptography. McGraw Hill Professional [Accessed 20 Dec. 2023].

[3] Obaid, T.S. (2020). Study A Public Key in RSA Algorithm. European Journal of Engineering Research and Science, 5(4), pp.395–398. doi:10.24018/ejers.2020.5.4.1843 [Accessed 20 Dec. 2023].

[4] GeeksForGeeks (2018). RSA Algorithm in Cryptography - GeeksforGeeks. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/rsa-algorithm-cryptography/ [Accessed 20 Dec. 2023].