Project Report

RSA-Based File Encryption and Digital Signature System

Name : Cho Davon

Group :01

IDTB100231

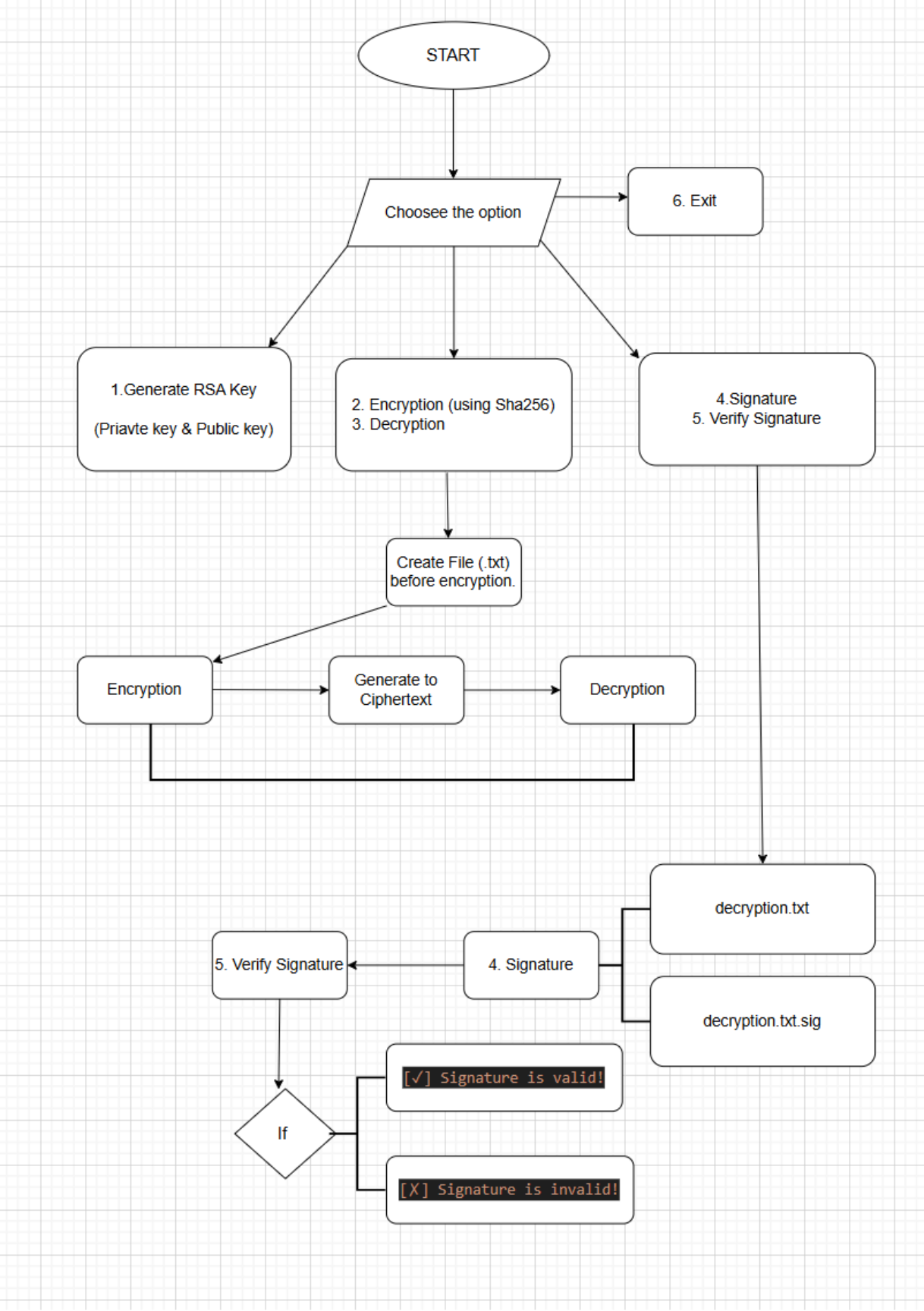
I. Introduction

In today's world, our important files (photos, documents, IDs, messages) can be easily stolen or changed. This project, **"RSA-Based File Encryption and Digital Signature System"**, is a simple Python program that uses **RSA** the same strong encryption used by banks and WhatsApp to keep files safe in two ways:

- **Encryption**: Locks any file so only the person with the private key can open it.
- **Digital Signature**: Adds a secure seal to prove the file is really from you and hasn't been altered.

The goal I want to Build an easy tool that gives real security to everyone and I want to learn real RSA hands-on, use a trusted professional library, and make powerful protection simple enough for non-tech people (friends, family, teachers) to use safely.

II. Design & Architecture


START
Choosee the option
6. Exit
1.Generate RSA Key
(Priavte key & Public key)
2. Encryption (using Sha256)
3. Decryption
4.Signature
5. Verify Signature
Create File (.txt)
before encryption.
Encryption
Generate to
Ciphertext
Decryption
decryption.txt
decryption.txt.sig
4. Signature
5. Verify Signature
[✓] Signature is valid!
If
[X] Signature is invalid!

III. implementation Detail

1. Libraries Used

This project uses the **cryptography** Python library, which provides secure and well-tested cryptographic functions.

The library is used to:

- Generate RSA key pairs
- Encrypt and decrypt files
- Sign files and verify digital signatures

Standard Python file input/output is also used to read and write files.

---

2. RSA Key Generation

The function generate_keys() creates two cryptographic keys:

- **Private Key** → kept secret by the owner
- **Public Key** → shared with others

These keys are saved as:

- private.pem
- public.pem

Both keys are required for encryption, decryption, signing, and verification.

---

3. File Encryption

File encryption is performed using the **public key**.

Function: encrypt_file()

Steps:

1. Read the plaintext file
2. Encrypt the content using RSA
3. Save the encrypted output as ciphertext.txt

Only the corresponding private key can decrypt the encrypted file.

---

4. File Decryption

File decryption requires the **private key**.

Function: decrypt_file()

Steps:

1. Read the encrypted file (ciphertext.txt)
2. Decrypt the content using the private key
3. Save the original data as decrypted.txt

---

5. File Signing

Digital signatures are created using the **private key**.

Function: sign_file()

Steps:

1. Read the file
2. Generate a SHA-256 hash

3. Create a digital signature
4. Save the signature as a .sig file (example: message.txt.sig)

This proves the file was created by the legitimate sender.

---

6. Signature Verification

Signature verification is done using the **public key**.

Function: verify_file_signature()

Steps:

1. Read the original file
2. Read the signature file
3. Verify the signature using the public key

Results:

- **Valid** → File is authentic and unchanged
- **Invalid** → File was modified or signature is incorrect

---

7. Program Structure

main.py → Menu and program controller

keys.py → RSA key generation

encryptor.py → File encryption and decryption

signer.py → Digital signing and verification

---

8. Cryptography Methods Used

- **RSA** → Encryption, decryption, signing, and verification
- **SHA-256** → Hashing during digital signing
- **OAEP & PSS Padding** → Improves RSA security and protects against attacks

IV. Usage Guide

Option 1 → Generate RSA Keys

- Select 1
- Program creates:
    - private.pem
    - public.pem
- These two files must exist before doing encryption or signing.

Option 2 → Encrypt File

1. Create any plaintext file (example: message.txt)
2. Choose 2
3. Enter file name:
4. message.txt
5. Program outputs:
    - ciphertext.txt (encrypted file)

Option 3 → Decrypt File

1. Choose 3
2. Enter encrypted file name:
3. ciphertext.txt
4. Enter output file:
5. decrypted.txt
6. Program creates:
    - decrypted.txt (original message restored)

Option 4 → Sign File

1. Choose 4
2. Enter file to sign:
3. message.txt
4. Program creates:
    - message.txt.sig (digital signature)

Option 5 → Verify Signature

1. Choose 5
2. Enter file:
3. message.txt

4. Enter signature file:
5. message.txt.sig
6. Program displays:
    - [✓] Signature is valid!
      or
    - [X] Signature is invalid!

V. Conclusion and Future Work

This project demonstrates how RSA cryptography can be used to secure files through encryption and digital signatures. It provides a clear understanding of how public and private keys work together to ensure confidentiality and authenticity.

Possible future improvements include:

- Password protection for private keys
- Support for large files using hybrid encryption (AES + RSA)
- Graphical User Interface (GUI)
- Logging and auditing features
- Improved error handling

VI. References

- https://www.geeksforgeeks.org/computer-networks/cryptography-and-its-types/
- https://stackoverflow.com/questions/20531474/decrypting-with-rsa-private-key

- https://www.geeksforgeeks.org/computer-networks/rsa-algorithm-cryptography/
- https://www.askpython.com/python/examples/rsa-algorithm-in-python