Security Vulnerabilities

Student number: 19225928

MSc Computer Science for Cyber Security

Module: COMP7030

File Encryption

Student number: 19225928

MSc Computer Science for Cyber Security

Module: COMP7024

**1 A description of your File Encryption**

The file encryption first generates an RSA key pair using the function generate\_key\_pair(), which creates a new RSA key with a length of KEY\_LENGTH bits and a public exponent of PUB\_EXP. The key pair consists of a private key and a corresponding public key.

The encrypt\_file() function then takes the name of an input file, an output file, and the public key as input parameters. It opens the input file in binary read mode and the output file in binary write mode. It then reads the input file in blocks of size BLOCK\_SIZE, encrypts each block using the public key and writes the resulting encrypted blocks to the output file.

Similarly, the decrypt\_file() function takes the name of an encrypted input file, an output file, and the private key as input parameters. It opens the encrypted input file in binary read mode and the output file in binary write mode. It then reads the encrypted input file in blocks of size KEY\_LENGTH/8, decrypts each block using the private key, and writes the resulting decrypted blocks to the output file.

In the main() function, the program generates an RSA key pair using the generate\_key\_pair() function. Then, it encrypts the input file "input.txt" using the encrypt\_file() function and writes the encrypted data to the output file "encrypted.bin". Finally, it decrypts the encrypted data from "encrypted.bin" using the decrypt\_file() function and writes the decrypted data to the output file "output.txt".

This implementation uses RSA with PKCS1 padding for encryption and decryption, and the block size used for encryption is determined by the size of the RSA key used (i.e., BLOCK\_SIZE is set to KEY\_LENGTH/8).

**2 A list of functional and non-functional requirements and security features of a File Encryption**

| **#** | **Functional Requirements** |
| --- | --- |
| 1 | Encryption and decryption of files using RSA encryption algorithm |
| 2 | Ability to select specific files or folders for encryption |
| 3 | Ability to set a password or key for encryption and decryption |
| 4 | Compatibility with different file types and formats |
| 5 | Ability to compress files before encryption |
| 6 | Ability to batch process multiple files at once |
| 7 | Ability to view encrypted files without decryption |
| 8 | Ability to change encryption settings or password/key |
| 9 | Ability to securely delete original files after encryption |

| **#** | **Non-Functional Requirements** |
| --- | --- |
| 1 | Performance: fast and efficient encryption and decryption |
| 2 | Usability: user-friendly interface and clear instructions |
| 3 | Compatibility: ability to work on multiple platforms and devices |
| 4 | Reliability: secure and error-free encryption and decryption |
| 5 | Scalability: ability to handle large amounts of data |
| 6 | Availability: continuous availability of encryption service |
| 7 | Maintainability: easy maintenance and updates |
| 8 | Portability: ability to move encrypted files across different systems and devices |
| 9 | Interoperability: ability to work with other encryption software or protocols |
| 10 | Adaptability: ability to adapt to changing encryption standards and regulations |

| **#** | **Security Features** |
| --- | --- |
| 1 | RSA encryption algorithm with strong key management |
| 2 | Protection against brute-force attacks |
| 3 | Malware and virus protection |
| 4 | Protection against unauthorized access and tampering |
| 5 | Secure deletion of original files after encryption |
| 6 | Secure transmission of encrypted files |
| 7 | Auditing and logging of encryption activities |
| 8 | Two-factor authentication for access to encrypted files |
| 9 | Compliance with industry and government regulations for data privacy and security |

**3 Design of your software/patch that includes communications with the OS**

Diagram

Description automatically generated

***Figure 1: Call Graph***

As seen in figure 1 the program communicates with the Minux operating system, it uses standard C library functions such as **fopen**, **fread**, **fwrite**, and **fclose** to read and write files. These functions are part of the C standard library, which is available on Minux and other operating systems.

Diagram

Description automatically generated

***Figure 2: Flow Chart***

Figure 2 shows that the file encryption program starts by generating an RSA key pair. It then proceeds to encrypt an input file using the public key and write the encrypted data to an output file. The program then reads the encrypted data from the output file, decrypts it using the private key, and writes the decrypted data to another output file. Finally, it frees the RSA key pair and ends.