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.

**4 Implementation of your File Encryption including annotated C code**

<https://github.com/JordanIrving1/MinixProject>

**5 Testing plan for validating your software**

| **Test Step** | **Test Action** | **Expected Result** | **Actual Result** |
| --- | --- | --- | --- |
| 1 | Create a plaintext file **test.txt** with some sample text | **test.txt** file created with sample text |  |
| 2 | Generate an RSA key pair using OpenSSL command | RSA key pair generated successfully |  |
| 3 | Extract the public key from the private key using OpenSSL | Public key extracted successfully |  |
| 4 | Encrypt the **test.txt** file using the public key | **test.enc** file created with encrypted data |  |
| 5 | Decrypt the **test.enc** file using the private key | **test\_decrypted.txt** file created with decrypted data |  |
| 6 | Compare the content of **test.txt** and **test\_decrypted.txt** | Content of both files should be the same |  |

**6 Description of integrating/adding the implemented component/patch to OS**

| **Steps** | **Integrating Process** | **Expected Result** |
| --- | --- | --- |
| 1 | pkgin install openssl | Install the required dependencies: OpenSSL library and development tools for C programming language. |
| 2 | pkgin install build-essential | Install development tools for C programming language, run the following command: |
| 3 | gcc -o file\_encryption file\_encryption.c -lcrypto | Compile the program |
| 4 | ./file\_encryption | Run program |

This should execute the program, which will generate an RSA key pair, encrypt the input file, and then decrypt the encrypted file. Make sure that the input file "input.txt" is in the same directory as the compiled executable file "file encryption". Also, make sure that the output files "encrypted.bin" and "output.txt" are not already present in the directory, as the program will overwrite any existing files with the same name.

**7 Integration testing plan for integrating your component/patch into the system. This includes designing and running an experiment that evaluates the performance of the implemented functionalities**

1. Objective: To ensure that the implemented functionalities of the code perform as expected and meet the system requirements when running on Minix.
2. Test Environment: Minix operating system.
3. Test Data: A sample input file "input.txt".
4. Test Setup: a. Compile and run the code on Minix.

b. Generate a key pair using the "generate\_key\_pair()" function.

c. Use the generated key pair to encrypt the input file using the "encrypt\_file()" function.

d. Use the same key pair to decrypt the encrypted file using the "decrypt\_file()" function.

e. Compare the decrypted output file with the original input file to ensure that they match.

1. Test Procedure:

a. Set up the test environment and test data.

b. Compile and run the code on Minix.

c. Generate a key pair using the "generate\_key\_pair()" function.

d. Use the generated key pair to encrypt the input file using the "encrypt\_file()" function.

e. Use the same key pair to decrypt the encrypted file using the "decrypt\_file()" function. f. Compare the decrypted output file with the original input file to ensure that they match.

g. Record the time taken to encrypt and decrypt the file.

h. Repeat steps c to g with different input files of varying sizes to test the performance of the code.

1. Expected Test Results:

a. The encrypted and decrypted output files match the original input file.

b. The time taken to encrypt and decrypt the files is within acceptable limits.

c. The code meets the system requirements and performs as expected.

1. Test Conclusion: The integration testing was unsuccessful, and the implemented functionalities of the code did not meet the system requirements and failed to run on Minix.

**8 Reporting the possible limitations, failures, and/or difficulties you experience in your work**

Problems that came up during the coursework include:

| **Limitations, Failures, and Difficulties in Implementing File Encryption on Minix** |
| --- |
| Difficulties mounting file encryption onto Minix |
| Limited storage space on personal laptop made it difficult at attempting to encrypt large files or directories. |
| Compatibility issues with certain encryption algorithms and software libraries. |
| Failed to properly mount code onto Minix |
| Failed at compiling code on Minix |

**9 A conclusion section that includes recommendations for extending the conducted work and personal reflection**

The code makes use of the OpenSSL library in C to perform RSA encryption and decryption. The code generates a 2048-bit RSA key pair, uses the public key to encrypt an input file in blocks of size 245 bytes, and saves the encrypted data to a binary file. Then, it uses the private key to decrypt the encrypted data back to the original input file.

Some recommendations for extending this work are to improve the security of the encryption by using a stronger key or a more secure padding scheme, such as OAEP. Additionally, error handling can be improved to provide more informative error messages and handle unexpected errors more gracefully.

Another recommendation for extending this project is to test it on other operating systems. By doing so, it can help identify any platform-specific issues that may arise and ensure that the code works as intended across different platforms. Moreover, it provides an opportunity to explore the various security features that Minix alone might not offer.

Working with encryption algorithms and attempting to mount this code to the Minix operating system to test its functionality in a different environment allowed for a deeper understanding of operating systems and has been an insightful experience into cryptography and its applications in secure data transmission. It is crucial to keep data safe from prying eyes, and using strong encryption algorithms is an essential step in achieving that goal. Overall, this project has been a rewarding and educational experience.