Faculty of Science and Engineering

Implementation of OpenPGP-CFB Mode and Comparative Analysis with RSA and Standard CFB Encryption

Noureldin Mohamed

Omar El-sakka

May 2023

Abstract

This project focuses on the development and performance evaluation of an integrated PGP-CFB mode for encryption and decryption using AES with PKCS padding. The proposed scheme will be compared to the standard CFB mode and RSA encryption. The implementation phase will involve developing the PGP-CFB mode with AES and PKCS, ensuring secure data transmission. The performance evaluation will assess encryption and decryption time for different message sizes (1 KB, 5 KB, 10 KB, 100 KB) compared to the standard CFB mode. Additionally, the performance of the RSA encryption scheme will be compared to the PGP-CFB mode using 1 KB and 5 KB files with a 256-bit security level. The project aims to provide insights into the efficiency and superiority of the PGP-CFB mode for secure data transmission.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

Secure data transmission is crucial in today's digital age, and the selection of an appropriate encryption scheme plays a vital role in maintaining confidentiality and integrity. However, there is a need for an improved encryption mode that combines the security features of OpenPGP and CFB while utilizing the AES block cipher with PKCS7 padding. Additionally, there is a lack of comprehensive performance evaluations comparing this integrated OpenPGP-CFB mode with both the standard CFB mode and the widely used RSA encryption scheme. Consequently, there is a need to address these gaps by developing and implementing an OpenPGP-CFB mode from scratch and conducting a comparative analysis with RSA encryption and the standard CFB mode, considering factors such as encryption/decryption time and message size.

1.2 Research Motivation

Python is a popular programming language for network programming due to its ease of use and availability of numerous libraries. Therefore, an implementation of the Open PGP in Python can be valuable for researchers and practitioners trying to understand OpenPGP. OpenPGP is also used in encrypting emails so understanding the implementation is the first step to determine the security of the algorithm, and the capability of attacking it.

1.3 Research Aim

The objective of this research is to implement OpenPGP encryption using the Cipher Feedback (CFB) mode of the Advanced Encryption Standard (AES) and compare its performance with the standard CFB mode and RSA encryption. The research aims to develop a robust and efficient implementation of OpenPGP-CFB, considering encryption speed, decryption speed, and resource requirements. By conducting a comprehensive performance analysis, including factors such as encryption/decryption time and computational overhead.

1.4 Research Structure

The paper is organized as follows. Section II provides an overview of the OpenPGP, RSA, and the Standard CFB algorithm. Section III describes the design and implementation of the OpenPGP. Section IV presents the experimental results obtained from running the simulation of the OpenPGP, it also handles the comparison between the algorithms. Finally, Section V concludes the paper by summarizing the key findings and highlighting the significance of the project in the context of cryptography.

Chapter 2

Literature Review

2.1 RFC 4880

RFC 4880 provides the specification for the OpenPGP message format, a widely used standard for secure email communication, file encryption, and digital signatures. This document outlines the technical details, data structures, algorithms, and procedures involved in OpenPGP.

RFC 4880 serves as a crucial reference for developers, researchers, and users interested in understanding and implementing OpenPGP. It provides a standardized framework for secure communication and data protection, facilitating the development of compatible OpenPGP implementations and promoting secure and interoperable communication across various platforms and systems.

This research will be implementing the OpenPGP algorithm described in RFC 4880.

2.2 Overview of OpenPGP

OpenPGP (Pretty Good Privacy) is a widely used encryption and decryption algorithm that provides secure communication and data transmission. It combines various cryptographic techniques to ensure confidentiality, integrity, and authenticity of messages and files.

RFC 4880 handles both symmetric, and asymmetric encryptions.

Symmetric encryption is used for the actual encryption and decryption of the message or data. In OpenPGP, a symmetric session key is generated, often using a symmetric encryption

algorithm like **AES** (Advanced Encryption Standard). This session key is then used to encrypt the message or data. Symmetric encryption is more efficient for bulk data encryption due to its computational speed.

Asymmetric encryption, specifically the **RSA** algorithm, is used for key management and secure exchange of the symmetric session key. In OpenPGP, each user has a key pair consisting of a public key and a private key. The sender encrypts the symmetric session key using the recipient's public key. The recipient then uses their private key to decrypt the session key, allowing them to decrypt the message or data encrypted with the symmetric algorithm.

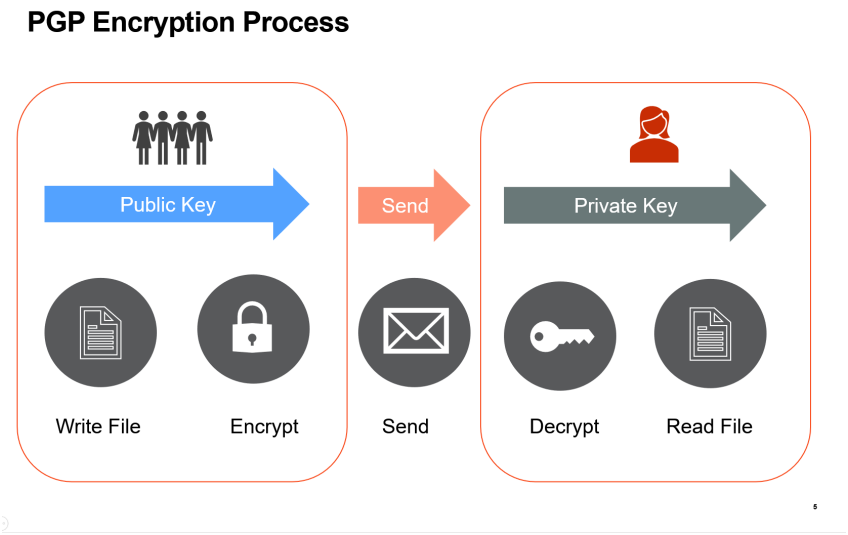


Figure 2.1: Open PGP.

This paper will focus on Symmetric encryption.

2.2.1 Applications

- **Secure Email Communication:**OpenPGP is widely used to secure email communication, ensuring that the contents of the email remain confidential and protected from unauthorized access. By encrypting the email content using the recipient's public key, the sender can ensure that only the intended recipient can read the message.
- **File Encryption:** OpenPGP can be used to encrypt files, providing an additional layer of security for sensitive data. By encrypting files using OpenPGP, users can protect their data even if it is stored or transmitted through insecure channels.
- **Digital Signatures:** OpenPGP supports the generation and verification of digital signa-

tures, enabling users to verify the authenticity and integrity of a message or file. Digital signatures provide assurance that the content has not been tampered with during transmission and that it originates from the claimed sender.

- **Key Exchange:** OpenPGP facilitates the secure exchange of public keys between users. By exchanging public keys through a trusted channel or a keyserver, users can establish secure communication channels without the risk of key interception or tampering.

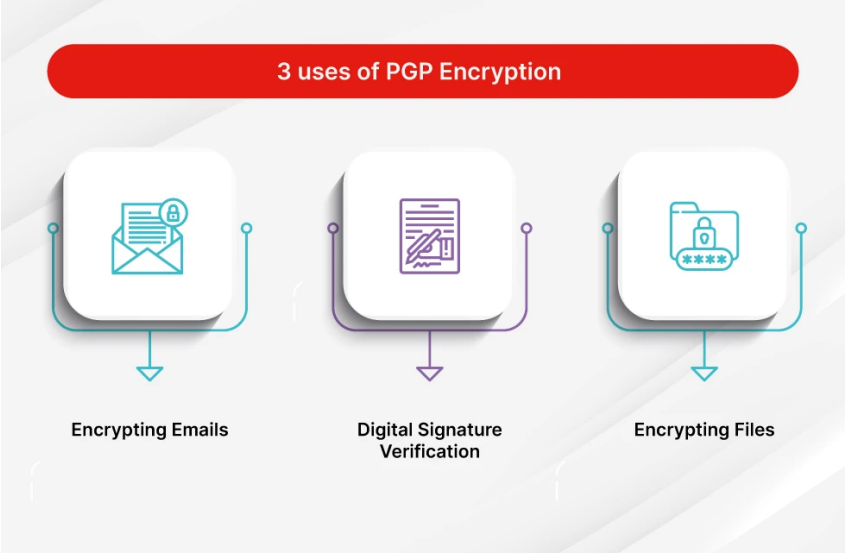


Figure 2.2: OpenPGP Applications.

2.3 Standard CFB

The Standard CFB (Cipher Feedback) mode is a method used in encryption algorithms to provide confi- dential and secure transmission of information. It works by encrypting the previous block of ciphertext and then combining it with the current plaintext block to generate the current block of ciphertext. This process is repeated for each block of plaintext.

The significance of Standard CFB lies in its ability to ensure that sensitive information cannot be accessed or tampered with by unauthorized parties. It is particularly useful for secure communication over insecure networks such as the Internet and for secure storage of data to prevent unauthorized access.

However, it's important to note that Standard CFB has some limitations in terms of performance and security. For instance, it may not be as secure as other modes like the Counter (CTR) mode, and it may not offer authenticated encryption. Therefore, it's crucial to evaluate the specific

requirements of a given application and choose an appropriate mode of operation accordingly.

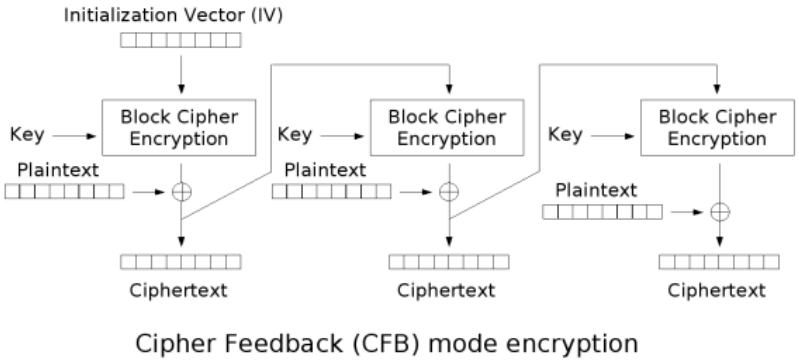


Figure 2.3: OpenPGP Applications.

2.4 OpenPGP CFB mode

OpenPGP supports various modes, one of which is the CFB mode.

The OpenPGP CFB (Cipher Feedback) algorithm is a mode of operation used within the OpenPGP encryption standard. It is designed to provide confidentiality and integrity of data during transmission.

In the OpenPGP CFB mode, the data to be encrypted is divided into fixed-size blocks, typically 8 bytes in length. It utilizes a symmetric encryption algorithm, such as AES (Advanced Encryption Standard), for encrypting these blocks.

The Cipher Feedback (CFB) mode operates in a feedback manner, where the ciphertext of the previous block is fed back into the encryption process to generate the key stream for encrypting the next block. This feedback mechanism allows for the encryption of individual blocks without the need for padding or pre-processing.

To encrypt data using OpenPGP CFB, an initialization vector (IV) is generated, serving as the starting point for the feedback mechanism. The IV is combined with the encryption key to produce the key stream, which is then XORed with the plaintext block to generate the ciphertext. This process is repeated for each block of data.

Decryption in the OpenPGP CFB mode follows a similar process. The IV and encryption key are used to generate the key stream, which is XORed with the ciphertext block to obtain the original plaintext.

The OpenPGP CFB algorithm provides several benefits, including the ability to encrypt and decrypt data in real-time without the need for padding, support for variable-length messages, and resistance to error propagation. It offers a good balance between security and efficiency for secure data transmission.

Overall, the OpenPGP CFB algorithm is an integral part of the OpenPGP encryption standard, providing a reliable and efficient mode of operation for ensuring the confidentiality and integrity of data during transmission.

The next section will be handling the implementation OpenPGP CFB mode.

2.5 RSA

RSA (Rivest-Shamir-Adleman) is a widely used asymmetric encryption algorithm named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman. It is based on the mathematical properties of large prime numbers and modular arithmetic.

The RSA algorithm involves a key pair consisting of a public key and a private key. The public key is used for encryption, while the private key is used for decryption. The security of RSA is based on the difficulty of factoring large composite numbers into their prime factors.

To generate an RSA key pair, two large prime numbers are selected. The product of these primes becomes the modulus for both the public and private keys. The public key also includes an exponent, typically a small prime number, and the private key includes a decryption exponent.

RSA encryption involves raising the plaintext message to the power of the public exponent and taking the modulus with the public modulus. This process transforms the plaintext into ciphertext. Decryption, on the other hand, involves raising the ciphertext to the power of the private exponent and taking the modulus with the private modulus, which recovers the original plaintext.

One of the key advantages of RSA is its ability to provide secure key exchange and digital signatures. The public key can be freely distributed, allowing anyone to encrypt messages intended for the owner of the corresponding private key. Additionally, RSA can be used to generate digital signatures by encrypting a cryptographic hash of a message with the private key, providing a way to verify the authenticity and integrity of the message.

While RSA is widely adopted and has a strong security foundation, its performance for encrypting and decrypting large amounts of data can be slower compared to symmetric encryption algorithms. Therefore, RSA is often used in combination with symmetric encryption, where the symmetric algorithm encrypts the actual data, and the RSA algorithm is used for securely exchanging the symmetric key.

Overall, RSA remains a fundamental and widely trusted algorithm for secure communication, digital signatures, and key exchange. Its strength lies in its asymmetric encryption approach, offering secure communication channels and enabling authentication and data integrity in various applications.

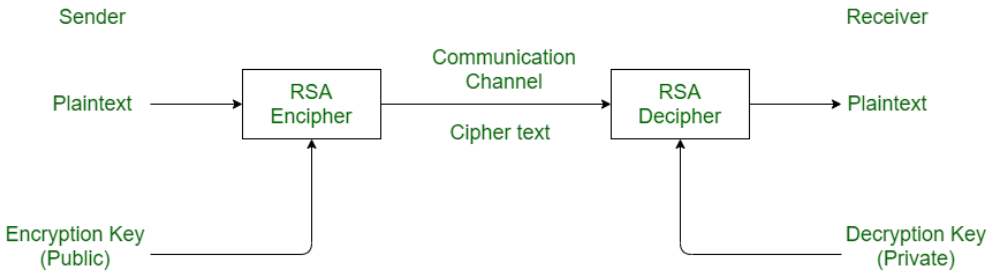


Figure 2.4: RSA.

Chapter 3

Methodology

3.1 Encryption

1. The feedback register (FR) is set to the IV, which is all zeros.
2. FR is encrypted to produce FRE (FR Encrypted). This is the encryption of an all-zero value.
3. FR is encrypted to produce FRE (FR Encrypted). This is the encryption of an all-zero value.
4. FR is loaded with C[1] through C[BS].
5. FR is encrypted to produce FRE, the encryption of the first BS octets of ciphertext.
6. The left two octets of FRE get xored with the next two octets of data that were prefixed to the plaintext. This produces C[BS+1] and C[BS+2], the next two octets of ciphertext.
7. (The resynchronization step) FR is loaded with C[3] through C[BS+2].
8. FR is encrypted to produce FRE.
9. FRE is xored with the first BS octets of the given plaintext, now that we have finished encrypting the BS+2 octets of prefixed data. This produces C[BS+3] through C[BS+(BS+2)], the next BS octets of ciphertext.
10. FR is loaded with C[BS+3] to C[BS + (BS+2)] (which is C11-C18 for an 8-octet block).
11. FR is encrypted to produce FRE.

12. FRE is xored with the next BS octets of plaintext, to produce the next BS octets of ciphertext. These are loaded into FR, and the process is repeated until the plaintext is used up.

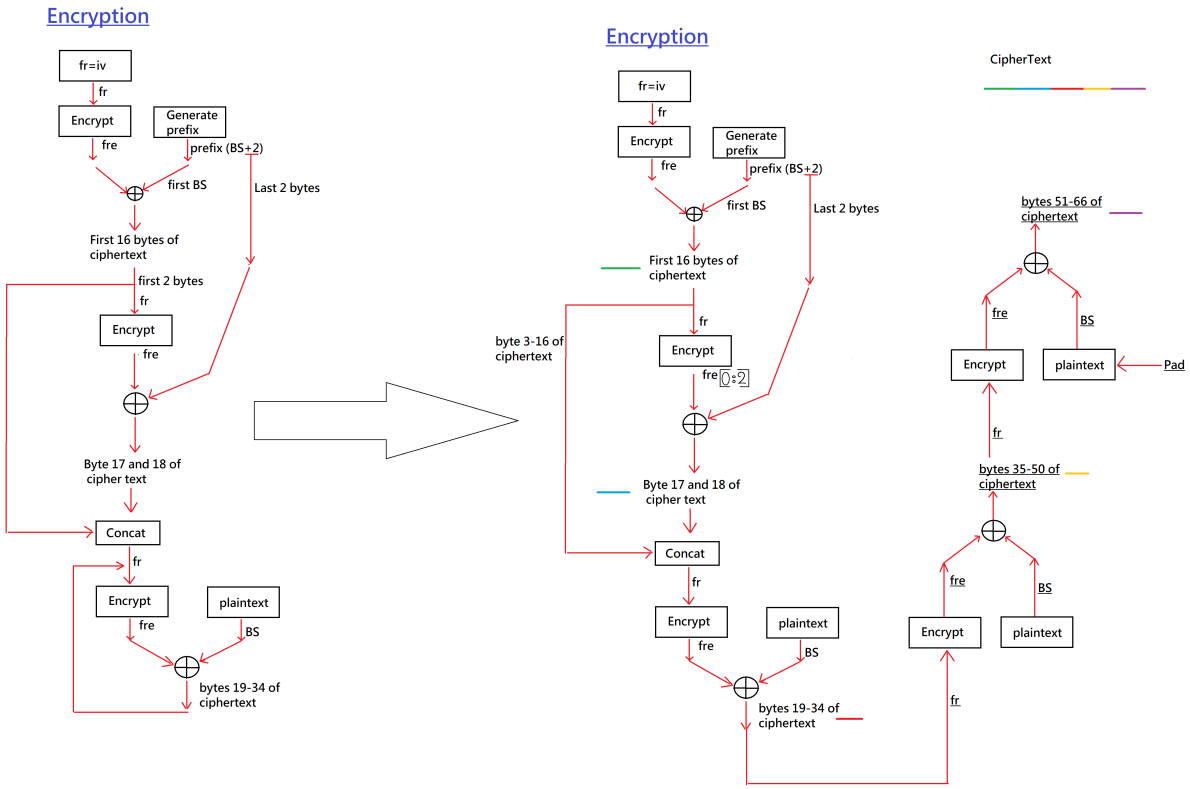


Figure 3.1: Encryption Diagram

This function takes a key and plaintext as input and performs encryption using the AES algorithm in CFB mode. The encryption process follows the specified block size, which is given by the length parameter (defaulting to 128 bits).

First, the code ensures that the block size is a multiple of 8, as required by AES. The block size is then calculated in bytes, and an empty byte array ciphertext is created to store the resulting ciphertext. The plaintext is converted to bytes and an initialization vector (IV) is generated.

Using the key, IV, and CFB mode, an AES cipher object is created. A random prefix of length BS+2 (block size plus 2) is generated. The feedback register (FR) is initialized with the IV, and the FR is encrypted to obtain FRE (FR Encrypted). The prefix is XORed with FRE to obtain the first 16 bytes of the ciphertext, which are added to the ciphertext byte array.

Next, the FR is updated with the obtained ciphertext block, and the encryption process continues. The plaintext is divided into blocks of the block size (BS), and each block is encrypted

by XORing it with the encrypted FR. The resulting ciphertext block is added to the ciphertext byte array, and the FR is updated with the ciphertext block.

If the plaintext length is less than the block size, PKCS7 padding is applied. The process repeats until all the plaintext is encrypted.

Finally, the resulting ciphertext is returned. This code provides a basic implementation of AES encryption in CFB mode, following the specified block size and using PKCS7 padding when necessary.

Equations

$$
\begin{aligned}
C[1:BS] &= \text{XOR}(FRE, \text{prefix}[0:BS]) \\
\text{FR} &= C[1:BS] \\
\text{FRE} &= \text{Encrypt}(\text{FR}) \\
C[BS+1:BS+2] &= \text{XOR}(\text{left two octets of FRE}, \text{prefix}[BS:BS+2]) \\
\text{FR} &= \text{Load FR with } C[3:BS+2] \\
\text{FRE} &= \text{Encrypt}(\text{FR}) \\
C[BS+3:BS+(BS+2)] &= \text{XOR}(FRE, \text{plaintext}[0:BS]) \\
\text{FR} &= C[BS+3:BS+(BS+2)] \\
\text{FRE} &= \text{Encrypt}(\text{FR}) \\
C[BS+(BS+2)+1:BS+(BS+2)+BS] &= \text{XOR}(FRE, \text{plaintext}[BS:2BS-1]) \\
&\vdots \\
C[i:i+BS-1] &= \text{XOR}(FRE, \text{plaintext}[(i-1)BS:iBS-1]) \\
&\vdots \\
\text{Ciphertext} &= C[1:]
\end{aligned}
$$

3.2 Decryption

This function takes a key and ciphertext as input and performs decryption using the AES algorithm in CFB mode. The decryption process follows the specified block size, given by the length parameter (defaulting to 128 bits).

Similar to the encryption function, the code ensures that the block size is a multiple of 8, calculates the block size in bytes, and generates an initialization vector (IV). An AES cipher object is created with the key, IV, and CFB mode.

The decryption process begins by recovering the prefix of the ciphertext. The FR is initialized with the IV, and the FR is encrypted to obtain FRE. The prefix is restored by XORing FRE with the first 16 bytes of the ciphertext. The prefix restoration process is divided into two parts: restoring the first two bytes (BS+1 and BS+2) and restoring the full prefix. The FR is updated with the restored prefix block.

Next, the ciphertext block for decryption is obtained from the input ciphertext. The FR is encrypted to obtain FRE, and the ciphertext block is decrypted by XORing it with FRE. The resulting plaintext block is stored.

The process continues by updating the FR with the decrypted ciphertext block, and the remaining ciphertext is decrypted block by block. Each ciphertext block is decrypted by XORing it with the encrypted FR, and the resulting plaintext blocks are concatenated.

After decrypting all the ciphertext blocks, the code checks if the plaintext has PKCS7 padding. If padding is detected, it is removed from the plaintext. Finally, the decrypted plaintext is returned as a string after decoding it from bytes.

This code provides a basic implementation of AES decryption in CFB mode, following the specified block size and handling PKCS7 padding when necessary.

Equations

$$
\begin{aligned}
\text{Prefix} &= \text{XOR}(FRE, \text{ciphertext}[0:16]) \\
\text{FR} &= \text{ciphertext}[0:16] \\
\text{FRE} &= \text{Encrypt}(\text{FR}) \\
\text{Prefix2} &= \text{XOR}(\text{first two octets of FRE}, \text{ciphertext}[16:18]) \\
\text{Prefix} &= \text{Prefix} + \text{Prefix2} \\
\text{FR} &= \text{ciphertext}[2:18] \\
\text{FRE} &= \text{Encrypt}(\text{FR}) \\
\text{Plaintext}[0:BS] &= \text{XOR}(FRE, \text{ciphertext}[18:34]) \\
\text{FR} &= \text{ciphertext}[18:34] \\
\text{FR} &= \text{Plaintext}[0:BS] \\
\text{FRE} &= \text{Encrypt}(\text{FR}) \\
\text{Plaintext}[BS:2BS-1] &= \text{XOR}(FRE, \text{ciphertext}[34:50]) \\
&\vdots \\
\text{Plaintext}[(i-1)BS:iBS-1] &= \text{XOR}(FRE, \text{ciphertext}[iBS:(i+1)BS-1]) \\
&\vdots \\
\text{Plaintext} &= \text{RemovePadding}(\text{Plaintext})
\end{aligned}
$$

3.3 GUI

A graphical user interface was designed to make the interaction neat.

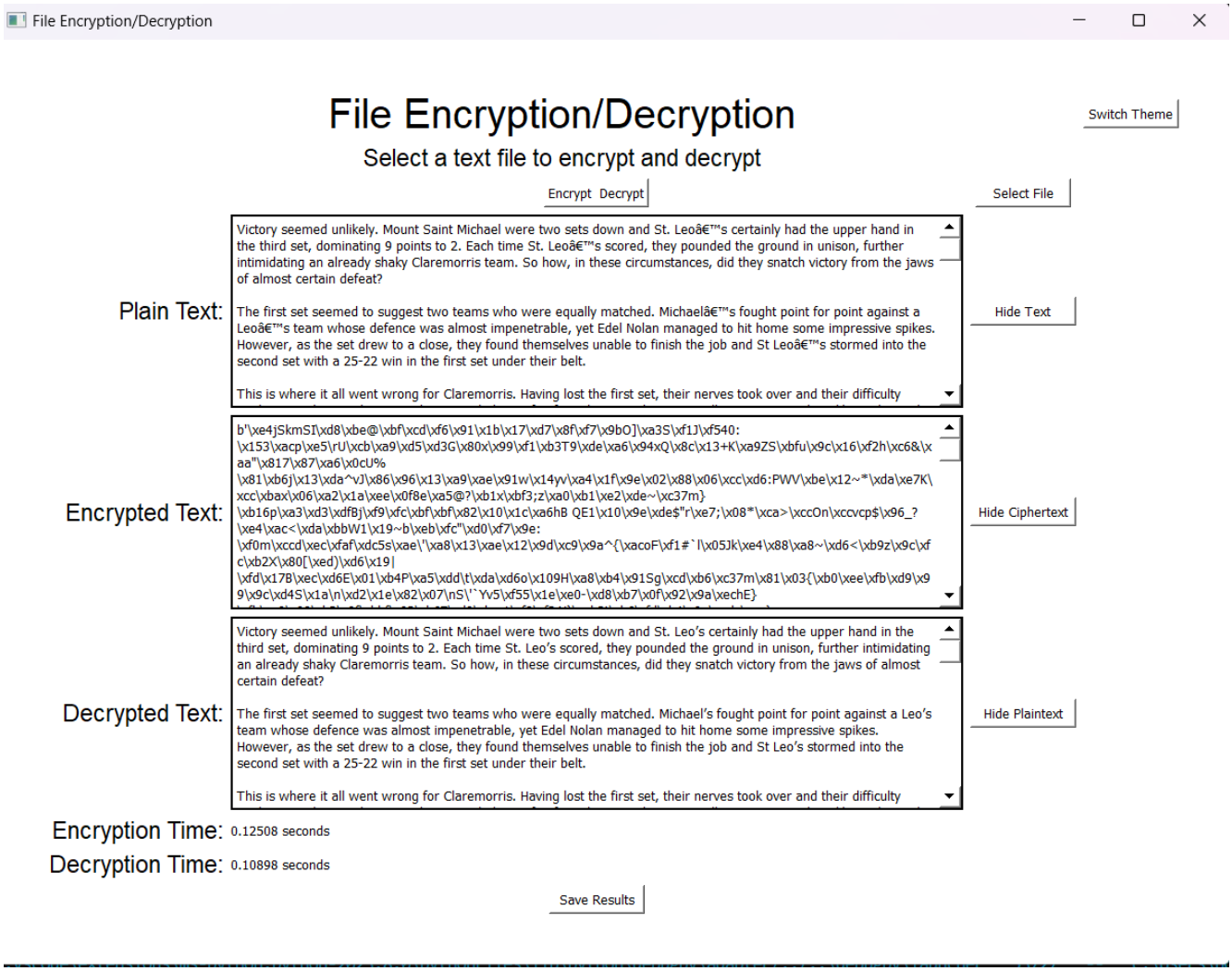


Figure 3.2: GUI

Chapter 4

Results and Evalution

4.1 PGP-CFB & Standard CFB & RSA

PGP-CFB, Standard CFB, and RSA are all cryptographic algorithms used for secure data transmission, but they differ in their approach and use cases. PGP-CFB and Standard CFB are symmetric encryption algorithms that work by encrypting data in blocks, while RSA is a public-key encryption algorithm that uses different keys for encryption and decryption. PGP-CFB is commonly used in email encryption, Standard CFB in disk encryption and messaging, and RSA in various applications for secure data transmission over the internet.

4.1.1 PGP-CFB & Standard CFB

Security Level 128

Maybe both curves have the same attitudes but they have two main different scales. Standard CFB is faster than 18 PGP-CFB.

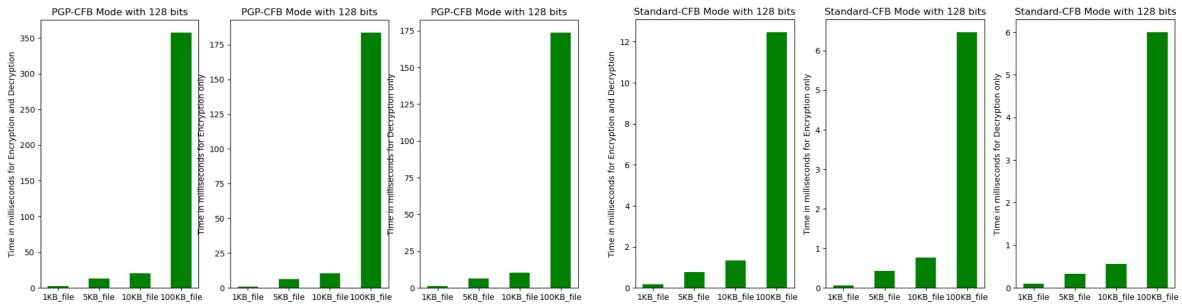
PGP-CFB (128 Security Level)

File Size (KB)	Encryption	Decryption	Encyption & Decryption
1	1.16556	1.23311	2.39867
5	6.26297	6.2631	12.52607
10	10.69428	9.26086	19.95514
100	200.42859	198.77144	399.20003

Standard CFB (128 Security Level)

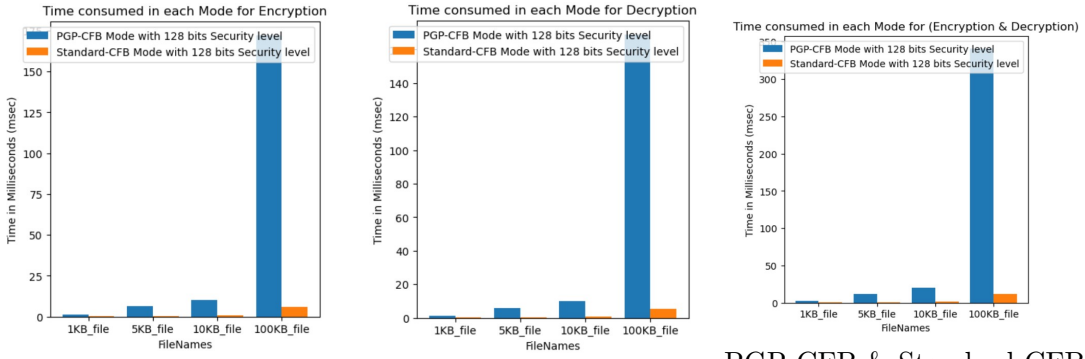
File Size (KB)	Encryption	Decryption	Encyption & Decryption
1	0.10046	0.09942	0.19988
5	0.39825	0.30135	0.6996
10	1.96642	0.83202	2.79844
100	11.2607	10.29353	21.55423

Figure 4.1: Security Level 128 time for Encryption and Decryption in milliseconds.



PGP-CFB Standard CFB

Figure 4.2: Comparison between PGP-CFB and Standard CFB.



PGP-CFB & Standard CFB Encryption PGP-CFB & Standard CFB Decryption PGP-CFB & Standard CFB (Encryption & Decryption)

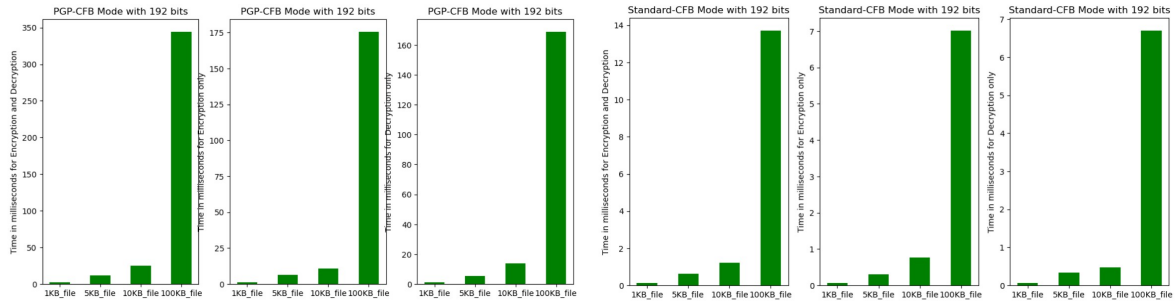
Figure 4.3: Comparison between PGP-CFB and Standard CFB Encryption and Decryption.

Security Level 192

Maybe both curves have the same attitudes but they have two main different scales. Standard CFB is faster than 27 PGP-CFB.

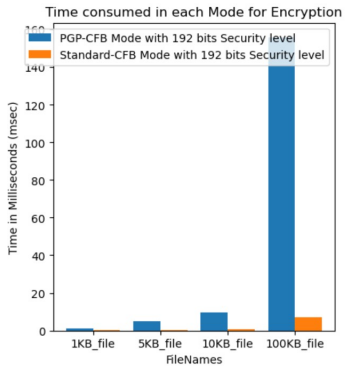
PGP-CFB (192 Security Level)			
File Size (KB)	Encryption	Decryption	Encyption & Decryption
1	1.09977	1.26551	2.36528
5	6.46308	6.32956	12.79264
10	12.25945	12.09324	24.35269
100	182.7104	179.40804	362.11844
Standard CFB (192 Security Level)			
File Size (KB)	Encryption	Decryption	Encyption & Decryption
1	0.0333	0.06655	0.09985
5	0.33322	0.39971	0.73293
10	0.63286	0.79961	1.43247
100	6.59628	6.41326	13.00954

Figure 4.4: Security Level 192 time for Encryption and Decryption in milliseconds.

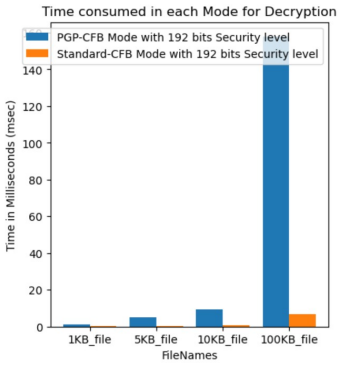


PGP-CFB Standard CFB

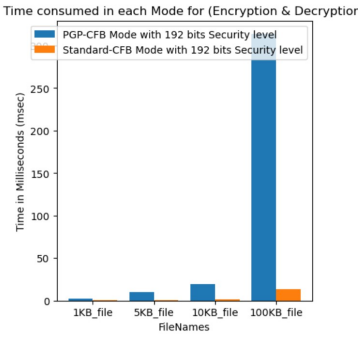
Figure 4.5: Comparison between PGP-CFB and Standard CFB.



PGP-CFB & Standard CFB Encryption



PGP-CFB & Standard CFB Decryption



PGP-CFB & Standard CFB (Encryption & Decryption)

Figure 4.6: Comparison between PGP-CFB and Standard CFB Encryption and Decryption.

Security Level 256

Maybe both curves have the same attitudes but they have two main different scales. Standard CFB is faster than 22 PGP-CFB.

PGP-CFB (256 Security Level)

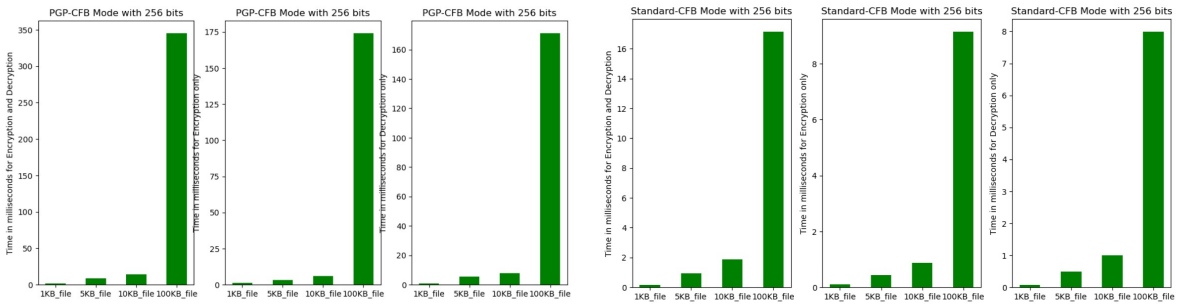
File Size (KB)	Encryption	Decryption	Encyption & Decryption
1	0.93266	0.533326	1.465986
5	3.5651	3.83062	7.39572
10	7.39639	6.59555	13.99194
100	151.98064	149.59603	301.57667

Standard CFB (256 Security Level)

File Size (KB)	Encryption	Decryption	Encyption & Decryption
1	0.16655	0.03332	0.19987
5	0.36639	0.43314	0.79953
10	0.73289	0.66627	1.39916
100	6.72916	6.56321	13.29237

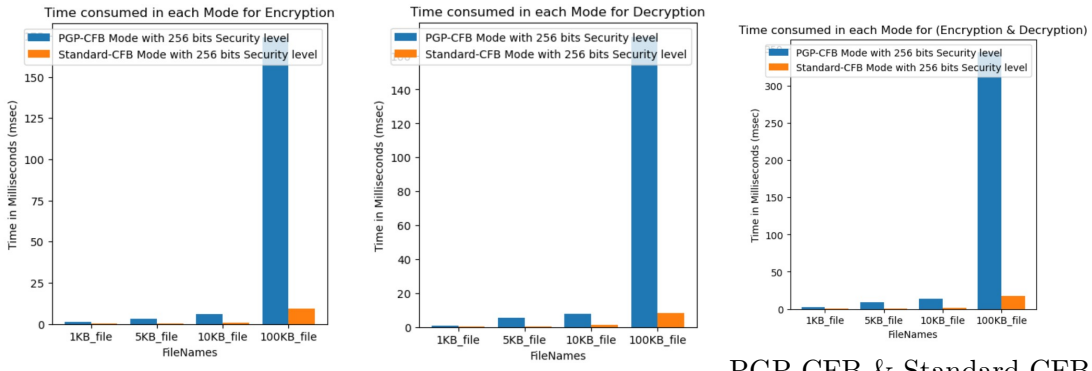
Figure 4.7: Security Level 256 time for Encryption and Decryption in milliseconds.

It is noticable that the time for both encryption and decryption is much larger in PGP-CFB than those time in Standard CFB.



PGP-CFB Standard CFB

Figure 4.8: Comparison between PGP-CFB and Standard CFB.



PGP-CFB & Standard CFB Encryption

PGP-CFB & Standard CFB Decryption

PGP-CFB & Standard CFB (Encryption & Decryption)

Figure 4.9: Comparison between PGP-CFB and Standard CFB Encryption and Decryption.

4.1.2 PGP-CFB & RSA

RSA (256 Security Level)

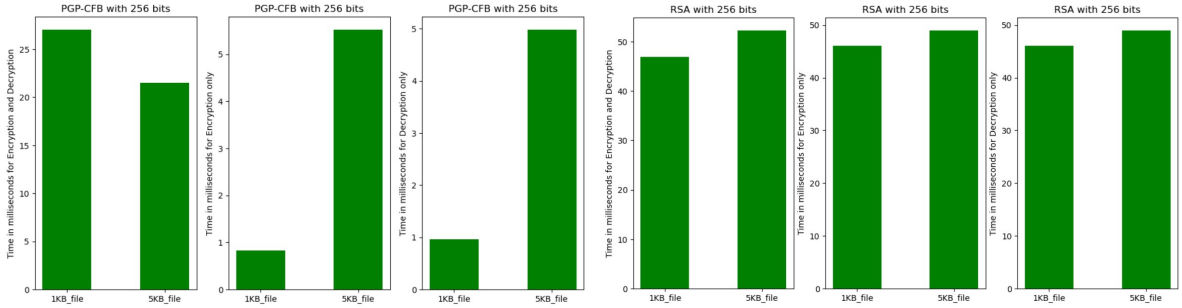
File Size (KB)	Encryption	Decryption	Encyption & Decryption
1	0.82068	46.10096	46.92165
5	3.28467	48.95232	52.237

PGP-CFB (256 Security Level)

File Size (KB)	Encryption	Decryption	Encyption & Decryption
1	0.64152	0.66619	1.30772
5	5.74082	2.71871	8.45953

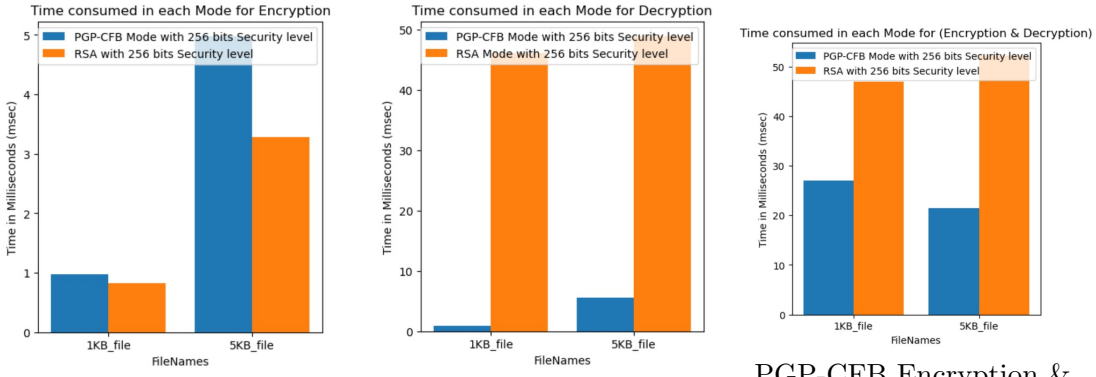
Figure 4.10: Security Level 256 time for Encryption and Decryption in milliseconds.

- Time for Encryption of both RSA and PGP-CFB is somehow close in same file size. but the main difference is on the decryption time.
- The decryption time of RSA is much larger than the encryption time of RSA.
- The decryption time of RSA much larger than decryption time of PGP-CFB.
- Time of Encryption and Decryption of RSA is not much affected by the size of file. On the other hand, Encryption and Decryption time for PGP-CFB is affected by the size of the file.



PGP-CFB RSA

Figure 4.11: Comparison between PGP-CFB and RSA.



PGP-CFB & RSA Encryption

PGP-CFB & RSA Decryption

PGP-CFB Encryption & Decryption

Figure 4.12: Comparison between PGP-CFB and RSA Encryption and Decryption.