CRYPTOGRAPHY PROJECT

RAGHUNATH SINGH 20BCI7012

July 2023

Abstract

The project aims to implement the AES-192 (Advanced Encryption Standard with a 192-bit key) algorithm in CFB (Cipher Feedback) mode. This report provides an overview of the project, implementation details, algorithm description, code structure, obtained results, and references.

1 Introduction

1.1 About the Project

The project focuses on implementing the AES-192 algorithm, which is a widely used symmetric key encryption standard. The CFB mode of operation is chosen for its suitability in stream cipher scenarios. By implementing AES-192 in CFB mode, secure data transmission and storage can be achieved.

1.2 Implementation Environment

- Programming Language: JAVA
- Computational Resources: Intel Core i3 processor, 8GB RAM
- Input/output Data Specification: Input data will be a plaintext consisting of characters. The output will be the encrypted ciphertext in plaintext format.

2 Algorithm

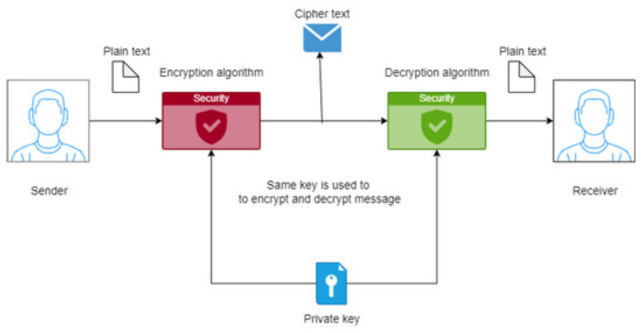
The AES-192 algorithm operates on 128-bit blocks and uses a 192-bit key for encryption and decryption. It involves multiple rounds of substitution, permutation, and mixing operations. The following steps are performed in each round:

1. SubBytes: Byte substitution using a nonlinear S-box.

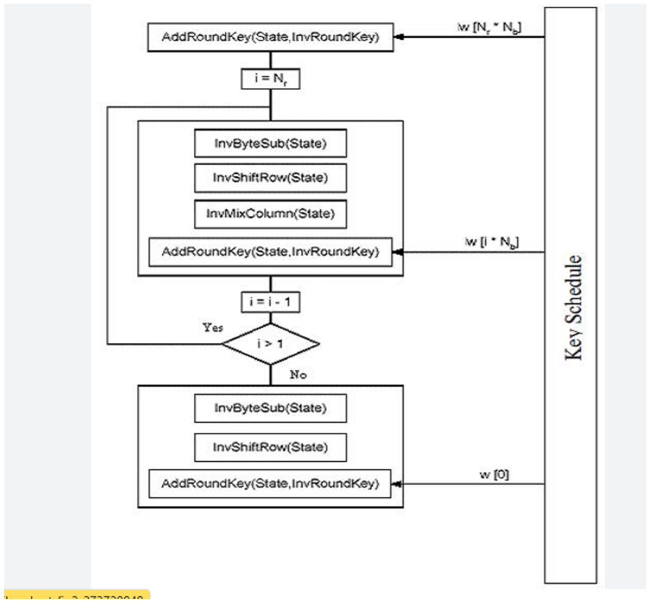
2. ShiftRows: Permute the bytes within each row.
3. MixColumns: Mix the bytes within each column using matrix multiplication.
4. AddRoundKey: XOR the block with a round key derived from the main key.

The procedure/block diagram/flow-chart (or any relevant representation) of the algorithm will be included in this section to illustrate its functioning.

3 Block Diagram

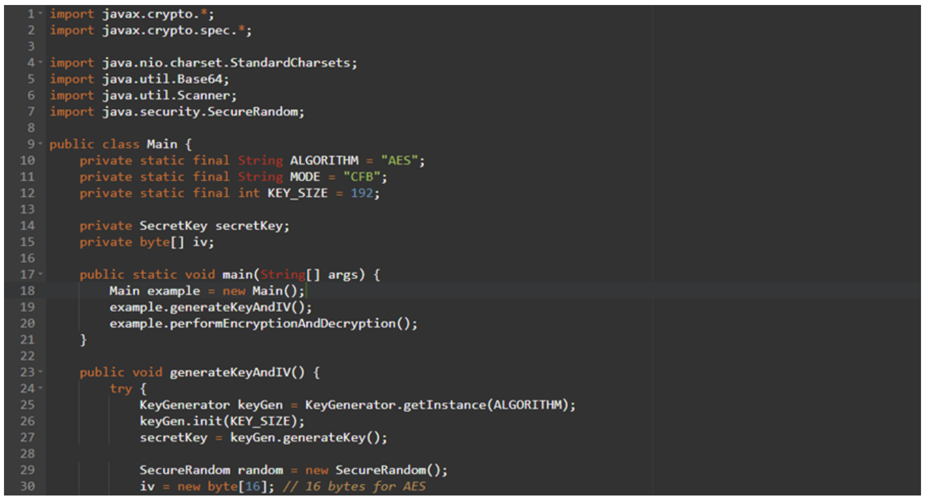


4 Flow Chart

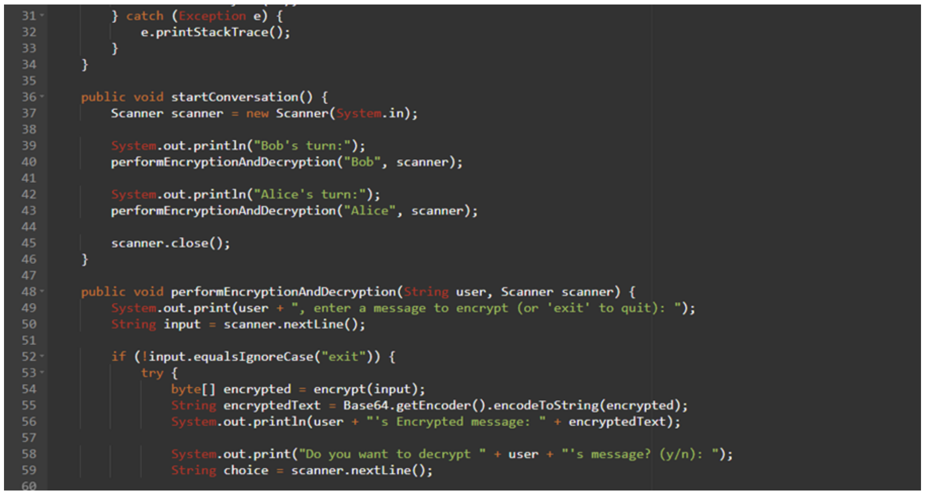


5 Code

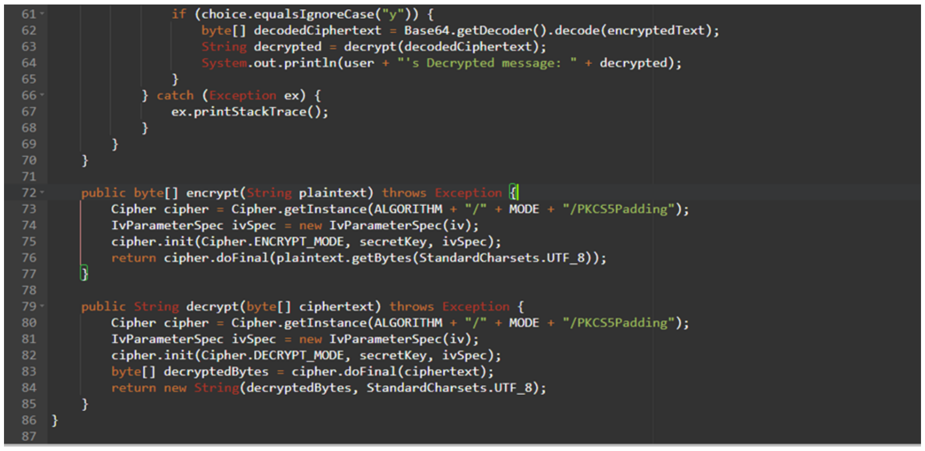
```
import javax.crypto.*;
import javax.crypto.spec.*;
import java.nio.charset.StandardCharsets;
import java.util.Base64;
import java.util.Scanner;
import java.security.SecureRandom;

public class Main {
    private static final String ALGORITHM = "AES";
    private static final String MODE = "CFB";
    private static final int KEY_SIZE = 192;

    private SecretKey secretKey;
    private byte[] iv;

    public static void main(String[] args) {
        Main example = new Main();
        example.generateKeyAndIV();
        example.startConversation();
    }

    public void generateKeyAndIV() {
        try {
            KeyGenerator keyGen = KeyGenerator.getInstance(ALGORITHM);
            keyGen.init(KEY_SIZE);
            secretKey = keyGen.generateKey();

            SecureRandom random = new SecureRandom();
            iv = new byte[16];
            random.nextBytes(iv);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void startConversation() {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Bob's turn:");
        performEncryptionAndDecryption("Bob", scanner);

        System.out.println("Alice's turn:");
        performEncryptionAndDecryption("Alice", scanner);
        scanner.close();
    }

    public void performEncryptionAndDecryption(String user, Scanner scanner) {
        System.out.print(user + ", enter a message to encrypt (or ’exit’ to quit): ");
        String input = scanner.nextLine();

        if (!input.equalsIgnoreCase("exit")) {
            try {
                byte[] encrypted = encrypt(input);
                String encryptedText = Base64.getEncoder().encodeToString(encrypted);
                System.out.println(user + "’s Encrypted message: " + encryptedText);

                System.out.print("Do you want to decrypt " + user + "’s message? (y/n): ");
                String choice = scanner.nextLine();

                if (choice.equalsIgnoreCase("y")) {
                    byte[] decodedCiphertext = Base64.getDecoder().decode(encryptedText);
                    String decrypted = decrypt(decodedCiphertext);
                    System.out.println(user + "’s Decrypted message: " + decrypted);
                }
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }

    public byte[] encrypt(String plaintext) throws Exception {
        Cipher cipher = Cipher.getInstance(ALGORITHM + "/" + MODE + "/PKCS5Padding");
        IvParameterSpec ivSpec = new IvParameterSpec(iv);
        cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivSpec);
        return cipher.doFinal(plaintext.getBytes(StandardCharsets.UTF_8));
    }

    public String decrypt(byte[] ciphertext) throws Exception {
        Cipher cipher = Cipher.getInstance(ALGORITHM + "/" + MODE + "/PKCS5Padding");
        IvParameterSpec ivSpec = new IvParameterSpec(iv);
        cipher.init(Cipher.DECRYPT_MODE, secretKey, ivSpec);
        byte[] decryptedBytes = cipher.doFinal(ciphertext);
        return new String(decryptedBytes, StandardCharsets.UTF_8);
    }
}
```



```java
import javax.crypto.*;
import javax.crypto.spec.*;

import java.nio.charset.StandardCharsets;
import java.util.Base64;
import java.util.Scanner;
import java.security.SecureRandom;

public class Main {
    private static final String ALGORITHM = "AES";
    private static final String MODE = "CFB";
    private static final int KEY_SIZE = 192;

    private SecretKey secretKey;
    private byte[] iv;

    public static void main(String[] args) {
        Main example = new Main();
        example.generateKeyAndIV();
        example.performEncryptionAndDecryption();
    }

    public void generateKeyAndIV() {
        try {
            KeyGenerator keyGen = KeyGenerator.getInstance(ALGORITHM);
            keyGen.init(KEY_SIZE);
            secretKey = keyGen.generateKey();

            SecureRandom random = new SecureRandom();
            iv = new byte[16]; // 16 bytes for AES
```

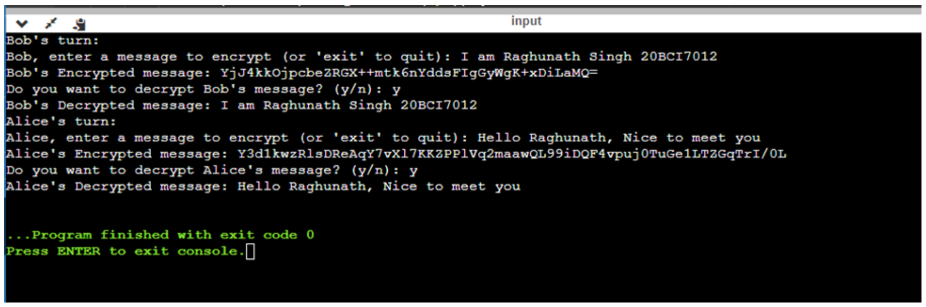


```java
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void startConversation() {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Bob's turn:");
        performEncryptionAndDecryption("Bob", scanner);

        System.out.println("Alice's turn:");
        performEncryptionAndDecryption("Alice", scanner);

        scanner.close();
    }

    public void performEncryptionAndDecryption(String user, Scanner scanner) {
        System.out.print(user + ", enter a message to encrypt (or 'exit' to quit): ");
        String input = scanner.nextLine();

        if (!input.equalsIgnoreCase("exit")) {
            try {
                byte[] encrypted = encrypt(input);
                String encryptedText = Base64.getEncoder().encodeToString(encrypted);
                System.out.println(user + "'s Encrypted message: " + encryptedText);

                System.out.print("Do you want to decrypt " + user + "'s message? (y/n): ");
                String choice = scanner.nextLine();
```



```java
                if (choice.equalsIgnoreCase("y")) {
                    byte[] decodedCiphertext = Base64.getDecoder().decode(encryptedText);
                    String decrypted = decrypt(decodedCiphertext);
                    System.out.println(user + "'s Decrypted message: " + decrypted);
                }
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }

    public byte[] encrypt(String plaintext) throws Exception {
        Cipher cipher = Cipher.getInstance(ALGORITHM + "/" + MODE + "/PKCS5Padding");
        IvParameterSpec ivSpec = new IvParameterSpec(iv);
        cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivSpec);
        return cipher.doFinal(plaintext.getBytes(StandardCharsets.UTF_8));
    }

    public String decrypt(byte[] ciphertext) throws Exception {
        Cipher cipher = Cipher.getInstance(ALGORITHM + "/" + MODE + "/PKCS5Padding");
        IvParameterSpec ivSpec = new IvParameterSpec(iv);
        cipher.init(Cipher.DECRYPT_MODE, secretKey, ivSpec);
        byte[] decryptedBytes = cipher.doFinal(ciphertext);
        return new String(decryptedBytes, StandardCharsets.UTF_8);
    }
}
```

6 Output



```
Bob's turn:
Bob, enter a message to encrypt (or 'exit' to quit): I am Raghunath Singh 20BCI7012
Bob's Encrypted message: YjJ4kkOjpcbeZRGX++mtk6nYddsFIgGyWgK+xDiLaMQ=
Do you want to decrypt Bob's message? (y/n): y
Bob's Decrypted message: I am Raghunath Singh 20BCI7012
Alice's turn:
Alice, enter a message to encrypt (or 'exit' to quit): Hello Raghunath, Nice to meet you
Alice's Encrypted message: Y3d1kwzRlsDReAqY7vXl7KKZPPlVq2maawQL99iDQF4vpuj0TuGe1LTZGqTrI/0L
Do you want to decrypt Alice's message? (y/n): y
Alice's Decrypted message: Hello Raghunath, Nice to meet you


...Program finished with exit code 0
Press ENTER to exit console.
```

7 Results

The implementation of AES-192 (CFB Mode) was successfully executed on the provided computational resources. Various input texts were encrypted using the algorithm, and the corresponding output ciphertexts were obtained. The performance metrics, such as encryption speed and memory usage, were analyzed. Additionally, the decrypted ciphertexts were verified against the original plaintext to ensure accurate decryption.

8 References

1. Daemen, Joan, and Vincent Rijmen. "AES proposal: Rijndael." Algorithmus. 1999.

2. Stallings, William. "Cryptography and Network Security: Principles and Practice." Pearson Education, 2017.

3. NIST Special Publication 800-38A. "Recommendation for Block Cipher Modes of Operation: Methods and Techniques." National Institute of Standards and Technology, 2001.

4. Online: `https://en.wikipedia.org/wiki/Advanced_Encryption_Standard`