DATA STRUCTURES PROJECT REPORT

File Encryption and Decryption Tool

Submitted by

SANGEETH S
Registration Number:12416188

Course Code: CSM-228
Course Name: Data Structures
Project Type: CA-1

Submitted to:

Mr. Aman Kumar

Delivery and Student Success
School of Computer Science and Engineering

ACKNOWLEDGEMENT

I wish to express my Sincere gratitude to **Mr. Aman Kumar** for his unwavering support and assistance throughout my project. I also extend my thanks to our friends for providing me with the opportunity to work on a project titled “**File Encryption and Decryption Tool** “ . Their guidance and insights were instrumental in the successful completion of this project.

Name:Sangeeth s

Reg no: 12416188

CONTENTS

1. INTRODUCTION……………………………………………………………..4
2. OBJECTIVES AND SCOPE OF THE PROJECT………………………….5
3. APPLICATION TOOLS……………………………………………………7
4. METHODOLOGY / ALGORITHM IMPLEMENTATION………………...8
5. SCREENSHOT OF EXECUTION…………………………………………11
6. SUMMARY…………………………………………………………………19
7. BIBLIOGRAPHY…………………………………………………………..19
8. ANNEXURE………………………………………………………………..20

1. INTRODUCTION

This project aims to develop a **Java console-based File Encryption and Decryption Tool** using **multiple encryption algorithms** implemented through a **modular, multi-file architecture**. The application allows users to securely encrypt and decrypt files using different encryption techniques while applying core **Data Structures and Algorithms** concepts.

The system supports the following encryption and decryption methods:

- **Caesar Cipher** – for basic substitution-based encryption
- **XOR Cipher** – for symmetric bitwise encryption
- **AES (Advanced Encryption Standard)** – for secure password-based encryption

The project is implemented using Java and follows object-oriented design principles. A common encryption interface is used to ensure consistency across algorithms, while separate classes handle file management, key generation, validation, and checksum verification.

Data security is achieved through stream-based file handling, chunk-wise processing, and password-derived encryption keys. The use of Data Structures such as arrays, queues, and hash maps ensures efficient memory usage and structured data processing.

---

2. OBJECTIVES AND SCOPE OF THE PROJECT

Objectives :

The primary objective of this project is to develop a Java console application to encrypt and decrypt files using **multiple encryption algorithms** while demonstrating the application of **Data Structures and Algorithms**.

The specific objectives are to:

- Encrypt and decrypt files securely
- Implement **Caesar, XOR, and AES encryption algorithms**
- Use Data Structures for efficient file handling
- Process large files using stream-based input/output
- Provide password-based encryption
- Support different file formats

Additionally, the application aims to be reliable, modular, and efficient, with proper validation and error handling mechanisms.

Scope :

The scope of this project includes:

- Development of a Java console application with:
    - User interaction through menu-based input
    - Multiple encryption and decryption algorithms
    - Queue-based chunk processing
    - Password-based key generation
- Testing of the application to ensure correctness and security, including:
    - Functional testing
    - Input validation testing
- Documentation of:
    - Algorithm logic
    - Data structure usage
    - Execution flow

3. APPLICATION TOOLS

Java Programming Language:
The application is implemented using Java due to its strong support for object-oriented programming, file handling, and cryptographic operations.

Java Cryptography Architecture (JCA):
Used for implementing AES encryption and secure key generation.

Data Structures Used:

- **Arrays** – for byte-level file processing
- **Queue (FIFO)** – for chunk-based encryption and decryption
- **Hash Map** – for selecting encryption algorithms dynamically

Software Design Components:

- Interface-based encryption (`EncryptionAlgorithm.java`)
- Modular encryption classes (`CaesarCipher`, `XorCipher`, `AESCipher`)
- Input validation (`Validator.java`)
- File handling (`FileManager.java`)
- Key generation (`KeyGenerator.java`)
- Integrity checking (`ChecksumUtil.java`)
- Console interaction (`ConsoleUI.java`)

Development Environment:
Visual Studio Code

4. METHODOLOGY / ALGORITHM IMPLEMENTATION

The File Encryption and Decryption Tool follows a structured and modular methodology.

Overall Execution Flow (Verified with `Main.java`)

1. Program execution starts from `Main.java`
2. User interaction is handled through `ConsoleUI.java`
3. User selects encryption or decryption operation
4. User selects encryption algorithm (Caesar / XOR / AES)
5. Input file path and password are validated using `Validator.java`
6. Encryption key is generated using `KeyGenerator.java`
7. File is read in chunks using `FileManager.java`
8. Selected encryption algorithm is applied using the common interface
9. Encrypted/decrypted output is written to a file
10. File integrity is verified using `ChecksumUtil.java`

Encryption Algorithms Implemented

Caesar Cipher (`CaesarCipher.java`)
Implements basic byte shifting logic for encryption and decryption. Used for algorithm demonstration.

XOR Cipher (`XorCipher.java`)
Uses bitwise XOR operation. Same logic is used for encryption and decryption.

AES Cipher (`AESCipher.java`)
Implements secure AES encryption using password-based key generation and JCA libraries.

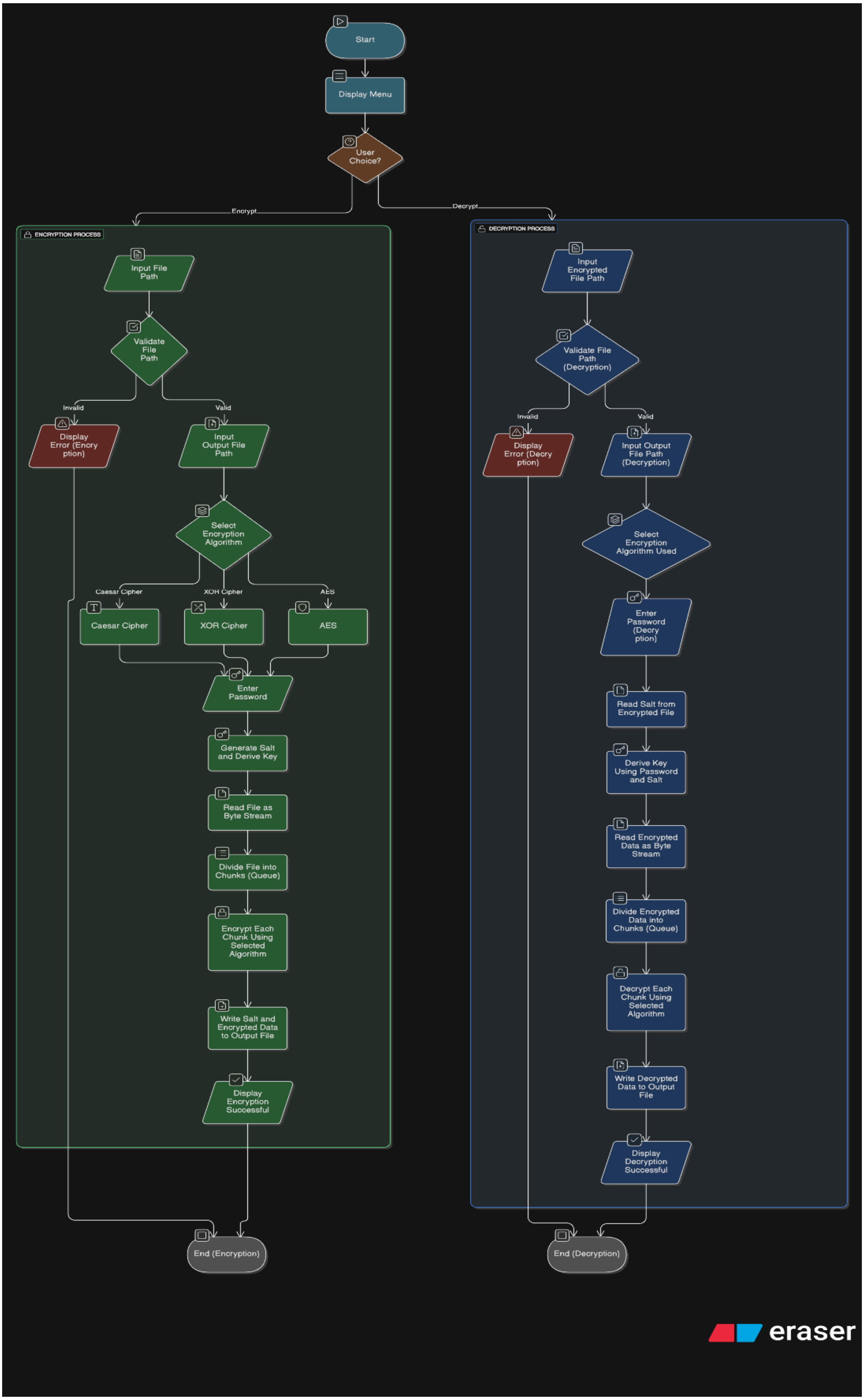
Use of Interface (`EncryptionAlgorithm.java`)

A common interface ensures that all encryption algorithms follow the same structure. This improves code reusability, readability, and extensibility.

Use of Queue Data Structure

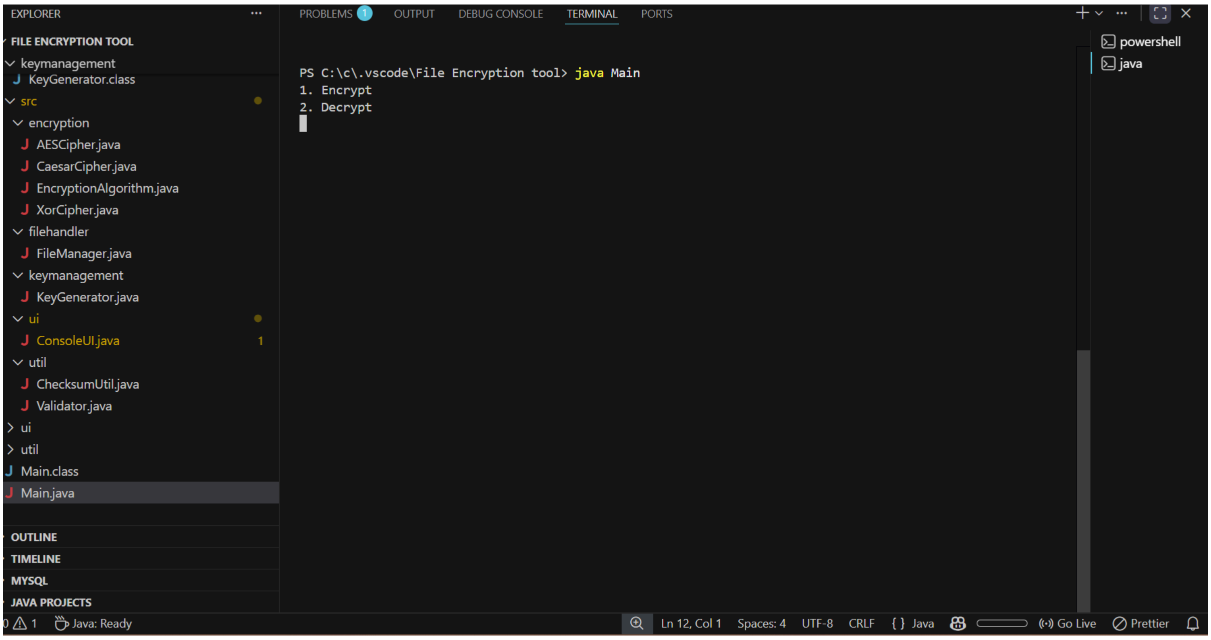
File data is processed in fixed-size chunks and stored in a queue following the FIFO principle. Each chunk is encrypted or decrypted sequentially, ensuring efficient memory usage and support for large files.

FLOW CHART

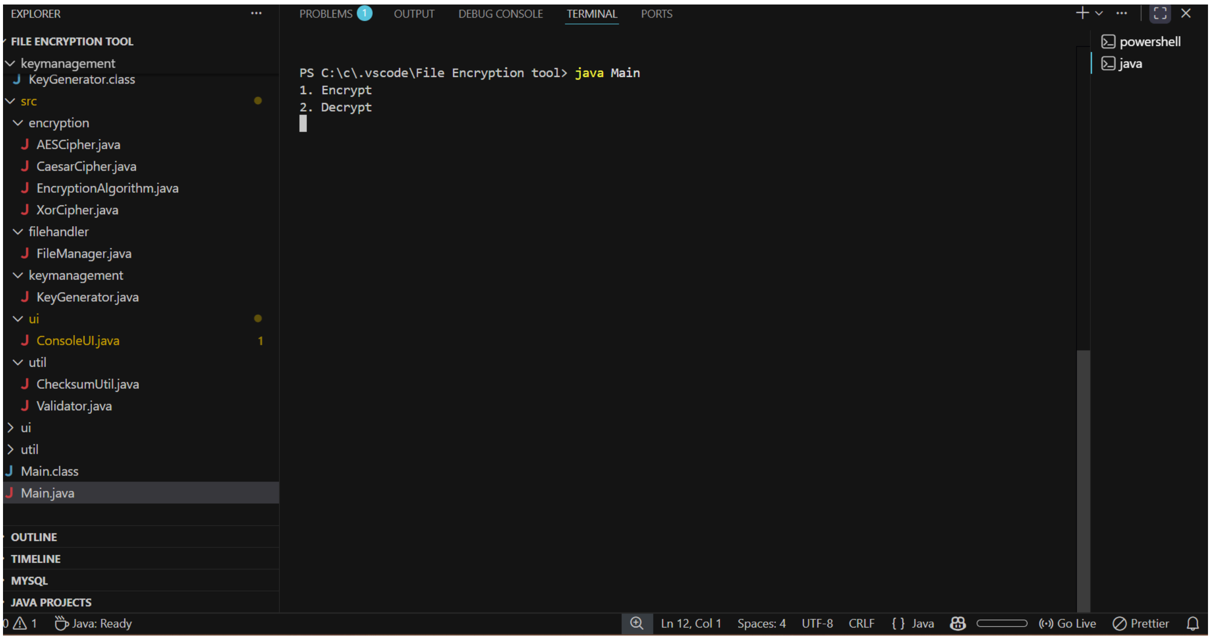

Start
Display Menu
User Choice?
Encrypt
Decrypt
ENCRYPTION PROCESS
Input File Path
Validate File Path
Invalid
Valid
Display Error (Encryption)
Input Output File Path
Select Encryption Algorithm
Caesar Cipher
XOR Cipher
AES
Caesar Cipher
XOR Cipher
AES
Enter Password
Generate Salt and Derive Key
Read File as Byte Stream
Divide File into Chunks (Queue)
Encrypt Each Chunk Using Selected Algorithm
Write Salt and Encrypted Data to Output File
Display Encryption Successful
End (Encryption)
DECRYPTION PROCESS
Input Encrypted File Path
Validate File Path (Decryption)
Invalid
Valid
Display Error (Decryption)
Input Output File Path (Decryption)
Select Encryption Algorithm Used
Enter Password (Decryption)
Read Salt from Encrypted File
Derive Key Using Password and Salt
Read Encrypted Data as Byte Stream
Divide Encrypted Data into Chunks (Queue)
Decrypt Each Chunk Using Selected Algorithm
Write Decrypted Data to Output File
Display Decryption Successful
End (Decryption)
eraser

5. SCREENSHOT OF EXECUTION

Running the program

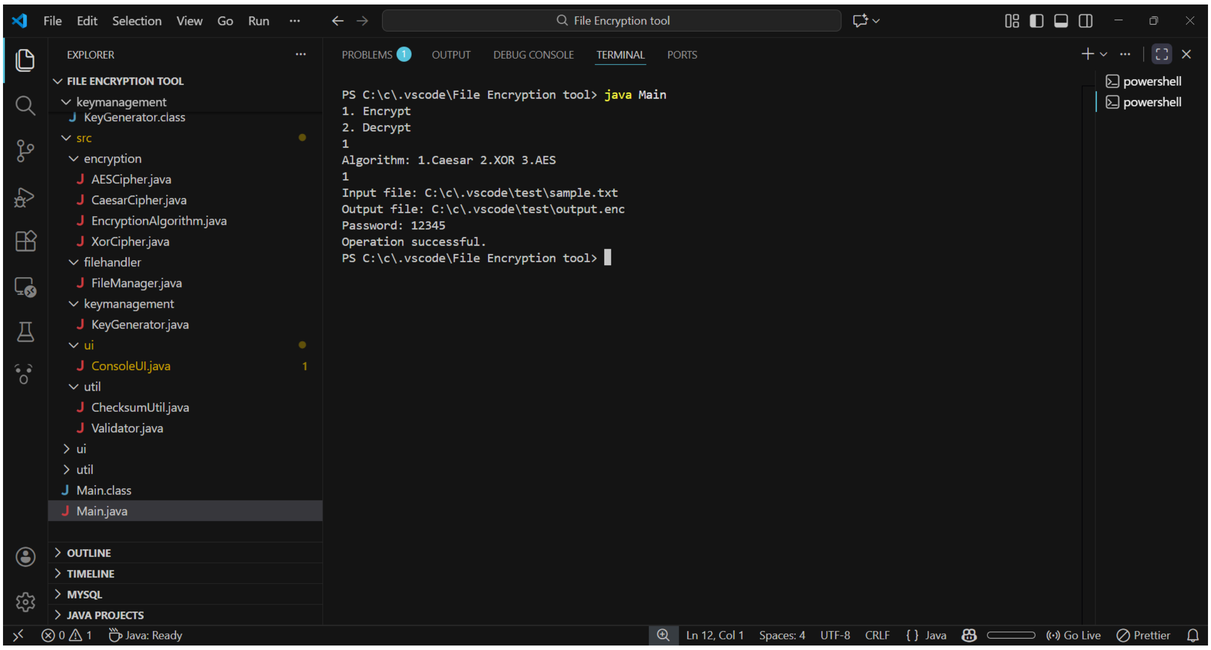


```
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
```



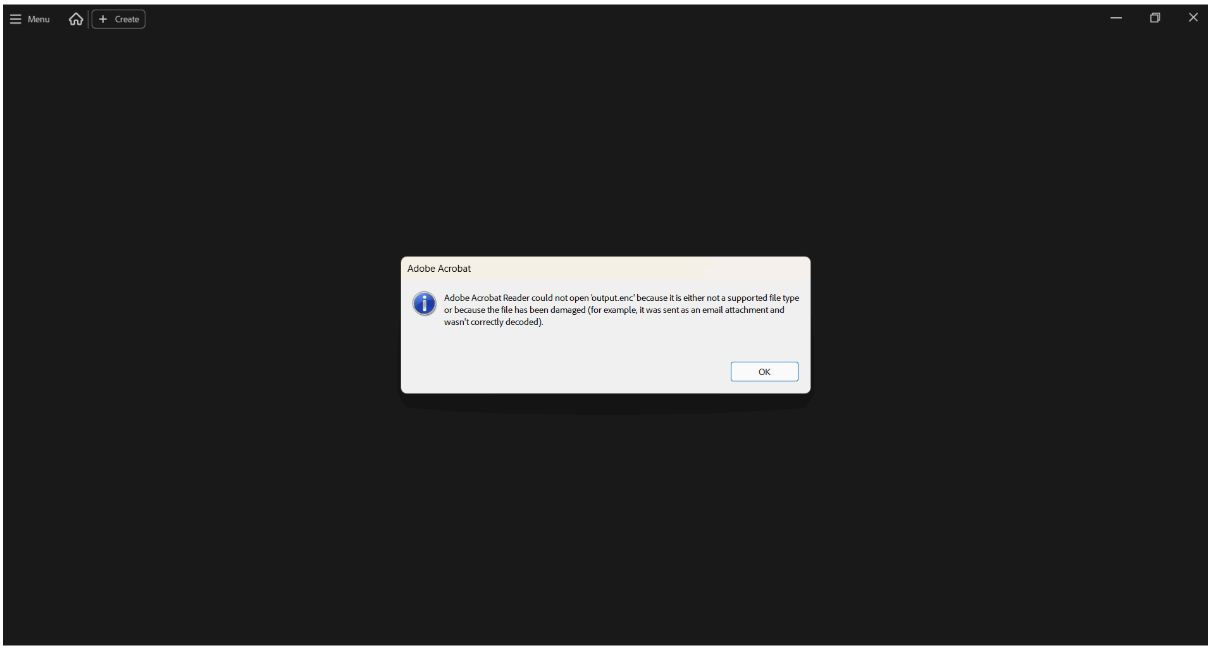
```
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
```

Successful execution



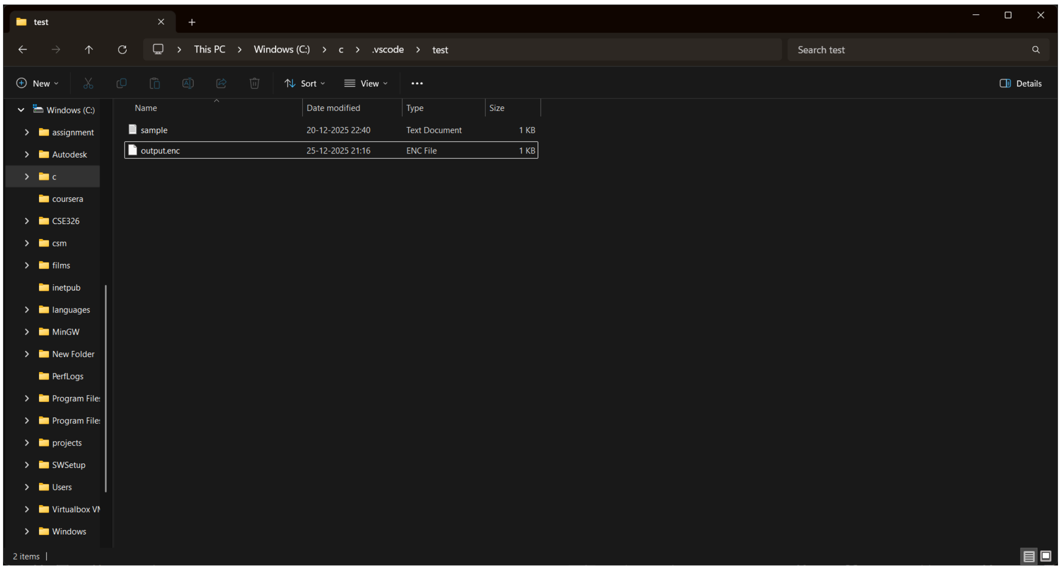
```
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
1
Algorithm: 1.Caesar 2.XOR 3.AES
1
Input file: C:\c\.vscode\test\sample.txt
Output file: C:\c\.vscode\test\output.enc
Password: 12345
Operation successful.
PS C:\c\.vscode\File Encryption tool> 
```

Encrypted file



Adobe Acrobat

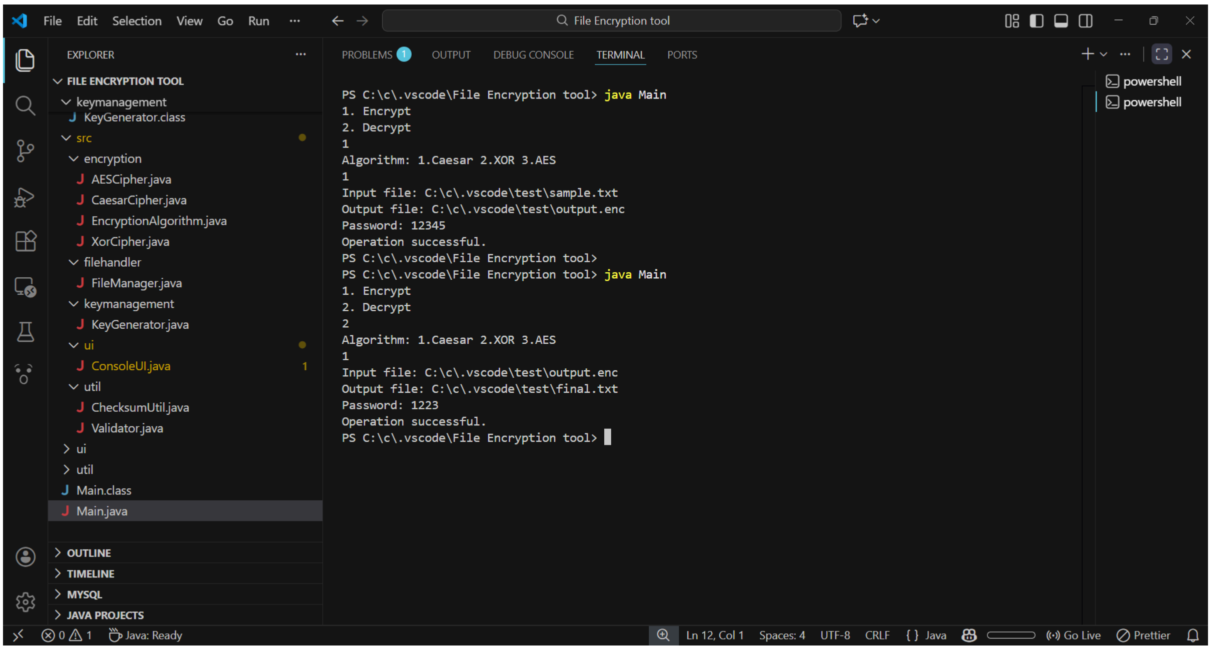
Adobe Acrobat Reader could not open 'output.enc' because it is either not a supported file type or because the file has been damaged (for example, it was sent as an email attachment and wasn't correctly decoded).



This PC > Windows (C:) > c > .vscode > test

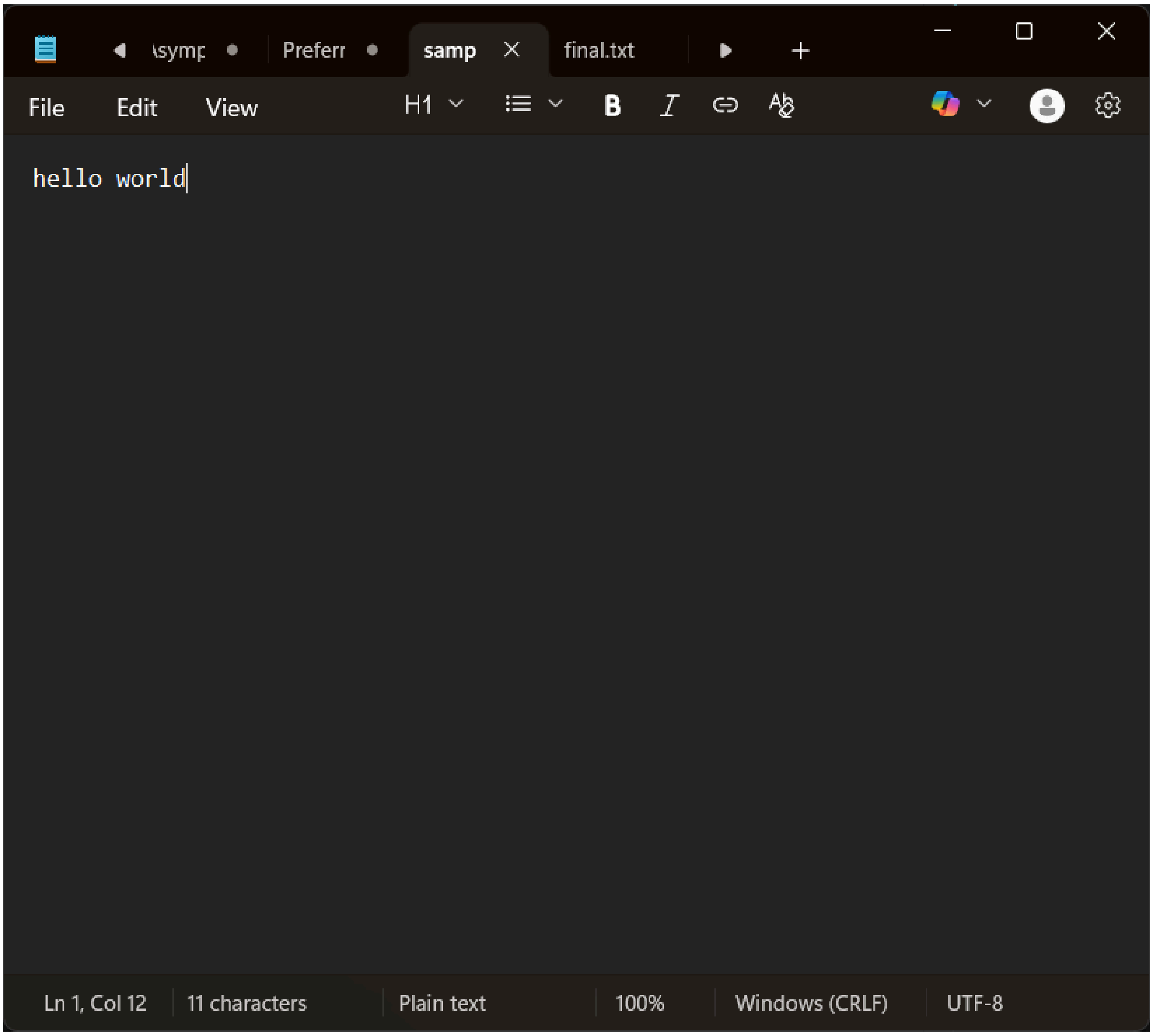
Name	Date modified	Type	Size
sample	20-12-2025 22:40	Text Document	1 KB
output.enc	25-12-2025 21:16	ENC File	1 KB

Decryption



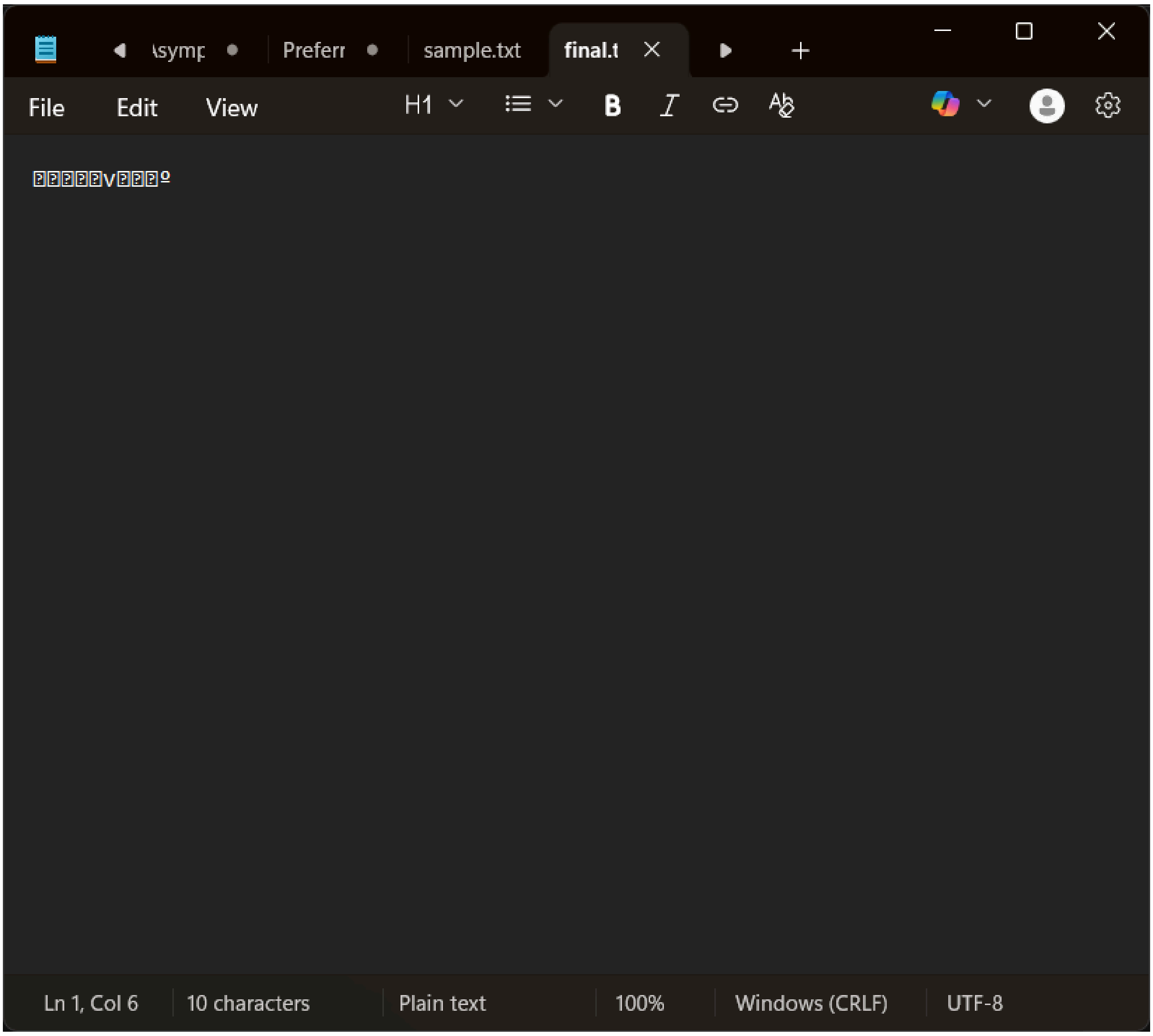
```
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
1
Algorithm: 1.Caesar 2.XOR 3.AES
1
Input file: C:\c\.vscode\test\sample.txt
Output file: C:\c\.vscode\test\output.enc
Password: 12345
Operation successful.
PS C:\c\.vscode\File Encryption tool>
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
2
Algorithm: 1.Caesar 2.XOR 3.AES
1
Input file: C:\c\.vscode\test\output.enc
Output file: C:\c\.vscode\test\final.txt
Password: 1223
Operation successful.
PS C:\c\.vscode\File Encryption tool> 
```

Sample file



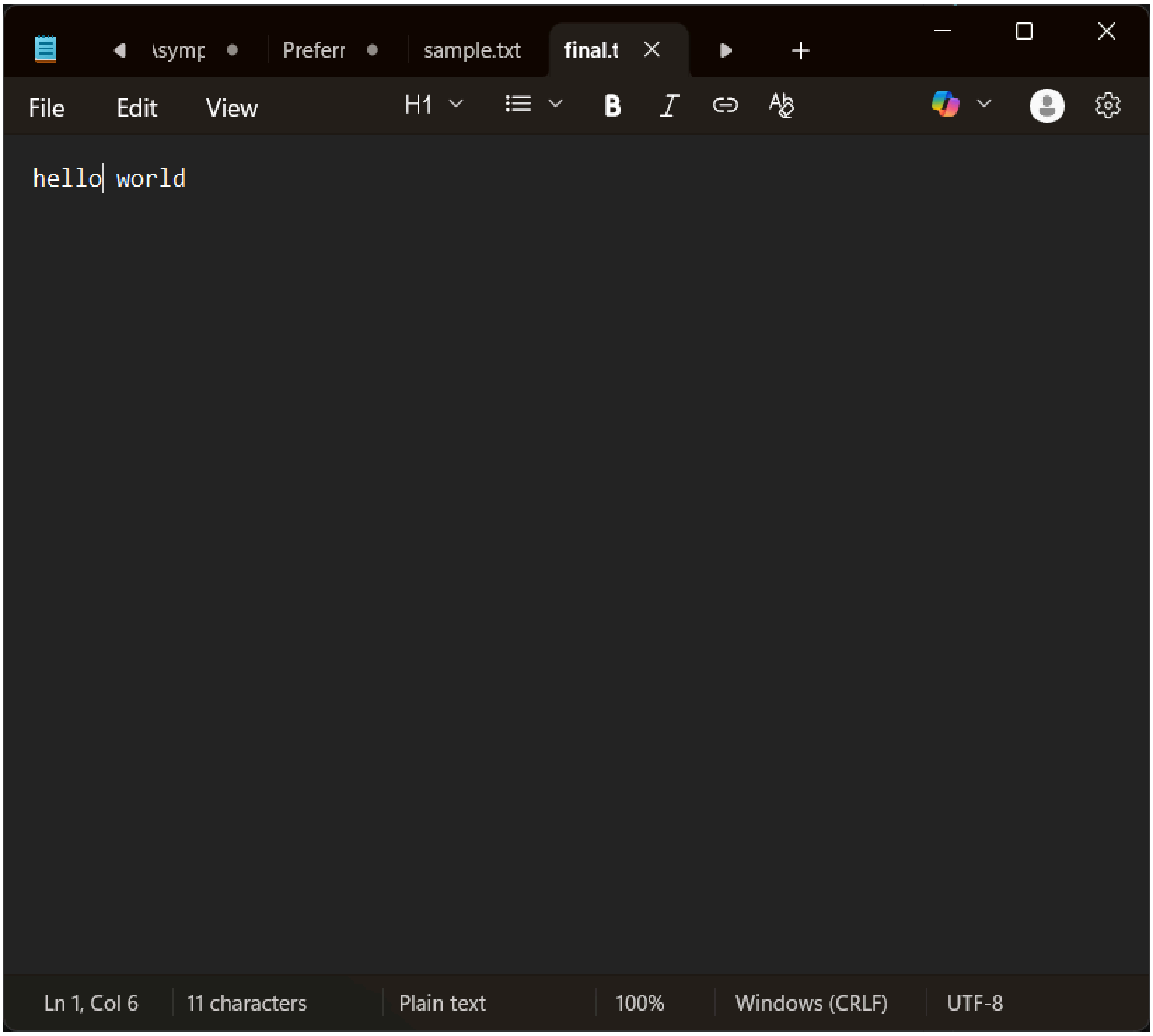
```
hello world
```

Decrypted file with wrong password



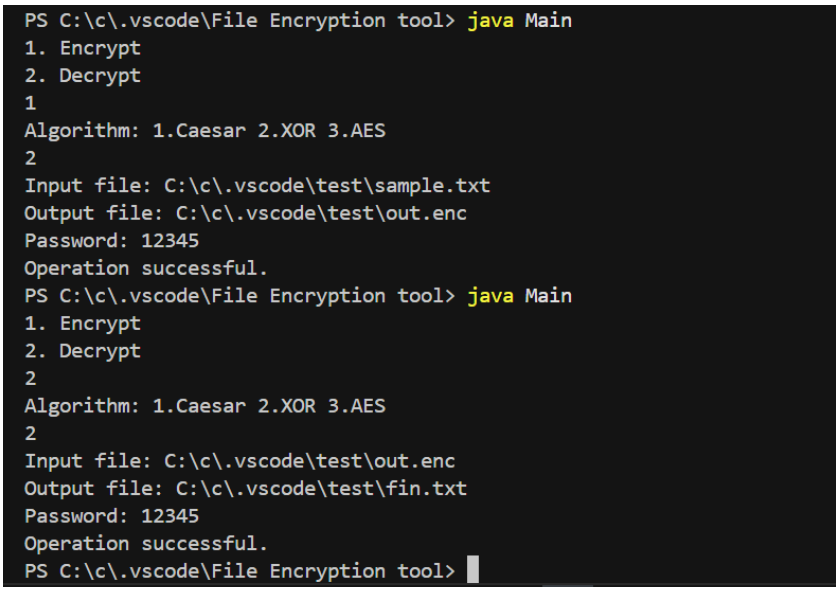
```
⍰⍰⍰⍰⍰v⍰⍰⍰º
```

With right password

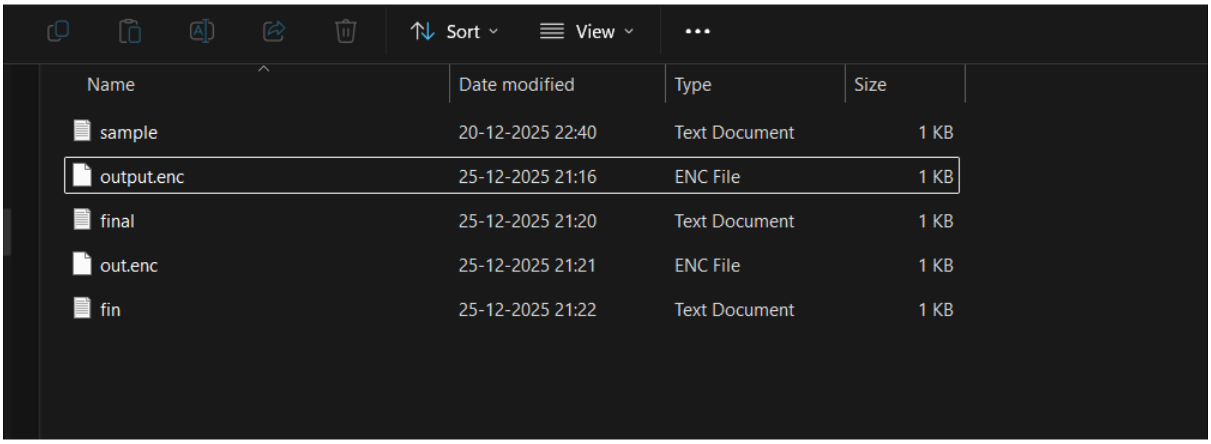


hello world

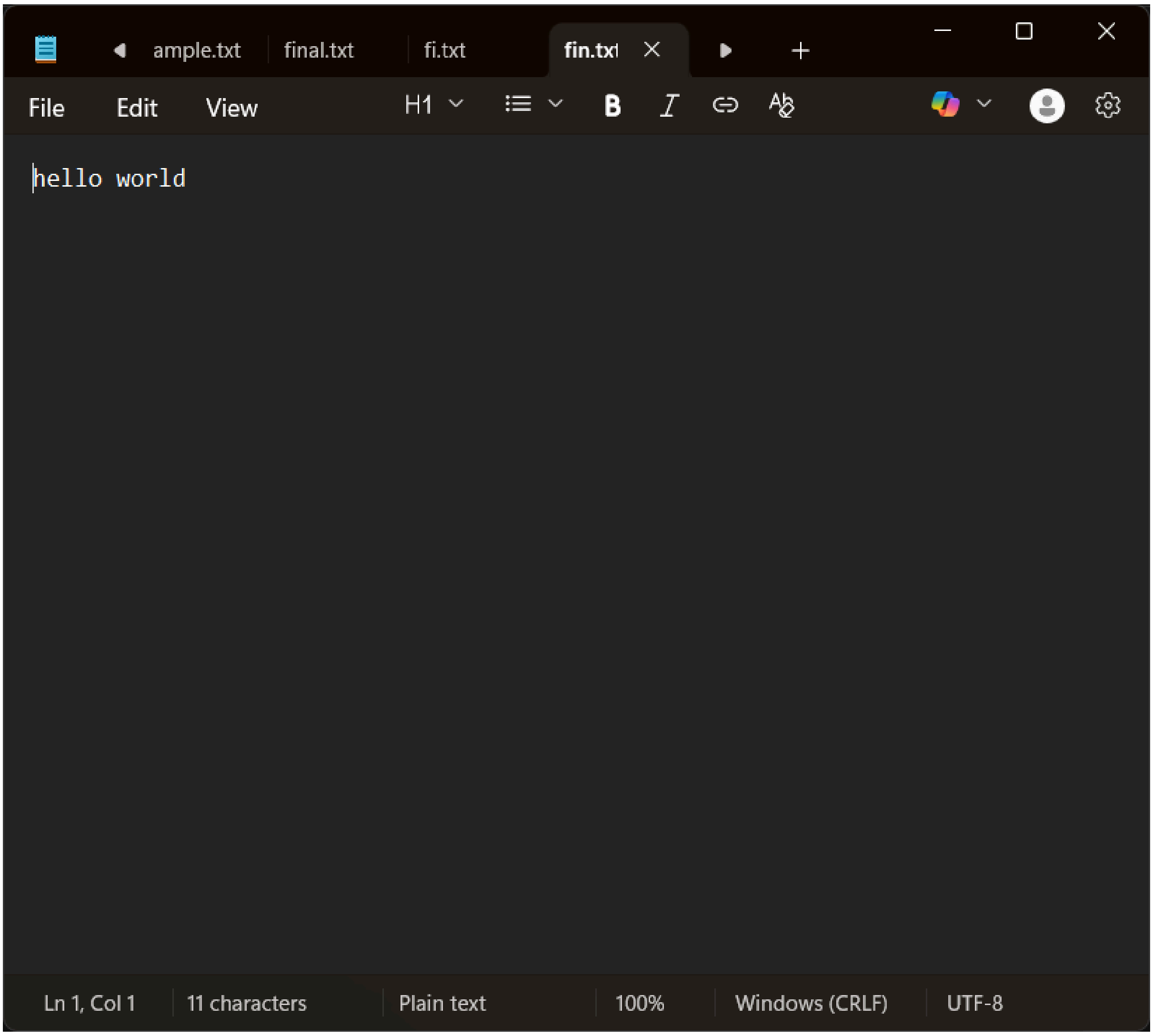
Execution with all algorithms and outputs



```
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
1
Algorithm: 1.Caesar 2.XOR 3.AES
2
Input file: C:\c\.vscode\test\sample.txt
Output file: C:\c\.vscode\test\out.enc
Password: 12345
Operation successful.
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
2
Algorithm: 1.Caesar 2.XOR 3.AES
2
Input file: C:\c\.vscode\test\out.enc
Output file: C:\c\.vscode\test\fin.txt
Password: 12345
Operation successful.
PS C:\c\.vscode\File Encryption tool>
```

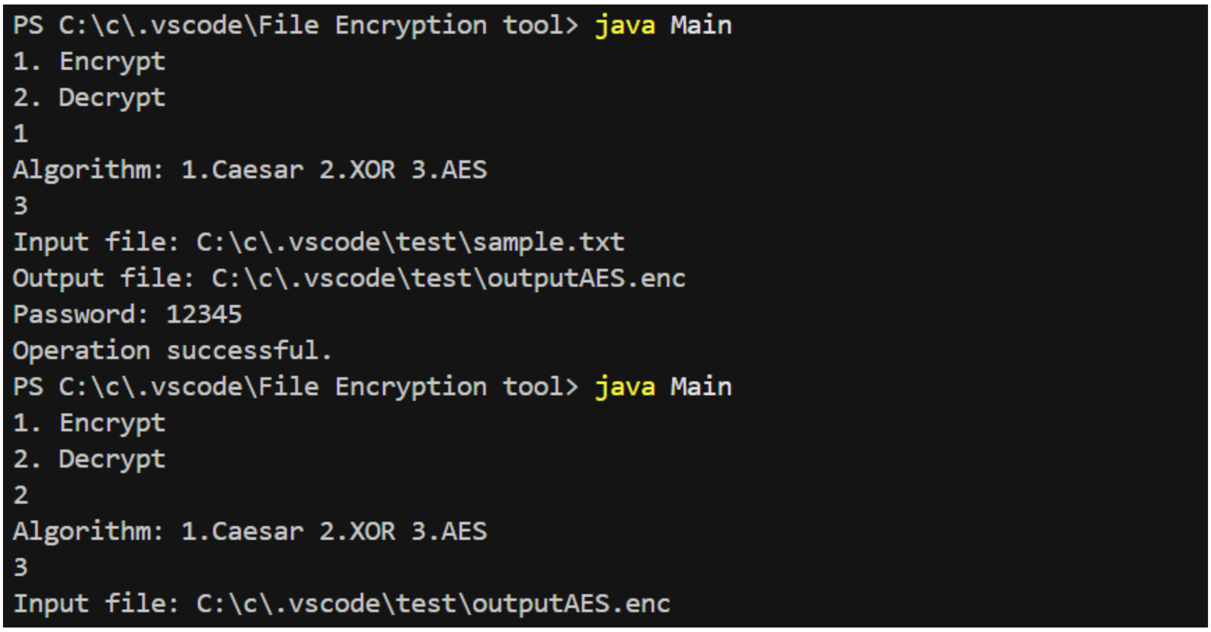


Name	Date modified	Type	Size
sample	20-12-2025 22:40	Text Document	1 KB
output.enc	25-12-2025 21:16	ENC File	1 KB
final	25-12-2025 21:20	Text Document	1 KB
out.enc	25-12-2025 21:21	ENC File	1 KB
fin	25-12-2025 21:22	Text Document	1 KB

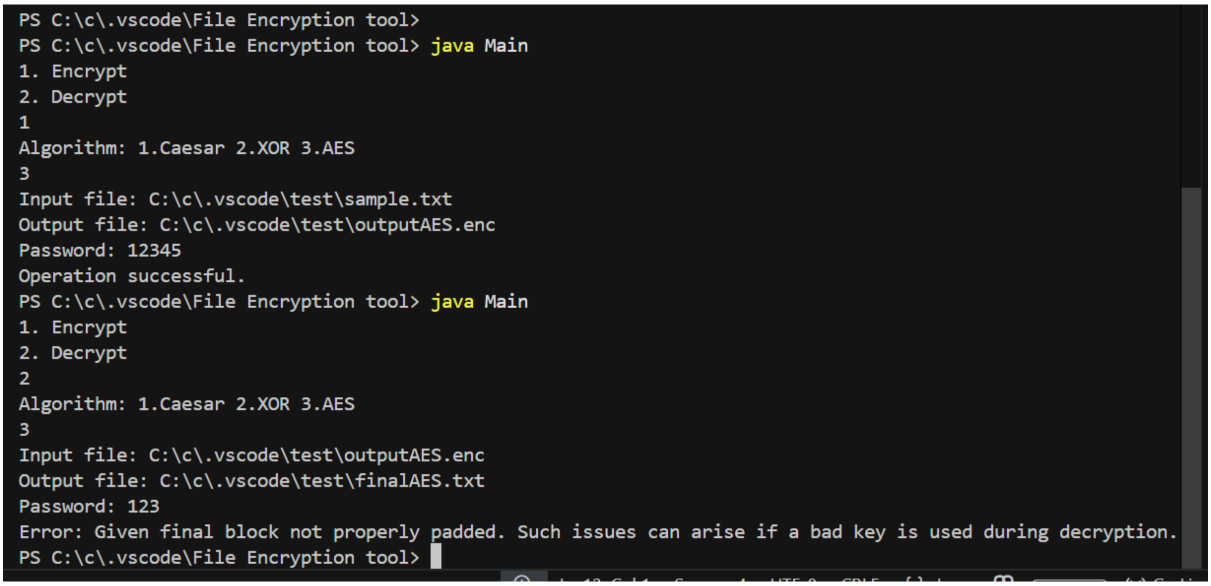


```
hello world
```

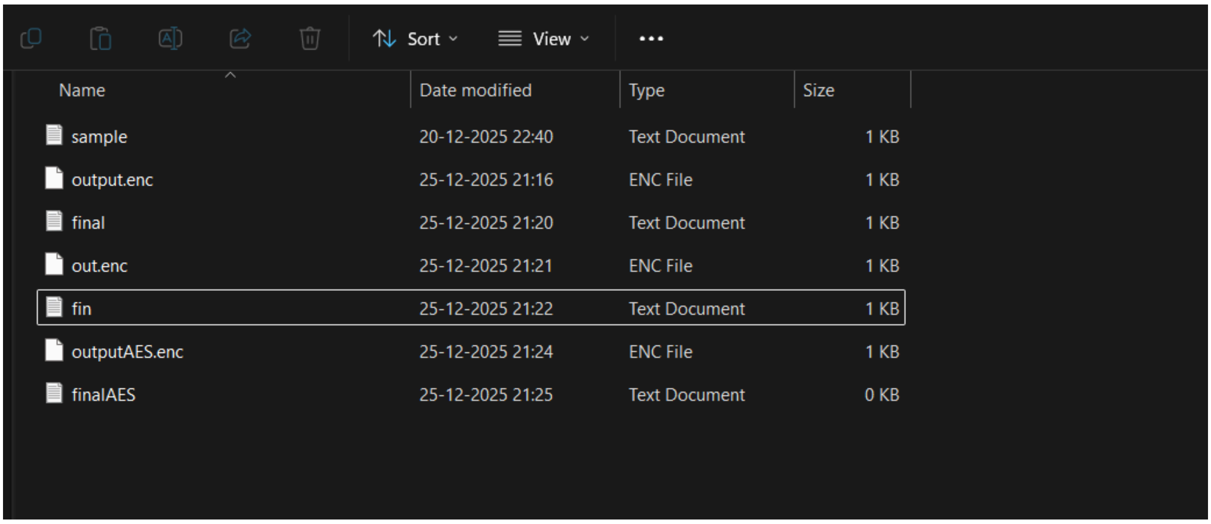
Ln 1, Col 1 | 11 characters | Plain text | 100% | Windows (CRLF) | UTF-8



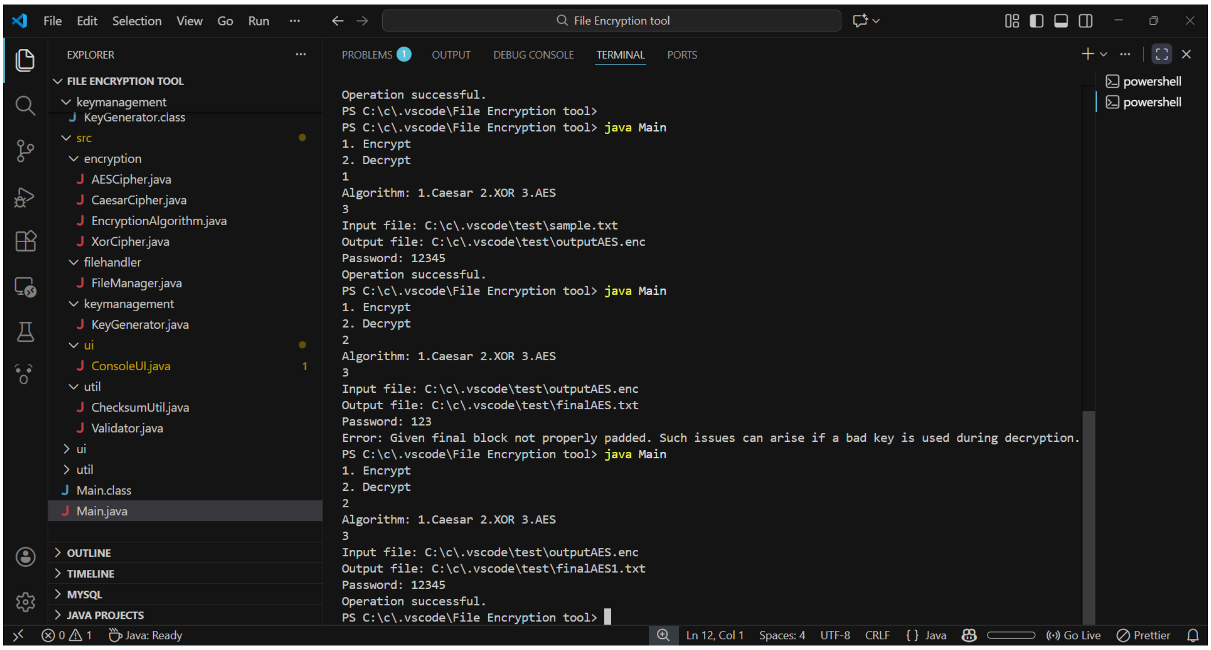
```
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
1
Algorithm: 1.Caesar 2.XOR 3.AES
3
Input file: C:\c\.vscode\test\sample.txt
Output file: C:\c\.vscode\test\outputAES.enc
Password: 12345
Operation successful.
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
2
Algorithm: 1.Caesar 2.XOR 3.AES
3
Input file: C:\c\.vscode\test\outputAES.enc
```



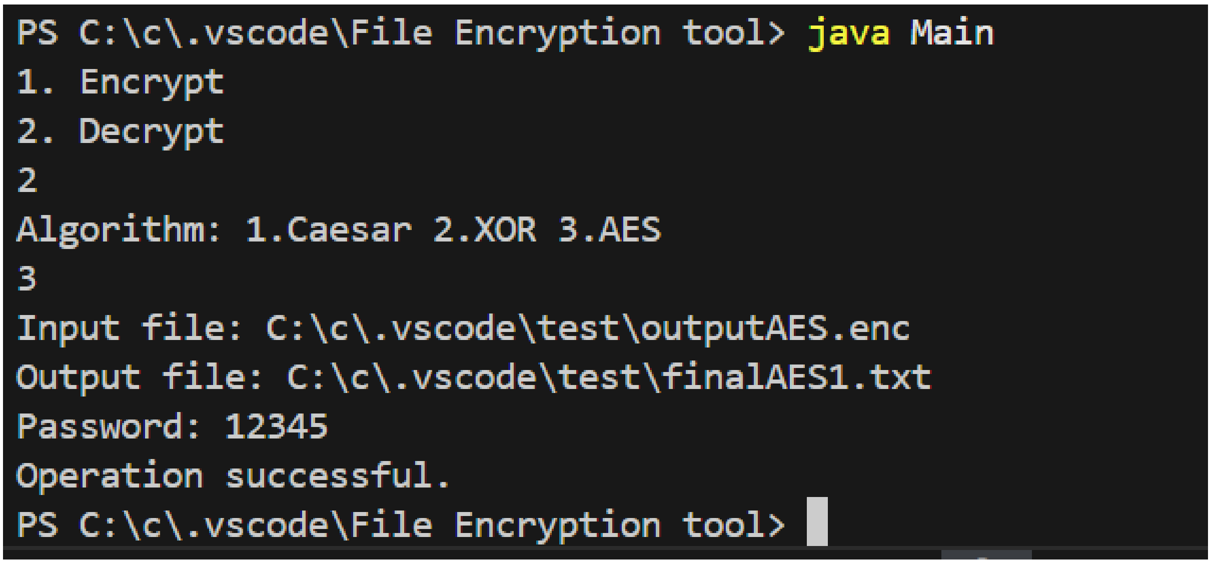
```
PS C:\c\.vscode\File Encryption tool>
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
1
Algorithm: 1.Caesar 2.XOR 3.AES
3
Input file: C:\c\.vscode\test\sample.txt
Output file: C:\c\.vscode\test\outputAES.enc
Password: 12345
Operation successful.
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
2
Algorithm: 1.Caesar 2.XOR 3.AES
3
Input file: C:\c\.vscode\test\outputAES.enc
Output file: C:\c\.vscode\test\finalAES.txt
Password: 123
Error: Given final block not properly padded. Such issues can arise if a bad key is used during decryption.
PS C:\c\.vscode\File Encryption tool>
```



Name	Date modified	Type	Size
sample	20-12-2025 22:40	Text Document	1 KB
output.enc	25-12-2025 21:16	ENC File	1 KB
final	25-12-2025 21:20	Text Document	1 KB
out.enc	25-12-2025 21:21	ENC File	1 KB
fin	25-12-2025 21:22	Text Document	1 KB
outputAES.enc	25-12-2025 21:24	ENC File	1 KB
finalAES	25-12-2025 21:25	Text Document	0 KB



```
Operation successful.
PS C:\c\.vscode\File Encryption tool>
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
1
Algorithm: 1.Caesar 2.XOR 3.AES
3
Input file: C:\c\.vscode\test\sample.txt
Output file: C:\c\.vscode\test\outputAES.enc
Password: 12345
Operation successful.
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
2
Algorithm: 1.Caesar 2.XOR 3.AES
3
Input file: C:\c\.vscode\test\outputAES.enc
Output file: C:\c\.vscode\test\finalAES.txt
Password: 123
Error: Given final block not properly padded. Such issues can arise if a bad key is used during decryption.
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
2
Algorithm: 1.Caesar 2.XOR 3.AES
3
Input file: C:\c\.vscode\test\outputAES.enc
Output file: C:\c\.vscode\test\finalAES1.txt
Password: 12345
Operation successful.
PS C:\c\.vscode\File Encryption tool>
```



```
PS C:\c\.vscode\File Encryption tool> java Main
1. Encrypt
2. Decrypt
2
Algorithm: 1.Caesar 2.XOR 3.AES
3
Input file: C:\c\.vscode\test\outputAES.enc
Output file: C:\c\.vscode\test\finalAES1.txt
Password: 12345
Operation successful.
PS C:\c\.vscode\File Encryption tool>
```

6. SUMMARY

The File Encryption/Decryption Tool is a practical implementation of data structures and encryption techniques. The project demonstrates how theoretical concepts of queues and algorithms can be applied to solve real-world security problems.

The system successfully encrypts and decrypts files using password-based encryption, ensuring data confidentiality. The use of AES provides strong security, while Caesar and XOR ciphers help demonstrate algorithmic concepts.

---

7. BIBLIOGRAPHY

- Class Notes
- Up grade
- GitHub References

---

8. ANNEXURE

SOURCE CODE

Encryption

AESCipher.java

```
package encryption;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

public class AESCipher implements EncryptionAlgorithm {

    @Override
    public byte[] encrypt(byte[] data, byte[] key) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, new SecretKeySpec(key,
"AES"));
        return cipher.doFinal(data);
    }

    @Override
    public byte[] decrypt(byte[] data, byte[] key) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, new SecretKeySpec(key,
"AES"));
        return cipher.doFinal(data);
    }
}
```

CeasarCipher.java

```java
package encryption;

public class CaesarCipher implements EncryptionAlgorithm {

    @Override
    public byte[] encrypt(byte[] data, byte[] key) {
        int shift = key[0];
        byte[] out = new byte[data.length];
        for (int i = 0; i < data.length; i++) {
            out[i] = (byte) (data[i] + shift);
        }
        return out;
    }

    @Override
    public byte[] decrypt(byte[] data, byte[] key) {
        int shift = key[0];
        byte[] out = new byte[data.length];
        for (int i = 0; i < data.length; i++) {
            out[i] = (byte) (data[i] - shift);
        }
        return out;
    }
}
```

EncryptionAlgorithm.java

```java
package encryption;

public interface EncryptionAlgorithm {
    byte[] encrypt(byte[] data, byte[] key) throws Exception;
    byte[] decrypt(byte[] data, byte[] key) throws Exception;
}
```

XORCipher.java

```java
package encryption;

public class XorCipher implements EncryptionAlgorithm {

    @Override
    public byte[] encrypt(byte[] data, byte[] key) {
        byte[] out = new byte[data.length];
        for (int i = 0; i < data.length; i++) {
            out[i] = (byte) (data[i] ^ key[i % key.length]);
        }
        return out;
    }

    @Override
    public byte[] decrypt(byte[] data, byte[] key) {
        return encrypt(data, key);
    }
}
```

Filehandler

Filemanager.java

```
package filehandler;

import encryption.EncryptionAlgorithm;
import keymanagement.KeyGenerator;

import java.io.*;
import java.util.LinkedList;
import java.util.Queue;

public class FileManager {

    private static final int CHUNK_SIZE = 4096;

    public static void encrypt(
            File input,
            File output,
            EncryptionAlgorithm algorithm,
            String password
    ) throws Exception {

        byte[] salt = KeyGenerator.generateSalt();
        byte[] key = KeyGenerator.deriveKey(password, salt);

        Queue<byte[]> queue = new LinkedList<>();

        try (InputStream in = new BufferedInputStream(new
FileInputStream(input))) {
            byte[] buffer;
            int read;
            while ((read = in.read(buffer = new byte[CHUNK_SIZE])) !=
-1) {
                byte[] block = new byte[read];
                System.arraycopy(buffer, 0, block, 0, read);
                queue.add(block);
            }
        }

        try (OutputStream out = new BufferedOutputStream(new
FileOutputStream(output))) {
            out.write(salt); //

            while (!queue.isEmpty()) {
                byte[] enc = algorithm.encrypt(queue.poll(), key);
                out.write(enc);
            }
        }
    }

    public static void decrypt(
            File input,
            File output,
            EncryptionAlgorithm algorithm,
            String password
    ) throws Exception {

        try (InputStream in = new BufferedInputStream(new
FileInputStream(input))) {

            byte[] salt = new byte[16];
            if (in.read(salt) != 16) {
                throw new SecurityException("Invalid encrypted file
format");
            }

            byte[] key = KeyGenerator.deriveKey(password, salt);

            try (OutputStream out = new BufferedOutputStream(new
FileOutputStream(output))) {
                byte[] buffer;
                int read;
                while ((read = in.read(buffer = new byte[CHUNK_SIZE]))
!= -1) {
                    byte[] block = new byte[read];
                    System.arraycopy(buffer, 0, block, 0, read);
                    out.write(algorithm.decrypt(block, key));
                }
            }
        }
    }
}
```

Key management

KeyGenerator.java

```
package keymanagement;

import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import java.security.SecureRandom;

public class KeyGenerator {

    public static byte[] generateSalt() {
        byte[] salt = new byte[16];
        new SecureRandom().nextBytes(salt);
        return salt;
    }

    public static byte[] deriveKey(String password, byte[] salt) throws
Exception {
        PBEKeySpec spec = new PBEKeySpec(
                password.toCharArray(),
                salt,
                65536,
                256
        );
        return SecretKeyFactory
                .getInstance("PBKDF2WithHmacSHA256")
                .generateSecret(spec)
                .getEncoded();
    }
}
```

Ui

ConsoleUI.java

```java
package ui;

import encryption.*;
import filehandler.FileManager;
import util.Validator;

import java.io.File;
import java.util.HashMap;
import java.util.Scanner;

public class ConsoleUI {

    private static final HashMap<Integer, EncryptionAlgorithm> algos =
new HashMap<>();

    static {
        algos.put(1, new CaesarCipher());
        algos.put(2, new XorCipher());
        algos.put(3, new AESCipher());
    }

    public static void start() throws Exception {
        Scanner sc = new Scanner(System.in);

        System.out.println("1. Encrypt\n2. Decrypt");
        int mode = sc.nextInt();
        sc.nextLine();

        System.out.println("Algorithm: 1.Caesar 2.XOR 3.AES");
        int algo = sc.nextInt();
        sc.nextLine();

        System.out.print("Input file: ");
        File in = new File(sc.nextLine());
        Validator.validateFile(in);

        System.out.print("Output file: ");
        File out = new File(sc.nextLine());

        System.out.print("Password: ");
        String pass = sc.nextLine();
        if (mode == 1) {
            FileManager.encrypt(in, out, algos.get(algo), pass);
        } else {
            FileManager.decrypt(in, out, algos.get(algo), pass);
        }

        System.out.println("Operation successful.");
    }
}
```

Util

ChecksumUtil

```
package util;

import java.io.FileInputStream;
import java.security.MessageDigest;

public class ChecksumUtil {

    public static String sha256(String path) throws Exception {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        try (FileInputStream fis = new FileInputStream(path)) {
            byte[] buf = new byte[4096];
            int r;
            while ((r = fis.read(buf)) != -1) {
                md.update(buf, 0, r);
            }
        }
        StringBuilder sb = new StringBuilder();
        for (byte b : md.digest())
            sb.append(String.format("%02x", b));
        return sb.toString();
    }
}
```

Validator.java

```java
package util;

import java.io.File;

public class Validator {

    public static void validateFile(File f) {
        if (f == null || !f.exists() || !f.isFile())
            throw new IllegalArgumentException("Invalid file path");
    }
}
```

Main.java

```java
import ui.ConsoleUI;

public class Main {
    public static void main(String[] args) {
        try {
            ConsoleUI.start();
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```