|  |  |
| --- | --- |
| **ITRI615** |  |
| **Name & Surname:**  Michael Mathebula |
| **Study unit:**  Project 1 Cryptography  (Assignment 3) |
| **Date:**  2025/05/06 |
| **Student number:**  38925958 |
| By handing in the homework, I declare it is my own work, and all sources used are acknowledged in full in the list of references. (MM) |

|  |  |
| --- | --- |
| **ITRI615** |  |
| **Name & Surname:**  Salim Sofinia |
| **Study unit:**  Project 1 Cryptography  (Assignment 3) |
| **Date:**  2025/05/06 |
| **Student number:**  37284177 |
| By handing in the homework, I declare it is my own work, and all sources used are acknowledged in full in the list of references. (SS) |

A black and white logo

AI-generated content may be incorrect.

Michael Mathebula & Salim Sofinia

38925958 & 37284177

File & Text Cryptography Application User Manual

Project 1 Cryptography (Assignment 3)

Table of Contents

[1. Introduction 4](#_Toc197461232)

[2. System Requirements 4](#_Toc197461233)

[3. Installation Instructions 4](#_Toc197461234)

[4. Graphical User Interface 5](#_Toc197461235)

[4.1. Text Encryption/Decryption Section 5](#_Toc197461236)

[4.2. File Encryption/Decryption Section 6](#_Toc197461237)

[4.3. Methods 6](#_Toc197461238)

[4.4. User Instructions 7](#_Toc197461239)

[4.4.1. Example keys for each cryptography algorithm 7](#_Toc197461240)

[4.5. Error Handling 7](#_Toc197461241)

[4.6. Example Outputs 8](#_Toc197461242)

[4.6.1. Text Encryption and Decryption 8](#_Toc197461243)

[5. Cryptography Algorithms 12](#_Toc197461244)

[5.1. Vigenère Cipher 12](#_Toc197461245)

[5.1.1. Overview 12](#_Toc197461246)

[5.1.2. Vigenère Text Cipher 12](#_Toc197461247)

[5.1.3. Vigenère File Cipher 14](#_Toc197461248)

[5.2. Vernam Cipher 15](#_Toc197461249)

[5.2.1. Overview 15](#_Toc197461250)

[5.2.2. Vernam Text Cipher 16](#_Toc197461251)

[5.2.3. Vernam File Cipher 17](#_Toc197461252)

[5.3. Columnar Transposition Cipher 18](#_Toc197461253)

[5.3.1. Overview 18](#_Toc197461254)

[5.3.2. Columnar Transposition Text Cipher 19](#_Toc197461255)

[5.3.3. Columnar Transposition File Cipher 21](#_Toc197461256)

[5.4. Custom Cipher 23](#_Toc197461257)

[5.4.1. Overview 23](#_Toc197461258)

[5.4.2. Custom Text Cipher 24](#_Toc197461259)

[5.4.3. Custom File Cipher 26](#_Toc197461260)

[5.5. Vernam + Transposition Cipher 26](#_Toc197461261)

[5.5.1. Overview 26](#_Toc197461262)

[5.5.2. Vernam + Transposition Text Cipher 27](#_Toc197461263)

[5.5.3. Vernam + Transposition File Cipher 27](#_Toc197461264)

[6. Security Considerations 29](#_Toc197461265)

# Introduction

This document’s purpose is to provide users with guidance and information on the cryptography application, as well as act as a technical report, for the Java-based Text and File Cryptography Application. This document will assist users with application installation and usage, as well as provide a concise and comprehensive explanation on how each of the cryptography algorithms function.

The supported cryptography algorithms in the application are:

* Vigenère algorithm
* Vernam algorithm
* Columnar Transposition algorithm
* Custom algorithm
* Vernam + Transposition algorithm

# System Requirements

To successfully execute this cryptography application, ensure the environment correctly set up. The environment used is Java Development Kit (JDK), version 26.0.6 (LTS) or later. Below is an example of a terminal example when the java version is tested with “java -version”:

*java version "21.0.6" 2025-01-21 LTS*

*Java(TM) SE Runtime Environment (build 21.0.6+8-LTS-188)*

*Java HotSpot(TM) 64-Bit Server VM (build 21.0.6+8-LTS-188, mixed mode, sharing)*

Memory: Minimum of 512 MB RAM

Disk Space: Minimum of 10 MB for application files

Operating System: Windows 10/11, macOS, or any Linux distribution

# Installation Instructions

In the case that the incorrect environment is set up, and the incorrect version of Java is used, the user may visit <https://www.oracle.com/java/technologies/javase-downloads.html> to download the latest version of Java.

The files of the cryptography .java files should all be placed in the “src” folder. To ensure all the files in the application is present in the folder. The user may use the following list to verify existing files. It should be noted that these files are not to be modified or moved.

* CustomSecureCipher.java
* MainApp.java
* TranspositionCipher.java
* VernamAndTranspositionCipher.java
* VernamCipher.java
* VigenereCipher.java
* VigenereTextCipher.java

In the application files, specifically in the path:

"ITRI615EncryptionApp\ITRI615EncryptionApp.jar", is the executable file that represents the cryptography application for the user to interact with.

For developers, the following steps may be taken to rebuild this executable file.

1. Ensure the Extension Pack For Java is installed in visual studio code to allow java development.
2. Open the Command Palette by Pressing Ctrl + Shift +P (Windows).
3. Type or locate “Java:Export Jar” then click on it.
4. Select the MainApp folder from the drop down list.
5. Ensure all elements are selected then click “OK” to start building the project.
6. Once the build is completed, the output will be stored in the file directory with the same name as the folder containing your java files.

# Graphical User Interface

The ITRI615 Encryption Application has a clean and user-friendly interface designed to help users perform encryption and decryption efficiently. It supports 5 distinct algorithms. The application interface is split into two main functional sections each dedicated to a specific type of data.

## Text Encryption/Decryption Section

|  |  |
| --- | --- |
| Algorithm Selection | |
| Label | Algorithm |
| Control | Drop down list/ Combo box |
| Default Selection | Vigenère Cipher |

|  |  |
| --- | --- |
| Encryption Key (Left Panel) | |
| Label | Encryption Key |
| Control | Text-box |

|  |  |
| --- | --- |
| Input Text & Output Text boxes | |
| Label | Input Text & Output Text |
| Control | Input textbox: Text inserted by the user for encryption or decryption.  Output text box: A read-only textbox that displays the result of text encryption or decryption. |

|  |  |
| --- | --- |
| Buttons | |
| Encrypt Text | Performs text encryption process |
| Decrypt Text | Performs test decryption process |

## File Encryption/Decryption Section

|  |  |
| --- | --- |
| Algorithm Selection | |
| Label | Algorithm |
| Control | Drop down list/ Combo box |
| Default Selection | Vigenère Cipher |

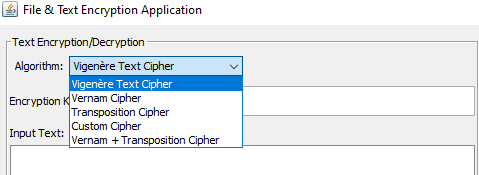
|  |  |
| --- | --- |
| Input & Output File | |
| Label | Input File/Output File |
| Control | * A read-only box that shows the path to the input file. * An editable text-box that shows the path to save the output file. |
| Button | Opens a file dialog to select files for encryption or decryption. |

|  |  |
| --- | --- |
| Buttons | |
| Encrypt File | Initiates the process of encryption files. |
| Decrypt File | Decrypts file provided an existing encrypted file with the right key is provided. |

|  |  |
| --- | --- |
| Bottom Panel: Status Display | |
| Label | Status |
| Control | Multiline read-only textbox used to display success messages, or logs. |

## Methods

As shown in the attached image, when the user clicks on the algorithm dropdown menu, they are presented with a list of encryption and decryption algorithms. These options allow the user to choose how they want to encrypt or decrypt either text or files, depending on their needs.



## User Instructions

### Example keys for each cryptography algorithm

* **Vigenère Algorithm:**
  + Anykey123
  + secureKey88
  + LIGHTHOUSE123
* **Vernam Algorithm:** 
  + Anykey123
  + CipherTrack12
  + KeyWave99#
* **Columnar Transposition Algorithm:** 
  + CopyrightPerkBundles
  + CampfiresBundGlowthy
  + StrongmanDeckBlowify
* **Custom Algorithm:**
  + Secret7Key
  + AnythingYouWant
  + unhackablefrfr
* **Vernam + Transposition Algorithm:**
  + CopyrightPerkBundles
  + CampfiresBundGlowthy
  + StrongmanDeckBlowify

## Error Handling

* The image below shows the error you get when you try to encrypt without entering a key.

**A screen shot of a computer error

AI-generated content may be incorrect.**

* The image below shows the error you get when you try to decrypt without entering a key.

**A screenshot of a computer error

AI-generated content may be incorrect.**

* The following error is displayed when an incorrect file is provided for decryption.

**A screenshot of a computer error

AI-generated content may be incorrect.**

## Example Outputs

### Text Encryption and Decryption

The following image depicts the outcome after text encryption.

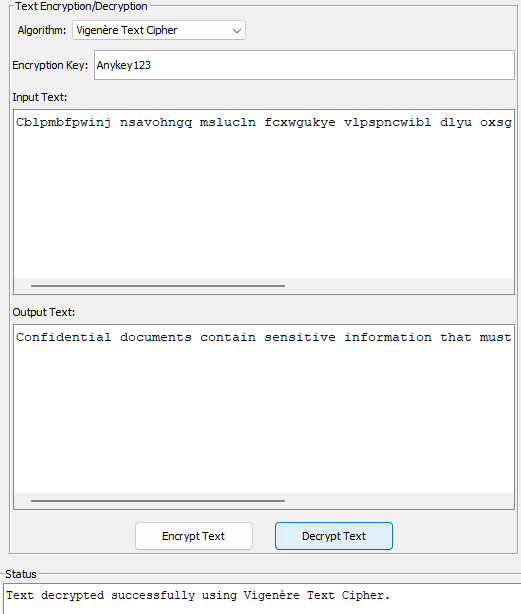
A screenshot of a computer

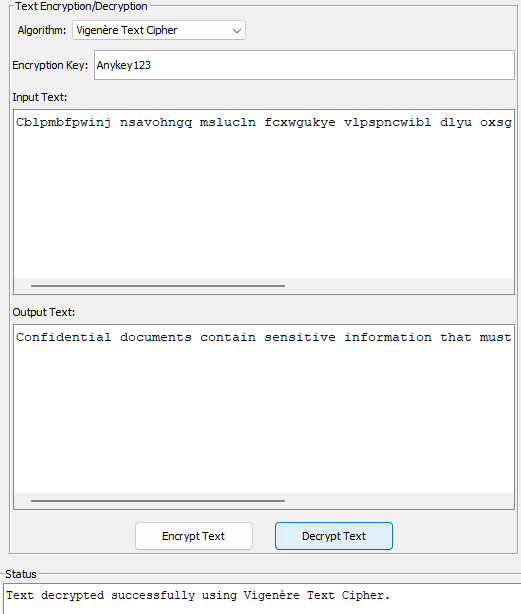
AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

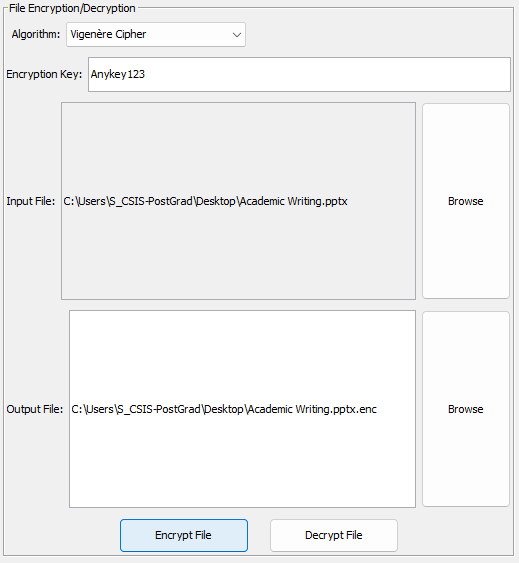
The following image shows the outcome after text decryption.

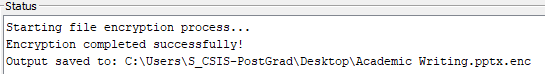




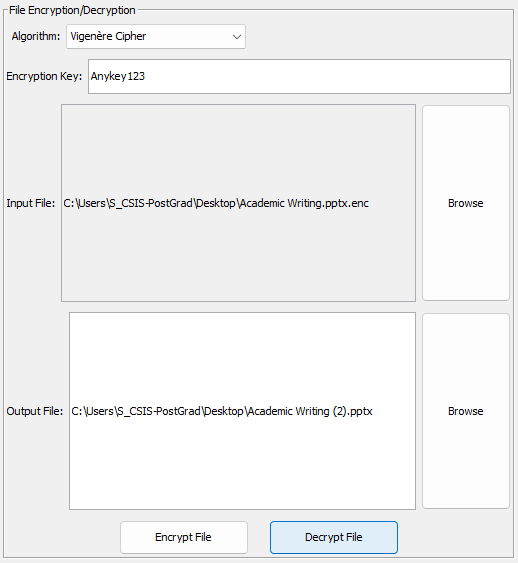
* + 1. File Encryption and Decryption

The following images shows the output of file encryption displayed on the status area.





The following images shows the output of file decryption displayed on the status area.



A screenshot of a computer

AI-generated content may be incorrect.

# Cryptography Algorithms

The table below demonstrates the persons responsible for the consequent ciphers:

|  |  |
| --- | --- |
| **Name and Surname** | **Cipher Responsibility** |
| Michael Mathebula | [Vigenère Cipher](#_Vigenère_Cipher) |
| Michael Mathebula | [Vernam Cipher](#_Vernam_Cipher) |
| Salim Sofinia | [Columnar Transposition Cipher](#_Columnar_Transposition_Cipher) |
| Salim Sofinia | [Custom Cipher](#_Custom_Cipher) |
| Salim Sofinia | [Vernam + Transposition Cipher](#_Vernam_+_Transposition) |

## Vigenère Cipher

### Overview

The Vigenère Cipher is a classical polyalphabetic substitution cipher. It encrypts alphabetic text by shifting each letter of the plaintext by an amount determined by a repeating key. This implementation has been made to handle non-letter characters which includes special handling for digits in the key.

### Vigenère Text Cipher

The Vigenère Cipher is also applied to text, not files in the application. Using a customized method to encrypt and decrypt text by shifting each letter based on a key. Non-alphabetic characters, like spaces and punctuation remain unchanged.

* Classic Vigenère Cipher but adapted to handle non-letter characters (keeps them unchanged).
* Special handling if the key contains digits (converts digits to letters, e.g. ‘0’ -> ‘A’, ‘1’ -> ‘B’).

**Key Preparation:** The key is repeated or truncated to match the length of the plaintext.

**Character Conversion:**

Plaintext and key characters are converted to uppercase.

Digits in the key (0–9) map to letters A–J.

**Encryption:**



The method loops through each plaintext character ensuring characters are letters or converted to letters and uses modulo 26 to ensure it stays within the alphabets range. Non-letter characters are appended directly to the encryption text without modification.

**Decryption:**





This method reverses the encryption process. The method subtracts the key value from the encrypted letter’s value and then applies modulo 26 to retrieve the original letter. Similar to encryption, non-letter characters remain unchanged.

### Vigenère File Cipher

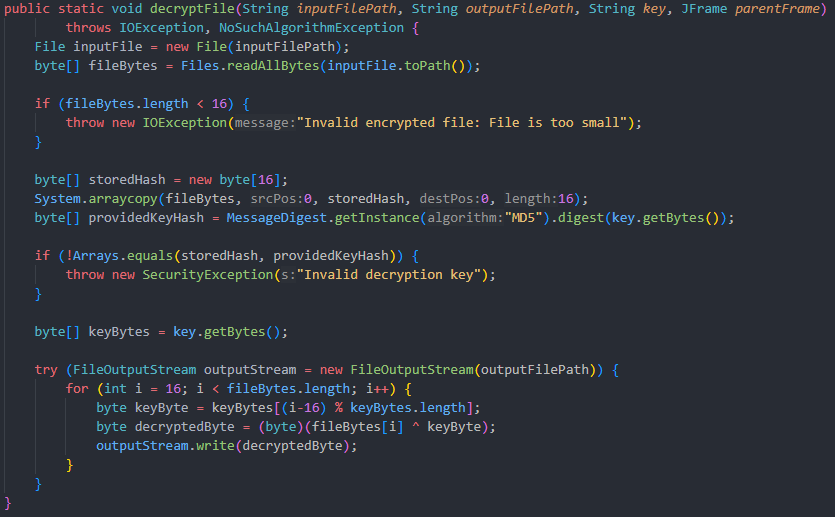
The VigenereCipher is not a traditional implementation of the Vigenère Cipher, but rather a simplified XOR-based file encryption system that mimics the concept of the Vigenère Cipher.

Files are read into a byte array.

**Encryption:**

Computes an MD5 hash of the key and writes it as the first 16 bytes of the output files. (This provides a simple way to verify the correct key during encryption). Bytes are XOR’ed with a corresponding byte of the key repeatedly.

**Decryption:**

****

Reads input file and extracts the 16 bytes as the stored hash key. It then verifies the key by computing its mD5 hash and comparing it with the stored one. After byte 17 and onwards, it XORs each byte with the corresponding byte of the repeating key to retrieve the original data.

**Important:**

A classic Vigenère Cipher works on letters (A-Z) not arbitrary bytes. This means it uses modular addition, however this code uses XOR operation. XOR operation is typically applied to text, not files, because it is based on a repeating alphabetic key.

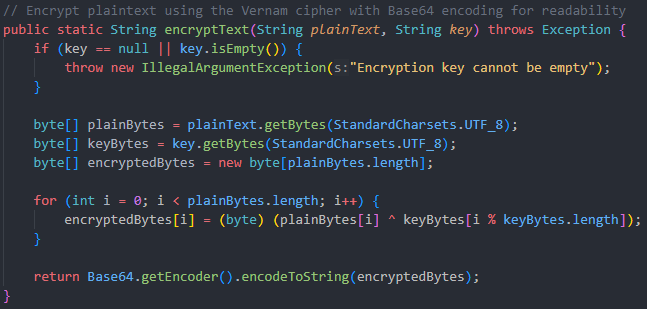
## Vernam Cipher

### Overview

The Vernam Cipher, known as the One-Time Pad (OTP), is a symmetric-key cipher which uses a key that is as long as the message to encrypt. Key is used once and then discarded, solidifying that each encryption remains unique. The key is combined with plain text using an XOR operation. In the application, if the key is shorter than the data being encrypted, it will start over from beginning once it has been fully used, this contrasts the difference between the true Vernam cipher and the implemented one.

### Vernam Text Cipher

**Encryption:**



Text encryption using the Vernam cipher works by applying an XOR operation between each byte of the plaintext and a corresponding byte from the key, repeating the key as needed. The result is then encoded into a Base64 string for readability and easy storage.

**Decryption:**



Text decryption reverses the encryption by Base64-decoding the ciphertext and applying the same XOR operation between each byte of the encrypted data and the repeating key. This restores the original plaintext, assuming the same key used for encryption is provided.

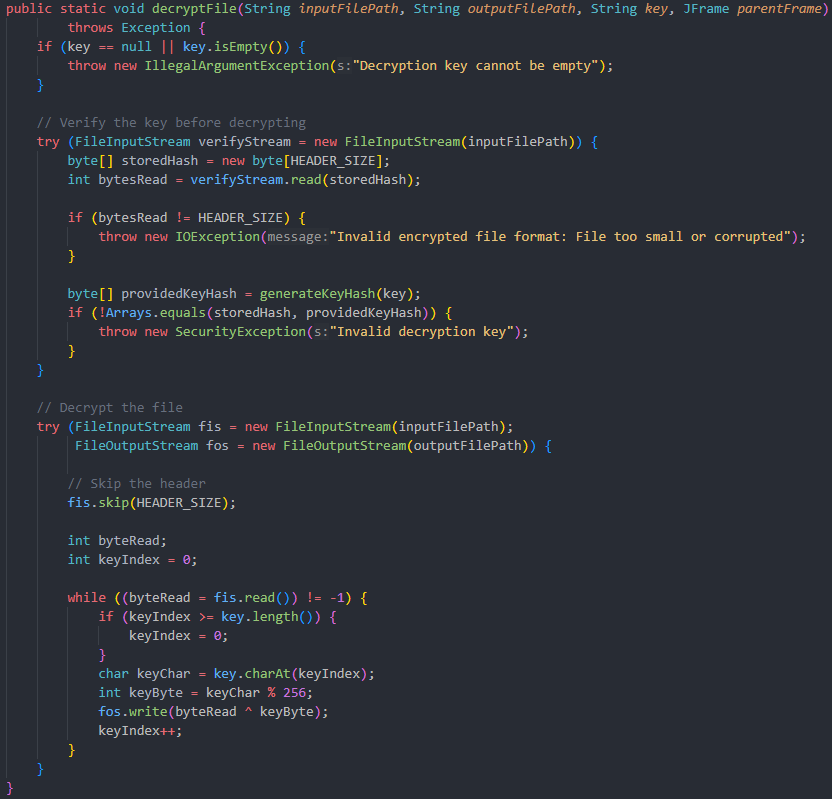
### Vernam File Cipher

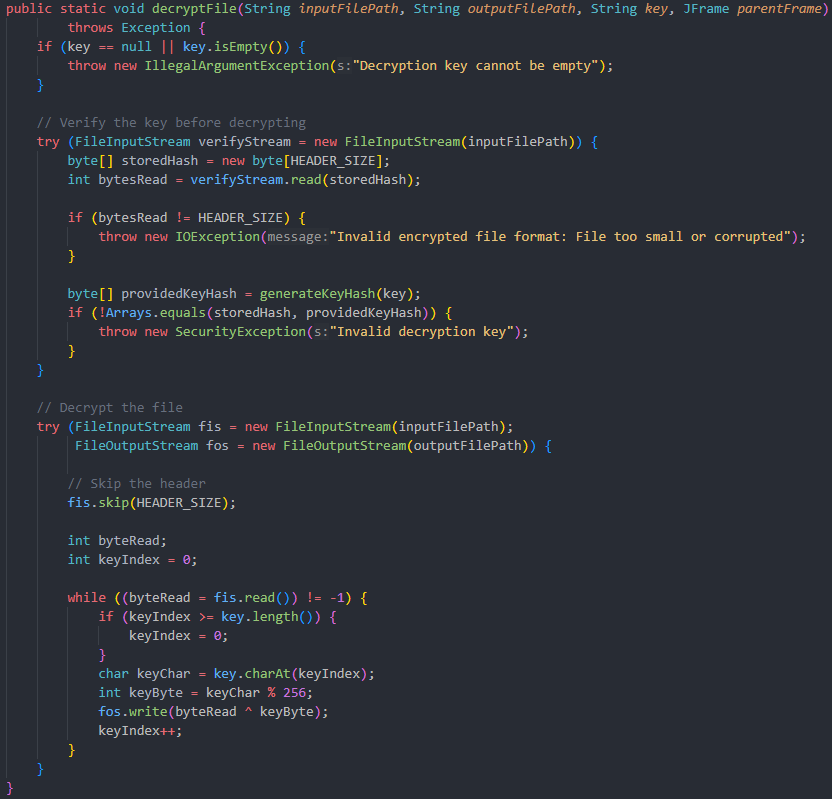
**Encryption:**



File encryption with the Vernam cipher operates similarly to text encryption but works on the file's bytes, writing a verification hash at the beginning of the file and applying the XOR operation between the file's bytes and the key's bytes. The encrypted file is then saved with this verification hash to ensure the correct decryption key is used later.

**Decryption:**



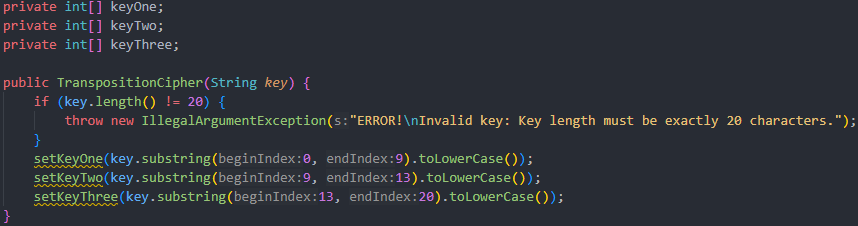


File decryption starts by reading and verifying the 16-byte hash header to ensure the correct key is used. After verification, the decryption process skips the header and XORs each byte of the encrypted content with the repeating key to restore the original file data.

## Columnar Transposition Cipher

### Overview

The columnar transposition cipher is a grid-based algorithm. By making use of a grid that is the size of the key’s length per iteration, the data is shuffled with a pattern based on the key provided. Initially, the data is populated into the grid from left-to-right, top-to-down. This algorithm is repeated three times for increased security implications. The key is the main component that drives how the pattern is determined. The key must be 20 characters long, and it is divided into 3 keys consisting of the first 9-characters, followed by the next 4-characters, and finally the last 7-characters. The first 9-character key is used for the first iteration, followed by the second iteration using the 3-character key, and finally followed by the last iteration using the 7-character key. For each iteration, the previous iteration’s ciphertext is used to for the next iteration. See screenshots below for code reference.



The pattern determined using the key is identified using the order as which the letters of the key appear in the alphabet. For example, the word “plane” would produce the pattern:

[4, 2, 0, 3, 1]. Using this order, the data that is placed into the grid is extracted by columns in the order of the pattern the key creates. This concludes the encryption of the algorithm.

For the decryption, the process is reversed. The key is divided into the keys, and the key is used from the right-to-left. Starting with the 7-character key to decrypt the ciphertext, followed by the 4-character key to decrypt in the next iteration, and finally the 9-character key is used in the third and last iteration to convert the ciphertext to plaintext. All texts and files are converted to bytes followed by the same procedure.

### Columnar Transposition Text Cipher

The columnar transposition cipher converts the text into bytes and proceeds with the procedures of encrypting and decryption.

**Encryption:**

A screen shot of a computer program

AI-generated content may be incorrect.

The 3 iterations of encryption are demonstrated in the encryptText() method, as seen in the screenshot above. Note that the encrypt() methods take two parameters, the byte array to be encrypted and the key to be used, and each iteration uses the previous iteration’s ciphertext to encrypt for its own encryption iteration. The screenshot below shows the called method, encrypt().

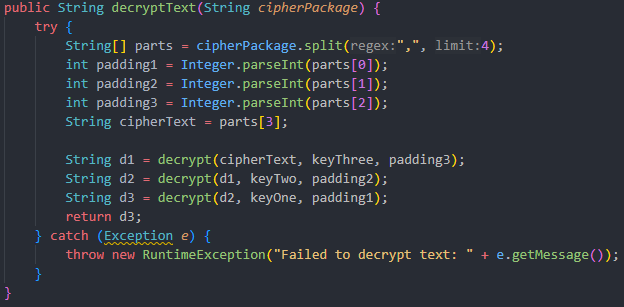


A computer screen shot of a program code

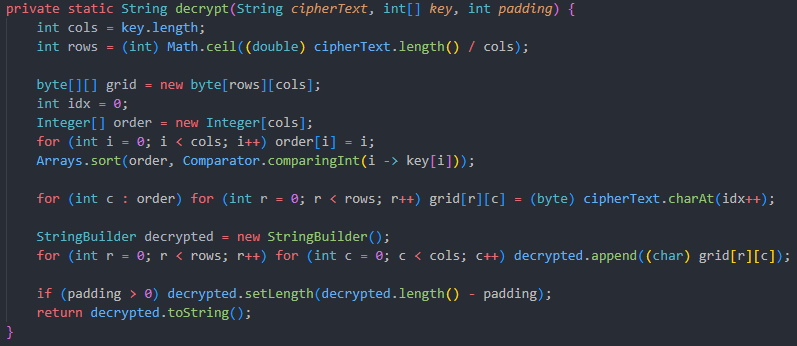
AI-generated content may be incorrect.

This code populates the grid that has the length of the key and adds padding of ‘X’ if not all grid blocks are filled, which occurs after all the data has been populated into the grid. Followed is the collection of the columns’ values, based on the order determined from the key, and added into a string that can be used for the next iteration.

**Decryption:**

****

The 3 iterations of decryption are demonstrated in the decryptText() method, as seen in the screenshot above. Note that the decrypt() methods take three parameters, the byte array to be encrypted, the key to be used, and the number of paddings added to fill the grid, and each iteration uses the previous iteration’s ciphertext to decrypt for its own decryption iteration. The screenshot below shows the called method, decrypt().

****

This decrypt() method reverses the order used for encryption and populated the grid using the order determined. Hence, the order is added to a string in a right-to-left, down-to-up order. Finally, the padding is removed. This will reverse engineer the encryption process.

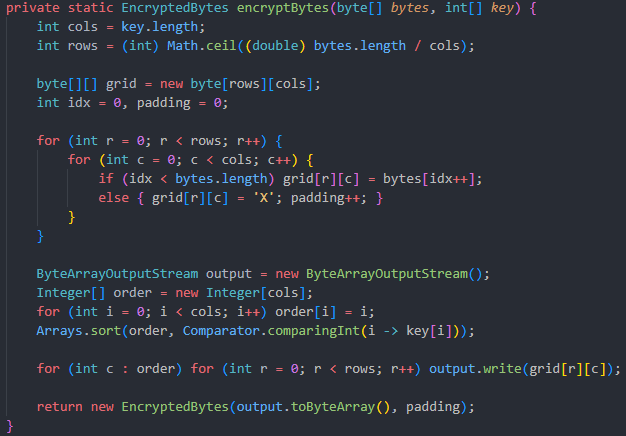
### Columnar Transposition File Cipher

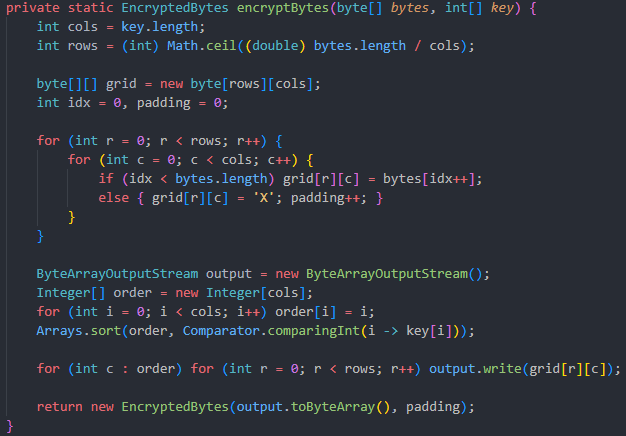
The columnar transposition cipher converts the file into bytes and proceeds with the procedures of encrypting and decryption.

**Encryption:**

****

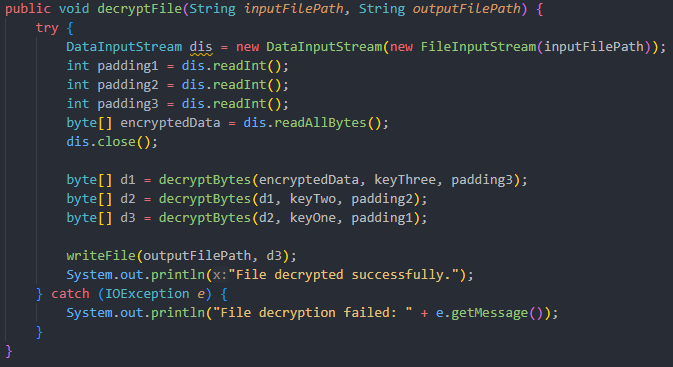
The 3 iterations of encryption are demonstrated in the encryptFile() method, as seen in the screenshot above. Note that the encryptBytes() methods take two parameters, the byte array to be encrypted and the key to be used, and each iteration uses the previous iteration’s ciphertext to encrypt for its own encryption iteration. The screenshot below shows the called method, encryptBytes().

****

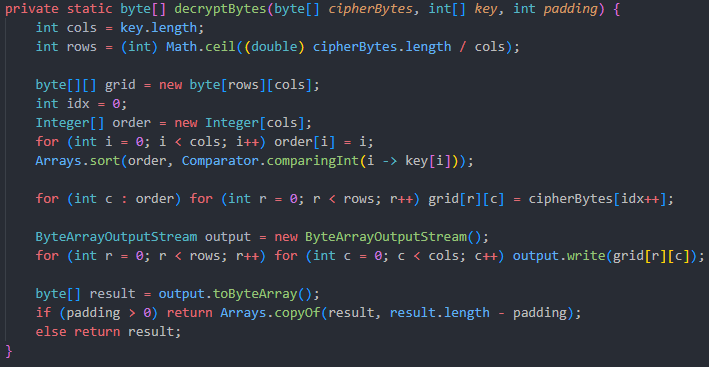
****

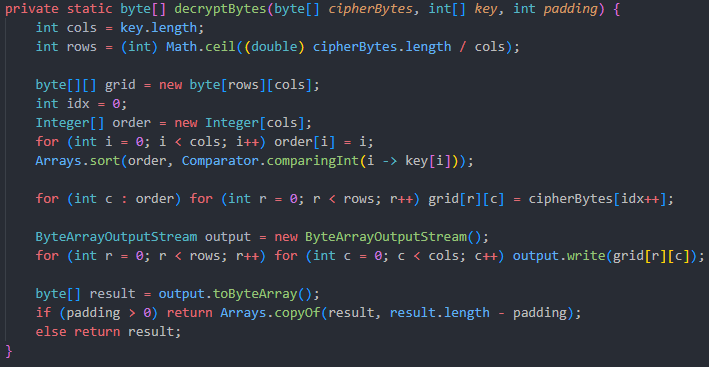
The encyptBytes() method populates the grid that has the length of the key and adds padding of ‘X’ if not all grid blocks are filled, which occurs after all the data has been populated into the grid. Followed is the collection of the columns’ values, based on the order determined from the key, and added into a string that can be used for the next iteration.

**Decryption:**

****

The 3 iterations of decryption are demonstrated in the decryptFile() method, as seen in the screenshot above. The decryptFile() method has two parameters that determine the path of the encrypted and decryption file. Note that the decryptBytes() methods take three parameters, the byte array to be encrypted, the key to be used, and the number of paddings added to fill the grid, and each iteration uses the previous iteration’s ciphertext to decrypt for its own decryption iteration. The screenshot below shows the called method, decrypt().

****

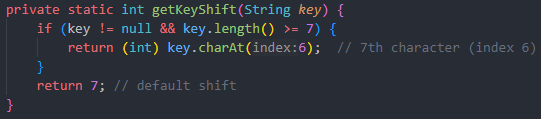
****

This decryptBytes() method reverses the order used for encryption and populated the grid using the order determined. Hence, the order is added to a byte array in a right-to-left, down-to-up order. Finally, the padding is removed. This will reverse engineer the encryption process.

## Custom Cipher

### Overview

The Custom Secure Cipher (CSC) cipher is a unique algorithm that uses byte transformations and a grid similar to the columnar transposition algorithm. The byte transformations are driven by the key value. Specifically, the seventh character of the key. The rest of the key is disposed and not used for any encryption or decryption process. This algorithm uses an unanticipated approach of utilising the key for encryption and decryption. The screenshot below demonstrates the process.



If the key is not long enough to have a seventh character to utilise, the default value of 7 is returned. The seventh character is converted to its ASCII value, and the data that is converted to bytes is shifted that many times and the usage of XOR operation with 0x5A. Note that the seventh character needs to be a number.

After this process, the cipher proceeds to placing the bytes into a grid that is fixed to 12 columns, from left-to-right, top-to-bottom. After which, the bytes in the grid are extracted diagonally, from bottom-left to top-right. Finally, the encrypted bytes are encoded in Base64 for safe storage. This concludes the encryption process. This process is reverse engineered to obtain the plaintext. The bytes are placed in a grid from top-right to bottom-left, followed by a shift of bytes determined by the seventh character of the key converted to its ASCII value.

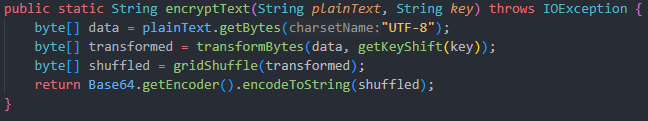
The screen shot below demonstrates the grid shuffle.



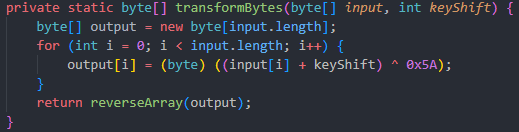
### Custom Text Cipher

The custom cipher converts the text into bytes and proceeds with the procedures of encrypting and decryption.

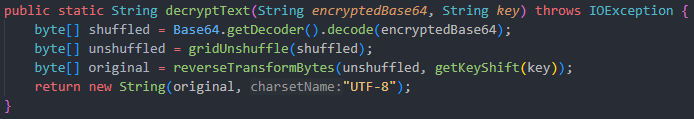
**Encryption:**

****

The encryptText() method obtains the bytes value of the input text and calls it in transformBytes() method. The transformBytes() method is displayed below in a screenshot.

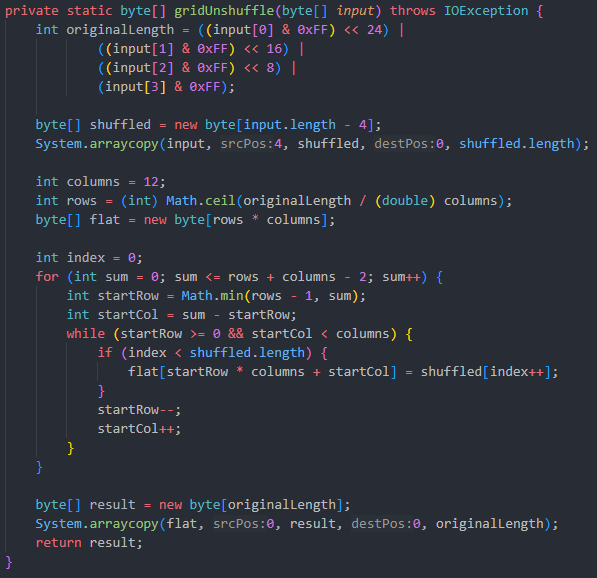


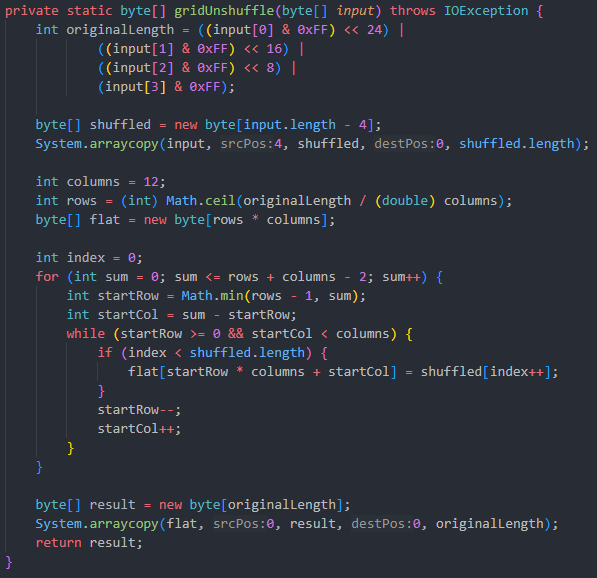
**Decryption:**

****

The decryption process is the reverse of the encryption, with all methods reverse engineering the procedures to obtain the plaintext. The decryptText() method takes the ciphertext and it is Base64-decoded. Then the gridUnshuffle() method is called, and the result of that is reversed with the ASCII value obtained from the key to result in the plaintext.

The gridUnshuffle() and reverseTransformBytes() are demonstrated below, respectively.





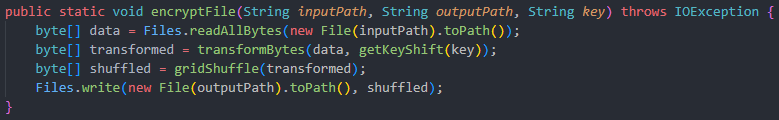
A screen shot of a computer code

AI-generated content may be incorrect.

### Custom File Cipher

The custom cipher converts the file into bytes and proceeds with the procedures of encrypting and decryption.

**Encryption:**



The encryptFile() method obtains the bytes value of the input file and calls it in transformBytes() method, followed by the gridShuffle() method. Similar to text encryption.

The ciphertext-file is then created at the path of the user’s choosing.

**Decryption:**

A screen shot of a computer code

AI-generated content may be incorrect.

The decryption process is similar to the decryption process of text. The decryptFile() methods ends with creating the plaintext-file in the path of the user’s choosing.

## Vernam + Transposition Cipher

### Overview

The Vernam and Transposition cipher is combination of 2 classic algorithms, the Vernam cipher that uses XOR operation, and Columnar Transposition cipher that uses a grid to shuffle values. The combination of these two algorithms results in a stronger algorithm than the usage of either algorithm only.

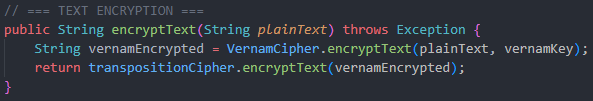
The complete code of the working Vernam and Transposition ciphers are imported into this cipher’s file, and the methods of and processes that Vernam and Transposition use are used sequentially. The integration of these two algorithms is implemented with ease as they both use bytes for texts and files, in this application.

The key utility of this cipher is inherited from the way the Vernam and Transposition cipher utilise the key. See [Vernam Cipher](#_Vernam_Cipher) and [Columnar Transposition Cipher](#_Columnar_Transposition_Cipher). The key provided is used for both encryption and decryption procedures.

### Vernam + Transposition Text Cipher

The combined cipher converts the text into bytes and proceeds with the procedures of encrypting and decryption by calling the text-related encryption and decryption methods of the Vernam and Transposition cipher. See screenshots below of how these methods are used.

**Encryption:**



**Decryption:**

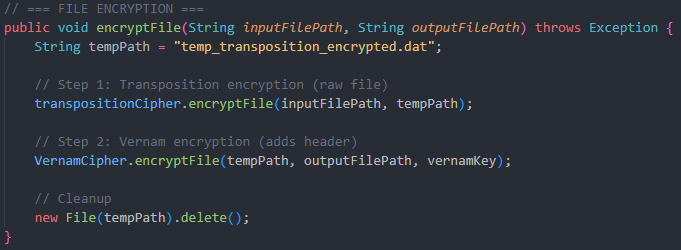
A screen shot of a computer code

AI-generated content may be incorrect.

### Vernam + Transposition File Cipher

The combined cipher converts the file into bytes and proceeds with the procedures of encrypting and decryption by calling the file-related encryption and decryption methods of the Vernam and Transposition cipher. See screenshots below of how these methods are used.

**Encryption:**



**Decryption:**

A screen shot of a computer program

AI-generated content may be incorrect.

# Security Considerations

The cryptography algorithms in this application are for developed strictly for educational purposes. As good as they are, we do not recommend users implementing these algorithms for high-security systems and productions use.

**Terms and Conditions apply:**

By using this application, you agree encrypt and decrypt responsibly and at your own risk. No responsibility will be taken be the developers for any loss of data. For legitimate security, please consult professional cryptographers.