ITRI615 – Encryption Project

Project Documentation

31715982 - Daniel Coetzee

30949955 – Lesetja Mojapelo

Table of Contents

Section 1

Background

1. Introduction

Welcome! Thank you for Crypto (random name) for your encryption needs.

2. Background of application

Crypto is an encryption application that addresses the Vernam, Vigenère and Transposition encryption methods. Additionally, crypto has an added Custom encryption. Crypto can also decrypt the encrypted ciphers.

3. Purpose of the application

Crypto is designed to provide ciphertext for any given plaintext or file format. Depending on the chosen encryption methods, Crypto will either generate or require an encryption key which will be used to encrypt the provided plaintext. Crypto can decrypt the ciphertext or file format back into its relevant plaintext or format.

4. Encryption methods background

4.1. Vernam

The Vernam Cipher is an encryption cipher that uses a one-time pad, this means that for every round of encryptions, a new key is randomly generated for encrypting the plaintext. This cipher uses positional values for alphabet letters and adds them to the key after which they are XOR'd.

4.1.1. Implementation and algorithm

The Vernam encryption takes in plaintext or a file and for that plaintext, a key is randomly generated for it to produce ciphertext.

Plaintext:

a) For each letter in the plaintext, generate a random byte value (0 – 255) and convert that value into its relevant ASCII value, this is the encryption key.

b) XOR the random byte value to the ASCII value's ordinal representation and add it to the cipher text.
c) Decrypting takes pairs a ciphertext letter with an encryption key letter.
d) XOR the cipher text encryption key pair and get their character value and return

Files follow a similar format, but bytes are used instead of characters.

4.2. Vigenère

The Vigenère Cipher was originating from Giovan Bellaso in 1553 but was associated with Blaise de Vigenère. This encryption works by overlaying a series of ciphers, which are combined to create a Vigenère Table that depends on the letters of a particular keyword. It is a polyalphabetic substitution cipher.

4.2.1. Implementation and algorithm

The Vigenère encryption works by taking a user keyword and matching the letters of the keyword to the letter in the plaintext in a recursive manner. This is done for both plaintext and files.

The Vigenère algorithm works by using the calculation from ASCII values of the letters/bytes to apply the mechanism of the Vigenère Table. The encryption adds these values through a modulo calculation that is based on the encryption keys length and decrypts through subtracting these values, this is also based on the encryption keys length. The algorithm uses a custom alphabet so that users can enter numbers and symbols as well. Encryption occurs as follows:

a) From the plaintext or file, get the index of the concerned letter or symbol.
b) From the encryption key, get the index of the concerning letter after applying a length modulo to the key.
c) If encrypting, add the letter index and key index and modulo the custom alphabet length.
d) If decrypting, subtract the letter and key index and modulo the custom alphabet length
e) For either case, add the text or file value to the ciphertext and return the ciphertext when complete.

4.3. Transposition

The transposition cipher encrypts plaintext by shifting characters according to a designated formation, this essentially reorders text.

4.3.1. Implementation and algorithm

Functions were created to handle text and file encryption and decryption. Within the encryption method:

a) the plaintext is converted to a matrix where each character is a value in the matrix.
b) The encryption key inputted by the user is used to determine the number of columns each row must contain. If the plaintext-matrix conversion leaves a last row with fewer columns than the inputted encryption key value, the final row is padded with space characters until the last row has the correct length.
c) The matrix is then transposed, and the cipher text is created by traversing the transposed matrix row by row.
d) The decryption function will convert the cipher text into a matrix once more and transpose it once more to obtain the plaintext.

4.4. Custom

4.4.1. Implementation and algorithm

This custom-made algorithm takes each character in plaintext and converts it to its numerical ASCII value. In this method:

a) For each character of the plaintext, a random integer is generated between the values of 1 and 256. These 2 numerical values are then multiplied together to gain the raw encrypted numerical value.
b) This value is then put through mod 128 and converted back to an ASCII character to gain the corresponding ciphertext character.
c) The raw encrypted values and the randomly generated numerical key are then sent to the decryption function.
d) These lists are then traversed, with each iteration of this traversal, the values at the current index are divided to reach the plaintext numerical ASCII value thus decrypting the ciphertext.

Section 2

Installation and Setup

2.1. Project files

The project files can be found on the following GitHub link:

https://github.com/DeradoZA/ITRI615-Encryption-Project.git

To get the code, follow these steps:

1. Ensure Git is installed. If you don't have Git installed, you can download it here https://git-scm.com/
2. Create an empty directory on your machine where the code will be copied to.
3. In that directory, run the following command on the terminal:

```
git clone https://github.com/DeradoZA/ITRI615-Encryption-Project.git
```

2.2. Packages

2.2.1. Python Package Manager (pip)

Since Python was used to create the application programming interface (API) is important to ensure that pip is installed.

To check for pip, do the following:

```
pip --version
```

That should display the version of your pip package manager should you have pip installed. If that is not the case then go to https://www.python.org/ to download Python as the download is packaged with pip by default.

2.2.2. Node Package Manager (npm)

The front end of the application was made using React. Before React can be used, node.js should be installed. To download node.js, follow this link: https://nodejs.org/en/download

Follow the installation instructions and on the terminal enter:
```
npm --version
```

The prompt will indicate the version of the node that is installed on your machine.

2.2.3. React

React is the primary framework for development. This framework is based on JavaScript XML. The documentation for this can be found at: https://react.dev/learn

Section 3

Programming of artefact

3.1. Development tools

3.1.1. Operating system

The primary system for development was Windows 10 and 11. This is where testing and debugging took place.

3.1.2. IDE

For coding, Integrated Development Environments were used, particularly Visual Studio Code (VSCode. To install VSCode, follow this link: https://code.visualstudio.com/

3.1.3. Database Management

No Database Management tools were used for the development of the project as all program instances are stored locally.

3.1.4. Hosting

Due to financial constraints, the project was run locally on the localhost server through port 3000. In full it is, http://localhost:3000

3.2. Prerequisites

3.2.1. CryptSite package requirements

Before the application can be run, some packages need to be installed to ensure functionality. On the terminal, navigate to the crypt-site directory and run npm install. This installs the following dependencies found in the application package.json file.

```
"dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "bootstrap": "^5.2.3",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.11.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  }
```

3.3. Programming

3.3.1. Vernam

```
import random
import os
import base64


class VernamMethods:
    def __init__(self, text=None, file=None):
        self.text = text
        self.file = file

    def textEncrypt(self, text):
        vernamKey = ""
        cipherText = ""

        for letter in text:
            keyByteValue = random.randint(0,255)
            keyChar = chr(keyByteValue)
            vernamKey = vernamKey + keyChar

            cipherText += chr(ord(letter) ^ keyByteValue)

        return cipherText, vernamKey

    def textDecrypt(self, text, vernamKey):
        decryptedText = ""

        for textLetter, vernamLetter in zip(text, vernamKey):
            decryptedByte = ord(textLetter) ^ ord(vernamLetter)

            decryptedText += chr(decryptedByte)

        return decryptedText

    def fileEncrypt(self, fileBytes):
        cipherBytes = b""
        vernamKeyBytes = b""

        for byte in fileBytes:
            keyByteValue = random.randint(0, 255)
            keyByte = bytes([keyByteValue])

            vernamKeyBytes = vernamKeyBytes + keyByte
            cipherBytes = cipherBytes + bytes([byte ^ keyByteValue])

        return cipherBytes, vernamKeyBytes

    def fileDecrypt(self, fileBytes, vernamKey):

        decryptedBytes = b""

        print(len(fileBytes))

        for byte, vernamByte in zip(fileBytes, vernamKey):

            decryptedByte = byte ^ vernamByte

            decryptedBytes = decryptedBytes + bytes([decryptedByte])

        return decryptedBytes

    def createEncryptedFile(self, cipherBytes, file):
        fileInfo = os.path.splitext(file)
        encryptedFileName = fileInfo[0] + " - E" + fileInfo[1]
        encryptedFile = open(encryptedFileName, "wb")
        encryptedFile.write(cipherBytes)

        encryptedFile.close()

    def createDecryptedFile(self, plainBytes, file):
        fileInfo = os.path.splitext(file)
        decryptedFileName = fileInfo[0] + " - D" + fileInfo[1]
        decryptedFile = open(decryptedFileName, "wb")
        decryptedFile.write(plainBytes)

        decryptedFile.close()
```

`VernamMethods` class:

`textEncrypt(self, text)`: This method encrypts a given text using the Vernam cipher method (also known as a one-time pad). It generates a random key of the same length as the text and applies the XOR operation to each character of the text with the corresponding character in the key. The result is an encrypted string, and it returns both this string and the key used for encryption.

`textDecrypt(self, text, vernamKey)`: This method decrypts the given text (which should be in encrypted form) using the provided Vernam key. It applies the XOR operation again to each character of the encrypted text with the corresponding character in the key, effectively reversing the encryption operation.

`fileEncrypt(self, fileBytes)`: This method is similar to `textEncrypt`, but it operates on bytes read from a file instead of text. It generates a random byte key, applies the XOR operation to each byte in the file with the corresponding byte in the key, and returns the encrypted bytes and the key.

`fileDecrypt(self, fileBytes, vernamKey)`: This method is the file equivalent of `textDecrypt`. It applies the XOR operation to each byte in the encrypted file with the corresponding byte in the key to decrypt the file.

`createEncryptedFile(self, cipherBytes, file)`: This method takes encrypted bytes and the original file name, and creates a new file with the same name, appended with " - E" to indicate it's encrypted. It then writes the encrypted bytes to this new file.

`createDecryptedFile(self, plainBytes, file)`: This method is similar to the `createEncryptedFile` method, but creates a decrypted file instead. It takes the decrypted bytes and the original file name, appends " - D" to the file name to indicate it's decrypted, and writes the decrypted bytes to this new file.

3.3.2. Vigenère

```python
import os

class Vigenere:
    def vigenere(
        plaintext,
        key,
        encrypt=True
    ):

        letters = "abcdefghijklmnopqrstuvwxyz"
        letters += "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        letters += "1234567890"
        letters += " !@#$%^&*()-_+=`~;:'[]{}|<>,./?"
        letters += "\"\\"
        ciphertext = ''
        plaintext_value = 0

        for i in range(len(plaintext)):
            letter_index = letters.index(plaintext[i])
            key_index = letters.index(key[i % len(key)])

            if encrypt:
                plaintext_value = (letter_index + key_index) % len(letters)

            if not encrypt:
                plaintext_value = (letter_index - key_index) % len(letters)

            ciphertext += letters[plaintext_value]

        return ciphertext

    def textEncrypt(plaintext, key):
        return Vigenere.vigenere(plaintext, key, True)

    def textDecrypt(ciphertext, key):
        return Vigenere.vigenere(ciphertext, key, False)

    def fileToBytes(file):
        readFile = open(file, 'rb')
        file_bytes = readFile.read()
        readFile.close()

        return file_bytes

    def fileEncrypt(plaintext, key):
        ciphertext = Vigenere.vigenere(plaintext, key, True)
        return ciphertext

    def fileDecrypt(ciphertext, key):
        return Vigenere.vigenere(ciphertext, key, False)

    def createEncryptedFile(ciphertext, file):
        fileInfo = os.path.splitext(file)
        encryptedFileName = "Encrypted" + fileInfo[1]
        encryptedFile = open(encryptedFileName, "wb")
        encryptedFile.write(ciphertext)

        encryptedFile.close()

    def createDecryptedFile(plainBytes, file):
        fileInfo = os.path.splitext(file)
        decryptedFileName = fileInfo[0] + " - D" + fileInfo[1]
        decryptedFile = open(decryptedFileName, "wb")
        decryptedFile.write(plainBytes)

        decryptedFile.close()
```

`Vigenere` class:

`vigenere(plaintext, key, encrypt=True)`: This is the core method that implements the Vigenère cipher algorithm. It takes a `plaintext` string (which can also be ciphertext in case of decryption), a `key` string, and a boolean `encrypt`. The method operates on a defined character set `letters`. For each character in the `plaintext`, it shifts it within `letters` by the position of the corresponding character in `key` (considering wrap-around in the `key`), forward for encryption and backward for decryption.

`textEncrypt(plaintext, key)`: This method simply calls `vigenere()` with `encrypt=True` to encrypt a text string using the Vigenère cipher.

`textDecrypt(ciphertext, key)`: This method calls `vigenere()` with `encrypt=False` to decrypt a text string that was encrypted with the Vigenère cipher.

`fileToBytes(file)`: This method opens a file in binary mode, reads all its bytes into memory, and then closes the file. It returns the read bytes.

`fileEncrypt(plaintext, key)`: This method calls `vigenere()` with `encrypt=True` to encrypt file content (expected to be a string of bytes) using the Vigenère cipher.

`fileDecrypt(ciphertext, key)`: This method calls `vigenere()` with `encrypt=False` to decrypt file content that was encrypted with the Vigenère cipher.

`createEncryptedFile(ciphertext, file)`: This method takes the ciphertext bytes and the original file name, and creates a new file with "Encrypted" appended before the extension to indicate that it contains encrypted data. It then writes the ciphertext bytes into this new file.

`createDecryptedFile(plainBytes, file)`: This method takes decrypted bytes and the original file name, appends " - D" to the name to indicate it contains decrypted data, and writes the decrypted bytes into a new file.

3.3.3. Transposition

```
import os

class TranspositionMethods:
    def __init__(self, text = None, file = None):
        self.file = file
        self.text = text


    def TextEncrypt(self, text, rowLength):
        textMatrix = []
        Row = []
        cipherText = ""

        for character in text:
            Row.append(character)
            if len(Row) == rowLength:
                textMatrix.append(Row)
                Row = []

        if len(Row) == rowLength:
            textMatrix.append(Row)
        elif len(Row) == 0:
            pass
        else:
            paddingValue = rowLength - len(Row)
            for _ in range(paddingValue):
                Row.append(" ")
            textMatrix.append(Row)

        for index in range(rowLength):
            for row in textMatrix:
                cipherText += row[index]

        return cipherText


    def TextDecrypt(self, cipherText, rowLength):
        textMatrix = []
        Row = []
        plainText = ""

        for character in cipherText:
            Row.append(character)
            if len(Row) == len(cipherText) // rowLength + (1 if len(cipherText) %
rowLength > 0 else 0):
                textMatrix.append(Row)
                Row = []

        for index in range(len(textMatrix[0])):
            for row in textMatrix:
                if index < len(row):
                    plainText += row[index]

        return plainText
```

```
def FileEncrypt(self, fileBytes, rowLength):
        byteMatrix = []
        Row = []
        cipherBytes = b""

        for byte in fileBytes:
            Row.append(byte)
            if len(Row) == rowLength:
                byteMatrix.append(Row)
                Row = []

        if len(Row) == rowLength:
            byteMatrix.append(Row)
        elif len(Row) == 0:
            pass
        else:
            paddingValue = rowLength - len(Row)
            for _ in range(paddingValue):
                Row.append(" ")
            byteMatrix.append(Row)

        for index in range(rowLength):
            for row in byteMatrix:
                cipherBytes += bytes([row[index]])

        return cipherBytes

    def FileDecrypt(self, cipherBytes, rowLength):
        byteMatrix = []
        Row = []
        plainBytes = b""

        for byte in cipherBytes:
            Row.append(byte)
            if len(Row) == len(cipherBytes) // rowLength + (1 if len(cipherBytes) % rowLength
> 0 else 0):
                byteMatrix.append(Row)
                Row = []

        for index in range(len(byteMatrix[0])):
            for row in byteMatrix:
                if index < len(row):
                    plainBytes += bytes([row[index]])

        return plainBytes

    def createEncryptedFile(self, cipherBytes, file):
        fileInfo = os.path.splitext(file)
        encryptedFileName = fileInfo[0] + " - E" + fileInfo[1]
        encryptedFile = open(encryptedFileName, "wb")
        encryptedFile.write(cipherBytes)

        encryptedFile.close()

    def createDecryptedFile(self, plainBytes, file):
        fileInfo = os.path.splitext(file)
        decryptedFileName = fileInfo[0] + " - D" + fileInfo[1]
        decryptedFile = open(decryptedFileName, "wb")
        decryptedFile.write(plainBytes)

        decryptedFile.close()
```

`TranspositionMethods` class:

`TextEncrypt(self, text, rowLength)`: This method performs a transposition cipher encryption on the input text. It constructs a 2D matrix (list of lists) of characters. If the text does not perfectly fit into the matrix, it pads the last row with spaces. It then reads off the characters column by column to form the encrypted text.

`TextDecrypt(self, cipherText, rowLength)`: This method performs the reverse of `TextEncrypt`. It constructs a 2D matrix. It then reads off the characters row by row to decrypt the text.

`FileEncrypt(self, fileBytes, rowLength)`: This method is similar to `TextEncrypt`, but works with file bytes instead of text. It creates a 2D byte matrix. If the file bytes do not perfectly fit into the matrix, it pads the last row with spaces (as byte values). It then reads off the bytes column by column to create the encrypted bytes.

`FileDecrypt(self, cipherBytes, rowLength)`: This method performs the reverse of `FileEncrypt`. It creates a 2D byte matrix. It then reads off the bytes row by row to decrypt the file.

`createEncryptedFile(self, cipherBytes, file)`: This method takes encrypted bytes and the original file name, and creates a new file with the same name, appended with " - E" to indicate it's encrypted. It then writes the encrypted bytes to this new file.

`createDecryptedFile(self, plainBytes, file)`: This method is similar to the `createEncryptedFile` method, but creates a decrypted file instead. It takes the decrypted bytes and the original file name, appends " - D" to the file name to indicate it's decrypted, and writes the decrypted bytes to this new file.

3.3.4. Custom

```python
import os
import random

class CustomAlgoMethods:
    def __init__(self, text=None, file=None):
        self.text = text
        self.file = file

    def TextEncrypt(self, text):
        customDecKey = []
        rawEncryptedDecs = []
        encryptedBytes = b""
        textToBytes = text.encode('ascii')

        for byte in textToBytes:
            randomKeyDec = random.randint(1, 256)
            customDecKey.append(randomKeyDec)

            encryptedRawDecimalValue = byte * randomKeyDec
            encryptedDecimalValue = encryptedRawDecimalValue % 128
            encryptedByteValue = encryptedDecimalValue.to_bytes(1, byteorder='big')
            rawEncryptedDecs.append(encryptedRawDecimalValue)

            encryptedBytes += encryptedByteValue

        cipherText = encryptedBytes.decode('ascii')

        return cipherText, encryptedBytes, rawEncryptedDecs, customDecKey

    def TextDecrypt(self, encryptedRawDecs, customDecKey):
        decryptedBytes = b""

        for index in range(0, len(encryptedRawDecs)):
            decryptedByteDec = int(encryptedRawDecs[index] / customDecKey[index])
            decryptedBytes += decryptedByteDec.to_bytes(1, byteorder='big')

        plainText = decryptedBytes.decode('ascii')

        return plainText

    def FileEncrypt(self, fileBytes):

        customDecKey = []
        rawEncryptedDecs = []
        cipherBytes = b""

        for byte in fileBytes:
            randomKeyDec = random.randint(1, 256)
            customDecKey.append(randomKeyDec)

            encryptedRawDecimalValue = byte * randomKeyDec
            encryptedDecimalValue = encryptedRawDecimalValue % 256
            encryptedByteValue = encryptedDecimalValue.to_bytes(1, byteorder='big')
            rawEncryptedDecs.append(encryptedRawDecimalValue)

            cipherBytes += encryptedByteValue

        return cipherBytes, rawEncryptedDecs, customDecKey

    def FileDecrypt(self, rawEncryptedDecs, customKey):
        decryptedBytes = b""

        for index in range(0, len(rawEncryptedDecs)):
            decryptedByteDec = int(rawEncryptedDecs[index] / customKey[index])
            decryptedBytes += decryptedByteDec.to_bytes(1, byteorder='big')

        return decryptedBytes
```

```
def createEncryptedFile(self, cipherBytes, file):
        fileInfo = os.path.splitext(file)
        encryptedFileName = fileInfo[0] + " - E" + fileInfo[1]
        encryptedFile = open(encryptedFileName, "wb")
        encryptedFile.write(cipherBytes)

        encryptedFile.close()

    def createDecryptedFile(self, plainBytes, file):
        fileInfo = os.path.splitext(file)
        decryptedFileName = fileInfo[0] + " - D" + fileInfo[1]
        decryptedFile = open(decryptedFileName, "wb")
        decryptedFile.write(plainBytes)

        decryptedFile.close()
```

`CustomAlgoMethods` class:

`TextEncrypt(self, text)`: This method takes a text string as an input, and encrypts it using a custom algorithm. This method generates a list of random keys, and then multiplies each ASCII byte value of the text by a corresponding random key. Then it takes the modulo 128 of this product to make sure the result still fits into an ASCII byte. The final encrypted text, the raw encrypted bytes, the list of raw encrypted decimal values, and the list of generated keys are returned.

`TextDecrypt(self, encryptedRawDecs, customDecKey)`: This method takes a list of raw encrypted decimal values and the corresponding list of keys used for encryption, and decrypts the text. It does this by dividing each raw encrypted decimal value by its corresponding key, and converts these back into ASCII byte values, which it then decodes into plain text.

`FileEncrypt(self, fileBytes)`: This method is similar to the `TextEncrypt` method, but instead takes raw file bytes as input. The process is similar: it generates random keys, encrypts the file bytes using these keys, and then returns the encrypted bytes, raw encrypted decimal values, and the keys.

`FileDecrypt(self, rawEncryptedDecs, customKey)`: This method is similar to the `TextDecrypt` method, but works with raw encrypted decimal values from a file. It takes these values and the list of keys used for encryption and decrypts the file bytes.

`createEncryptedFile(self, cipherBytes, file)`: This method takes encrypted bytes and the original file name, and creates a new file with the same name, appended with " - E" to indicate it's encrypted. It then writes the encrypted bytes to this new file.

`createDecryptedFile(self, plainBytes, file)`: This method is similar to the `createEncryptedFile` method, but creates a decrypted file instead. It takes the decrypted bytes and the original file name, appends " - D" to the file name to indicate it's decrypted, and writes the decrypted bytes to this new file.

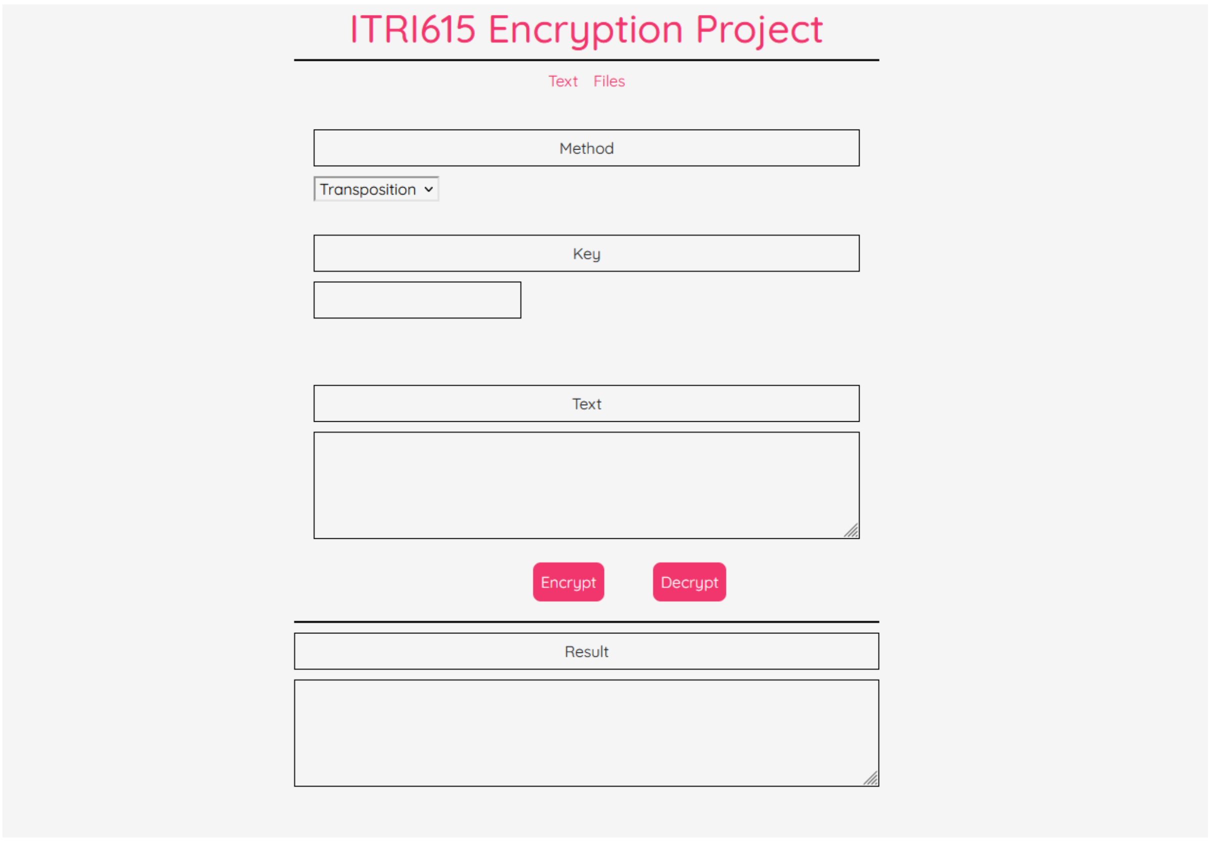
Section 4

User Manual

4.1. User Manual

This section will provide a guide to the user regarding how to make use of the ITRI615 Encryption Project website. The website contains functionality for text encryption/decryption and file encryption/decryption with these functionalities being split into different pages on the website.

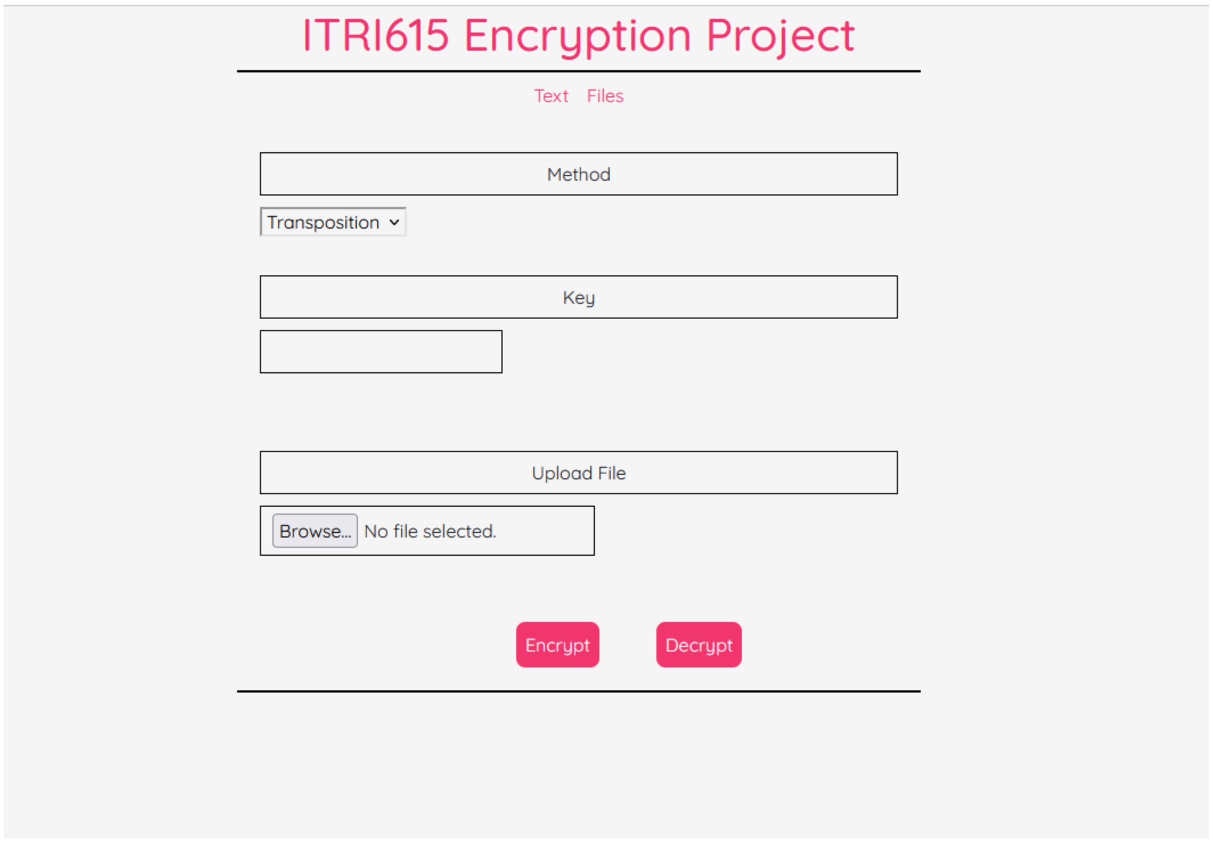
4.2. Text Encryption & Decryption



The website offers text encryption and decryption services making use of 4 different cryptographic algorithms which can be selected by the user. Text can be encrypted or decrypted using the following steps:

1. Select an algorithm to make use of from the method drop-down menu.
2. Select a key to use for encryption or decryption. If the method chosen does not take a key from the user, this input box will show the value "None".
3. Enter the text to be encrypted or decrypted within the input box under the text heading.
4. Click on the encrypt button to encrypt the text. If the user wants to decrypt text, they must provide the cipher text within the text input box and click on the decrypt button.

4.3. File encryption & decryption



The website also offers file encryption and decryption services using these same 4 cryptographic algorithms. A user can input any file format and receive encrypted and decrypted versions of these files. The steps to make use of this service are as follows:

1. Select an algorithm to make use of from the method drop-down menu.
2. Select a key to use for encryption or decryption. If the method chosen does not take a key from the user, this input box will show the value “None”.
3. Click on the browse button and select the file you would like to encrypt or decrypt.
4. The user can then click on the Encrypt button which will download an encrypted version of the file uploaded onto the user’s device.
5. If the user wants to decrypt this encrypted file, the encrypted file must be uploaded using the same Browse button. The user then must click the Decrypt button which will download the decrypted version of the encrypted file.