

CS203 PROJECT

Encryption-Decryption Using Matrix Multiplication



[source](#)

Under the Guidance of Dr. Neeraj Goel,

Group Members:

Ankit Sharma : 2020CSB1072

Arshdeep Singh : 2020CSB1074

INTRODUCTION

Cryptography: need of the hour

In our day-to-day lives, the use of cryptography is everywhere. Bank servers and email clients save your passwords using cryptography.

Cryptography is associated with the process of converting ordinary plain text into unintelligible text and vice-versa.

It secures information and communications using a set of rules that allows only those intended—and no one else—to receive the information to access and process it.

Working Principle :

We are seeking inspiration from an early cryptographic technique, viz. Hill Cipher, in this project, we have designed an encrypter-decrypter using matrix multiplication. This technique for encryption-decryption is efficient in the transmission of text-based messages.

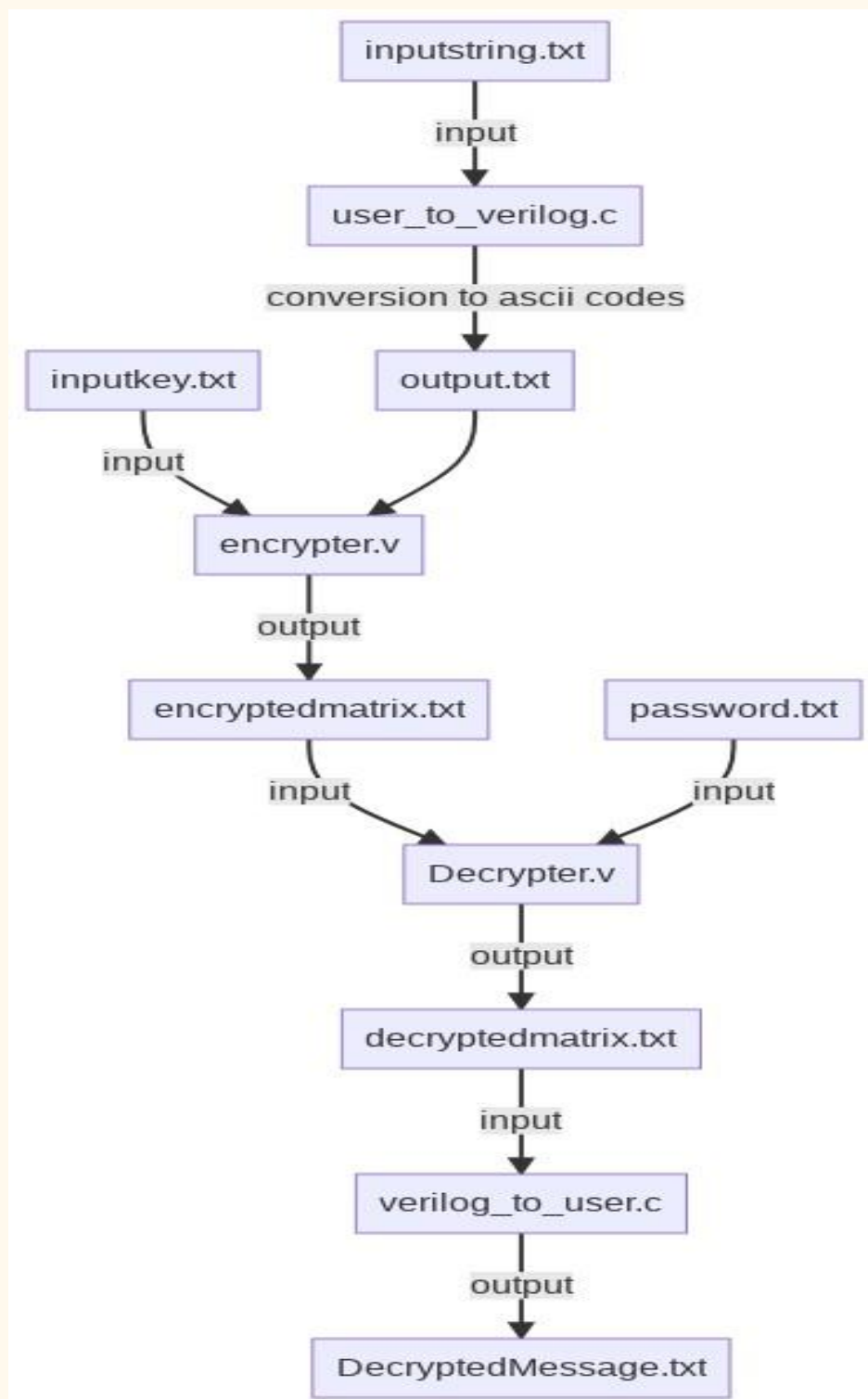
We have exploited a fundamental principle of Linear algebra, that given a non-singular matrix, there exists a unique inverse for it.

Thus, we consider a Matrix(say P) as a password and multiply it with another matrix(say A), i.e., PA . Then this matrix (PA) could be considered as an encrypted matrix. A matrix(PA) can be decrypted only if we multiply it with the inverse of a matrix(P).

i.e,
$$[P^{-1}][PA] = [A]$$

Since the inverse of a matrix is unique, thus decryption can be done only if the original matrix (P) is known.

MODULE HIERARCHY :



DESCRIPTION OF DESIGN

- **Submodules:**

Encryption:

- ❖ A given message is considered a string of characters for encryption and is first converted into its corresponding “ASCII” codes.
- ❖ These “ASCII” coded characters are stored in the form of a matrix in a reg, viz
`reg [7:0] ascii_matrix[0:1][0:1000]`
- ❖ Input key, i.e. entered by the user in inputkey.txt, is stored in `reg [7:0] key_matrix [0:1][0:1]`.
- ❖ Now, matrix multiplication is performed between `key_matrix` & `ascii_matrix`, and output is stored in `reg [14:0] encrypted_matrix[0:1][0:1000]`, which is further written in `encryptedmatrix.txt` as an encrypted message.

$$[\text{encrypted_matrix}] = [\text{key_matrix}] \times [\text{ascii_matrix}]$$

Decryption:

- ❖ For decryption, an encrypted message is read from `encryptedmatrix.txt` and stored in `reg [14:0] encrypted_matrix[0:1][0:1000]`.
- ❖ Now the `password`, which the user enters in password.txt, is read and is stored in the form of its inverse in `reg signed [7:0] inverse[0:1][0:1]`.
- ❖ Again matrix multiplication is performed between `inverse` & `encrypted_matrix`, and each term is divided by the `determinant` of the `inverse` matrix and stored in the `decrypted_matrix[0:1][0:1000]`

$$[\text{decrypted_matrix}] = \left(\frac{1}{\text{determinant}} \right) [\text{inverse}] \times [\text{encrypted_matrix}]$$

- ❖ Thus, if the `password` is correct, the message’s valid “ASCII” values are reproduced.
- ❖ Now, with the help of `verilog_to_user.c`, these ASCII codes are converted into corresponding characters to get the desired message.

DEMONSTRATIONS AND RESULTS:

Let us understand how our encrypter-decrypter work with the help of the following demonstration:

Encryption steps:

- ❖ **Step 1:** We first input the text that we intend to transmit in a text file, i.e., *input.txt*

```
inputstring.txt M X
text_files > inputstring.txt
1 we are team.
2 |
```

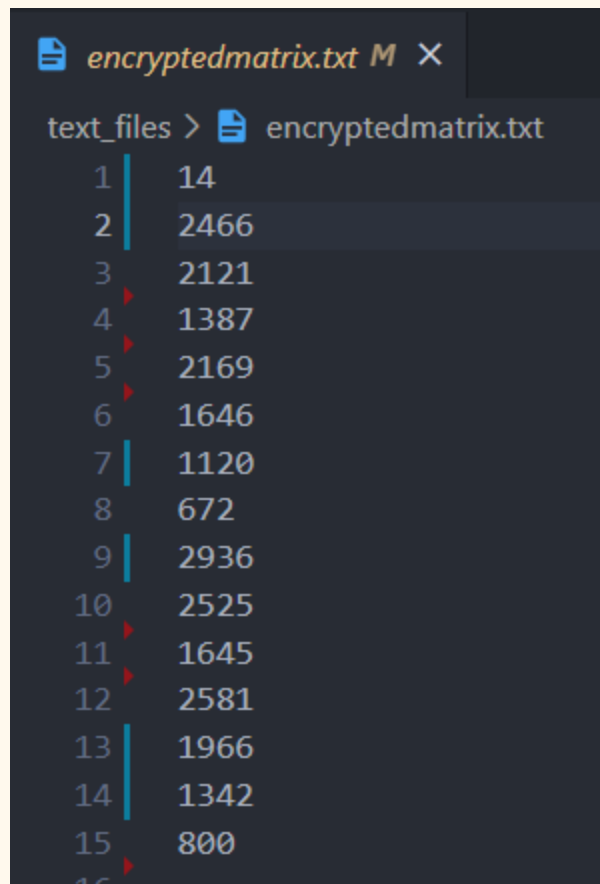
- ❖ **Step 2:** Then, we enter the key in the form of a 2 x 2 non-singular matrix that will be used to encrypt our message. (Note: This key should not be disclosed to anyone except the intended person.)

```
inputkey.txt M X
text_files > inputkey.txt
1 10 11
2 12 13
3 |
```

- ❖ **Step 3:** Running *user_to_verilog.c*: running this program will convert each text character into its corresponding ASCII codes. This file will write all ASCII codes into *output.txt*. This file will be used in the coming operations.

```
output.txt M X
text_files > output.txt
1 14
2 119
3 101
4 32
5 97
6 114
7 101
8 32
9 116
10 101
11 97
12 109
13 46
14 10
15 32
```

- ❖ **Step 4:** Now, with the help of *encrypter.v*, these ASCII codes are encrypted.
- ❖ **Step 5:** The newly created output file, i.e., *encryptedmatrix.txt* will then be transmitted.
This *encryptedmatrix.txt* is analogically equivalent to codewords.



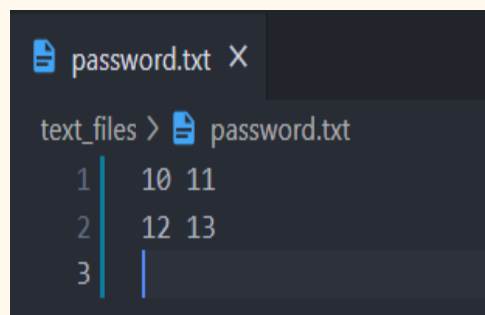
```

encryptedmatrix.txt M X
text_files > encryptedmatrix.txt
1 | 14
2 | 2466
3 | 2121
4 | 1387
5 | 2169
6 | 1646
7 | 1120
8 | 672
9 | 2936
10 | 2525
11 | 1645
12 | 2581
13 | 1966
14 | 1342
15 | 800
16 |

```

Decryption steps:

- ❖ **Step 1:** The receiver will then receive *encryptedmatrix.txt* and enter a password in the form of a 2 x 2 non-singular matrix in *password.txt*. (Note: For meaningful decryption, this password should be the same as the key set by the transmitter.)



```

password.txt X
text_files > password.txt
1 | 10 11
2 | 12 13
3 |

```

- ❖ **Step 2:** This 2 x 2 matrix entered in *password.txt*, is used by the *decrypter.v* to decrypt the message back into ASCII codes.
- ❖ **Step 3:** Then by running *user_to_verilog.c*, corresponding characters from these ASCII codes are written in *decryptedmessage.txt*.

```

decryptedmatrix.txt M X
text_files > decryptedmatrix.txt
1 119
2 101
3 32
4 97
5 114
6 101
7 32
8 116
9 101
10 97
11 109
12 46
13 10
14 32

```

- ❖ **Step 4:** This message is readable iff, the password entered is the same as the original key.

```

decryptedmessage.txt M X
text_files > decryptedmessage.txt
1 we are team.
2

```

(a)

```

decryptedmessage.txt M X
text_files > decryptedmessage.txt
1 | f?? 1 BS ? DC3 ? ; R* ?

```

(b)

Decrypted message (a) when the original key is used as a password, (b) when the wrong password is used.

SCOPE FOR FURTHER IMPROVEMENT:

This method of encryption-decryption heavily relies on a key matrix. If an attacker gets the key matrix, then the algorithm to encrypt-decrypt will not remain secure. With the advent of modern computers, there is a need to adopt enhanced encryption techniques.

To improve this algorithm further, we can **change the dimensions of key and target**. The higher dimension of the matrix signifies more cases a computer has to go through. More cases mean more time for computation, which may not be feasible after a limit. Thus making the algorithm better. However, increasing dimension comes at the cost of increasing reserved memory in **reg**.

CONCLUSIONS:

This Project touched upon the modern and needy topic of cryptography. We are presenting one of the earliest algorithms of using matrix multiplication as an encryption-decryption algorithm. The only thing that should be protected and guarded during the whole transmission is the invertible key matrix. If the key matrix falls in the hands of unintended recipients, they may decrypt the message.