|  |
| --- |
| *Depart. Of CSE, KNU* ***Network Programming*** *Due day: June 28, 2020*  **DOUBLE LOCK ENCRYPTION**  Name: Athul Shibu Student ID: 2017118100 |

**INTRODUCTION**

**Abstract:**

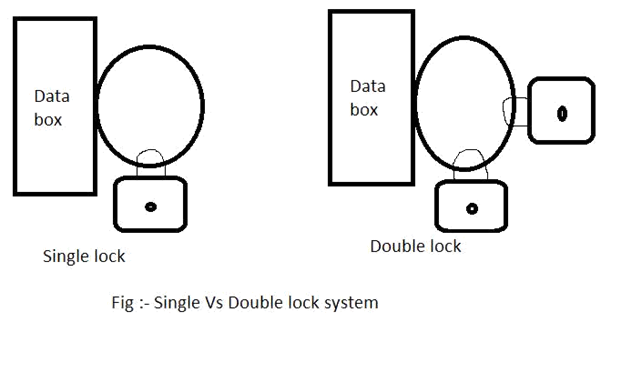
In cryptography, encryption is the process of encoding information. This process converts the original representation of the information, known as plaintext, into an alternative form known as Cipher text. Only authorized parties can decipher a Cipher text back to plaintext and access the original information. Although all communication-based applications require security for their data, the methods of symmetric and asymmetric algorithms for Cryptography are limited and easily crackable with enough resources. This project report describes the design and implementation of a simplified algorithm based on the concept of elementary row operation on the information stored in matrix form.

**Key Words:** Data Encryption, Decryption, Double Lock

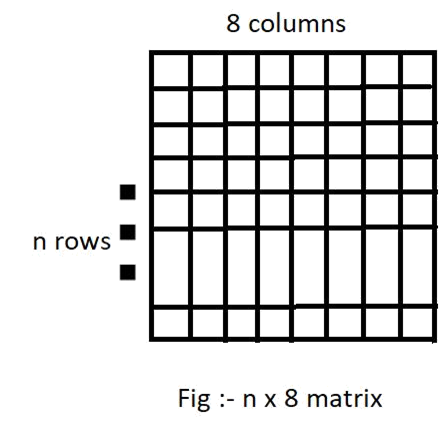
**Expected Result:**

The Program should be able to implement the Double Lock method of Encryption to encrypt and decrypt all messages between the Server and the Client, and vice versa. The Client connects to the Server’s socket, and begins to send messages. The Server will be able to reply to the messages. Each message between the Server and the Client will be sent and received according to the algorithms of Double-lock encryption/decryption. For the Algorithm, the message is first converted into an array of each character’s ASCII values. These ASCII values are then converted into 8-bit Strings of its Binary values. The loop then runs once for each character in the array, i.e., each line in the String of Binary values. Each line (which is the Binary form of a Character’s ASCII value) is XOR’d with the sender’s Private key. Thus the Sender locks the character before sending it. The receiver now receives a Binary String, which it promptly locks by XOR’ing it with its own Private Key. It is then send back to the Sender. The sender now unlocks the line by XOR’ing it with its Private key, then sends it back to the receiver. The receiver unlocks it with its own private key, thus receiving the Binary value of one character of the message. This line is stored into an array, and the process repeats until it receives a String “done” from the sender (not in Binary form). Then it converts all lines it stored earlier into Decimal, and then to characters to read and print the message.

**Overview of the Solution:**

****

For a given message, each of its characters will be converted into its equivalent 8-bit binary. This binary equivalent will be stored in a matrix, i.e., if there are *‘n’* characters in a message then the order of matrix will be ‘*n* x 8’. Each row corresponds to a single character.



First row of the Matrix is then encrypted with the Private Key of the Server, and sent to the Client. The Client then encrypts the message with their Private Key, and sends it back to the Server. The Server decrypts the new message and sends it back to the Client. The client now decrypts the final message to extract the original message from the Server. This process is repeated until the server sends the whole message to the client.

Let **“**P**”** and **“**Q**”** be the private key of Server and Client respectively and consider some random character **“**M1**”** which corresponds to 8-bits first row of the matrix then **–**

**Server’s Side :-** M1 (XOR) P = Say X

* X is send to Client (First lock is applied)

**Client’s Side :-** X (XOR) Q = Say Y

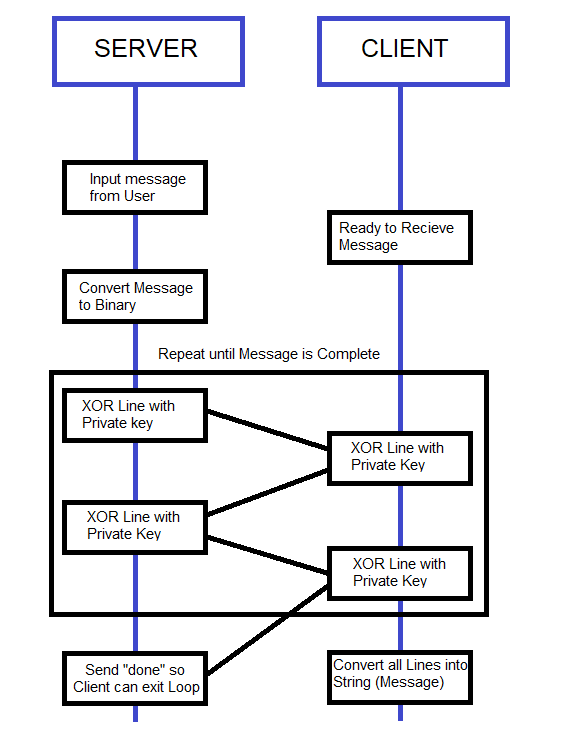
* Y is send to Server (Second lock is applied)

**Server’s Side :-** Y (XOR) P = Say Z

* Z is send to Client (First lock is released)

**Client’s Side :-** Z (XOR) Q = M1

* Now the Client will release their lock, i.e., decrypt the message to see the original message.



**Programming Components:**

There are 6 functions excluding **main()** in both “Client.c” and “Server.c”.

1. **void error\_handling()** prints out errors and exits the program. This function is called in case of errors that can be accounted for by the programmer. Eg. If the **socket()**, **bind()** or **listen()** in “Server.c” returns -1, there was an error in the connect, binding or listening. Since the connection cannot be established in case of such errors, the program prints the error and exits the execution.
2. **void dec\_to\_bin()** converts the given String, first into an array of ASCII values of each of its characters, then into binary forms of each ASCII value. The final conversion is stored in the aray **bin\_mes**.
3. **void bin\_to\_dec()** converts the given array of binary values, which is given in an array of Strings, into the characters and strings it represents. The final String is stored in **message.**
4. **void xor()** applies XOR function to the given string (which is always a String of 8 characters of 1’s and 0’s) with the static String **private\_key**. The Private Key can be specified to be any 8-bit String of 1’s and 0’s in either of the programs for the messages to be encrypted and decrypted properly.
5. **void receive\_message()** runs an Infinite loop that exits only when it receives a String “done” from the Server/Client. The function (inside the Loop) first checks if the received String is “done”. If not, it is assumed to be a String of 1’s and 0’s corresponding to a character in the Server’s/Client’s message. It then performs the XOR operation on the String with the Private Key and locking the character, courtesy of **xor()** function, and sends the new String back. It then receives the the new String and calls **xor()** again, thus unlocking the character. The unlocked character is stored as a row in **bin\_mes**. Finally, once out of the loop, the **bin\_mes**, still in Binary form, is converted to Decimal and then String by calling the **bin\_to\_dec()** function. The **bin\_mes\_str** is, which was called by reference from this function, now contains the message from the Server/Client.
6. **void send\_message()** runs a loop equivalent to the length of the message to be sent. Before entering the loop however, the function calls **dec\_to\_bin()** to convert the message into an array of Binary Strings, each corresponding to the ASCII value of each character in the message. Inside the loop, the Binary Strings, one String (i.e., on character of the message) is XOR’ed with the Private Key using **xor()** function. It is the sent to the Server/Client through the socket. After reading the reply, the new String, which is also an array of 1’s and 0’s is XOR’ed with the Private Key and re-sent to the Server/Client. After exiting the loop, the function sends a message with string “done” to allow the receiver function to exit from it’s loop.
7. **int main()** of Server.c is where the Server connects to the Client. This function has all the variables necessary to connect to a Client. After creating a Socket, **server\_socket**, it initializes the Server with the System’s IP address and a port number given as an argument to the **main()** during execution. After binding and listening, the Server connects to the Client. The rest of the program happens inside an Infinite Loop. In this loop, the Server accepts messages from the Client and sends messages to the client. First, it calls the **receive\_message()** function to receive messages, which stores the message in the String **message** (passed by reference from **main()**). If the message from the Client is equivalent to “q” or “Q”, the program sends message “q” to the client, allowing the client also to exit the loop and disconnect from the Server, then the Server disconnect. Else, it prints the message from the client (the message that it just received), and accepts the reply message of the Server. The function now calls **send\_message()** with the reply as an argument to send a message to the Client.
8. **int main()** of Client.c is where the Client connects to the Server’s socket. This function has all the variables necessary to connect to the Server. After creating a Socket, **client\_socket**, it initializes the Server with the IP address and port number given as arguments to the **main()** during execution. After the Server connects to the Client, the rest of the program happens inside an Infinite Loop. In this loop, the Server accepts messages from the Client and sends messages to the client. First, the program accepts an input message into **message**. If the message is equivalent to “q” or “Q”, the Client sends a message “q” to the server and exit the loop. Else the function calls **send\_message()** with this String as an argument to send a message to the Server. It then calls the **receive\_message()** function to receive messages, which stores the message again in the String **message** (passed by reference from **main()**). It the prints the message from the Server (the message that it just received), and the loop resumes.

**RESULTS:**

**Screenshots:**

