MINI-PROJECT REPORT

RSA ENCRYPTION IN C

Submitted To,
Mr. Shiju Thomas M.Y.,
Assistant Professor,
Dep. Of Computer Science,
Rajagiri College of Social Science

Submitted By,
Milan Varghese (No. 21),
M.Sc. CS (Data Analytics)
msccs113@rajagiri.edu
Date: 11th January 2021

Mini-Project: RSA Encryption in C

I started of this project on “RSA Encryption in C” to try out cryptography for the first time. I was searching for different encryption algorithms and I ended up choosing RSA encryption. In order for me to understand how the encryption works I brain stormed with a few people I know and also searched on google for the how RSA works, I also had a book on Cryptography and network security by Pearson publications and they had the algorithm explained in the book.
Initially I was searching for different codes on RSA available on the internet. Most codes I found on the internet were not working correctly or were working correctly and was badly documented, which made it difficult to study. I somehow ended up finding a good code for reference studies and initially worked on understanding the code and researching on the subject. Later after exploring for a while I decided to program the code myself.

During programming part, I realized I needed to implement other algorithm in order to implement RSA encryption like Miller-Rabin Primality test in order to check whether a given number is prime or not, I also used an algorithm to find the modular multiplicative inverse of a number, A Basic Euclidean Algorithm was used to find the GCD of two numbers, etc. I stored my primality test and modular multiplicative inverse functions in a separate c program and called it into the main function.

The important part of RSA encryption is divided into 3 sections: The public and private key generation, encryption of information using the public key and decryption using private key. The encryption is done in *generate_keypair()* function in my code. The Encryption is done in the *encryption()* and *encryptText()* functions and Decryption is done using *decryption()* and *decryptText()* functions. I also added supplementary functions in order to display the saved keys and cypher text using the *displaykeys()* and *displayCypher()* functions respectively.

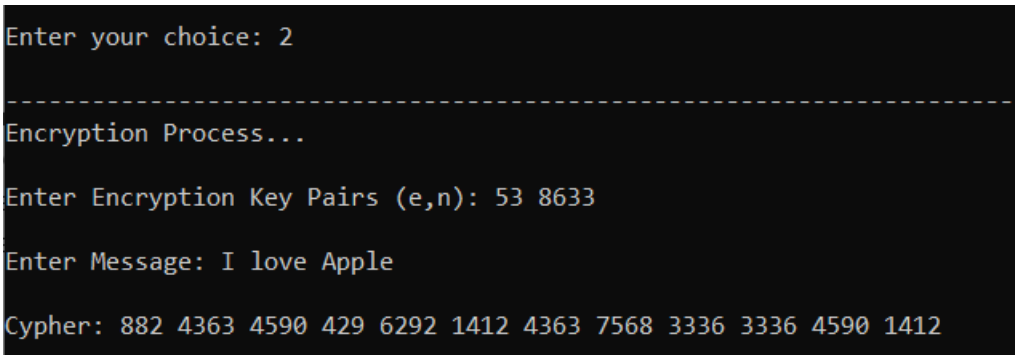
I added a menu so that all our functions can be organized and called, the menu was written inside the *menu()* function and our main function will call the menu function in an infinite loop what will only stop only if the exit option in the menu is selected. I also added an option in the menu in order to clear the console screen and display the menu.

```
------------------------------------------------------------
Menu - Rivest Shamir Adleman (RSA) Encryption
------------------------------------------------------------
1. Key Pair Generation
2. Encrypt Text
3. Decrypt Text
4. Display Cypher Text!
5. Display Key Pairs!
6. Clear Screen
7. Exit

Enter your choice:
```

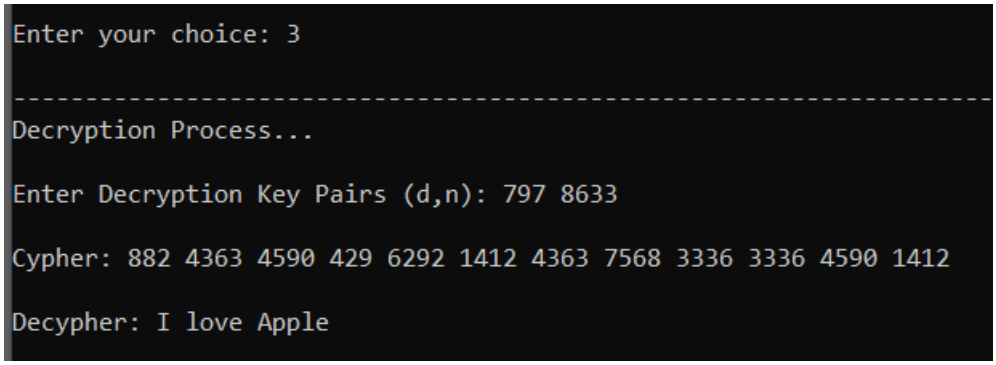
After running our program, the menu will be displayed in the console and we should start by generating the key pair so we should choose option 1. We need to have two prime numbers in order to generate a public and private key. The larger the prime number the more difficult it will be for hackers to break our encryption. Hence we need to use two large prime numbers in order to generate public and private keys. For demonstration I will be using prime numbers 89 and 97. After generating, the public and private key pairs will be displayed. Which we should keep in mind for the encryption and decryption process.

```
------------------------------------------------------------
Enter Two Prime Numbers (Larger Pairs More Secure): 89 97

------------------------------------------------------------
Generating Key Pairs...
Please Take Note of the Public and Private Key Pairs:

Encryption Key (e,n) = (53,8633)
Decryption Key (d,n) = (797,8633)
```

Only after generation of key pair is done we can move on to encryption of the text and then after encryption decryption. We should now select option 2 for the encryption process. For encryption we will be required to enter the public key pair and then the message we will be needing to encrypt is to be entered. The message we entered is stored in global array *text[1000]* and they cypher text will be stored in global array *cypher[1000]*. After encrypting the text, the cypher text will be displayed with a number representing each character and spaces in the message.



This cypher text is stored in a global array and now we will be able use the decryption function in menu. After selecting decryption, we are asked to enter the private key pair and our program will use the private key pair to decrypt the crypt one word at a time and display the decrypted message.



Hence our RSA algorithm successfully worked.
I also added other functions to display the key pairs generated, and display the encoded cypher text. I also added restrictions to the usage of these functions such that key pairs could only be displayed after generating it and also cypher text will only be displayed after generating it. Also I put a restriction on the initial input of two prime numbers such that the product of the two prime numbers won't be less than 255. This is done so that our program will be able to encrypt all the characters listed in the 8 Bit ASCII. So, we need to enter a two prime numbers whose product is larger than 256.

RSA Algorithm (Source: Cryptography and Network Security by William Stallings)

Key Generation:

1. Select two prime numbers, p and q.
2. Calculate n = p * q.
3. Calculate $\phi(n) = (p - 1) * (q - 1)$.
4. Select e such that e is relatively prime to $\phi(n)$ and less than $\phi(n)$.
5. Determine d such that $d * e \equiv 1 \pmod{\phi(n)}$ and $d < \phi(n)$. This can be calculated using Euclid's Algorithm.

The resulting keys are public key **PU = { e, n }** and private key **PR = { d, n }.**

Let M be the message and C be the cypher text. Now we can encrypt and decrypt using modular multiplicative inverse method and the key pairs.

Encryption: $C = M^e \bmod(n)$

Decryption: $M = C^e \bmod(n)$