CS203 : DIGITAL LOGIC DESIGN
PROJECT REPORT

Group Members -

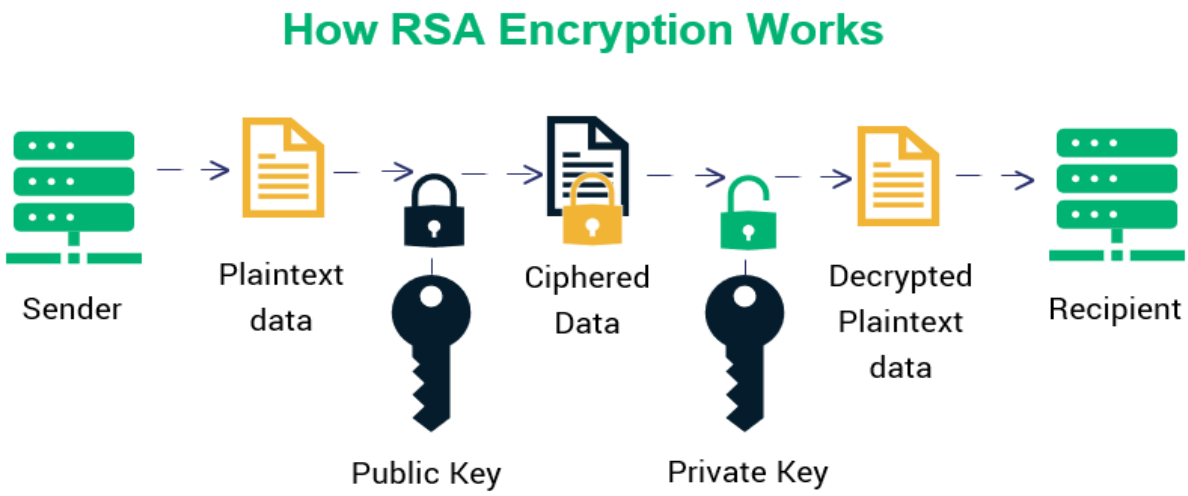
Name -	**Entry Number -**
Prakhar Saxena	2020CSB1111
Janmeet Singh Makkar	2020CSB1175

Introduction

RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key.** As the name describes that the Public Key is given to everyone and Private key is kept private.

An example of asymmetric cryptography :

1. A client (for example browser) sends its public key to the server and requests for some data.
2. The server encrypts the data using client's public key and sends the encrypted data.
3. Client receives this data and decrypts it.



Since this is asymmetric, nobody else except browser can decrypt the data even if a third party has public key of browser.

The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially.

Here , we have implemented a 256 bit RSA cryptosystem.

Let us learn the mechanism behind RSA algorithm (example):
>> Generating Public Key :
Select two prime no's. Suppose **P = 53 and Q = 59**.
Now First part of the Public key : **n = P*Q = 3127**.
We also need a small exponent say **e** :
But e Must be an integer. Not be a factor of n.
1 < e < Φ(n) [Φ(n) is discussed below],
Let us now consider it to be equal to 3.
Our Public Key is made of n and e

>> Generating Private Key :
We need to calculate Φ(n) :
Such that **Φ(n) = (P-1)(Q-1)**
so, Φ(n) = 3016

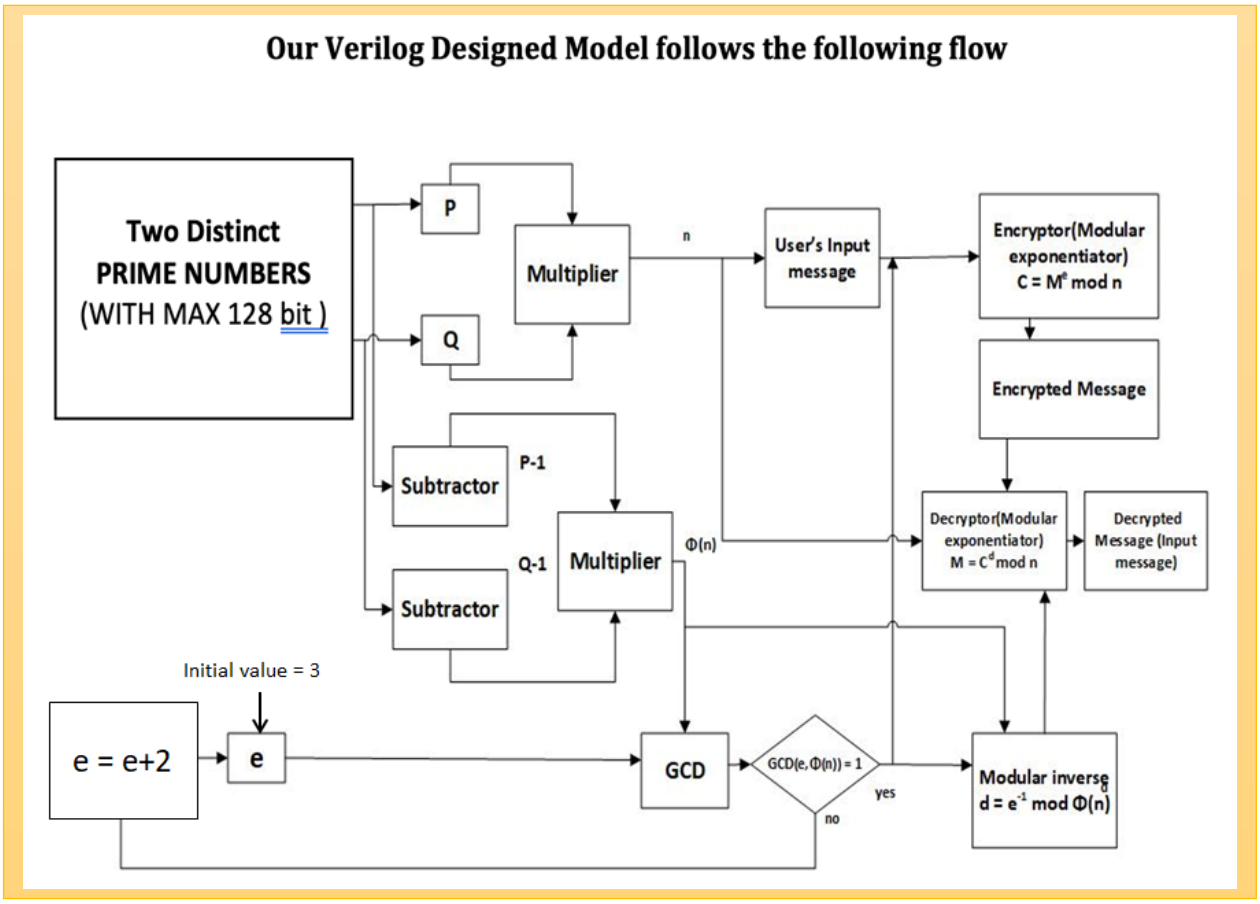
Now calculate Private Key, **d** :
d = (k*Φ(n) + 1) / e for some integer k
For k = 2, value of d is 2011.
Now we are ready with our – Public Key (n = 3127 and e = 3) and Private Key(d = 2011)
Now we will encrypt **"HI"** :
Convert letters to numbers : H = 8 and I = 9
Thus **Encrypted Data c = 89^e mod n**.
Thus our Encrypted Data comes out to be 1394
Now we will decrypt **1394** :
Decrypted Data = c^d mod n.
Thus our Encrypted Data comes out to be 89
8 = H and I = 9 i.e. "HI".

We were able to implement RSA algorithm using Verilog. The following flow was used for the purpose.

Generating RSA keys

The following steps are involved in generating RSA keys −

- Create two large prime numbers namely **p** and **q**. The product of these numbers will be called **n**, where **n= p*q**
- Generate a random number which is relatively prime with **(p-1)** and **(q-1).** Let the number be called as **e**.
- Calculate the modular inverse of e. The calculated inverse will be called as **d**.



The modules used to implement the idea above are defined as follows on the next page.

Module Definitions

1) **Control module** – This module is the main driver code for deciphering/encrypting the message.

```
module control(
    input [WIDTH-1:0] p,q,//prime numbers
    input clk,
    input reset,
    input reset1,
    input encrypt_decrypt,//if =1 it is used for encryption, otherwise decryption.
    input [WIDTH-1:0] msg_in,//msg input to be encrypted/decrypted
    output [WIDTH*2-1:0] msg_out,//output message after running the program
    output mod_exp_finish
    );
```

2) **Inverter module** – This module is responsible for generating “e” and “d” which are the public and private keys, respectively, for ciphering/deciphering the message. This procedure is done by setting e = 3, checking if the combination of e and d is valid. If valid, e and d are set. Else, e is incremented by 2 and the process repeats.

```
module inverter(
  input [WIDTH-1:0] p,
        input [WIDTH-1:0] q,
        input clk,
        input reset,
        output finish,
        output [WIDTH*2-1:0] e,
        output [WIDTH*2-1:0] d
    );
```

3) **Modulus exponentiator module –** Using the generated keys, this module calculates the deciphered/ciphered message using above mentioned formulae.

```
module mod_exp(
    input [WIDTH*2-1:0] base,
        input [WIDTH*2-1:0] modulo,
        input [WIDTH*2-1:0] exponent,
        input clk,
        input reset,
        output finish,
    output [WIDTH*2-1:0] result
    );
```

Also the mod module for calculating modulo arithmetic was also implemented.

Implementation (Simulation using Testbench)

```
RSA 256-BIT ENCRYPTION/DECRYPTION TESTER


Prime Numbers Used are:
1.p=                                                    113680897410347
2.q=                                                79990808077935876437321

Input Message is        : 000000000000000000000000000000000000262d806a3e18f03ab37b2857e7e149
Your decrypted Message is:000000000000000000000000000000000000000048656c6c6f20576f726c6421

Input Message is        : 000000000000000000000000000000000000000048656c6c6f20576f726c6421
Your Encrypted Message is:000000000000000000000000000000000000262d806a3e18f03ab37b2857e7e149

Input Message is        : 0000000000000000000000000000000008caea22b8f7a6b9dfa6c57896e0ed7bc
Your decrypted Message is:000000000000000000000000000000000000048dc24a38d4de8e2e3cf516cb970f6

Input Message is        : 0000000000000000000000000000000008caea22b8f7a6b9dfa6c57896e0ed7bc
Your Encrypted Message is:000000000000000000000000000000000000004271d336b05ada02df5711f579ef82

Done
```

A still from result obtained when program is executed on eda-playground using icarus verilog

Thus, we can verify working of our RSA crypto system using sample cases 1 and 2 , we can observe that the output message in case 1 (decryption), when used as an input message for encryption in case 2 will give same input as case 1 as an output .Thus our calculations work!

Conclusion

From this project we learnt regarding the industry wide data security using RSA cryptography which is majorly in work whenever we use an internet browser, this security practice ensures the elimination of malicious man In the middle attack along with protecting user privacy by ciphering a plain text the moment it is sent and deciphering it only when received with the help of public and private keys.

We implemented digital logic while designing the Verilog models which came together to form this RSA 256 bit cryptosystem program as can be seen using the flowchart above and code on our github.