This application is an implementation of various methods needed for the **RSA Cryptography** using NUMBER THEORY algorithms. In this application, I have implemented some of the algorithms to find out various things related to RSA Cryptography taught in the NUMBER THEORY AND CRYPTOGRAPHY COURSE at COURSERA. The names of the functionalities are the names of the buttons.

Here is a small description of the methods and how to use them in this application.

**Random Prime Generator:**

It generates random prime number of 1024 bitlength.

NB: 2048 bitlength is also possible, but it takes much time and sometimes application shows unexpected behaviour. So, for simplicity I use 1024 bitlength.

And, 1024 bitlength prime will also be used throughout the application.

**Big Multiplication:**

It generates the multiplication of two big number. Again ,for simplicity, two numbers consist of less than or equal to 1024 digits length.

**String to ASCII code:**

Here the user has to input a message(string) in the required field and the program will return the ASCII equivalent of that message.

**ASCII Code to String:**

Opposite of the previous method. Input is an ASCII equivalent and output will be the string/message

**Powmod Function:**

A basic method for problem solving at number theory and also in the encryption process of the RSA. Here *PowMod*(*a*,*n*,*modulo*) which computes and shows (a^n mod {modulo})

**a** is the ASCII equivalent of a message/string .The user has to enter the value of a, n and modulo.

**InvertModulo Function:**

Another basic method for number theory and RSA.

*InvertModulo*(*a*,*n*) which takes coprime integers a and n as inputs and returns integer b such that  *ab*≡1mod*n.so,the program will show the multiplicative inverse of a.*

**Encryption:**

Here comes the original functionality of RSA. The user has to enter the MESSAGE/string,

Public key exponent and the private key pair(p,q)

The program will return the encrypted CIPHERTEXT of the message.

Ciphertext 𝑐 = 𝑚^e mod 𝑛

**Decryption:**

Here the user has to enter the encrypted CIPHERTEXT, both component of the private key pair(p,q) and the public key exponent(e).The program will return the

Decrypted that means the original MESSAGE/string(d).

𝑐^d ≡ 𝑚 mod 𝑛

**NOW COMES SOME ATTACKS THROUGH WHICH AN ENCRYPTED TEXT CAN BE DECRYPTED IF THE ENCRYPTION PROCESS IS NOT STRONG ENOUGH**

**Decipher Simple:**

Firstly, suppose you can imagine the message somehow because you know and pretty sure that the sender of the message will create very small message for example: ’Attack’, ’wait’, ’hello world’ etc. So, in this case, the user has to input the CIPHERTEXT, the MODULO, the public key EXPONENT and finally three MESSAGE IMAGINATIONS. The program will return the actual message if any one of the imagination was right. Otherwise, the program will return a message saying

“Sorry, Wrong imagination. Please try again”.

**Decipher Small Prime:**

The sender is using RSA encryption with a public key modulo, exponent such that modulo = *p*⋅*q* with one of the primes *p* and q being less than 1000000, and you know about it. You want to break the cipher and decrypt her message.

This function will help the user in this case, the user has to input the CIPHERTEXT,

The MODULO and the public key EXPONENT and the program will show the original message. If the prime is larger than 1000000, then the program will show a Warning saying “sorry, can’t determine the original message”

**Decipher Small Difference:**

The sender is using RSA encryption with a public key modulo, exponent such that modulo =

*p*⋅ *q* with |p - q| < 5000, that is the difference of the private key pair is very small, and you know about it. You want to break the cipher and decrypt the message.

In this case, this function will help, the user has to input the CIPHERTEXT,

The MODULO and the public key EXPONENT and program will show the original message. Again, the program can’t find it in case the difference is bigger than 10000 then the program will show a warning.

**Decipher Common Divisor:**

You've discovered that the first prime number *p* for the private key was generated with the same algorithm and the same random seed by two different senders Alice and Angelina due to insufficient randomness, while the second prime q is different for those two private keys. You want to break both ciphers and decipher messages from both Alice and Angelina.

In this case, the user will get the advantage, the user has to input the CIPHERTEXT, The MODULO and the public key EXPONENT for both senders and the program will show the messages for the both senders

**Decipher Hastad:**

This is known as the famous “Hastad’s” Broadcast Attack.

Bob has sent the same message to Alice and Angelina using two different public keys (n1, e = 2) and (n2, e = 2) with the same public key exponent e = 2.

In this case,the user has to enter the CIPHERTEXT and the MODULO for both.

The program will show the original message. For simplicity,I have restricted the implementation e=2.But,for any e, it is possible. but the condition for working this algorithm is that e should be same as the number of receivers. That means, in this case for two receivers,e is 2.

For three receivers, It should be 3 and so on.

NB: these attacks function may take little bit of time(20-60 seconds) if the inputted numbers are too huge, but they will definitely work. Again, also sometimes application gets freezed and doesn’t respond, but be patient. It will automatically respond when the answer will be ready.

[However, tremendous course on NUMBER THEORY AND CRYPTOGRAPHY on COURSERA (By University of CALIFORNIA San Diago and Higher School of Economics) has encouraged me to implement this simple application.]

Course link: <https://www.coursera.org/learn/number-theory-cryptography>