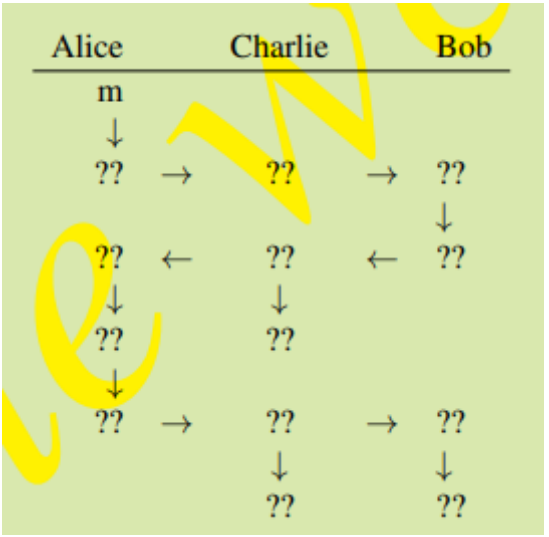
Mini Project Coursework

This report aims to help document the work that was done to help solve Assignment I of the mini project coursework

Assignment I

Develop a software prototype in Java to demonstrate how the RSA algorithms work using the simplified algorithms and examples studied in the lectures/workshops. In particular, your prototype should demonstrate how two primes p and q are generated, how the random number e is generated, where 0 < e < r and e has no factor in common with r, and how the private key d and public key (e,n) are generated. As part of testing, a good coursework may also demonstrate a special case when your RSA program would not work securely. Your program should prompt the user to input certain parameters that would lead to the problematic state. There is no specific requirement to the user interface of your prototype but you should design at least a simple user interface to allow the user to simulate a communication scenario, where Alice sends an encrypted message to Bob, and Bob decrypts the ciphertext to read the message. Also, Charlie may intercept the data flow and obtain unauthorised information.

For example, the following format may be adopted to demonstrate what happens with the plaintext m that from Alice to Bob, where "??" parts are for you to design.



You may decide where to start your design but it would often be easier to first divide the task into a number of subtasks. For example,

1. Implement a cryptorandom key generator and the algorithm for modular exponentiation.

2. Implement the RSA encryption algorithm.

3. Implement the RSA decryption algorithm.

You may add if necessary assumptions for details to ease your implementation, but you must explain them clearly to gain credits.

Abstract

RSA is an asymmetric system, which means that a key pair will be generated: a public key and a private key. In RSA, the public key n is generated by multiplying two large prime numbers p and q together, and you then compute r = (p-1) * (q-1). After these have been computed you can start to generate a public key e which is a number that is less than r and greater than 1 and has no common factors with r. To calculate the private key d you would need to use modular inverse on e and r which allows you to find the value of d. A user can then distribute their public key e and n, and anyone wishing to send the user a message would encrypt their message using the public key. The receiver can use their private key d to decrypt the encrypted message.

Notes

To run the system:

- Run Menu
- In the console there are 3 options
    - Run console menu
    - Run the GUI
    - Exit the program

If you press the console menu, there are two further options to select:

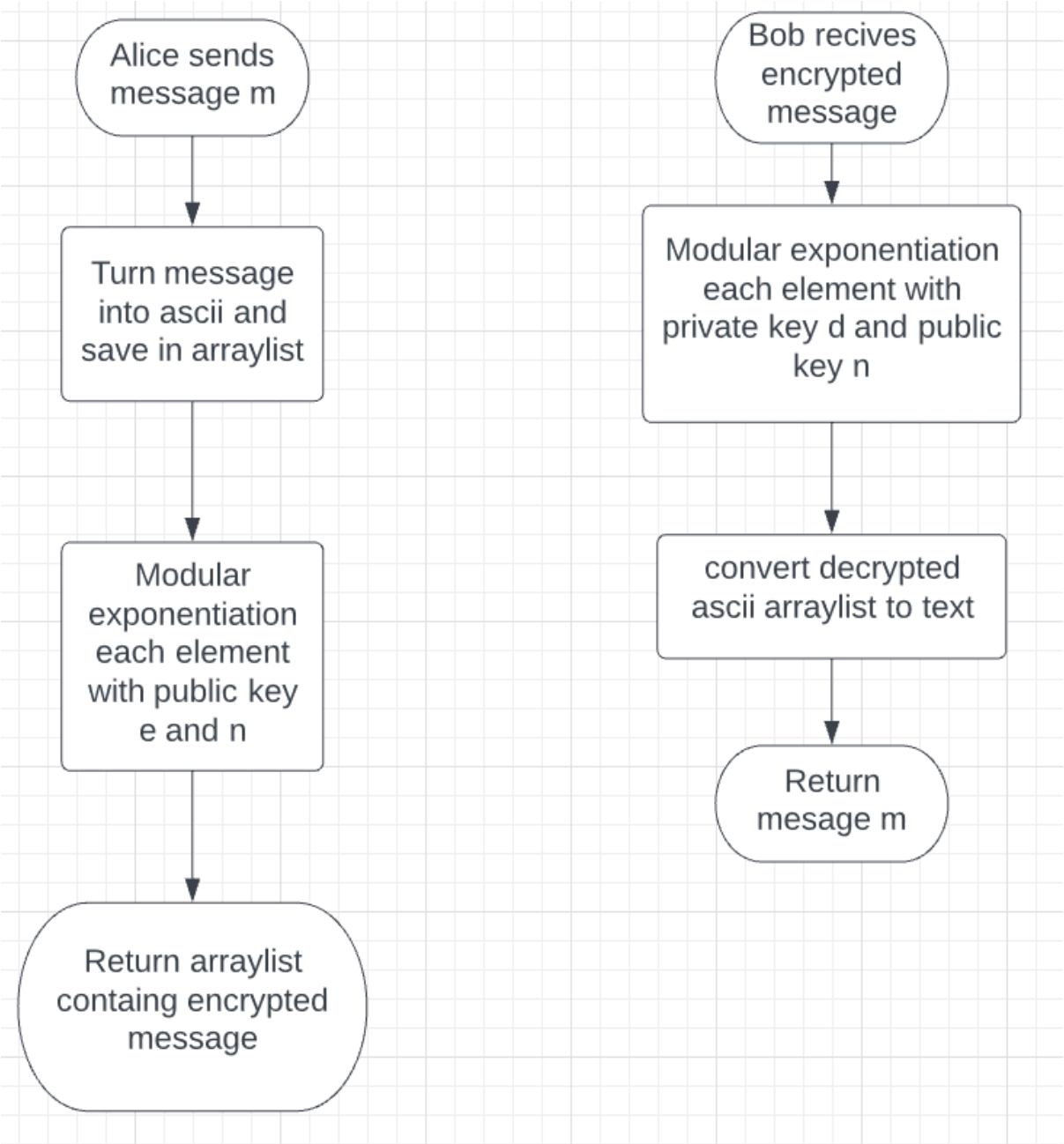
- Run the RSA System
- Exit the program

If you select the RSA System then it will print out the computations of keys being generated to the console. It then requires you to input a message for you to send. After you send the message, the system will show the ASCII form and the encrypted form of the message. Then it will decrypt the message and show what Bob will see after decryption. Then you have the option to quit the program or send a new message.

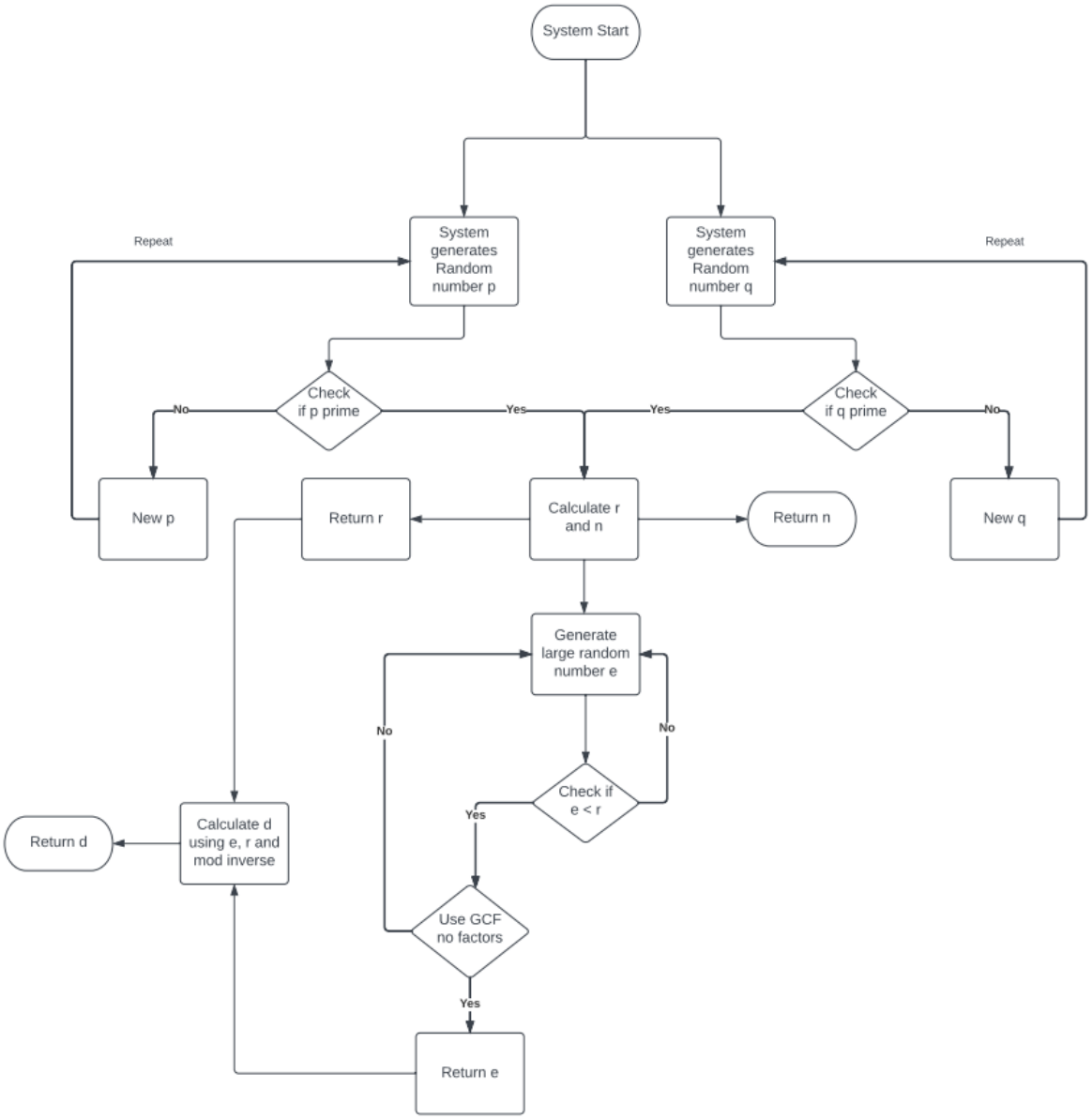
After quitting, you will go back to the start menu where you will get the 3 options as said above.

If you press the GUI system, a window will open which is shown below in our demonstration.

Algorithms

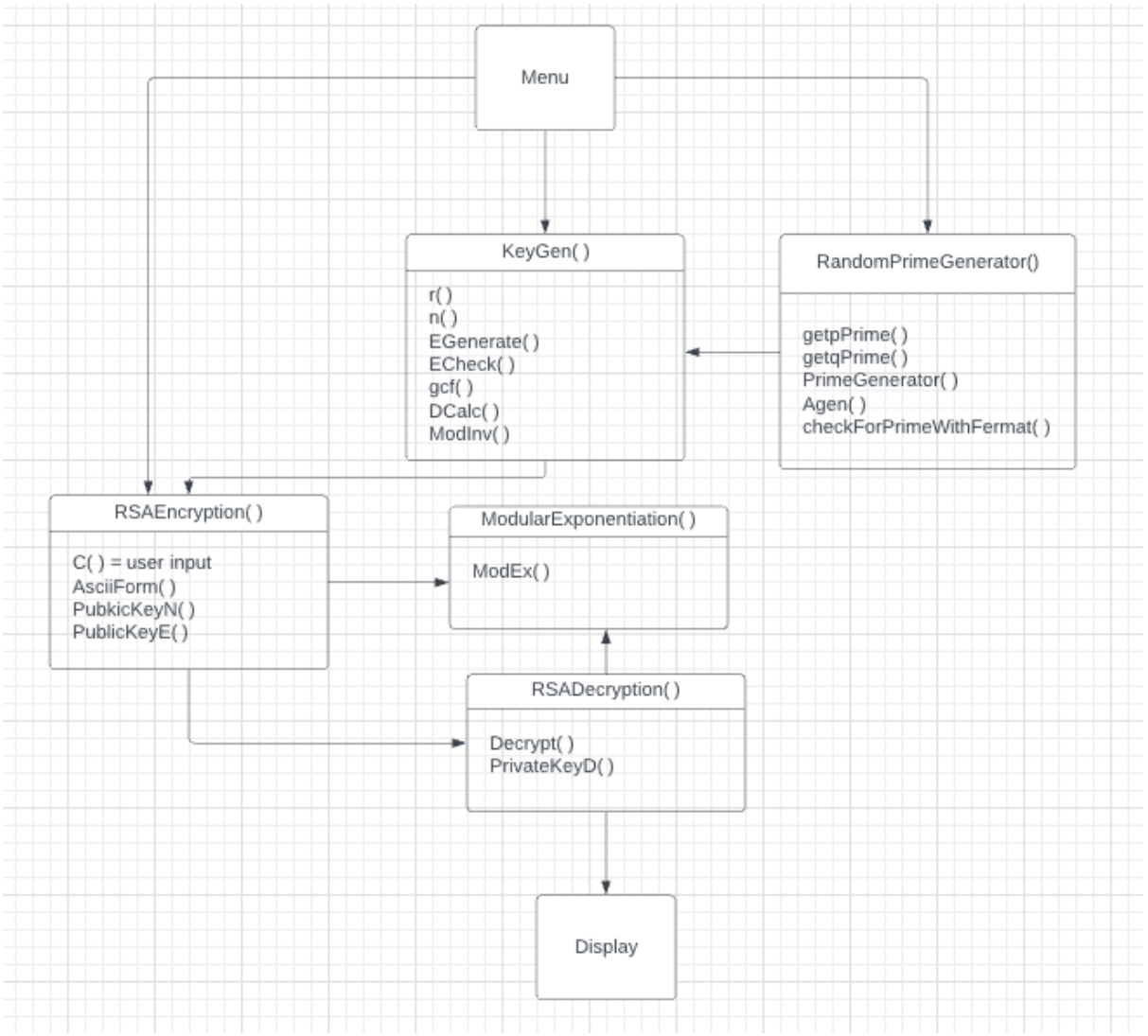


User Flowchart for Alice and Bob



System Flowchart

Design



UML Diagram

Demonstration

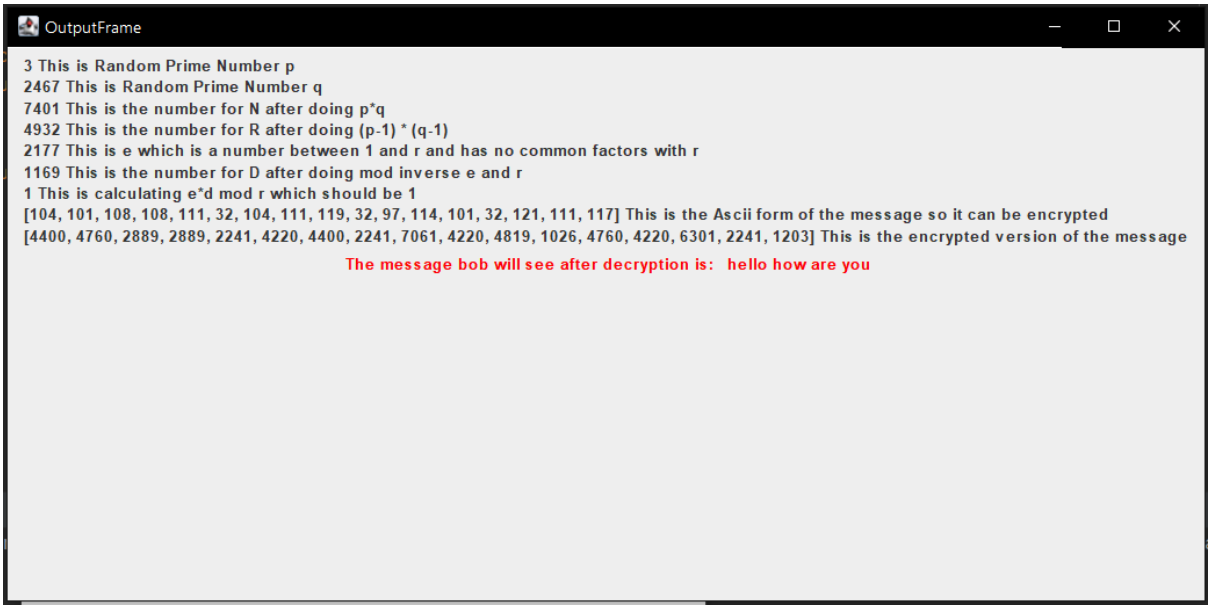
When the code is run, this is the window that will open:

Opening UI

User simple has to type in a message into the textbox and then press submit:

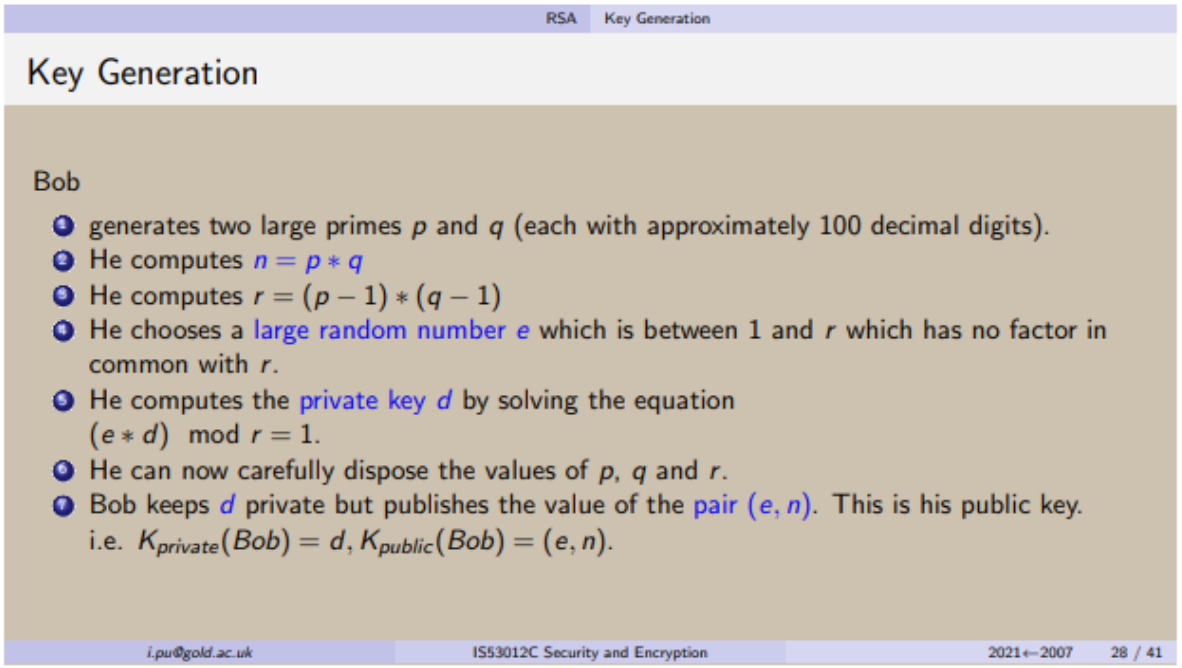
Sending a message

After doing so, a new window will open:



Output and explanations

Here we have all our values needed to implement a cryptorandom key generator and the algorithm for modular exponentiation with the RSA encryption and decryption algorithm. This has been cross referenced with the lecture notes:

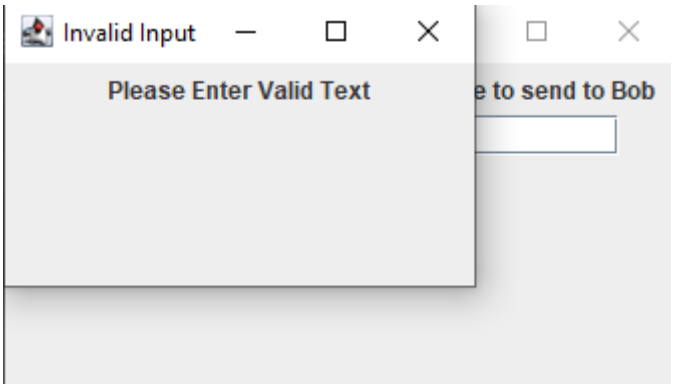


Taken from Week 4 Public Key Cryptosystems: RSA and El GamalFile lecture notes

RSA is an asymmetric system, which means that a key pair will be generated: a public key and a private key. In RSA, the public key n is generated by multiplying two large prime numbers p and q together. Once you have calculated n then you are required to compute r by doing (p-1) * (q-1). After these have been computed you can start to generate e which is going to be a part of the public key, e is a number that is less than r and greater than 1 and

has no common factors with r. This is going to be done by calculating the greatest common factor between e and r. If the algorithm for greatest common factor returns 1 then it means that there are no common factors between e and r meaning that you can use e. To calculate the private key d you would need to use modular inverse on e and r which allows you to find the value of d. A user can then distribute their public key e and n, and anyone wishing to send the user a message would encrypt their message using the public key. The receiver can use their private key d to decrypt the encrypted message.

We also have some validation checking. When no text has been inputted a pop up appears telling the user to enter valid text:



Validation

Discussion

Overall we are satisfied with the work we have completed for this coursework. We are pleased with the outcome and believe we have met all the requirements along the way. The process was quite lengthy and difficult but after many hours put in we have created a fully functional software prototype in Java which demonstrates how the RSA algorithms work using the simplified algorithms.

We did have some problems however which are discussed below in how we could have improved the work we have done:

The RSA keys for our code need to fall within certain parameters to be secure. An example of this parameter is the prime p an q, if these two are close together they can easily be discovered. Similarly the number d which represents the private key can not be too small as low value makes it easy to solve. The numbers used in our encryption need to be an adequate length to keep the key safe, as smaller numbers can easily be found out using brute force and factoring.

Charlie can be used as an example of a hacker that will try to decipher the message in this particular case using brute force algorithm, this is because we do not generate 100 digit

primes. Brute force algorithm is a very straightforward method of solving a problem, it works by trying every possibility of cracking the cipher using computing power.

```
c ← first(P)
while c ≠ Λ do
    if valid(P,c) then
        output(P, c)
    c ← next(P, c)
end while
```

Pseudo code for a basic brute force algorithm.

What could be improved

- Generating large primes
    - It was encouraged for us to generate 100 digit primes for p and q but we started getting errors when we started our encryption process. When we were able to fix this we then started getting errors in our decryption process. Essentially it was just a chain reaction of errors so we thought we would be better off using 10000 digits and using a long data type to store it.
- Prime Checker
    - Currently we manually check if our p and q are prime using if statements. To increase the complexity of the function we could have tried to implement an algorithm to do it instead. This way greater marks may be awarded.
- Improve efficiency of code to work with larger numbers
    - The prototype takes a little while to compute our inputted numbers. We could do this by implementing appropriate algorithms and minimising the use of if else statements. However, it is also possible that it is just our machines that are not fast enough.
- Improving UI
    - Although we acknowledge the UI is meant to be simple, we believe it could be better so that it reduces confusion of how it works. Increasing usability is something we can do to improve on this.
- Brute Force algorithm (Charlie)

- We currently don't have Charlie the hacker implemented in our code but if we had more we would be able to implement a basic brute force algorithm using the pseudo code we researched.

Total Number of Hours Spent	42
Hours Spent for Algorithm Design	3
Hours Spent for Programming	33
Hours Spent for Writing Report	3
Hours Spent for Testing	3
Note for the examiner (if any):	N/A

Bibliography

Modular multiplicative inverse. (2022, February 19). GeeksforGeeks. Retrieved March 22, 2022, from https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/

Modular Exponentiation (Power in Modular Arithmetic). (2022, February 1). GeeksforGeeks. Retrieved March 22, 2022, from https://www.geeksforgeeks.org/modular-exponentiation-power-in-modular-arithmetic/

Prime Numbers Generator and Checker. (n.d.). Number Empire. Retrieved March 22, 2022, from https://www.numberempire.com/primenumbers.php

How to find d, given p, q, and e in RSA? (2013, May 1). Stack Overflow. Retrieved March 22, 2022, from https://stackoverflow.com/questions/16310871/how-to-find-d-given-p-q-and-e-in-rsa

Modular Exponentiation (Power in Modular Arithmetic). (n.d.). TutorialsPoint.dev. Retrieved March 22, 2022, from https://tutorialspoint.dev/algorithm/mathematical-algorithms/modular-exponentiation-power-in-modular-arithmetic

Garg, N. (2022, February 3). Euclidean algorithms (Basic and Extended). GeeksforGeeks. Retrieved March 22, 2022, from https://www.geeksforgeeks.org/euclidean-algorithms-basic-and-extended/

java - BigIntegers to the power of BigIntegers. (n.d.). Stack Overflow. Retrieved March 22, 2022, from https://stackoverflow.com/questions/4026111/bigintegers-to-the-power-of-bigintegers

BigInteger pow() Method in Java. (2022, February 18). GeeksforGeeks. Retrieved March 22, 2022, from https://www.geeksforgeeks.org/biginteger-pow-method-in-java/

Brute-force search. (n.d.). Wikipedia. Retrieved March 24, 2022, from https://en.wikipedia.org/wiki/Brute-force_search