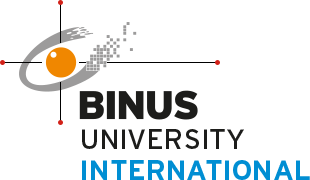
Object Oriented Programming Final Report

****

*Creating an RSA encrypted chat system*

**Lecturer:**

Jude Joseph Lamug Martinez MCS

**Arranged by:**

Joshua Efraim Rawatan

**Object Oriented Programming**

**Computer Science Faculty**

**Binus International University**

**Chapter 1. Introduction**

**1.1 Genesis**

I have always seen the text “Messages to ‘ ‘ are end-to-end encrypted” in my Whatsapp app, but I never really understood what it meant. I only generally understood that it meant my chat with the other person is protected from online attacks. When I started diving into the world of Computer Science, I wanted to know about the world of cyber security which is where I decided I wanted to understand about encryption. From there, I took the opportunity of creating this RSA encrypted chat system as my object oriented programming project.

**1.2 Purpose**

The purpose of this report is to explain this project of creating an RSA encrypted chat system. The main focus is on the security of encrypting and decrypting the messages based on the RSA encryption concept. The goal is to have a fully working RSA encrypted chat system that is able to encrypt and decrypt messages and understand how everything works.

**1.3 Solution Scheme**

Encryption is an important aspect when it comes to handling privacy. When setting up a chat system or app, privacy is crucial to make sure that outsiders are not able to read private conversations. RSA is one type of encryption that can help to handle privacy.

Rivest-Shamir-Adleman or also known as RSA is an asymmetric cryptography algorithm that uses public and private keys. Where the public key is the key that can be shared while the private key is exclusive.

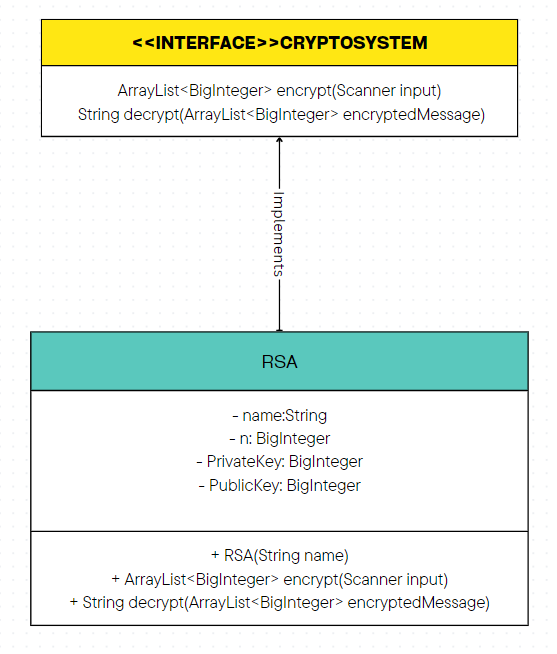
The first step is by generating two large prime numbers which are named as p and q. p and q needs to be in equal length and is then multiplied to get n which is the modulus. Then using Euler’s totient function, phi(n) can be found where phi(n) = (p - 1) \* (q - 1). The use of phi(n) is to find the public and private key. After phi(n) is found, an integer e is generated by the 2 rules where the first rule is that e needs to be more than 1 but less than phi(n) and the second rule is that it needs to be coprime with phi(n). Then, an integer d will be generated by having the rules of when multiplied by e and then modulo phi(n) needs to be 1. Once both e and d are found, the public key is (n, e) and the private key is (n, d).

When wanting to encrypt and decrypt a text, the first step is to convert the text into an integer using ASCII encoding. Once the text has been converted, the formula:

Can be applied to get the encrypted text and to get the decrypted text, the formula can be switched to

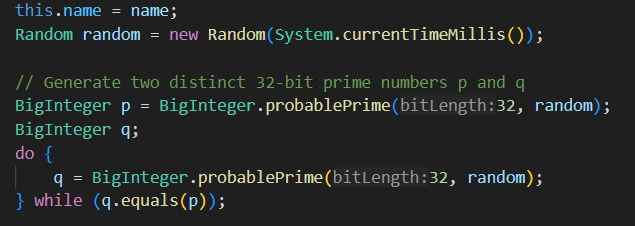
When applied, the encrypted text would get decrypted back to its original integer. The ASCII integer can be reversed to get the original text (GeeksforGeeks, 2023).

**Chapter 2. Class Diagram**



**Chapter 3. Explanation of Code**

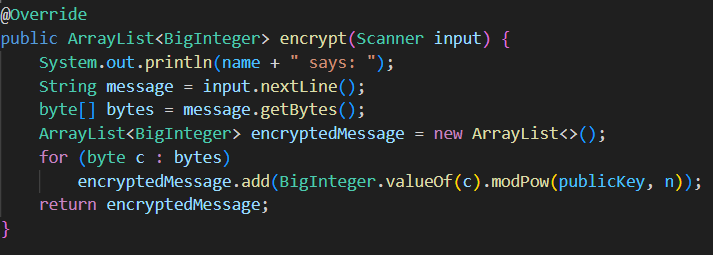
**3.1 Algorithms**



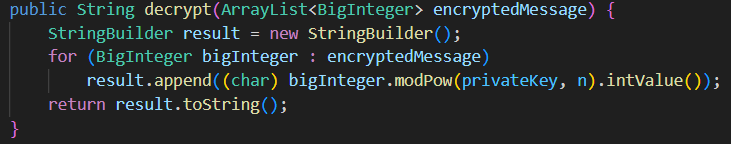
The algorithm above shows how the two large prime integers are generated, first by importing the Random class from Java’s standard library. Using the currentTimeMillis() for generating random numbers to make sure that each number generated will be different. Then the function BigInteger.probablePrime(32, random) is used to generate a 32 digit prime number. Then q will continuously generate 32 digit prime numbers as long as q is equal to p.



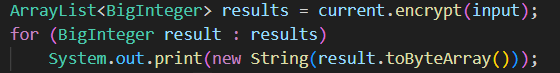
In the code snippet above shows the creation of BigInteger one and also BigInteger phi, where as previously mentioned about the Euler’s totient function that phi is (p - 1) \* (q - 1). Then e is generated where e needs to be coprime with phi and needs to be coprime with phi and is below phi. A prime number with length of 16 digits is continuously generated as long as one of the conditions of (!e.gcd(phi).equals(one) || e.compareTo(phi) >= 0) is true and will only stop when both are false which is what is desired. Then in the next line is assigning n as p \* q. The public key is set as e and the private key is the number that when multiplied with the public key and modulo phi is 1 which is what modInverse does. Then it will print each name with their public key.



The snippet above shows how the message is encrypted. Firstly the method will have an output of ArrayList<BigInteger> from the input given. The program reads the input and turns it into bytes from the .getBytes() function. Then an ArrayList is created to hold each byte/character where each byte goes through the encryption process of getting powered by the public key and then modulus n using the modPow() function. When all the bytes have been encrypted into the arraylist then the encrypted word will be produced.



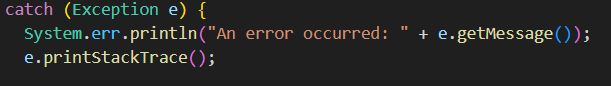
The StringBuilder is initialised as ‘result’ to create the decrypted message. Then every byte from the encryptedMessage is powered by the privateKey and then modulus n using the modPow() function. Then using intValue(), where it converts the BigInteger into its integer value which is then converted into a char to get the character representation from the ASCII integer value. The toString() is used to convert the StringBuilder content to a string.



In the main class, the for loop will print out the encrypted representation of each character where it is turned into a byte array and converted into a string.

System.out.println("(" + (current.equals(Bob) ? Alice.getName() : Bob.getName()) + " reads (after decoding): " + current.decrypt(results) + ")");

This part will print out what the receiver of the message reads, in this case if the current equals Bob, the program will choose Alice and if it doesn’t equal Bob which means it is currently Alice then Bob will be indicated as the receiver. Then the current instance is decrypted to show what the message before encrypted looks like.

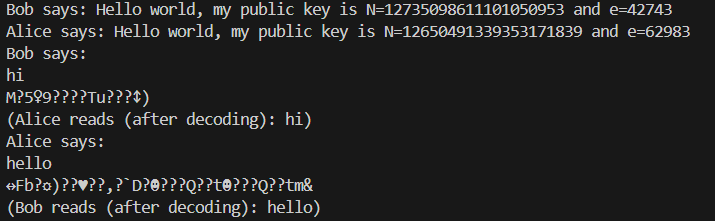


This is the try-catch block where if the program encounters an exception it will be handled in the catch block. It will print the message and the details of where the exception occurred.

**3.2 Data Structure and OOP concepts**

* ArrayList: The arraylist is used for gathering the encrypted message. Where the encrypted representation of each byte is stored in the ArrayList and will later be printed out and decrypted.
* Encapsulation: In the RSA class, the attributes are declared as private fields and methods are public fields.
* Inheritance: The RSA inherits the CryptoSystem interface. Where it follows the characteristics for the encryption and decryption.
* Polymorphism: RSA overrides the encryption and decryption from the CryptoSystem interface while still being able to set the algorithms on how the messages are encrypted and decrypted.

**Chapter 4. Evidence**

****

**Chapter 5. Lessons and Reflection**

I learned a lot from doing this project. I was able to not only learn the nature of RSA encryption but also familiarise myself with the different algorithms that work together to create the RSA encryption. I was able to understand the formula of RSA and understand more about what encryption is. I also used data types like BigInteger and the way it was converted to ASCII and storing it into an ArrayList.

An improvement that could have been done was by making a GUI, this project was mainly about the RSA encryption and not really about the chat, so by making a GUI could help in making the project more presentable and complex with different UML diagrams.

**Chapter 6. Github**

<https://github.com/JoshuaEfraim/OOPFINALPROJECT>

Citation

procodeAL. (2021, May 22). *RSA Algorithm in java (Bob, Alice and Charlie)* [Video]. YouTube. https://www.youtube.com/watch?v=ld3tGvzuvnY

GeeksforGeeks. (2023, November 9). *RSA Algorithm in Cryptography*. GeeksforGeeks. https://www.geeksforgeeks.org/rsa-algorithm-cryptography/