Design Document Project 3 : RSA Encryption/Decryption

Ibrahiem Mohammad and Dhruv Patel

Introduction

This project and its supporting documentation is made by Ibrahiem Mohammad and Dhruv Patel. The project is a RSA Encryption/Decryption tool which is used to create a public and private key pair which is then used to encrypt and decrypt messages, the details of which will be discussed in the sections following. This joint project is for CS 342 at UIC during the Spring 2016 semester. The project version is 1.0.0.

Section 1 - Purpose

The purpose of this project is to create a program which is able to first generate a public key by first selecting two primes and multiplying them together get the first of the key. This in itself is a challenge as the prime numbers will be large unsigned integers, and therefore we need an efficient way of storing them. For now we will do this using a system of dynamic arrays. The second part must be a small exponent that is an integer, not a factor of the first part and must be between 1 and $\varphi(n)$ (which we'll come to later). Now we have our public key.

We then calculate our private key by calculating $\varphi(n)$, and then using that to calculate d, which is our private key. The exact process of the calculations will be explained in detail in the following sections.

So once we have obtained the public and private key, the program must be able to then use the public key to encrypt a given message. The first step is to convert the letter of the message to ASCII values which correspond to those. For larger messages, a more complicated blocking process is involved. The encryption will be explained in detail below as well.

Once the message is encrypted, we must be able to then decrypt the message using the private key. So at this point we have out encrypted data, private key, and public key. Using a modulus calculation (which is explained further below) we get the numbers which correspond to certain letters--which gives us the original message.

The problem we are attempting to solve in this solution is creating a secure method of sending (encrypting and decrypting) messages without a 3rd party being able to intercept and decipher it. We want to ensure security, safety, and integrity. This tool will mostly be used by us to simply test if the solution works--as this is not a commercial product. Although a solution already exists to this problem and is used widely by almost every technological interface around the globe, understanding how it works and building it for ourselves will allow us to better understand the issue of encryption/decryption on a larger scale.

Section 2 - High Entity Design Definitions

Our implementation of the solutions contains many entities which we will describe and define below (*this design is subject to change as we work on the project*):

Key Creation:

- **Key Creation and Storage Class:** This class will be used to create the public and private keys using the mathematical calculations (explained in lower entity information).

Storage:

- **Unsigned Integer Class:** This will hold a dynamic array system which will interpret the prime number, break it down into single digits and store them into an array. It will also be able to piece together all the integers to form the original prime number so that it can be used/represented elsewhere or some other function. This class must also be able to perform the arithmetic operations on the given integer.

Encryption:

- **RSA Algorithm Class:** This class will contain the RSA algorithm itself--which will then be implemented onto the unsigned integer.

Decryption:

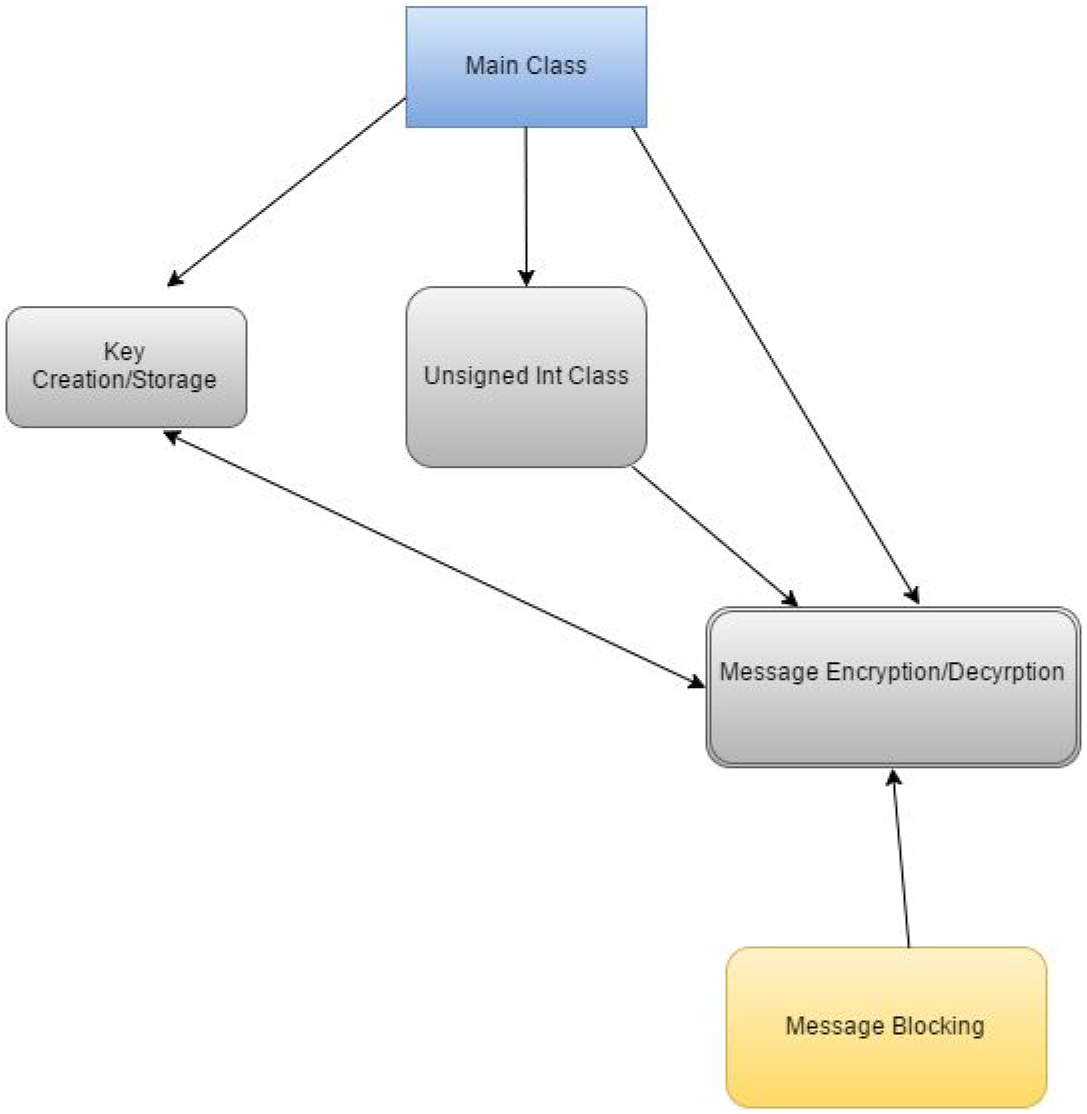
- **Decryption Class:** This class will utilize the private key to then decipher the given message.

Blocking:

- **Message Blocking Class:** This class aids a part of the RSA encryption/decryption in case of encountering a message that is larger

Main Class

This class will be used mainly to call upon the other classes to run the program. It will also include the user input/console output information.



Section 3 - Low Level Design Implementations for High Entities

Main Class

- **Key Pair Object:** This object will call the Key Creation/Storage Class and create a object which contains both the private and public key to be used by other methods/functions within the class. **It will be used primarily by the encryption/decryption object to encrypt and decrypt a given message.** This class will interact with the other objects: the encryption object and through the encryption object, also the message blocking object.
- **Unsigned Int Object:** This object will either create or a key, that consist of the cipher text which can be encoded using the RSA algorithm. This class consist of basic mathematical functions needed for the algorithm. This object will be utilized by the

main class for storing and creating the prime number. It will also contain access modifiers which will let me access the number so that I can use it to then encrypt/decrypt the message itself. This object will interact with only the message encryption/decryption class directly. However, through the message encryption/decryption class it will interact with objects from all other classes.

- **Message Encryptor:** This object will take the given message and apply the public key object to it, and using the RSA Algorithm class (or perhaps without requiring the RSA Algorithm class, and simply using the algorithm within the encryptor class) it will encrypt the message. This object will have to interact with all other objects in order to encrypt. It will interact with the long int object to obtain the message, apply the key object to it, and then if the message is large enough it may need to use the message blocking on it.
- **Message Decryptor:** This object will work as a decryption to decrypt messages. It will be used by the same object and interact with the same object as the message encryptor class.
- **Message Blocker:** This object will be doing the actual crux of the encrypting. It will be used only by the message encryptor but it will need to then interact with the other objects.

Message Encryptor/Decryptor:

- The message encryptor/decryptor objects will do the actual encryption and decryption. Once we achieve the converted ASCII symbolic set of characters corresponding to the message given, we do:

  encrypted data = (ascii converted set)^(private key) mod (public key)
- The decryption will happen as follows:

  (encrypted data)^(private key) mod (public key)
- These objects will then need to interact with the message blocker, unsigned int object, and the key pair object.

Key Creation

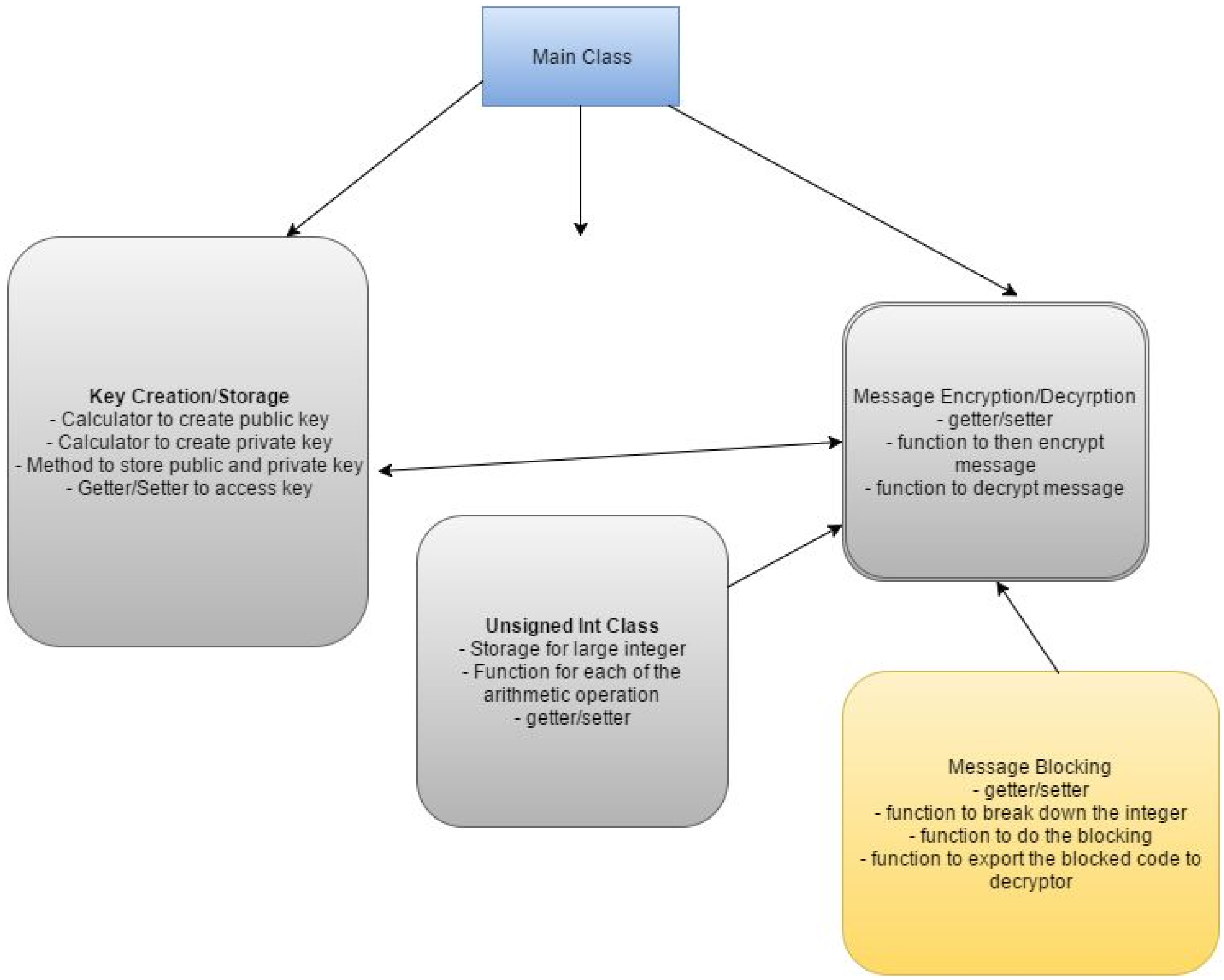
- The key creation works essentially as a calculation and is very easy. The creation itself will not need to interact with any other class because the key creation depends only on the calculation below:

  Select two primes, A and B. Multiply them together to obtain the public key. We then need a smaller integer e which will be an exponent. The other restrictions are that it must not be a factor of n and it must be between 1 and phi(n) -- phi(n) will be explained below. But for now our public key consists of this prime product and this selected exponent integer.

  To get the private key we do the following:

We first obtain phi(n) = (A-1)(B-1). Once we obtain phi(n) we then calculate:
d = 2(phi(n)) + 1/e
where d is the private key and e is an exponent.

This key storage/creation will then need to interact with the message encryptor/decryptor adn through that it will need to interact with the other objects as well.


Main Class
Key Creation/Storage
- Calculator to create public key
- Calculator to create private key
- Method to store public and private key
- Getter/Setter to access key
Message Encryption/Decyrption
- getter/setter
- function to then encrypt message
- function to decrypt message
Unsigned Int Class
- Storage for large integer
- Function for each of the arithmetic operation
- getter/setter
Message Blocking
- getter/setter
- function to break down the integer
- function to do the blocking
- function to export the blocked code to decryptor

●

Section 4 - Benefits/Assumptions/Risks/Issues

Below are the benefits of using the Design:

- The most important benefit of this design is the longInt class, which helps to encode longer message.
- Storing the string as dynamic array makes the element of the array easily accessible.
- Since the entire ASCII table for encoding, we can convert any special characters into a cipher text.

- The main advantage of using this design was the use of inheritance, which makes
- Having a GUI, which helps a user friendly environment for the user.
- Having 3 constructor in the longInt class makes the program and the code for it easier.

Below are the risk that we made for this project:

- The risk by using this implementation would be the runtime, since there are bunch of different computations that have to be done to encode/decode the message.
- The other most important issue we will ran out to will be if the user types or enters a message, which the longInt class cannot handle, which will result in the program to fail.
- The other risk and issue will be to link the keys to the message to encode/decode the text.

Below are the assumptions that we made for this project:

- The assumptions made is the longInt class can handle the length of the cipher message.
- Other assumptions is that the key creation, creates public and private keys using the formula.
- Since the project hasn’t been implemented, we are assuming that the RSA algorithm can work to encode and decode the message.