Design Document: RSA Encryption/Decryption

Paul Nguyen & Dhrumin Patel

Introduction

Our development team, Paul Nguyen and Dhrumin Patel, will be developing a RSA Encryption/Decryption tool. This program creates a pair of public key and private key, and uses them to encrypt and decrypt the message. The keys are generated by two prime values, which will be provided by the user, or randomly generated. Arithmetic operation will be performed to encrypt/decrypt the message. The purpose of this program is to allow the user to keep secret messages between themselves and the select group of people whom they want to send the message to. This program is acquired for Software Design (CS 342) at University of Illinois in Chicago in spring of 2016.

Purpose

The development of this program allows the user to send their messages safely to another specified party. This prevents unwanted users to have access to the file without the proper private and public keys. As a result, we will break down the process into three parts; key creation, blocking, and encryption/decryption.

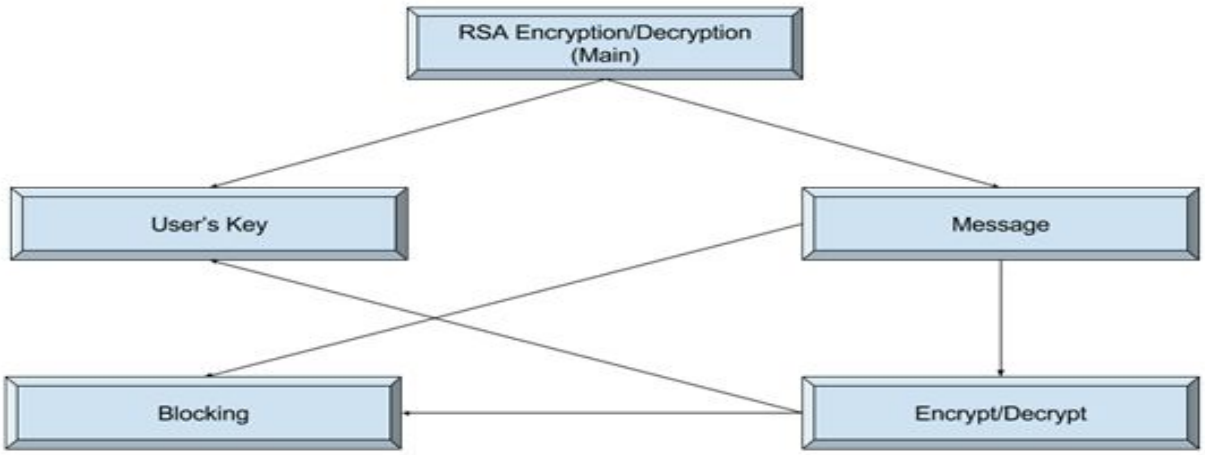
The user will input two prime numbers or randomly generate two prime numbers from a given file set, and compute several arithmetic operations. The prime numbers have to be relatively large, so in our case we will require the user to input a prime number larger than 127. We will also want to consider that this program will be generating large unsigned integers, thus we will be using a dynamic array for our data structure. The output will produce a public key and a private key in XML format, which will be used for encrypting and decrypting. In our case, we will be blocking the input by ten characters per block. Blocking is utilized because we can only compute up to a certain large integer. This prevents us from overflowing on our computation. This program will require us to block and unblock a file. Once we parse the message by parts, we will perform the algorithm on each block. Also, if we wanted to reconstruct the file we will have to do the reverse operation of parsing to get our original message.

With the given keys, the program will encrypt the message. Once the message is encrypted, the only way to decipher the message is by using the public key. The algorithm will consist several arithmetic operations along each ASCII value within the message.

This program provides a method of transfer message in a secure way. Unwanted user will not be able to access the file with the proper keys. Privacy is very important to the public, and we want to have the proper security for these messages. The more complex our algorithms the harder it will be to crack one of these encrypted messages. This program allows us to see and understand how encryption and decryption works in a real life scenario.

High Level Entities

RSA Encryption/Decryption:
This class will act as our main class. It will call other class when needed, allowing us to have easier access through the GUI.
User's Key:
This class will store the store the user's public and private key, which will be generated from the user input of their prime numbers.
Message:
The Message class will hold a dynamic array that holds large unsigned numbers, since we will be using the user's key and break down the message into large unsigned integers.
Encrypt/Decrypt:
This class will use the private key from the user's key class to decrypt the message, and uses the public key to encrypt the message.
Blocking:
This class will perform operations of breaking the message into parts that will be stored onto a dynamic array.



Low Level Design

RSA Encryption/Decryption:

- *Read Prime File:*

  This function will read in the file, which will contain a list of at least 20 prime numbers. We will store the prime numbers into a dynamic array. Upon the call of generate prime numbers, we will grab 2 prime numbers randomly from this array.

- *Find User's Key:*

  This will look up the files where the public/private key will be located. If we can access these files, we will want to store the keys. Otherwise, it will send a flag that we need to generate a new user's key.

- *Check Prime:*

  This function will check if the user input is a prime number. This will scan through all numbers from zero to the prime number and see if any modulus operation has zero remainder.

- *Read Message File:*

  This function will attempt to find the file that we want to either decrypt or encrypt. We will store the message into some sort of storage for bookkeeping.

- *Retrieve Prime Numbers:*

  This function may call the function "Read Prime File," if the user request the program to generate two prime numbers. Otherwise the user will attempt to input two prime numbers, with the help of "Check Prime"

Within this class, it will contain the objects of the "User's Key" and "Message." This will allow us to access all the functions required to operate this program. More detail of these class will be discuss below.

User's Key

- *Key Generator:*

  This function will require two prime numbers to generate our keys. With the given input, it will produce two keys, the public and private key.

- *Key File:*

  This will allow us to create a file, once the keys are generated. The format of the keys will be in XML format.

- *Key Retrieve:*

  If the user already has a key, we will need to access the file from the main class. Then we will transfer it into the proper container of the program by calling this function.

This class is a standalone class, since we do not require any other operations then creating the keys. The keys will allow us the encrypt and decrypt messages by using this class to obtain the key.

Message:

- *Retrieve Message:*

  This function will retrieve a message either from the user, or from file. We will store the string into a bookkeeping container. Later on we will block and encrypt this message.

This class is very simple because it will hold the string of the message, and it will also contain the objects "Blocking" and "Encrypt/Decrypt." The following object will hold most of the computation required to finish the program.

Encrypt/Decrypt:

- *Encrypt:*

  Once the file is blocked, we can encrypt the file. We will prevent the user from trying to encrypt the original message unless it has been blocked. Going through the dynamic array of the blocked file, we will use each ASCII value to calculate the proper encryption.

  Encryption algorithm:

  New encrypted information = $(\text{ASCII})^{\text{Private Key}}$ mod (Public Key)

- *Decrypt:*

  With the encrypted file we will use this function to decrypt the message. The output will return the original message if the user prompts the proper user's key.

  Decryption algorithm:

  Original message = $(\text{Encrypted Information})^{\text{Private Key}}$ mod (Public Key)

This class will use "Blocking" class to unblock the encrypted message when we call "Decrypt."

Blocking:

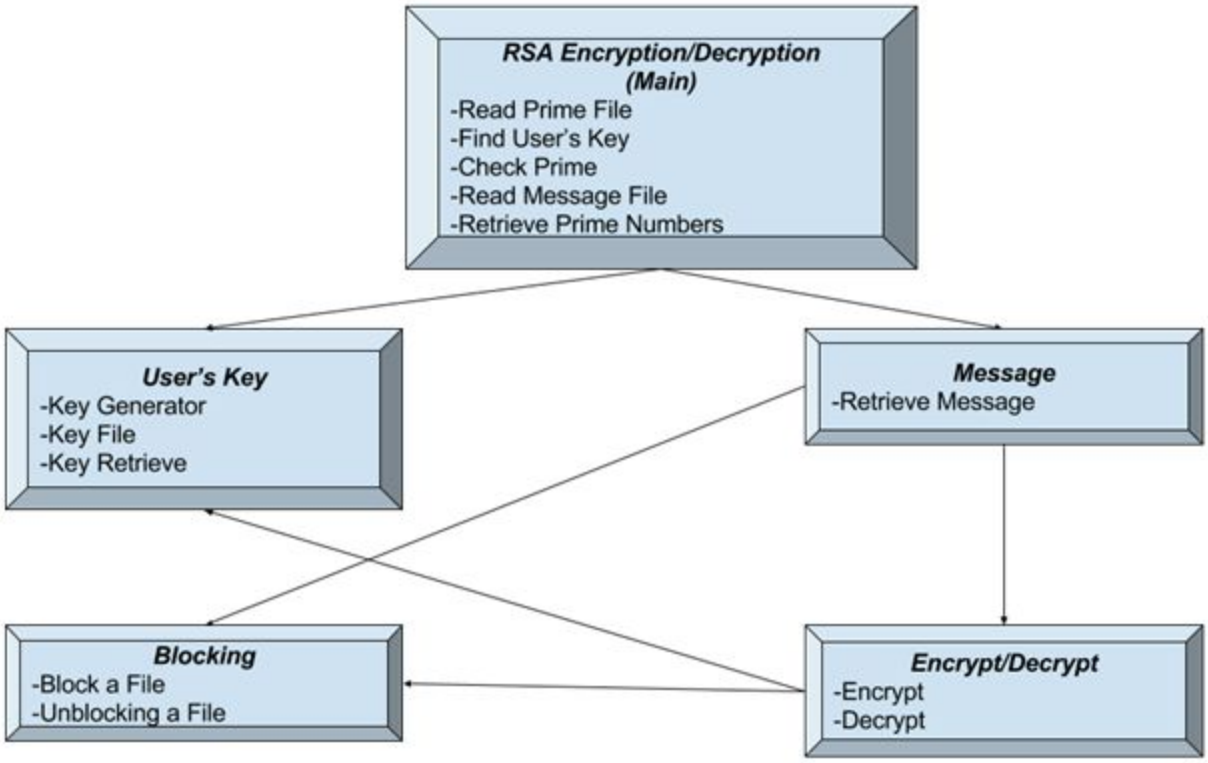
- *Block a File:*

  This function will take the message from the "Message" class, by using the string it will parse it into 10 characters per block. Then we will store this into a dynamic array which will hold each block.

- *Unblocking a File:*

This will do the reverse job of a “Block a File.” It will unblock the decrypted block file, so we would have to decrypt the message first.

This class does not require other classes, since it acts as a function to block and unblock items.



Benefits, Assumption, Risk/Issues

Benefits:

- We will be storing the message into a dynamic array through the program, which should allow us to handle bigger data.
- We will be using typedef of “long unsigned int,” to hold larger numbers for when we encrypt the message.
- Creating each class as a functional object allows us to easily implement the GUI.
- This program will have a friendly user interface by using GUI.
- Bookkeeping will allow us to use each class very efficiently.

Assumption:

- We will use the algorithm to generate our keys through a given formula provided by the director of the class.

- The program will only take large prime numbers greater than or equal to 127.
- Assuming we are handling with large unsigned integer, during encryption phase.

Risk:

- With all the bookkeeping we will be using up a lot of memory to store all the data.
- Runtime with larger data might take a long time, due to several arithmetic operation being performed.