# Networked Chat with RSA Encryption/Decryption

### Due: Thursday, December 7, 2017 at 11:59 pm

*Copyright (c) Patrick Troy, 2017. All rights reserved. The text below cannot be copied, distributed or reposted without the owner's written consent.*

### Group Selection Deadline: Monday, 11/6/17 at 11:59pm

For this project, all CS undergrad students must work in teams of 2 or 3 students. All members of the each team must be currently taking CS 342. Students will be allowed to select team members themselves. Students in the class who are not CS undergrad students may work on their own or in a group of 2 or 3 students. Groups must be selected and reported in Piazza by Monday, 11/6/17 before 11:59pm. The reporting in Piazza must be done as a REPLY to the instructor's original post for Program 4 Group Selection. No restrictions on group membership for this project.

For Project 5, your team is to create a set of GUI based programs in Java Swing that will allow multiple people to connect together and send encrypted messages to a specific connected person.

The Java Socket and ServerSocket classes must be used for the networking implementation. The RSA encryption/decryption code is to be designed and written by the group (using the Java built in security and crypto libraries defeats any learning experience gained so are not allowed in this code).

One program will act as the "central server". Another program will act as the "client program".

All clients will connect to the central server. When sending a message, the client will first send the message to the central server which will then forward the message to the desired client or clients. A clients does not connect directly to another client. We will assume that the central server's network information (ip address and port number) is "well known" by the user of the client program.

Each client program will display a list of the persons currently connected to the central server. As a new person connects to the central server, a displayed list of names at each client needs to be updated to reflect the new person joining the group (or chat room). Also as a person disconnects/leaves from the central server, the displayed list of names at each client need to again be updated. The names used for each person must be unique.

The client is to show a list of messages received and who sent those messages.

Since normal messages can be "stolen" from the packets during transmission, we will use the RSA algorithm to provide Encryption/Decryption of the message. To keep things simple for this project, we will use encryption on a much smaller bit size than is considered "secure" (28 bits as opposed to 1024 bits). This will allow us to get an understanding of the algorithm but not be slowed down by the extreme run-times of the large bit sizes.

# CS 342 – Software Design Project 5 Fall 2017

## The RSA Algorithm

Links to pages that discuss this algorithm include the following.

- http://pajhome.org.uk/crypt/rsa/rsa.html (Note, this page gives a very high level discussion of the RSA Algorithm, but has parts that are unclear/confusing when looking at the details.)
- http://www.di-mgt.com.au/rsa_alg.html
- http://www.geeksforgeeks.org/rsa-algorithm-cryptography/ (Note, this page contains an error stating e must be a factor of n. This should be "e must be a factor of φ(n).)

The RSA algorithm needs three related numbers to work. These numbers will be called e, d and n. The pair (e,n) compose the public-key, while the pair (d,n) compose the private key. To encrypt a value M into C we perform the operation of:

$C = M^e \bmod n$

To decrypt the value of C back to M, we perform the operation of:

$M = C^d \bmod n$

The creation of the values of e, d and n starts with the selection of two different prime numbers p and q. We need to make sure that the values p and q are large enough to properly encrypt the values we will be using. If we would just be using the ASCII values, the range would be from 0 to 127. Thus to encrypt an ASCII value we need values of p and q such that p * q is larger than 127.

The value of n is simply the result of multiplying p times q.

$n = p * q$

To create e and d, we first must create a value φ which is:

$\varphi = (p-1) * (q-1)$

The value of e some arbitrary number that is less than n and relatively prime to φ. There should be many numbers that fit this criteria and any one is valid. A number x is relatively prime of y if they have no divisor in common other than 1 (the term "co-prime" is also used to refer to this relationship). This means the GCD (Greatest Common Divisor) is 1. The GCD of two numbers can be computed via the Euclidean Algorithm.

The value of d can be calculated once e has been determined. The value of d is the inverse of e modulo φ. Which means

$(e * d) \bmod \varphi = 1$

Or for us computationally minded people:

$d = (1 + k\varphi) / e$ ➔ for some integer value k, d is the quotient of $(1 + k\varphi) / e$ when $(1 + k\varphi)$ can be evenly divided by e.

The calculation of both e and d is done via trial and error. You start with a possible value and see if it is valid. If not, you change the value and try again. This is done in a loop which exits when a valid value is found. Some algorithms will use the Xth valid value to try to add a bit more security to the values.

## Message Blocking

The way a truly secure RSA program works is to encode the entire message into a single numeric value. For example, let us say the message is:

> Meet me outside SCE at 10pm.

We could translate this into a single numeric value by summing the ASCII numeric value for each character multiplied by $128^{pos}$ where pos is the character's position in the message. The decimal values for the first few and last few ASCII characters in the message are:

> M – 77 e – 101 m – 109 p – 113 t – 116 space – 32 0 – 48 1 – 49 . – 46

The value of the above message would be calculated as follows (note the message contains 28 characters):

$$77*128^0 + 101*128^1 + 101*128^2 + 116*128^3 + 32*128^4 + 109*128^5 + 101*128^6 + \ldots + 32*128^{22} + 49*128^{23} + 48*128^{24} + 113*128^{25} + 109*128^{26} + 46*128^{27}$$

The problem with the above idea is that the numeric value gets insanely large even when dealing with fairly short messages. So most often, RSA will "block" a message into smaller chunks and then encode each of those smaller chunks as separate numeric values. Note that industrial sized RSA programs may combine up to 100 characters into a single block. Our program will be using block size of 4 (or 2 if you can't get 4 to work). (You may try to push things to a block size of 8 or 16 to see if your program can handle them or not. Actually, if we properly write/use the big-integer class, there should be nothing stopping us from using really big block sizes.)

IMPORTANT NOTE: The prime numbers used for p & q when generating the keys MUST result in the n value that is larger than the maximum number for the message. If we were using a blocking size of 4 characters, the maximum number is somewhere up near $2.7x10^8$ (i.e. $128^4$). This would require the prime numbers of p and q to be at the very least 16411. For a blocking size of 2 characters, prime numbers of 131 or larger would be needed.

Some big prime numbers can be found at:

- **http://www.bigprimes.net/archive/prime/14000000/**
- **http://primes.utm.edu/lists/small/millions/**

When using a blocking size of 4, the 28 characters for the original message gets broken down into 7 message each of 4 characters (note: space characters are shown with underscores).

| | | | |
|---|---|---|---|
| Meet | = | $77*128^0 + 101*128^1 + 101*128^2 + 116*128^3$ | = 163148532 |
| _me_ | = | $32*128^0 + 109*128^1 + 101*128^2 + 32*128^3$ | = 68907680 |
| outs | = | . | |
| ide_ | = | . | |
| SCE_ | = | . | |

at_1 =
0pm. = $48*128^0 + 113*128^1 + 109*128^2 + 46*128^3$ = 102512302

If the length of the original message is not a multiple of the block size, pad the end of the message with null characters (ASCII value 0) so the length become a multiple of the block size.

Note that this blocking operation will also need to be performed in reverse as the final step when decrypting a message. This reverse operation will need to read in a number from each line in the file and break this down into B ASCII characters, where B is the blocking size. Since the NULL character (ASCII value 0) should never be part of the original file and are only used to pad in the last block, if a NULL character is "unblocked", don't write it out to the final decoded file.

## Use of Encryption/Decryption in the Networked Chat and Other Items

When a user first starts up the Network Chat Client, they will need to create a public/private key pair.

When creating the public and private keys, your program is to allow the user the option to either provide his/her own prime numbers for p and q or have the program randomly generate some prime numbers. (So your program must allow for BOTH options. These are the options of the user NOT the options of the development team!) Your program must verify that the prime numbers selected will work with the blocking size of your program. Thus the following condition must hold:

$p * q > 128^{\textbf{blocking\_size}}$

If the user will provide the prime numbers, prompt the user for each prime number and verify that the numbers are actually prime. There are algorithms that can be found via a google search that will verify if a number is prime. Research the internet or a numerical analysis book and INCLUDE A REFERNCE to what you found in your code file and design documentation. Failure to do this will result in a lower score on this project. Please verify that the program/code used actually does work.

If the user wants the program to generate the prime numbers, your program can randomly select the prime numbers from a file that contains at least 20 prime numbers. This file should contain one prime number per line of the file. You are to create this file with appropriate prime numbers, but expect that the user of your program may replace this file with one of their own. Thus the name and location of this file need to be included in the documentation/README file for the program. This type of supplemental file is sometime referred to as a "resource file" so a name like "primeNumbers.rsc" is suggested.

Once the public/private key pairs have been created, the client will then connect to the central server of the chat program. It is assumed that the IP address and the Port number being used are "well known" to the user of the client program and the user will be prompted by the client program to enter in this info.

After the connection with the server is made, the server will need to get the public key information from the user. The server will also need to get a unique name from the client to be used to identify this client from the other clients using the system. This name may be restricted to some reasonable length.

The server will need to push the names and public keys of all existing clients back to this new client. The server will need to push the name and public key of this new client to all of the existing clients.

When a client disconnects from the server, the server must inform the other clients so each client can remove that clients name from each "Connected Client List".

You must add the traditional About box and Help box(es) to your program.

## Design Patterns and the Supporting Documents

Your program is to use Design Patterns whenever possible. Please highlight there use in the Supporting Documents assignment related to this programming project. Ideas for Design Patterns might include:

- Singleton Design Pattern for GUI items
- Singleton Design Pattern for Shared Data Items across Threads
- Observer Pattern for addition and removal of Clients
- Decorator Pattern for the frosting on the cupcakes to be given to the instructors (just seeing who is paying attention to what they are reading)
- Factory Pattern for the creation of Socket Message Types
- Adapter Pattern when team members don't create parts that seamlessly work together

The Supporting Documents Assignment is to follow the same Design Document Guidelines as done for Project 3. Please note that this document is due before the code. Only one Design Document is required per group. The Design Documents are due on Tuesday, 11/28/17 at 11:59pm.

## Programming Style

Your program must be written in good programming style. This includes (but is not limited to) meaningful identifier names, a file header at the beginning of each source code file, a function header at the beginning of the function, proper use of blank lines and indentation to aide in the reading of your code, explanatory "value-added" in-line comments, etc.

The work you turn in must be 100% by your group. You are not allowed to share code with any other group (inside this class or not). You may discuss the project with other persons/groups; however, you may not show any code you write to another person/groups nor may you look at any other person's/group's written code.

## Project Submission

You are to submit your program via GitHub following the directions provided by the TA.