Final Project
RSA Encryption

by: Jackson de Gruiter

How RSA works

I. Generate public & private key

II. Public key (available to anyone) encrypts information

III. Private key (available only to us) decrypts decrypt that information

Public Key Generation

I. Choose 2 random large prime numbers, **p** & **q**

a. p & q should both be at least 300 digits long

b. they should be randomly generated

II. Multiply **p** & **q** to get **n**

III. Choose a small number **e** which is coprime to **m**

IV. Variables **e** & **n** will serve as our public key

Private Key Generation

I. **m** will be equal to $\Phi(n)$ or $(p - 1) * (q - 1)$

II. Find a number **d** such that **d***e* mod ($\Phi(n)$) = 1

III. Variables **m** & **d** will serve as our private key.

Encryption

1. Convert ascii text into sets of numbers

2. Convert those numbers into a digit

3. Optionally, pad the new number with random digits at the end in a way that can still be decrypted easily, but cannot be brute forced

4. Apply the number to the following function:

$$\text{cipher} = (\text{message})^{e} \bmod \mathbf{n}$$

Decryption (not implemented)

I was unable to implement decryption due to the fact that I was unable to generate a number **d** as descried earlier. This may be because of a misunderstanding or perhaps misuse of the GMP library.

I. To undo the cipher, apply the following function:

   message = $(\text{cipher})^{e}$ mod **n**

II. Convert the resulting number from a decimal number into a string by doing the inverse of what we did to make the message into numbers

My Implementation: Program requirements

Requirements:

- Enough memory to store incredibly long numbers
- The GMP library

My Implementation: Files

Program Files and Folders:

- generatePrimes.cpp // Generate p & q, store in ./tmp/primes.txt
- publickey_gen.cpp // Generate n & e, store in ./keys/publicKey.txt
- encrypt.cpp // Prompt for message, then print cipher text
- Extra/ // Folder contains work not required by project proposal
- Extra/encrypt_strong.cpp // Uses random number padding for improved security
- Extra/encrypt_weak.cpp // Same as encrypt.cpp, doesn't pad number
- Extra/private/ // Folder contains my attempt to generate a private key for decryption
- README.txt // A readme file containing usage instructions for the program

My Implementation: Usage Overview

- Compile all .cpp files in the base folder, following instructions contained in README.txt
- Generate prime numbers (./generatePrimes)
- Generate public key (./publickey_gen)
- Encrypt a message (./encrypt)

What I’ve learned

- How RSA works
- Why implementing RSA myself is a bad idea
- How to do math with huge numbers
- How to go about debugging external library problems
- How to work with C strings

What I would do differently

- Not try to implement fast modular exponentiation myself
- Should have learned more about RSA before writing code
- Implement more secure encryption by default
- Use Unicode instead of ascii for encrypted emojis

What I did right

- Finding a library that could deal with large numbers
- Making backups frequently (rudimentary version control)
- Lots of YouTube videos