This project is worth up to 40 points of extra credit in the assignments category. Use any programming language of your choice.

1 Introduction to RSA Encryption

In cryptography, encryption is the process of encoding a message or information in such a way that only authorized parties can access it. RSA encryption is a ubiquitous method, used often in industry, which is an example of public key encryption. The security of RSA hinges on the idea that finding large prime numbers is easy, but factoring large composite numbers is hard. While the idea is simple, the encryption itself is incredibly robust; a modest 512 bit integer key could take years to factor on a modern supercomputer–that's a tough code to crack! In this project you will be constructing an RSA encryption/decryption system and seeing if you can crack it.

2 Large Integer Arithmetic

An integer in C/C++ is typically 32 bits, of which 31 can be used for positive integer arithmetic. This is good for representing numbers up to about two billion (2 times 109). Some compilers, such as GCC, offer a "long long" type, giving 64 bits capable of representing about 9 quintillion (9 times 1018)

This is good for most purposes, but some applications require many more digits than this. For example, public-key encryption with the RSA algorithm typically uses 512 bit numbers, and it is not uncommon to see 2048 or 4096 bit integers among the more paranoid types.

Task: Build an arbitrary integer arithmetic system

Use an array object to create a new type of number with arbitrary length. For example, you could create an integer array where each element of the array represents a single digit of a larger number (of course this is terribly inefficient with memory). See if you can construct it is such a way that memory is not wasted.

Equip this new integer type with three operations: addition, multiplication and mod (%). If you are working in C++, you might want to use operator overloads. Now you have a framework for performing arithmetic on arbitrarily large integers!

3 Prime Number Sieve

As mentioned above, doing RSA encryption is based on the idea that finding large prime numbers can be done quickly. In class we saw how the Sieve of Eratosthenes can generate lists of prime numbers. While it is neat, the Sieve of Eratosthenes is a bit outdated (∼250BC). Today, wheel sieves are much more popular due to their superior efficiency. See the wiki page on wheel sieves

Task: Generate a list of primes

Build a wheel sieve to generate a list of primes up to some limit. You will have to do some investigating on the web and find an implementation that you like. Use your arbitrary integer type so that your sieve can generate arbitrarily large primes.

4 The Cipher

We saw in class how we can use the binary representation of an integer to perform fast modular exponentiation. For example:

$$\begin{aligned} 5^{13} \mod 7 &= 5^8 \cdot 5^4 \cdot 5^1 \mod 7 \\ &= (5^8 \mod 7)(5^4 \mod 7)(5^1 \mod 7) \mod 7 \\ &= 4 \cdot 2 \cdot 5 \mod 7 \\ &= 40 \mod 7 = \mathbf{5} \end{aligned}$$

Now you have everything you need for RSA encryption. Remember we can encrypt a message M with

$$C = M^e \mod n$$

and decrypt the cipher C with

$$M = C^d \mod n$$

Task: Generate your keys
Use the list of primes you generated to select two large prime numbers p and q. This will give you your encryption modulus $n = pq$. Find an integer e such that e is relatively prime with $(p-1)(q-1)$. Use Euclid's algorithm to prove that $\gcd(e, (p-1)(q-1)) = 1$. Finally, find d, an inverse of e modulo $(p-1)(q-1)$

Task: Put it all together
Implement a function that will perform fast modular exponentiation using your arbitrary integer type and use it to implement the RSA encryption/decryption method detailed above.

This process will involve encoding your message M as an integer (a simple ascii encoding is fine). For large messages M, you will need to use block encryption. Why is this? What is the maximum length of message for a given modulus n?

5 Cracking the Encryption

There are lots of algorithms out there for factoring large integers, some more efficient than others. Do some investigation on the web to find one that you like and implement it. Then send your instructor an email. We will respond with an encrypted message and a public key. See if you can factor the key and decrypt the message for extra credit on the final exam! (you will need something more efficient than trial division)

6 Evaluation

Submit your code and a short writeup to your instructor via email. In the writeup explain your implementation and how to use it. You will be graded individually depending on your effort and progress. Learning is about pushing your boundaries, not just meeting some standard. Your evaluation will come from a written report that you submit, an interview to ask any follow-up questions, as well as a qualitative judgment of your effort level determined by the conversations we had during your work and how long you spent on the exercise.

Task: Reflection
What did you learn from this exercise? What parts did you enjoy most and least? What were the major challenges you had to overcome? What do you wish you had done differently? How long did you spend on this project?