Fall 2021

Section 1: TTh 3:30pm - 4:45pm - JFSB B092 (changed from MARB 130)

Project 5: RSA

Objectives

In this lab you will develop a fully functional implementation of RSA. You will gain experience with finding multiplicative inverses and more practice with modular exponentiation.

Background

In the Diffie-Hellman lab, you wrote your own modular exponentiation function. We'll be using it a lot in this lab.

RSA encryption and signing

Encryption and decryption are just a matter of taking your message or ciphertext m (less than n) and computing $m^e \mod n$ or $m^d \mod n$. n is called the public modulus, e the public exponent and d the private exponent. n and e together form the public key. n and d together form the private key. e is currently conventionally set to 65537 because it's cheap to compute as a modular exponent.

Encryption and decryption work because $(m^e)^d = m^e d = m(\mod n)$. e and d are chosen so that their product is equal to a multple of $\phi(n)$ plus 1, that is to say: $e * d = 1(mod \phi(n))$, or $e * d = k * \phi(n) + 1$ for some (unimportant) value of k. See the slides on RSA for details of why this is.

Given $(m^e) \mod n$, it's hard to compute m unless you know $\phi(n)$ (in which case you can calculate d). Computing $\phi(n)$ is hard unless you know the factorization of n. To ensure that factoring n is hard, we make it the product of 2 large random primes.

Signing is just encryption with the private exponent (what we normally consider decryption). The receiver can “decrypt” the signature with the public key to and verify that it matches the document. Since only the owner of the key knows d, only she can sign documents.

Calculating an RSA keypair

e is conventionally set to 65537 (as long as it is relatively prime to $\phi(n)$), so we just need to compute d and n. As we mentioned, n should be selected as the product of two large primes, p and q. To compute d, recall that it must be chosen such that $ed = 1 \mod \phi(n)$. But this just means that d is e's multiplicative inverse modulo $\phi(n)$. $\phi(n)$ is just $(p-1)(q-1)$ since n only has two factors, p and q. We can use the extended Euclidian algorithm to compute modular inverses.

Euclid's Algorithm

To test whether two numbers are relatively prime, you can use Euclid's Algorithm. You will find pseudocode for several implementations on the Wikipedia page.

Generating primes

For the purposes of this lab, you are free to use any available resource for generating your primes. OpenSSL provides a way to generate primes, and many programming languages (such as Java) have ways of generating large primes. This should be familiar to you from the Diffie-Hellman lab.

There are several tests which are used to find large prime numbers. Most of them are probabilistic; they'll tell you if a number is certainly composite, but can't tell for sure whether a number is prime. The Miller-Rabin test is probably the most respected of the probabilistic algorithms. OpenSSL uses Miller-Rabin primality testing to generate it's primes.

Computing multiplicative inverses

The extended Euclidian algorithm is often used to compute multiplicative inverses modulo some number. In this lab you will implement it to find d. Remember that the method for computing the d by hand is somewhat different than what you will want to do on the computer. The Wikipedia page on the extended

Euclidean algorithm has pseudocode for computing modular inverses that may be useful.

Requirements

Restrictions: You may use the language of your choice for this lab. You may use a bignum library as in the Diffie-Hellman lab. You may not use any built-in modular exponentiation, multiplicative inverse, Euclid's algorithm, etc., but you may use code to help you generate your prime numbers p and q.

- Generate 2 512-bit primes p and q. Ensure that their high order bit is set. Verify that $(p-1)(q-1)$ is relatively prime to 65537 (which we will be using for e). If it isn't, choose new p and q values.
- Using $n = pq$ and $e = 65537$, calculate the secret exponent d, such that $ed = 1 \bmod \phi(n)$. Verify that for numbers m less than n, $((m^e \bmod n)^d) \bmod n == m$.
- On the passoff webpage, you will be given a value to encrypt. Use modular exponentation ($m^e \bmod n$) to encrypt the value.
- On the passoff webpage, you will be given a value to decrypt. Use modular exponentation ($m^d \bmod n$) to decrypt the value.

Passoff

There is a passoff website

Submission

Submit a zip or gzip file that contains:

1. Your source code.
2. Instructions for compiling (if needed) and running your code.
3. Screenshots of your successful submission to the website, including every step.