Ileana Palesi

Cryptography

Dr. Briscoe

5/8/20

Lab #4 Report

The code included in the RSA\_brute\_force python file is very basic and not a very efficient way to factor semi-primes for a given N. For this reason, I sought out a better (meaning more efficient) factorization of RSA semi-primes. The factorization algorithm I settled upon was that of Fermat’s which was briefly mentioned in the Chapter 8 lesson slides. Named after named after Pierre de Fermat, the method he devised is based on the representation of an odd integer as the difference of two squares. For an integer N, it seeks to derive prime factors a and b such that:

N = a2 - b2 = (a+b)(a-b), where (a+b) and (a-b) are the factors of N. I converted a basic algorithm for doing so to Python code to visualize how it performs in comparison to the basic factorization code. I also printed execution times for both factorization algorithms to determine how much more efficient Fermat’s algorithm is by comparison. I found that for 20 bits used in the prime numbers, the non-efficient given factorization code took 0.030998706817626953 seconds to execute, whereas the basic Fermat factorization implementation took 0.011933326721191406 seconds to execute. To further quantify the difference, I changed the number of bits from 20 to 30 and found that the given code took 18.044116497039795 seconds to execute whereas the Fermat code took a much less 5.164040803909302 seconds to execute. Thus, while this is only a basic implementation of Fermat’s factorization code and it has been known to be improved upon a number of ways, including but not limited to a combination of Fermat’s method and trial division, the Sieve improvement, and the Multiplier improvement, it is evidently a brute force improvement on the given factorization method when it comes to factoring larger prime numbers.



