Max Tiriobo

MTH 330: Cryptography

12/12/2025

RSA Implementation Explanation

For my final project I chose to implement RSA with OAEP padding. RSA is a key encryption algorithm that uses public and private key encryption to encrypt data. The encryption is done with the assumption that finding large primes in prime factorization is hard. So in order for an attacker to obtain the secret message, they would need to factor a very large prime number. In my python code, I used a hierarchical system in which each class uses functions from the previous. In order, these classes are, Math Functions, Prime Generation, Key Generation, OAEP padding, String to Int Converter, then an RSA encryption class itself. I originally started with simply defining functions in the order that they were used and while there can be organization with comments, it can be very easy to fall into disorganization. I designed it this way because I wanted to keep readability and scalability available.

For the generation of large primes, I created a function, generate prime, that takes an int, bits, as a parameter and returns a random number using a CSPRNG. This is used in generate_rsa_primes, which uses Miller Rabin testing to verify if the number is prime. After the generation of primes, both the private and public keys are generated using these prime numbers. In Key Generation, both n and phi(n) are calculated to find e for the public key and d for the private key. Key generation returns the keys, (e,n) and (d,n).

OAEP padding was definitely the hardest to understand because of how much studying and understanding I had to go through before the implementation. The turning of m to em turns

the message from just a message to hashed function and returns a string with 0 padding, a key and a data block at the end. The hashing relies on the hash length, which in the case of SHA 256, is 256. After padding the message, em would look like 000..00 || maskedSeed || maskedDB → which the DB looks like: 000..000 || 00 01 || M. This is the actual message that would be encrypted with RSA.

There were free problems that arose throughout this project. Firstly, the message m cannot be larger than n, part of the public key. This was harder to fix on my local rendition because long messages turned into integers would result in very large numbers. What I did to fix this was to create an error message that made sure that did not allow encryption if the message m was larger than n. Another problem I ran into was that padding was done after the encryption. This was done in the wrong order. After I fixed the order, the encrypted OAEP could be encrypted no matter how long the message was.