Anish Laddha
Professor Avinash Kak
ECE404
February 19 2024

Homework 5

Problem 1: rsa.py

In rsa.py, I implemented both encrypt and decrypt, as well as key generation.

To encrypt, I first read the p and q keys from the specified files, and calculate that n = p*q. Then I pad the vector so that I can proceed to iterate through 128 chunk blocks. I then convert that 128 bit block to an integer, calculate c = (m^e) mod n. Then I convert that to a 256 bit bitvector and write it in ascii.

To decrypt, I calculate n = p*q, and also the totient of n = (p-1)*(q-1). Using the totient, I calculate d to be the multiplicative inverse of e mod totient n. I also calculate the inverses of q and p, mod p and q respectively. Using that, I calculate xp and xq which are just themselves times the multiplicative inverse. Then, iterating through 256 bit chunks, I first extract the integer from the bitvector, calculate (C^d) mod p and q, then do (vp*xp + vq*xq) mod n to get my msg integer. I convert that to a 128 bit bitvector, and then write that as ascii to my file.

For generating primes, I use the prime generator algorithm to find 2 primes, check that they are good primes, and if they are, to write those primes into the files. I check if they are good primes by making sure they are both not equal to each other, that the last 2 bits are set, and that they are coprime to e.

Problem 2: breakrsa.py

To encrypt I first generate 3 sets of primes instead of reading from a file, and then encrypt them the way I did in rsa.py except I write all 3 public keys (n) to a separate file.

To decrypt I use the Chinese remainder theorem as mentioned by Prof. Kak in Page 98 of Lecture 12, to go block by block and decrypt every chunk. Using all 3 cipher chunks and their keys, we can find M3 mod N, and then use the solve_proot function to find m. Then we write that to the decrypted file.