Joonyeoup Kim

ECE 40400

HW06 Description

2/27/2024

1. For problem 1, we were required to implement a 256-bit RSA algorithm to encrypt and decrypt a given text in message.txt with a given value of e, p, and q. For encryption, I would first calculate the value of n by multiplying the values of p and q. Then, I would convert the given plaintext into a BitVector. As long as there are more BitVector to read, I would read 128 bits of the BitVector, and pad zeros to the left of those 128 bits. If the padded bits' length was less than 256 bits, I would pad zeros to the right till it is 256 bits long. For every block of 256 bits, I would find an integer value that represents those bits, and calculate M^e, where e is given. Then, I would mod the value by calculated n, and convert that integer value into a BitVector with the size of 256 bits. I would transform the BitVector into the hexstring representation, and append it to the ciphertext file. For decryption, I would first read in the values of p and q, and calculate n by multiplying those two values. I would also find the totient of n by multiplying (p-1) and (q-1). Then, I would find a multiplicative inverse of the value e to the mod of totient of n by converting e into a bitvector and using the multiplicative_inverse function and store it to d. After all the calculations of the parameters, I would convert the ciphertext that is written in hex into binary and store it into a separate file "new_cipher.txt". I would then convert the file "new_cipher.txt" into BitVector. As long as there are more bits to read from that BitVector, I would read in every 256 bits. If the length of the bits are less than 256 bits, I would pad zeros to the right till it is 256 bits long. I would convert each blocks of 256

bits into an integer value, which would be stored as C. We could have found C^d and moded it with n, but that would've taken too much time. So, I used Chinese Remainder Theorem to find the value instead. I would first find Vp and Vq by finding C^d and mod that valye with p and q. Then, I would find the multiplicative inverse of q in regards to mod p using the MI function, and find the multiplicative inverse of p in regards to mod q. Then, I would find Xp by multiplying the MI of q and q. I would find Xq by multiplying MI of q by q. Then, I would multiply Vp and Xp and add the value with the multiplied value of Vq and Xq. I would then mod that value by the calculated n value and name it to be M. I would store M into a BitVector with size of 256 bits. And, since we padded 128 bits of zeros in the encryption step, I would read the last 128 bits of that BitVector and write it out to the decrypted file as ascii representation to get the decrypted text.

2. For problem 2, part 1, we were given a value of e to be 3. Then, we were required to encrypt the plaintext into 3 different ciphertexts all using different values of ps and qs and store them along with the value of n given by those different keys. I first created a function to generate keys. This was done by using the PrimeGenerator function given by professor Avi. I would first create a random prime number with the length of 128 bits using PrimeGenerator function. Then, as long as the gcd of e and the value of (p-1) (q-1) was 1, I would write the values of p and q into a text file. For encryption, I would generate 3 different ps and qs and store them into files P1, P2, P3 and Q1, Q2, Q3. Then, I would calculate the values of n1, n2, and n3 by multiplying P1 and Q1, multiplying P2 and Q2 for n2, and P3 and Q3 for n3. Then, I stored the values of n1, n2, and n3 into a text file. After calculating all the parameters, I would convert the given plaintext into BitVector. And, as long as there are more bits to read from the BitVector, I would read in

128 bits. If the length of the BitVector was less than 128 bits long, I would pad 128 – length of the BitVector of zeros to the right. Then, I would pad 128 bits of zeros to the left. I would convert the value of each block into an integer value and call it M. I would find C1, C2, and C3 by powering M to the given e and modding it with each n1, n2, and n3 values. Then, I would convert C1, C2, and C3 into a hex file and store it into file enc1.txt, enc2.txt, and enc3.txt. The second part of problem 2 was to decrypt/crack the encrypted file after knowing only e and ns. First, I would read in the values of n1, n2, and n3. I would then calculate the product of n1, n2, and n3 and store it to N. I would then convert the hexfiles of encrypted file C1, C2, and C3 using the same method in problem 1. As long as there were more bits to read in each C1, C2, C3's BitVector, I would read in 256 bits of each BitVector. If the length wasn't long enough to be 256 bits, I would pad zeros to the right to fulfill the length requirement. I would convert each C1, C2, and C3 into an integer value and store it as c1, c2, and c3. Then, I would calculate M1 by multiplying n2 and n3, M2 by multiplying n1 and n3, and M3 by multiplying n1 and n2. I would find the multiplicative inverse of M1 in regards to mod n1, MI of M2 in regards to mod n2, and MI of M3 in regards to mod n3 and store it as invM1, invM2, and invM3. Using Chinese Remainder Theorem, we know the value of M^3 = (C1 * M1 * M1^-1 + C2 * M2 * M2^-1 + C3 * M3 * M3^-1). So, I would calculate the value of M^3 using that equations and the calculated values. Then, using the function solve_pRoot given by Professor Avi, I would find the cube root of M^3, successfully getting the value of M, the original plaintext. I would then store the value of M into a BitVector with the size of 256. For the same reason that I mentioned in problem 1, I would only read in the last 128 bits

of the BitVector and convert such BitVector into ascii representation. I would then write out the converted ascii text into the output file.