Ethan James
33378430

HW06

Problem 1:

To implement RSA encrypt, I first had to come up with a function to generate P and Q. I first used a prime number generator from the professor that took 128 bits. Then in the while loop that I had created, I found a random P and Q value. I first checked to see if the two MSB were set. Then I created an if checker that checked to make sure the p and q weren't the same numbers, make sure both their MSB bits were set, and make sure the gcd of p and q with e was 1. If all this was satisfied, we have found our P and Q, otherwise the function kept running until this was satisfied.

Then I begin working on the encryption portion. It was pretty simple. I first converted the input message to a bit vector. I also got my p and q values from the file given. I also found my modulus which was p multiplied with q. Then I read it in 128 bits and padded it from the right. I also padded from the left as indicated in the document. Then I found my encryption modulus essentially. I then output this to the file.

Finally, I worked on decryption. It was also pretty straightforward. I first open my encrypted text and read it in as a hex string in Bitvector. I also obtained my p and q values. I get my modulus and totient. Then I convert every value I have up to this point into Bitvector form. I then find my XP and XQ using multiplicate inverses as per the notes given by AVI KAK. Then we start decrypting. I got a block. I found my VP and VQ using the formula given in the notes from AVI KAK. Then I multiply my bv*VP and add it to my VQ*XQ. Then I take the modulus of this using the modulus I had calculated. Now the block is decrypted, and we have that chunk of bits decrypted and can now write it to the output file.

Problem 2:

To implement break RSA, I first must write the encryption portion. To do this I had to generate three keys. I first use my generate P and Q function to figure out those values. Now onto the encryption function. I open each of the three output files first. Then I get my p and q for the first key. I also obtain my modulus and totient. I got my d value; however, it was not needed. I then call my actual encrypting function. This function opens the plain text as a bitvector. I recalculated my modulus, but I am now realizing there was no reason to this and that I could have just passed it in. Then I read a block and pad from the right. I then do basically what my encryption did in problem one. I repeat all these steps for the other two keys. I output my modulus to a file and my encrypted text using the three keys.

The harder function was the crack function. First thing is I obtain my three moduli as an int and multiply them together to create one big modulus. I also read each encrypted file in Bitvector as a hex string. Then using TA help, I came up with a function that multiplies the modulus2 and 3 together. I then take the multiplicative inverse using the first modulus and convert it to an int. I do this for the other two modulus also. This will help with the Chinese Remainder theorem later in the function. Once I get into actually breaking RSA, I enter my for loop. We get a block of bits from each encrypted file and convert it to an integer value. Then I go ahead and use the Chinese remainder theorem formula which was given to me by a TA and partially derived by me from AVI KAK 11.7. Essentially the formula I use is the multiplicate inverse found earlier multiplied by the second and third modulus which is then multiplied by the ciphertext. I add this to the math done on the second cipher block and third cipher block. Math varies slightly for each block as we use different moduli values. Then I found the cubed root using the

formula given to me by AVI KAK. This gives me the plaintext. I then convert it to a Bitvector so that it can be written to an output file. However, the bitvector has 128 bits of 0 on the left and 128 bits of what I want to output on the right. I divide this into two halves and only write the right half to the output file for each iteration.