RSA explanation:

For encrypt I read in the p and q vars from files and made n = q\*p. then I read from the bitvector formed from the plaintext file 128 bits at a time. I padded 128 bits to the right, and padded more if the chunk size after that was less than 256 to account for the end block. Then I raised that chunk to the power of self.e, modulus n, and the output was the result of that operation, written to the output file 128 bits at a time. A computer screen shot of a program code

Description automatically generated

Decrypt was very similar. I got the var d = e^-1 \* (p-1)\*(q-1), and recovered the plaintext by taking 256 bit chunks at a time, raising them to d, modulus n. then I cut it back down to 128 bits for the output BitVector, and wrote that to the output file to re-form the plaintext.

A screen shot of a computer code

Description automatically generated

CRT to break RSA:

I first read in the moduli from the n\_1\_2\_3 file and multiplied them together to get the final large modulus N. Then I made an array N1N2N3 to hold N // Ni where Ni was one of the individual input moduli. Then I made array Ni\_inv where I got the mult inverse of N1N2N3[i] in mod n. This prepared me for the loop where I iterated through the 256 bit input blocks. I iterated through all the input files at the same time using start and end block variables and splicing the bitvectors. Then I would make each chunk, for example: chunk1 = chunk1.int\_val() \* N1N2N3[0] \* Ni\_inv[0]. The other 2 were made likewise, then I added together the 3 chunks and modded by N. Then I took the cube root, and casted to a bitvector of size 128. I would then write that bitvector to output file in ascii, so the output was written 128 bits at a time.

Loop contents: A screen shot of a computer screen

Description automatically generated