RSA/AES, Bryce Kurek

General:

In this scheme I use the Boost multiprecision library to define my own signed int datatype int_t which holds integer values up to 4194304-bits in length, since a lot of variables need to be this large for RSA. the size of the stack reserve size is 134217728.

Primes in the scheme are coded in and can be replaced with any prime number or other value respectively. And the only way that I have found to initialize variables to a number greater than long long is to cast them as an int_t as a string, for example.

int_t key = int_t("p"); // where p is any number

The code for RSA, and AES are located in their own header files, and a separate driver .cpp to run/test them.

RSA ("RSA.h"):

First the RSAGen function should be run to get the values for n(public key), e, and d(private key), and also takes a parameter bool limiter. The limiter exits to make this run in a reasonable amount of time (albeit with significantly smaller p, and q), unfortunately I can't run p, and q at 256-bit. However, the stack reserve size, and the int_t bit length above should be enough.

Other than that, this RSA should be similar to the lecture, calculating n, and phi, though e is the lowest number with a gcd equal to 1 with phi. And d is the modular inverse of e, and phi. With the RSAGen calculation run the encrypt and decrypt functions are ready to use, and take in an int_t as the message along with e, and the public/private key

AES ("AES.h"):

This may be a complicated implementation. AES.h almost exclusively uses bitset variables, encrypting a message character array of 32 characters, the keys are calculated by the createKeys function into 15 -for each round- 8x4 -8*4*8 =256- bitset matrix (just a multidimensional array), similarly the message is converted into an 8x4 bitset matrix. Where each bitset arrays cell holds a 8bit value.

Each piece of the encryption modifies the 8x4 cipher matrix. So we have functions for byte substitution, shift rows, mix columns, and key adder, and their inverse functions which just do the opposite of their original, except key adder which is its own inverse.

One major difference is I was not able to use the sub byte function for encryption or decryption for most of the process, I did get to use it however slim, but I did get it working with 14 rounds of encryption/decryption.

Driver ("RSAAES.cpp"):

First, the key is split into 64 4bit blocks for encryption a decryption using RSA to keep memory use down. This is meant to represent the exchange of keys, so the key used for AES is encrypted using RSAs public key and decrypted with the private key d. The message can then be encrypted and decrypted using the shared private key, named "redKey" in the code.

*note: second submission, included key gen – from Boost Library - for unlimited, and square&multiply algo, did not improve performance noticeably.