Tommy Olney

Cryptography and Network Security 1

Final Project Write Up

Project Description:

For this project, Ryan Stillings and I decided to create a peer-to-peer messaging system, meaning that the creation of a dedicated server was not needed. To do this, we had to design the system in a way allowed for each client to act as either the client or the server. We did this by implementing a single main file that allows for incoming connections, outgoing connections, and interaction with our GUI. In our program, when two people want to establish a connection, one client sends a request to the other one, where the client receiving the request acts as the server (meaning that client receiving the request also decides which cryptographic systems to use, if the reverse was implemented, the program would be exceedingly easy to break.), and the sender acts as the client to the server, meaning that our network is truly peer-to-peer.

As briefly mentioned before, our program (and therefore whichever client is acting as the server) allows for multiple encryption and key distribution schemes. Specifically, we chose to implement Needham-Schroeder/Diffie Hellman and RSA for it's key distribution, and Paillier, AES, RSA, and Blum Goldwasser for it's encryption(Figure 1). We chose these methods to follow the directions specified in the project document, meaning that we needed to have a symmetric

key ciphers, PKC under ciphertext only adversary, semantically secure PKC, and homomorphic cipher, as well as including SHA1 and HMAC.

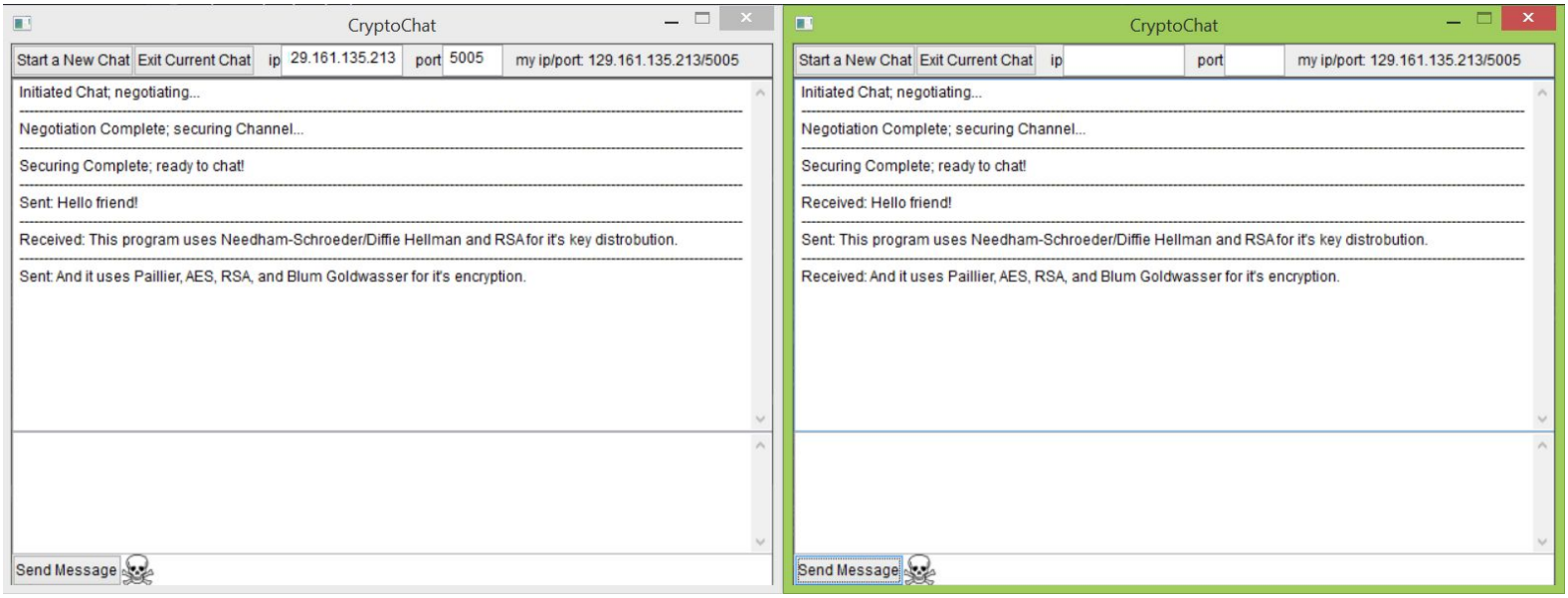

Figure 1.

Codebase Overview:

Our codebase is structured fairly logically; firstly, the various cyphers we implemented are each stored in their own named package, containing the source file and a blank file with the full name of the algorithm we were implementing.

We store the client's preference cryptography preferences in the preferences.cfg file. This file loads on startup, and indicated to the client which cipher is prefered, in descending order. This file can be edited by the user to switch the order of cipher preferences, allowing the user to choose which one they want to use.

Next up is the primality.py, which is our prime functions file containing our implementation of Miller Rabin, Pollard Rho, and a prime generator.

The hashing.py and sha1.py files are our implementation of HMAC and SHA1, which are used in message authentication.

The cryptoutilites.py file is a pretty general file used in our other cipher files, as it contains a bunch of useful functions such as Extended Euclid's Algorithm and smaller functions like xor and to/from bits, functions that converts a string to a list of ints and visa versa.

Finally there's networking.py, which is a file that contains our global and commonly used variables, such as ip, port, public and private keys, chat cryptography preferences, and the sockets used in the networking. This means that the file essentially acts as a focal point for storing data that may be required by other parts of the network.

RSA and AES:

The two primary cryptosystems that I worked on were RSA and AES. My RSA implementation was basically textbook RSA: It make 2 large primes, found n=p*q, found the lcm(p-1,q-1), made a random e < lcm and coprime to lcm, used the extended euclid and inverse multiplicative to find d, giving the public key (n,e) and the private key (n,d). Encryption and decryption were fairly easy, with encryption being (m**e)%n, and decryption being (c**d)%n.

AES was a little less mainstream, as I essentially ran out of time before the due date (This was easily my largest project at RPI, so I definitely learned some project time management

from this), so I implemented a simplified version of AES. This version of AES is basically the same as regular AES, except it uses much, much smaller s-boxes, as for reasons unknown to me, I couldn't get AES working for production using larger s-boxes (and therefore struggled with larger keys). My AES does have the regular AES functions however, as it has it's key expansion, add round key, sub bytes, shift rows, and mix columns. I suspect that the issue with my original implementation of AES had an issue with the (now removed) function lookupTables, in sub bytes. This now removed function interacted with the s-boxes, and I suspect that somewhere along the line the indice for traversing the sbox got shifted one way or another, messing up the lookup.

Figure 2, console output during RSA/AES

Final Thoughts:

Although this project was kind of stressful (especially at the beginning when I wasn't sure how to approach the task), once the different pieces started coming together, it got quite interesting, as trying to implement the various cryptosystems not just as esoteric projects, but as an actual cryptographic tool definitely made you think much more about not just the theoretical security, but the practical security elements of its implementation like considering the security of using the random library, or thinking how to expose as little data as possible.

This was especially important thinking, as we also had to consider potential ways our blackhat group could exploit what we wrote, meaning that anything we implemented, (cipher or not), had to have extra security scrutiny.

All and all, the project was definitely a learning experience, in more ways than one.