This program attempts to simulate the Needham-Schroeder Symmetric Key Protocol. Running hw2.exe will run a step by step simulation of a key exchange between alice and bob. The program will save data to two folders /Alice and /Bob to store the data available to each person. The program begins by acting as a server, randomly generating three 10 bit key to be used with the encryption/decryption algorithm we used for homework 1. It secretly gives one key to Alice, one key to Bob, and keeps the third private for now. It then encrypts some data to be sent to Alice who will decrypt it and send some to Bob. First, it encrypts the session key, the ID of Alice (which i represented with a string that's just "Alice"), and a timestamp. This is done using the key it shared with Bob. Now, it encrypts that encrypted data again, as well as the session key again, the ID of Bob, and a timestamp. This is done using the key it shared with Alice. It then sends that encrypted information to Alice.

Since Alice knows one of the keys, she can decrypt the first layer. Doing this gives her the session key, the ID of Bob, and a timestamp. She verifies that this is a new session key by comparing the timestamp to her own system clock. I allowed for a difference of at most 10 seconds. Alice sends the remaining encrypted data to Bob, who has the other key.

Since Bob knows the other key, he can decrypt the second layer. Doing this gives him the session key, the ID of Alice, and a timestamp. He also verifies that this is as new session key by comparing the timestamp to his own system clock.

Now that both parties have the session key, Bob (since he got his key last) initiates a test of the connection by sending an encrypted message to Alice on which

she will perform a simple operation (in this case, using an in-joke between the two), then send back through the same encrypted channel.

Until now, the server has just been directly distributing the initial two keys to Alice and Bob without any encryption. We can improve this by using a Diffie-Hellman key exchange protocol. The server chooses some prime number and a primitive root of that prime number, which it shares publicly with both Alice and Bob. For this example we choose the prime 1021 since we are using a 10 bit key. The primitive root of 1021 that we will use is 260. Both the server, Alice, and Bob take that root to the power of some random number between 1 and 1021. They take that number modulo 1021 and make it public. The server takes the public values from Bob and Alice and exponentiate it by the server's original random number modulo 1021. Alice and Bob take the server's number and exponentiate it by their own original random numbers modulo 1021. This should end with Alice and Bob sharing a secret number between 0 and 1021 with the server.

This works because $x^{ab} \bmod n = (x^a \bmod n)^b \bmod n = (x^b \bmod n)^a \bmod n$. This is also very difficult to compute without knowing either a or b, which are not made public. In order to compute one, one would have to solve a logarithmic equation. In the equation $y = x^a \bmod n$, you are required to compute $\log(y)_x \bmod n$.