Cavell Teng
HW2 Programming A

1. Needham-Schroeder Protocol:

The planned implementation of the Needham-Schroeder Protocol begins with *c1* connected to the *kdc* and sends it its identity, and the *c2*'s identity along with the nonce that it generates with the gen_nonce function. The kdc responds with an encrypted message containing the nonce that was sent, the session key, *c2*'s identity and an encrypted messaged for *c2*. This message is encrypted using the shared between the server and *c1*. The encrypted message for *c2* is encrypted with the shared key between *kdc* and *c2* and contains the session key along with *c1*'s identity. *c1* sends this message to *c2*. By this point *c1* and *c2* both have the session key. *c2* sends *c1* a message containing *c2*'s nonce encrypted by the session key. As a response, *c1* returns the same message except after an operation has been performed on *c2*'s decrypted nonce.

In order to resolve the issue of the replay attack. At the start of this protocol, *c1* messages *c2* with its identity. *c2* responds with c1's identity and a new nonce different from the one in the main protocol. This response is encrypted with the shared key between *c2* and the *kdc*. In the main protocol, this message that was sent by *c2* is sent to the *kdc* in *c1*'s request. The *kdc*'s reply also includes in the message encrypted with the shared key between *c2* and *kdc* the newly formed nonce from *c2*.

The assumptions in this protocol are that the *kdc* is secure. We also assume that the Diffie-Hellman exchange has already been preformed between *kdc* and *c1*, and *kdc* and *c2*. This means that all parties have their corresponding shared keys.

2. Diffie-Hellman Key Exchange:

The process of this implementation of the Diffie-Hellman Key Exchange begins with *c1*, *c2* and the *kdc* calculates (g^{s1} mod p), which will be call A, B, and C respectively. s1 is the secret integer that *c1*, *c2*, *kdc* randomly generate and g and p are publicly available prime numbers. The result of the operation, in the case of *kdc* is sent to *c1* and *c2*, and in the case of *c1* and *c2*, sent to the *kdc*. Once all parties have received that message. *c1* will perform A^{s1} mod p and the result of that operation is the secret key Kas (shared between the *kdc* and *c1*). *c2* will perform B^{s1} mod p and the result is the secret key Kbs (shared between the *kdc* and *c2*). In the program, the *kdc* expects c2 to run first followed by c1.

An assumption for this setup is that p and g are known. This allowed me to hard code the variables into all three files. Another assumption is that the *kdc* is a secure setup.