Needham-Schroeder Symmetric Key Protocol establishes a shared session key between two parties on a network with the purpose of protecting further communication between these two parties. In my implementation, we let Alice and Bob represent the two parties wishing to establish communication, and the Key Distribution Center (KDC) acts as the server trusted by both parties with generating and distributing keys. The variables required to initiate and carry out this protocol are as follows:

**IDA** : Alice’s identifier, 16-bit random number in my implementation

**IDB** : Bob’s identifier, similar construction

**N1** : random 32-bit nonce value generated by Alice. Conventionally, nonces are timestamps, but because this code will run so quickly, I use random values to ensure they are different each time.

**N2** : random 32-bit extra nonce generated by KDC to help secure against replay attacks (see Lecture SlideDeck 3.2, slide 15)

**Ka** : private key shared by Alice and the KDC, from Computational Diffie-Hellman (CDH)

**Kb** : private key shared by Bob and the KDC, from instance of CDH

**Ks** : random 10-bit session key generated by the KDC for end use between Alice and Bob

The protocol begins with Alice and Bob acting as clients, each separately connecting to the KDC to initiate the Computational Diffie-Hellman (CDH) key-exchange protocol. Diffie-Hellman requires the following:

a *prime* ***p*** = 1021 which forms the cyclic group : public value

a *generator* ***g*** = 10 of GF[1021] : public value

*random numbers* ***a, b*** in the range {0, … , 1020) : values private to client, KDC

CDH is first initiated by Alice, who initiates a connection with the KDC server. She calculates integer value *a* in the above convention, and then raises *ga mod p*, a value which she then sends to the KDC. Separately, the KDC generates *b* in the same convention, raising *gb mod p* and sending that value to Alice. Once Alice receives *gb mod p*, she then raises that value with her previously generated *a*, as follows: *(gb)a mod p*. Similarly, with received value *ga mod p*, the KDC calculates *(ga)b mod p*. Since *gab mod p = gba mod p*, Alice and the KDC now have a shared private key value *Ka* which is guaranteed to be the required 10 bits long because of our choice of *p*. Bob then initiates the same protocol with the KDC to generate shared private key value *Kb*. Once CDH has been successfully completed by both clients, the Needham-Schroeder Protocol can begin. At this point, Alice has private key *Ka* , Bob has private key *Kb* and the KDC has both private key values *Ka , Kb*.

Needham-Schroeder Protocol begins with Alice opening a client connection to the Key Distribution Center server. Alice generates *N1* according to the above convention. For the purposes of my implementation, *IDA IDB* and *N1* are all Binary Strings, which Alice then concatenates together in that order and sends to the KDC. This concludes Step 1.

For Step 2, the KDC receives this concatenated binary string from Alice. Parsing the string, it grabs the two 16-bit values *IDA* and *IDB* in addition to the 32-bit nonce *N1*. Now, the KDC generates random 10-bit session key value *KS* which will ultimately be the key Alice and Bob use for their future secure communications. The KDC adds 6 leading 0s to binary string *KS* in order to make the value 16 bits, or 2 complete bytes. In order to prevent against replay attacks, the KDC generates a second 32-bit nonce value *N2*. It then concatenates this new padded value *KS* with *IDA* and *N2* and then, with private key *Kb* uses the Simplified DES algorithm written for HW1 to encrypt that packet byte-by-byte, which will be referred to as the Kb Packet. The KDC then concatenates padded session key *Ks* with *IDB* and nonce *N1*, which it then concatenates with the encrypted Kb Packet. The resulting concatenated binary string is then encrypted byte-by-byte with private key *Ka*, and will be referred to as the Ka Packet. The KDC sends the Ka Packet to Alice, which concludes Step 2.

For Step 3, Alice remains a client, but Bob opens up a server connection in order to talk to Alice. Alice receives the Ka Packet from the KDC, and decrypts it byte-by-byte with their shared private key Ka. This partially-decrypted binary string reveals the padded session key *KS* to Alice, along with the still-encrypted Kb Packet to be sent to Bob. Alice removes the 6 leading 0s from the padded session key, revealing the 10-bit value *KS*. Because Alice knows the values *IDB* and *N1* she is able to identify when the Kb Packet begins. She then sends this to Bob, concluding Step 3.

Step 4 is the final step of Needham-Schroeder, as it is the step when Alice and Bob verify they have received the same session key *KS*. Bob receives the Kb Packet from Alice, and decrypts it byte-by-byte using his shared private key *Kb*. He now has the following three values: the padded session key *KS*, Alice’s identifier *IDA*and the unique nonce *N2*. Bob removes the 6 leading 0s, revealing the true 10-bit session key *KS*. Once Alice and Bob verify that their *KS* value is the same, the protocol has been successfully completed and the program terminates.

My program contains 4 main Java files: SDES.java, Alice.java, Bob.java and KDC.java. To successfully compile and run this protocol,

> javac ./SDES.java

must first be called. Then, compile KDC, Alice, and Bob and run them in that order from separate terminal windows, or separate computers.