Objective

The focus of this Project was on developing a secure channel between two entities, each of which reside on two different machines. The entities are engaged in some specific application. For the purpose of this security related example, I have used a chat server and client as the sample applications.

Overview of the Procedure

The two entities, A and B, authenticate each other using public-key system (RSA algorithm using 1024-bit keys). For this purpose, RSA public and private keys have to be generated for both the entities. Then while they authenticate each other, they also generate and exchange a 64-bit shared key generated using Diffie-Hellman key exchange protocol. Once that is done they use the shared key to encrypt data in both directions using any of the private key encryption schemes. The scheme I have implemented is DES. Also, it was a requirement that the protocol should handle Man-in-the-middle and Replay attacks. For this purpose, I have implemented the aggressive Oakley Key Exchange protocol which in addition to the above mentioned attacks also handles the denial of service attack arising from address spoofing. More specifically, each of the attacks is handled in the Oakley Protocol as follows:

1. Man-in-the-middle: Using RSA authentication
2. Replay attack: Using nonces
3. Denial of service attack, which arises from address spoofing: Using cookies

Details of the Procedure

Aggressive Oakley key exchange protocol

It consists of the following steps:

1. *Initiation by* A (A to B)

   The Initiator sends a message containing the following components:
   - ID of A, ID of B
   - Initiator cookie, CK-A : This is a unique cookie which is generated and associated with the expected IP address of the responder.
   - Encryption, hash, authentication algorithms : This is the initiator's choice of algorithms which will be used in setting up the secure channel.
   - Specific Diffie Hellman group (q, a)
   - public key $y_A = a^{X_A} \bmod q$: This is the initiator's public key which will be used by the responder later on to generate the shared secret key (using the Diffie-Hellman algorithm).
   - Nonce NA : This nonce is generated to avoid replay attacks
   - $\text{Signed}_{KR(A)}$[ID of A, ID of B, NA, q, a, y_A] : This part consists of data in the brackets which has been encrypted using the private RSA key of A. This signed part is for authenticating initiator's identity to the responder.

2. *Response by* B (B to A)

   The contacted party B authenticates the message sent by A using the signed part and replies with a message containing the following fields:
   - ID of B, ID of A
   - Responder cookie, CK-B, Returned initiator cookie, CK-A : B creates its own cookie to send to A and also sends back the cookie sent by A.
   - Encryption, hash, authentication algorithms
   - Specific Diffie Hellman group (q, a)
   - public key $y_B = a^{X_B} \bmod q$: This is the public key of B generated using Diffie-Hellman algorithm and sent to A so that A can generate the shared secret key

- Nonce NA, NB : NA is the nonce sent by A and NB is a nonce generated by B.
- Signed$_{KR(B)}$[ID of B, ID of A, NA, NB, q, a, y_B y_A] : This signed part is to authenticate B's identity to A.

3. *Response by A to B's message* (A to B)

   This is the final message in the aggressive oakley key exchange protocol. After receiving B's message, A authenticates it using the signed part and replies back with a message containing the following fields :
   - ID of A, ID of B
   - Returned cookie, CK-B, initiator cookie, CK-A
   - Encryption, hash, authentication algorithms
   - Specific Diffie Hellman group (q, a)
   - public key $y_A = a^{X_A} \bmod q$
   - Nonce NA, NB
   - Signed$_{KR(A)}$[ID of A, ID of B, NA, NB, q, a, y_B y_A]

The fields have the same meaning as in the above cases.
After this step, both the parties can generate a shared secret key as in the Diffie-Hellman algorithm. This shared secret key can then be used for encrypting all the following communication between A and B. Any algorithm can be used for encryption. In our case, I generated a 64-bit key which was used to encrypt the data using the DES algorithm.

<u>Diffie-Hellman Algorithm which is used in Oakley Key Exchange</u>
The original Diffie-Hellman consists of the following steps :

1. Station A or Station B selects a large, secure prime number p and a primitive root a (mod p). Both p and a can be made public.

2. Station A chooses a secret random x with 1 <= x <= p-2, and Station B selects a secret random y with 1 <= y <= p-2.

3. Station A send a^x (mod p) to Station B, and Station B sends a^y (mod p) to Station A.

4. Using the messages that they each have received, they can each calculate the session key K. Station A calculates K by K congruent to (a^y)^x (mod p), and Station B calculates K by K congruent to (a^x)^y (mod p).

<u>Implementation Details</u>
A sample chat server and chat client were encoded in Java which create a secure channel using Aggressive Oakley Key Exchange. The steps for making the application work are as follows:
1. Untar the file final_asg.tgz
2. cd Chat\ Program.
3. To compile the classes, please do ./compile.sh
4. To run the server on a machine, please do ./server.
5. Logon to another machine, change to the appropriate directory and do ./client IP_of_server

Also, if required, generate the RSA public and private keys for both the server and client using the command
1. ./generateRSAKeys