Crypto 1 HW 2 Implementation

This homework is an implementation of the Needham Schroeder Protocol to prevent replay attacks from the original Diffie-Hellman protocol to share symmetric keys with the use of a key distribution center (KDC). In this example, Alice is requesting a session key for symmetric DES encryption with BOB. The KDC will create and send a package with all of the properly encrypted information to have the symmetric key be shared between Alice and Bob without anyone else sharing it.

Diffie-Hellman Details

Diffie-Hellman works by creating a request to the KDC to start a session with another user. The KDC has (by one way or another) symmetric keys between itself and every user in the network. Any user can send a request by encrypting a request with that symmetric key. This protocol is relatively effective when it comes to distributing symmetric keys. However, the base concept can still be quite insecure. In particular, a replay attack can be used by an adversary to reveal the session key by resending the encrypted message sent to Bob, leading Bob to believe it is receiving another request for a session key from Alice. This vulnerability can lead to the adversary gaining access to the private session key.

A computational Diffie-Hellman approach to key distribution requires parties to already have some prior secret information (large prime q and primitive root α) with each other. For the purposes of this assignment, those values are hard-coded into the program. In a real-world example, of course, this would be a terrible way to ensure privacy. However, as this example is more about the implementation of the key distribution scheme than it is about creating a perfectly secure system, for now it is acceptable.

With the shared information α and q, both parties can create a secret and public key by exponentiating α with a secret key mod q to create public key. The security of Diffie-Hellman relies on the difficulty of computing the discrete log of a number a mod p, even when both α and p are known, and the exponent is the only value that is not known. However, with more modern computing power, implementations of this key distribution scheme must find larger and larger prime numbers and base roots to ensure that the discrete log problem is sufficiently difficult to crack using a brute force method.

Timestamps

Because of the vulnerability in a simple Diffie-Hellman key distribution, in the Needham-Schroeder Protocol, timestamps are used to both ensure the validity of the session and encryption, as well as checks to ensure that replay attacks are minimized. A timestamp can also be a safeguard against old messages. By including a check for expiration, even if the timestamp between two messages is the same, old messages will still be discarded or flagged as suspicious, and ensure the integrity of the cryptosystem more. That being said, including a timestamp may not be the only preventative measure necessary to ensure complete secrecy. Much more may need to be done in order to create a sufficiently difficult one-way trapdoor function and satisfactory authentication measures.

Implementation Details

For the purposes of this assignment, every part of the Needham-Schroeder Protocol is mapped out, from the KDC server, to Alice, the requester of the key, and Bob, the receiver of the session key. Once the session key is received by both parties, Alice and Bob will communicate with each other using their private session key. Messages will be encrypted using the DES protocol implemented in homework 1.

Usage

First, start up the KDC server:

```
python3 KDC.py
```

Then start both Bob and Alice's respective user programs:

```
python3 BOB.py
```

Finally Alice:

```
python3 ALICE.py
```

Every program must be run one after the other, in different terminal windows. By default, Alice and Bob will look for the KDC at ip address 127.0.0.1, but this can be changed by adding a command line argument:

```
python3 ALICE.py <ip.address>
```

Advantages

There are a few advantages to the Needham Schroeder protocol. most notably, the system is lightweight and easy to implement, as well as fast to encrypt and decrypt. If implemented correctly, can also be protected against replay attacks using extra authentication and time stamps.

Drawbacks

As effective time stamps are in verification and authentication, they are far from perfect. Consider for example people that wish to communicate with each other across timezones. A lot of care needs to be taken to ensure that timezones are converted properly. If not, people that are trying to genuinely communicate with each other might be flagged as adversaries that are attempting replay attacks, and possible adversaries might pass through and receive messages without problem. We can include more checks and authentication, however with more and more authentication, we begin to lose the simplicity or speed we gain with using the Needham Schroeder protocol.

Furthermore, this example only explores a single message being sent with one session key. in a real world example, we would need to be mindful that our private session key is not being used too often, so our security is not compromised.

Further Work

For this implementation to be more secure to replay attacks, standard time zones need to be agreed upon to check timestamps, and we may need to incorporate extra nonces and possibly MAC's. While increasing the complexity of the Needham-Schroeder protocol, these are in this case necessary measures to take to ensure the security of the KDC, and the session protocol. Additionally, every so often, the session key needs to be updated again, to ensure that we are not leaking information about the key as we use the symmetric key more and more.