**CSE 535: Final Report**

**Implementation of Needham–Schroeder public key protocol and Needham–Schroeder- Lowe protocol implementation using Python**

# Part 1: Problem statement

## What is Needdham-Schroeder protocol?

The Needham–Schroeder Public Key Protocol[4] is a communication protocol intended to be used over insecure network using public keys.

## What is Needdham-Schroeder-Lowe protocol?

## The Needham–Schroeder Public Key Protocol fails in a typical attack. In

## order to overcome the attack, small modification was done in message format, which is Needham–Schroeder-Lowe[3] Protocol.

## What is my goal?

## My first motivation is to understand the concepts of Needham Schroeder Public Key [4] Protocol and Needham Schroeder Lowe Protocol [3] using Public Key and understand how does the intrusion in the protocol happen and how it is resolved using the Needham Schroeder Lowe Protocol. I will implement it using DistAlgo/Python. [1]

## Desired input/output

### Desired Input

* The program gives 2 options of choosing which object in the protocol will do the communication first.
* Originally from the protocol, Alice is always assumed to communicate first and should be the one who communicate with the server and Bob is always the one who get the request of communication from Alice. But in my implementation, I give all objects the equal right to initiate the communication.

### Desired Output

* Communication established successfully if the input is correct
* All the messages between Alice and Bob should be encrypted before sending to each other
* All the decrypted messages should be successfully and correctly decrypted

## Performance

* Timestamp is added to each step in protocol for time evaluation and replay attack prevention purposes

## Applications

* This protocol works on the basis of 'tickets' to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner

## State of the art

Currently, there is no available or complete enough version of this protocol in Python to build on top or improve from it. The best existing implementation that I found are in some other languages.

The source code which was written in C can be found at [4].

Besides, I also found a project extending NS protocol using C-library at [5].

And Java version available at [6].

## Tasks

* Understand the protocol
* Learn Python language, especially in socket programming part
* Implement the protocol, both flawed and fixed version
* Improve the protocol by adding timestamp for evaluation and replay attack prevention purpose
* Correctness proof
* Optimization if possible
* Comparison with improved version proposed by another member in our group

# Part 2: Design

For detail of Design part, please refer to Design document[7] linked in reference part of this report.

For detail API document[8][9][10][11] generated by Pydoc, please refer to document linked in reference part of this report

## Original version

Here, Alice (A) or Bob (B) are clients who try to initiate the communication with the other one. S is a server trusted by both parties. In the communication:

* A and B are identities of Alice and Bob respectively
* KAS is a symmetric key known only to A and S
* KBS is a symmetric key known only to B and S
* NA and NB are nonces generated by A and B respectively
* KAB is a symmetric, generated key, which will be the session key of the session between A and B

![](data:None;base64,)

## Fixed version

The protocol is vulnerable to a replay attack (as identified by Denning and Sacco[12]). If an attacker uses an older, compromised value for KAB, he can then replay the message  to Bob, who will accept it, being unable to tell that the key is not fresh.

This flaw is fixed in the Kerberos protocol by the inclusion of a timestamp. It can also be fixed with the use of nonces as described below.

![](data:None;base64,)

# Part 3: Implementation

For complete implementation, please refer to Implementation file[13] linked in reference part of this report.

In the protocol, A and B plays a role as clients who want to communicate to each other and S as server that everyone trust. In each version, we have 4 main files:

For Needham Schroeder Symmetric Key Protocol - Original version

alice\_original : plays a role as Alice object in the protocol

bob\_original: plays a role as Bob object in the protocol

server\_original: plays a role as Server object in the protocol

ns: driver program serves as a small program of using the API:

At the beginning, program will ask user whether he wants to choose Alice or Bob as the first communicator by choosing A or B, or else, it will return a message alerting the possible options. This is a little difference than the original protocol since the original one assumes that Alice is always the one who wants to communicate first. I give Bob a right to stand up and get a chance if he wants to communicate first.

The default IP address for each role is 127.0.0.1 since I'm implementing the protocol on localhost, I will try to run it on different machine if time is allowed; the default ports are given

After choosing the one who communicates first, Alice and Bob object starts running each step in protocol and end up establishing the communication or failing if something wrong happen (wrong Nonce, long timestamp, etc)

If connection succeeds, Alice and Bob starts communicating, user can type down some sample message to test how they encrypt/decrypt and send/receive to/from each other.

The implementation for Needham Schroeder Symmetric Key Protocol - Fixed version is pretty similar to the previous one except the extra steps as we see in the protocol.

# Part 4. Testing and evaluation

For complete testing and evaluation, please refer to Testing output file[14] linked in reference part of this report.

## Instruction

* Running each module in different terminals
* Enter your choice in terminal running Server module to choose who you want to initiate the communication
* Program follows each step in protocol to establish the communication
* Type down some sample message in terminal running Alice/Bob module to check how message encrypted and decrypted

## Evaluation

* At the first glare, I want to make the scenario look like a real situation. It means that not only Alice, but anyone in the network also can request to communicate with other one. The original protocol simplifies to the situation that only Alice wants to communicate with Bob. But it turns out that for each time I want to test the program, I need to manually choose who will communicate with whom; which prevents me from testing program for a huge number of time. For solving this problem, I add timestamp which evaluates time for each run time, it also plays a role in preventing replay attack.
* The program runs well under local network, but I don’t have enough time to test the program on different machine where the network get other effect from error, traffic,…
* Compare with other language like C or Java, implementation in Python is much smaller and simpler.

# Part 5. References

[1]<https://docs.google.com/a/stonybrook.edu/document/d/1luZH1JavM65Mib5I1Fbevn6IuLVk-If_V6EQgmwwWsY/edit> (Accessed on November 8, 2013)

[2] [http://en.wikipedia.org/wiki/Needham%E2%80%93Schroeder\_protocol](http://en.wikipedia.org/wiki/Needham–Schroeder_protocol), Wikipedia, Needham-Schroeder Protocol (Accessed on November 8, 2013)

[3]<http://en.wikipedia.org/wiki/Kerberos_protocol>, Wikipedia, Kerberos Protocol, (Accessed on November 8, 2013)

[4] <https://github.com/krb5/krb5>(January 16, 1996) (Accessed on November 8, 2013)

[5] <https://github.com/abender/needham-schroeder> (Accessed on Dec 2, 2013)

[6] <https://github.com/ekoontz/jaas_and_kerberos>. (Accessed on Dec2, 2013)

[7] <https://sites.google.com/a/stonybrook.edu/sbcs535/students/tunguyen> (Accessed on Dec 2, 2013)

[8] <http://tunguyen.org/alice_original.html> (Accessed on Dec 3, 2013)

[9] <http://tunguyen.org/bob_original.html> (Accessed on Dec 3, 2013)

[10] <http://tunguyen.org/server_original.html> (Accessed on Dec 3, 2013)

[11] <http://tunguyen.org/ns_original.html> (Accessed on Dec 3, 2013)

[12] <http://en.wikipedia.org/wiki/Dorothy_E._Denning> (Accessed on November 8, 2013)

[13]<https://github.com/hikaruyh88/Fall2013_2/tree/master/AsynchronousSystem/ProjectV1.4> (Accessed on Dec 6, 2013)

[14]