**CSE 535: Final Report**

**Implementation Needham-Schroeder Public Key and Needham-Schroeder-Lowe Protocol using Python**

# Part 1: Problem statement

## What is Needham-Schroeder Public Key protocol?

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

## What is Needham-Schroeder-Lowe Key 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 was to understand the concepts of Needham Schroeder Public Key [4] Protocol and Needham Schroeder Lowe Protocol [3] using Public Key and understand how the intrusion in the protocol does happen and how it is resolved using the Needham Schroeder Lowe Protocol. I have implemented it using Python. [1]

## Entities:

## Clients Alice (A) and Bob (B): These are the clients which have to communicate.

## Server (S): This is the server S which is trusted by both clients A and B.

## Input and Output Format:

## **Input:**

## 1. The nodes/parties Alice (A) and Bob (B) which uses a trusted server (S) to distribute public keys on request. The nodes will be processes/threads in our term.

## 2. Public Key(K-PA) and Secure Key(K-SA) for A.

## 3. Public Key(K-PB) and Secure Key(K-SB) for B.

## 4. Public Key(K-PS) and Secure Key(K-SS) for S.

## **Output:**

## Series of messages which are passed between the nodes to establish authentication among client nodes.

## Previous Implementations:

There are some implementations available for Needham–Schroeder Public Key Protocol in languages like Java and C. But there is no complete implementation available in Python for the same. I have taken reference from this partial implementation of client side of the protocol on github for [Pqauth](https://github.com/teddziuba/pqauth/tree/master/python/pqauth).[6] Using this I found the libraries to be used for encryption and basic client server architecture.

## Libraries used:

1. **json** : Json module to communicate the data in the format of key-value pair

2. **socket** : Socket module for client server communication

3. **threading** : threading interface for creating the threads for clients(alice and bob) and server.

4. **Crypto** : library used for applying cryptographic algorithm like RSA and public key cryptography

5. **colorama** : library used for putting colors in print statement for better visualization of data and messages.

## Important Design decisions incurred during implementation:

1. Choice between DistAlgo and Client Server architecture implementation: I first started the implementation with Distalgo and Client-Server architecture both. But I found the client server architecture more close to the real implementation.

The integration was easier with encryption libraries like ‘Crypto’ used for RSA and public key cryptography. Hence there was lots of flexibility regarding the use of algorithms and the API’s used for the same.

Also it was easier to use some of the previous work on the same topic.

Only downside was to re implement the message passing architecture for client and server.

1. Usage of socket programming and threads for synchronization: To synchronize the message passing between the client and server on the socket, we have to merge the concepts of socket and threads. I faced certain hiccups, but finally I could achieve it.

**Class Diagram**:

Nodes/Clients

Server

Member variables

Server ID: int (16 bit)

Port Number: int (16 bit)

Private Key: int (64 bit)

Public Key: int (64 bit)

Array of public key of all clients: Int\_64 [2]

Member Variables

Client ID: int (16 bit)

Port Number: int (16 bit)

Private Key: int(64 bit)

Public Key: int (64 bit)

Member Functions

void Send\_message (Msg M, Node N1)

void Receive\_Message (Msg M)

Msg Encrypt\_Message (Msg M)

Msg Decrypt\_Message(Msg M)

Member Functions

void Send\_message (Msg M, Node N1)

void Receive\_Message (Msg M)

Msg Encrypt\_Message (Msg M)

Msg Decrypt\_Message(Msg M)

Int\_64 Send\_Public\_Key(Node N1)

Sends Receives

Sends Receives

Message (Msg)

Member Variables

Number of Strings

Array of Strings

Is-encrypted

Member functions

bool Check\_is\_encrypted(void)

## Implementation Methodology:

1. Learning the protocol steps
2. Learning Python language and encryption algorithms and socket programming part.
3. Implement the protocol : Needham Schroeder Public Key
4. Implemented the protocol : Needham Schroeder Lowe Public Key

Design Steps:

We specify the steps involved in the implementation of the protocol and the supporting libraries of Python used for the implementation.

**Basic Implementation of protocols in Python**

1. Creation of clients and server: We will create a client-server architecture to start the communication.

2. Encryption and Decryption of messages using public/private keys: Both client and server have to encrypt/decrypt messages using public/private keys. For the same, we will use the Pycrypto toolkit module of python.

Functional Description of Classes:

1. **Nodes/Clients**: These are the sockets which are assigned different port numbers. The nodes communicate through these ports.

**Member variables**:

* 1. **Port\_Number (Int\_16):** This is the socket though which the process will communicate.
  2. **Private\_Key (Int\_64) :** This is private key of the node
  3. **Public\_Key (Int\_64) :** This is public key of the node
  4. **Client\_ID (Int\_16) :** This is serial id of the client.

**Member functions:**

**bool init (int\_64 pk, int\_64 sk)**

**Description:** Initialize the client with the public and private keys.

**Input :**  Int\_64 pk : public key of the server

Int\_64 sk : public key of the server

**Output:** check whether the client was properly started or not

**bool send\_message (Msg M, Node N1)**

**Description:** This function is used to send message M to Node N1

**Input :**  Message M : message that need to be sent

Node N1 : Node to which the message is need to sent

**Output:** check whether the message was sent or not

**void receive\_message (Msg M)**

**Description:** This function is used to receive message M.

**Input :**  Message M : message which is received.

**Output:** Null

**Msg encrypt\_message (Msg M)**

**Description:** This function is used to encrypt the message M.The

function first performs the check whether the message is encrypted.

**Input :**  Message M : message which need to be encrypted.

**Output:** Encrypted message

**Msg decrypt\_message (Msg M)**

**Description:** This function is used to decrypt the message M.The

function first performs the check whether the message is encrypted

**Input :**  Message M : message which need to be encrypted.

**Output:** Encrypted message

1. **Server**: These are also the sockets with the name ‘localhost’ and specific port number.

**Member variables**:

**Port\_Number (Int\_16):** This is the socket though which the process will communicate.

**Private\_Key (Int\_64) :** This is private key of the node

**Public\_Key (Int\_64) :** This is public key of the node

**Socket\_ID (Int\_16) :** This is serial id of the client.

**Int\_64 Key\_Arr [2] :** This is the array of the public key of the protocol.

**Member functions:**

**bool init (int\_64 pk, int\_64 sk)**

**Description:** Initialize the server with the public and private keys.

**Input :**  Int\_64 pk : public key of the server

Int\_64 sk : public key of the server

**Output:** check whether the server was properly started or not

**bool send\_message (Msg M, Node N1)**

**Description:** This function is used to send message M to Node N1

**Input :**  Message M : message that need to be sent

Node N1 : Node to which the message is need to sent

**Output:** check whether the message was sent or not

**void receive\_message (Msg M)**

**Description:** This function is used to receive message M.

**Input :**  Message M : message which is received.

**Output:** Null

**Msg encrypt\_message (Msg M)**

**Description:** This function is used to encrypt the message M.The

function first performs the check whether the message is encrypted

**Input :**  Message M : message which need to be encrypted.

**Output:** Encrypted message

**Msg decrypt\_message (Msg M)**

**Description:** This function is used to edecrypt the message M.The

function first performs the check whether the message is encrypted.

**Input :**  Message M : message which need to be encrypted.

**Output:** Encrypted message

**Int\_64 send\_public\_key (Node N1)**

**Description:** This function sends the public key of the node N1 when

asked by other nodes in the network.

**Input :**  Node N1 : the node for which public key need to be sent

**Output:** Public Key

**Utility functions:**

**Int\_16 get\_node\_id (self)**

**Description:** It provides the node id of the client/server

**Output:** Node id of itself

**Int\_16 get\_port\_no (self)**

**Description:** It provides the port number of the client/server

**Output:** Port number of itself

# Part 5. References

[1.] http://en.wikipedia.org/wiki/Needham%E2%80%93Schroeder\_protocol (Accessed on November 8, 2013)

[2.] Cremers, C., and Sjouke Mauw. "A Family of Multi-Party Authentication Protocols." First Benelux Workshop on Information and System Security (WISSec). 2006.

[3.]. Lowe, Gavin. "Breaking and fixing the Needham-Schroeder public-key protocol using FDR." Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 1996. 147-166.

[4.] Needham, Roger; Schroeder, Michael , "Using encryption for authentication in large networks of computers.". Communications of the ACM 21, December 1978

[5.] <https://www.dlitz.net/software/pycrypto/> (Accessed on November 8, 2013)

[6.] <https://github.com/teddziuba/pqauth/tree/master/python/pqauth> (Accessed on December 8.2013)

# Part 5. Appendix:

Output Snippet:

*Starting the Server*

*Step 1:*

*Connecting from alice to server*

*alice -> Server: Requesting bob's public key*

*Step 2:*

*Server -> Alice : Sending Bob's public key and identity*

*bob's public key fingerprint:04:c1:86:7a:5c:12:39:4f:72:b9:a6:dd:ac:b3:02:21*

*bob's public key :-----BEGIN PUBLIC KEY-----*

*MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAjwB6bjxhY7tMA1iRMIyR*

*jebk+h0THeJNhagmRlSCGHw9O9MbPCUcNZ1ffUTCP3UR0KsuxaEhDUZL2lC8//uR*

*uHyTMyIph3ZsGuNx4sDktOG0GARf/guCTHLQCQZezsofyOlEgns2PeSMBgAXPWZN*

*f9EzUtnEqkojukTVW5Ok7M2uXAWTIPeePjzfMZJ2wt0z8LIwyPi4tLuHZhnr0fC/*

*UsY4FgPdvglL9gw8gDU8uf7SgI+4g+0wjj/Dc2fH2B67ddPsKoFxLjxaDnAYVX8L*

*/ZE+Fv34+tCghFYMG3Fm9I9Ntoc79/PWm2fsfRbIzCWemC58wpG3D2GE9yEu6RYb*

*mwIDAQAB*

*-----END PUBLIC KEY-----*

*Step 3:*

*Connecting from bob to server*

*bob -> Server: Requesting alice's public key*

*Step 4:*

*Server -> Bob : Sending Alice's public key and identity*

*alice's public key fingerprint:5d:4b:1a:28:af:94:44:9a:a8:b2:1e:82:a1:d7:32:d1*

*alice's public key :-----BEGIN PUBLIC KEY-----*

*MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAlF1+/57G/bWMFu2GTyq0*

*nGZGBtb+1KP2Yiq3f7zs0zA++6qidxqOZMGmGAeXFY/mHzhdMvReGI74HPumByA/*

*grb+oHc8N4kyNDdBgrN+xLfzMsCh32FS1hm955H1wSu3QwYVhXaik/q3CIqGA6ek*

*OXxojf3VRtx9Sjz1WSp5YgRjQTtXl+imssr4bZdBgd8bmw6HggJOjuiX7WPf0fsH*

*ESRlP1cTjvRZlY21WsbDL7gSBAOYxjnZf9AejXr3OFcIz6RodDfqCSsnVM9ZVKgF*

*7+lHc9AP4A6SrRKCyHHB5F9D/6WZuXUkX1RiMUB2pT0Lad3go1EXcte0EOQJjH6S*

*4QIDAQAB*

*-----END PUBLIC KEY-----*

*Start Bob as server and Alice as client*

*Step 5:*

*alice sending the nuance to bob*

*alice ->bob: {"A": "5d:4b:1a:28:af:94:44:9a:a8:b2:1e:82:a1:d7:32:d1", "NA": "c501eb82-4fcb-57c8-0438-b38f49617c98"}*

*Step 6:*

*bob sending the nuance to alice*

*bob ->alice: {"NA": "c501eb82-4fcb-57c8-0438-b38f49617c98", "NB": "83b20905-9d64-cec9-edcd-b65421c59d21"}*

*alice got the message{"NA": "c501eb82-4fcb-57c8-0438-b38f49617c98", "NB": "83b20905-9d64-cec9-edcd-b65421c59d21"}*

*Step 7:*

*alice sending the nuance back to bob*

*alice ->bob: {"NB": "83b20905-9d64-cec9-edcd-b65421c59d21"}*

*Done with communication*

: