实 验 报 告

课程名称 密码学基础

学　　院 计算机学院

专　　业 保密管理

姓　　名 王君可

学　　号 **17300240009**

开 课 时 间 **2019** 至 **2020** 学年第 二 学期

实验项目名　　称	Needham-Schroeder Protocol	成绩	

一、实验目的

1. Understanding Needham-Schroeder (Public Key) Protocol 理解 Needham-Schroeder (Public Key) Protocol 协议

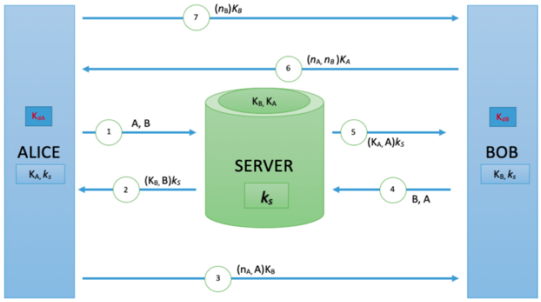
2. Understanding man-in-the-middle (MITM) attack against Needham- Schroeder (Public Key) Protocol 理解中间人攻击

二、实验内容

1. Needham-Schroeder (Public Key) Protocol 协议内容

Client 和 server 共同心信任一个 PKI，他们在通信的时候通过向 PKI 发送请求获取对方的公钥，也即是说 PKI 拥有 client 的公钥K_{PA} 和 server 的公钥K_{PB}。同时，client 和 server 分别拥有自己的私钥K_{RA}和K_{RB}，私钥是只有他们自己知道的。如果 client 想要和 server 通信来获取 server 的服务，他们的通信过程应当是这样的：

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{K_{PB}, B\}_{K_{SS}}$
3. $A \rightarrow B : \{N_A, A\}_{K_{PB}}$
4. $B \rightarrow S : B, A$
5. $S \rightarrow B : \{K_{PA}, A\}_{K_{SS}}$
6. $B \rightarrow A : \{N_A, N_B\}_{K_{PA}}$
7. $A \rightarrow B : \{N_B\}_{K_{PB}}$



从上述过程可知，二者用一个随机数N_A和N_B来与对方进行质询应答验证对方的身份。在最后一步里，client 还要向 server 发送本次通信的会话密钥$ssn\ key$，这个会话密钥是对称密钥，只有二人知晓。

2. 中间人攻击内容

从上述双方的通信过程我们可以知道，Needham-Schroeder (Public Key) Protocol 协议对于中间人攻击是脆弱的，具体的过程是这样的：

$A \rightarrow I : \{N_A, A\}_{K_{PI}}$

$I \rightarrow B : \{N_A, A\}_{K_{PB}}$

$B \rightarrow I : \{N_A, N_B\}_{K_{PA}}$

$I \rightarrow A : \{N_A, N_B\}_{K_{PA}}$

$A \rightarrow I : \{N_B\}_{K_{PI}}$

$I \rightarrow B : \{N_B\}_{K_{PB}}$

也就是说，adversary 首先使得 client 与其通信后，通过截取 client 的信息、操作后发给 server，来达到 server 误以为 client 在与其通信而不是 adversary 的效果。

三、实验步骤

1. 实现 PKI：

   extract ()

2. 实现 NS 公钥协议：

   $ns\_authentication(sock, server\_name)$

   $ns\_authentication(conn)$

3. 打开终端，在命令行执行

```
$ python server.py
```

然后重新打开一个终端，执行

```
$ python client.py -s server -u my_file.txt
```

4. 实现对 NS 公钥协议的中间人攻击

   attack(conn)

5. 打开两个终端，分别执行

```
$ python server.py
$ python adversary.py
```

重新打开终端，执行

```
$ python client.py -s adversary my_file.txt
```

四、实验结果及分析

具体的实现详见提交的 ppt 里

特别说明，我在实现的时候，基于 PKI 一直 online 的思想，所以说，无论是执行 NS 协议还是中间人攻击，都需要先打开命令行，执行$python\ pki.py$的命令，来使得 PKI 上线，接受通讯请求。

1. NS 协议执行的结果

   Server 端：

```
(base) macbook@macbookdeMacBook-Pro server % python server.py
Server: storage server
Server: beginning to serve clients...
Server: connection from client with address ('127.0.0.1', 56593)
Server: connection verified!
(b'HIj29T1ds1xrF7pM', 'client')
Server: using session key b'HIj29T1ds1xrF7pM' from client client
Server: recieved request of file my_file.txt for mode u
Server: beginning transfer for my_file.txt...
Server: completed transfer for my_file.txt
Server: file saved in client/my_file.txt
Server: transfer complete, shutting down...
```

   Server 端首先接受来自 client 的通信请求，具体来说应该是$\{N_A, A\}K_{PB}$，然后其分别与 PKI 和 client 通信，完成 NS 协议的过程，实现认证，最后接受来自 client 的服务请求。

   Client 端：

```
(base) macbook@macbookdeMacBook-Pro client % python client.py -s server -u my_fi
le.txt

Client: connection verified!
Client: using session key b'HIj29T1ds1xrF7pM'
Client: sent file name my_file.txt for mode u
Client: my_file.txt is read and ready for upload
Client: beginning file upload...
Client: uploading file... (1/3)
Client: uploading file... (2/3)
Client: uploading file... (3/3)
Client: successful upload for my_file.txt
Client: client shutting down...
```

Client 首先向 server 发送通信请求，具体来说应该是$\{N_A, A\}K_{PB}$，然后与 server 执行 NS 协议完成身份认证（包含对 server 的身份认证），最后向 server 提交服务请求。

同时在 server 文件夹下，会出现 client 文件夹以及 my_file.txt 的文档，内容是“Hello there. I'd like to say SJTU NB!”

2. 中间人攻击的执行结果

Server 端：

```
(base) macbook@macbookdeMacBook-Pro server % python server.py
Server: storage server
Server: beginning to serve clients...
Server: connection from client with address ('127.0.0.1', 56771)
Server: connection verified!
(b'joXqGTWtkhGEn3Uq', 'client')
Server: using session key b'joXqGTWtkhGEn3Uq' from client client
Server: recieved request of file bad_file.txt for mode u
Server: beginning transfer for bad_file.txt...
Server: completed transfer for bad_file.txt
Server: file saved in client/bad_file.txt
Server: transfer complete, shutting down...
(base) macbook@macbookdeMacBook-Pro server %
```

Server 端收到来自 adversary 转发的 A 的通信（adversary 解密后用 server 的公钥重新加密）$\{N_A, A\}K_{PB}$，然后向 M 应答$\{N_A, N_B\}K_{PA}$，最后收到 M 发来的$\{K, N_B\}K_{PB}$。

Client 端：

```
(base) macbook@macbookdeMacBook-Pro client % python client.py -s adversary -d my
_file.txt

Client: connection verified!
Client: using session key b'joXqGTWtkhGEn3Uq'
Client: sent file name my_file.txt for mode d
Client: beginning download for my_file.txt...
Client: completed download for my_file.txt
Client: file saved in my_file.txt
Client: client shutting down...
(base) macbook@macbookdeMacBook-Pro client %
```

Client 端首先向 adversary 发送了通信的请求，所以 adversary 让 client 误以为其是服务端，然后收到 adversary 的$\{N_A, N_B\}K_{PA}$(这条消息原封不动的来自 server)，最后向 adversary 发送了$\{K, N_B\}K_{PM}$。

Adversary 端：

```
(base) macbook@macbookdeMacBook-Pro adversary % python adversary.py
Adversary: malicious storage server
Adversary: beginning to 'serve' clients...
Adversary: connection from client with address ('127.0.0.1', 56768)
Adversary: I got in!
Adversary: bad_file.txt is read and ready for upload
Adversary: uploaded file name bad_file.txt
Adversary: beginning file upload...
Adversary: uploading file... (1/2)
Adversary: uploading file... (2/2)
Adversary: successful upload for bad_file.txt
Adversary: recieved request of file my_file.txt for mode d
Adversary: beginning transfer for my_file.txt...
Adversary: transferring file... (1/3)
Adversary: transferring file... (2/3)
Adversary: transferring file... (3/3)
Adversary: successful upload for my_file.txt
Adversary: shutting down server...
(base) macbook@macbookdeMacBook-Pro adversary %
```

Adversary 首先让 client 向其发送通信请求$\{N_A, A\}K_{PM}$，然后私钥解密后用 server 的公钥重新加密并发送给 server，然后收到 server 的$\{N_A, N_B\}K_{PA}$并原封不动发给 client，接着收到 client

的$\{K,N_B\}K_{PM}$,最后向 server 发送$\{K,N_B\}K_{PB}$，实现了 server 误以为 client 与其通信。
同时在 server 文件夹下，上一步生成 client 的文件夹下面会多出 bad_file.txt 的文档，内容是“Fudan NB! Stupid!”，还有就是 adversary 文件夹下面会生成 client 的文件夹，并出现了 my_file.txt 的文档。Adversary 假装成 client 的身份向 server 发送了 bad_file.txt,然后为了不让 client 怀疑，完成了其服务的需求上传 my_file.txt.

五、实验总结

1. NS 协议中两次使用了质询与应答的思想，完成了 client 和 server 之间相互的身份认证。
2. NS 协议中最终的会话密钥是有 client 端选择的，而且往往选择对称密钥。
3. NS 协议最终实现的效果就是 client 和 server 之间的相互身份认证以及 client 与 server 建立会话密钥。
4. NS 协议对于中间人攻击是脆弱的，中加人只需要让 client 对自己发送通信请求，并可以在 client 和 server 之间通信，就可以以较低成本实施攻击，让 server 误以为 client 在与其通信，其实本质原因是没有保证通信消息的鲜活性。
5. 为了抵抗中间人攻击，一种方案是可以用时间戳代替随机数N_A和N_B进行隐式的质询与应答，也就是 NS 的协议内容是这样的：
   A->B: $\{T1,A\}K_{PB}$
   B->A: $\{T1,T2\}K_{PA}$
   A->B: $\{ssn\ key,T2\}K_{PB}$
   B->A: VERIFIED
   这样，即使 M 截取到了$\{T1,T2\}K_{PA}$再转发给 A，A 会检查时间戳与本地时钟发现已经失效。