实 验 报 告

课程名称	密码学基础
学　　院	计算机科学技术学院
专　　业	信息安全
姓　　名	冉津豪
学　　号	17307130179

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
2. Understanding man-in-the-middle(MITM) attack against NeedhamSchroeder (Public Key) Protocol

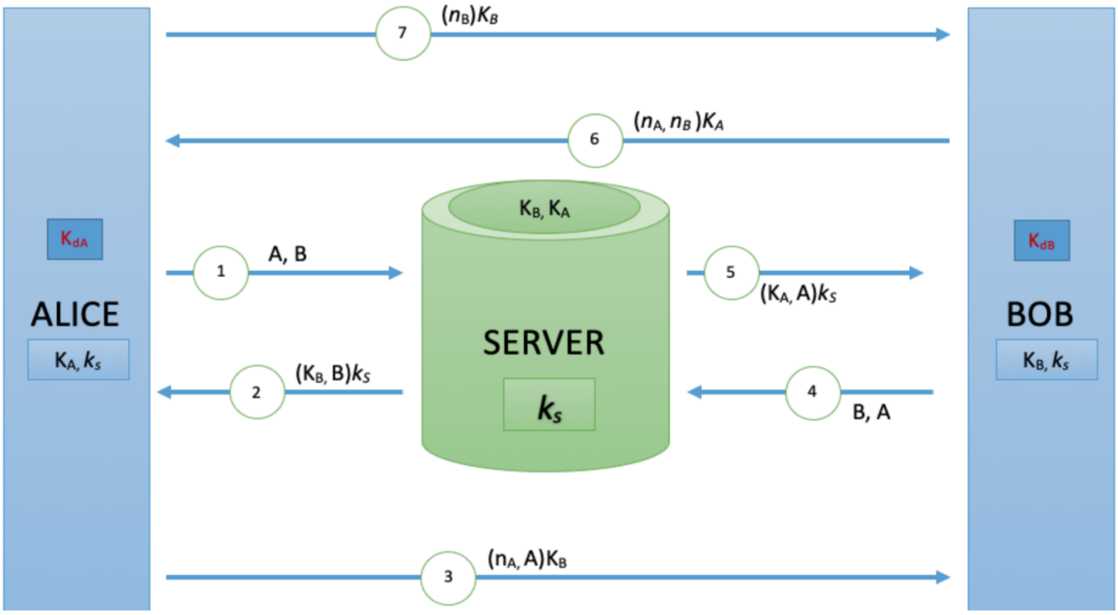
二、实验内容

1. The public-key protocol

Kpx 和 Ksx 分别为 x 的一对非对称密钥。

当 Alice 和 Bob 想要通过服务器分发公钥，服务器拥有 Alice 和 Bob 的公钥。S 的公钥是公开的。详细过程如下。完成该过程后，A，B 都获得了对方的公钥，并完成随机数验证。

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



2. Attacking the Needham-Scroeder (Public Key) Protocol

这个协议容易受到中间人攻击。只需要中间人 I 与 A 建立通信，将 A 的随机数用 B 的公钥加密发送给 B，使得 B 认为 A 正与其进行通信即可，但实际上，A 和 B 的随机数都已被 I 知晓。

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

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

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

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

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

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

三、实验步骤

1. 实现 PKI

PKI 是对公钥获取请求进行相应，对于 A 请求 B 的公钥就用 A 的公钥加密 B 的公钥返回给 A 即可。

代码主要与 helpers.ns 中的 get_public_key 相对应，使得其它主机可以通过 get_public_key 获得相应的公钥。

```
def extract():
    """() -> NoneType
    Opens the public key infrastructure server to extract RSA public keys.
    The public keys must have already been in the server's folder.
    """
    with socket(AF_INET, SOCK_STREAM) as sock:
        sock.bind((PKI_HOST, PKI_PORT))
        sock.listen()
        while True:
            conn, addr = sock.accept()
            with conn:
                print('PKI: connection from address', addr)
                # A, B --->
                M = conn.recv(1024)
                A, B = M.decode("UTF-8").split(',')
                file_name_A = A + ".asc"
                file_name_B = B + ".asc"
                with open(file_name_A, "r") as fileStream_A:
                    buffer_A = fileStream_A.read()
                with open(file_name_B, "r") as fileStream_B:
                    buffer_B = fileStream_B.read()
                A_pk = rsa.import_key(str.encode(buffer_A))

                # <--- {K_PB, B}(K_PA)
                data = buffer_B + "," + B
                cipher = rsa.big_encrypt(A_pk, data)
                response = b''
                for chunk in cipher:
                    response += chunk + b','
                conn.send(response[:-1])
```

2. 实现 NS 公钥协议

完成 client 和 server 的 Needham-Scroeder Protocal 的交互，包括向 PKI 获取公钥、随机数交换、确定会话密钥等步骤。

Client:

```python
def ns_authentication(sock, server_name):
    """(socket, str) -> bytes or NoneType
    Performs authentication via Needham-Schroeder public-key protocol.
    Returns a symmetric session key if authentication is successful,
    a None otherwise.

    :sock: connection to storage server
    :server_name: name of storage server
    """
    # WRITE YOUR CODE HERE!
    address = (PKI_HOST, PKI_PORT)

    # get RSA key of Client
    with open("RsaKey.asc", "r") as fileStream_A:
        buffer_A = fileStream_A.read()
    A_sk = rsa.import_key(str.encode(buffer_A))

    # get public key of file transfer server
    buffer_B = ns.get_public_key(address, server_name, NAME, A_sk)
    B_pk = rsa.import_key(buffer_B)

    # A -- {N_A, A}(K_PB) --> B
    N_A = ns.generate_nonce()
    send_data = str(N_A) + ',' + NAME
    cipher = rsa.big_encrypt(B_pk, send_data)
    send_byte = b''
    for chunk in cipher:
        send_byte += chunk + b','
    sock.send(send_byte[:-1])

    # A <-- {N_A, N_B}(K_PA) -- B
    recv_data = sock.recv(1024)
    plaintext = rsa.big_decrypt(A_sk, recv_data.split(b','))
    N_AB = plaintext.split(",")

```

```python
        # check if Server actually did recieve Client's nonce
        if str(N_A) != N_AB[0]:
            return print("Nonce wrong from {}, exiting...
    ".format(server_name))

        # A -- {K, N_B}(K_PB) --> B
        ssn_key = aes.generate_key()
        send_data = ssn_key.decode("UTF-8") + ',' + N_AB[1]
        cipher = rsa.big_encrypt(B_pk, send_data)
        send_byte = b''
        for chunk in cipher:
            send_byte += chunk + b','
        sock.send(send_byte[:-1])

        # get confirmation
        if int(sock.recv(1024)) == RESP_VERIFIED:
            print("Client: connection verified!")
            return ssn_key
        else:
            print("Client:connection failed!")
```

Server:

```
def ns_authentication(conn):
    """(socket, str) -> bytes or NoneType
    Performs authentication via Needham-Schroeder public-key protocol.
    Returns a symmetric session key and client's name if authentication
    is successful, a None otherwise.

    :sock: connection to storage server
    :NAME: name of storage server
    """
    # WRITE YOUR CODE HERE!
    # get RSA key of Server for decrypting
    address = (PKI_HOST, PKI_PORT)
    with open("RsaKey.asc", "r") as fileStream_B:
        server_key_byte = fileStream_B.read()
    server_sk = rsa.import_key(str.encode(server_key_byte))

    # A -- {N_A, A}(K_PB) --> B
    recv_data = conn.recv(1024)
    plaintext = rsa.big_decrypt(server_sk, recv_data.split(b','))
    N_A, client_name = plaintext.split(",")
    N_B = ns.generate_nonce()
    send_data = str(N_A) + ',' + str(N_B)

    # get client's public key
    client_pk_byte = ns.get_public_key(address, client_name, NAME, server_sk)
    client_pk = rsa.import_key(client_pk_byte)

    # A <-- {N_A, N_B} -- B
    cipher = rsa.big_encrypt(client_pk, send_data)
    send_byte = b''
    for chunk in cipher:
        send_byte += chunk + b','
    conn.send(send_byte[:-1])

    # A -- {K, N_B} --> B
    recv_data = conn.recv(1024)
    plaintext = rsa.big_decrypt(server_sk, recv_data.split(b','))
    ssn_key, N_B_recv = plaintext.split(",")
```

```python

        # check if client did actually recieve Server's nonce
        if N_B_recv != str(N_B):
            conn.send(str(RESP_DENIED).encode("UTF-8"))
            return print("Nonce wrong from", client_name)
        conn.send(str(RESP_VERIFIED).encode("UTF-8"))
        print("Server: connection verified!")
        return bytes(ssn_key, "utf-8"), client_name
```

完成后：会话密钥确定，文件传输正确。

```
Server: storage server
Server: beginning to serve clients...
Server: connection from client with address ('127.0.0.1', 12476)
Server: connection verified!
Server: using session key b'edlv5WOPhYXCqBmU' 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...
```

```
Client: connection verified!
Client: using session key b'edlv5WOPhYXCqBmU'
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...
```

3. 实现对 NS 公钥协议的中间人攻击

在 adversary 中实现中间人攻击，对于 client，其充当服务端，对于 server，其充当客户端。运行 PKI、server、adversary 之后，运行

python client.py -s adversary my_file.txt

此时，无论来自 client 的命令是什么，adversary 都会向 server 上传 bad_file.txt。同时根据 client 的命令对 client 做出响应。

```python
def attack(conn):
    """(socket) -> (bytes, str) or NoneType
    Performs a man-in-the-middle attack between the client and Bob's storage server.
    Returns the session key and clients name if attack was successful, otherwise
    returns None.

    :conn: connection to the client (victim)
    """
    # get RSA key of Adversary for decrypting
    with open("RsaKey.asc", "r") as fileStream:
        buffer = fileStream.read()
    M_sk = rsa.import_key(str.encode(buffer))

    # A -- {N_A, A}(KP_M) --> M
    recv_data = conn.recv(1024)
    plaintext = rsa.big_decrypt(M_sk, recv_data.split(b','))
    client_name = plaintext[plaintext.rfind(','):]
    # get public key of Server for encrypting
    PKI_address = (PKI_HOST, PKI_PORT)
    buffer_B = ns.get_public_key(PKI_address, "server", NAME, M_sk)
    B_pk = rsa.import_key(buffer_B)

    # reencrypt request for Server
    cipher = rsa.big_encrypt(B_pk, plaintext)
    send_byte = b''
    for chunk in cipher:
        send_byte += chunk + b','

    # open connection with Server
    server_address = (SERVER_HOST, SERVER_PORT)
    with socket(AF_INET, SOCK_STREAM) as sock:
        sock.connect(server_address)
        # M -- {N_A, A}(KP_B) --> B
        sock.send(send_byte[:-1])
```

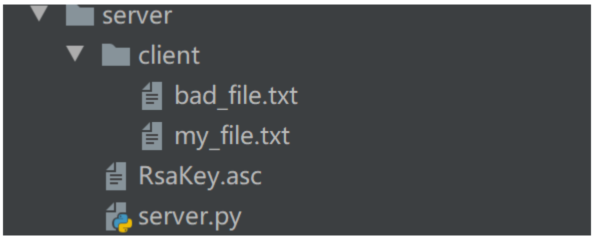
```python
            # M <-- {N_A, N_B}(KP_A) -- B
            recv_data = sock.recv(1024)
            # A <-- {N_A, N_B}(KP_A) -- M
            conn.send(recv_data)
            # A -- {K, N_B}(KP_M) --> M
            recv_data = conn.recv(1024)
            plaintext = rsa.big_decrypt(M_sk, recv_data.split(b','))
            ssn_key = plaintext[:plaintext.rfind(',')].encode("UTF-8")
            # M -- {K, N_B}(KP_B) --> B
            cipher = rsa.big_encrypt(B_pk, plaintext)
            send_byte = b''
            for chunk in cipher:
                send_byte += chunk + b','
            sock.send(send_byte[:-1])
            # check if MITM attack was successful
            if int(sock.recv(1024)) == RESP_VERIFIED:
                print("Adversary: I got in!")
                upload_bad_file(sock, ssn_key)
                return ssn_key, client_name
            else:
                print("Adversary: wtf...")
            print("Adversary: attack completed")
```

```
Adversary: malicious storage server
Adversary: beginning to 'serve' clients...
Adversary: connection from client with address ('127.0.0.1', 1599)
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 u
Adversary: beginning transfer for my_file.txt...
Adversary: completed transfer for my_file.txt
Adversary: file saved in ,client/my_file.txt
Adversary: shutting down server...

Process finished with exit code 0
```

四、实验结果

攻击完成，成功向 adversary 中的 client 文件夹上传了 bad_file.txt。文件内容正确。



my_file.txt: Hello there. I'd like to say SJTU NB!

bad_file.txt: Fudan NB! Stupid!

五、实验总结

Needham-Scroeder 协议能有效的使用 PKI 来制定临时会话密钥做到加密通信，但协议本身不够完善，使得中间人攻击能够利用重放操作，使得 server 以为 client 正在共享新的会话密钥，但实际上是一个 jiu 密钥，adversary 完成伪造 client 身份向 server 通信，这在实际应用中是极为不安全的。