**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. 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。  **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]) |  1. 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：   |  |  | | --- | --- | |  | 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(",") | |  |  | |  | # 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(",") | |  |  | |  | # 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。   1. 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。   |  |  | | --- | --- | |  | 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]) | |  | # 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") |     **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。 |