

基于口令的身份验证协议 实验报告

计97 朱美霖 2019013294

实验目的

本实验尝试实现 `Bellevin-Merritt` 身份验证协议。这一协议将传统的对称、非对称加密算法作为子模块使用，通过对基础密码算法库的调用可以加深对公钥密码、对称密码、散列函数工作方式的理解，并且有助于熟悉密码学编程接口。这一身份认证协议的实现过程中涉及大整数操作，通过编码实现这一协议还可以体验 `OpenSSL` 等安全协议库的实现流程。

实验内容

环境设置

- 语言: `Java`
- JDK 版本: `openjdk 11.0.14 2022-01-18`

设计思路

按照实验提示中的设计思路，本实验实现了 `Client` 和 `Server` 两个类用来模拟通信双方，通过 `Socket` 实现通信，由于仅在本机进行实验，因此具体的端口我直接写死了；其中 `Server` 作为 `B` 的角色用来代表该协议中的密钥分发方，`Client` 作为 `A` 的角色用来代表认证方；以 `Algorithm` 作为基类实现了 `RSA`，`AES` 和 `DES` 的算法类，用于进程中具体的加解密操作。

协议流程基本按照给出的步骤实现，下面介绍一些细节：

- 事先共享的口令 `pw` 写死在了 `Client` 和 `Server` 的进程里，为 `String pw = "Ep1phanyFillTo16"`，由于 `AES` 需要用 `pw` 作为密钥，因此进程最开始需要对其做散列处理，我使用的是 `MD5` 函数，具体操作如下：

```
MessageDigest md = MessageDigest.getInstance("MD5");
md.update(pw.getBytes("utf-8"));
byte[] digest = md.digest();
```

- 为了保证通信双方编码的一致性，通过 `Socket` 传输的信息都采用 `byte[]` 的形式，便于转换编码；
- 我在 `Algorithm` 基类中设置了协议中随机生成 N_A 和 N_B 的位数，默认为 `Integer N_len = 64`，即每个 $N_{A/B}$ 都是 `byte[64]`，能够保证不易被破解；
- 具体的算法加解密操作，我都调用了 `java.security` 包下的模块，封装起来较为容易，只需要知道如何调用即可；
- 在认证完成后，我增加了模拟 `Client` 和 `Server` 以 K_s 为密钥进行对称式密钥通信的过程，两边互相发送了一条写死在进城里的信息，然后 `Client` 退出连接，结束进程。
- 我实现的 `Client` 支持多线程并行，比较符合一个密钥分发中心对应多个终端的实际环境，且 `Server` 能够检测到 `Client` 结束进程，退出 `Socket` 连接并在终端发日志提示。`Server` 会持续等待下一个连接的 `Client`，不会退出。

其他的具体信息可以参看提交的视频。

实验结果

运行方式

```
cd bin
java Server
java Client x(x 为你希望运行的线程数)
```

结果

```
meilinzhu@DESKTOP-OJSLH38 /mnt/c/Users/meilinzhu/Desktop/Cyber
space Safety/Experiments/基于口令的安全身份认证协议/Bellovin-Merr
itt/bin > java Server SIGINT(2) # 1130 19:51:21
[Server] Bellovin-Merritt Protocol server is running ...
[Server] Accepted connection from /127.0.0.1:54604
[Server] Received unverified identity: CLIENT 0
[Server] Received pk_A cipher!
[Server] Sent double encrypted K_s cipher!
[Server] Received N_A cipher!
[Server] Sent N_A || N_B cipher!
[Server] Received N_2 cipher!
[Server] Decrypting & Verifying N_2 cipher ...
[Server] Authentication succeeded, Client identity: CLIENT 0
[Server] Read Message from CLIENT 0
[Server] <Client> RNG wins the MSI 2022!
[Server] Sending message ...
[Server] Sent message to CLIENT 0!
[Server] All simulation done!
[Server] Waiting for client to close connection ...
[Server] Connection closed!
[Server] Accepted connection from /127.0.0.1:54606
[Server] Received unverified identity: CLIENT 2
[Server] Received pk_A cipher!
[Server] Sent double encrypted K_s cipher!
[Server] Received N_A cipher!
[Server] Sent N_A || N_B cipher!
[Server] Received N_2 cipher!
[Server] Decrypting & Verifying N_2 cipher ...
[Server] Authentication succeeded, Client identity: CLIENT 2
[Server] Read Message from CLIENT 2
[Server] <Client> RNG wins the MSI 2022!
[Server] Sending message ...
[Server] Sent message to CLIENT 2!
[Server] All simulation done!
[Server] Waiting for client to close connection ...
[Server] Connection closed!
[Server] Accepted connection from /127.0.0.1:54610
[Server] Received unverified identity: CLIENT 1
[Server] Received pk_A cipher!
[Server] Sent double encrypted K_s cipher!
[Server] Received N_A cipher!
[Server] Sent N_A || N_B cipher!
[Server] Received N_2 cipher!
[Server] Decrypting & Verifying N_2 cipher ...
[Server] Authentication succeeded, Client identity: CLIENT 1
[Server] Read Message from CLIENT 1
[Server] <Client> RNG wins the MSI 2022!
[Server] Sending message ...
[Server] Sent message to CLIENT 1!
[Server] All simulation done!
[Server] Waiting for client to close connection ...
[Server] Connection closed!
^C

meilinzhu@DESKTOP-OJSLH38 /mnt/c/Users/meilinzhu/Desktop/Cyber
space Safety/Experiments/基于口令的安全身份认证协议/Bellovin-Merr
itt/bin > java Client 3 1131 19:51:21
[Client] Bellovin-Merritt Protocol client is running ...
[Client] Bellovin-Merritt Protocol client is running ...
[Client] Bellovin-Merritt Protocol client is running ...
[Client] Connected to server localhost/127.0.0.1:7654
[Client] Connected to server localhost/127.0.0.1:7654
[Client] Sent identity: CLIENT 0
[Client] Sent pk_A cipher!
[Client] Received pw_cipher!
[Client] Connected to server localhost/127.0.0.1:7654
[Client] Sent N_A cipher!
[Client] Received N_A || N_B cipher!
[Client] Verifying N_A ...
Client Authentication succeeded
[Client] Sending message ...
[Client] Message sent!
[Client] Read Message from server!
[Client] <Server> Congratulations for LPL!
[Client] All simulation done!
[Client] Closing connection ...
[Client] Sent identity: CLIENT 2
[Client] Sent pk_A cipher!
[Client] Received pw_cipher!
[Client] Sent N_A cipher!
[Client] Received N_A || N_B cipher!
[Client] Verifying N_A ...
Client Authentication succeeded
[Client] Sending message ...
[Client] Message sent!
[Client] Read Message from server!
[Client] <Server> Congratulations for LPL!
[Client] All simulation done!
[Client] Closing connection ...
[Client] Sent identity: CLIENT 1
[Client] Sent pk_A cipher!
[Client] Received pw_cipher!
[Client] Sent N_A cipher!
[Client] Received N_A || N_B cipher!
[Client] Verifying N_A ...
Client Authentication succeeded
[Client] Sending message ...
[Client] Message sent!
[Client] Read Message from server!
[Client] <Server> Congratulations for LPL!
[Client] All simulation done!
[Client] Closing connection ...
```

综上所述，我们已经基本实现了 Bellovin-Merritt 协议。