MyTLS report

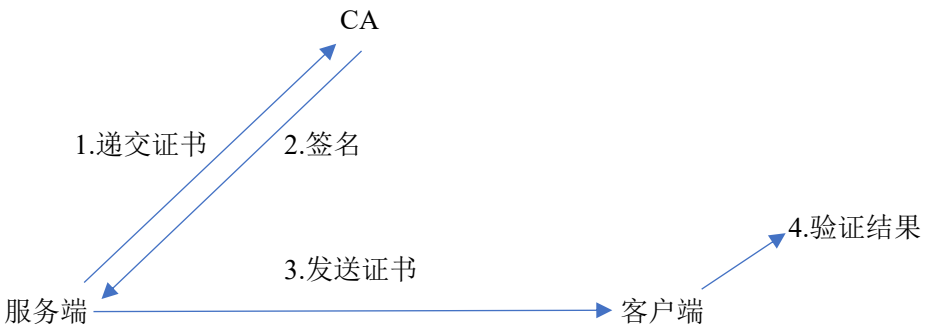
MyTLS 模拟系统主要的实现如下：

- 简单的 pki 体系
- rsa 密钥协商体系
- HMAC 算法体系
- 串行 socket 通信
- 基于上述技术构建的 MyTLS 协议

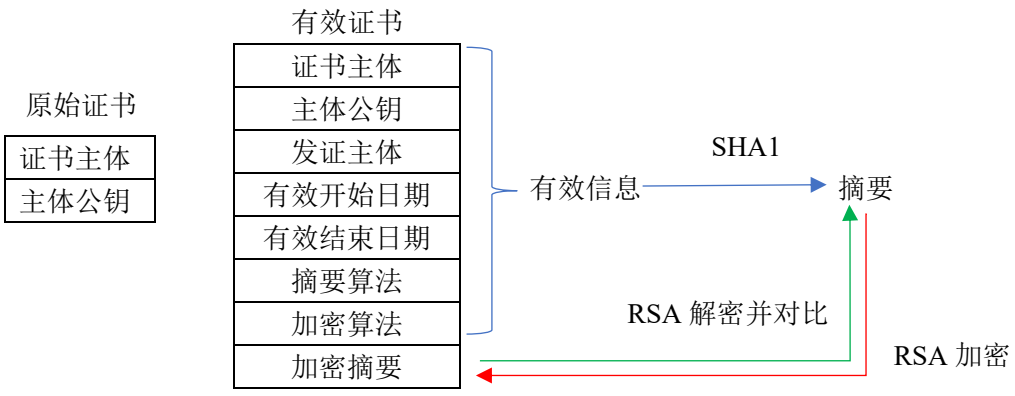
实现语言为 python3.7。

一、简单的 pki 体系

pki 体系主要由三个主体构成：CA，服务端，客户端。MyTLS 系统实现的是单向验证，通过 ca.py 提供的接口我们非常容易实现双向验证。本 pki 主要实现的功能是服务端向 ca 递交包含自己信息和公钥的证书，第三方 CA 用自己的私钥对证书进行签名并将证书返还服务端保存，当客户端需要验证服务端身份时就用 CA 的公钥进行验证，验证通过说明证书有效。示意图如下：

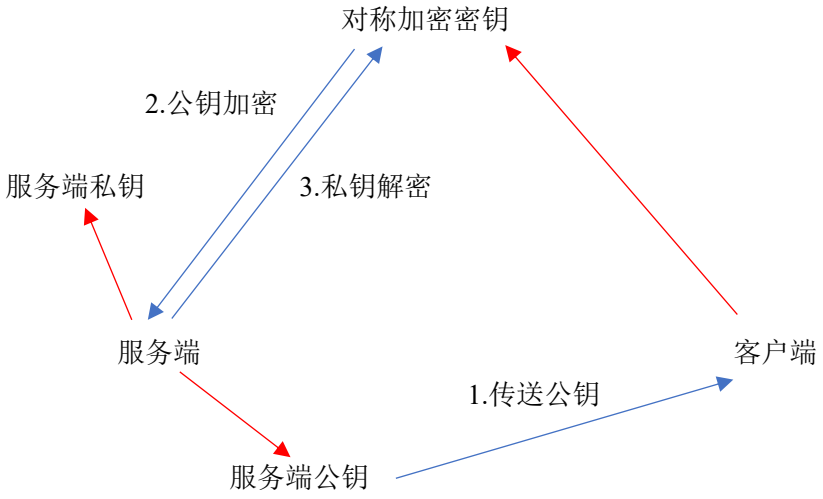


证书分为两种，为了方便实现通过 json 格式存储。一种是服务端产生的原始证书，一种是 CA 签名后发回的有效证书。签名过程根据传入的信息通过 SHA1 获得信息摘要，然后用本地的私钥加密信息摘要，并将加密后的摘要附在证书中返回；验证时根据传入的信息通过 SHA1 获得信息摘要，同时将加密的摘要部分用公钥解密，对比这两部分结果是否一致，然后返回结果。证书主要信息和验证方式示意图如下：



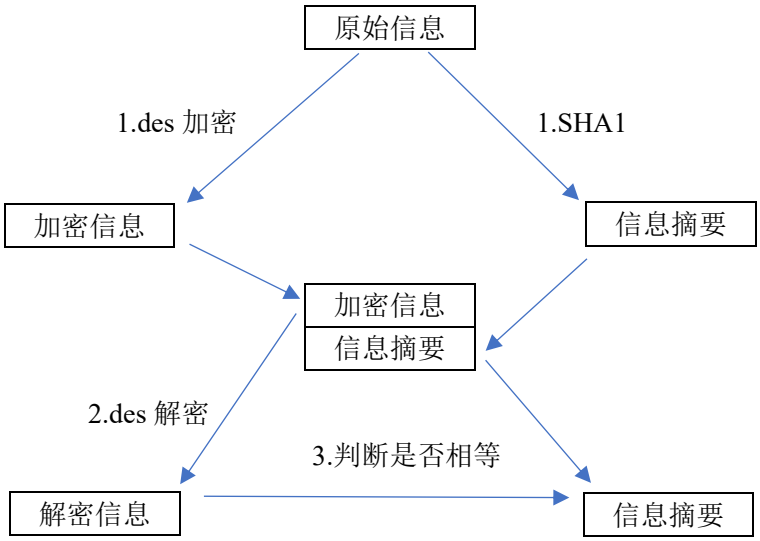
二、rsa 密钥协商体系

密钥协商体系的主要考虑因素是 rsa 虽然安全，但是加密效率低下，如果我们希望使用对称加密算法提高加密的效率，而对称加密使用的密钥又不能安全传递，就可以通过 rsa 加密对称加密使用的密钥后传输，这样就可以保证高效以及安全性。实现于 tlslib/librsa.py。示意图如下：



三、HMAC 算法体系

HMAC 体系的作用是，基于下层 des 算法，将明文加密，信道中只能截取到加密后的密文，获取明文信息摘要并且将摘要附在消息中传输。注意我们获取的是明文的摘要，因为这样可以保证即使攻击者篡改了信息，他也很难找到用我们的密钥解密之后得到的信息摘要。其实现位于 tlslib/libhmac.py。示意图如下：



四、串行 socket 通信

出于实现简单的考虑，我们使用 MySSL 类中包装了发送消息和接听消息的类，因为双方都是单线程实现，所以在 server.py 和 client.py 中采取的是一问一答的方式进行通信，即客户端先发送消息，服务端首先阻塞等待消息，客户端发送后进入阻塞等待消息的阶段，服务端成功收到消息后可以开始发送消息，如此循环往复。

五、MyTLS 协议实现

由上面各种安全技术组成的 MyTLS 体系分为三个部分，下面就每个部分实现的细节进行详细说明。

1.CA

CA 是一个单独的主体，在 MyTLS 系统运行前通过 rsa 算法产生一对密钥，私钥保密，公钥可以传输给其他主体。CA 的实现位于 tlslib/ca.py 和 tlslib/libca.py，其中 libca.py 提供的是签名和验证的函数。ca.py 产生了一个独立的 CA 主体，它提供三个函数。getCa 将以 json 格式传入的证书所有的字段全部加起来交给 libca 加密，verify 函数将待验证证书除了数字指纹信息之外的部分加起来与指纹一起交给 libca 验证，getPublicKey 函数可以为其他主体返回自己证书的公钥便于传输。

2.服务端

开始通信之前服务端需要准备好一对公私钥，并且存储 CA 签名后的服务端证书。

服务端首先在某一端口进行监听，等待客户端发来的 hello 消息。收到 hello 消息之后，从客户端支持的密钥协商算法中选一个，然后发送一个包含服务端证书、选择的协商算法以及是否需要客户端提供证书的消息（本实现中默认不需要），然后开始等待客户端的下一个消息。客户端下一次的消息中包含的是用证书中公钥加密的主密钥，服务端可以用私钥解开并存储。之后回复一个准备好加密套件的结束消息，等待客户端回复。收到回复后可以开始通信。

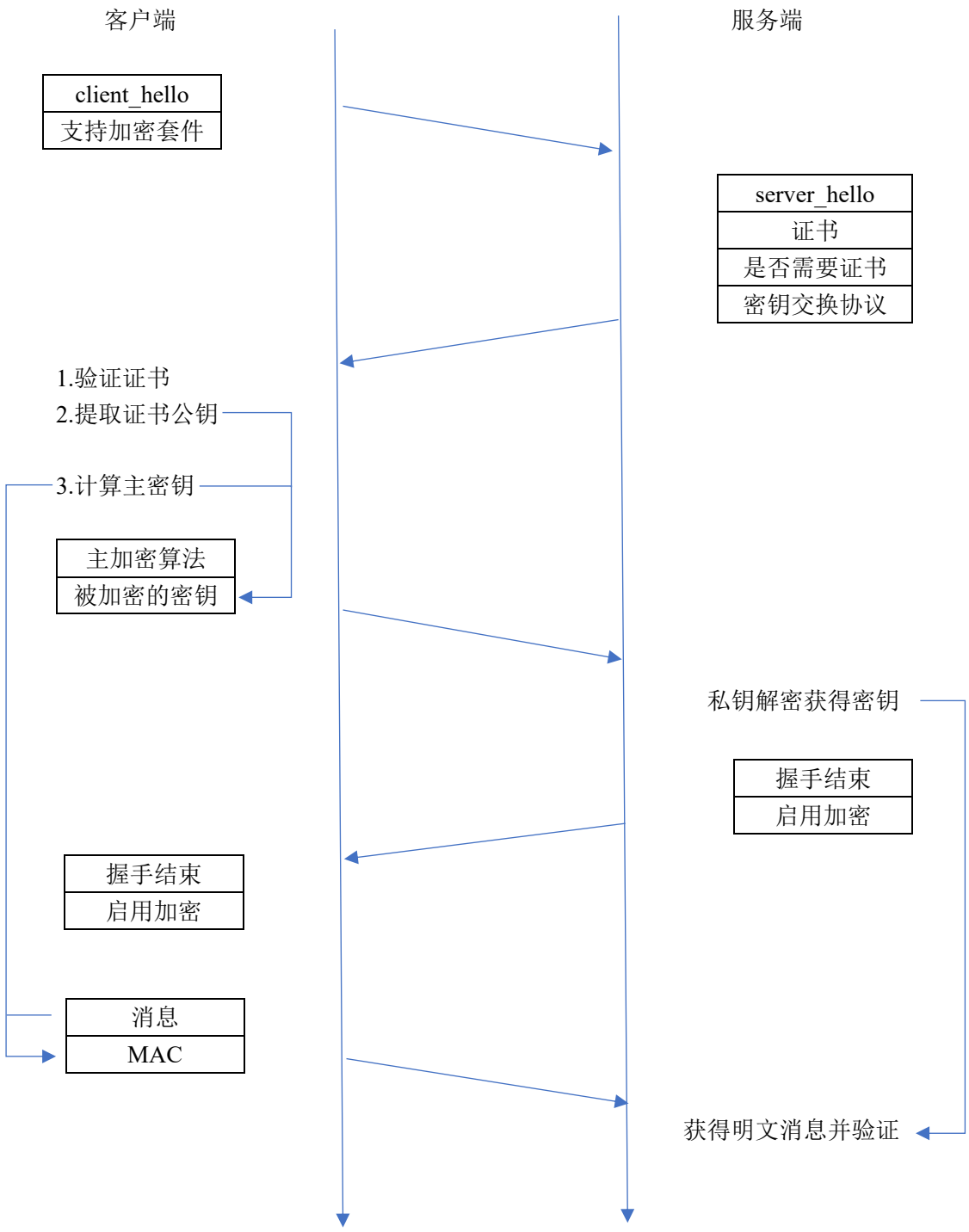
通信过程中使用 hmac 加密消息，并且验证消息完整性，如果验证失败就可以认为消息被篡改，需要结束本次通信。

3.客户端

客户端已知服务端的开放端口和 ip，向服务端发送一个包含自己支持的加密套件的 hello 消息，然后开始等待服务端回复。客户端在收到服务端发送的证书之后可以结合服务端的公钥进行验证，如果验证通过可以确认证书在签名后没有被篡改，否则弹出警告。然后客户端生成一个 8 位的随机数作为主密钥，用已验证的证书中包含的公钥加密这个密钥，再加上主密钥使用的加密算法传送给服务端。最后收到服务端的结束握手消息后同样发一个确认启用加密套件的消息，准备开始正式通信。

通信时使用 hmac 加密传送的消息，并且收到消息后验证消息的完整性，验证失败弹出警告并关闭通信接口。

4.MyTLS 系统的示意图



5.安全性分析

pki 系统的安全性主要依赖于 CA 的私钥、SHA1 算法以及 RSA 算法的安全性。如果 CA 私钥泄露，攻击者就可以通过私钥签发非法证书，而 SHA1 和 RSA 加密保证了证书不被非法篡改。就本系统而言，服务端发送给 CA 的原始证书可以被截获和篡改，不过现实中 CA 在签发证书前有严格的审核过程，这些不在本实验的讨论范围之内。

密钥交换系统的安全性取决于 RSA 算法的安全性，攻击者通过在信道截获的加密后的主密钥无法还原出密钥明文。HMAC 的安全性依赖于密钥交换系统得到的安全的加密密

钥，不过位数太小的密钥依然可能被猜中。而信息完整性的验证依赖于 SHA1 算法和 DES 算法的安全性，攻击者很难根据密文和摘要还原明文，或者在没有密钥的情况下构造出合法的密文和摘要。

6.实验结果

首先运行 server.py 等待连接，

```
PS C:\learning\learning5\信息系统安全\lab\tls\tls> python server.py
[ server ]  client_hello
algorithm:RSA

check True
[ server ]  change sessionKey
key : 33518757

[ server ]  client finish_handshake

[ server ]  hello

server > hi
[ server ]  goodboy
```

运行 client.py 之后可以看到双方进行握手，结束后开始正常通信。

```
PS C:\learning\learning5\信息系统安全\lab\tls\tls> python client.py
[ client ]  server_hello
public_key :
 -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAoMYd1ZXKqKFWvw/gDdNP
GPDyEBlZpy6+tqNeqCEKUz08dokAeWGb0HII3uoxO4+xHlUhFYI99d3vd+LV0FPD
z+WEje+pxHww3pPwph0cMjjkctMfp9Ke+ohz3nZSftyVfkumcgHLTinm9ZPZUO/6
DJgYjcaUMERZohSrElpYBciBPi08f63GBl91aszrN7w3fioOwUwmiKwMmwlAFVe9
tZRN7VjUIMPFvBVMRIY6unkMAvRJBwB+uObCNX0TN74LbvPMLdwrYEA1NQzgbM2E
mkP5W0clo0L1CtFFawN2WZJciOXwgl+99he/68RKmhPWP2egd0pmlWK3g6iuSSL7
UwIDAQAB
-----END PUBLIC KEY-----
certificate verification :  True

[ client ]  server finish_handshake

client > hello
[ client ]  hi

client > goodboy
[ client ]  yes
```

Reference

des 加密：https://www.wj0511.com/site/detail.html?id=130
rsa 加解密与签名验证：https://blog.csdn.net/weixin_43790276/article/details/100063132

Appendix

用户 api 介绍

class MySSL：封装了 MyTLS 的握手和通信的类

def __init__(self,subject:str,port:int):

通过 subject 为”server”或”client”生成不同的类，port 指定为监听（服务端）或连接（客户端）的端口号。

def handshake(self):

实现 tls 需要的握手确认阶段，如果成功可以继续传递消息，否则 socket 连接被关闭，通信结束。

def sendMsg(self,msg:str):

传递要发送的消息字符串给通信另一方。

def recvMsg(self):

接受对方消息，如果 hmac 验证失败通信关闭返回 False 和空字符串，如果成功返回 True 和收到的消息字符串。