网络与信息安全课内实验三报告

对称、非对称加解密

### 本次实验目的

〉 1. 加深对非对称，对称加解密算法、散列函数的理解。

〉 2. 了解常用加密工具包以及相关库函数的使用。

〉 3. 了解ssl协议

### 实验原理

**一、对称加密**

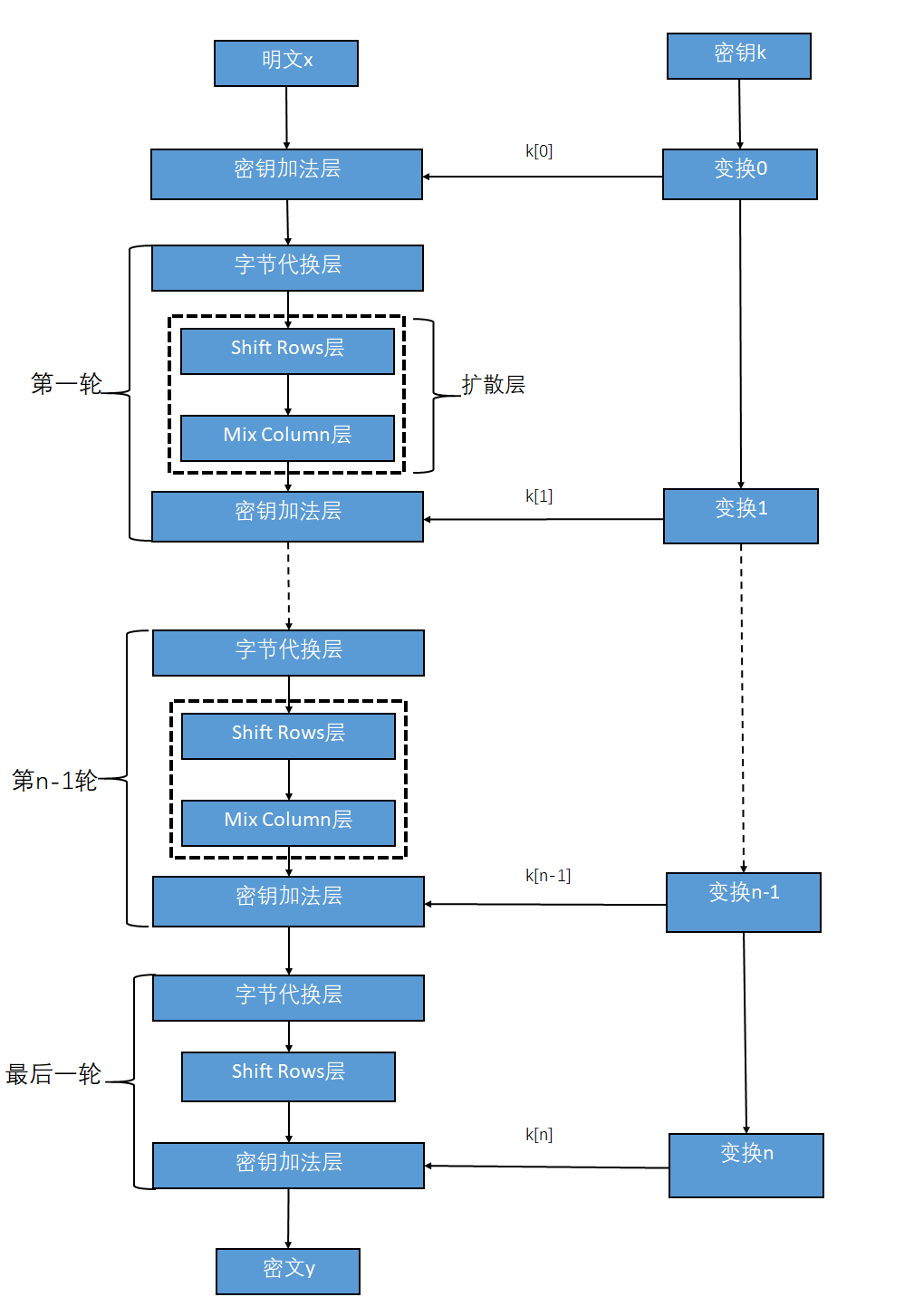
加密算法是公开的，采用单钥密码系统，使用一个秘钥加密，必须使用相同的秘钥才解密。

优点：算法公开、计算量小、加密速度快、加密效率高

缺点：密钥管理复杂，在数据传送前，发送方和接收方必须商定好秘钥，然后使双方都能保存好秘钥，一旦密钥泄露，加密信息将不安全

常见算法：DES、3DES、Blowfish、IDEA、RC4、RC5、RC6、AES

**AES算法**



AES算法主要有四种操作处理：

1.密钥加法层：

将明文或上一轮的密文与扩展密钥进行按位异或操作。

扩展密钥是从原始密钥通过密钥扩展算法生成的，每轮使用一个不同的扩展密钥，

在加密过程开始之前，还有一次初始的轮密钥加。

2.字节代换层：

使用一个固定的S\_box表对状态矩阵中的每个字节进行替换。S\_box表是一个16×16的二维数组，存储了每个可能的字节值到另一个字节值的映射。

3.行位移层：

对状态矩阵的行进行循环位移。

保持矩阵的第一行不变，第二行向左移动一个字节、第三行向左移动2个字节、第四行向左移动3个字节。

4.列混淆层：

使用一个固定的矩阵对状态矩阵的列进行线性变换。

列混淆操作是AES算法中主要的扩散元素，它混淆了输入矩阵的每一列，使输入的每个字节都会影响到4个输出字节。行位移子层和列混淆子层的组合使得经过三轮处理以后，可能使得矩阵的每个字节都依赖于16个明文字节。

1. 加密轮次：

对于128位密钥，加密轮次有10轮；

对于192位密钥，加密轮次有12轮；

对于256位密钥，加密轮次有14轮。

每一轮都包括上述的四个步骤，但最后一轮略有不同，最后一轮不包括列混淆层，因为列混淆层在解密过程中需要其逆操作，而直接省略它可以使解密过程更加高效。

解密过程：

解密是加密的逆过程，步骤的顺序相反，使用的也是相应的逆操作：逆列混淆、逆行位移、逆字节代换、轮密钥加

**二、非对称加密**

需要两个密钥，公开密钥和私有密钥，加密和解密使用不同的秘钥，公钥加密的信息只有用对应的私钥才能解密，私钥加密的信息只有用对应的公钥才能解密。

优点：安全，即使密文被拦截、公钥被获取，但是无法获取到私钥，也就无法破译密文。作为接收方，务必要保管好自己的密钥。

缺点：加密算法复杂，安全性依赖算法与密钥，而且加密和解密效率很低

常见算法：RSA、DSA、ECC

工作流程：A生成一对非对称秘钥，将公钥向所有人公开，B拿到A的公钥后使用A的公钥对信息加密后发送给A，经过加密的信息只有A手中的私钥能解密。这样B可以通过这种方式将自己的公钥加密后发送给A，两方建立起通信，可以通过对方的公钥加密要发送的信息，接收方用私钥解密信息。

**RSA算法**

1. 密钥计算方法

选择两个大素数p和q

计算n = p × q和z = (p-1) × (q-1) /\*n表示欧拉函数\*/

选择一个与z互质的数d

找到一个e使满足e × d = 1 (mod z)

公开密钥为(e,m)，私有密钥为(d,m)

1. 加密方法

将明文看成比特串，并划分成k位的块P（k是满足2 × k < n的最大整数）。对每个数据块P，加密，C即为P的密文

1. 解密方法

对每个密文块C，解密，P即为明文

**三、散列函数（Hash）**

就是把任意长度的输入（又叫做预映射pre-image）通过散列算法变换成固定长度的输出，该输出就是散列值。这种转换是一种压缩映射，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，因此不可能从散列值来唯一的确定输入值。简单的说就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。

**数字签名**

在正式的数字签名中，发送方首先对发送文件采用哈希算法，得到一个固定长度的消息摘要( Message Digest)；再用自己的私钥( Secret key，SK)对消息摘要进行签名，形成发送方的数字签名。数字签名将作为队件和原文一起发送给接收方；接收方首先用发送方的公钥对数字签名进行解密得到发送方的数字摘要，然后用相同的哈希函数对原文进行哈希计算，得到一个新的消息摘要，最后将消息摘要与收到的消息摘要做比较。

普通数字签名算法有RSA、ElGamal、Fiat-Shamir、Guillou- Quisquarter、Schnorr、Ong-Schnorr-Shamir数字签名算法、Des/DSA，椭圆曲线数字签名算法和有限自动机数字签名算法等。

1.首先，用户A向证书颁发机构提交个人信息，申请证书。通过CA审核后，CA生成用户A的证书，证书中包括了A的公钥和私钥还有CA的数字签名。证书颁发机构CA本身拥有一对密钥，这是对CA所颁发的证书进行数字签名和保密的基础，绝不能泄露。

2.用户A收到的证书中包括了带有CA数字签名的，专属A公钥和私钥，CA的数字签名确保了别人不能伪造用户A的公钥和私钥。

3.同时，用户B也必须信任给用户A颁发证书的第三方认证机构CA，即用户B拥有CA颁发的"CA公钥"。

4.通信时，用户A向用户B发送的数据包中的"加密的摘要"上有用户A的数字签名，“A公钥”上有认证机构CA的数字签名。用户B收到数据包之后，先要验证收到的“A公钥”是否来源合法：是认证机构颁发的带有CA签名的公钥吗？用户B并不信任用户A，但是用户B信任第三方认证机构CA。所以，用户B先使用证书颁发机构颁发的“CA公钥”验证收到的“A公钥”是否由同一认证机构颁发，是否在颁发之后更改过。

5.验证通过后，用户B便相信收到的“A公钥”确实来自真实的用户A。随后再使用“A公钥”对“加密的摘要”进行解密，进行上文提到的对比操作，以判断文件是否更改。

**四、ssl协议通信握手过程**

**基本概念**

SSL：Secure Socket Layer，安全套接字层，位于可靠的面向连接的网络层协议和应用层协议之间的一种协议层。SSL通过互相认证、使用数字签名确保完整性、使用加密确保私密性，以实现客户端和服务器之间的安全通讯。该协议由两层组成：SSL记录协议和SSL握手协议。

TLS：Transport Layer Security，传输层安全协议，用于两个应用程序之间提供保密性和数据完整性。该协议由两层组成：TLS记录协议和TLS握手协议。

SSL和TLS协议可以为通信双方提供识别和认证通道，从而保证通信的机密性和数据完整性。TLS协议是从Netscape SSL 3.0协议演变而来的，不过这两种协议并不兼容，SSL已经逐渐被TLS取代，所以下文就以TLS指代安全层。TLS握手是启动HTTPS通信的过程，类似于TCP建立连接时的三次握手。在TLS握手的过程中，通信双方交换消息以相互验证，相互确认，并确立它们所要使用的加密算法以及会话密钥（用于对称加密的密钥）。可以说，TLS握手是HTTPS通信的基础部分。

**握手过程**

1. "client hello"消息：

客户端通过发送"client hello"消息向服务器发起握手请求，该消息包含了客户端所支持的TLS版本和密码组合以供服务器进行选择，还有一个"client random"随机字符串。

1. "server hello"消息：

服务器发送"server hello"消息对客户端进行回应，该消息包含了数字证书，服务器选择的密码组合和"server random"随机字符串。

1. 验证：

客户端对服务器发来的证书进行验证，确保对方的合法身份，验证过程可以细化为以下几个步骤：

·检查数字签名

·验证证书链

·检查证书的有效期

·检查证书的撤回状态（撤回代表证书已失效）

1. "premaster secret"字符串：

客户端向服务器发送另一个随机字符串"premaster secret (预主密钥)"，这个字符串是经过服务器的公钥加密过的，只有对应的私钥才能解密。

1. 使用私钥：

服务器使用私钥解密"premaster secret"。

1. 生成共享密钥：

客户端和服务器均使用 client random，server random 和 premaster secret，并通过相同的算法生成相同的共享密钥KEY。

1. 客户端就绪：

客户端发送经过共享密钥KEY加密过的"finished"信号。

1. 服务器就绪：

服务器发送经过共享密钥KEY加密过的"finished"信号。

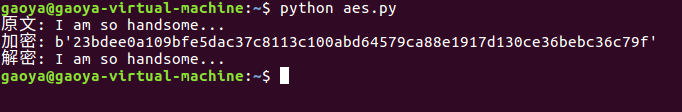
1. 达成安全通信：

握手完成，双方使用对称加密进行安全通信。

### 实验步骤

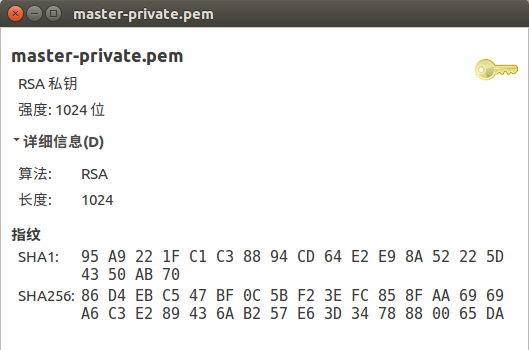
**一、对称加密算法实验**

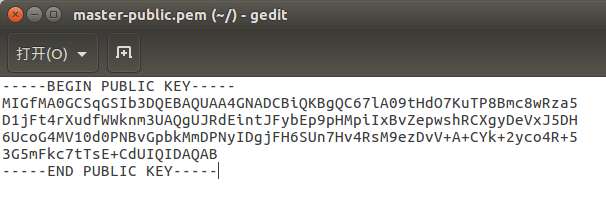
使用AES算法加密字符串，并成功解密



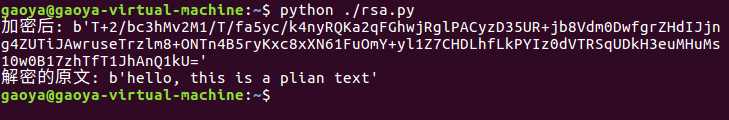
**二、非对称加密算法实验**

生成RSA公私钥对



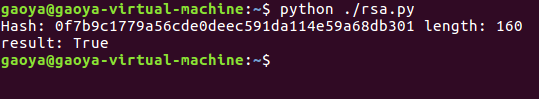


使用公钥加密字符串，并使用私钥成功解密



**三、数字签名**

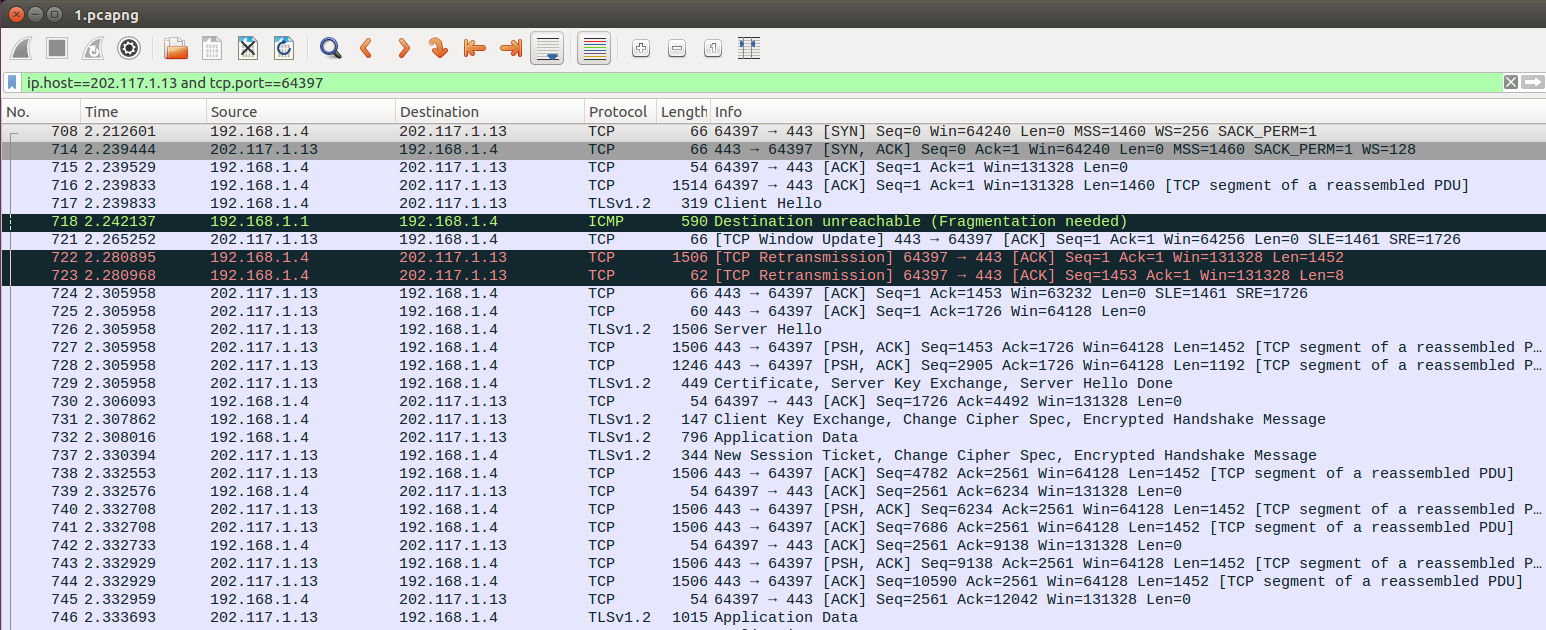
使用RSA算法进行数字签名



**四、抓包观察 ssl 协议通信握手过程**

使用Wireshark抓包工具观察HTTPS通信过程中的SSL握手过程

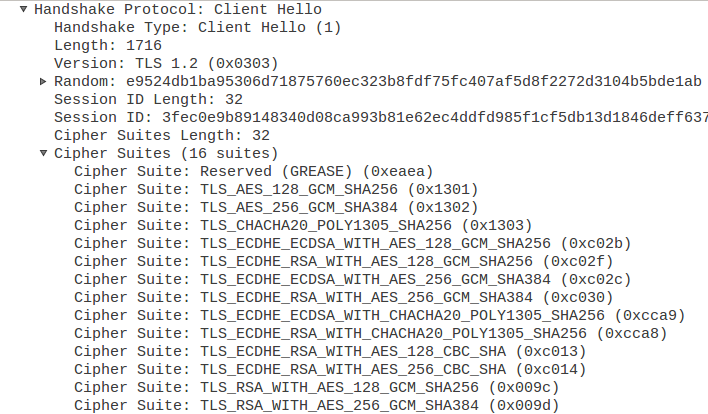
**数据包总览**



**数据包解析**

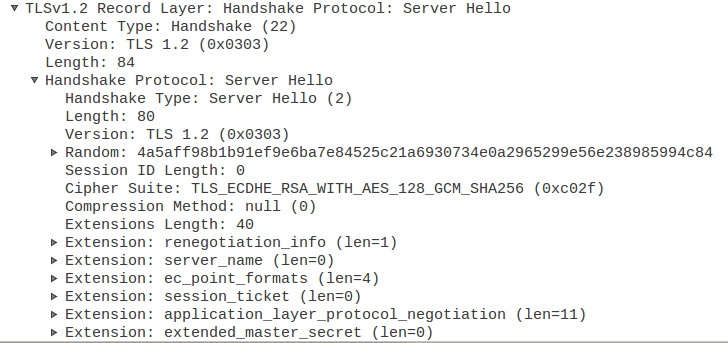
**TLS第一次握手：**

客户端发送Client Hello消息，跟服务器打招呼，其中主要携带：客户端的TLS版本号(Version)、客户端支持的加密套件列表(Cipher Suites)、客户端生成的随机数(Client Random)

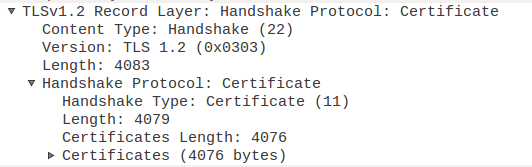


**TLS第二次握手：**

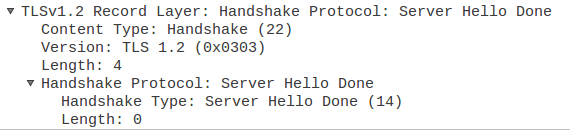
服务器收到客户端的Client Hello消息后，首先回复Server Hello消息，其中主要包括：服务器确认支持客户端的TLS版本(Version)、服务器从客户端发来的加密套件列表中选出一个最合适的加密组合(Cipher Suite)、服务器生成的随机数(Server Random)



随后，服务器为了证明身份，会给客户端发送数字证书，即Certificate消息，包含CA机构为服务器颁发的公钥、CA机构签名等信息



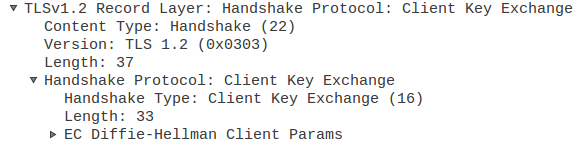
最后，服务器发送Server Hello Done消息给客户端，这条消息中并没有其他有价值的信息，仅仅是为了通知客户端第二次握手中服务器的所有消息都已发送完毕



**TLS第三次握手：**

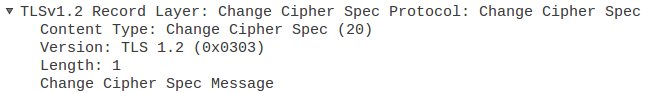
客户端在收到服务器返回的Server Hello和Certificate消息后，会使用操作系统内置的CA机构的公钥对证书进行解密，如果解密成功，得到数据原文及摘要值H1，然后客户端使用与CA机构相同的摘要算法（散列算法，SHA或MD5）对数据原文进行计算得到 摘要值H2，比较H1与H2，若完全相同则说明证书合法且未被其他人篡改，从而拿到服务器的RSA公钥。

  随后，客户端生成TLS握手过程中的第三个随机数：PreMaster，并用服务器的公钥对其加密，通过Client Key Exchange消息发给服务器：

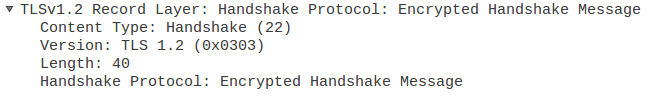


接着，客户端使用它所拥有的三个随机数 Client Random 、Server Random、PreMaster 去生成 对称加密的密钥：Master Secret。客户端通过 Change Cipher Spec 消息将Master Secret 对称密钥发送给服务器，并通知服务器开始使用对称加密的方式进行通信。

  在此之前的握手消息都是明文的，但只要出现了"Change Cipher Spec" 消息，之后的握手消息就都是密文了，wireshark抓到的数据包也会是乱码。

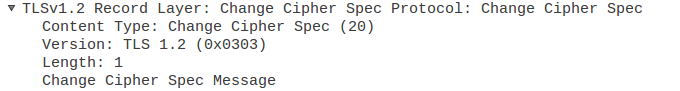


最后，客户端发送Encrypted Handshake Message消息，将之前发送的所有数据做成摘要，使用Master Secret 对称密钥加密（这条消息已经是对称加密），供服务器验证之前握手过程中的数据是否被其他人篡改。

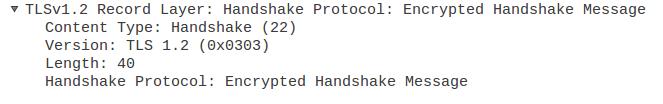


**TLS第四次握手：**

服务器在收到客户端的Client Key Exchange消息后，使用RSA私钥对其解密，得到客户端生成的随机数PreMaster，至此服务器也拥有了与客户端相同的三个随机数：Client Random、Server Random、PreMaster，服务器也使用这三个随机数计算对称密钥，将计算后的结果通过Change Cipher Spec消息返回给客户端。



服务器通过Encrypted Handshake Message消息将之前握手过程中的数据生成的摘要使用对称密钥加密后 发给 客户端，供客户端进行验证，至此TLS四次握手完毕。



**回答问题**

1.AES加密中的iv变量：

iv（Initialization Vector，初始化向量）是AES加密中的一个重要参数。

它与密钥一起用于加密过程，确保即使相同的明文块在多次加密时也能产生不同的密文。

iv通常是随机生成的，并且与密文一起存储或传输，以便解密时使用。

RSA算法的不同使用方式：

RSA算法可以用于公钥加密、私钥解密，也可以用于私钥加密、公钥解密。

公钥加密、私钥解密通常用于加密通信，确保信息只能由持有私钥的接收方解密。

私钥加密、公钥解密则用于数字签名，确保信息由持有私钥的发送方签名，且签名可以被任何持有公钥的接收方验证。

1. 回答AES加密中iv变量的作用。

iv参数是一个固定长度的随机数，通常与密钥一起作为输入，用于初始化加密算法的状态。在每次加密操作中，都需要使用一个唯一的iv参数，防止明文重复加密后的密文重复出现。如果使用相同的iv参数对相同的明文进行多次加密，由于密钥相同，密文也会相同。这样一来，攻击者可以根据密文的重复性进行破解。也能增加密码的安全性。iv参数的引入可以使得每次加密的结果都不一样，即使明文相同。这样可以有效防止一些已知明文攻击和选择明文攻击。

1. 思考问题：RSA【公钥加密，私钥解密】和【私钥加密，公钥解密】算法一样吗？为什么？

公钥加密，私钥解密：

是RSA加密算法的常规用法。发送者使用接收者的公钥来加密数据，然后接收者使用自己的私钥来解密数据。这种方式确保了只有持有私钥的接收者才能解密数据，从而保证了数据的安全性。

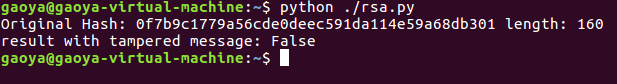
私钥加密，公钥解密：

这种用法在RSA算法中并不常见。发送者使用接收者的私钥来加密数据，然后接收者使用自己的公钥来解密数据。这种做法存在安全隐患。因为私钥是保密的，如果发送者能够获取接收者的私钥来进行加密，那么这意味着私钥可能已经泄露，从而破坏了RSA算法的安全性基础。

1. 书写数字签名的注释，每行都干了些什么？并任意举一个例子使得result=False。
2. #导入必要的库
3. import base64
4. from Crypto.Hash import SHA
5. from Crypto.Signature import PKCS1\_v1\_5 as Signature\_pkcs1\_v1\_5
6. from Crypto.PublicKey import RSA
8. n = b'This is a test message'  #待签名的消息
9. h = SHA.**new**()  #创建一个SHA哈希对象
10. h.update(n)  #使用待签名的消息更新哈希对象
11. print('Hash:', h.hexdigest(), 'length:', len(h.hexdigest()) \* 4)  #打印哈希值和它的长度
12. sign\_txt = 'sign.txt'  #定义签名文件的名称
14. #读取私钥并导入
15. with open('master-private.pem') as f:
16. key = f.read()
17. private\_key = RSA.importKey(key)
19. hash\_obj = SHA.**new**(n)  #创建一个新的哈希对象，用于签名
20. signer = Signature\_pkcs1\_v1\_5.**new**(private\_key)  #使用私钥创建一个签名对象
21. d = base64.b64encode(signer.sign(hash\_obj))  #对哈希对象进行签名，并将签名结果进行Base64编码
23. #将签名写入文件
24. f = open(sign\_txt, 'wb')
25. f.write(d)
26. f.close()
28. #读取公钥并导入
29. with open('master-public.pem') as f:
30. key = f.read()
31. public\_key = RSA.importKey(key)
33. sign\_file = open(sign\_txt, 'r')  #打开签名文件并读取签名
34. sign = base64.b64decode(sign\_file.read())  #对读取的签名进行Base64解码
35. h = SHA.**new**(n)  #创建一个新的哈希对象，用于验证
36. verifier = Signature\_pkcs1\_v1\_5.**new**(public\_key)  #使用公钥创建一个验证对象
37. print('result:', verifier.verify(h, sign))  #验证签名是否有效

将result改为False

1. tampered\_message = b'This is a test message'
2. h\_tampered = SHA.**new**(tampered\_message)
3. verifier = Signature\_pkcs1\_v1\_5.**new**(public\_key)
4. print('result with tampered message:', verifier.verify(h\_tampered, sign))



1. 详细分析ssl协议通信握手过程。

见上

1. 写出实验中遇到的问题与心得体会。

本次加密技术实验让我收获颇丰，我掌握了各种加密技术的工作原理，还通过实践操作加深了对这些技术的理解。我相信这些知识对我未来的学习会产生积极的影响。同时，我也深刻认识到信息安全的重要性以及加密技术在保护信息安全方面的巨大作用。