实验三：加解密

一、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

二、实验平台

Server：ubuntu 虚拟机，安装 Apache24
Attacker：ubuntu 虚拟机，与 server 处于同一网段（局域网）

三、实验过程及结果分析

步骤一：对称加密算法的字符串的加解密

本步骤中，使用 Python 对字符串进行加解密。Python 中常用的加解密相关库为 Crypto。安装命令为 pip3 install pycrypto。安装过程如下：

```
nakno@nakno-VirtualBox:~/桌面$ sudo pip3 install pycrypto
[sudo] nakno 的密碼：
Collecting pycrypto
  Downloading pycrypto-2.6.1.tar.gz (446 kB)
     |                                | 446 kB 928 kB/s
Building wheels for collected packages: pycrypto
  Building wheel for pycrypto (setup.py) ... done
  Created wheel for pycrypto: filename=pycrypto-2.6.1-cp38-cp38-linux_x86_64.whl
 size=498474 sha256=32dfad41f81eafb49ba018ff55c432afd379ca5e3bb2108be57e13cae870
f3c8
  Stored in directory: /root/.cache/pip/wheels/d0/99/d0/0298ea019d63f1d63a0965b9
944b719e875f9bd6ffc6dcf293
Successfully built pycrypto
Installing collected packages: pycrypto
Successfully installed pycrypto-2.6.1
```

安装加解密库 pycrypto 后在命令行中运行 aes.py。

所运行的 Python 代码使用 AES 对称加密算法进行字符串的加密与解密。对称加密即使用相同的密钥进行加密和解密。在此代码中，AES 被用于加密和解密。CBC 模式是 AES 的工作模式之一，其需要一个初始化向量来增加加密的复杂性。

操作过程大致为首先将密钥 key 转换为字节，并设置 AES 加密模式为 CBC。

再使用 add_to_16()函数确保文本长度为 16 的倍数。

创建 AES 加密器 cryptos = AES.new(key, mode, iv)，其中 iv 是固定的 16 字节初始化向量。使用 cryptos.encrypt(text) 对文本进行加密。随后 b2a_hex()将加密后的字节转换为十六进制字符串，以便能够打印和存储。返回的是加密后的文本，通常是不可打印的字节，通过转为十六进制字符串来展示。

相似的，decrypt(text, key)解密经过加密后的文本 text。将密钥 key 转换为字节。创建 AES 解密器 cryptos = AES.new(key, mode, iv)。

再使用 cryptos.decrypt(a2b_hex(text)) 解密十六进制字符串 text，a2b_hex() 将十六进制字符串转回字节数组。解密后的文本可能包含填充的 \0 字符，使用 rstrip('\0')

去除这些填充字符，恢复原始文本。

运行后输出结果如下：

```
nakno@nakno-VirtualBox:~$ sudo nano aes.py
[sudo] nakno 的密碼：
nakno@nakno-VirtualBox:~$ sudo python3 aes.py
原文: I am so handsome...
加密: b'23bdee0a109bfe5dac37c8113c100abd64579ca88e1917d130ce36bebc36c79f'
解密: I am so handsome...
nakno@nakno-VirtualBox:~$
```

思考题 1 ：回答 AES 加密中 iv 变量的作用。

iv，即 Initialization Vector，初始化向量的主要作用是增强加密的强度。 通过引入 iv，即使加密相同的数据，在相同密钥下也会生成不同的密文。iv 的存在确保了相同的明文在不同加密过程中会生成不同的密文，从而避免因相同密文泄露而导致的攻击风险。它增加了加密过程的唯一性，使得攻击者更难通过分析密文推断出明文或密钥。

步骤二：使用 RSA 的加解密

在命令行中运行 rsa.py，生成公钥和私钥结果如下：

```
nakno@nakno-VirtualBox:~$ sudo python3 rsa2.py
-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgQCw7CuN/rVN8vI9bcdDIIotdbDxnJRY2nFlA5+M6n36MUZODPo9
mhh/8Yi66ytduyEzzdTWHk+gZjTq9ZqD5gSZKdcoYkH9vVJDiFlxQs84Ku/KWblQ
QMk2msiKYPVLUsUP8IGiNXcYPkc115pse57HNqy9btCktE6OttHRq0z/TQIDAQAB
AoGAAJKDAY6VX48toW09e8YbLFJ5JA7xm+f4/e6pEXeguLJ4G8CG/rQE6EULwC8K
kN8OGHDOA3NNpqMuAKSSv2J7GVJcJ3t424ItHxu1qqiZL29zdkurNpLenKetAmu7
x8wf8wATeO1dzAkZIlTfSYT+zJ2Wt+JdaPm4onlNlIZWeTECQQDikRy6DaJQ56SB
wM6Lg8WmwdvdV1+HUwNbc+xt3iUBNDptaIq3EhzCZWCKeBRFmNINLPuY8A9wlpBs
4bCAgmKzAkEAx+gMG7gA0Rb09yuC/++iDMGO7nw2qYoYVXi6CJVNExROPH1QcXHH
r5oRq8iDV3MbVcZuagzZ76lEZtqaBVwV/wJAYkqyO8VOdesI44HSGZTBWucfi5cs
SnkpK7PPDpshUrRl5LFF/MZO08dHastNgCQcfLrXLDJ5gE7RuWrwN9BZIQJAPna1
gdvG9Y6U0L7Z2hsIqa+5d698bRFndR2aE6CDStjncWM+xZPl72rW5rRqxeKDwhOU
4LJPTuG3K+RWiCI0BQJAZfOUWCDg41dx8PTmolPt6/m5NppEHzrymAAkSLHcfv++
3CGW6YCtc8f7aK7Fw39XcsLJf4nYT1pJJJX89huABw==
-----END RSA PRIVATE KEY-----

-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCw7CuN/rVN8vI9bcdDIIotdbDx
nJRY2nFlA5+M6n36MUZODPo9mhh/8Yi66ytduyEzzdTWHk+gZjTq9ZqD5gSZKdco
YkH9vVJDiFlxQs84Ku/KWblQQMk2msiKYPVLUsUP8IGiNXcYPkc115pse57HNqy9
btCktE6OttHRq0z/TQIDAQAB
-----END PUBLIC KEY-----
```

同时使用公钥加密，并使用私钥解密，得到如下结果。

```
加密后: b'CksshfpvJgoZ5xc9I22+dzHWxM7lDF+3irSBJ68aO4DEP5QXwwtdLcBZFVvHpcsAxcnBVNbQEjTebsxjV34rQU
m8YE9Ghc+ss8jd6EZXujRwPjr+s3Y9h22s7mxvk66w7kD4eNSfnvrjmc2VZ0oARbn5kiQXvLLqX+uvXihHgwQ='
解密后的原文: hello, this is a plain text
```

由于指导书所给出的示例代码在运行过程中出现 AttributeError: module 'time' has no attribute 'clock'的报错，即在 pycrypto 或 pycryptodome 中使用了已被移除的 time.clock()函数，这在 Python 3.8 及更高版本中已不再可用。

因此修改示例代码将 time.clock()替换为 time.perf_counter()

```
import time
t = time.perf_counter()
```

代码使用 RSA 公钥加密、私钥解密消息。采取 OAEP 填充方案和 SHA-256 哈希算法来增强安全性。加密和解密过程中的数据通过 Base64 编码和解码处理，以便于传输和存储。

思考题 2 ：RSA 的【公钥加密，私钥解密】和【私钥加密，公钥解密】是一样的吗？为什么？

在 RSA 算法中，公钥加密和私钥解密与私钥加密和公钥解密并不是相同的过程。

【公钥加密，私钥解密】用于确保机密性，加密过程中信息由接收者的公钥加密。解密过程只有拥有对应私钥的接收者能够解密信息。这种方式用于保证信息的保密性，确保只有

接收者能够读取消息内容。

【私钥加密，公钥解密】用于验证真实性和完整性，加密过程中发送方使用自己的私钥对信息进行加密。解密过程任何人都可以使用发送方的公钥解密信息。这种方式通常用于数字签名，确保消息是由发送方生成的，且在传输过程中未被篡改。这里加密的目的是为了验证真实性，而非加密传输。

总而言之，两种方式的目的不同。公钥加密，私钥解密用于机密性传输，保护数据不被他人获取。私钥加密，公钥解密用于数字签名，验证消息的真实性和完整性。

步骤三：数字签名的使用

在此次实验中，尝试使用 RSA 进行数字签名。运行结果如下：

```
nakno@nakno-VirtualBox:~$ sudo python3 rsa3.py
加密后: b'HDid64V9WwStnOFMaQZ8K+na6r0eeIEYEAV5WVrgVURzWvtn8wJM7zzeQwjUxp9Is9mziLKp8cE6+ofjelMvV7
MHuTgbcPxS298XigvyUEbnQlmt30fkTfdPJJNxsTcWgbjIag74IyRiTaNEg2pZTMsZYdaUN+n80HgwEJM88kM='
解密的原文: b'hello, this is a plain text'
Hash: 0f7b9c1779a56cde0deec591da114e59a68db301 length: 160
result: True
```

思考题 3 ：数字签名代码注释每行的作用是什么？并举例使 result=False。

假设在签名前，将消息 n 的内容进行修改，使得签名后的 result 为 False。对代码进行修改，由于 received_message 不同于原始消息，重新计算的哈希值与解密后的哈希值不一致，导致 result=False，签名验证失败。

如将代码修改成如下：

```
# 签名和验证
n = b'This is not a test message'
h = SHA.new()
h.update(n)
print('Hash:', h.hexdigest(), 'length:', len(h.hexdigest()) * 4)

sign_txt = 'sign.txt'

with open('master-private.pem') as f:
    key = f.read()
    private_key = RSA.importKey(key)
    hash_obj = SHA.new(n)
    signer = Signature_pkcs1_v1_5.new(private_key)
    d = base64.b64encode(signer.sign(hash_obj))

with open(sign_txt, 'wb') as f:
    f.write(d)

with open('master-public.pem') as f:
    key = f.read()
    public_key = RSA.importKey(key)
    with open(sign_txt, 'r') as sign_file:
        sign = base64.b64decode(sign_file.read())
        h = SHA.new(n)
        verifier = Signature_pkcs1_v1_5.new(public_key)
        print('result:', verifier.verify(h, sign))
```

得到如下输出，可以看出返回的结果因为哈希值的验证失败而显示为 False。

```
nakno@nakno-VirtualBox:~$ sudo python3 rsa4.py
Hash: b2cad21e87dbd57f1a1928e4013b56fb553bb82a length: 160
result: False
```

以下是数字签名过程的代码注释：

#%%

1. 定义测试消息

n = b'This is a test message'

```
# 2. 创建一个 SHA 哈希对象
h = SHA.new()
# 3. 更新哈希对象，计算消息的哈希值
h.update(n)
# 4. 打印哈希值及其长度（长度以位为单位）
print('Hash:', h.hexdigest(), 'length:', len(h.hexdigest()) * 4)
# 5. 设置数字签名存储的文件名
sign_txt = 'sign.txt'
# 6. 读取私钥文件，用于数字签名
with open('master-private.pem') as f:
    key = f.read()
    private_key = RSA.importKey(key)  # 导入私钥
    hash_obj = SHA.new(n)  # 创建哈希对象并更新消息
    signer = Signature_pkcs1_v1_5.new(private_key)  # 使用私钥创建签名对象
    d = base64.b64encode(signer.sign(hash_obj))  # 签名并进行 Base64 编码
# 7. 将签名写入文件
f = open(sign_txt, 'wb')
f.write(d)
f.close()
# 8. 读取公钥文件，用于验证签名
with open('master-private.pem') as f:
    key = f.read()
    public_key = RSA.importKey(key)  # 导入公钥
    sign_file = open(sign_txt, 'r')  # 打开签名文件
    sign = base64.b64decode(sign_file.read())  # 读取并解码签名
    # 9. 创建哈希对象并更新消息内容
    h = SHA.new(n)
    # 10. 创建验证对象并进行验证
    verifier = Signature_pkcs1_v1_5.new(public_key)
    print('result:', verifier.verify(h, sign))  # 输出验证结果
```

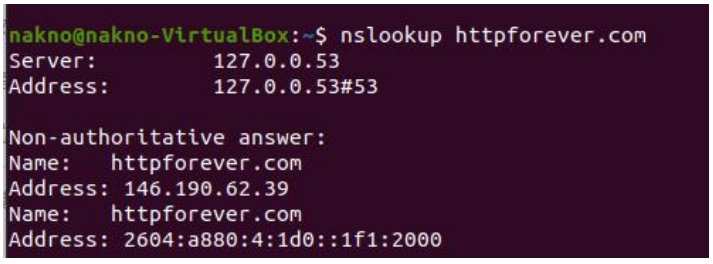
步骤四：抓包观察 ssl 协议通信握手过程

最常见的使用方式为 https，因此在本步骤中，使用 HTTPS 协议访问网站。

但是由于如今大部分网站都已经全面迁移到 HTTPS 协议以提高安全性，因此在使用 wireshark 进行抓包的时候，通常情况下访问网站都只能抓到加密后的 TLSv，即 TLS 加密数据包。为了完成实验，本步骤采用支持 HTTP 的测试用途网站进行操作。

本步骤目标网站为 http://httpforever.com，其为一个用于测试和分析 HTTP 流量的网站，具有简单页面，适合抓包和测试工具的特点。

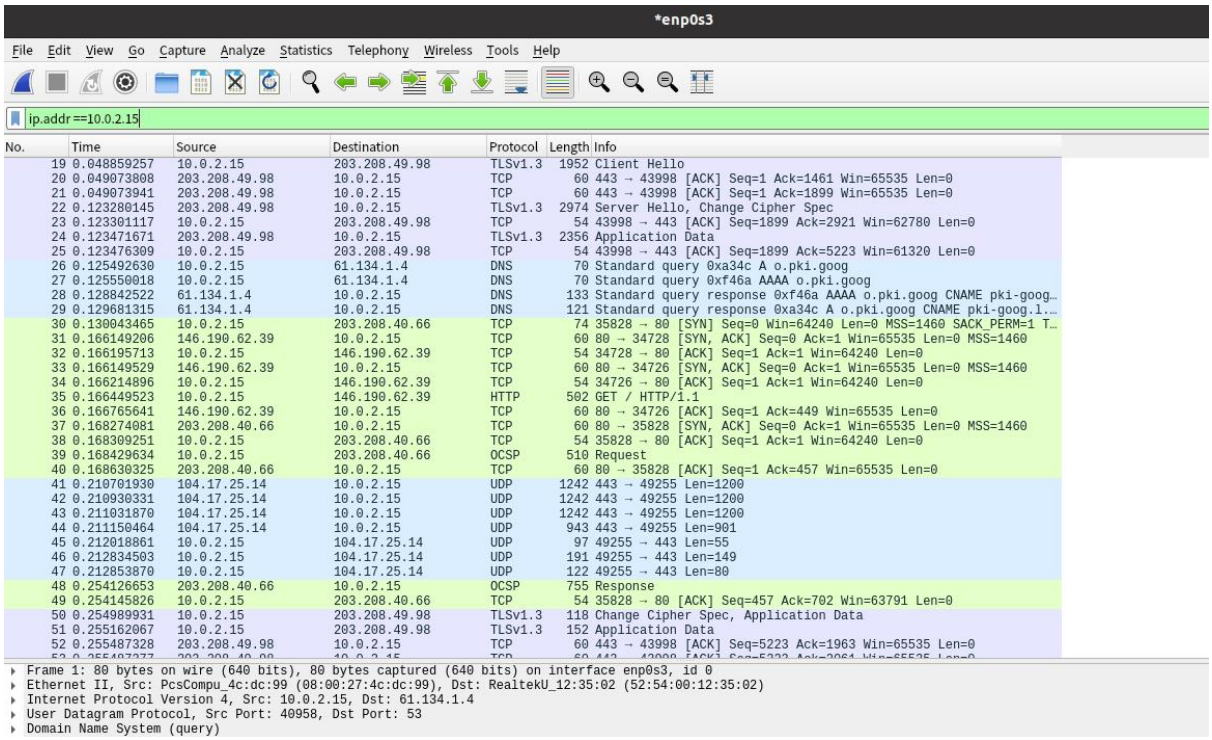
使用命令 nslookup httpforever.com，得到返回结果显示 IP 地址为 146.190.62.39。



随后查看本机的内网 IP 地址，发现本机 IP 为 10.0.2.15。

```
nakno@nakno-VirtualBox:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe4c:dc99  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:4c:dc:99  txqueuelen 1000  (Ethernet)
        RX packets 35104  bytes 52032015 (52.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9672  bytes 627609 (627.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

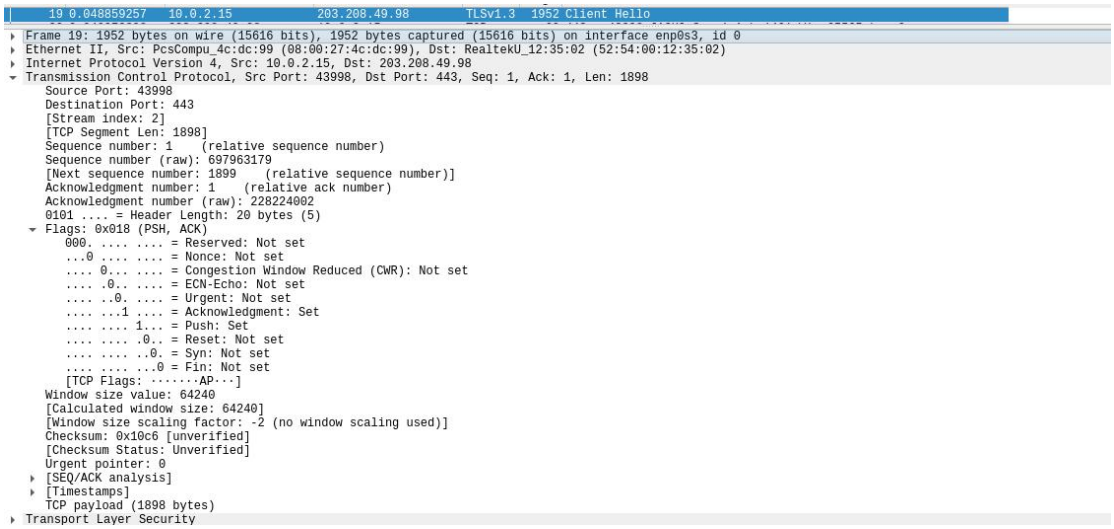
打开 Wireshark，开始抓包观察 SSL 协议通信的握手过程。



思考题 4：详细分析 SSL 协议通信握手过程。

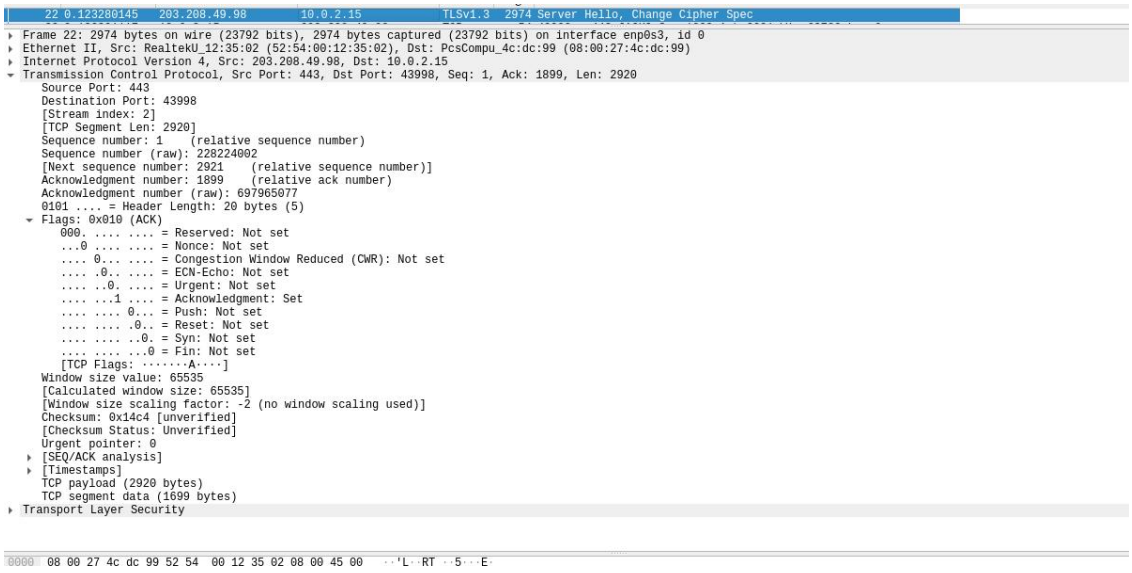
SSL，即 Secure Sockets Layer 是为网络通信提供安全性的一种协议。通过抓包数据分析 SSL/TLS 协议握手流程如下。

1. 客户端向服务器发起握手请求，发送一个 ClientHello 消息。



消息内容包括客户端支持的最高版本为 TLS 1.3。以及用于生成会话密钥的随机值的随机数、支持的加密套件列表和支持的压缩算法列表。目的是向服务器告知客户端的能力，以便协商合适的加密参数。

2. 服务器收到 ClientHello 后，回复一个 ServerHello 消息。



消息内容包括服务器选择的版本、服务器生成的随机值、选定的加密套件和选定的压缩算法。目的是确认通信双方使用的加密和通信参数。

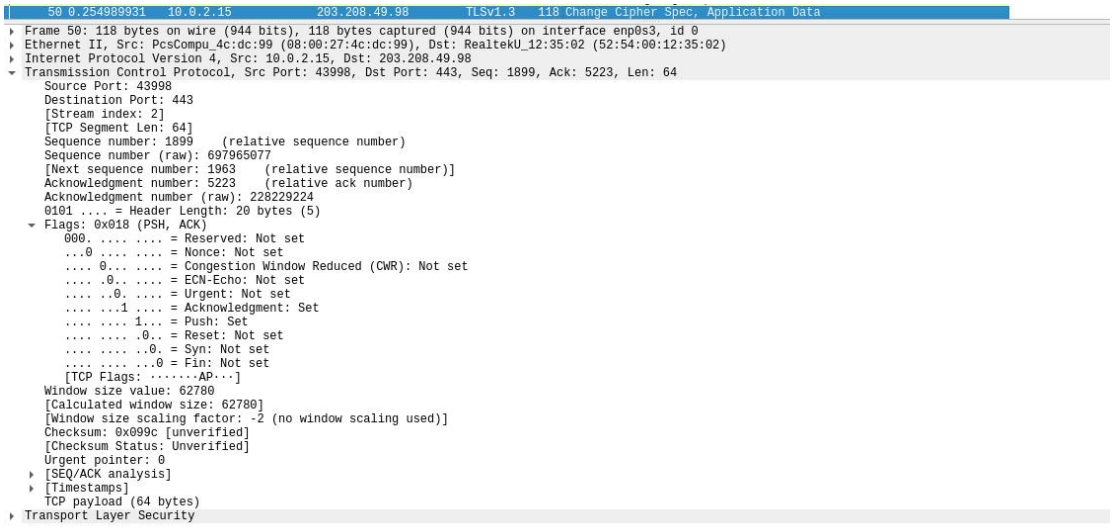
3. 服务器向客户端发送数字证书，用于证明服务器的身份。证书内容包括：服务器的公钥、服务器的域名、证书的颁发机构、证书的有效期及证书的数字签名。目的是向客户端证明服务器的身份，并提供公钥用于后续的密钥协商。

4. 密钥交换和生成会话密钥。

根据使用的密钥交换算法，步骤会有所不同：

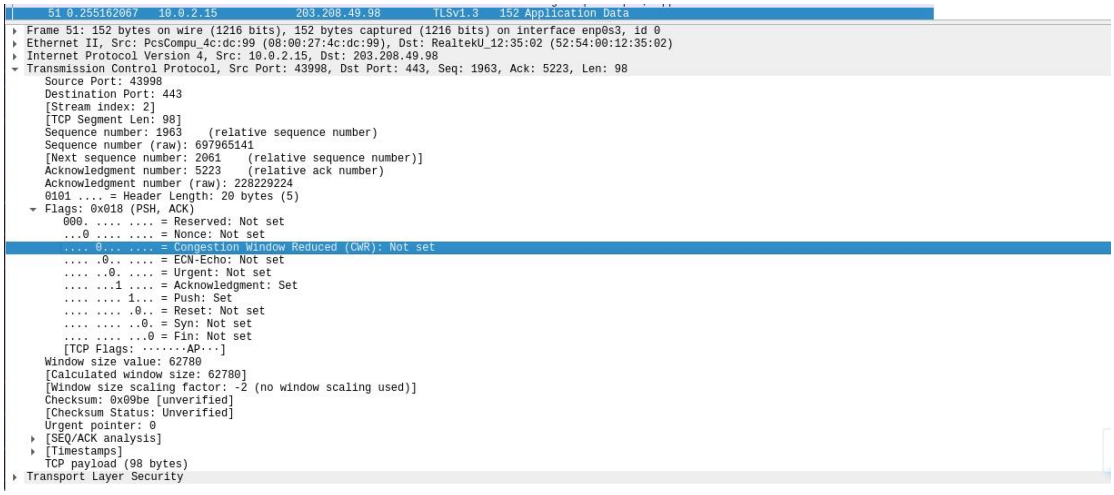
若为 RSA 密钥交换，则客户端生成一个预主密钥，并使用服务器的公钥加密。客户端将加密后的预主密钥发送给服务器。服务器使用自己的私钥解密，得到预主密钥。客户端和服务器基于预主密钥、Client Random 和 Server Random 共同生成会话密钥。

若为 DH/ECDH 密钥交换，则客户端和服务器各自生成一对公钥和私钥。双方交换公钥。使用对方的公钥和自己的私钥，计算共享的会话密钥。



5. 客户端根据生成的会话密钥，对消息"Finished"进行加密后发送给服务器。该消息

包括之前握手过程的摘要，用于验证握手过程是否被篡改。服务器也对消息 "Finished" 加密后发送给客户端。同样包含之前握手的摘要，用于验证客户端的握手过程。以确认服务器已经生成了正确的会话密钥，并验证握手的完整性。



6. 双方确认会话密钥一致后，进入数据传输阶段。所有通信数据将使用协商的加密算法和会话密钥加密。

因此，SSL/TLS 握手的目的是在客户端与服务器之间建立一个安全的加密通信通道。握手过程确保了双方的身份验证，同时进行会话密钥的安全协商，确保数据传输的保密性和完整性。

四、实验原理

1、AES 加密与 IV 变量的作用

高级加密标准（AES）是一种对称加密算法，使用固定长度的块对数据进行加密。IV（Initialization Vector，初始化向量）是随机生成的非机密值，其作用是增强加密的强度，确保相同的明文在不同加密过程中产生不同的密文，即使密钥相同。同时避免因重复模式导致的攻击风险，如已知明文攻击。增加加密过程的唯一性，确保加密的随机性和不可预测性。

2、RSA 加解密与公私钥的使用

非对称加密，如 RSA 使用一对密钥进行加密和解密。公钥加密，私钥解密用于保护数据的机密性，确保只有持有私钥的接收者能解密数据。私钥加密，公钥解密用于数字签名，验证数据的真实性和完整性，确保数据来源于持有私钥的发送方且未被篡改。

3、数字签名的原理

数字签名通过对消息的哈希值使用发送方的私钥加密生成签名，接收方通过公钥解密验证确保消息的来源真实性同时确保消息在传输过程中未被篡改，即哈希值匹配。如果消息被篡改，接收方重新计算的哈希值与签名解密后的哈希值将不匹配，签名验证失败。

4、SSL/TLS 协议通信握手过程

SSL/TLS 协议是一种用于在客户端和服务器之间提供加密通信和数据完整性的协议。

握手过程服务器通过数字证书证明身份，客户端可选验证身份。并进行会话密钥协商，客户端和服务器协商对称密钥，用于后续数据传输的加密。握手完成后，双方使用协商的会话密钥进行加密通信。

核心目标是确保数据的机密性、确保数据的完整性、确保通信双方的身份。

五、实验小结

通过安装加解密库 PyCrypto，运行 AES 加密程序，了解对称加密的基本过程。了解了初始化向量是增强 AES 加密强度的重要组件，通过在相同密钥下生成不同密文，提高了加密过程的随机性和安全性。

RSA 加解密方面，通过实验掌握公钥加密私钥解密（机密性）与私钥加密公钥解密（真实性）的用途及其原理。使用 RSA 进行签名验证，学习如何确保消息的来源真实性和传输完整性。实验中验证了当消息被篡改时，签名验证会失败。

实验通过 Wireshark 抓包观察 SSL/TLS 协议握手过程，分析握手中的核心步骤。验证握手的主要目的是在客户端和服务器之间建立安全加密通道，确保通信的保密性、完整性和身份认证。

六、遇到的问题和解决方法。

本实验为偏向验证性，因此较为简单，主要遇到的问题在于原理的理解和代码报错的处理。对于实验原理的不明白，通过指导书给出的链接和网络资料得到了解决。对于代码出现 AttributeError: module 'time' has no attribute 'clock'的报错，通过修改代码将 time.clock()替换为 time.perf_counter()得到解决。

七、实验心得

通过这次实验，我深入学习了对称加密和非对称加密算法的基本原理和应用，同时了解了加密通信的实现细节。通过分析 SSL 握手过程，我对 SSL 的基本概念和作用有了更全面的理解。特别是通过使用 Wireshark 抓包分析，熟悉了 SSL 建立过程，并且对网络安全和协议设计有了更深入的认识。

实验让我掌握了 AES 和 RSA 的加解密过程，并理解了 IV 在 AES 加密中的重要性，以及 RSA 数字签名在验证消息真实性和完整性中的应用。除此之外，我还学会了 SSL/TLS 协议的通信机制，进一步为网络安全通信的理解打下了基础。通过这次实验，我不仅提升了使用抓包工具分析协议的能力，也加深了对加密原理及其应用场景的理解。