

# 网络与信息安全课内实验 3： 对称、非对称加解密实验报告

班级：计算机 2101 班

姓名：田濡豪

学号：2203113234

## 1 实验平台及环境

---

本次实验使用个人计算机完成。

使用 WSL（Windows Subsystem for Linux）完成，所安装的 Ubuntu 系统版本为：

`Ubuntu 22.04.4 LTS (GNU/Linux 5.15.167.4-microsoft-standard-WSL2 x86_64)`

对于需要 Ubuntu 图形化界面或需要多台虚拟机同时运行的实验步骤，选择使用 Docker 进行容器化部署。Docker 镜像可以方便的进行修改、销毁、容器并发，性能高于不同虚拟机且不影响宿主机，为实验提供了高灵活性和容错空间。所选用的镜像版本为：

`kasmweb/desktop:1.16.1-rolling-weekly`

对于每个实验步骤的具体环境配置，将在对应章节说明。

## 2 实验步骤

---

1. 对称加密算法的字符串的加解密
2. 使用 RSA 的加解密
3. 数字签名
4. 抓包观察 ssl 协议通信握手过程实验过程

## 3 实验过程

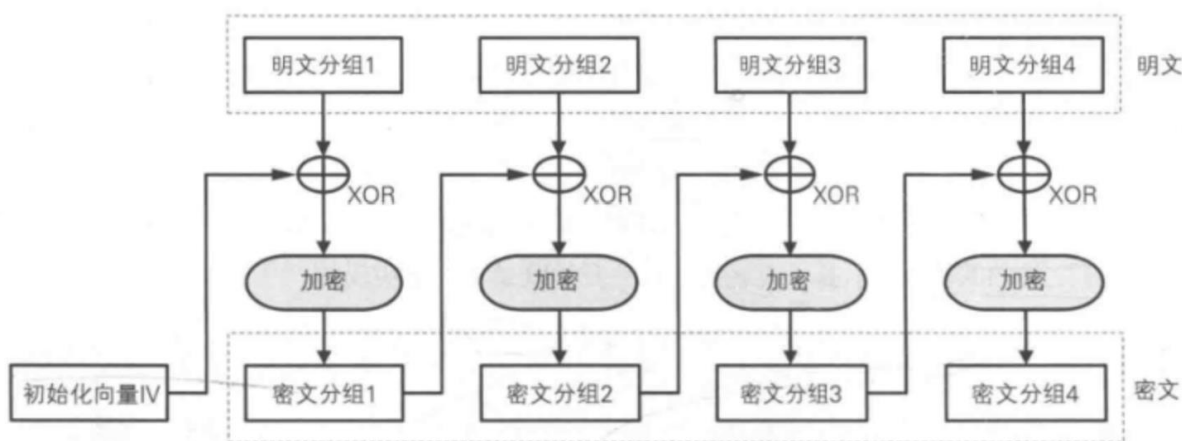
### 3.1 对称加密算法的字符串的加解密

# 加密函数

```
def encrypt(text, key):  
    key = key.encode('utf-8')  
    mode = AES.MODE_CBC  
    iv = b'qqqqqqqqqqqqqqqq'  # 16个q的二进制代码  
    text = add_to_16(text)  
    cryptos = AES.new(key, mode, iv)  
    cipher_text = cryptos.encrypt(text)  
    # 因为AES加密后的字符串不一定是ascii字符集的，输出保存可能存在问题，所以这里转为16  
    # 进制字符串  
    return b2a_hex(cipher_text)
```

观察实验初始代码，可见本实验使用 CBC 模式的 AES 加密方法。在 AES 进行加密时，明文会被切割成若干个相同长度的文字段，例如在 AES256 加密算法中，每个文字段的长度为 256 位。之后 AES 算法有多种可选方式处理这些文字段。在经典的 ECB 模式中，将使用密钥对每个文字段单独加密。该方法的问题是如果两个明文分组相同，则加密后的密文分组也相同，降低了算法的安全性。

CBC模式的加密



CBC 模式通过在加密一个明文分组前将其与上一个明文分组的密文做异或，保证了即时两个明文分组相同，加密的密文分组也一定是不同的。但是对于第一个明文分组，并不存在“前一个明文分组”，因此 CBC 引入了一个初始化向量 Initialization Vector，即实验初始代码中 iv 变量的含义。

显然，iv 的长度应该和明文分组的长度相同。实验中设置的 iv 由 16 个英文字母 q 的二进制代码组成，共 256 位，对应 AES256 算法。

```

(xjtu_ns) ruhaotian@Book3Ultra-Ruhao:~/xjtu_ns_exp/exp3$ python 实验三aes.py
Traceback (most recent call last):
  File "/home/ruhaotian/xjtu_ns_exp/exp3/实验三aes.py", line 45, in <module>
    e = encrypt(str1,'0123456789101112') # 加密
  File "/home/ruhaotian/xjtu_ns_exp/exp3/实验三aes.py", line 27, in encrypt
    cryptos = AES.new(key, mode, iv)
  File "/home/ruhaotian/miniconda3/envs/xjtu_ns/lib/python3.10/site-packages/Crypto/Cipher/AES.py", line 95, in new
    return AESCipher(key, *args, **kwargs)
  File "/home/ruhaotian/miniconda3/envs/xjtu_ns/lib/python3.10/site-packages/Crypto/Cipher/AES.py", line 59, in __init__
    blockalgo.BlockAlgo.__init__(self, _AES, key, *args, **kwargs)
  File "/home/ruhaotian/miniconda3/envs/xjtu_ns/lib/python3.10/site-packages/Crypto/Cipher/blockalgo.py", line 141, in __init__
    self._cipher = factory.new(key, *args, **kwargs)
SystemError: PY_SSIZE_T_CLEAN macro must be defined for '#' formats

```

第一次运行未能成功。经查阅是因为实验中使用的 pycrypto 包在我的虚拟环境 Python 版本下已经停止维护了。安装其对应的新版本 pycryptodome 后即可正常加解密。

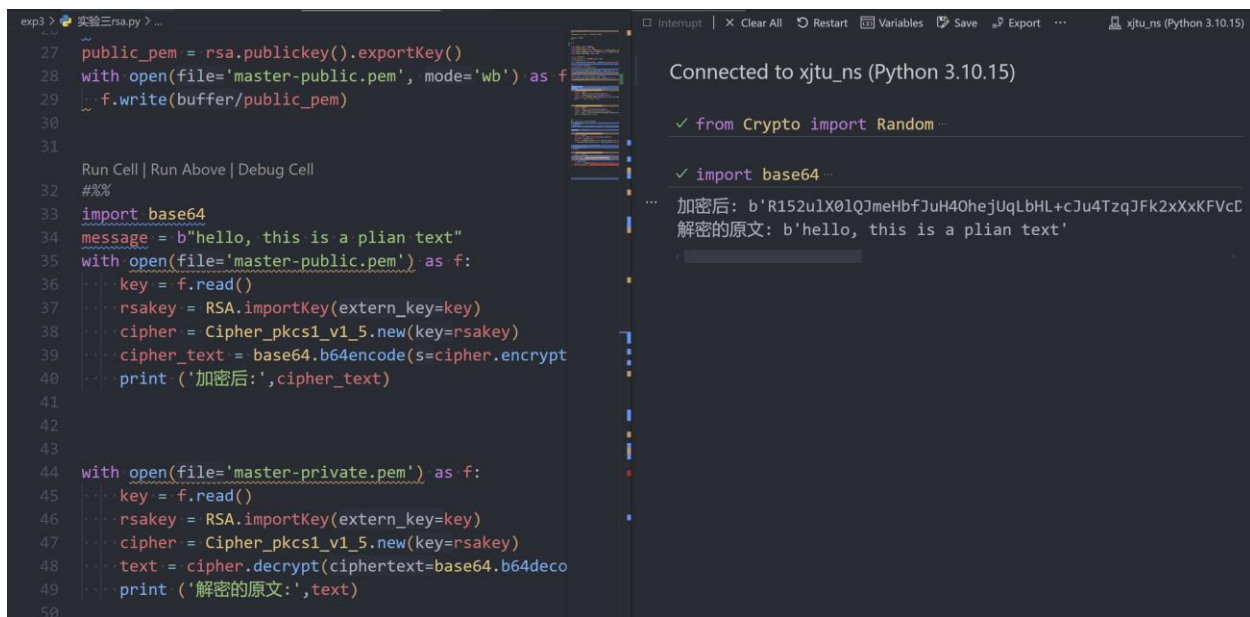
```

(xjtu_ns) ruhaotian@Book3Ultra-Ruhao:~/xjtu_ns_exp/exp3$ pip install pycryptodome
Collecting pycryptodome
  Downloading pycryptodome-3.21.0-cp36-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.4 kB)
  Downloading pycryptodome-3.21.0-cp36-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.3 MB)
    2.3/2.3 MB 80.7 kB/s eta 0:00:00
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.21.0
(xjtu_ns) ruhaotian@Book3Ultra-Ruhao:~/xjtu_ns_exp/exp3$ python 实验三aes.py
原文: XJTU Network Security 2024 Spring Lab3, Ruhao Tian
加密: b'11779b1b5b88ca6dc5186a6b726ff727d83763fcb80e2107932f5bd1d3162988b2df168e86414710dce4acbfb7b22de3d47070f36f7826ec3d224b14f3f0940'
解密: XJTU Network Security 2024 Spring Lab3, Ruhao Tian
(xjtu_ns) ruhaotian@Book3Ultra-Ruhao:~/xjtu_ns_exp/exp3$ 

```

## 3.2 使用 RSA 的加解密

运行实验代码，确认 RSA 公钥加密-私钥解密可以正常运行：



```

exp3 > 实验三rsa.py > ...
27 public_pem = rsa.publickey().exportKey()
28 with open(file='master-public.pem', mode='wb') as f:
29     f.write(buffer/public_pem)
30
31
32 Run Cell | Run Above | Debug Cell
33 #%%
34 import base64
35 message = b"hello, this is a plian text"
36 with open(file='master-public.pem') as f:
37     key = f.read()
38     rsakey = RSA.importKey(extern_key=key)
39     cipher = Cipher_pkcs1_v1_5.new(key=rsakey)
40     cipher_text = base64.b64encode(s=cipher.encrypt
41     print ('加密后:', cipher_text)
42
43
44 with open(file='master-private.pem') as f:
45     key = f.read()
46     rsakey = RSA.importKey(extern_key=key)
47     cipher = Cipher_pkcs1_v1_5.new(key=rsakey)
48     text = cipher.decrypt(ciphertext=base64.b64deco
49     print ('解密的原文:', text)
50

```

Connected to xjtu\_ns (Python 3.10.15)

```

✓ from Crypto import Random...
✓ import base64...
...
加密后: b'R152u1X0lQJmeHbfJuH40hejUqLbHL+cJu4TzqJFk2xXxKFVC
解密的原文: b'hello, this is a plian text'

```

为了研究使用公钥加密私钥解密和使用私钥加密公钥解密计算过程是否一致，查看 encrypt 函数的源码：

```
def encrypt(self, message):
    """Produce the PKCS#1 v1.5 encryption of a message.

    This function is named ``RSAES-PKCS1-V1_5-ENCRYPT``, and it is specified in
    section 7.2.1 of RFC8017
    <https://tools.ietf.org/html/rfc8017#page-28>`_.
```

可以看到该函数依照 RFC8017 协议规范指定。查阅 RFC8017 公开文档的核心加密算法部分：

#### 5.1.1. RSAEP

RSAEP ((n, e), m)

Input:

(n, e) RSA public key

m message representative, an integer between 0 and n - 1

Output: c ciphertext representative, an integer between 0 and n - 1

Error: "message representative out of range"

Assumption: RSA public key (n, e) is valid

Steps:

1. If the message representative m is not between 0 and n - 1, output "message representative out of range" and stop.
2. Let  $c = m^e \bmod n$ .
3. Output c.

可以看到除去错误检查外，核心算法是  $c = m^e \bmod n$  其中 n、e 为公钥，m 为待加密的信息，c 为输出。查看私钥解密算法：

Steps:

1. If the ciphertext representative  $c$  is not between 0 and  $n - 1$ , output "ciphertext representative out of range" and stop.
2. The message representative  $m$  is computed as follows.
  - a. If the first form  $(n, d)$  of  $K$  is used, let  $m = c^d \bmod n$ .
  - b. If the second form  $(p, q, dP, dQ, qInv)$  and  $(r_i, d_i, t_i)$  of  $K$  is used, proceed as follows:
    - i. Let  $m_1 = c^{dP} \bmod p$  and  $m_2 = c^{dQ} \bmod q$ .
    - ii. If  $u > 2$ , let  $m_i = c^{(d_i)} \bmod r_i, i = 3, \dots, u$ .
    - iii. Let  $h = (m_1 - m_2) * qInv \bmod p$ .
    - iv. Let  $m = m_2 + q * h$ .
    - v. If  $u > 2$ , let  $R = r_1$  and for  $i = 3$  to  $u$  do
      1. Let  $R = R * r_{i-1}$ .
      2. Let  $h = (m_i - m) * t_i \bmod r_i$ .
      3. Let  $m = m + R * h$ .
3. Output  $m$ .

可见，对于 $(n, d)$ 格式的私钥，公钥加密和私钥解密步骤在数学上是等价的。因此 $(n, e)$ 和 $(n, d)$ 格式的公私钥对理论上可以使用相同的加密/解密算法。但是，私钥还有另外一种格式 $(p, q, dP, dQ, qInv)$ ，对于这种解密算法，显然不可能将 $(n, e)$ 格式的公钥传入充当私钥。因此，【公钥加密，私钥解密】和【私钥加密，公钥解密】的算法是否相同与密钥的格式有关，如果公钥和私钥的格式不同，那么该两种算法也不相同。

### 3.3 数字签名

运行实验代码，实现签名验证：

The screenshot shows a VS Code editor with a Python script for RSA digital signatures. The script is divided into two main parts: signing and verification. The signing part takes a private key, a message, and a public key as input, generates a SHA1 hash, signs it with the private key, and saves the signature to a file. The verification part reads the signature file, uses the public key to verify the signature, and prints the result. The output window on the right shows the execution results, including the hash, the signed message, and the verification result.

```
56 print('Hash:',h.hexdigest(),'length:',len(obj/h.hexdigest()))
57
58 sign_txt = 'sign.txt' # 签名文件名
59
60 with open(file='master-private.pem') as f: # 打开私钥文件
61     key = f.read() # 读取私钥
62     private_key = RSA.importKey(key) # 将私钥转换为RSA 密钥对象
63     hash_obj = SHA.new(n) # 从签名字符串生成信息摘要
64     signer = Signature_pkcs1_v1_5.new(private_key) # 使用PKCS1 填充算法和私钥创建数字签名对象
65     d = base64.b64encode(signer.sign(hash_obj)) # 对信息摘要进行数字签名并导出为base64 编码
66
67 f = open(file=sign_txt,mode='wb') # 保存导出的base64 签名到文件
68 f.write(d)
69 f.close()
70
71 with open(file='master-public.pem') as f: # 打开公钥文件
72     key = f.read()
73     public_key = RSA.importKey(key) # 将公钥转换为RSA 密钥对象
74     sign_file = open(file=sign_txt,mode='r') # 读取签名文件
75     sign = base64.b64decode(sign_file.read()) # 将读取的base64 签名编码转换为二进制字符串
76     h = SHA.new(n) # 生成内容的SHA1 算法信息摘要
77     verifier = Signature_pkcs1_v1_5.new(public_key) # 使用公钥创建数字签名对象
78     print('result:', verifier.verify(msg_hash=h.hexdigest(),sig=sign))
79
80
81
82
```

Connected to xjtu\_ns (Python 3.10.15)

```
✓ # -*- coding: utf-8 -*-
✓ import base64
...
加密后: b'd9BWQIIwI10hbf0Egd0AT19UP7tRD3T4Ujm8uS3JyhRVVruLQL'
解密的原文: b'hello, this is a plian text'
...
✓ n = b'This is a test message from Ruhao Tian' # 将测试消息转换为二进制表示
...
Hash: 422a3a5c37bc7855526705863e30d243cd038178 length: 160
result: True
```

查阅 RSA 数字签名原理并对实验初始代码进行注解：

```
n = b'This is a test message from Ruhao Tian' # 将要签名的内容字符串转换为二进制表示
h = SHA.new() # 建立新的SHA1 哈希对象
h.update(n) # 使用SHA1 加密算法对要签名的内容字符串生成信息摘要
print('Hash:',h.hexdigest(),'length:',len(h.hexdigest()*4)) # 使用hexdigest()方法
获取摘要的16 进制表示并输出
sign_txt = 'sign.txt' # 签名文件名
with open('master-private.pem') as f: # 打开私钥文件
    key = f.read() # 读取私钥
    private_key = RSA.importKey(key) # 将私钥转换为RSA 密钥对象
    hash_obj = SHA.new(n) # 从签名字符串生成信息摘要
    signer = Signature_pkcs1_v1_5.new(private_key) # 使用PKCS1 填充算法和私钥创建数字签名对象
    d = base64.b64encode(signer.sign(hash_obj)) # 对信息摘要进行数字签名并导出为base64 编码
f = open(sign_txt,'wb') # 保存导出的base64 签名到文件
f.write(d)
f.close()
with open('master-public.pem') as f: # 打开公钥文件
    key = f.read()
    public_key = RSA.importKey(key) # 将公钥转换为RSA 密钥对象
    sign_file = open(sign_txt,'r') # 读取签名文件
    sign = base64.b64decode(sign_file.read()) # 将读取的base64 签名编码转换为二进制字符串
    h = SHA.new(n) # 生成内容的SHA1 算法信息摘要
    verifier = Signature_pkcs1_v1_5.new(public_key) # 使用公钥创建数字签名对象
```

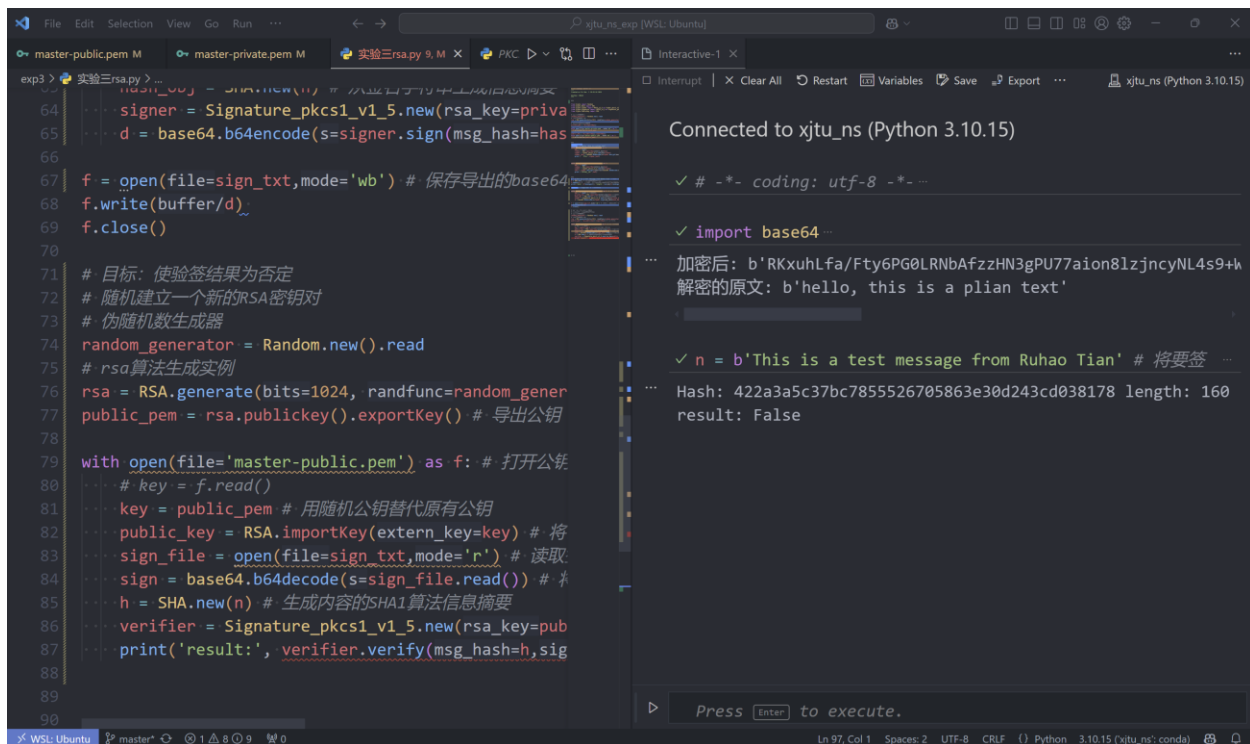
```
print('result:', verifier.verify(h,sign)) # 验证对内容的签名是否和公钥签名结果一致
```

根据签名过程，如果公钥-私钥无法组成 RSA 密钥对，则签名验证结果一定为 False，因此，添加一段代码随机生成另外一个公钥，并用它替代签名时所用私钥对应的公钥。

```
62 private_key = RSA.importKey(extern_key=key) # 将私钥转换为RSA密钥对象
63 ... hash_obj = SHA.new(n) # 从签名字符串生成信息摘要
64 ... signer = Signature_pkcs1_v1_5.new(rsa_key=private_key) # 使用PKCS1填充算法和私钥创建数字签名对象
65 ... d = base64.b64encode(s=signer.sign(msg_hash=hash_obj)) # 对信息摘要进行数字签名并导出为base64编码
66
67 f = open(file=sign_txt,mode='wb') # 保存导出的base64签名到文件
68 f.write(buffer/d)
69 f.close()
70
71 # 目标: 使验签结果为否定
72 # 随机建立一个新的RSA密钥对
73 # 伪随机数生成器
74 random_generator = Random.new().read
75 # rsa算法生成实例
76 rsa = RSA.generate(bits=1024, randfunc=random_generator)
77 public_pem = rsa.publickey().exportKey() # 导出公钥
78
79 with open(file='master-public.pem') as f: # 打开公钥文件
80 ... # key = f.read()
81 ... key = public_pem # 用随机公钥替代原有公钥
82 ... public_key = RSA.importKey(extern_key=key) # 将公钥转换为RSA密钥对象
83 ... sign_file = open(file=sign_txt,mode='r') # 读取签名文件
84 ... sign = base64.b64decode(s=sign_file.read()) # 将读取的base64签名编码转换为二进制位串
85 ... h = SHA.new(n) # 生成内容的SHA1算法信息摘要
86 ... verifier = Signature_pkcs1_v1_5.new(rsa_key=public_key) # 使用公钥创建数字签名对象
87 ... print('result:', verifier.verify(msg_hash=h,signature=sign)) # 验证对内容的签名是否和公钥签名结果一致
88
```

运行签名和验签过程，可见验签结果已经变为 False。





```
exp3> 实验三rsa.py ...
64 ... signer = Signature_pkcs1_v1_5.new(rsa_key=priva
65 ... d = base64.b64encode(s=signer.sign(msg_hash=has
66
67 f = open(file=sign_txt,mode='wb') # 保存导出的base64
68 f.write(buffer/d)
69 f.close()
70
71 # 目标: 使验签结果为否定
72 # 随机建立一个新的RSA密钥对
73 # 伪随机数生成器
74 random_generator = Random.new().read
75 # rsa算法生成实例
76 rsa = RSA.generate(bits=1024, randfunc=random_gener
77 public_pem = rsa.publickey().exportKey() # 导出公钥
78
79 with open(file='master-public.pem') as f: # 打开公钥
80 ... # key = f.read()
81 ... key = public_pem # 用随机公钥替代原有公钥
82 ... public_key = RSA.importKey(extern_key=key) # 将
83 ... sign_file = open(file=sign_txt,mode='r') # 读取
84 ... sign = base64.b64decode(s=sign_file.read()) # 解
85 ... h = SHA.new(n) # 生成内容的SHA1算法信息摘要
86 ... verifier = Signature_pkcs1_v1_5.new(rsa_key=pub
87 ... print('result:', verifier.verify(msg_hash=h,sig
88
89
90
```

Connected to xjtu\_ns (Python 3.10.15)

```
✓ # -*- coding: utf-8 -*-
✓ import base64 ...
加密后: b'RKxuhLfFa/Fty6PG0LRNbAfzzHN3gPU77aion81zjncyNL4s9+k
解密的原文: b'hello, this is a plian text'
✓ n = b'This is a test message from Ruhao Tian' # 将要签 ...
Hash: 422a3a5c37bc7855526705863e30d243cd038178 length: 160
result: False
```

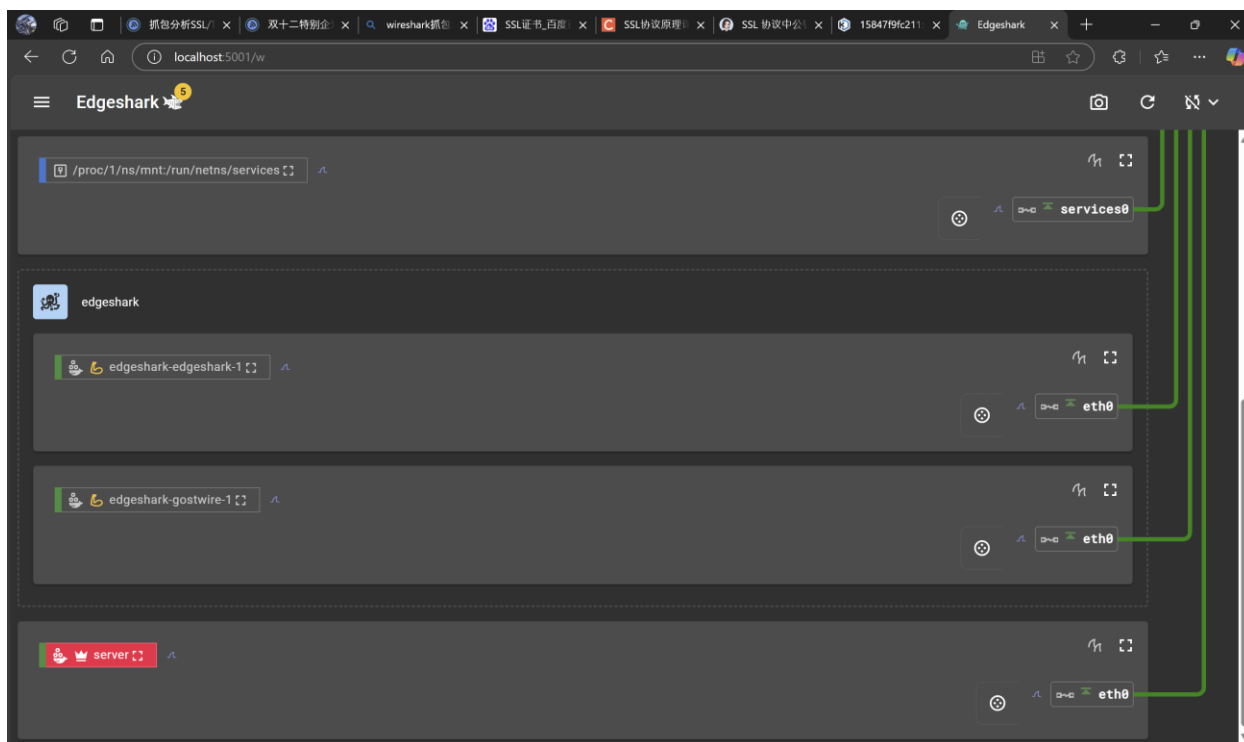
Press **Enter** to execute.

### 3.4 抓包观察 SSL 协议通信握手过程实验过程

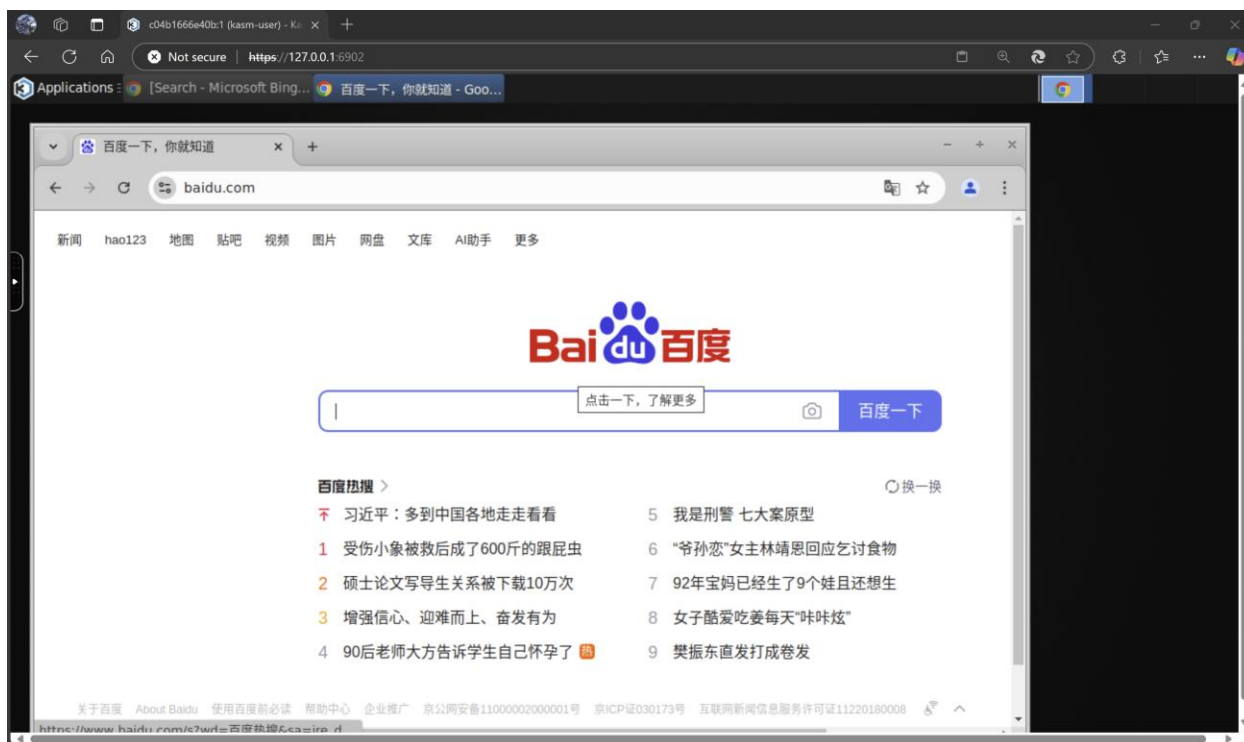
本次实验中选择微软 Baidu 网址作为目标域名进行抓包。由于主机中的网络连接比较复杂，而且为了查阅资料开启了 VPN，因此实验中采用前几次实验中的经验：使用 Docker 对一个容器进行网络隔离，在容器中运行浏览器并访问 Baidu，后使用 Edgeshark 探针配合 Wireshark 软件对容器进行单独抓包，以此针对性捕获流量。

运行一个名叫 Server 的 Ubuntu 容器和 Edgeshark 探针，可见探针已经连接到容器上。





访问 Baidu 并开始抓包：



使用 dig 查找 Baidu 的 IP 地址，得到 IP 地址为 110.242.68.3，在 Wireshark 中使用相应命令过滤对应的包：

ip.dst == 110.242.68.3 or ip.src == 110.242.68.3

Wireshark packet capture showing a TLS handshake between a client and a server. The client sends a Client Hello message (packet 3737) to the server (110.242.68.3). The server responds with a Server Hello message (packet 3738). The handshake continues with Key Exchange and Certificate exchange.

No.	Time	Source	Destination	Protocol	Length	Info
3728	9.848874	10.172.10.127	110.242.68.3	TCP	54	55720 → 443 [ACK] Seq=1 Ack=1 Win=132096 Len=0
3729	9.848993	10.172.10.127	110.242.68.3	TCP	54	55719 → 443 [ACK] Seq=1 Ack=1 Win=132096 Len=0
3736	9.851239	10.172.10.127	110.242.68.3	TCP	1506	55719 → 443 [ACK] Seq=1 Ack=1 Win=132096 Len=1452 [TCP PDU reassembled in 3737]
3737	9.851239	10.172.10.127	110.242.68.3	TLSv1.2	325	Client Hello (SNI=www.baidu.com)
3738	9.851488	10.172.10.127	110.242.68.3	TCP	1506	55720 → 443 [ACK] Seq=1 Ack=1 Win=132096 Len=1452 [TCP PDU reassembled in 3739]
3739	9.851488	10.172.10.127	110.242.68.3	TLSv1.2	325	Client Hello (SNI=www.baidu.com)
3744	9.909406	110.242.68.3	10.172.10.127	TCP	66	[TCP Dup ACK 3711#1] 443 → 55720 [ACK] Seq=1 Ack=1 Win=79488 Len=0 SLE=1453 SRE=1724
3745	9.909406	110.242.68.3	10.172.10.127	TCP	60	443 → 55720 [ACK] Seq=1 Ack=1724 Win=82432 Len=0
3746	9.909406	110.242.68.3	10.172.10.127	TCP	66	[TCP Dup ACK 3711#1] 443 → 55719 [ACK] Seq=1 Ack=1 Win=30208 Len=0 SLE=1453 SRE=1724
3747	9.909406	110.242.68.3	10.172.10.127	TCP	60	443 → 55719 [ACK] Seq=1 Ack=1724 Win=33024 Len=0
3748	9.909406	110.242.68.3	10.172.10.127	TLSv1.2	1506	Server Hello
3749	9.909406	110.242.68.3	10.172.10.127	TCP	1506	443 → 55720 [ACK] Seq=1453 Ack=1724 Win=82432 Len=1452 [TCP PDU reassembled in 3751]
3750	9.909406	110.242.68.3	10.172.10.127	TCP	1506	443 → 55720 [ACK] Seq=2905 Ack=1724 Win=82432 Len=1452 [TCP PDU reassembled in 3751]
3751	9.909406	110.242.68.3	10.172.10.127	TLSv1.2	901	Certificate, Server Key Exchange, Server Hello Done
3752	9.909406	110.242.68.3	10.172.10.127	TLSv1.2	1506	Server Hello
3753	9.909406	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=1453 Ack=1724 Win=33024 Len=1452 [TCP PDU reassembled in 3755]
3754	9.909406	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=2905 Ack=1724 Win=33024 Len=1452 [TCP PDU reassembled in 3755]
3755	9.909406	110.242.68.3	10.172.10.127	TLSv1.2	901	Certificate, Server Key Exchange, Server Hello Done
3756	9.909406	110.242.68.3	10.172.10.127	TCP	901	[TCP Retransmission] 443 → 55720 [PSH, ACK] Seq=4357 Ack=1724 Win=82432 Len=847
3757	9.909406	110.242.68.3	10.172.10.127	TCP	901	[TCP Retransmission] 443 → 55719 [PSH, ACK] Seq=4357 Ack=1724 Win=33024 Len=847
3759	9.909559	10.172.10.127	110.242.68.3	TCP	66	55720 → 443 [ACK] Seq=1724 Ack=5204 Win=132096 Len=0 SLE=4357 SRE=5204
3760	9.909617	10.172.10.127	110.242.68.3	TCP	66	55719 → 443 [ACK] Seq=1724 Ack=5204 Win=132096 Len=0 SLE=4357 SRE=5204
3762	9.912113	10.172.10.127	110.242.68.3	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
3763	9.912554	10.172.10.127	110.242.68.3	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
3764	9.912742	10.172.10.127	110.242.68.3	TCP	1506	55719 → 443 [ACK] Seq=1850 Ack=5204 Win=132096 Len=1452 [TCP PDU reassembled in 3765]
3765	9.912742	10.172.10.127	110.242.68.3	TLSv1.2	505	Application Data
3802	10.003968	110.242.68.3	10.172.10.127	TCP	60	443 → 55720 [ACK] Seq=5204 Ack=1850 Win=82432 Len=0
3803	10.003968	110.242.68.3	10.172.10.127	TLSv1.2	280	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message

Frame 3737: 325 bytes on wire (2600 bits), 325 bytes captured (2600 bits) on interface \Device\NPF\_{63021234-E510-455D-9C46-34B6A0E76886}, id 0

Ethernet II, Src: Intel\_Ib:33:0c (e4:0d:36:1b:33:0c), Dst: H3CTechnolog\_b4:e0:01 (38:97:d6:b4:e0:01)

Internet Protocol Version 4, Src: 10.172.10.127, Dst: 110.242.68.3

Transmission Control Protocol, Src Port: 55719, Dst Port: 443, Seq: 1453, Ack: 1, Len: 271

[2 Reassembled TCP Segments (1723 bytes): #3736(1452), #3737(271)]

Transport Layer Security

Ready to load or capture

Packets: 6641 · Displayed: 342 (5.1%) · Dropped: 0 (0.0%)

Profile: Default

可见，首先客户端通过 TLS 协议向服务器发送了 Client Hello。其中包含了用于生成对称加密密钥的随机数 Random，客户端支持的加密套件 Cipher Suites 和其他基本信息。

Wireshark packet capture showing the details of the Client Hello message (packet 3737). The message contains a random number, supported cipher suites, and other TLS parameters.

Frame 3737: 325 bytes on wire (2600 bits), 325 bytes captured (2600 bits) on interface \Device\NPF\_{63021234-E510-455D-9C46-34B6A0E76886}, id 0

Ethernet II, Src: Intel\_Ib:33:0c (e4:0d:36:1b:33:0c), Dst: H3CTechnolog\_b4:e0:01 (38:97:d6:b4:e0:01)

Internet Protocol Version 4, Src: 10.172.10.127, Dst: 110.242.68.3

Transmission Control Protocol, Src Port: 55720, Dst Port: 443, Seq: 1453, Ack: 1, Len: 271

[2 Reassembled TCP Segments (1723 bytes): #3736(1452), #3737(271)]

Transport Layer Security

TLSv1.2 Record Layer: Handshake Protocol: Client Hello

Content Type: Handshake (22)

Version: TLS 1.0 (0x0301)

Length: 1718

Handshake Protocol: Client Hello

Length: 1714

Version: TLS 1.2 (0x0303)

Random: 9e743fbb5912273a1f0e12176c5d2dc0e7e783340df4c579e4a54e4fe5d92fb4

Session ID Length: 32

Session ID: d0308195322a220429a1b5e2b28d16c8d42330352e67187d5e49d5a52d77ed7

Cipher Suites Length: 32

Cipher Suites (16 suites)

Compression Methods Length: 1

Compression Methods (1 method)

Extensions Length: 1609

Extension: Reserved (GREASE) (len=0)

Extension: session\_ticket (len=0)

Extension: signed\_certificate\_timestamp (len=0)

Extension: key\_share (len=263) Unknown (4588), x25519

Extension: ec\_point\_formats (len=2)

Extension: server\_name (len=18) name=www.baidu.com

Extension: encrypted\_client\_hello (len=186)

Extension: psk\_key\_exchange\_modes (len=2)

Extension: signature\_algorithms (len=18)

Extension: status\_request (len=5)

Extension: supported\_groups (len=12)

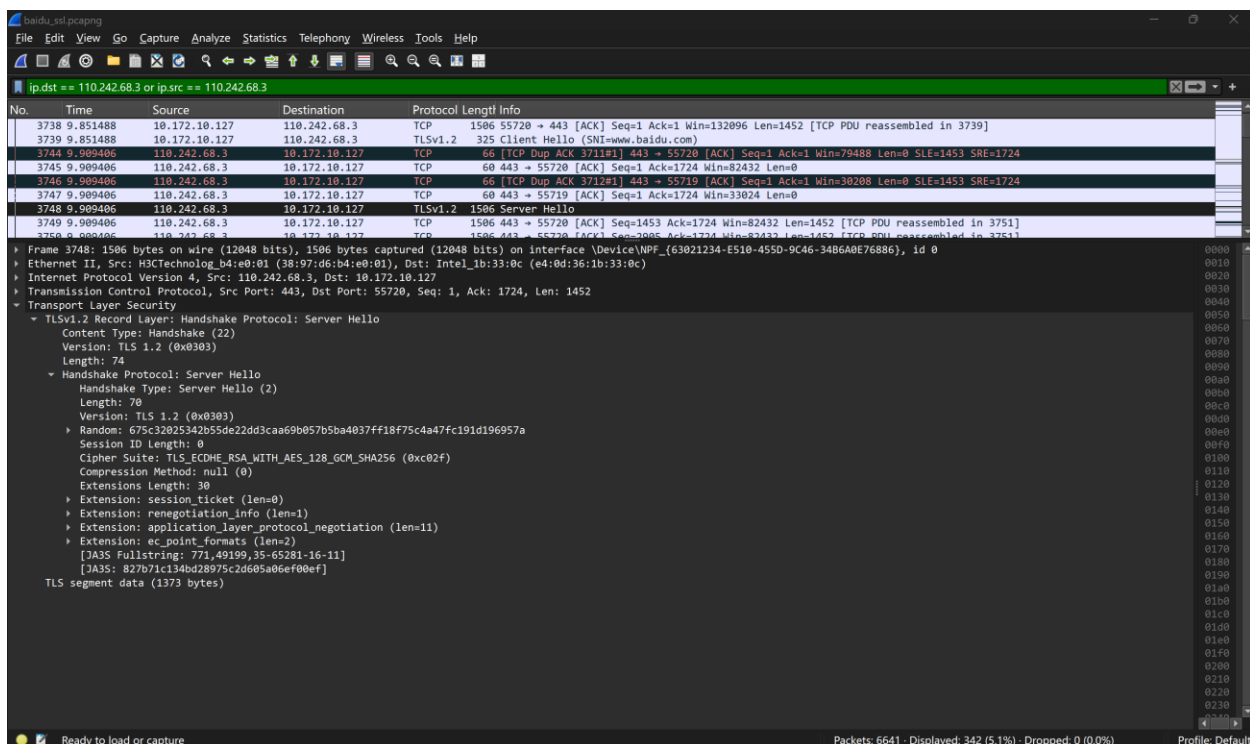
Extension: application\_settings (len=5)

Transmission Control Protocol (tcp), 20 bytes

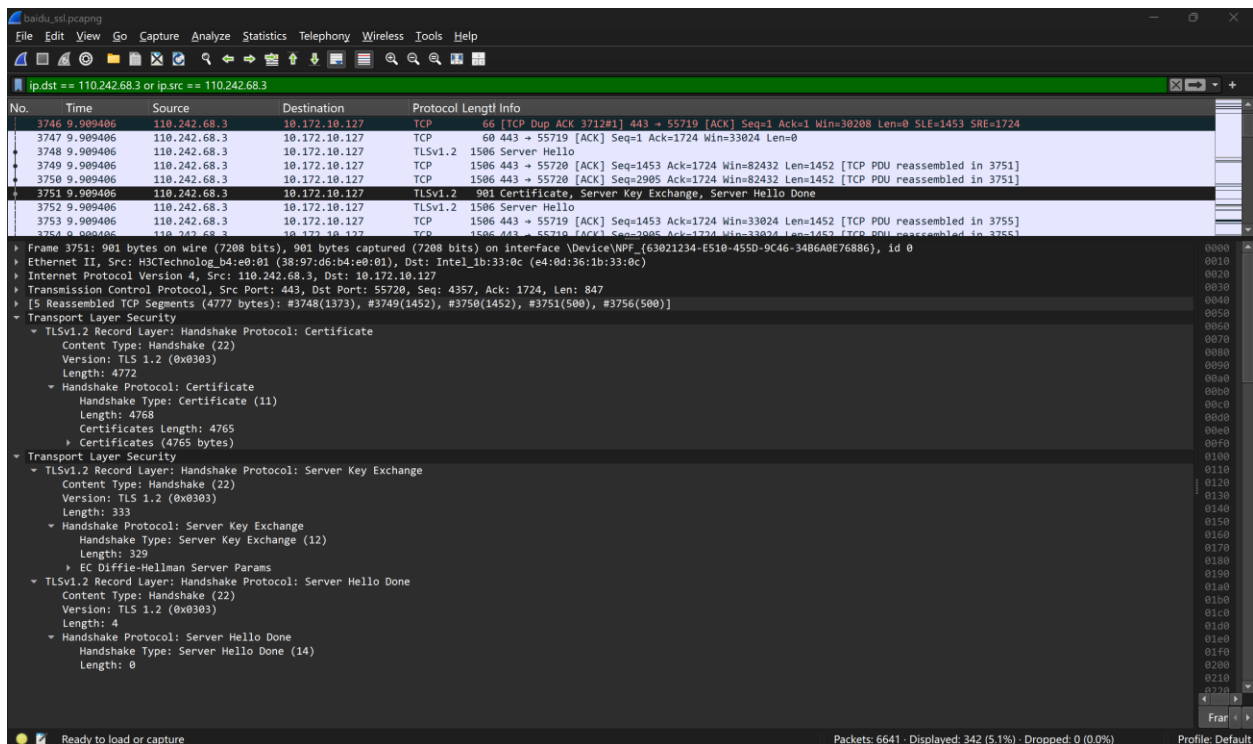
Packets: 6641 · Displayed: 342 (5.1%) · Dropped: 0 (0.0%)

Profile: Default

服务端同样也发送了 Server Hello，传送了 SSL 基本信息。



接下来，Server 和 Client 将会使用数字证书 CA 的形式交换公钥，这是为了进行非对称加密，防止中间人攻击等侵入行为。在本实验中，Server 先后发送了数字整数、公钥和 Server Hello Done 报文，表明 Server 侧的证书和公钥发送结束。在 SSL 协议中，Server 还可能发送 Certificate Request 报文，也就是指名 Client 端需要提交自己的证书，但是在本实验中 Server 并没有这样做。



在 Server 完成了向 Client 的证书和公钥提供后，由 Client 向 Server 提供自己的公钥和证书（可选）。在本实验中，由于 Server 并未要求 Client 提供证书，可见 Client 并没有向 Server 发送证书，但仍进行了公钥交换。同时，Client 发送了一个 Change Cipher Spec 报文和一个 Encrypted handshake message 报文。前者报文表示 Client 已经切换到了先前约定的加密套件中，也就是完成了加密通信的准备。后者则是在准备完成后发送了第一个被加密的报文。该报文的特殊性在于双方要对其进行校验，验证其没有被篡改以最终确认双方的加密措施的可靠性。

baidu.sslpcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ip.dst == 110.242.68.3 or ip.src == 110.242.68.3

No.	Time	Source	Destination	Protocol	Length	Info
3756	9.909486	110.242.68.3	10.172.10.127	TCP	981	[TCP Retransmission] 643 → 55720 [PSH, ACK] Seq=4357 Ack=1724 Win=82432 Len=847
3757	9.909405	110.242.68.3	10.172.10.127	TCP	981	[TCP Retransmission] 443 → 55719 [PSH, ACK] Seq=4357 Ack=1724 Win=33824 Len=847
3759	9.909559	10.172.10.127	110.242.68.3	TCP	66	55720 → 443 [ACK] Seq=1724 Ack=5204 Win=132096 Len=0 SLE=4357 SRE=5204
3760	9.909617	10.172.10.127	110.242.68.3	TCP	66	55719 → 443 [ACK] Seq=1724 Ack=5204 Win=132096 Len=0 SLE=4357 SRE=5204
3762	9.912113	10.172.10.127	110.242.68.3	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
3763	9.912554	10.172.10.127	110.242.68.3	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
3764	9.912742	10.172.10.127	110.242.68.3	TCP	1506	55719 → 443 [ACK] Seq=1850 Ack=5204 Win=132096 Len=1452 [TCP PDU reassembled in 3765]
3765	9.912742	10.172.10.127	110.242.68.3	TLSv1.2	505	Application Data

Frame 3762: 180 bytes on wire (1440 bits), 180 bytes captured (1440 bits) on interface \Device\NPF\_{63021234-E510-455D-9C46-34B6A0E76886}, id 0

Ethernet II, Src: Intel\_1b:33:0c (e4:8d:36:1b:33:0c), Dst: H3CTechnolog\_b4:e0:01 (38:97:d6:b4:e0:01)

Internet Protocol Version 4, Src: 10.172.10.127, Dst: 110.242.68.3

Transmission Control Protocol, Src Port: 55719, Dst Port: 443, Seq: 1724, Ack: 5204, Len: 126

Transport Layer Security

- TLsv1.2 Record Layer: Handshake Protocol: Client Key Exchange
  - Content Type: Handshake (22)
  - Version: TLS 1.2 (0x0303)
  - Length: 70
- Handshake Protocol: Client Key Exchange
  - Handshake Type: Client Key Exchange (16)
  - Length: 66
  - EC Diffie-Hellman Client Params
- TLsv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
  - Content Type: Change Cipher Spec (20)
  - Version: TLS 1.2 (0x0303)
  - Length: 1
  - Change Cipher Spec Message
- TLsv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
  - Content Type: Handshake (22)
  - Version: TLS 1.2 (0x0303)
  - Length: 40
  - Handshake Protocol: Encrypted Handshake Message

Record Layer (tls.record), 6 bytes

Packets: 6641 · Displayed: 342 (5.1%) · Dropped: 0 (0.0%)

Profile: Default

在 Client 声明准备完成后，Server 也向 Client 发送了 Change Cipher Spec 和 Encrypted Handshake Message 报文，表明自己也已经做好准备并提供了一个测试报文供对方测试。与 Client 的确认报文不同，Server 端还发送了一个 New Session Ticket，这代表客户端和服务端之前曾经建立过连接（因为我在抓包前测试时曾经访问过 Baidu 网页进行网络测试）。Server 将先前的 Session ID 以加密的方式附加在该报文中发送给 Client，因此 Client 可以重用之前的 Session 信息，加速握手过程。



baudu,ssl.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ip.dst == 110.242.68.3 or ip.src == 110.242.68.3

No.	Time	Source	Destination	Protocol	Length	Info
3756	9.909486	110.242.68.3	10.172.10.127	TCP	901	[TCP Retransmission] 443 → 55720 [PSH, ACK] Seq=4357 Ack=1724 Win=82432 Len=847
3757	9.909405	110.242.68.3	10.172.10.127	TCP	901	[TCP Retransmission] 443 → 55710 [PSH, ACK] Seq=4357 Ack=1724 Win=33024 Len=847
3759	9.909559	10.172.10.127	110.242.68.3	TCP	66	55720 → 443 [ACK] Seq=1724 Ack=5204 Win=132096 Len=0 SLE=4357 SRE=5204
3760	9.909617	10.172.10.127	110.242.68.3	TCP	66	55719 → 443 [ACK] Seq=1724 Ack=5204 Win=132096 Len=0 SLE=4357 SRE=5204
3762	9.912113	10.172.10.127	110.242.68.3	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
3763	9.912554	10.172.10.127	110.242.68.3	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
3764	9.912742	10.172.10.127	110.242.68.3	TCP	1506	55719 → 443 [ACK] Seq=1850 Ack=5204 Win=132096 Len=1452 [TCP PDU reassembled in 3765]
3765	9.912742	10.172.10.127	110.242.68.3	TLSv1.2	505	Application Data
3802	10.003968	110.242.68.3	10.172.10.127	TCP	60	443 → 55720 [ACK] Seq=5204 Ack=1850 Win=82432 Len=0
3803	10.003968	110.242.68.3	10.172.10.127	TLSv1.2	280	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
3804	10.003968	110.242.68.3	10.172.10.127	TCP	60	443 → 55719 [ACK] Seq=5204 Ack=1850 Win=33024 Len=0
3805	10.003968	110.242.68.3	10.172.10.127	TCP	66	[TCP Dup ACK 3804#1] 443 → 55719 [ACK] Seq=5204 Ack=1850 Win=35068 Len=0 SLE=3302 SRE=3753
3806	10.003968	110.242.68.3	10.172.10.127	TCP	60	443 → 55719 [ACK] Seq=5204 Ack=3753 Win=38912 Len=0
3807	10.003968	110.242.68.3	10.172.10.127	TLSv1.2	280	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
3818	10.054435	10.172.10.127	110.242.68.3	TCP	54	55720 → 443 [ACK] Seq=1850 Ack=5430 Win=131840 Len=0
3819	10.054648	10.172.10.127	110.242.68.3	TCP	54	55719 → 443 [ACK] Seq=3753 Ack=5430 Win=131840 Len=0
3831	10.191251	110.242.68.3	10.172.10.127	TLSv1.2	1262	Application Data
3832	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=6638 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3833]
3833	10.195367	110.242.68.3	10.172.10.127	TLSv1.2	989	Application Data
3834	10.195367	110.242.68.3	10.172.10.127	TLSv1.2	1506	Application Data, Application Data
3835	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=10477 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3837]
3836	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=11929 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3837]
3837	10.195367	110.242.68.3	10.172.10.127	TLSv1.2	441	Application Data
3838	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=13768 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3842]
3839	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=15220 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3842]

Ethernet II, Src: H3CTechnolog\_b4:e0:01 (38:97:d6:b4:e0:01), Dst: Intel\_1b:33:0c (e4:0d:36:1b:33:0c)

Internet Protocol Version 4, Src: 110.242.68.3, Dst: 10.172.10.127

Transmission Control Protocol, Src Port: 443, Dst Port: 55720, Seq: 5204, Ack: 1850, Len: 226

Transport Layer Security

- TLSv1.2 Record Layer: Handshake Protocol: New Session Ticket
  - Content Type: Handshake (22)
  - Version: TLS 1.2 (0x0303)
  - Length: 170
  - Handshake Protocol: New Session Ticket
    - Handshake Type: New Session Ticket (4)
    - Length: 166
    - TLS Session Ticket
- TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
  - Content Type: Change Cipher Spec (20)
  - Version: TLS 1.2 (0x0303)
  - Length: 1
  - Change Cipher Spec Message
- TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
  - Content Type: Handshake (22)
  - Content Type (tls.record.content type): 1 byte

Packets: 6641 · Displayed: 342 (5.1%) · Dropped: 0 (0.0%) Profile: Default

继续查看之后的报文，发现 Server 和 Client 之间开始不断交换 Application Data，这表明 Client 和 Server 已经验证并重用了之前的 Session 并开始了加密通信。通信的具体载荷即 HTTPS 协议报文，也就是百度主页的相关信息。

baudu,ssl.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ip.dst == 110.242.68.3 or ip.src == 110.242.68.3

No.	Time	Source	Destination	Protocol	Length	Info
3804	10.003968	110.242.68.3	10.172.10.127	TCP	60	443 → 55719 [ACK] Seq=5204 Ack=1850 Win=33024 Len=0
3805	10.003968	110.242.68.3	10.172.10.127	TCP	66	[TCP Dup ACK 3804#1] 443 → 55719 [ACK] Seq=5204 Ack=1850 Win=35068 Len=0 SLE=3302 SRE=3753
3806	10.003968	110.242.68.3	10.172.10.127	TCP	60	443 → 55719 [ACK] Seq=5204 Ack=3753 Win=38912 Len=0
3807	10.003968	110.242.68.3	10.172.10.127	TLSv1.2	280	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
3818	10.054435	10.172.10.127	110.242.68.3	TCP	54	55720 → 443 [ACK] Seq=1850 Ack=5430 Win=131840 Len=0
3819	10.054648	10.172.10.127	110.242.68.3	TCP	54	55719 → 443 [ACK] Seq=3753 Ack=5430 Win=131840 Len=0
3831	10.191251	110.242.68.3	10.172.10.127	TLSv1.2	1262	Application Data
3832	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=6638 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3833]
3833	10.195367	110.242.68.3	10.172.10.127	TLSv1.2	989	Application Data
3834	10.195367	110.242.68.3	10.172.10.127	TLSv1.2	1506	Application Data, Application Data
3835	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=10477 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3837]
3836	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=11929 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3837]
3837	10.195367	110.242.68.3	10.172.10.127	TLSv1.2	441	Application Data
3838	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=13768 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3842]
3839	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=15220 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3842]
3840	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=16672 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3842]
3841	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=18124 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3842]
3842	10.195367	110.242.68.3	10.172.10.127	TLSv1.2	1506	Application Data
3843	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=21028 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3846]
3844	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=22480 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3846]
3845	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=23932 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3846]
3846	10.195367	110.242.68.3	10.172.10.127	TLSv1.2	1506	Application Data, Application Data
3847	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=26836 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3849]
3848	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=28288 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3849]
3849	10.195367	110.242.68.3	10.172.10.127	TLSv1.2	1506	Application Data
3850	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=31192 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3852]
3851	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=32644 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3852]
3852	10.195367	110.242.68.3	10.172.10.127	TLSv1.2	1506	Application Data
3853	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=35548 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3918]
3854	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=37008 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3918]
3855	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=38452 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3918]
3856	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=39904 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3918]
3857	10.195367	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=41356 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3918]
3859	10.195516	10.172.10.127	110.242.68.3	TCP	54	55719 → 443 [ACK] Seq=3753 Ack=42808 Win=132096 Len=0
3918	10.223945	110.242.68.3	10.172.10.127	TLSv1.2	1506	Application Data, Application Data
3919	10.223945	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=44260 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3921]
3920	10.223945	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=45712 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3921]
3921	10.223945	110.242.68.3	10.172.10.127	TLSv1.2	1506	Application Data
3922	10.223945	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=48616 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3924]
3923	10.223945	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=50068 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3924]
3924	10.223945	110.242.68.3	10.172.10.127	TLSv1.2	1506	Application Data
3925	10.223945	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=52972 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3934]
3926	10.223945	110.242.68.3	10.172.10.127	TCP	1506	443 → 55719 [ACK] Seq=54424 Ack=3753 Win=38912 Len=1452 [TCP PDU reassembled in 3934]

Frame 3839: 1506 bytes on wire (12048 bits), 1506 bytes captured (12048 bits) on interface \Device\NPF {63021234-E510-455D-9C46-34B6A0E76886}, id 0

Ready to load or capture

Packets: 6641 · Displayed: 342 (5.1%) · Dropped: 0 (0.0%) Profile: Default

至此，SSL 协议的握手部分分析结束。

## 4 心得体会

---

本次实验我通过实践认识了互联网中各种重要的加密算法，同时对当前互联网中基础的 SSL 加密协议有了基本认识。也了解了中间人攻击、对称加密和非对称加密、网络证书等概念。之前我一直不知道为何网络证书在公钥交换中的重要性，并认为公钥交换就是互相发送公钥。通过资料查阅我明白了 SSL 协议为何需要通过交换加密后的测试报文确认没有中间人攻击，让我深刻认识到了网络证书的意义。同时我也迅速删掉了学校的 EasyConnect VPN，因为这个软件会强制安装深信服的数字证书，而这个公司已经被多次曝光使用恶意连接和注入等方式扫描用户信息。如果深信服扮演成中间人，那么当我使用 EasyConnect VPN 时我的信息安全将不再得到保障。