0305sherrynan / - Public

网络安全实验

0 stars 0 forks

Star Unwatch

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master

0305sherrynan ... 1 minute ago

View code

README.MD

PKI实验

1、实验内容

安装PKI平台软件，练习证书的申请，创建，分发等操作

2、实验过程

2.1、 软件的安装

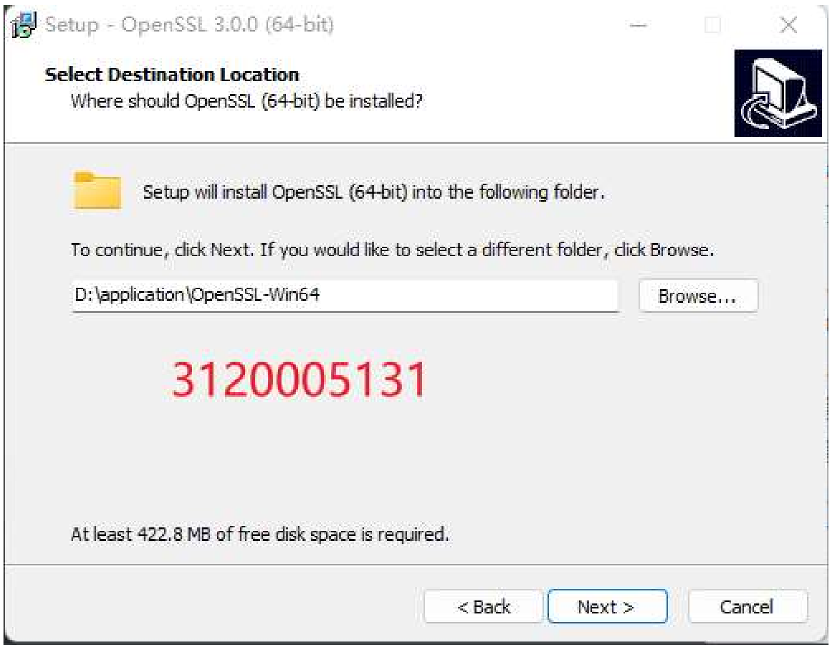
这里我们选择openssl平台软件 理由如下：

- 1、密钥和证书管理是PKI的一个重要组成部分，OpenSSL为之提供了丰富的功能，支持多种标准。
- 2、OpenSSL提供的CA应用程序就是一个小型的证书管理中心（CA），实现了证书签发的整个流程和证书管理的大部分机制。
- 3、OpenSSL支持Linux、Windows、BSD、Mac、VMS等平台，这使得OpenSSL具有广泛的适用性

2.2、前提准备

2.2.1、安装openssl

我们通过渠道下载windows版64位的安装包


Setup - OpenSSL 3.0.0 (64-bit)

Select Destination Location

Where should OpenSSL (64-bit) be installed?

Setup will install OpenSSL (64-bit) into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

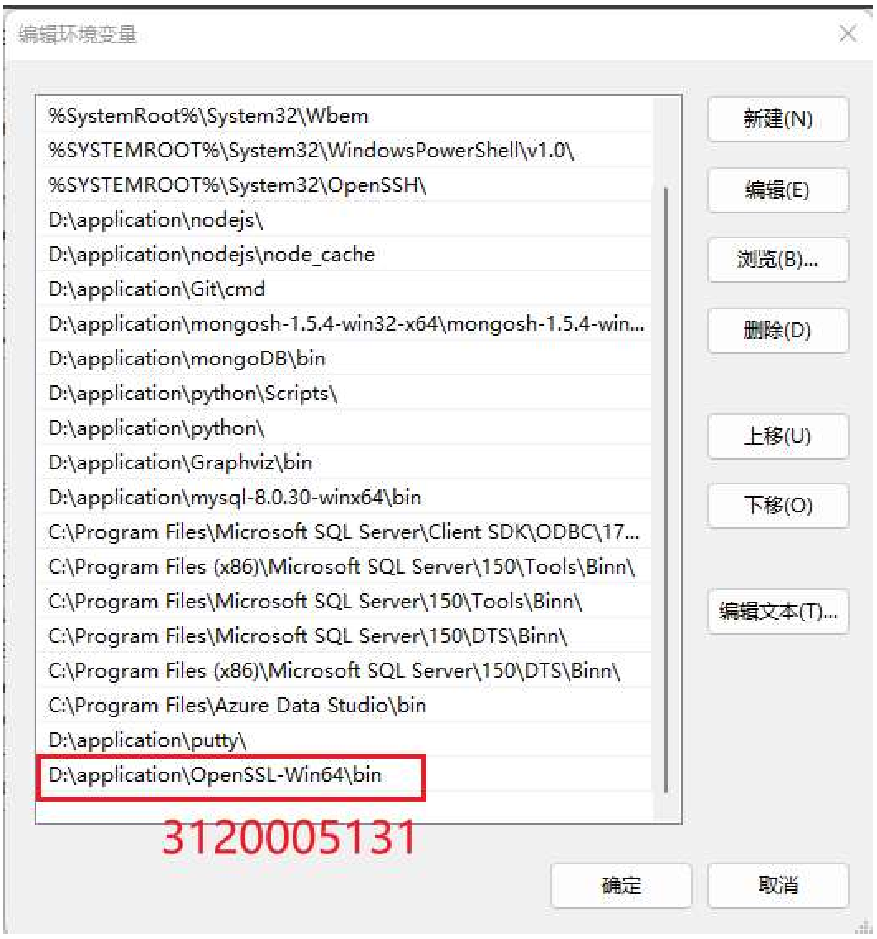
D:\application\OpenSSL-Win64

3120005131

At least 422.8 MB of free disk space is required.

2.2.2、环境配置

下载完毕后设置系统环境变量，将刚刚所安装的路径下的bin的路径，加入到path

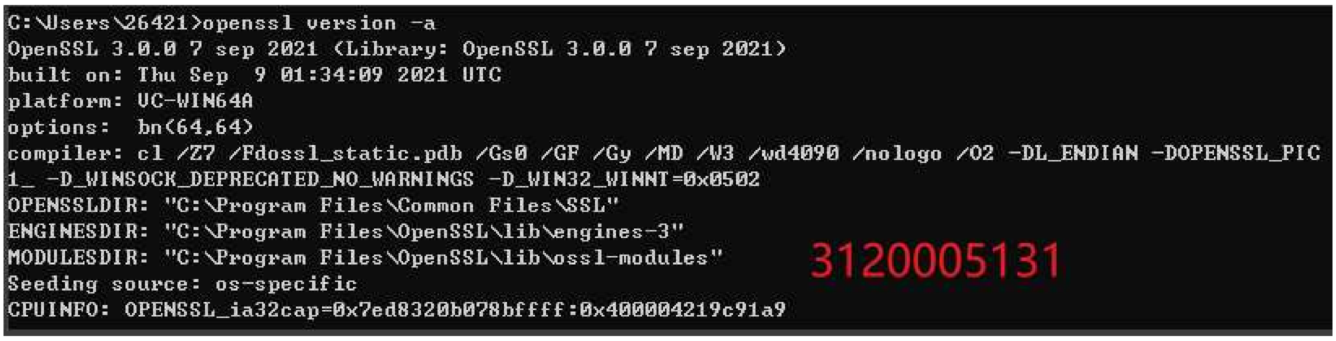

编辑环境变量

```
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
D:\application\nodejs\
D:\application\nodejs\node_cache
D:\application\Git\cmd
D:\application\mongosh-1.5.4-win32-x64\mongosh-1.5.4-win...
D:\application\mongoDB\bin
D:\application\python\Scripts\
D:\application\python\
D:\application\Graphviz\bin
D:\application\mysql-8.0.30-winx64\bin
C:\Program Files\Microsoft SQL Server\Client SDK\ODBC\17...
C:\Program Files (x86)\Microsoft SQL Server\150\Tools\Binn\
C:\Program Files\Microsoft SQL Server\150\Tools\Binn\
C:\Program Files\Microsoft SQL Server\150\DTS\Binn\
C:\Program Files (x86)\Microsoft SQL Server\150\DTS\Binn\
C:\Program Files\Azure Data Studio\bin
D:\application\putty\
D:\application\OpenSSL-Win64\bin
```

3120005131

然后cmd打开 可通过以下命令查看是否安装成功

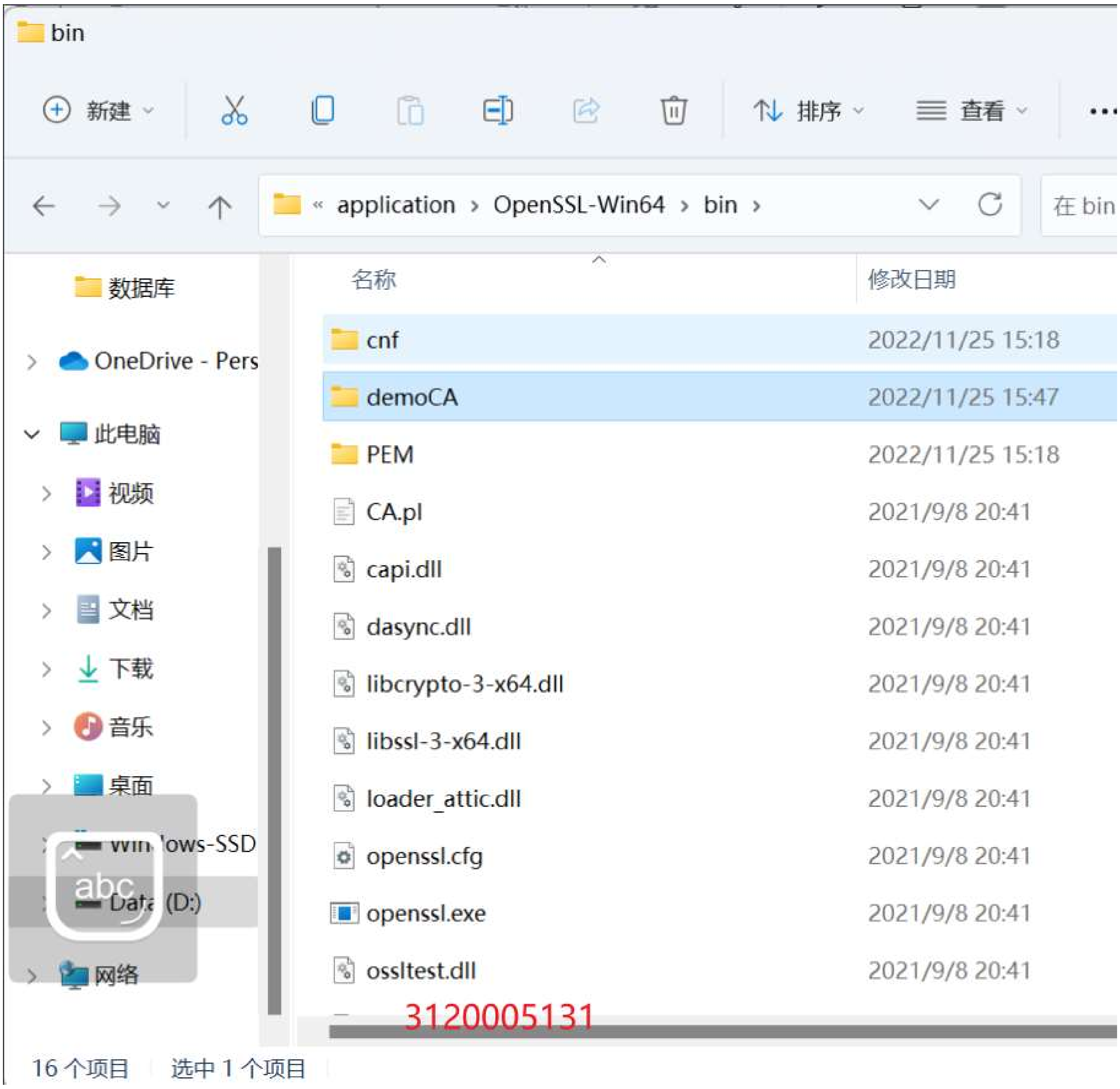
> openssl version -a



```
C:\Users\26421>openssl version -a
OpenSSL 3.0.0 7 sep 2021 (Library: OpenSSL 3.0.0 7 sep 2021)
built on: Thu Sep  9 01:34:09 2021 UTC
platform: VC-WIN64A
options:  bn(64,64)
compiler: cl /Z7 /Fdossl_static.pdb /Gs0 /GF /Gy /MD /W3 /wd4090 /nologo /O2 -DL_ENDIAN -DOPENSSL_PIC
1_ -D_WINSOCK_DEPRECATED_NO_WARNINGS -D_WIN32_WINNT=0x0502
OPENSSLDIR: "C:\Program Files\Common Files\SSL"
ENGINESDIR: "C:\Program Files\OpenSSL\lib\engines-3"
MODULESDIR: "C:\Program Files\OpenSSL\lib\ossl-modules"
Seeding source: os-specific
CPUINFO: OPENSSL_ia32cap=0x7ed8320b078bffff:0x400004219c91a9
```

2.2.3、功能文件的配置

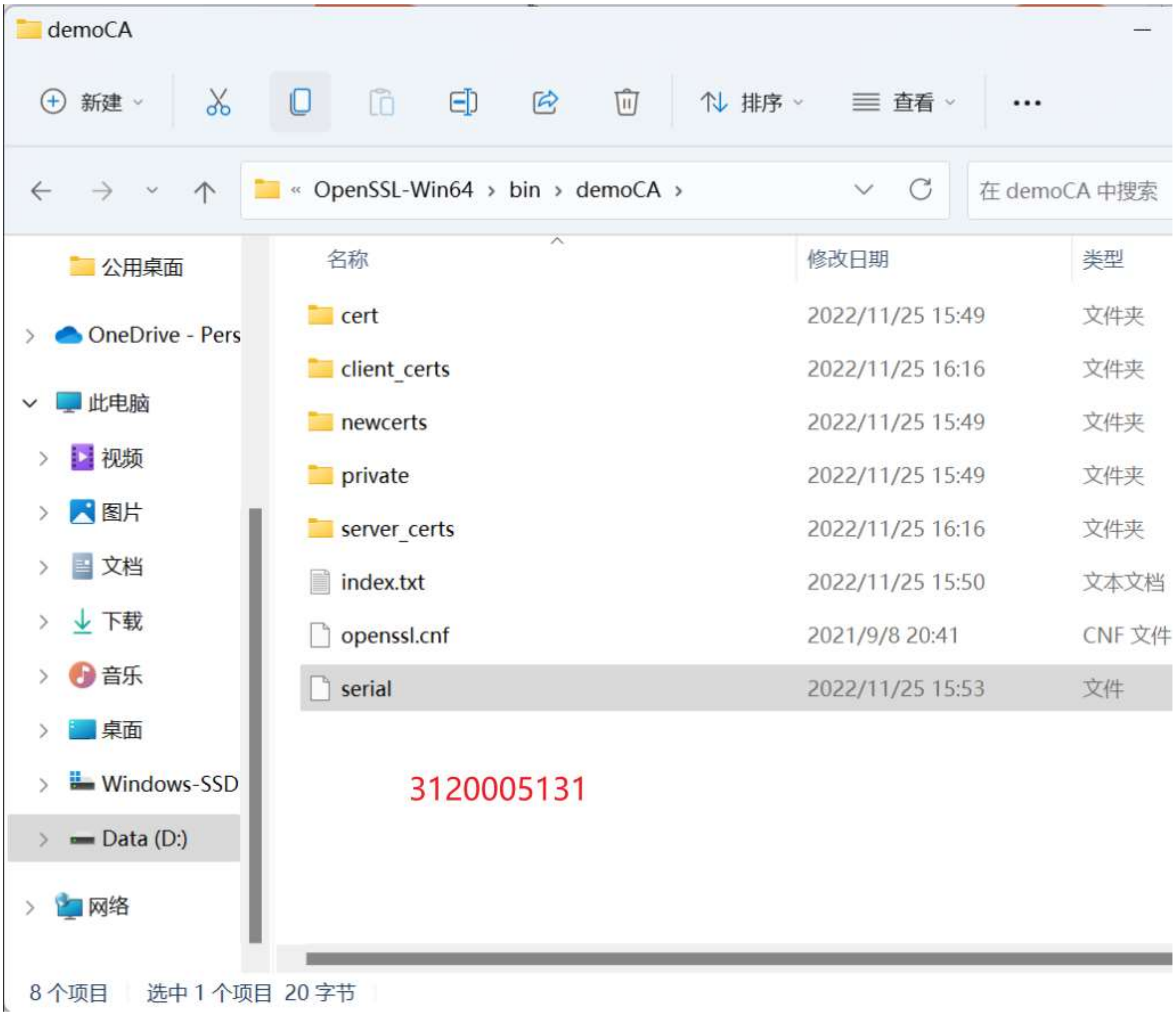
之后到任意地方新建一个文件夹，这里以demoCA为例子



名称	修改日期
cnf	2022/11/25 15:18
demoCA	2022/11/25 15:47
PEM	2022/11/25 15:18
CA.pl	2021/9/8 20:41
capi.dll	2021/9/8 20:41
dasync.dll	2021/9/8 20:41
libcrypto-3-x64.dll	2021/9/8 20:41
libssl-3-x64.dll	2021/9/8 20:41
loader_attic.dll	2021/9/8 20:41
openssl.cfg	2021/9/8 20:41
openssl.exe	2021/9/8 20:41
ossltest.dll	2021/9/8 20:41

在该目录下，我们需要新建以下文件

> 文件功能 private: 用于存放CA私钥 cert：用于存放CA自签的CA证书 client_certs: 用于存放经CA签发好的客户端证书集合 server_certs: 用于存放经CA签发好的服务器证书集合 newcerts: 用于存放经CA签发好的所有证书集合并编号 index：用于存放证书索引信息 serial： 用于存放证书编号，并在serial文件中添加10000001初始编号，之后每签名一个新证书，编号自动+1 openssl.cnf为配置文件，可以从目录bin/cnf中拷贝过来

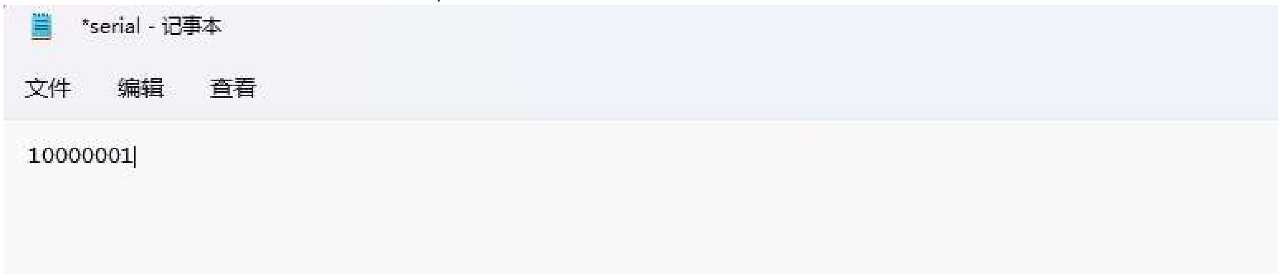


名称	修改日期	类型
cert	2022/11/25 15:49	文件夹
client_certs	2022/11/25 16:16	文件夹
newcerts	2022/11/25 15:49	文件夹
private	2022/11/25 15:49	文件夹
server_certs	2022/11/25 16:16	文件夹
index.txt	2022/11/25 15:50	文本文档
openssl.cnf	2021/9/8 20:41	CNF 文件
serial	2022/11/25 15:53	文件

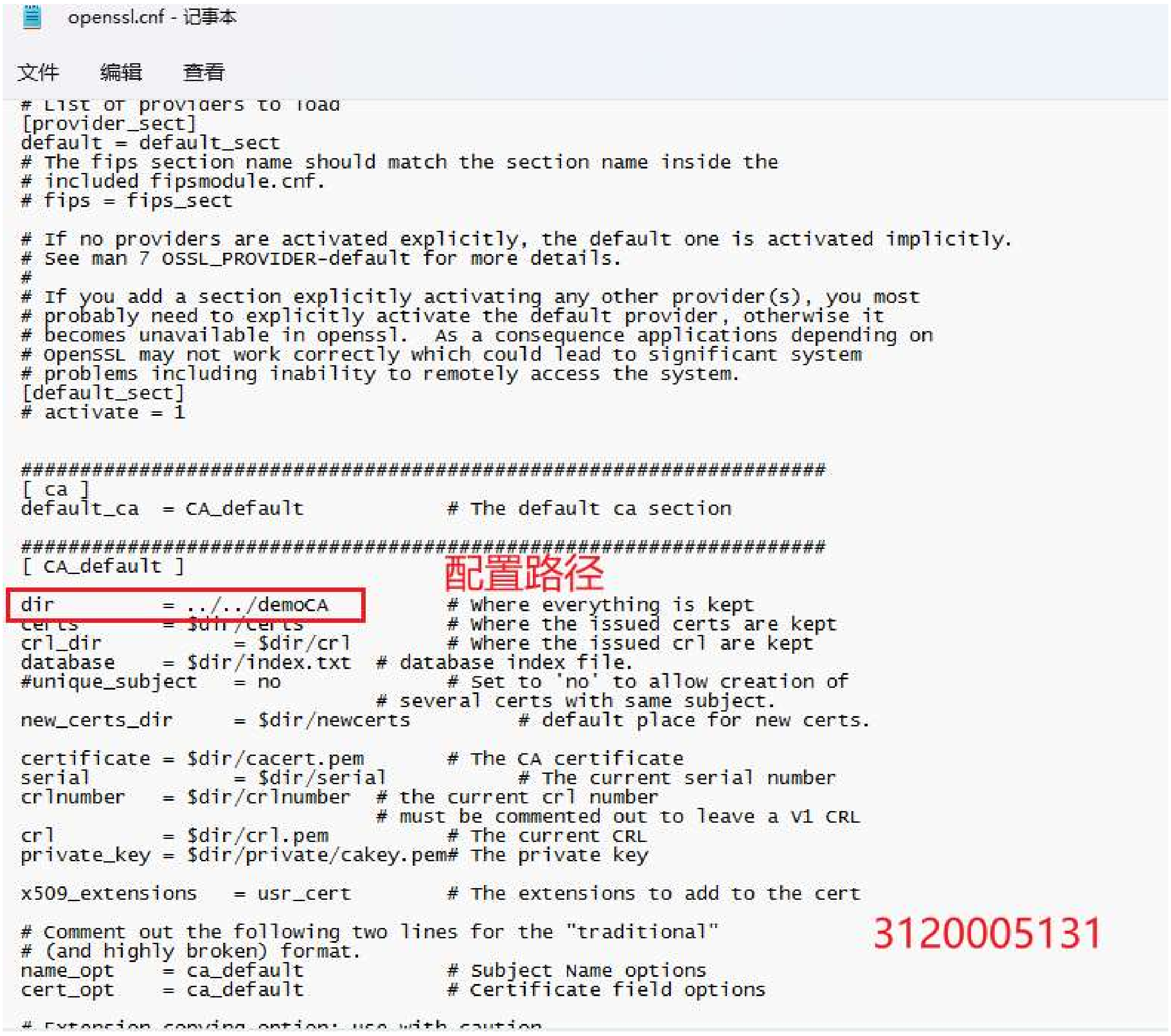
3120005131

8 个项目 选中 1 个项目 20 字节

最后，我们需要将bin下cnf文件内的openssl.cnf文件拷贝一份到上述内，并在serial文件填入初始编号10000001



```
10000001
```



```
# List of providers to load
[provider_sect]
default = default_sect
# The fips section name should match the section name inside the
# included fipsmodule.cnf.
# fips = fips_sect

# If no providers are activated explicitly, the default one is activated implicitly.
# See man 7 OSSL_PROVIDER-default for more details.
#
# If you add a section explicitly activating any other provider(s), you most
# probably need to explicitly activate the default provider, otherwise it
# becomes unavailable in openssl.  As a consequence applications depending on
# OpenSSL may not work correctly which could lead to significant system
# problems including inability to remotely access the system.
[default_sect]
# activate = 1


####################################################################
[ ca ]
default_ca  = CA_default            # The default ca section

####################################################################
[ CA_default ]

dir         = ../../demoCA           # Where everything is kept
certs       = $dir/certs            # Where the issued certs are kept
crl_dir           = $dir/crl        # Where the issued crl are kept
database    = $dir/index.txt  # database index file.
#unique_subject    = no             # Set to 'no' to allow creation of
                              # several certs with same subject.
new_certs_dir     = $dir/newcerts         # default place for new certs.

certificate = $dir/cacert.pem       # The CA certificate
serial            = $dir/serial           # The current serial number
crlnumber   = $dir/crlnumber  # the current crl number
                              # must be commented out to leave a V1 CRL
crl         = $dir/crl.pem          # The current CRL
private_key = $dir/private/cakey.pem# The private key

x509_extensions   = usr_cert        # The extensions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt    = ca_default            # Subject Name options
cert_opt    = ca_default            # Certificate field options

# Extension copying option: use with caution.
```

2.3、建立CA，创建自签名证书

在自己创建的文件夹下以命令行形式进入（这里为demoCA）

2.3.1、生成CA ECC密钥

输入以下命令

> openssl ecparam -out private/ec-cakey.pem -name prime256v1 -genkey

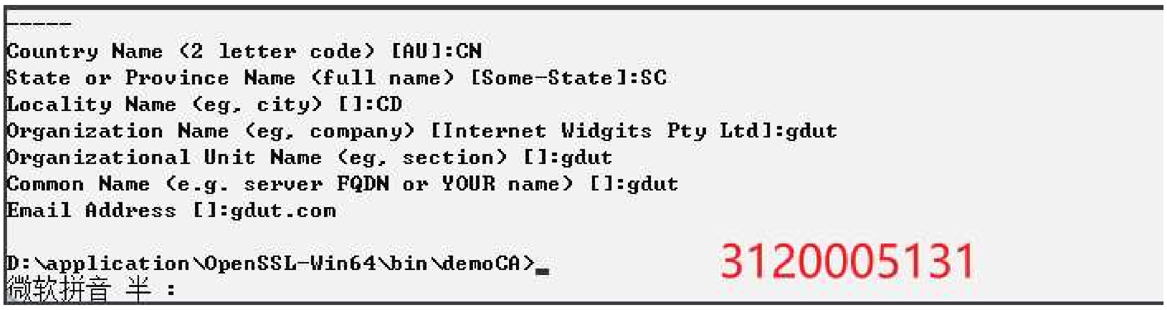
生成秘钥时，openssl默认仅存储曲线的名字 输入以下命令

> openssl ecparam -in private/ec-cakey.pem -text -noout

2.3.1、生成CA证书

使用上一步生成的CA私钥，生成CA证书

> openssl req -new -x509 -days 3650 -config openssl.cnf -extensions v3_ca -key private/ec-cakey.pem -out cert/ec-cacert.pem

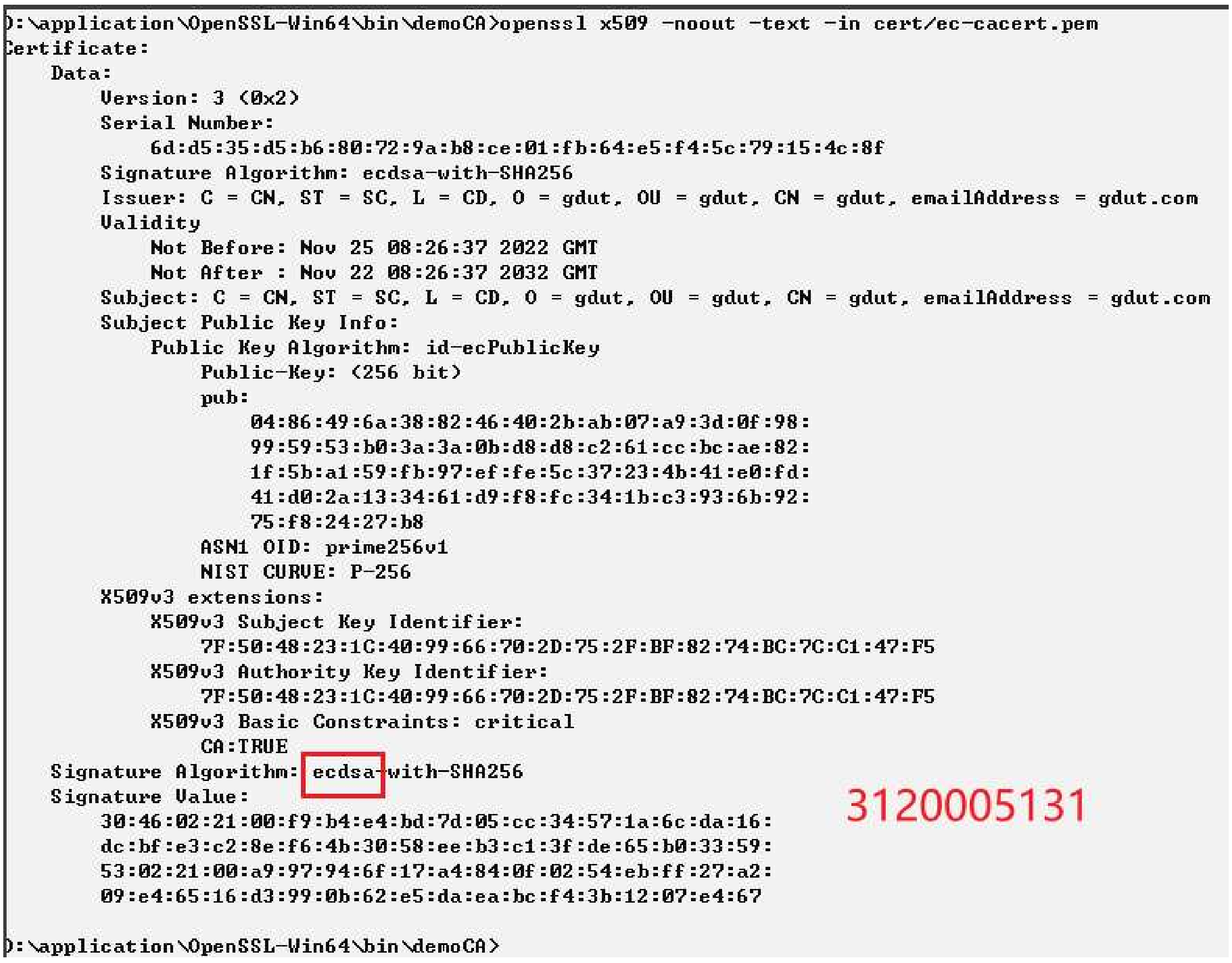


```
-----
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:SC
Locality Name (eg, city) []:CD
Organization Name (eg, company) [Internet Widgits Pty Ltd]:gdut
Organizational Unit Name (eg, section) []:gdut
Common Name (e.g. server FQDN or YOUR name) []:gdut
Email Address []:gdut.com

D:\application\OpenSSL-Win64\bin\demoCA>_          3120005131
微软拼音 半 :
```

接下来，我们可以验证下CA证书的内容和使用的签名算法

> openssl x509 -noout -text -in cert/ec-cacert.pem

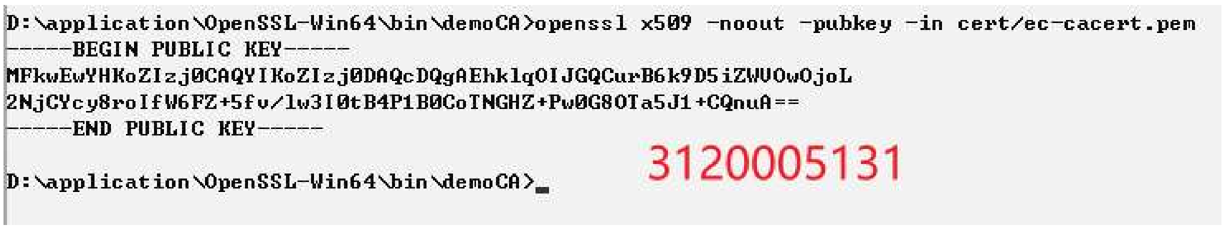
可以看到，我们使用的是ECDSA签名算法来生成我们的CA证书，而不是使用的RSA。



```
D:\application\OpenSSL-Win64\bin\demoCA>openssl x509 -noout -text -in cert/ec-cacert.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            6d:d5:35:d5:b6:80:72:9a:b8:ce:01:fb:64:e5:f4:5c:79:15:4c:8f
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = CN, ST = SC, L = CD, O = gdut, OU = gdut, CN = gdut, emailAddress = gdut.com
        Validity
            Not Before: Nov 25 08:26:37 2022 GMT
            Not After : Nov 22 08:26:37 2032 GMT
        Subject: C = CN, ST = SC, L = CD, O = gdut, OU = gdut, CN = gdut, emailAddress = gdut.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:86:49:6a:38:82:46:40:2b:ab:07:a9:3d:0f:98:
                    99:59:53:b0:3a:3a:0b:d8:d8:c2:61:cc:bc:ae:82:
                    1f:5b:a1:59:fb:97:ef:fe:5c:37:23:4b:41:e0:fd:
                    41:d0:2a:13:34:61:d9:f8:fc:34:1b:c3:93:6b:92:
                    75:f8:24:27:b8
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                7F:50:48:23:1C:40:99:66:70:2D:75:2F:BF:82:74:BC:7C:C1:47:F5
            X509v3 Authority Key Identifier:
                7F:50:48:23:1C:40:99:66:70:2D:75:2F:BF:82:74:BC:7C:C1:47:F5
            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:                                        3120005131
        30:46:02:21:00:f9:b4:e4:bd:7d:05:cc:34:57:1a:6c:da:16:
        dc:bf:e3:c2:8e:f6:4b:30:58:ee:b3:c1:3f:de:65:b0:33:59:
        53:02:21:00:a9:97:94:6f:17:a4:84:0f:02:54:eb:ff:27:a2:
        09:e4:65:16:d3:99:0b:62:e5:da:ea:bc:f4:3b:12:07:e4:67

D:\application\OpenSSL-Win64\bin\demoCA>
```

使用私钥验证CA证书

> openssl x509 -noout -pubkey -in cert/ec-cacert.pem

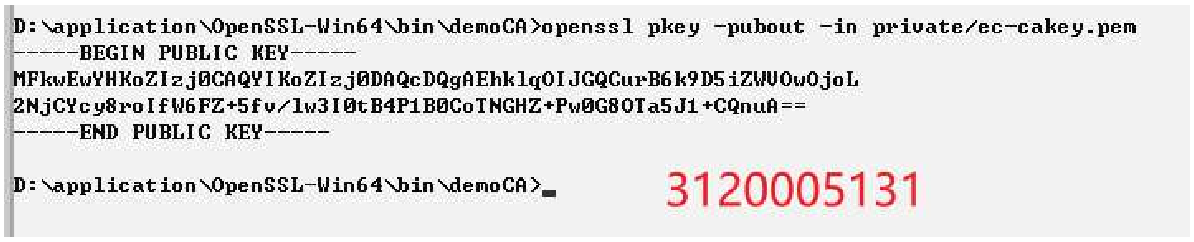


```
D:\application\OpenSSL-Win64\bin\demoCA>openssl x509 -noout -pubkey -in cert/ec-cacert.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhklqOIJGQCurB6k9D5iZWVOw0joL
2NjCYcy8roIfW6FZ+5fv/lw3I0tB4P1B0CoTNGHZ+Pw0G80Ta5J1+CQnuA==
-----END PUBLIC KEY-----

D:\application\OpenSSL-Win64\bin\demoCA>_          3120005131
```

类似的我们可以从私钥导出公钥

> openssl pkey -pubout -in private/ec-cakey.pem

可以看到生成的公钥是相同的



```
D:\application\OpenSSL-Win64\bin\demoCA>openssl pkey -pubout -in private/ec-cakey.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhklqOIJGQCurB6k9D5iZWVOwOjoL
2NjCYcy8roIfW6FZ+5fv/lw3I0tB4P1B0CoTNGHZ+Pw0G8OTa5J1+CQnuA==
-----END PUBLIC KEY-----

D:\application\OpenSSL-Win64\bin\demoCA>
```

2.4、使用CA私钥和证书签发服务端证书

2.4.1、生成服务器端私钥

命令行进入server_certs文件夹

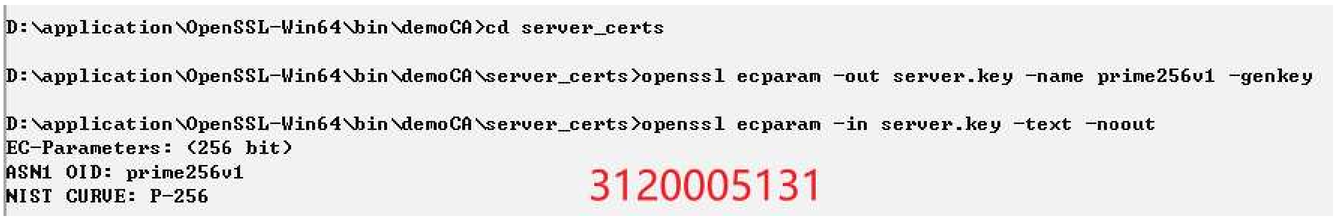
> cd server_certs

我们再一次使用曲线prime256v1生成ECC的私钥

> openssl ecparam -out server.key -name prime256v1 -genkey

验证曲线：

> openssl ecparam -in server.key -text -noout

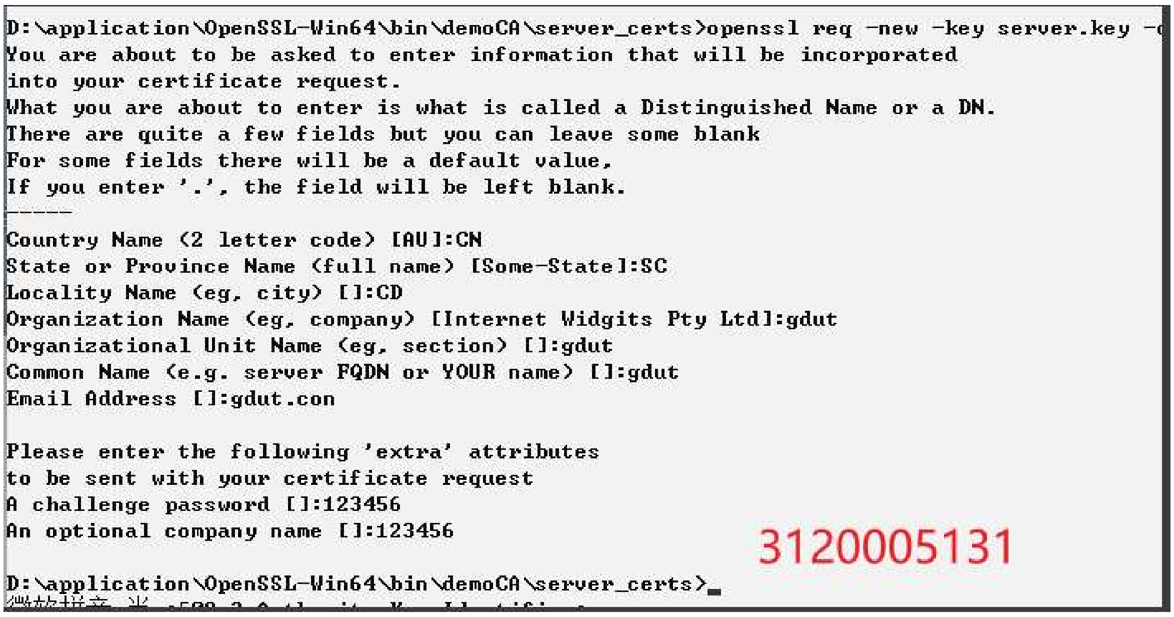


```
D:\application\OpenSSL-Win64\bin\demoCA>cd server_certs

D:\application\OpenSSL-Win64\bin\demoCA\server_certs>openssl ecparam -out server.key -name prime256v1 -genkey

D:\application\OpenSSL-Win64\bin\demoCA\server_certs>openssl ecparam -in server.key -text -noout
EC-Parameters: (256 bit)
ASN1 OID: prime256v1
NIST CURVE: P-256
```

2.4.2、 生成服务端证书请求文件CSR

生成CSR请求：

> openssl req -new -key server.key -out server.csr -sha256



```
D:\application\OpenSSL-Win64\bin\demoCA\server_certs>openssl req -new -key server.key -
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:SC
Locality Name (eg, city) []:CD
Organization Name (eg, company) [Internet Widgits Pty Ltd]:gdut
Organizational Unit Name (eg, section) []:gdut
Common Name (e.g. server FQDN or YOUR name) []:gdut
Email Address []:gdut.con

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123456
An optional company name []:123456

D:\application\OpenSSL-Win64\bin\demoCA\server_certs>_
```

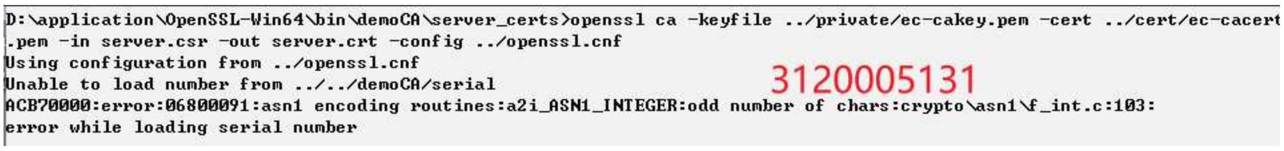
3120005131

现在我们使用ECC CA私钥，CA证书，对server.csr进行签名，生成服务端证书：

> openssl ca -keyfile ../private/ec-cakey.pem -cert ../cert/ec-cacert.pem -in server.csr -out server.crt -config ../openssl.cnf

这里我们需要注意路径问题

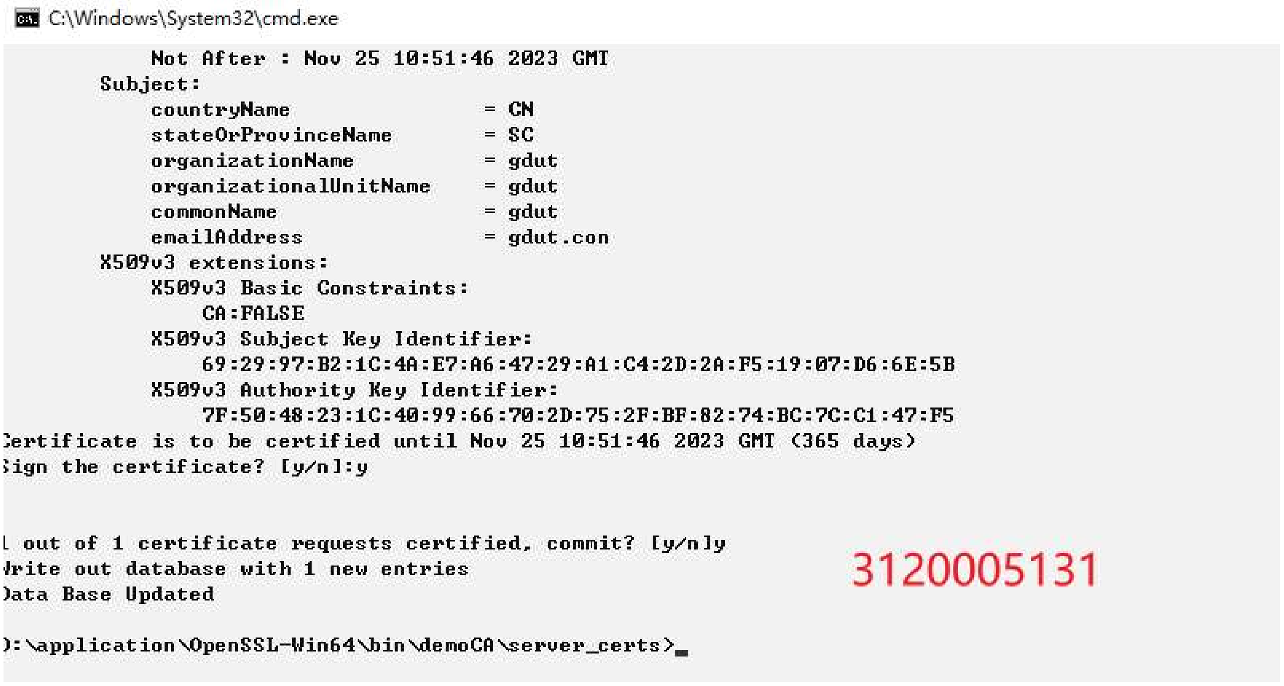
- 报错情况



```
D:\application\OpenSSL-Win64\bin\demoCA\server_certs>openssl ca -keyfile ../private/ec-cakey.pem -cert ../cert/ec-cacert
.pem -in server.csr -out server.crt -config ../openssl.cnf
Using configuration from ../openssl.cnf
Unable to load number from ../../demoCA/serial
ACB70000:error:06800091:asn1 encoding routines:a2i_ASN1_INTEGER:odd number of chars:crypto\asn1\f_int.c:103:
error while loading serial number
```

3120005131

- 我们可以根据报错情况，和当前所处目录位置，修改之前的openssl.cnf中的dir。修改后再次执行：



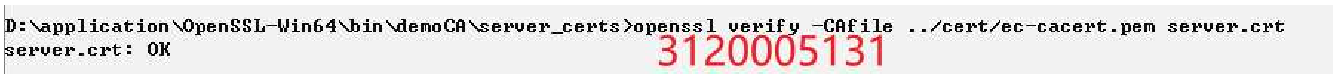
C:\Windows\System32\cmd.exe

```
            Not After : Nov 25 10:51:46 2023 GMT
        Subject:
            countryName               = CN
            stateOrProvinceName       = SC
            organizationName          = gdut
            organizationalUnitName    = gdut
            commonName                = gdut
            emailAddress              = gdut.con
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                69:29:97:B2:1C:4A:E7:A6:47:29:A1:C4:2D:2A:F5:19:07:D6:6E:5B
            X509v3 Authority Key Identifier:
                7F:50:48:23:1C:40:99:66:70:2D:75:2F:BF:82:74:BC:7C:C1:47:F5
Certificate is to be certified until Nov 25 10:51:46 2023 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated

D:\application\OpenSSL-Win64\bin\demoCA\server_certs>_
```

3120005131

2.4.3、验证证书是否有效时使用CA证书进行验证

> openssl verify -CAfile ../cert/ec-cacert.pem server.crt

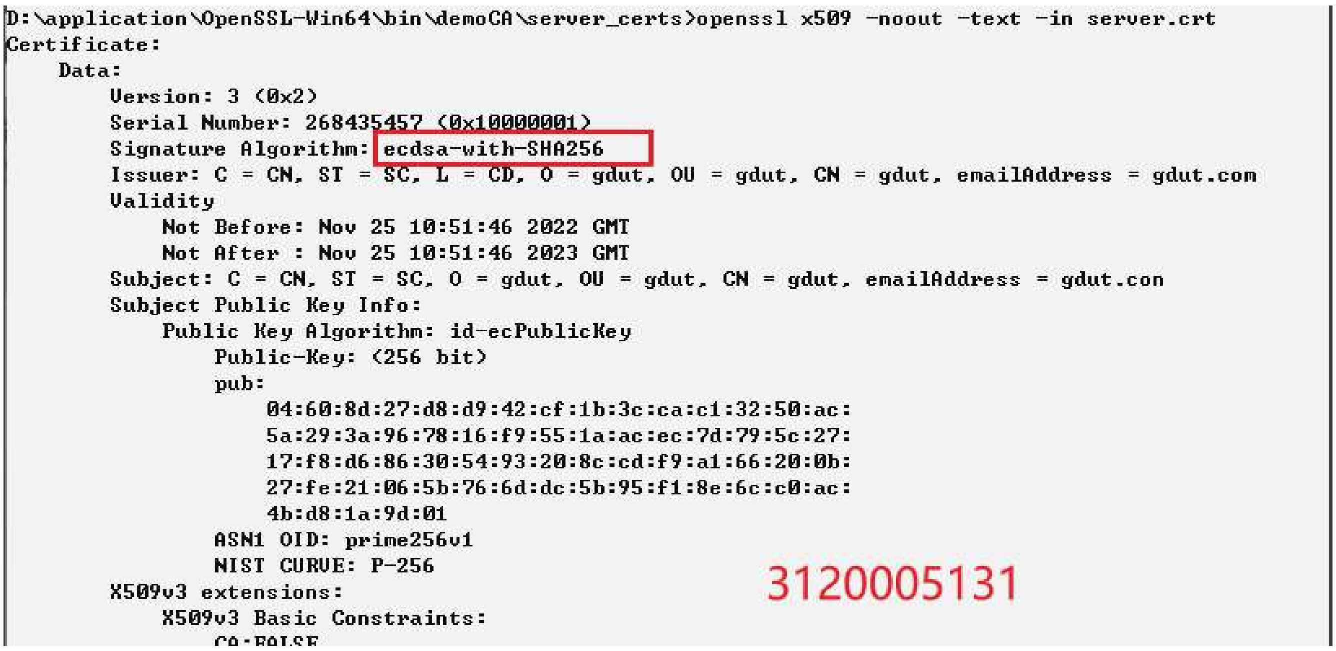


```
D:\application\OpenSSL-Win64\bin\demoCA\server_certs>openssl verify -CAfile ../cert/ec-cacert.pem server.crt
server.crt: OK
```

3120005131

我们也可以验证服务器证书的签名算法，确保使用的是ECC私钥

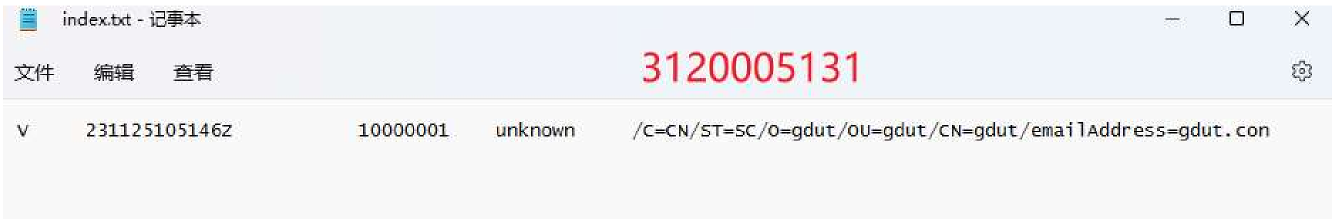
> openssl x509 -noout -text -in server.crt



```
D:\application\OpenSSL-Win64\bin\demoCA\server_certs>openssl x509 -noout -text -in server.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 268435457 (0x10000001)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = CN, ST = SC, L = CD, O = gdut, OU = gdut, CN = gdut, emailAddress = gdut.com
        Validity
            Not Before: Nov 25 10:51:46 2022 GMT
            Not After : Nov 25 10:51:46 2023 GMT
        Subject: C = CN, ST = SC, O = gdut, OU = gdut, CN = gdut, emailAddress = gdut.con
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:60:8d:27:d8:d9:42:cf:1b:3c:ca:c1:32:50:ac:
                    5a:29:3a:96:78:16:f9:55:1a:ac:ec:7d:79:5c:27:
                    17:f8:d6:86:30:54:93:20:8c:cd:f9:a1:66:20:0b:
                    27:fe:21:06:5b:76:6d:dc:5b:95:f1:8e:6c:c0:ac:
                    4b:d8:1a:9d:01
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
```

3120005131

2.4.4、**验证index.txt**



```
V	231125105146Z		10000001	unknown	/C=CN/ST=SC/O=gdut/OU=gdut/CN=gdut/emailAddress=gdut.con
```

2、实验小结

- 学习到了如何安装openssl，并配置相关环境变量，与之前配置python、 npm 类似，较为上手
- 了解到CA的工作功能：负责签发、认证和管理证书。并且学会使用openssl命令生成CA证书
- 能够生成服务器端证书，并且发送请求，且记录在index文件中

SSL实验

1、实验内容

配置并抓包分析 SSL协议

2、实验过程

2.1、 软件的安装

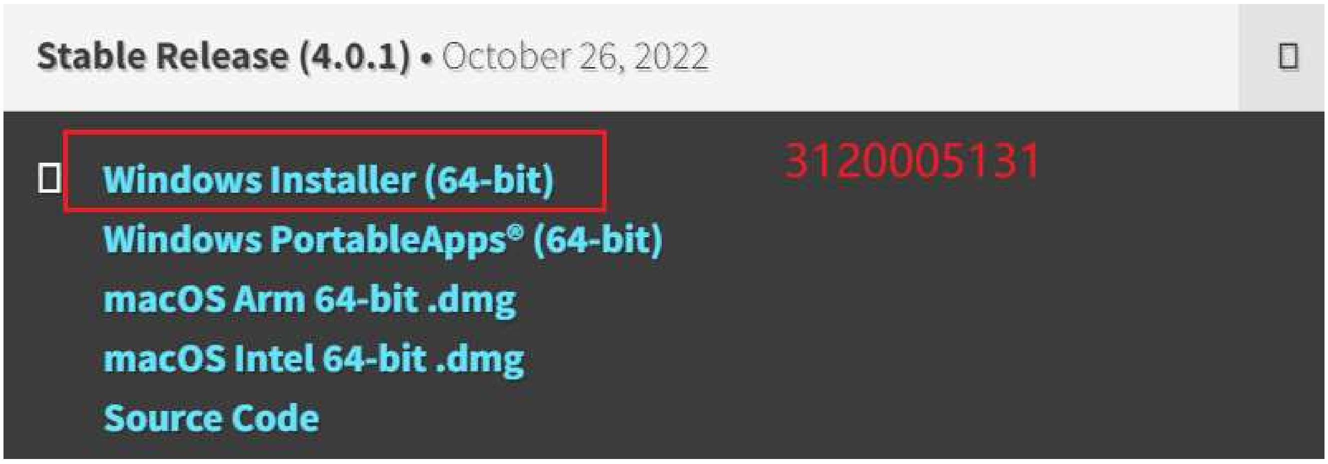
这里我们选择比较熟悉的抓包软件Wireshark 理由如下：

- 1、Wireshark是一个网络封包分析软件。网络封包分析软件的功能是截取网络封包，并尽可能显示出最为详细的网络封包资料。
- 2、Wireshark是一款免费且功能强大的软件，版本不断迭代更新，功能完善强大
- 3、Wireshark不仅仅在截取数据包上有着强大的功能，而且在不同协议下分析包也有着强大的追踪流，非常符合实验的功能要求

2.2、前提准备

2.2.1、安装Wireshark

我们在官网安装最新版本的wireshark，不建议安装较旧版本的，因为旧版本功能比较不齐全，可能会没有本实验所需要的TLS追踪流



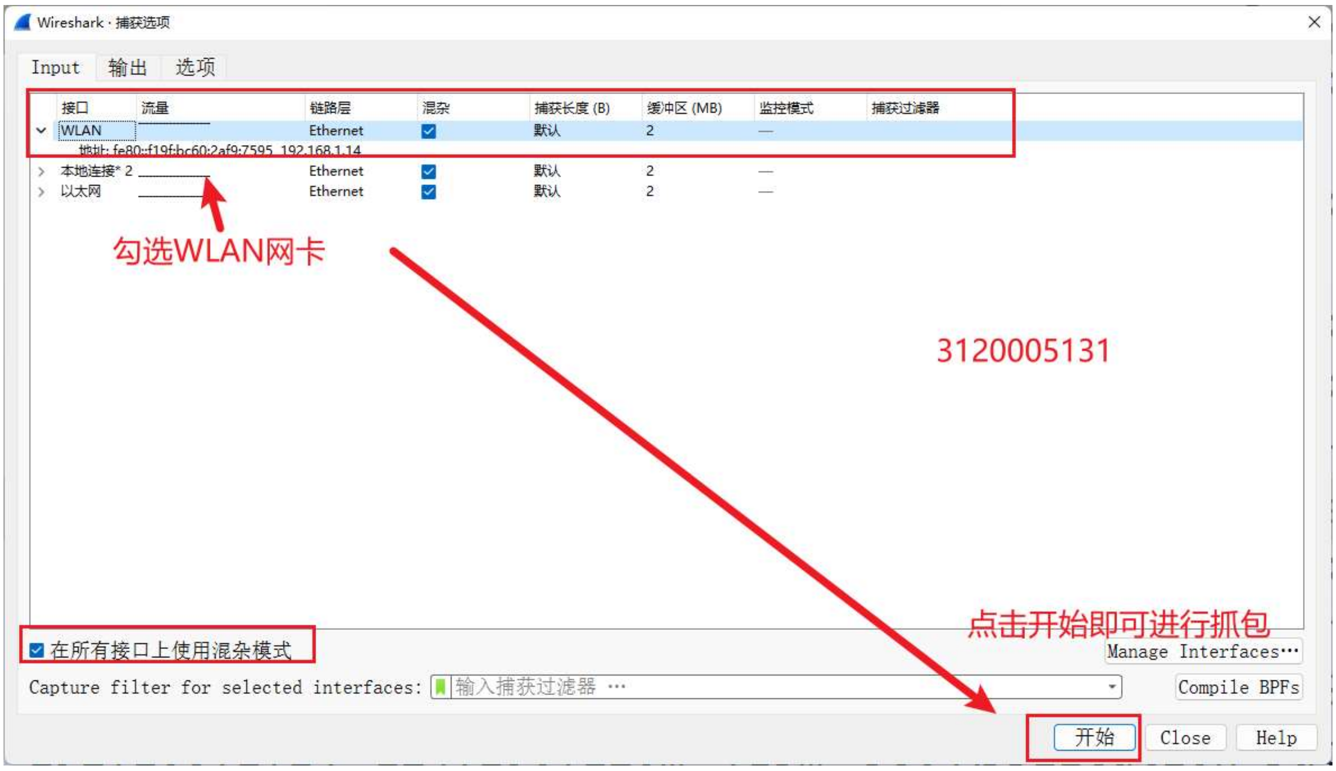
Stable Release (4.0.1) • October 26, 2022

Windows Installer (64-bit)
Windows PortableApps® (64-bit)
macOS Arm 64-bit .dmg
macOS Intel 64-bit .dmg
Source Code

3120005131

2.3、抓包流程

2.3.1、 在抓包前先配置好抓包所要监视的IP地址，这里我们选择WLAN的网卡



Wireshark · 捕获选项

Input 输出 选项

接口	流量	链路层	混杂	捕获长度 (B)	缓冲区 (MB)	监控模式	捕获过滤器
WLAN 地址: fe80::f19f:bc60:2af9:7595, 192.168.1.14		Ethernet	✓	默认	2	—	
本地连接* 2		Ethernet	✓	默认	2	—	
以太网		Ethernet	✓	默认	2	—	

勾选WLAN网卡

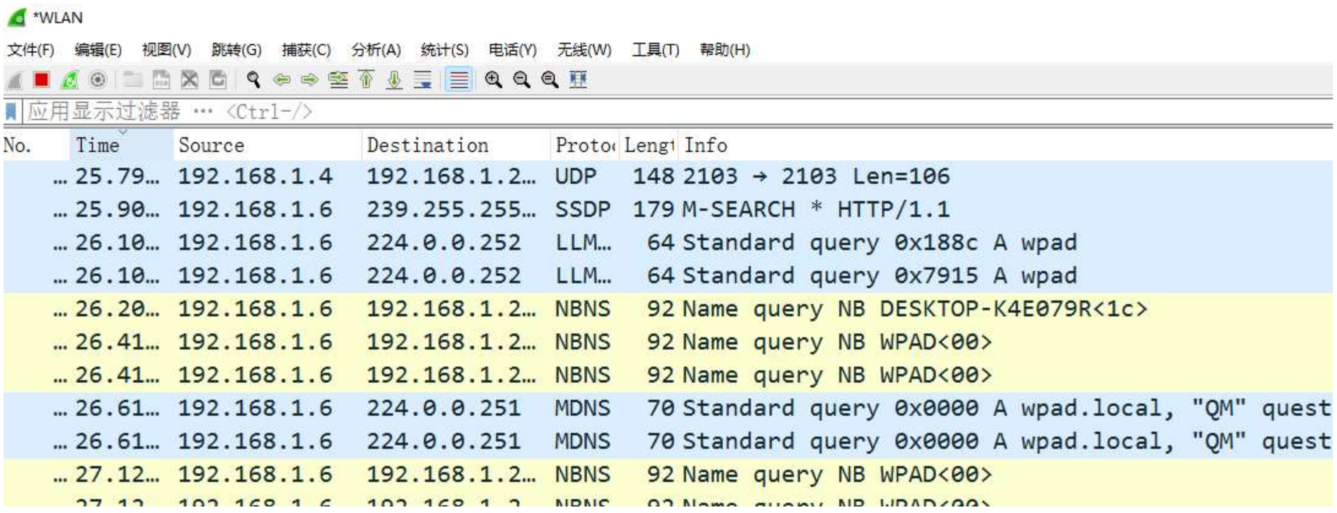
3120005131

点击开始即可进行抓包

[✓] 在所有接口上使用混杂模式

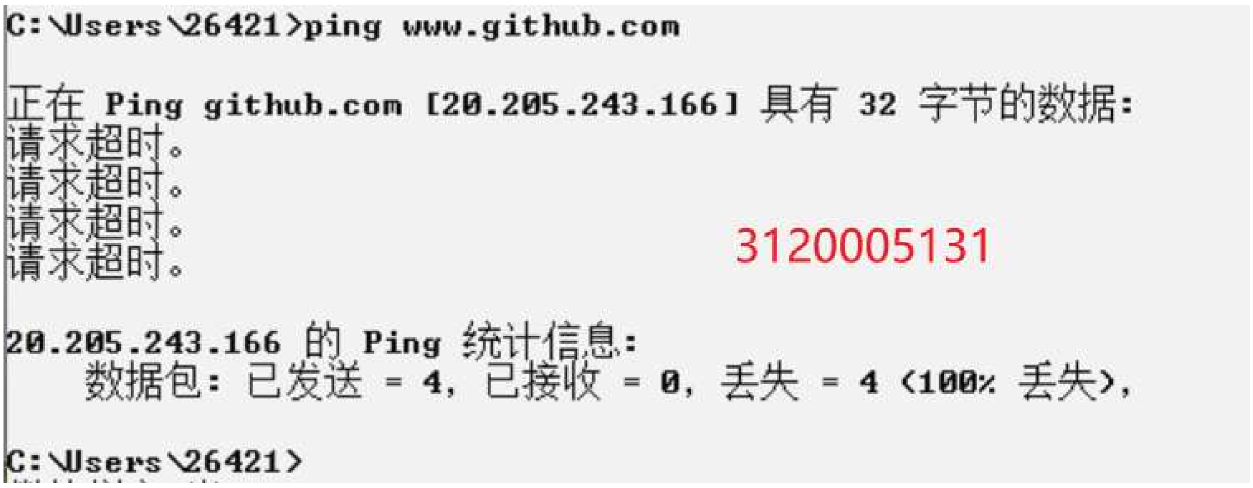
Capture filter for selected interfaces: 输入捕获过滤器 …

2.3.2、 点击开始后抓包开始，可以看到wireshark正处于抓包状态


*WLAN
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
应用显示过滤器 … <Ctrl-/>

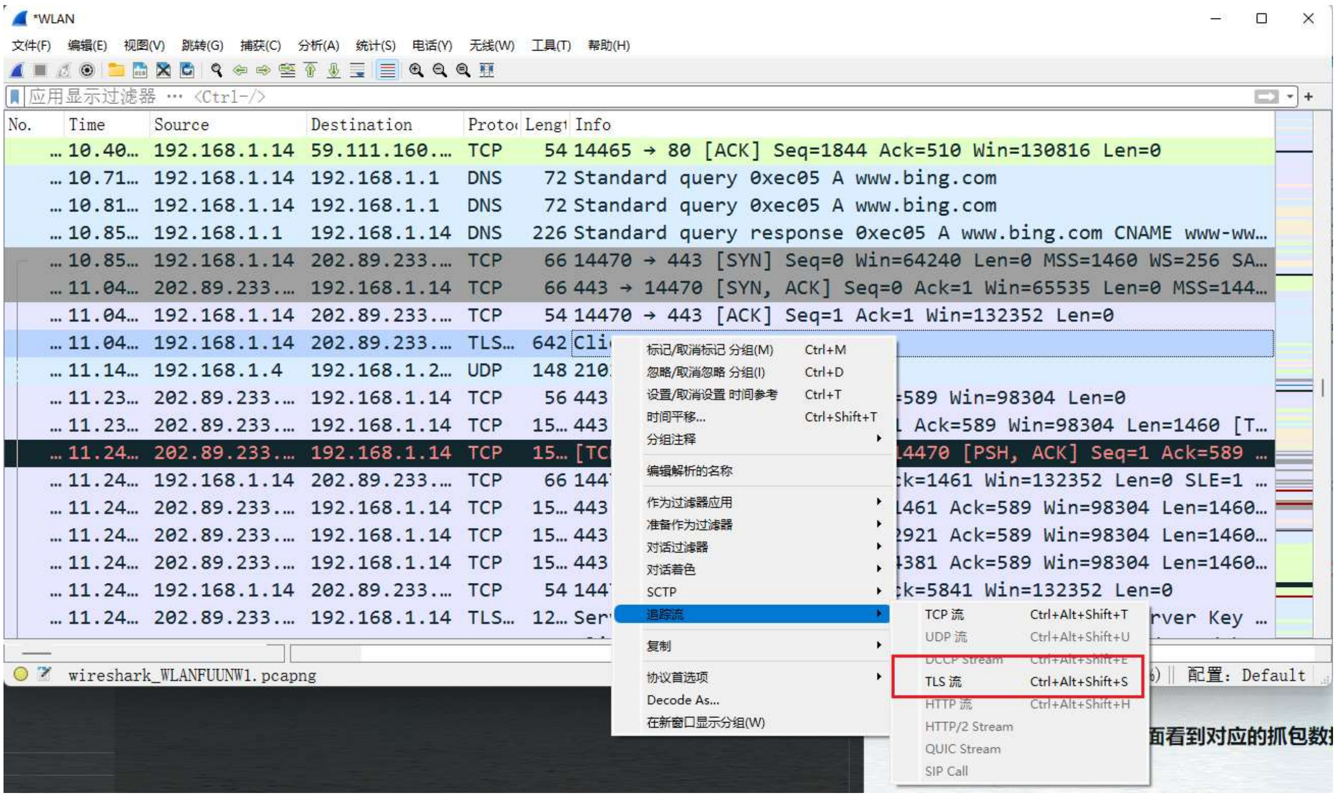
```
No.  Time      Source        Destination      Proto  Lengt Info
 … 25.79…  192.168.1.4   192.168.1.2…     UDP    148 2103 → 2103 Len=106
 … 25.90…  192.168.1.6   239.255.255…     SSDP   179 M-SEARCH * HTTP/1.1
 … 26.10…  192.168.1.6   224.0.0.252      LLM…    64 Standard query 0x188c A wpad
 … 26.10…  192.168.1.6   224.0.0.252      LLM…    64 Standard query 0x7915 A wpad
 … 26.20…  192.168.1.6   192.168.1.2…     NBNS    92 Name query NB DESKTOP-K4E079R<1c>
 … 26.41…  192.168.1.6   192.168.1.2…     NBNS    92 Name query NB WPAD<00>
 … 26.41…  192.168.1.6   192.168.1.2…     NBNS    92 Name query NB WPAD<00>
 … 26.61…  192.168.1.6   224.0.0.251      MDNS    70 Standard query 0x0000 A wpad.local, "QM" quest
 … 26.61…  192.168.1.6   224.0.0.251      MDNS    70 Standard query 0x0000 A wpad.local, "QM" quest
 … 27.12…  192.168.1.6   192.168.1.2…     NBNS    92 Name query NB WPAD<00>
```

2.3.3、执行需要抓包的操作，在cmd下输入ping www.github.com（这里以github为例子）



```
C:\Users\26421>ping www.github.com

正在 Ping github.com [20.205.243.166] 具有 32 字节的数据:
请求超时。
请求超时。
请求超时。
请求超时。

20.205.243.166 的 Ping 统计信息:
    数据包: 已发送 = 4，已接收 = 0，丢失 = 4 (100% 丢失)，

C:\Users\26421>
```

2.3.4、执行后可以在抓包界面看到对应的抓包数据，右键选择TSL跟踪流即可进行TLS分析

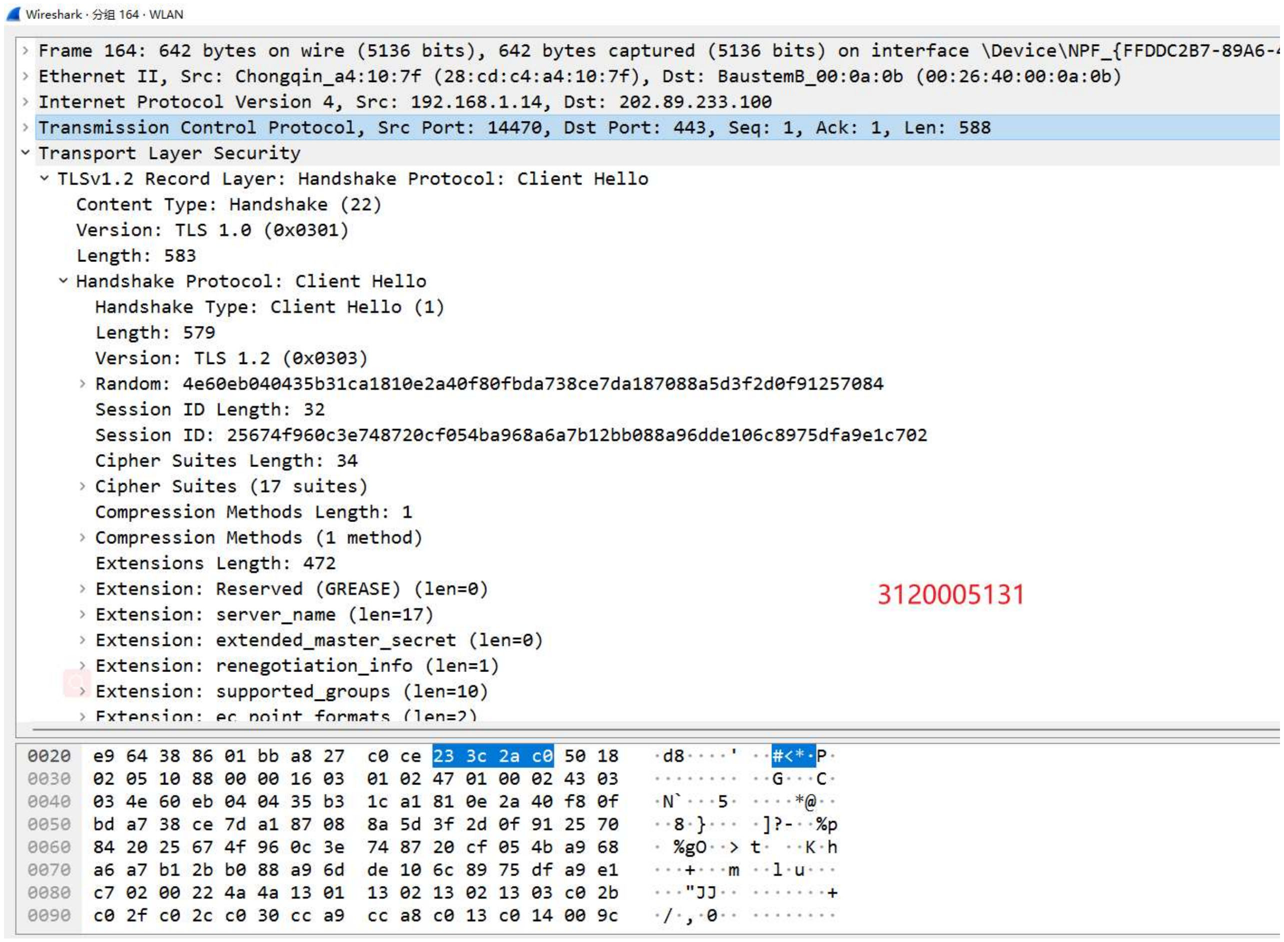


```
No.  Time     Source         Destination     Proto  Lengt Info
 … 10.40…  192.168.1.14  59.111.160.…    TCP     54 14465 → 80 [ACK] Seq=1844 Ack=510 Win=130816 Len=0
 … 10.71…  192.168.1.14  192.168.1.1     DNS     72 Standard query 0xec05 A www.bing.com
 … 10.81…  192.168.1.14  192.168.1.1     DNS     72 Standard query 0xec05 A www.bing.com
 … 10.85…  192.168.1.1   192.168.1.14    DNS    226 Standard query response 0xec05 A www.bing.com CNAME www-ww…
 … 10.85…  192.168.1.14  202.89.233.…    TCP     66 14470 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SA…
 … 11.04…  202.89.233.…  192.168.1.14    TCP     66 443 → 14470 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=144…
 … 11.04…  192.168.1.14  202.89.233.…    TCP     54 14470 → 443 [ACK] Seq=1 Ack=1 Win=132352 Len=0
 … 11.04…  192.168.1.14  202.89.233.…    TLS…   642 Cli
 … 11.14…  192.168.1.4   192.168.1.2…    UDP    148 210
 … 11.23…  202.89.233.…  192.168.1.14    TCP     56 443                       589 Win=98304 Len=0
 … 11.23…  202.89.233.…  192.168.1.14    TCP    15… 443                       l Ack=589 Win=98304 Len=1460 [T…
 … 11.24…  202.89.233.…  192.168.1.14    TCP    15… [TC                       14470 [PSH, ACK] Seq=1 Ack=589 …
 … 11.24…  192.168.1.14  202.89.233.…    TCP     66 144                       k=1461 Win=132352 Len=0 SLE=1 …
 … 11.24…  202.89.233.…  192.168.1.14    TCP    15… 443                       1461 Ack=589 Win=98304 Len=1460…
 … 11.24…  202.89.233.…  192.168.1.14    TCP    15… 443                       2921 Ack=589 Win=98304 Len=1460…
 … 11.24…  202.89.233.…  192.168.1.14    TCP    15… 443                       4381 Ack=589 Win=98304 Len=1460…
 … 11.24…  192.168.1.14  202.89.233.…    TCP     54 144                       k=5841 Win=132352 Len=0
 … 11.24…  202.89.233.…  192.168.1.14    TLS…   12… Ser                       rver Key …
```

2.4、SSL握手过程

2.4.1、第一个SSL握手是客户端向服务器发起的Client Hello消息--初始化阶段

- 1、可以看到第四层是TCP协议，源端口是随机的端口，目的端口是443
- 2、并且在TLS协议的第一行中写道：是一个Handshake握手协议，内容是Client Hello

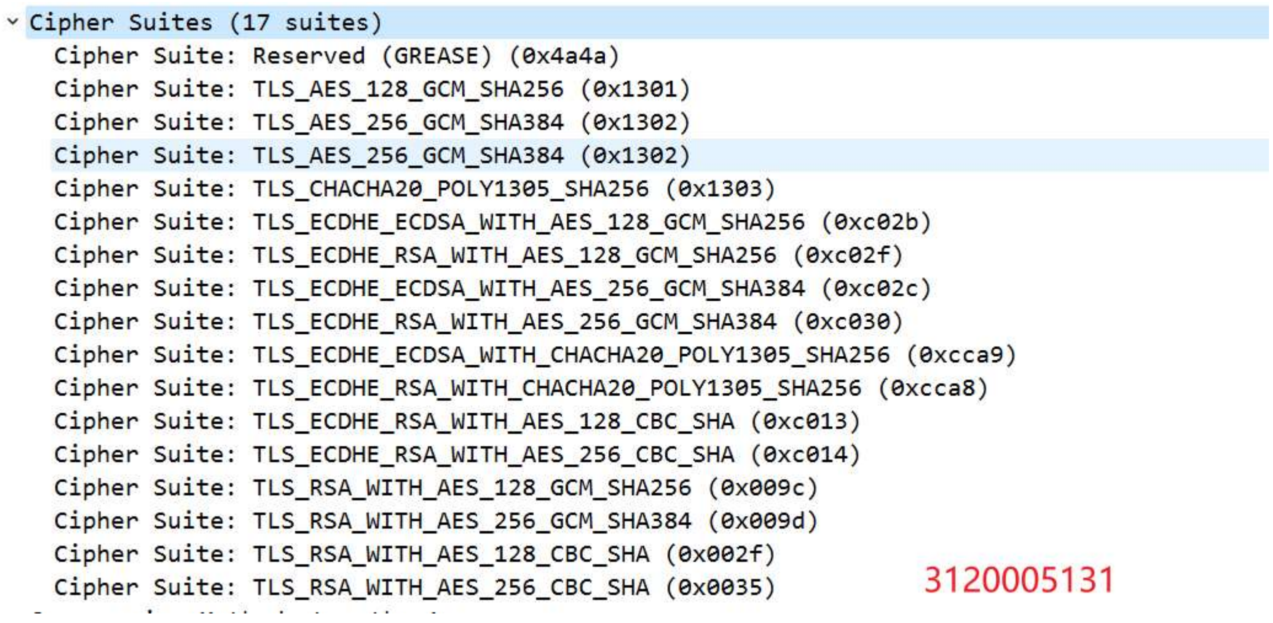


Wireshark · 分组 164 · WLAN

```
> Frame 164: 642 bytes on wire (5136 bits), 642 bytes captured (5136 bits) on interface \Device\NPF_{FFDDC2B7-89A6-
> Ethernet II, Src: Chongqin_a4:10:7f (28:cd:c4:a4:10:7f), Dst: BaustemB_00:0a:0b (00:26:40:00:0a:0b)
> Internet Protocol Version 4, Src: 192.168.1.14, Dst: 202.89.233.100
> Transmission Control Protocol, Src Port: 14470, Dst Port: 443, Seq: 1, Ack: 1, Len: 588
v Transport Layer Security
  v TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 583
    v Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 579
        Version: TLS 1.2 (0x0303)
      > Random: 4e60eb040435b31ca1810e2a40f80fbda738ce7da187088a5d3f2d0f91257084
        Session ID Length: 32
        Session ID: 25674f960c3e748720cf054ba968a6a7b12bb088a96dde106c8975dfa9e1c702
        Cipher Suites Length: 34
      > Cipher Suites (17 suites)
        Compression Methods Length: 1
      > Compression Methods (1 method)
        Extensions Length: 472
      > Extension: Reserved (GREASE) (len=0)
      > Extension: server_name (len=17)
      > Extension: extended_master_secret (len=0)
      > Extension: renegotiation_info (len=1)
      > Extension: supported_groups (len=10)
      > Extension: ec_point_formats (len=2)

0020  e9 64 38 86 01 bb a8 27  c0 ce 23 3c 2a c0 50 18   ·d8····'  ··#<*·P·
0030  02 05 10 88 00 00 16 03  01 02 47 01 00 02 43 03   ········  ··G···C·
0040  03 4e 60 eb 04 04 35 b3  1c a1 81 0e 2a 40 f8 0f   ·N`···5·  ····*@··
0050  bd a7 38 ce 7d a1 87 08  8a 5d 3f 2d 0f 91 25 70   ··8·}···  ·]?-··%p
0060  84 20 25 67 4f 96 0c 3e  74 87 20 cf 05 4b a9 68   · %gO··> t·  ··K·h
0070  a6 a7 b1 2b b0 88 a9 6d  de 10 6c 89 75 df a9 e1   ···+···m  ··l·u···
0080  c7 02 00 22 4a 4a 13 01  13 02 13 02 13 03 c0 2b   ···"JJ··  ·······+
0090  c0 2f c0 2c c0 30 cc a9  cc a8 c0 13 c0 14 00 9c   ·/·,·0··  ········
```

3120005131

- 3、也可以看到TLS协议所支持的密文族



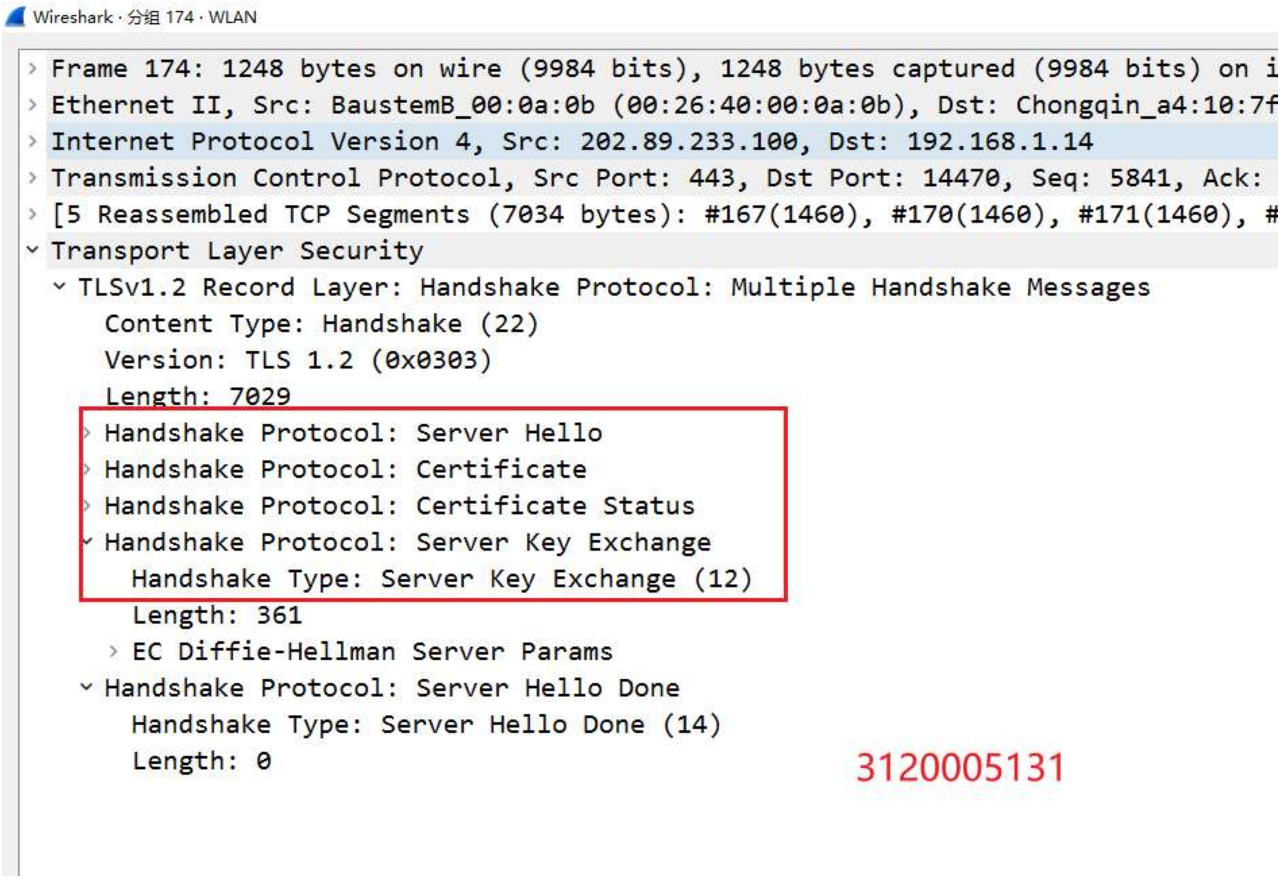
```
v Cipher Suites (17 suites)
    Cipher Suite: Reserved (GREASE) (0x4a4a)
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
    Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
    Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
    Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
```

3120005131

2.4.2、 服务器回复--认证阶段

- 1、首先是回复客户端：Server Hello

- 2、之后是发送证书给客户端



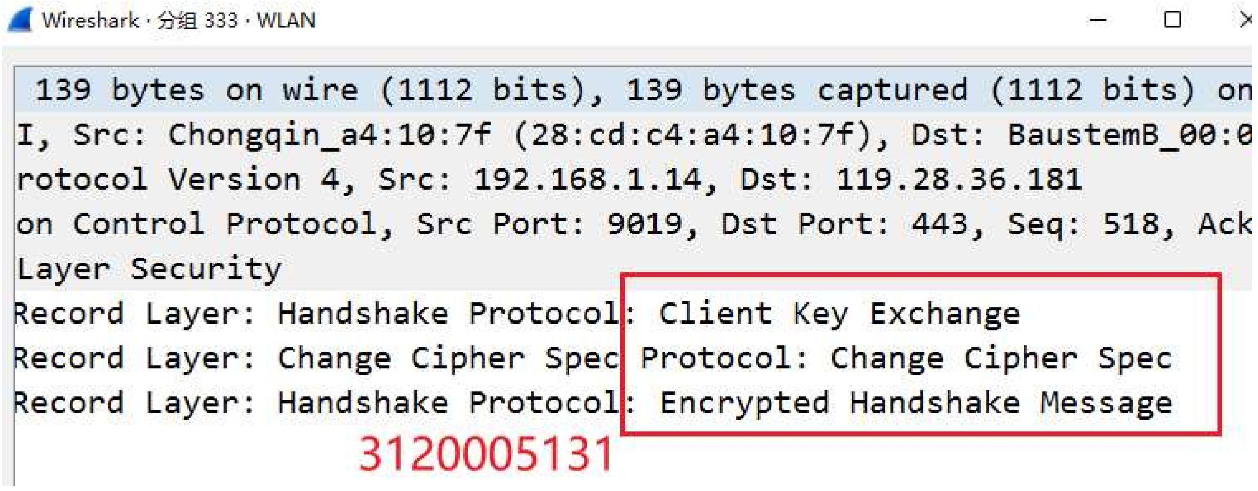
Wireshark · 分组 174 · WLAN

```
> Frame 174: 1248 bytes on wire (9984 bits), 1248 bytes captured (9984 bits) on i
> Ethernet II, Src: BaustemB_00:0a:0b (00:26:40:00:0a:0b), Dst: Chongqin_a4:10:7f
> Internet Protocol Version 4, Src: 202.89.233.100, Dst: 192.168.1.14
> Transmission Control Protocol, Src Port: 443, Dst Port: 14470, Seq: 5841, Ack:
> [5 Reassembled TCP Segments (7034 bytes): #167(1460), #170(1460), #171(1460), #
v Transport Layer Security
  v TLSv1.2 Record Layer: Handshake Protocol: Multiple Handshake Messages
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 7029
    > Handshake Protocol: Server Hello
    > Handshake Protocol: Certificate
    > Handshake Protocol: Certificate Status
    v Handshake Protocol: Server Key Exchange
        Handshake Type: Server Key Exchange (12)
        Length: 361
      > EC Diffie-Hellman Server Params
    v Handshake Protocol: Server Hello Done
        Handshake Type: Server Hello Done (14)
        Length: 0
```

3120005131

2.4.3、客户端验证--密钥协商阶段

- 1、客户端收到服务器发来的证书，并加以验证，验证完毕后生成一个本地的随机密码，并且把密码发给服务器，并且开始加密后续的握手信息



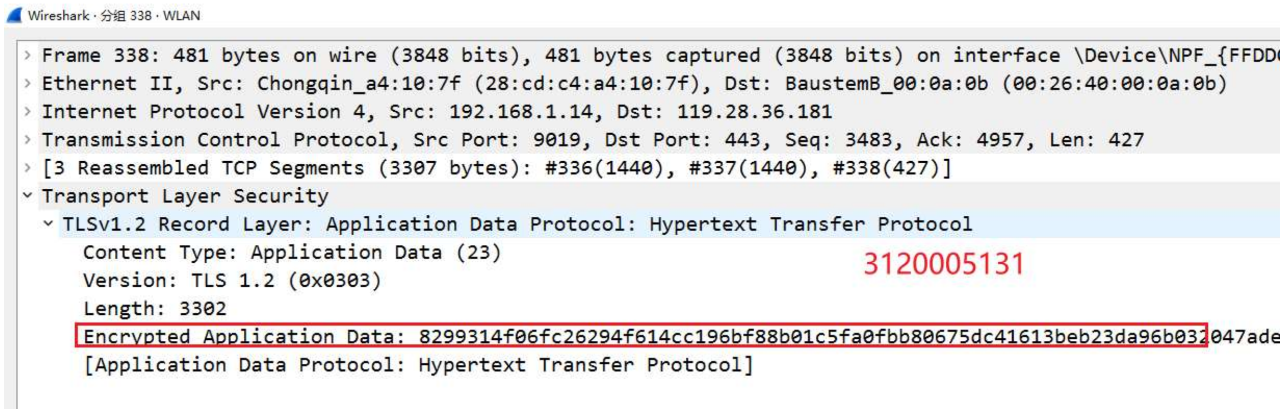
Wireshark · 分组 333 · WLAN

```
 139 bytes on wire (1112 bits), 139 bytes captured (1112 bits) on
I, Src: Chongqin_a4:10:7f (28:cd:c4:a4:10:7f), Dst: BaustemB_00:0
rotocol Version 4, Src: 192.168.1.14, Dst: 119.28.36.181
on Control Protocol, Src Port: 9019, Dst Port: 443, Seq: 518, Ack
Layer Security
Record Layer: Handshake Protocol: Client Key Exchange
Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
Record Layer: Handshake Protocol: Encrypted Handshake Message
```

3120005131

2.4.4、握手结束

- 握手结束后，后面就以协商好的密钥开始加密并发送HTTP数据包了



```
Wireshark · 分组 338 · WLAN
> Frame 338: 481 bytes on wire (3848 bits), 481 bytes captured (3848 bits) on interface \Device\NPF_{FFDD
> Ethernet II, Src: Chongqin_a4:10:7f (28:cd:c4:a4:10:7f), Dst: BaustemB_00:0a:0b (00:26:40:00:0a:0b)
> Internet Protocol Version 4, Src: 192.168.1.14, Dst: 119.28.36.181
> Transmission Control Protocol, Src Port: 9019, Dst Port: 443, Seq: 3483, Ack: 4957, Len: 427
> [3 Reassembled TCP Segments (3307 bytes): #336(1440), #337(1440), #338(427)]
v Transport Layer Security
  v TLSv1.2 Record Layer: Application Data Protocol: Hypertext Transfer Protocol
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 3302
      Encrypted Application Data: 8299314f06fc26294f614cc196bf88b01c5fa0fbb80675dc41613beb23da96b032047ade
      [Application Data Protocol: Hypertext Transfer Protocol]
```

3、实验小结

- 1、进一步熟悉对wireshark的使用和运用，更加了解了其丰富的功能
- 2、通过实验加深对SSL/TLS协议的印象，知道在其中运用了非对称和对称加密算法，了解HTTPS背后的协议内涵
- 3、更加明白数据交互过程中对于安全性的操作和重要性

IPsec实验

1、实验内容

配置并抓包分析 IPsec协议

2、实验过程

2.1、 软件的安装

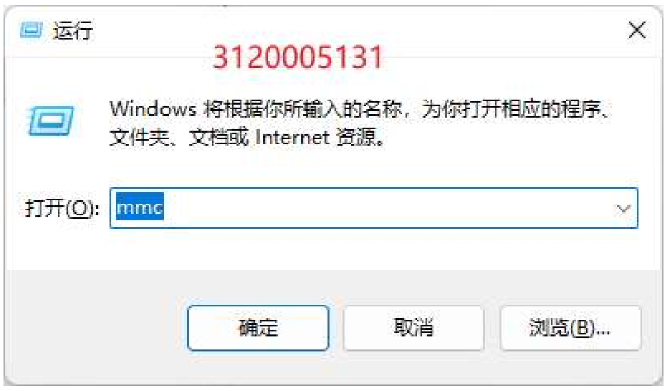
用到的软件为wireshark和RawCap抓包软件

2.2、IPSec的设置

2.2.1、实验目标 使客户端不能够用TCP协议访问服务器的8888端口

2.2.2、创建IPSec规则

- 通过本地mmc指令进入本地安全设置

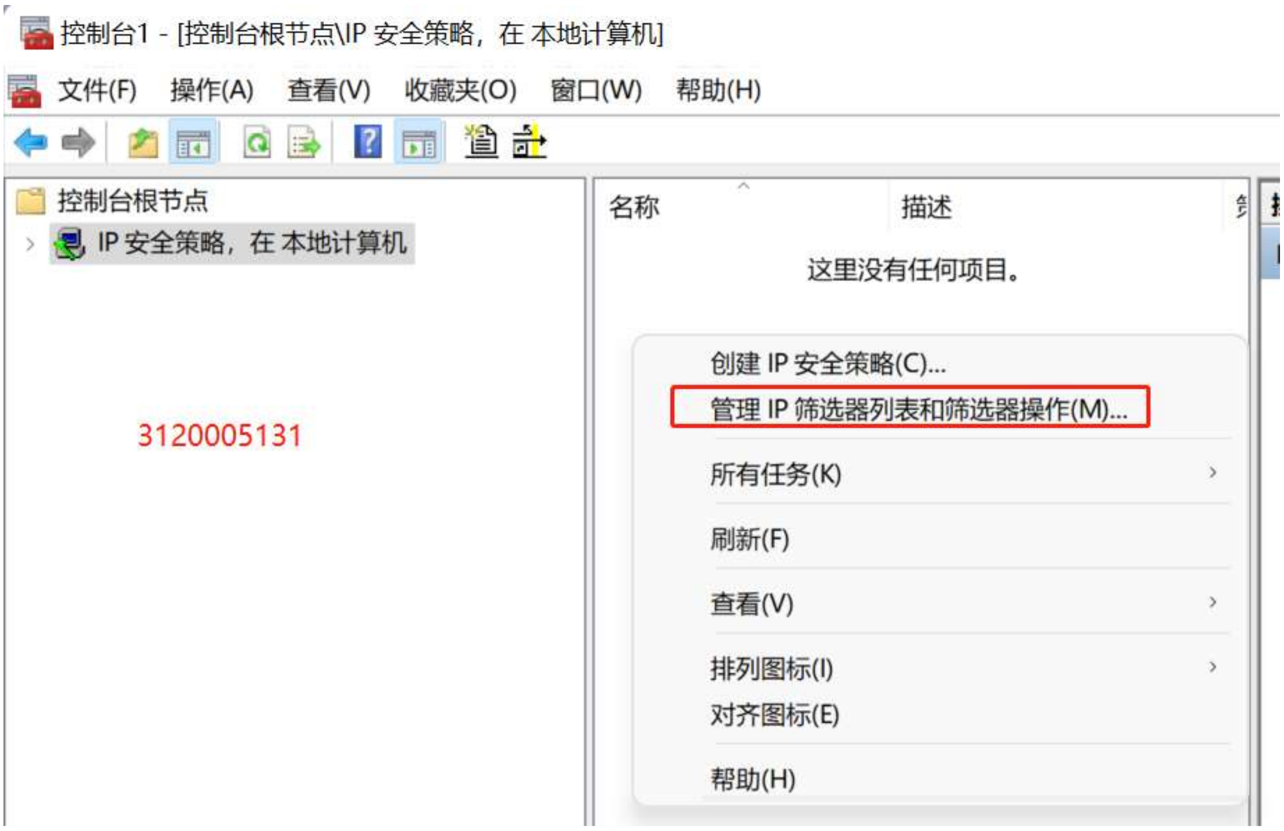


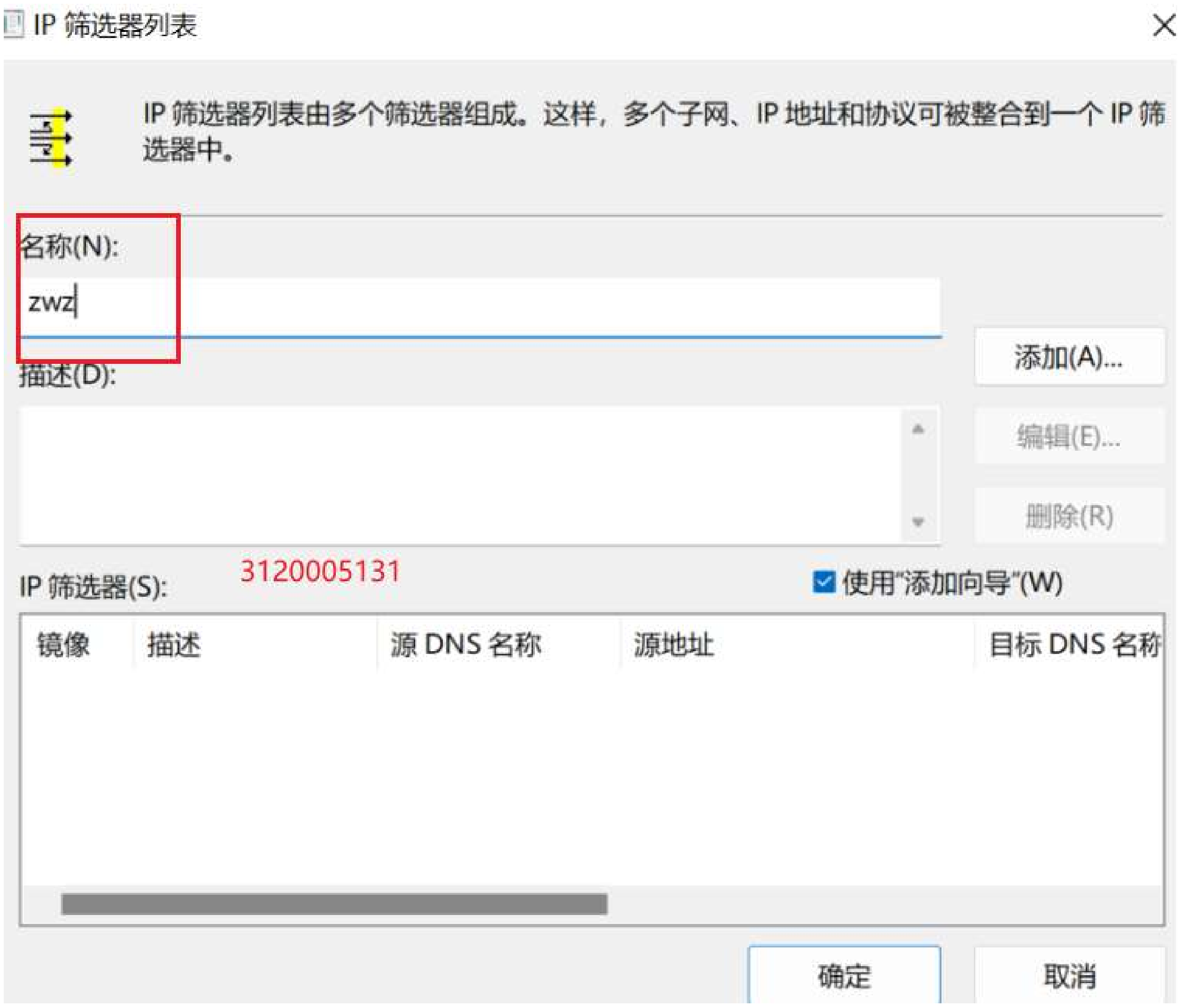
运行

Windows 将根据你所输入的名称，为你打开相应的程序、文件夹、文档或 Internet 资源。

打开(O): mmc

- 进入管理IP筛选器后添加，输入名字

- 
控制台1 - [控制台根节点\IP 安全策略，在 本地计算机]
文件(F) 操作(A) 查看(V) 收藏夹(O) 窗口(W) 帮助(H)
控制台根节点
IP 安全策略，在 本地计算机
3120005131
名称
描述
这里没有任何项目。
创建 IP 安全策略(C)...
管理 IP 筛选器列表和筛选器操作(M)...
所有任务(K)
刷新(F)
查看(V)
排列图标(I)
对齐图标(E)
帮助(H)



IP 筛选器列表由多个筛选器组成。这样，多个子网、IP 地址和协议可被整合到一个 IP 筛选器中。

名称(N): zwz

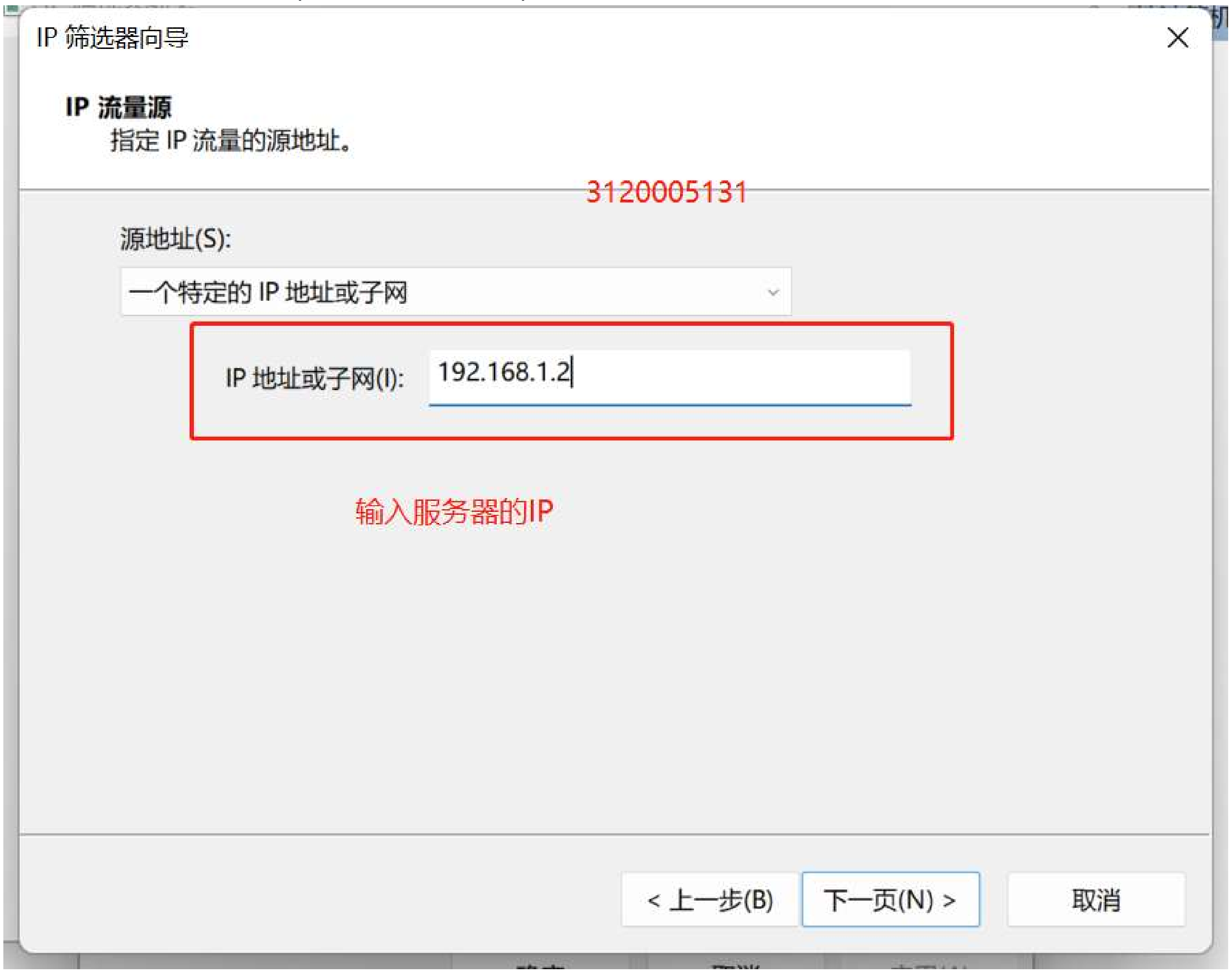
描述(D):

IP 筛选器(S):

☑ 使用"添加向导"(W)

镜像	描述	源 DNS 名称	源地址	目标 DNS 名称

- 源地址输入服务器IP地址（已改为192.168.1.13）



IP 筛选器向导

IP 流量源

指定 IP 流量的源地址。

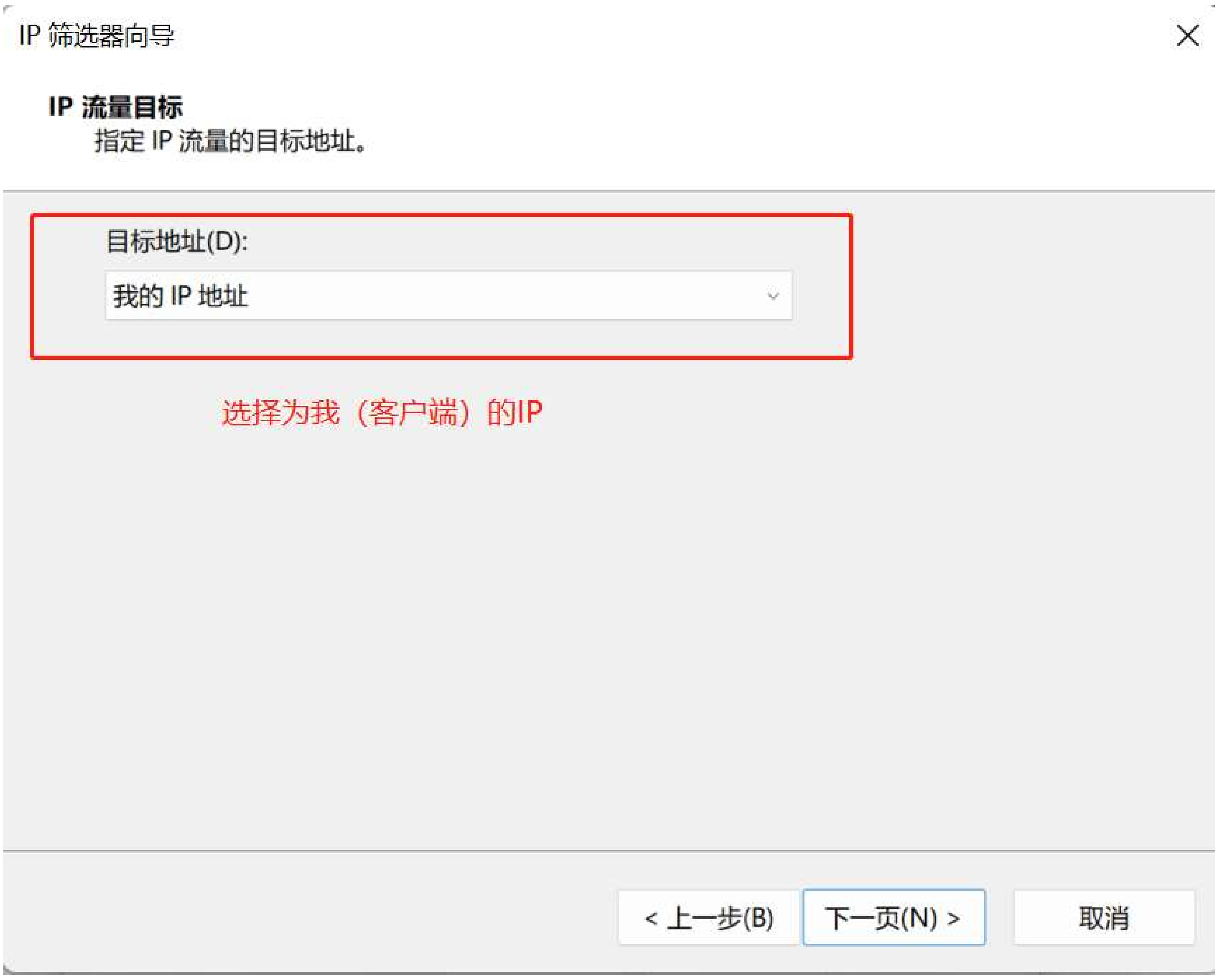
3120005131

源地址(S): 一个特定的 IP 地址或子网

IP 地址或子网(I): 192.168.1.2

输入服务器的IP

- 目的地址设为本身的IP地址

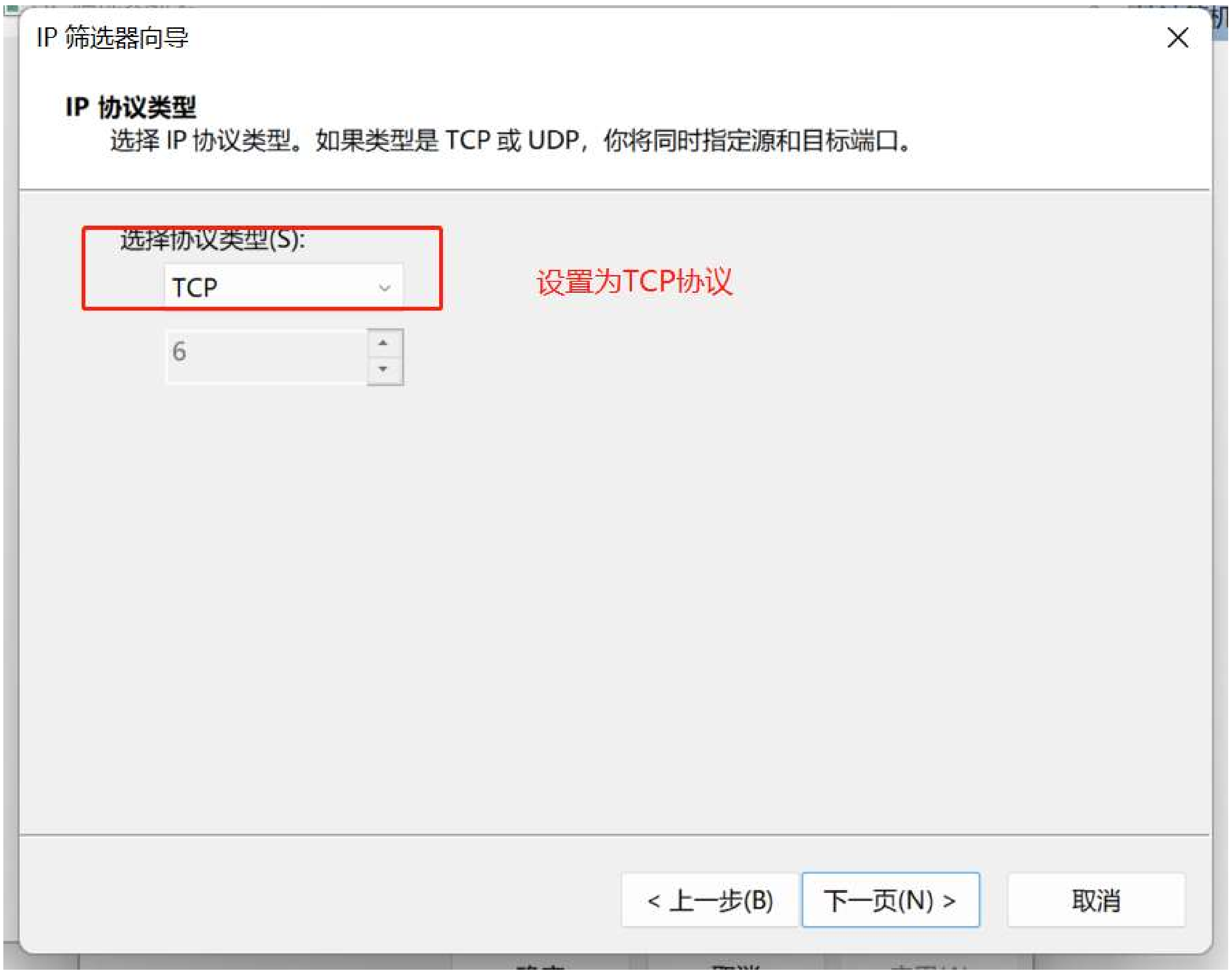

IP 筛选器向导

IP 流量目标
指定 IP 流量的目标地址。

目标地址(D):
我的 IP 地址

选择为我（客户端）的IP

- IP协议类型设置为TCP类型



IP 筛选器向导

IP 协议类型

选择 IP 协议类型。如果类型是 TCP 或 UDP，你将同时指定源和目标端口。

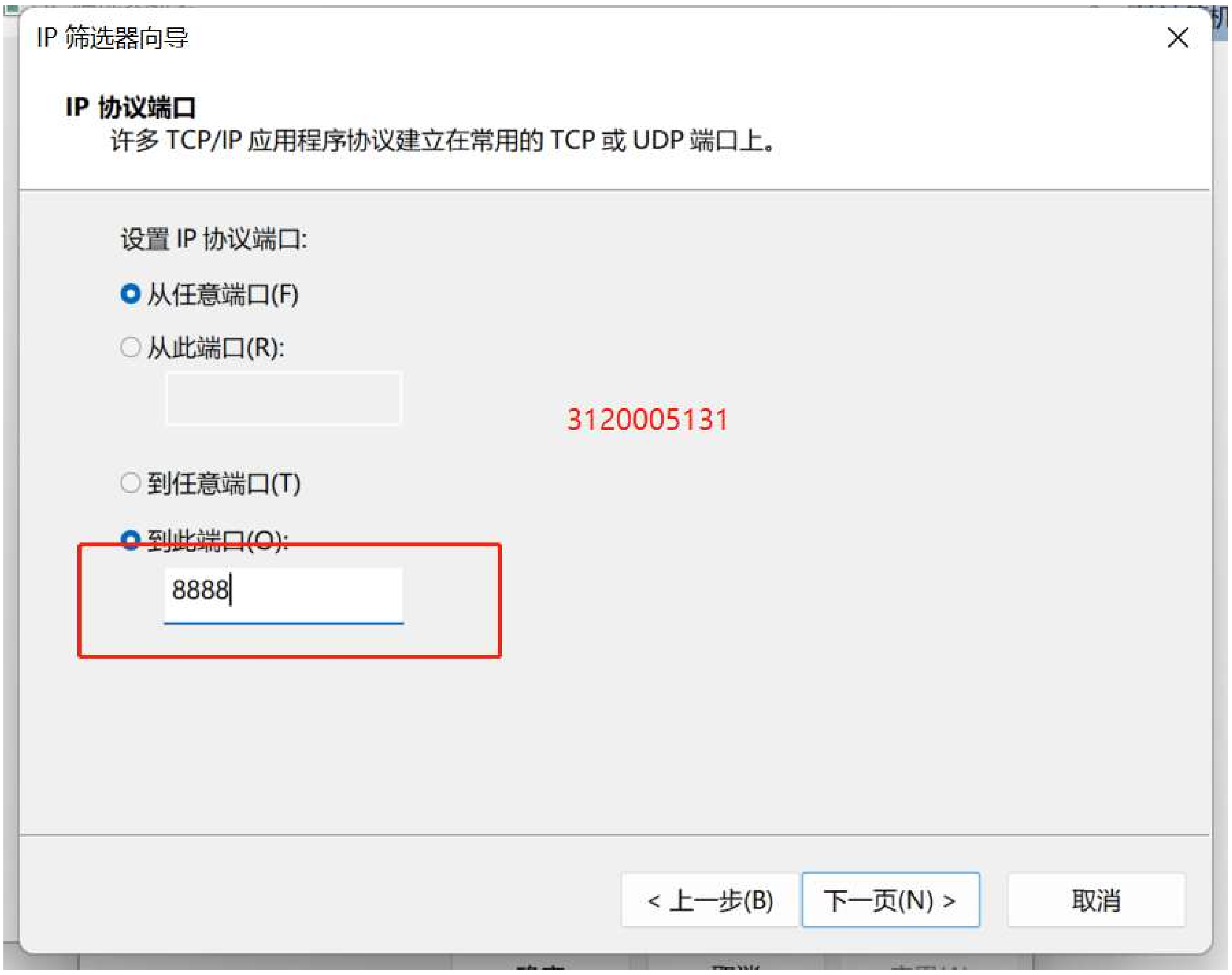
选择协议类型(S):

TCP

6

设置为TCP协议

- 设置从任意端口到8888端口



IP 筛选器向导

IP 协议端口

许多 TCP/IP 应用程序协议建立在常用的 TCP 或 UDP 端口上。

设置 IP 协议端口:

[✓ 从任意端口(F)]

从此端口(R):

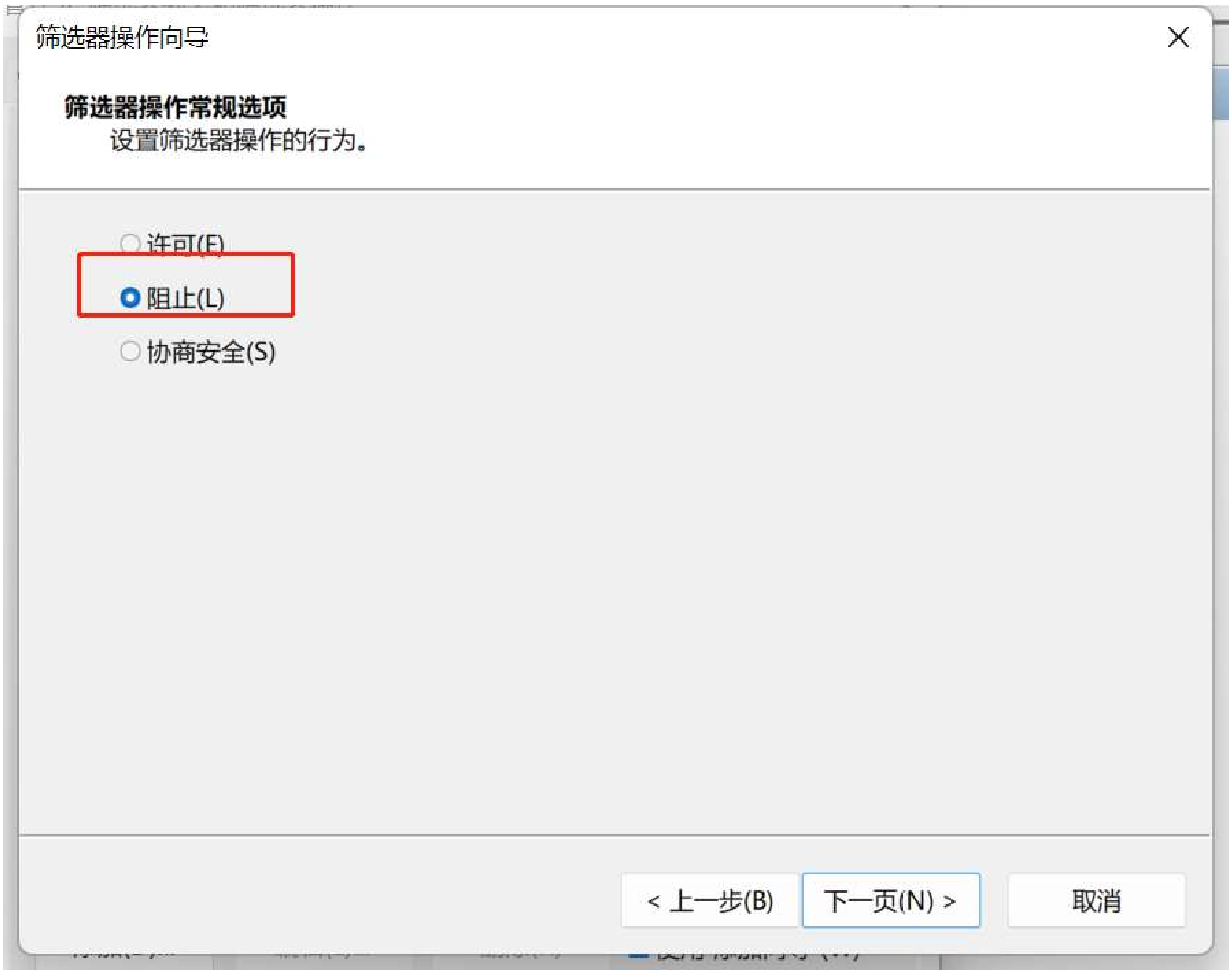
3120005131

到任意端口(T)

[✓ 到此端口(O):]

8888

- 切换到管理筛选器操作，因为创建了IP筛选器，但还需要设置是禁止还是允许



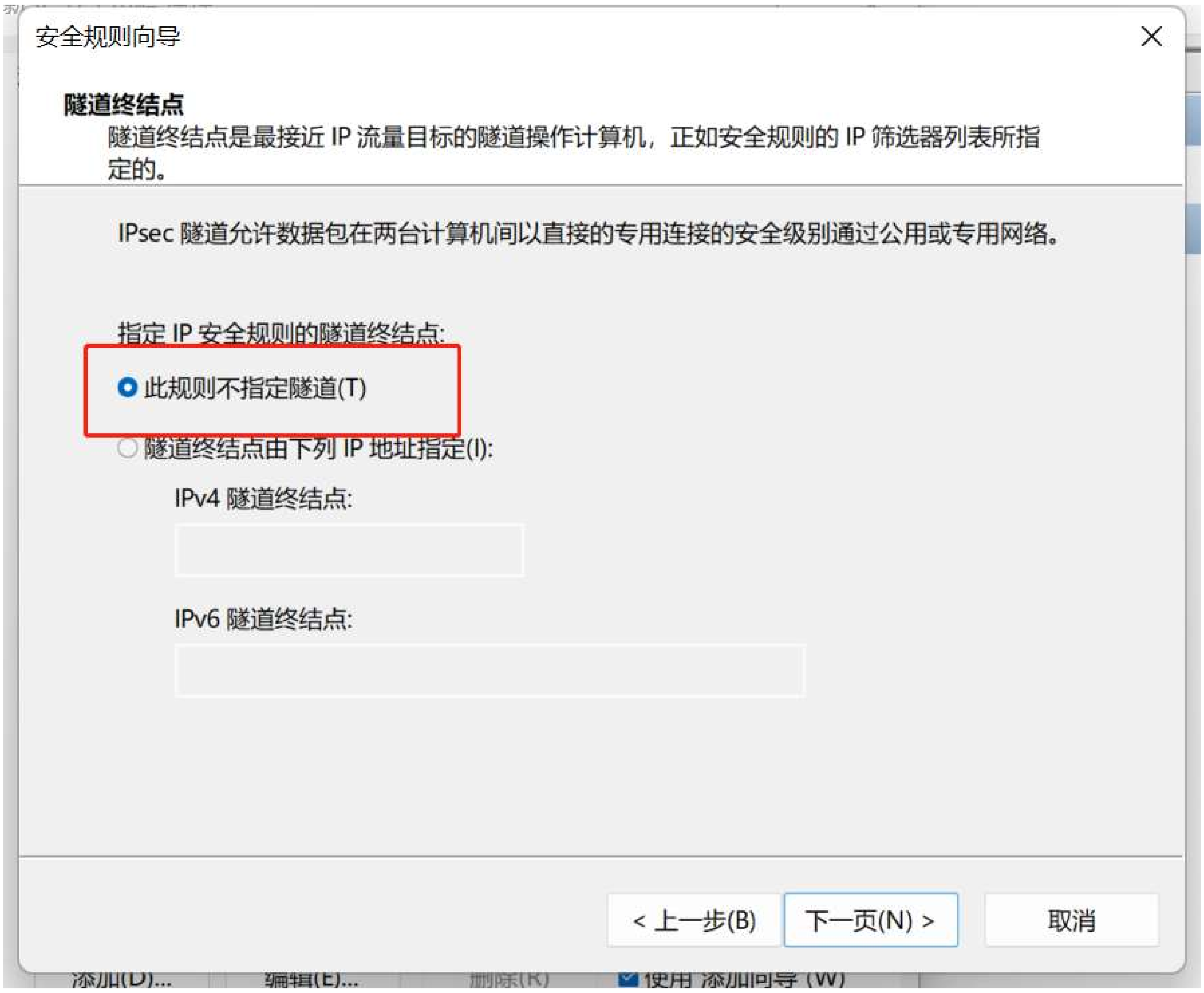
筛选器操作向导

筛选器操作常规选项
设置筛选器操作的行为。

许可(E) [✓ 阻止(L)] 协商安全(S)

- 右键创建IP安全策略 Alt
- 右键属性进入刚刚创建的IP安全策略，添加

- 选择不指定隧道



安全规则向导

隧道终结点

隧道终结点是最接近 IP 流量目标的隧道操作计算机，正如安全规则的 IP 筛选器列表所指定的。

IPsec 隧道允许数据包在两台计算机间以直接的专用连接的安全级别通过公用或专用网络。

指定 IP 安全规则的隧道终结点:

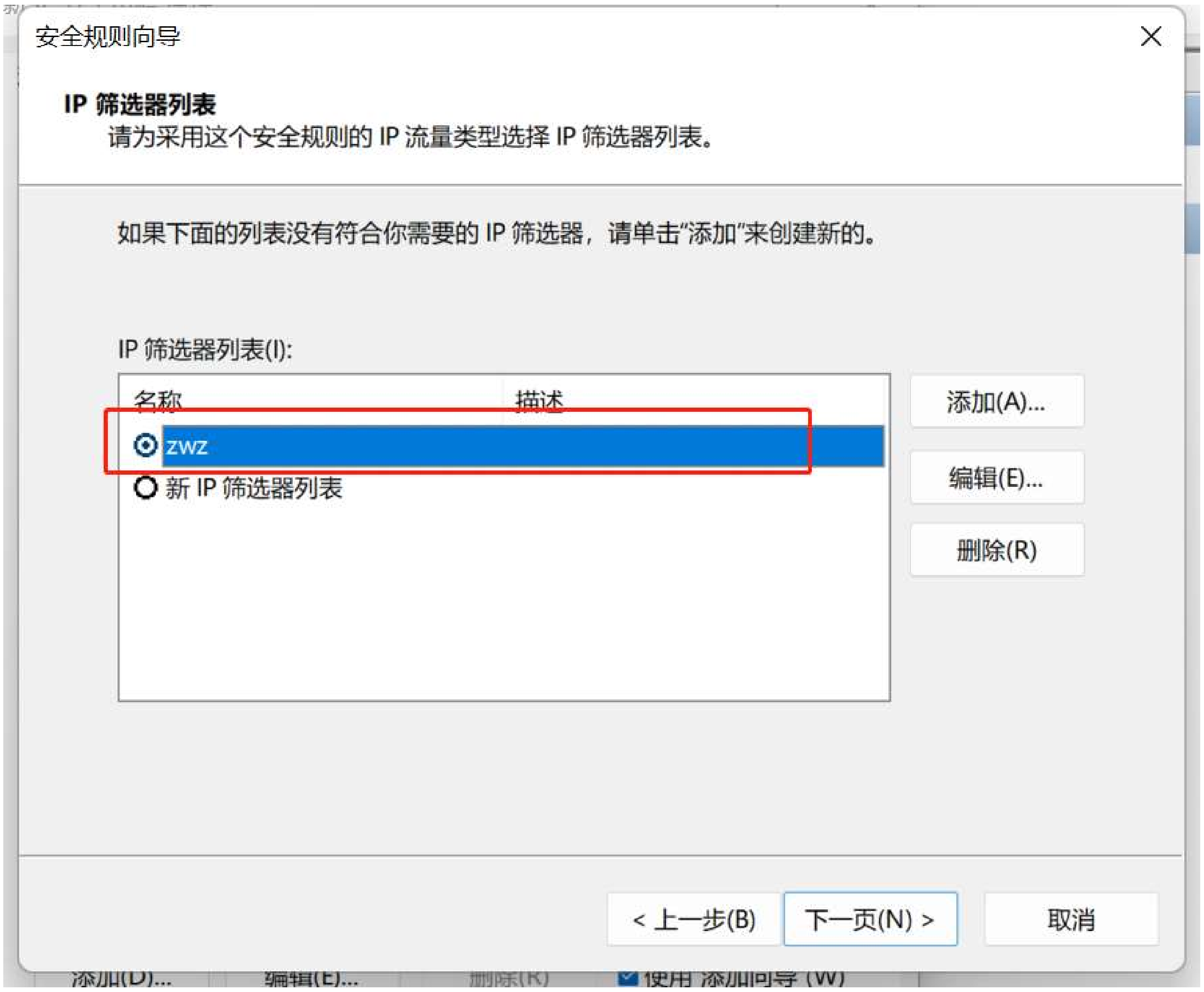
[✓ 此规则不指定隧道(T)]

隧道终结点由下列 IP 地址指定(I):

IPv4 隧道终结点:

IPv6 隧道终结点:

- 选择新建的筛选表zwz

- 

安全规则向导

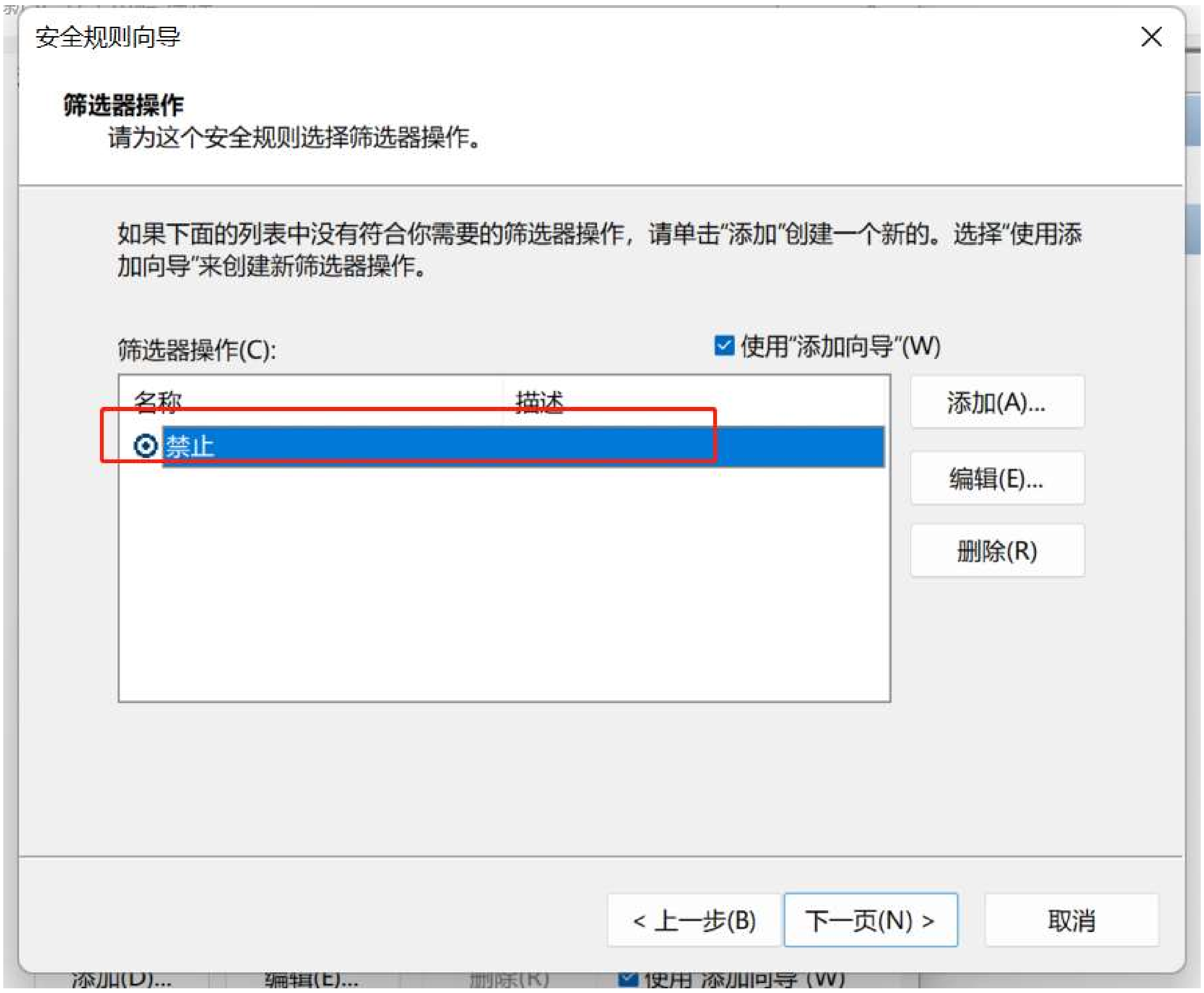
IP 筛选器列表
请为采用这个安全规则的 IP 流量类型选择 IP 筛选器列表。

如果下面的列表没有符合你需要的 IP 筛选器，请单击"添加"来创建新的。

IP 筛选器列表(I):

名称	描述
◉ zwz	
○ 新 IP 筛选器列表	

- 选择筛选器操作


安全规则向导

筛选器操作
请为这个安全规则选择筛选器操作。

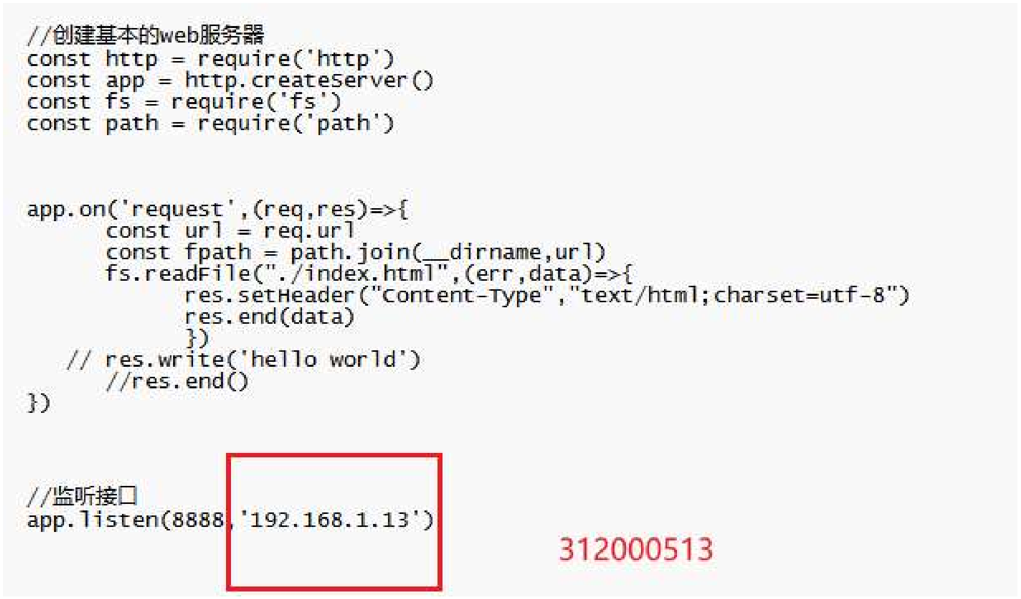
如果下面的列表中没有符合你需要的筛选器操作，请单击"添加"创建一个新的。选择"使用添加向导"来创建新筛选器操作。

筛选器操作(C):　　☑ 使用"添加向导"(W)

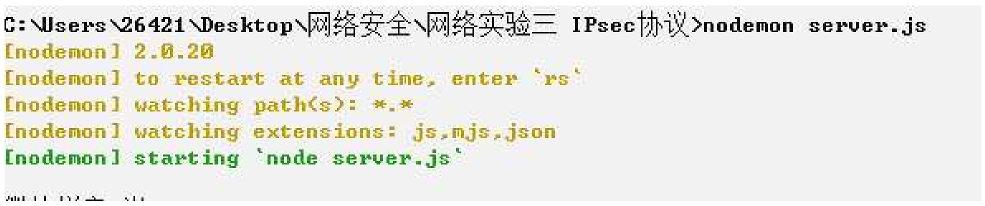
名称	描述
⊙ 禁止	

2.2.3、服务器配置

- 这里我们使用nodejs开启一个本地的服务器，代码如下

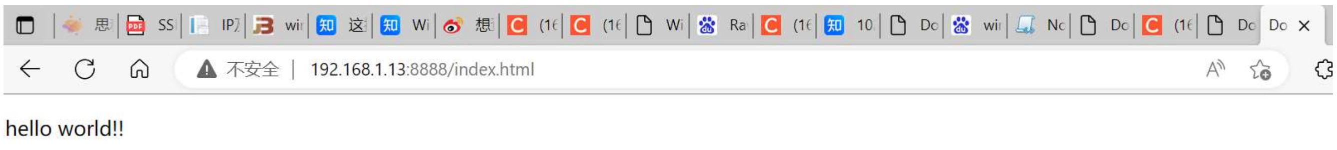

```
//创建基本的web服务器
const http = require('http')
const app = http.createServer()
const fs = require('fs')
const path = require('path')


app.on('request',(req,res)=>{
      const url = req.url
      const fpath = path.join(__dirname,url)
      fs.readFile("./index.html",(err,data)=>{
            res.setHeader("Content-Type","text/html;charset=utf-8")
            res.end(data)
            })
   // res.write('hello world')
      //res.end()
})


//监听接口
app.listen(8888,'192.168.1.13')
```

- 并且使用nodemon启动


```
C:\Users\26421\Desktop\网络安全\网络实验三 IPsec协议>nodemon server.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
```

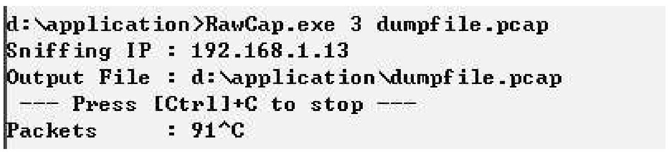
2.3 筛选器启动前抓包服务器

2.3.1、网页访问



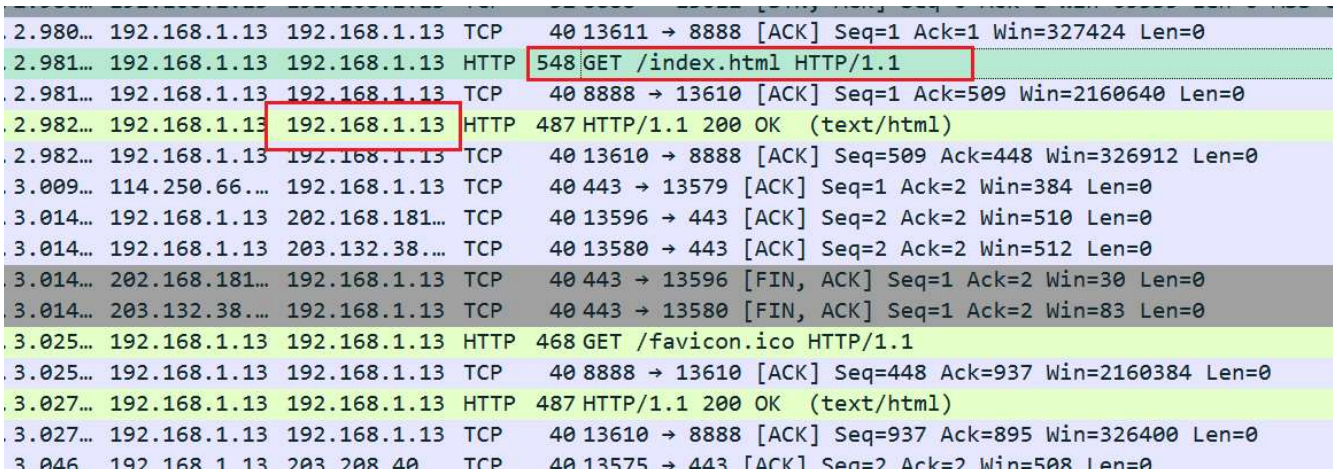
hello world!!

RawCap抓包获取



```
d:\application>RawCap.exe 3 dumpfile.pcap
Sniffing IP : 192.168.1.13
Output File : d:\application\dumpfile.pcap
 --- Press [Ctrl]+C to stop ---
Packets      : 91^C
```

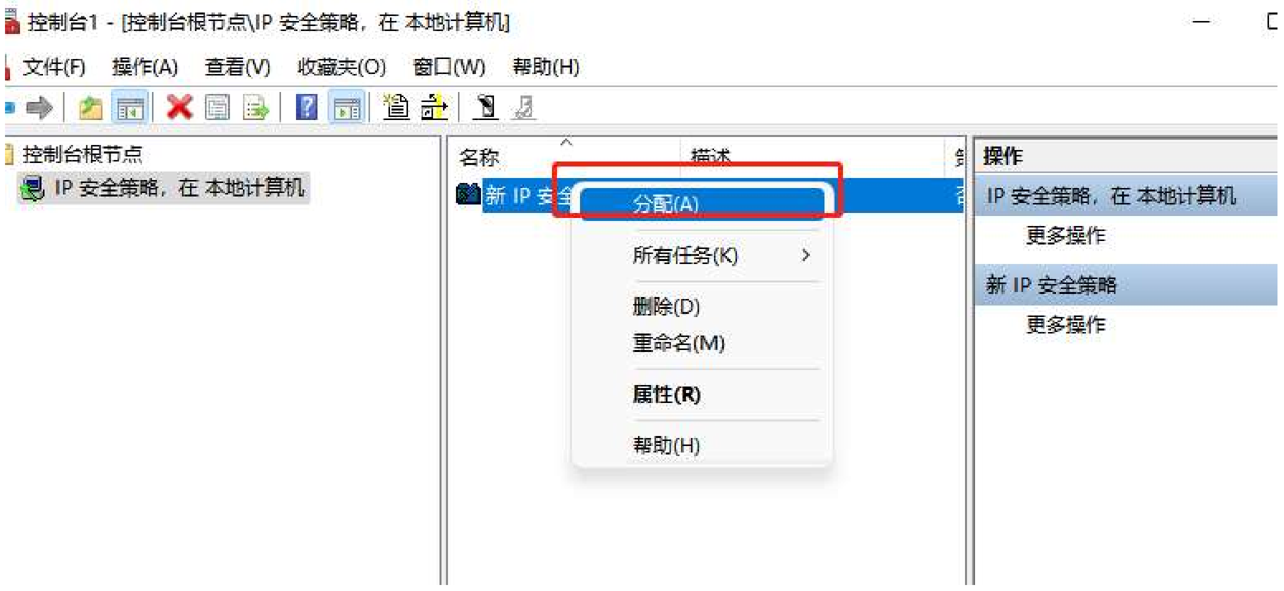
将生成的文件在wireshark打开，可以看到是成功的



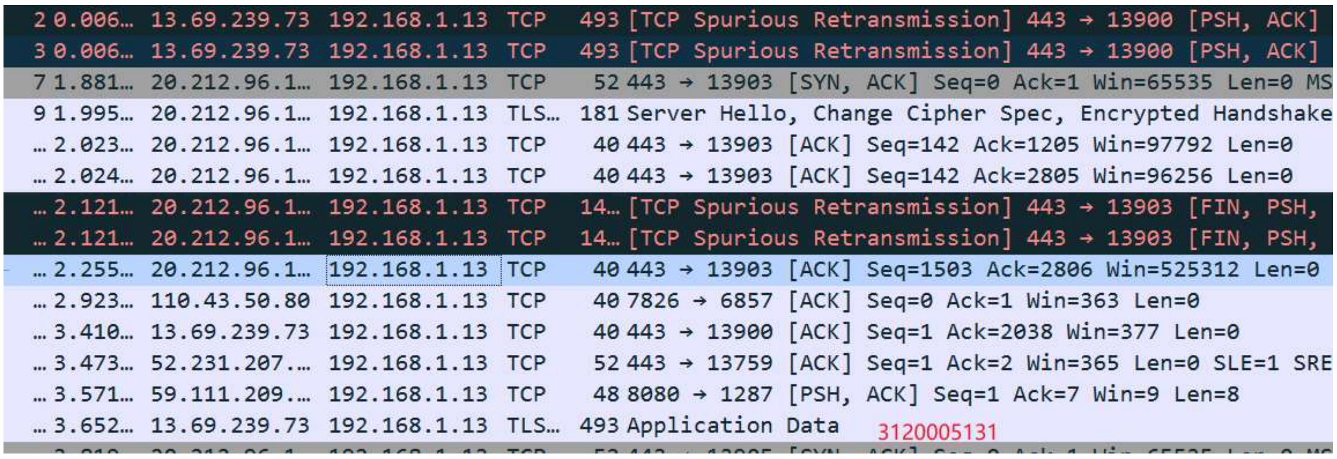
```
.2.980… 192.168.1.13 192.168.1.13 TCP    40 13611 → 8888 [ACK] Seq=1 Ack=1 Win=327424 Len=0
.2.981… 192.168.1.13 192.168.1.13 HTTP  548 GET /index.html HTTP/1.1
.2.981… 192.168.1.13 192.168.1.13 TCP    40 8888 → 13610 [ACK] Seq=1 Ack=509 Win=2160640 Len=0
.2.982… 192.168.1.13 192.168.1.13 HTTP  487 HTTP/1.1 200 OK  (text/html)
.2.982… 192.168.1.13 192.168.1.13 TCP    40 13610 → 8888 [ACK] Seq=509 Ack=448 Win=326912 Len=0
.3.009… 114.250.66.… 192.168.1.13 TCP    40 443 → 13579 [ACK] Seq=1 Ack=2 Win=384 Len=0
.3.014… 192.168.1.13 202.168.181… TCP    40 13596 → 443 [ACK] Seq=2 Ack=2 Win=510 Len=0
.3.014… 192.168.1.13 203.132.38.… TCP    40 13580 → 443 [ACK] Seq=2 Ack=2 Win=512 Len=0
.3.014… 202.168.181… 192.168.1.13 TCP    40 443 → 13596 [FIN, ACK] Seq=1 Ack=2 Win=30 Len=0
.3.014… 203.132.38.… 192.168.1.13 TCP    40 443 → 13580 [FIN, ACK] Seq=1 Ack=2 Win=83 Len=0
.3.025… 192.168.1.13 192.168.1.13 HTTP  468 GET /favicon.ico HTTP/1.1
.3.025… 192.168.1.13 192.168.1.13 TCP    40 8888 → 13610 [ACK] Seq=448 Ack=937 Win=2160384 Len=0
.3.027… 192.168.1.13 192.168.1.13 HTTP  487 HTTP/1.1 200 OK  (text/html)
.3.027… 192.168.1.13 192.168.1.13 TCP    40 13610 → 8888 [ACK] Seq=937 Ack=895 Win=326400 Len=0
.3.046… 192.168.1.13 203.208.40… TCP    40 13575 → 443 [ACK] Seq=2 Ack=2 Win=508 Len=0
```

2.4、打开IPSec筛选器

2.4、开始分配



2.4.3、重复2.3步骤，再次打开抓包文件，发现已找不到刚刚的http包



```
2 0.006… 13.69.239.73 192.168.1.13 TCP  493 [TCP Spurious Retransmission] 443 → 13900 [PSH, ACK]
3 0.006… 13.69.239.73 192.168.1.13 TCP  493 [TCP Spurious Retransmission] 443 → 13900 [PSH, ACK]
7 1.881… 20.212.96.1… 192.168.1.13 TCP   52 443 → 13903 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MS
9 1.995… 20.212.96.1… 192.168.1.13 TLS… 181 Server Hello, Change Cipher Spec, Encrypted Handshake
… 2.023… 20.212.96.1… 192.168.1.13 TCP   40 443 → 13903 [ACK] Seq=142 Ack=1205 Win=97792 Len=0
… 2.024… 20.212.96.1… 192.168.1.13 TCP   40 443 → 13903 [ACK] Seq=142 Ack=2805 Win=96256 Len=0
… 2.121… 20.212.96.1… 192.168.1.13 TCP  14… [TCP Spurious Retransmission] 443 → 13903 [FIN, PSH,
… 2.121… 20.212.96.1… 192.168.1.13 TCP  14… [TCP Spurious Retransmission] 443 → 13903 [FIN, PSH,
… 2.255… 20.212.96.1… 192.168.1.13 TCP   40 443 → 13903 [ACK] Seq=1503 Ack=2806 Win=525312 Len=0
… 2.923… 110.43.50.80 192.168.1.13 TCP   40 7826 → 6857 [ACK] Seq=0 Ack=1 Win=363 Len=0
… 3.410… 13.69.239.73 192.168.1.13 TCP   40 443 → 13900 [ACK] Seq=1 Ack=2038 Win=377 Len=0
… 3.473… 52.231.207.… 192.168.1.13 TCP   52 443 → 13759 [ACK] Seq=1 Ack=2 Win=365 Len=0 SLE=1 SRE
… 3.571… 59.111.209.… 192.168.1.13 TCP   48 8080 → 1287 [PSH, ACK] Seq=1 Ack=7 Win=9 Len=8
… 3.652… 13.69.239.73 192.168.1.13 TLS… 493 Application Data
```

3、实验总结

- 1、首先了解到IPsec是一组开放的网络安全协议。它并不是一个单独的协议，而是一系列为IP网络提供安全性的协议和服务的集合。
- 2、明白其中的工作原理：

> 当两台计算机相互连接的时候，它们之间的数据包会通过TCP/IP协议传送出去。但是TCP/IP协议并不是绝对安全的：一方面因为TCP/IP协议本身并不安全；另一方面因为一些攻击者可以通过伪造的数据包进行欺骗式攻击。 针对上述情况IPSEC技术应运而生：利用一种特殊的加密方法将数据包的源地址与目的地址分开来进行加密并封装成一个个独立的分组。 然后由这些分组构成一个虚拟的网络接口，再由这个虚拟的网络接口去连接其他设备或服务器等资源以实现数据传输的目的。

- 3、学会在电脑上配置筛选器，从而实现IPSec协议功能，并能够通过抓包软件进行验证

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

● JavaScript 62.7% ● HTML 37.3%