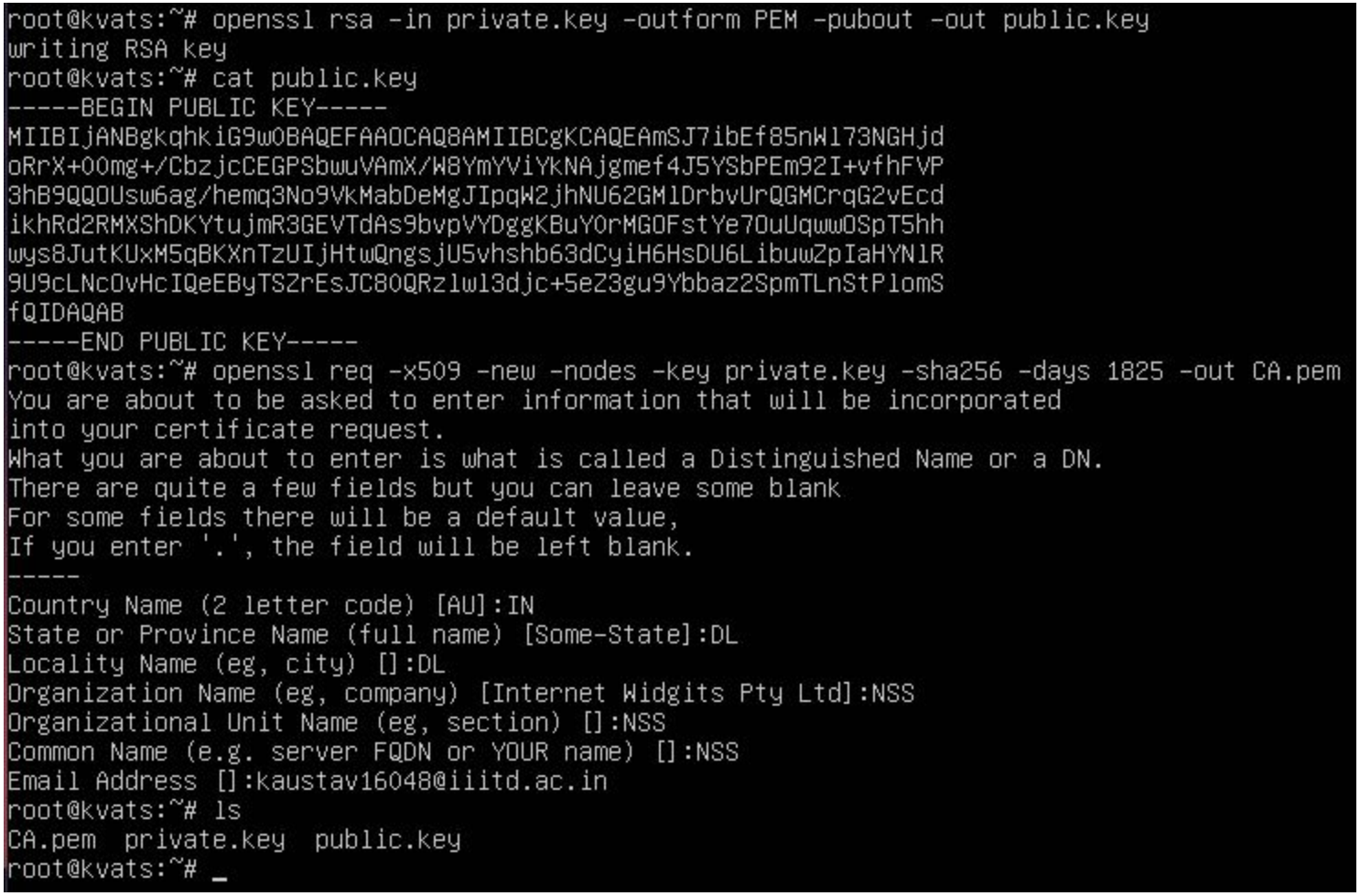
NSS Lab 5
Report

By Sushant and Kaustav

A1. Here we have created CA Private Key using openssl genrsa command of 2048 bits.



A1. Created public key of the CA using using first command.

Using 2nd command we created CA's Certificate, which will be later used to sign server and client certificate.

In Last line you can see that we have created all required items(CA certificate, private and public key of CA).

```
root@kvats:~/Server# openssl genrsa -out server.key 2048
```

```
root@kvats:~/Server# openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be incorporat
into your certificate request.
What you are about to enter is what is called a Distinguished Name or
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.
```

A1. Similarly we created server key and server certificate.

In the last screenshot we signed Server Certificate using CA private key and CA Certificate.

```
root@kvats:~/Server# openssl x509 -req -in server.csr -CA ../CA.pem -CAkey ../private.key -CAcreates
erial -out MyServer.pem -days 2048 -sha256
Signature ok
subject=C = IN, ST = Delhi, L = Ohkla, O = Network, OU = System, CN = Security, emailAddress = kaust
avvats@gmail.com
Getting CA Private Key
root@kvats:~/Server# ls
MyServer.pem  server.csr  server.key
root@kvats:~/Server# openssl x509 -text -noout -in MyServer.pem
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            ac:c8:0f:6b:61:af:47:f2
    Signature Algorithm: sha256WithRSAEncryption
```

```
root@kvats:~/Client# openssl genrsa -out Client.key 2048
Generating RSA private key, 2048 bit long modulus
......+++
.........................+++
e is 65537 (0x010001)
root@kvats:~/Client# openssl req -new -key Client.key -out Client.csr
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]:IN
State or Province Name (full name) [Some-State]:Delhi
Locality Name (eg, city) []:DL
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SNN
Organizational Unit Name (eg, section) []:SNN
Common Name (e.g. server FQDN or YOUR name) []:SNN
Email Address []:sushant@iiitd.ac.in

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:sushant
An optional company name []:
root@kvats:~/Client# ls
Client.csr  Client.key
```

A1. Using openssl command we created Clients private key and public key.
Then we created Client certificate.

In the last screenshot we signed Clients Certificate using CA private key and CA Certificate.

```
root@kvats:~/Client# openssl x509 -req -in Client.csr -CA ../CA.pem -CAkey ../private.key -CAcreates
erial -out MyClient.pem -days 2048 -sha256
Signature ok
subject=C = IN, ST = Delhi, L = DL, O = SNN, OU = SNN, CN = SNN, emailAddress = sushant@iiitd.ac.in
Getting CA Private Key
root@kvats:~/Client# ls
Client.csr  Client.key  MyClient.pem
root@kvats:~/Client#
```

```
ruby@ruby-rose:~/Client$ openssl s_client -connect 20.0.0.6:1234 -CAfile CA.pem

CONNECTED(00000003)
```

```
root@kvats:~/Server# openssl s_server -key server.key -cert MyServer.pem -CAfile ../CA.pem -accept 1
234
Using default temp DH parameters
ACCEPT
```

```
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID: 5E0BF4AEDD42D7E88BA2ECFC2988D6840EDF81CDA25B1BCCF3CB208AC92BCF7
4
    Session-ID-ctx:
    Master-Key: 8B0AAC3780C1EEA8EF86C1669EC6802FFCB9153A1365CF1B2A07C434E74DBC3
C9491ADC8409D6835AB12F84006D88939
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 7200 (seconds)
    TLS session ticket:
    0000 - 42 4f 3e de 8c e0 a4 07-69 56 00 d4 93 9b f7 29   BO>.....iV.....)
    0010 - d6 13 79 9f 38 95 ff 51-0b e3 de 65 5f a8 74 6d   ..y.8..Q...e_.tm
    0020 - e1 a9 4d 49 9b 92 14 39-2d 15 60 5e 33 30 63 a4   ..MI...9-.`^30c.
    0030 - 50 35 ce 6a 3e 04 c5 2d-f8 4d 85 d2 2a 85 96 ed   P5.j>..-.M..*...
    0040 - 40 6b 2e 02 f7 bc 83 38-da 35 68 81 85 52 ef 17   @k.....8.5h..R..
    0050 - c0 83 03 fc 50 65 cc 90-73 9f 79 e5 d7 db e8 da   ....Pe..s.y.....
    0060 - 3a 6a c7 e4 85 57 3c f1-ae 41 79 2a 5a 57 ed 7b   :j...W<..Ay*ZW.{
    0070 - df 51 2f 2e 0c 50 89 dc-d6 ee 8d a2 7c 6a 8b 90   .Q/..P......|j..
    0080 - 21 c9 94 07 a8 08 8b 9b-57 f7 49 69 dd 95 e9 08   !.......W.Ii....
    0090 - c7 29 c8 a5 c8 51 25 b8-d1 a7 49 7d 87 65 74 75   .)...Q%...I}.etu

    Start Time: 1553682600
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
    Extended master secret: yes
```

GUI version - Client, username - Ruby
Linux Server version - Server, username - Kvats

A2. We created a server using openssl s_server which listens on port 1234 and presents CA signed certificate to any incoming connection. (screenshot 2)

In screenshot 1- Client tries to connect to the server using openssl s_client command and specified certificate of the CA it trust.

A3. In the last screenshot, we can clearly see that TLS connection has started and server is verified, return code is 0 (OK). Client successfully authenticates the server.

```
ruby@ruby-rose:~/Client$ openssl s_client -connect 20.0.0.6:1234 -CAfile CA.pem

CONNECTED(00000003)
depth=1 C = IN, ST = DL, L = DL, O = NSS, OU = NSS, CN = NSS, emailAddress = ka
ustav16048@iiitd.ac.in
verify return:1
depth=0 C = IN, ST = Delhi, L = Ohkla, O = Network, OU = System, CN = Security,
 emailAddress = kaustavvats@gmail.com
verify return:1
---
Certificate chain
 0 s:/C=IN/ST=Delhi/L=Ohkla/O=Network/OU=System/CN=Security/emailAddress=kausta
vvats@gmail.com
   i:/C=IN/ST=DL/L=DL/O=NSS/OU=NSS/CN=NSS/emailAddress=kaustav16048@iiitd.ac.in
 1 s:/C=IN/ST=DL/L=DL/O=NSS/OU=NSS/CN=NSS/emailAddress=kaustav16048@iiitd.ac.in
   i:/C=IN/ST=DL/L=DL/O=NSS/OU=NSS/CN=NSS/emailAddress=kaustav16048@iiitd.ac.in
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDgDCCAmgCCQCsyA9rYa9H8jANBgkqhkiG9w0BAQsFADB6MQswCQYDVQQGEwJJ
```

```
Acceptable client certificate CA names
/C=IN/ST=DL/L=DL/O=NSS/OU=NSS/CN=NSS/emailAddress=kaustav16048@iiitd.ac.in
Client Certificate Types: RSA sign, DSA sign, ECDSA sign
Requested Signature Algorithms: RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+SHA384:D
SA+SHA384:ECDSA+SHA384:RSA+SHA256:DSA+SHA256:ECDSA+SHA256:RSA+SHA224:DSA+SHA224
:ECDSA+SHA224:RSA+SHA1:DSA+SHA1:ECDSA+SHA1
Shared Requested Signature Algorithms: RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+S
HA384:DSA+SHA384:ECDSA+SHA384:RSA+SHA256:DSA+SHA256:ECDSA+SHA256:RSA+SHA224:DSA
+SHA224:ECDSA+SHA224:RSA+SHA1:DSA+SHA1:ECDSA+SHA1
Peer signing digest: SHA512
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 2675 bytes and written 281 bytes
Verification: OK
---
New, TLSv1.2, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID: 6F963EEB2BA045D186605D38F0508EBF934ED82B3E77022697D4D4FF5B5DB16
3
    Session-ID-ctx:
    Master-Key: E75C2687DD09CB297C5B4C87525F5754936DB0BC39F17A9519294D50AB6F1C5
```

A4. For this question report this and next slide.

Here we have attached output of client connecting to server. Client verifies the certificate of the server

In the second screen shot, we can see that the server also verifies client and gave a list of acceptable client certificate CA names to client. After Server Verification OK, new TLS connection is established.

Next slide contains screenshot of server side.

```
root@kvats:~/Server# openssl s_server -key server.key -cert MyServer.pem -CAfile ../CA.pem -accept 1
234 -verify 1
verify depth is 1
Using default temp DH parameters
ACCEPT
-----BEGIN SSL SESSION PARAMETERS-----
MFoCAQECAgMDBALAMAQABDDnXCaH3QnLKXxbTIdSX1dUk22wvDnxepUZKU1Qq28c
XGLhSslQzUN4/ixrkOQGr/OhBgIEXKIkQ6IEAgIcIKQGBAQBAAAArQMCAQE=
-----END SSL SESSION PARAMETERS-----
Shared ciphers:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:E
CDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES12
8-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-R
SA-AES256-SHA384:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:DHE-RSA-AES
128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA:ECD
HE-RSA-AES128-SHA:DHE-RSA-AES128-SHA:AES256-GCM-SHA384:AES128-GCM-SHA256:AES256-SHA256:AES128-SHA256
:AES256-SHA:AES128-SHA
Signature Algorithms: RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+SHA384:DSA+SHA384:ECDSA+SHA384:RSA+SHA2
56:DSA+SHA256:ECDSA+SHA256:RSA+SHA224:DSA+SHA224:ECDSA+SHA224:RSA+SHA1:DSA+SHA1:ECDSA+SHA1
Shared Signature Algorithms: RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+SHA384:DSA+SHA384:ECDSA+SHA384:R
SA+SHA256:DSA+SHA256:ECDSA+SHA256:RSA+SHA224:DSA+SHA224:ECDSA+SHA224:RSA+SHA1:DSA+SHA1:ECDSA+SHA1
Supported Elliptic Curve Point Formats: uncompressed:ansiX962_compressed_prime:ansiX962_compressed_c
har2
Supported Elliptic Curves: X25519:P-256:P-521:P-384
Shared Elliptic curves: X25519:P-256:P-521:P-384
CIPHER is ECDHE-RSA-AES256-GCM-SHA384
Secure Renegotiation IS supported
```

Server listen at port 1234 and verifies every incoming connection. It also specified the accepted CA list as shown in previous slide.

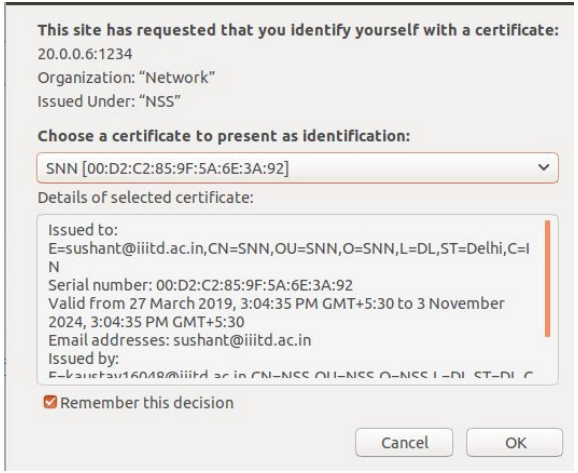
Command used is shown the screenshot


```
root@kvats:~/Server# openssl s_server -key server.key -cert MyServer.pem -CAfile ../CA.pem -accept 1
234 -verify 1
verify depth is 1
Using default temp DH parameters
ACCEPT
```

```
ruby@ruby-rose:~/Client$ openssl s_client -connect 20.0.0.6:1234 -CAfile CA.pem
 -cert MyClient.pem -key Client.key
CONNECTED(00000003)
depth=1 C = IN, ST = DL, L = DL, O = NSS, OU = NSS, CN = NSS, emailAddress = ka
ustav16048@iiitd.ac.in
verify return:1
depth=0 C = IN, ST = Delhi, L = Ohkla, O = Network, OU = System, CN = Security,
 emailAddress = kaustavvats@gmail.com
verify return:1
---
```

```
n
verify return:1
depth=0 C = IN, ST = Delhi, L = DL, O = SNN, OU = SNN, CN = SNN, emailAddress = sushant@iiitd.ac.in
verify return:1
ACCEPT
ACCEPT
ACCEPT
```

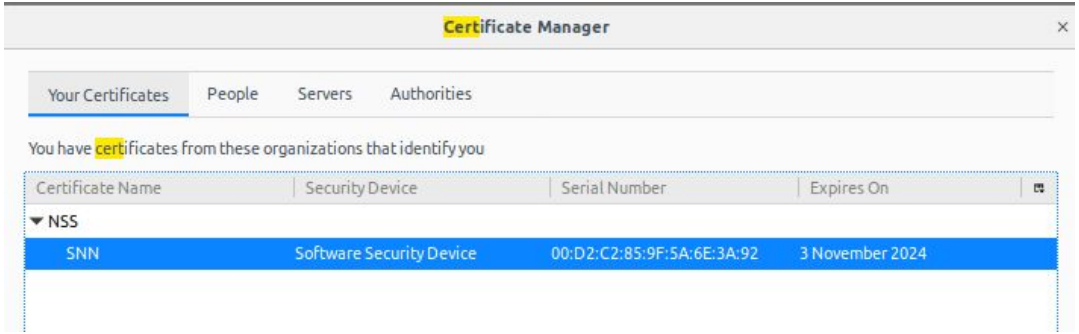
```
ruby@ruby-rose:~/Client$ openssl pkcs12 -export -in MyClient.pem -inkey Client.
key -out client.p12
Enter Export Password:
Verifying - Enter Export Password:
```



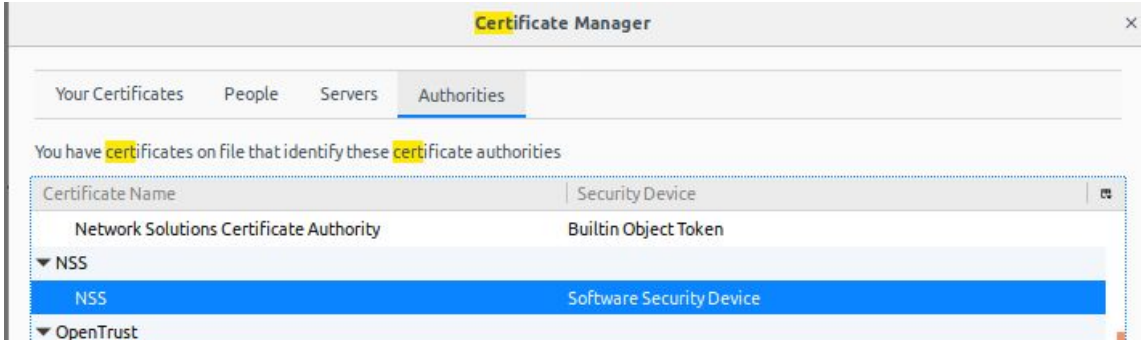

```
    Verify return code: 0 (ok)
    Extended master secret: yes
---
    0 items in the session cache
    0 client connects (SSL_connect())
    0 client renegotiates (SSL_connect())
    0 client connects that finished
    1 server accepts (SSL_accept())
    0 server renegotiates (SSL_accept())
    1 server accepts that finished
    0 session cache hits
    0 session cache misses
    0 session cache timeouts
    0 callback cache hits
    0 cache full overflows (128 allowed)
---
Client certificate
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            d2:c2:85:9f:5a:6e:3a:92
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=IN, ST=DL, L=DL, O=NSS, OU=NSS, CN=NSS/emailAddress=kaustav16048@iiitd.ac.in
        Validity
            Not Before: Mar 27 09:34:35 2019 GMT
            Not After : Nov  3 09:34:35 2024 GMT
        Subject: C=IN, ST=Delhi, L=DL, O=SNN, OU=SNN, CN=SNN/emailAddress=sushant@iiitd.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
```

A5. Here we configured our browser by adding server, client, CA certificate.

When server is running in verification mode and client tries to connect to the server, then server ask for the clients certificate. Which it verifies and create a SSL Connection.



Here we added client certificate which is used to verify client.



In this screenshot you can see that we added CA certificate as trusted authorities

Below is the server side response which shows that the client is verified.

```
ACCEPT
depth=1 C = IN, ST = DL, L = DL, O = NSS, OU = NSS, CN = NSS, emailAddress = kaustav16048@iiitd.ac.i
n
verify return:1
depth=0 C = IN, ST = Delhi, L = DL, O = SNN, OU = SNN, CN = SNN, emailAddress = sushant@iiitd.ac.in
verify return:1
ACCEPT
```

Task 2

The VM configurations used are

VM 1 ip 10.0.0.5 **server** hostname= beehive

VM 2 ip 10.0.0.6 **client** hostname = beehive2

Reference:

https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nfs-mount-on-ubuntu-16-04

https://www.digitalocean.com/community/tutorials/how-to-encrypt-traffic-to-redis-with-stunnel-on-ubuntu-16-04#what-is-stunnel

1.Installed nfs-kernel-server on server
2. Installed nfs-common on client

```
sushant@beehive:/home$ sudo apt-get install nfs-kernel-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  nfs-kernel-server
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 94.0 kB of archives.
After this operation, 344 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/main amd64 nfs-kernel-server amd64 1:1.3.4-2.1ubuntu5
[94.0 kB]
Fetched 94.0 kB in 1s (80.5 kB/s)
Selecting previously unselected package nfs-kernel-server.
(Reading database ... 65467 files and directories currently installed.)
Preparing to unpack .../nfs-kernel-server_1%3a1.3.4-2.1ubuntu5_amd64.deb ...
Unpacking nfs-kernel-server (1:1.3.4-2.1ubuntu5) ...
Processing triggers for ureadahead (0.100.0-20) ...
Setting up nfs-kernel-server (1:1.3.4-2.1ubuntu5) ...
Created symlink /etc/systemd/system/multi-user.target.wants/nfs-server.service → /lib/systemd/system
/nfs-server.service.
Job for nfs-server.service canceled.

Creating config file /etc/exports with new version

Creating config file /etc/default/nfs-kernel-server with new version
Processing triggers for systemd (237-3ubuntu10.12) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for ureadahead (0.100.0-20) ...
sushant@beehive:/home$
```

```
sudo asushant@beehive2:~$ sudo apt-get install nfs-common
[sudo] password for sushant:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  keyutils libnfsidmap2 libtirpc1 libwrap0 rpcbind
Suggested packages:
  watchdog
The following NEW packages will be installed:
  keyutils libnfsidmap2 libtirpc1 libwrap0 nfs-common rpcbind
0 upgraded, 6 newly installed, 0 to remove and 50 not upgraded.
Need to get 443 kB of archives.
After this operation, 1,462 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://archive.ubuntu.com/ubuntu bionic/main amd64 keyutils amd64 1.5.9-9.2ubuntu2 [47.9 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic/main amd64 libnfsidmap2 amd64 0.25-5.1 [27.2 kB]
Get:3 http://archive.ubuntu.com/ubuntu bionic/main amd64 libwrap0 amd64 7.6.q-27 [46.3 kB]
Get:4 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libtirpc1 amd64 0.2.5-1.2ubuntu0.1
[75.7 kB]
Get:5 http://archive.ubuntu.com/ubuntu bionic/main amd64 rpcbind amd64 0.2.3-0.6 [40.6 kB]
Get:6 http://archive.ubuntu.com/ubuntu bionic/main amd64 nfs-common amd64 1:1.3.4-2.1ubuntu5 [205 kB
]
```

3. Now we modify the **/etc/exports** file to allow remote access to **client vm 10.0.0.6**

We are allowing read and write access, and forcing the sync, and turning of root squash and avoiding subtree checks to prevent checking of files on every request.

```
sushant@beehive:/home$ sudo vim /etc/exports_
```

```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/home/sushant    10.0.0.6(rw,sync,no_root_squash,no_subtree_check)_
```

4. Restart the nfs server for changes to take place.
5. Confirm that firewall is off
6. Create directory for mounting remote server's home directory
7. Mount the home directory of server to client using **mount** command
8. Check that mount is successful using **df -h**

```
sushant@beehive:/home$ sudo systemctl restart nfs-kernel-server
sushant@beehive:/home$ _
```

```
sushant@beehive:/home$ sudo ufw status
Status: inactive
sushant@beehive:/home$ _
```

```
sushant@beehive2:~$ sudo mkdir -p /nfs/home/sushant
sushant@beehive2:~$ sudo mount 10.0.0.5:/home/sushant /nfs/home/sushant
sushant@beehive2:~$ _
```

```
sushant@beehive2:~$ sudo mount 10.0.0.5:/home/sushant /nfs/home/sushant
sushant@beehive2:~$ df -h
Filesystem                Size  Used Avail Use% Mounted on
udev                      1.9G     0  1.9G   0% /dev
tmpfs                     395M  988K  394M   1% /run
/dev/sda2                 9.8G  4.7G  4.7G  50% /
tmpfs                     2.0G     0  2.0G   0% /dev/shm
tmpfs                     5.0M     0  5.0M   0% /run/lock
tmpfs                     2.0G     0  2.0G   0% /sys/fs/cgroup
/dev/loop0                 91M   91M     0 100% /snap/core/6350
/dev/loop1                 91M   91M     0 100% /snap/core/6405
tmpfs                     395M     0  395M   0% /run/user/1000
/dev/loop2                 90M   90M     0 100% /snap/core/6673
10.0.0.5:/home/sushant    9.8G  4.2G  5.2G  45% /nfs/home/sushant
sushant@beehive2:~$ _
```

9. Create a file nss_client in /home/sushant
10. Check that corresponding file is created both in client and server (hence read and write is allowed)

```
sushant@beehive2:~$ sudo touch /nfs/home/sushant/nss_client
sushant@beehive2:~$ ls -l /nfs/home/sushant
total 40
-rw-r--r-- 1 root    root    1167 Mar  8 13:31 capture.pcap
-rw-rw-r-- 1 sushant sushant 1252 Mar  8 10:43 client2_hmac.sh
-rw-rw-r-- 1 sushant sushant  231 Mar  8 10:43 client2.sh
-rw-rw-r-- 1 sushant sushant  865 Mar  9 18:09 client_hmac.sh
-rw-rw-r-- 1 sushant sushant  117 Mar  8 11:07 client.sh
-rw-rw-r-- 1 sushant sushant   50 Mar  7 22:40 input.txt
-rw-rw-r-- 1 sushant sushant   32 Mar  7 20:59 iv3.txt
-rw-rw-r-- 1 sushant sushant   32 Mar  7 20:57 key3.txt
-rw-r--r-- 1 root    root       0 Apr  1 07:58 nss_client
-rw-rw-r-- 1 sushant sushant   29 Mar  7 22:40 output.txt
-rw-rw-r-- 1 sushant sushant   83 Mar  7 20:19 server.sh
sushant@beehive2:~$
```

← client

```
sushant@beehive:/home$ cd sushant/
sushant@beehive:~$ ls -l
total 40
-rw-r--r-- 1 root    root    1167 Mar  8 13:31 capture.pcap
-rw-rw-r-- 1 sushant sushant 1252 Mar  8 10:43 client2_hmac.sh
-rw-rw-r-- 1 sushant sushant  231 Mar  8 10:43 client2.sh
-rw-rw-r-- 1 sushant sushant  865 Mar  9 18:09 client_hmac.sh
-rw-rw-r-- 1 sushant sushant  117 Mar  8 11:07 client.sh
-rw-rw-r-- 1 sushant sushant   50 Mar  7 22:40 input.txt
-rw-rw-r-- 1 sushant sushant   32 Mar  7 20:59 iv3.txt
-rw-rw-r-- 1 sushant sushant   32 Mar  7 20:57 key3.txt
-rw-r--r-- 1 root    root       0 Apr  1 07:58 nss_client
-rw-rw-r-- 1 sushant sushant   29 Mar  7 22:40 output.txt
-rw-rw-r-- 1 sushant sushant   83 Mar  7 20:19 server.sh
sushant@beehive:~$ _
```

← server

11. Now add some content to the file **nss_client**
12. Capture packets using tcpdump on server
13. Analyse packets using tcpflow

```
sushant@beehive:~$ sudo tcpdump -i enp0s8 -s0 -w capture.pcap
```

```
sushant@beehive:~$ tcpflow -C -r capture.pcap
```

```
`65\A-
7 Obeehive25\A-$n%=@#2%]0\A-
75\A-0(\80P\80P\80P
3 Obeehive25\A-n%=@#2 n%=@#2
nss_clientnss_client~
B5\A- \9N\9.\9N\9.
$9 Obeehive25\A-n%=@#2\A-open id:3(
nss_client
-:
t95\A-\A-\9.\9.$
:\9.=@#2%00\9.\9.\9.%
: Obeehive25\A-$n%=@#2%e-/"@:
:5\A-"@:\9.=@#2%00\9.\9.\9.%
; Obeehive25\A-$n%=@#2%e-/8\A-'hello from client
hi to server
changes
0
:5\A-8'\O8\10(\9.'\9.\9.
< Obeehive25\A-$n%=@#2%e-/:
<5\A-:\9.'=@#2%00\9.\9.\9.%
= Obeehive25\A-$n%=@#2%e-/":
=5\A-":\9'=@#2%00\9.\9\9.%
> Obeehive25\A-$n%=@#2%e-/O\A-
>5\A-O(\9'\9\9.
? Obeehive25\A-n%=@#2nss_client~
t?5\A-\9.\:%"
B210#"! Utpad\A-$n%=@#2%U8\A-ObOVIM 8.08\%rootbeehive2/nfs/home/sushant/nss_clientutf-8
hi to serverhello from client0
@5\A-80\O8\10(\:%"O\:%"\:%"
A Obeehive25\A-$n%=@#2%UO\A-
A5\A-O(\:%"O\:%"\:%"
B Obeehive25\A-n%=@#2.nss_client.swp
tB5\A-\:%"\:B.
```

Content added to nss_client clearly visible

1. Install stunnel4 on client and server
2. We are using the self signed certificate created in previous task
3. Copy the server certificate to /etc/stunnel/server.crt
4. Copy the server key file to /etc/stunnel/server-key.key
5. Transfer the server certificate to client as well
6. Create a file /etc/stunnel/nfs.conf on **server** and put content as shown
7. Create a file /etc/stunnel/nfs.conf on **client** and put content as shown
8. Check that stunnel is connected after restarting the service of stunnel

```
sushant@beehive:~$ sudo apt-get install stunnel4
```

Below screenshot shows that stunnel is listening on desired port 2049.
We used **sudo netstat -plunto** for checking this

```
sushant@beehive2:~$ sudo netstat -plunto
[sudo] password for sushant:
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
 Timer
tcp        0      0 0.0.0.0:44521           0.0.0.0:*               LISTEN      1903/rpc.statd
 off (0.00/0/0)
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      475/rpcbind
 off (0.00/0/0)
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      686/systemd-resolve
 off (0.00/0/0)
tcp        0      0 0.0.0.0:2049            0.0.0.0:*               LISTEN      2334/stunnel4
 off (0.00/0/0)
tcp6       0      0 :::111                  :::*                    LISTEN      475/rpcbind
 off (0.00/0/0)
tcp6       0      0 :::42171                :::*                    LISTEN      1903/rpc.statd
 off (0.00/0/0)
```

← client

```
pid = /run/stunnel-nfs.pid

[nfs]
client = yes
accept = 127.0.0.1:2363
connect = 10.0.0.5:2049
CAfile = /etc/stunnel/server.crt
verify = 4

"/etc/stunnel/nfs.conf" 8L, 139C
```

Server →

```
pid = /run/stunnel-nfs.pid

[nfs]
client = no
cert = /etc/stunnel/server.crt
key = /etc/stunnel/server-key.key
accept = 10.0.0.5:2363
connect = 127.0.0.1:2049

"/etc/stunnel/nfs.conf" 8L, 159C
```

9. Modify the /etc/exports file accordingly to export home directory to localhost where stunnel connects

```
sushant@beehive:~$ egrep -i 'nfs' /etc/services
nfs             2049/tcp                        # Network File System
nfs             2049/udp                        # Network File System
```

← Checking which port allows tcp

```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/home/sushant    10.0.0.6(rw,sync,no_root_squash,no_subtree_check)
/home/sushant    127.0.0.1(rw,insecure,sync,no_root_squash,no_subtree_check)
```

10. Mount the home directory on local ip for stunnel accepting connections
11. Restart the services
12. Try creating a file and making changes
13. Capture the traffic using tcpdump and save to file **capst.pcap**
14. Cat the capst.pcap file to see if the content is encrypted

```
sushant@beehive:~$ sudo systemctl restart stunnel4.service
sushant@beehive:~$ sudo systemctl restart nfs-kernel-server
```

```
sushant@beehive2:~$ sudo mount 127.0.0.1:2363:/home/sushant /nfs/home/sushant
```

```
sushant@beehive:~$ sudo tcpdump -i enp0s8 -s0 -w capst.pcap_
```

```
sushant@beehive:~$ cat capst.pcap
```