Lab3 report

Donghang Lu

Task1: Create a Host-to-Host Tunnel using TUN/TAP.

In this lab we need to create a host to host tunnel using TUN/TAP. Roughly we need to realize the network structure below:

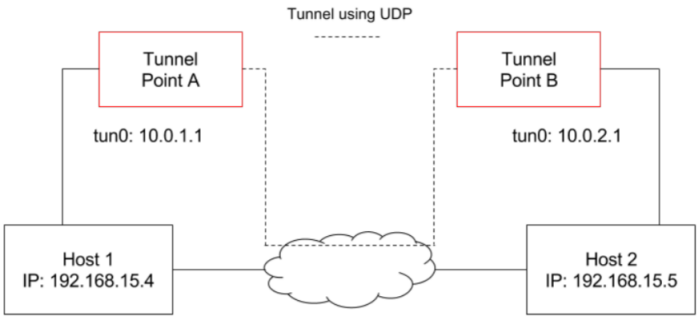

Figure 1: Host-to-Host Tunnel

1.Launch two virtual machines.

In this task IP of my machine A is 192.168.15.11. And IP of machine B is 192.168.15.12.

2.Tunnel point A:

First type this command to start the program:

```
On Machine 192.168.15.11:
# sudo ./simpletun -i tun0 -s -d
```

After this we need to open another window to configure information about tun0 interface:

```
On Machine 192.168.15.11:
# sudo ip addr add 10.0.1.1/24 dev tun0
# sudo ifconfig tun0 up
```

And the information of machine A is as bellows:

```
[03/16/2017 20:41] cs528user@cs528vm:~$ sudo ip addr add 10.0.1.1/24 dev tun0
[sudo] password for cs528user:
[03/16/2017 20:41] cs528user@cs528vm:~$ sudo ifconfig tun0 up
[03/16/2017 20:42] cs528user@cs528vm:~$ ifconfig
eth14     Link encap:Ethernet  HWaddr 08:00:27:c2:8b:13
          inet addr:192.168.15.11  Bcast:192.168.15.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fec2:8b13/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1911 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2143 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:309812 (309.8 KB)  TX bytes:241106 (241.1 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:164 errors:0 dropped:0 overruns:0 frame:0
          TX packets:164 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:17295 (17.2 KB)  TX bytes:17295 (17.2 KB)

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:10.0.1.1  P-t-P:10.0.1.1  Mask:255.255.255.0
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

[03/16/2017 20:42] cs528user@cs528vm:~$
```

Then add routing path in machine A:

```
[03/16/2017 20:48] cs528user@cs528vm:~$ sudo route add -net 10.0.2.0 netmask 255
.255.255.0 dev tun0
[sudo] password for cs528user:
[03/16/2017 20:49] cs528user@cs528vm:~$
```

3.Tunnel Point B:

On machine B type this command to run the program:

On Machine 192.168.15.12: # sudo ./simpletun -i tun0 -c 192.168.15.4 -d

And add routing path and ip addresses in machine B:

```
[03/16/2017 20:46] cs528user@cs528vm:~$ sudo ip addr add 10.0.2.1/24 dev tun0
[sudo] password for cs528user:
[03/16/2017 20:47] cs528user@cs528vm:~$ sudo ifconfig tun0 up
[03/16/2017 20:47] cs528user@cs528vm:~$ sudo route add -net 10.0.1.0 netmask 255
.255.255.0 dev tun0
[03/16/2017 20:48] cs528user@cs528vm:~$
```

4.Using the tunnel

Here is the result of ping. (from machineA (192.168.15.11 & 10.0.1.1) to machine B(10.0.2.1 & 192.168.15.12)

```
[03/16/2017 20:49] cs528user@cs528vm:~$ ping 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
64 bytes from 10.0.2.1: icmp_req=1 ttl=64 time=2.56 ms
64 bytes from 10.0.2.1: icmp_req=2 ttl=64 time=40.2 ms
64 bytes from 10.0.2.1: icmp_req=3 ttl=64 time=39.1 ms
64 bytes from 10.0.2.1: icmp_req=4 ttl=64 time=40.8 ms
^C
--- 10.0.2.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 2.563/30.696/40.884/16.256 ms
```

This is output of machine A:

```
[03/16/2017 20:40] cs528user@cs528vm:~$ sudo ./simpletun -i tun0 -s -d
Successfully connected to interface tun0
SERVER: Client connected from 192.168.15.12
TAP2NET 1: Read 84 bytes from the tap interface
TAP2NET 1: Written 84 bytes to the network
NET2TAP 1: Read 84 bytes from the network
NET2TAP 1: Written 84 bytes to the tap interface
TAP2NET 2: Read 84 bytes from the tap interface
TAP2NET 2: Written 84 bytes to the network
NET2TAP 2: Read 84 bytes from the network
NET2TAP 2: Written 84 bytes to the tap interface
TAP2NET 3: Read 84 bytes from the tap interface
TAP2NET 3: Written 84 bytes to the network
NET2TAP 3: Read 84 bytes from the network
NET2TAP 3: Written 84 bytes to the tap interface
TAP2NET 4: Read 84 bytes from the tap interface
TAP2NET 4: Written 84 bytes to the network
NET2TAP 4: Read 84 bytes from the network
NET2TAP 4: Written 84 bytes to the tap interface
```

This is output of machine B:

```
[03/16/2017 20:44] cs528user@cs528vm:~$ sudo ./simpletun -i tun0 -c 192.168.15.11 -d
[sudo] password for cs528user:
Successfully connected to interface tun0
CLIENT: Connected to server 192.168.15.11
NET2TAP 1: Read 84 bytes from the network
NET2TAP 1: Written 84 bytes to the tap interface
TAP2NET 1: Read 84 bytes from the tap interface
TAP2NET 1: Written 84 bytes to the network
NET2TAP 2: Read 84 bytes from the network
NET2TAP 2: Written 84 bytes to the tap interface
TAP2NET 2: Read 84 bytes from the tap interface
TAP2NET 2: Written 84 bytes to the network
NET2TAP 3: Read 84 bytes from the network
NET2TAP 3: Written 84 bytes to the tap interface
TAP2NET 3: Read 84 bytes from the tap interface
TAP2NET 3: Written 84 bytes to the network
NET2TAP 4: Read 84 bytes from the network
NET2TAP 4: Written 84 bytes to the tap interface
TAP2NET 4: Read 84 bytes from the tap interface
TAP2NET 4: Written 84 bytes to the network
```

Here is the result of ssh from machine A to machine B:

```
[03/16/2017 20:50] cs528user@cs528vm:~$ ssh 10.0.2.1
The authenticity of host '10.0.2.1 (10.0.2.1)' can't be established.
ECDSA key fingerprint is 81:82:a9:af:bd:93:78:f9:1a:a7:ca:7f:e8:d6:6c:04.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.1' (ECDSA) to the list of known hosts.
cs528user@10.0.2.1's password:
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-37-generic i686)

 * Documentation:  https://help.ubuntu.com/

New release '14.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Thu Mar 16 20:46:37 2017 from mc15.cs.purdue.edu
```

We can see this tunnel is successfully constructed.

Questions: why tcp over tcp is a bad idea?

Because this will cause long delays and frequent connection aborts. And this is due to TCP timeout policy. If we run tcp over tcp, for the same packet the upper layer and the lower layer TCP have two different timers. When connection begins losing packets the lower layer will request a retransmission and increases its timeouts. Then as the result the upper layer will also request a retransmission since no ACK recreived. Because the timeout is still less than the lower layer timeout, the upper layer will queue up more retransmissions faster than the lower layer can process them. This makes the upper layer connection stall very quickly and every retransmission just adds to the problem。

That's why we will choose udp tunnel in this lab.

2. Task 2: Create a Private Network Using a Gateway

Note: the screenshots here are made after I finish all the tasks.. So there might be some additional information in screenshots.

In this task we need to finish network like this:

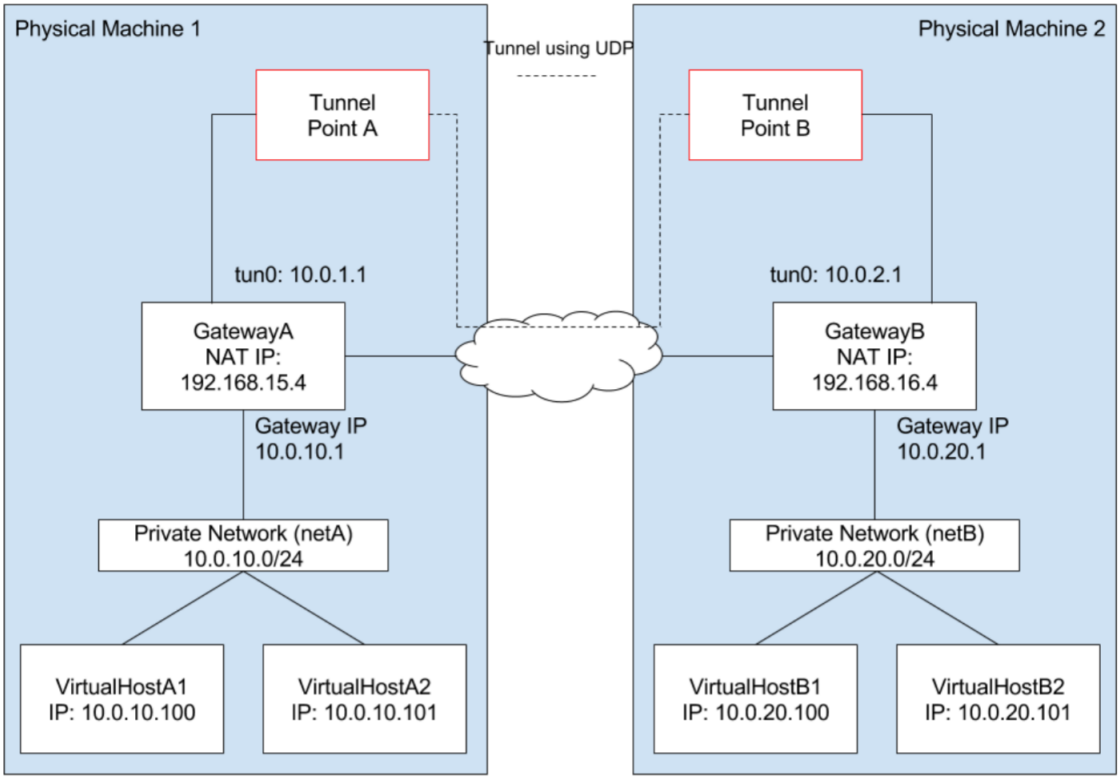

Figure 2: Gateway to Gateway Tunnel

2.1: Create gatewayA and Client:

My gatawayA machine has IP 192.168.16.4 (which is different from tutorial since I once rebuilt all the network and removed all the former IPs..)

This is information of lu562netA:

```
NetworkName:    lu562netA
IP:             192.168.16.1
Network:        192.168.16.0/24
IPv6 Enabled:   No
IPv6 Prefix:
DHCP Enabled:   Yes
Enabled:        Yes
Port-forwarding (ipv4)
        guestssh1:tcp:[]:30333:[192.168.15.4]:22
        ssh:tcp:[]:30303:[192.168.16.4]:22
        vpn:tcp:[]:30399:[192.168.16.4]:30399
        vpnudp:udp:[]:30398:[192.168.16.4]:30398
loopback mappings (ipv4)
        127.0.0.1=2
```

And this is gatewayA NIC info:

Use this command:

```
mc15 54 $ vboxmanage showvminfo gatewayA
```

Result:

```
NIC 1:           MAC: 080027AE7BA7, Attachment: NAT Network 'lu562netA', Cable connected: on,
 Trace: off (file: none), Type: 82540EM, Reported speed: 0 Mbps, Boot priority: 0, Promisc Po
licy: allow-all, Bandwidth group: none
NIC 2:           MAC: 0800277BBCE2, Attachment: Internal Network 'lu562inetA', Cable connecte
d: on, Trace: off (file: none), Type: 82540EM, Reported speed: 0 Mbps, Boot priority: 0, Prom
isc Policy: allow-all, Bandwidth group: none
```

We can see NIC1 of gatewayA is connected to the NAT network “lu562netA” and the second NIC is responsible for intent.

And we can check something in gateway A:

Setting a static Ip for the gateway:

```
[03/25/2017 22:59] cs528user@cs528vm:~$ cat /etc/network/interfaces
auto lo
iface lo inet loopback

auto eth14
iface eth14 inet dhcp

auto eth15
iface eth15 inet static
address 10.0.10.1
netmask 255.255.255.0
gateway 192.168.16.1
```

Set ip forwarding on gateway:

```
[03/20/2017 17:57] cs528user@cs528vm:~$ sudo sysctl net.ipv4.ip_forward=1
[sudo] password for cs528user:
net.ipv4.ip_forward = 1
```

Setting IP table:

```
[03/20/2017 17:57] cs528user@cs528vm:~$ sudo iptables-save
# Generated by iptables-save v1.4.12 on Mon Mar 20 17:59:49 2017
*nat
:PREROUTING ACCEPT [93:8027]
:INPUT ACCEPT [7:816]
:OUTPUT ACCEPT [90:5987]
:POSTROUTING ACCEPT [20:1352]
-A POSTROUTING -o eth14 -j MASQUERADE
COMMIT
# Completed on Mon Mar 20 17:59:49 2017
# Generated by iptables-save v1.4.12 on Mon Mar 20 17:59:49 2017
*filter
:INPUT ACCEPT [596:74897]
:FORWARD ACCEPT [92:51263]
:OUTPUT ACCEPT [487:46489]
-A FORWARD -i eth15 -j ACCEPT
COMMIT
# Completed on Mon Mar 20 17:59:49 2017
```

Creating dhcp server for the internal network:

```
[03/20/2017 17:59] cs528user@cs528vm:~$ head -n 20 /etc/udhcpd.conf
# Sample udhcpd configuration file (/etc/udhcpd.conf)

# The start and end of the IP lease block

start           10.0.10.100     #default: 192.168.0.20
end             10.0.10.200     #default: 192.168.0.254


# The interface that udhcpd will use

interface       eth15           #default: eth0


# The gateway IP that will be given to clients

opt     router  10.0.10.1

# The maximim number of leases (includes addressesd reserved
# by OFFER's, DECLINE's, and ARP conficts
```

And we can use the same method to create gatewayB and client.

After all this settings Virtualhost will be able to connect to “outside world”. For instance, Virtualhost A can ping google:

```
[03/26/2017 14:39] cs528user@cs528vm:~$ ping www.google.com
PING www.google.com (216.58.192.164) 56(84) bytes of data.
64 bytes from ord36s02-in-f4.1e100.net (216.58.192.164): icmp_req=1 ttl=52 time=13.6 ms
64 bytes from ord36s02-in-f164.1e100.net (216.58.192.164): icmp_req=2 ttl=52 time=10.1 ms
64 bytes from ord36s02-in-f4.1e100.net (216.58.192.164): icmp_req=3 ttl=52 time=19.5 ms
64 bytes from ord36s02-in-f164.1e100.net (216.58.192.164): icmp_req=4 ttl=52 time=13.0 ms
^C
--- www.google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 10.142/14.088/19.541/3.416 ms
[03/26/2017 14:39] cs528user@cs528vm:~$
```

3. Task3 Create a Gateway-to-Gateway Tunnel.

When I worked through here, I rebuilt all the network so now the IPs are like this:

GatewayA: 192.168.16.4 (on mc15)

GatewayB:192.168.18.5 (on mc14)

Lu562netA:192.168.16.0/24 (on mc15)

Lu562netB:192.168.18.0/24 (on mc14)

1.create port forwarding rules:

This is network information after setting port forwarding rules:

```
mc15 52 $ vboxmanage list natnets
NetworkName:    lu562netA
IP:             192.168.16.1
Network:        192.168.16.0/24
IPv6 Enabled:   No
IPv6 Prefix:
DHCP Enabled:   Yes
Enabled:        Yes
Port-forwarding (ipv4)
        guestssh1:tcp:[]:30333:[192.168.15.4]:22
        ssh:tcp:[]:30303:[192.168.16.4]:22
        vpn:tcp:[]:30399:[192.168.16.4]:30399
        vpnudp:udp:[]:30398:[192.168.16.4]:30398
loopback mappings (ipv4)
        127.0.0.1=2
```

```
mc14 51 $ vboxmanage list natnets
NetworkName:    lu562netB
IP:             192.168.18.1
Network:        192.168.18.0/24
IPv6 Enabled:   No
IPv6 Prefix:
DHCP Enabled:   Yes
Enabled:        Yes
Port-forwarding (ipv4)
        ssh:tcp:[]:30304:[192.168.18.5]:22
        udpvpn:udp:[]:30398:[192.168.18.5]:30398
        vpn:tcp:[]:30399:[192.168.18.5]:30399
loopback mappings (ipv4)
        127.0.0.1=2
```

Noted that I use port 30398 for udp port forwarding since our tunnel is a udp tunnel. And I use port 30399 for tcp port forward since control channel will use tcp to establish connection and finish key agreement.

2. Starting the Tunnel Point A:

Just run our program whith command:

On GatewayA (192.168.16.4):
sudo ./simpletun -i tun0 -p -s -d

After this we need to open another window and run these commands(for convenience I wrote them into a sh file):

```
  GNU nano 2.2.6                File: set.sh

#!/bin/sh
sudo ip addr add 10.0.1.1/24 dev tun0
sudo ifconfig tun0 up
sudo route add -net 10.0.20.0 netmask 255.255.255.0 dev tun0
sudo route add -net 192.168.18.0 netmask 255.255.255.0 dev tun0
```

3. Starting the Tunnel Point B:

Run the program with this command:

On GatewayB (192.168.18.5):
sudo ./simpletun -i tun0 -p -c -d

And open another window then type these commands:

```
  GNU nano 2.2.6                    File: set.sh

#!/bin/sh
sudo ip addr add 10.0.2.1/24 dev tun0
sudo ifconfig tun0 up
sudo route add -net 10.0.10.0 netmask 255.255.255.0 dev tun0
```

Then our connection is established. When I write this report my program has already achieve the functions needed for task4 and task5. So I will not show the result here but in later chapters of this report.

4.About certificates:

In order to use OpenSSL to create certificates, you have to have a configuration file. I just copy /usr/lib/ssl/openssl.cnf as my configuration file.

And next I will create my own CA:

```
[03/24/2017 14:07] cs528user@cs528vm:~/demo_openssl/test$ openssl req -new -x509 -keyout ca.k
ey -out ca.crt -config openssl.cnf
Generating a 1024 bit RSA private key
....++++++
.....................................++++++
writing new private key to 'ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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]:DL
State or Province Name (full name) [Some-State]:IN
Locality Name (eg, city) []:lafayette
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Purdue
Organizational Unit Name (eg, section) []:CS528
Common Name (e.g. server FQDN or YOUR name) []:DonghangLu
Email Address []:lu562@purdue.edu
[03/24/2017 14:10] cs528user@cs528vm:~/demo_openssl/test$
```

Build certificate for Server:

```
[03/24/2017 14:19] cs528user@cs528vm:/usr/lib/ssl$ sudo openssl genrsa -des3 -out server.key
1024
Generating RSA private key, 1024 bit long modulus
........++++++
.....................................................................++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
[03/24/2017 14:20] cs528user@cs528vm:/usr/lib/ssl$ sudo openssl req -new -key server.key -out
 server.csr -config openssl.cnf
Enter pass phrase for 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]:DL
State or Province Name (full name) [Some-State]:WL
Locality Name (eg, city) []:WL
Organization Name (eg, company) [Internet Widgits Pty Ltd]:PD
Organizational Unit Name (eg, section) []:CS
Common Name (e.g. server FQDN or YOUR name) []:donghanglu
Email Address []:foo@purdue.edu

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:server
An optional company name []:none
[03/24/2017 14:22] cs528user@cs528vm:/usr/lib/ssl$
```

Build certificate for Client:

```
[03/24/2017 14:25] cs528user@cs528vm:~/demo_openssl/test$ openssl genrsa -des3 -out client.ke
y 1024
Generating RSA private key, 1024 bit long modulus
.................++++++
........++++++
e is 65537 (0x10001)
Enter pass phrase for client.key:
Verifying - Enter pass phrase for client.key:
[03/24/2017 14:26] cs528user@cs528vm:~/demo_openssl/test$ openssl req -new -key client.key -o
ut client.csr -config openssl.cnf
Enter pass phrase for client.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]:DL
State or Province Name (full name) [Some-State]:IN
Locality Name (eg, city) []:WL
Organization Name (eg, company) [Internet Widgits Pty Ltd]:PD
Organizational Unit Name (eg, section) []:CS
Common Name (e.g. server FQDN or YOUR name) []:donghanglu
Email Address []:foo@purdue.edu

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:client
An optional company name []:none
[03/24/2017 14:27] cs528user@cs528vm:~/demo_openssl/test$
```

The CSR file needs to have the CA's signature to form a certificate. So I use my ca to generate certificates:

For server:

```
[03/24/2017 14:34] cs528user@cs528vm:~/demo_openssl/test$ openssl ca -in server.csr -out serv
er.crt -cert ca.crt -keyfile ca.key -config openssl.cnf
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4096 (0x1000)
        Validity
            Not Before: Mar 24 21:34:39 2017 GMT
            Not After : Mar 24 21:34:39 2018 GMT
        Subject:
            countryName               = DL
            stateOrProvinceName       = WL
            localityName              = WL
            organizationName          = PD
            organizationalUnitName    = CS
            commonName                = donghanglu
            emailAddress              = foo@purdue.edu
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                99:8C:2D:15:20:BF:83:ED:BD:00:6B:26:C8:8A:51:C2:5C:AB:5F:40
            X509v3 Authority Key Identifier:
                keyid:13:76:DD:91:2F:57:FA:FA:64:85:A0:70:DA:CE:37:F6:61:AF:F8:50

Certificate is to be certified until Mar 24 21:34:39 2018 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
[03/24/2017 14:34] cs528user@cs528vm:~/demo_openssl/test$
```

For Client:

```
[03/24/2017 14:34] cs528user@cs528vm:~/demo_openssl/test$ openssl ca -in client.csr -out clie
nt.crt -cert ca.crt -keyfile ca.key -config openssl.cnf
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4097 (0x1001)
        Validity
            Not Before: Mar 24 21:43:49 2017 GMT
            Not After : Mar 24 21:43:49 2018 GMT
        Subject:
            countryName               = DL
            stateOrProvinceName       = IN
            localityName              = WL
            organizationName          = PD
            organizationalUnitName    = CS
            commonName                = donghanglu
            emailAddress              = foo@purdue.edu
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                9E:19:44:14:81:43:3A:51:FA:A1:02:FC:09:48:7C:C8:B4:80:21:D2
            X509v3 Authority Key Identifier:
                keyid:13:76:DD:91:2F:57:FA:FA:64:85:A0:70:DA:CE:37:F6:61:AF:F8:50

Certificate is to be certified until Mar 24 21:43:49 2018 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
```

Until now I have enough certificates to build a VPN tunnel.

5. Task 4 && Task5: Create a Virtual Private Network (VPN)

I will combinate task4 and task5 together in the report since the final goal of these two tasks is to build a secured VPN.

5.1 Authentication:

Before the authentication, both client and server have already set up their public/private keys and certificates properly, so they can verify each other's public-key certificate.

And I use OpenSSL's SSL functions to make an SSL connection and deal with authentication.

So when establishing connection, server and client will exchange their certificate and verify the other's certificate.

5.2 Key Exchange:

Here we use TCP channel to achieve key exchange. At first I tried to use ECDH to finish key exchange but it is kind of too complicated and I had trouble dealing with EVP_PKEY. So instead, I chose to design another way to achieve key exchange. And here is my strategy:

At first both server and client will generate a random 256 bits key Kserver and Kclient. Server will send Kserver to client and client will send Kclient to server. And the shared key will be the xor of two keys: Kserver^Kclient.

However, this design will not prevent man in the middle attack. If we want to prevent man in the middle attack Server needs to send both Kserver and its signature on Kserver. So the client has to get server's public signing key in advance and due to limited time I didn't achieve that.

But When I write code for key agreement, the packets are sent and received using SSL_write and SSL_read and I'm not so sure if OpenSSL will do the job mentioned above for me. If hopefully SSL_write and SSL_read provide this property then our VPN tunnel will be protected from man in the middle attack and replay attack. (anyway replay attack will not work since each time we choose random keys).

5.3 IV Exchange

I use the similar way to deal with IV. At first both server and client will generate a random 128 bits IV IVserver and IVclient. Server will send IVserver to client and client will send IVclient to server. And the shared IV will be the xor of two IVs: IVserver^IVclient.

My idea for this design is very simple. That is each side has to do some contribution to the final shared data so the most basic way is both of them provide a value and xor these two values.

5.4 Securing the Tunnel:

So when we get the session key we can achieve this goal. There are two properties we need to consider: confidentiality and integrity. So we need to choose an encryption algorithm and an MAC algorithm since what we agreed on is a symmetric key.

Encryption algorithm we chose is AES256 and MAC algorithm is HMAC-SHA256. The reason is quite simple: AES256 and HMAC-SHA256 are both implemented in OpenSSL and these two algorithms are a balance of security and efficiency. The programs for AES 256 and HMAC_SHA256 are provided in

https://wiki.openssl.org/index.php/EVP_Signing_and_Verifying and
https://wiki.openssl.org/index.php/EVP_Symmetric_Encryption_and_Decryption so I use them directly to achieve the security goal.

The advantage of these two choices is that they are both very fast, especially compared with asymmetric methods. And the have good securicy protect because of long keys and long IVs. Disadvantage is these methods cannot provide forward security and there are stronger envryption methods and integrity methods. But I think AES and HMAC-SHA256 are enough for this lab.

And there is one more thing to be confirmed: Encrypt-then-MAC or MAC-then-Encrypt. Since the professor said that TLS1.2 uses MAC-then-Encrypt , I will follow this way. But the truth is that Encrypt-then-MAC is more secure in some scenario. So either choice is OK and my choice is MAC-then-Encrypt based on TLS 1.2.

So after all things are done, we can test our program now:

Run program on server:

```
[03/26/2017 17:04] cs528user@cs528vm:~/demo_openssl$ sudo ./simpletun -i tun0 -p 30399 -s -d
[sudo] password for cs528user:
Successfully connected to interface tun0
Enter PEM pass phrase:
```

Run program on client:

```
[03/26/2017 17:04] cs528user@cs528vm:~/demo_openssl$ sudo ./simpletun -i tun0 -p 30399 -c 128
.10.12.215 -d
[sudo] password for cs528user:
Sorry, try again.
[sudo] password for cs528user:
Successfully connected to interface tun0
Enter PEM pass phrase:
```

Type in PEM pass phrase (phrase of server is “server” and phrase of client is “client”):

Server:

```
[03/26/2017 17:13] cs528user@cs528vm:~/demo_openssl$ sudo ./simpletun -i tun0 -p 30399 -s -d
Successfully connected to interface tun0
Enter PEM pass phrase:
SSL connection using AES256-GCM-SHA384
Client certificate:
         subject: /C=DL/ST=IN/L=WL/O=PD/OU=CS/CN=donghanglu/emailAddress=foo@purdue.edu
         issuer: /C=DL/ST=IN/L=lafayette/O=Purdue/OU=CS528/CN=DonghangLu/emailAddress=lu562@p
urdue.edu
SERVER: Client connected from 128.10.12.214
key exchange finished
session key has size 32 and session key is :
ad7792b24f2bda92afc4bf63f5efccf6ceb4fb476262f889212440a9e295eb16
```

Client:

```
[03/26/2017 17:13] cs528user@cs528vm:~/demo_openssl$ sudo ./simpletun -i tun0 -p 30399 -c 128
.10.12.215 -d
Successfully connected to interface tun0
Enter PEM pass phrase:
SSL connection using AES256-GCM-SHA384
Server certificate:
haha
         subject: /C=DL/ST=WL/L=WL/O=PD/OU=CS/CN=donghanglu/emailAddress=foo@purdue.edu
         issuer: /C=DL/ST=IN/L=lafayette/O=Purdue/OU=CS528/CN=DonghangLu/emailAddress=lu562@p
urdue.edu
CLIENT: Connected to server 128.10.12.215
key exchange finished
session key has size 32 and session key is :
ad7792b24f2bda92afc4bf63f5efccf6ceb4fb476262f889212440a9e295eb16
```

We can see that this channel is secured and both sides have the same shared key.

Open another window to deal with IP setting and routing.(set.sh is the same as sh file described before)

```
[03/26/2017 17:16] cs528user@cs528vm:~/demo_openssl$ sudo sh set.sh
[sudo] password for cs528user:
[03/26/2017 17:16] cs528user@cs528vm:~/demo_openssl$
```

Then ssh into virtualhost A and let it ping 10.0.20.100:

```
[03/26/2017 17:18] cs528user@cs528vm:~$ ssh cs528user@10.0.10.100
cs528user@10.0.10.100's password:
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-37-generic i686)

 * Documentation:  https://help.ubuntu.com/

New release '14.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Mar 26 14:39:08 2017 from cs528vm-2.local
[03/26/2017 17:18] cs528user@cs528vm:~$
```

```
[03/26/2017 17:18] cs528user@cs528vm:~$ ping 10.0.20.100
PING 10.0.20.100 (10.0.20.100) 56(84) bytes of data.
64 bytes from 10.0.20.100: icmp_req=1 ttl=62 time=11.1 ms
64 bytes from 10.0.20.100: icmp_req=2 ttl=62 time=7.36 ms
64 bytes from 10.0.20.100: icmp_req=3 ttl=62 time=6.79 ms
64 bytes from 10.0.20.100: icmp_req=4 ttl=62 time=6.07 ms
64 bytes from 10.0.20.100: icmp_req=5 ttl=62 time=6.67 ms
64 bytes from 10.0.20.100: icmp_req=6 ttl=62 time=6.57 ms
64 bytes from 10.0.20.100: icmp_req=7 ttl=62 time=6.03 ms
^C
--- 10.0.20.100 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6011ms
rtt min/avg/max/mdev = 6.031/7.244/11.189/1.664 ms
[03/26/2017 17:19] cs528user@cs528vm:~$
```

We can see the network is OK. And screenshots of gatewayA and gatewayB are as bellows:

Client:

```
key exchange finished
session key has size 32 and session key is :
ad7792b24f2bda92afc4bf63f5efccf6ceb4fb476262f889212440a9e295eb16
NET2TAP 1: Read 128 bytes from the network
NET2TAP 1: Written 84 bytes to the tap interface
TAP2NET 1: Read 84 bytes from the tap interface
TAP2NET 1: Written 128 bytes to the network
NET2TAP 2: Read 128 bytes from the network
NET2TAP 2: Written 84 bytes to the tap interface
TAP2NET 2: Read 84 bytes from the tap interface
TAP2NET 2: Written 128 bytes to the network
NET2TAP 3: Read 128 bytes from the network
NET2TAP 3: Written 84 bytes to the tap interface
TAP2NET 3: Read 84 bytes from the tap interface
TAP2NET 3: Written 128 bytes to the network
NET2TAP 4: Read 128 bytes from the network
NET2TAP 4: Written 84 bytes to the tap interface
TAP2NET 4: Read 84 bytes from the tap interface
TAP2NET 4: Written 128 bytes to the network
NET2TAP 5: Read 128 bytes from the network
NET2TAP 5: Written 84 bytes to the tap interface
TAP2NET 5: Read 84 bytes from the tap interface
TAP2NET 5: Written 128 bytes to the network
NET2TAP 6: Read 128 bytes from the network
NET2TAP 6: Written 84 bytes to the tap interface
TAP2NET 6: Read 84 bytes from the tap interface
TAP2NET 6: Written 128 bytes to the network
NET2TAP 7: Read 128 bytes from the network
NET2TAP 7: Written 84 bytes to the tap interface
TAP2NET 7: Read 84 bytes from the tap interface
TAP2NET 7: Written 128 bytes to the network
```

Server:

```
Client certificate:
         subject: /C=DL/ST=IN/L=WL/O=PD/OU=CS/CN=donghanglu/emailAddress=foo@purdue.edu
         issuer: /C=DL/ST=IN/L=lafayette/O=Purdue/OU=CS528/CN=DonghangLu/emailAddress=lu562@p
urdue.edu
SERVER: Client connected from 128.10.12.214
key exchange finished
session key has size 32 and session key is :
ad7792b24f2bda92afc4bf63f5efccf6ceb4fb476262f889212440a9e295eb16
TAP2NET 1: Read 84 bytes from the tap interface
TAP2NET 1: Written 128 bytes to the network
NET2TAP 1: Read 128 bytes from the network
NET2TAP 1: Written 84 bytes to the tap interface
TAP2NET 2: Read 84 bytes from the tap interface
TAP2NET 2: Written 128 bytes to the network
NET2TAP 2: Read 128 bytes from the network
NET2TAP 2: Written 84 bytes to the tap interface
TAP2NET 3: Read 84 bytes from the tap interface
TAP2NET 3: Written 128 bytes to the network
NET2TAP 3: Read 128 bytes from the network
NET2TAP 3: Written 84 bytes to the tap interface
TAP2NET 4: Read 84 bytes from the tap interface
TAP2NET 4: Written 128 bytes to the network
NET2TAP 4: Read 128 bytes from the network
NET2TAP 4: Written 84 bytes to the tap interface
TAP2NET 5: Read 84 bytes from the tap interface
TAP2NET 5: Written 128 bytes to the network
NET2TAP 5: Read 128 bytes from the network
NET2TAP 5: Written 84 bytes to the tap interface
TAP2NET 6: Read 84 bytes from the tap interface
TAP2NET 6: Written 128 bytes to the network
NET2TAP 6: Read 128 bytes from the network
NET2TAP 6: Written 84 bytes to the tap interface
TAP2NET 7: Read 84 bytes from the tap interface
TAP2NET 7: Written 128 bytes to the network
NET2TAP 7: Read 128 bytes from the network
NET2TAP 7: Written 84 bytes to the tap interface
```

Everything works fine now.

So all the tasks for lab3 are finished successfully.