RES212 Lab #1
Certificates, TLS and VPN

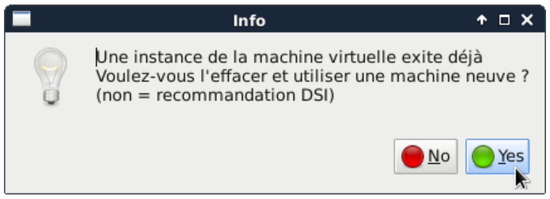
The goal of this lab is to let you become acquainted with creating and managing cryptographic certificates for use in your application, for the Web and VPNs. Given the limited amount of time, emphasis is given on the use of up-to-date cryptographic mechanism in practice, as opposite as to fully replicating a faithful and accurate networking environment (that would make the Lab last longer).

The Lab is divided in 4 main phases:

1. ***Create and manage certificates***
2. ***Using certificates for a TLS Web browser session***
3. ***Using certificates to setup a OpenVPN virtual private network***
4. ***Understand why TCP VPNs are a bad idea (TCP-over-TCP)***

The Lab is (mostly) carried on using a Virtual Machine that is readily available in the VM catalog. You can work on Mint17.3_mate or RES201 (an Ubuntu fallback in case Mint17.3_mate is not available). Please start a new VM (see picture below). Since some of the commands require super-user privileges, you will need to know the following VM passwords:

	Mint17.3_mate	*RES201*
user:	*user*	*res201*
pwd:	*spdyquic*	*res201*
root_pwd:	*spdyquic*	*res201*



Une instance de la machine virtuelle exite déjà
Voulez-vous l'effacer et utiliser une machine neuve ?
(non = recommandation DSI)

0. Preliminary set-up of your environment

Note: this part helps you in setting up your VM environment. It gives you very basic hints but assume you are familiar with Linux. The use of the same VM ensures that everybody has an (almost) consistent setup.

The following notation is used to designate command execution (and output)

```
user@user-VirtualBox ~ $ command
command output
```

Some commands require root privileges. You can achieve this by running a command as a *su*per user would *do* (sudo):

```
user@user-VirtualBox ~ $ sudo command
*****  (typing the password)
```

However, if you plan to run several commands requiring root privileges, you can become root right away, and then execute several commands as root. For instance, you will be installing OpenVPN and Web server software: you can do this with several commands such as `apt`, `apt-get` or `aptitude` as in the example below:

```
user@user-VirtualBox ~ $ sudo su
user-VirtualBox ~ # aptitude install nginx
```

These suggestions aside, the next sections will tend to assign you **high-level objectives**, as opposite to giving you **detailed low-level instruction** to achieve such goals. You can consult documents over the Internet, however please choose your source (don't just Google for some keywords) and language (it should be hopefully clear now).

Note: this icon will pop up whenever there is important points that you should think about and have understood (also for the exam!)

1. Generate certificates

The goal of this part is to become capable of generating cryptographic certificates that you can use in your applications, Web servers, or Virtual private networks.

1.1 The certificate toolchains

There are a number of ways to create X509 cryptographic certificates to be used in network applications. Shortly:

- If you want to setup a Web server certificate for an HTTPS server connected to the Internet over a domain you own/administer, instead of self-signing certificates, you need to send signing requests to a (cheap) certification authority. Here is a free one: https://letsencrypt.org/
- Application running over TLS can make use of OpenSSL to generate self-signed certificates. It is easy to make these certificates to be accepted by your own application, but no browser on earth will ever accept them.
    - A good place to start is https://pki-tutorial.readthedocs.io/en/latest/#simple-pki
    - Note: “simple” would be enough for this Lab, but at least the “Expert” tutorial is required for the real-world
- If you want to setup certificates to be used for VPNs, you not only need a certificate for the server (as in a typical TLS browsing session) but also one certificate for each client of your VPN. OpenSSL can be a burden in this case, so application such as OpenVPN come bundled with scripts (that use OpenSSL under the hood) to make your life easier.
    - As specified in /usr/share/doc/openvpn/README, the EasyRSA scripts that used to be bundled with OpenVPN are now independently managed so get them at https://github.com/OpenVPN/easy-rsa
    - A good place to start, and that we will follow, is then the official OpenVPN tutorial https://community.openvpn.net/openvpn/wiki/EasyRSA3-OpenVPN-Howto

1.2 The Certification Authority (CA)

In the interest of simplicity, we will hide the “ugly” low-level OpenSSL commands and use OpenVPN’s EasyRSA3 set of scripts to facilitate the certificate creation. We will thus follow https://community.openvpn.net/openvpn/wiki/EasyRSA3-OpenVPN-Howto.

Note that the EasyRSA tutorial is even simpler than the OpenSSL PKI-tutorial, in that it directly uses a Root CA to sign an end-system certificate (ie., there is no signing CA authority, nor intermediate CA). Systematic use of the Root CA key exposes it to cryptanalysis, which is why

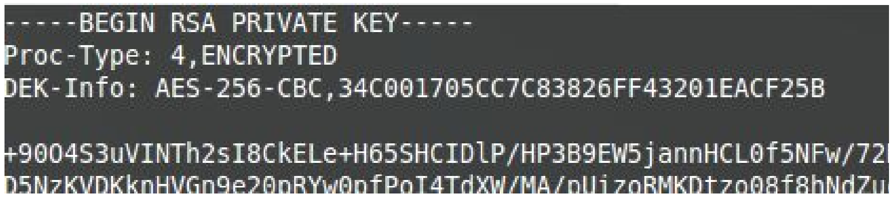
this is not how you would do it *in practice.* However, the EasyRSA3 tutorial can be considered to be fair enough for this lab.
First of all, open a terminal and download the latest version of the software for the Lab. If you do not have git clone installed (which is sure), please install it before proceeding. For the sake of simplicity, we will do everything (CA, server and client) in the same VM (i.e., we do as if different folders in the filesystem of the VM were actually systems). To separate these properly, let us create separate folders. Let us start with the certification authority.

```
user@user-VirtualBox ~ $ mkdir CA; cd CA
user@user-VirtualBox ~/CA $ git clone https://github.com/OpenVPN/easy-rsa
```

Then, change directory to your easy-rsa folder that is going to become the CA PKI management folder and start following the tutorial steps. We will not cut'n'paste the tutorial here (the point is not in cut'n'pasting commands from the Lab description to the VM terminal). Thus, while following the tutorial, be sure that you understand what you are doing by answering the following questions!

If you feel you are unsure, revise the certificate course and have a look at documentation at https://github.com/OpenVPN/easy-rsa/tree/v3.0.5/doc

- For instance, after having created a CA, can you tell what is the purpose of the DEK-Info field in the CA private key file (`~/CA/easy-rsa/easyrsa3/pki/private/ca.key`) ?
- Also, do you feel safe that AES-256-CBC is being used to protect the private key in light of the BEAST attack ?



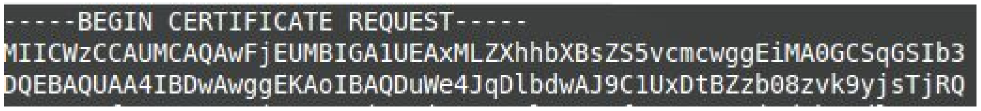
```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-256-CBC,34C001705CC7C83826FF43201EACF25B

+9004S3uVINTh2sI8CkELe+H65SHCIDlP/HP3B9EW5jannHCL0f5NFw/72
D5NzKVDKknHVGn9e20nRYw0nfPoI4TdXW/MA/nUizoRMKDtzo08f8hNdZu
```

1.3 Generating Certificate Signing Requests (CSRs)

Then, let move to the server and keep following the OpenVPN tutorial: in this Lab, we will be using a single VM, which means creating a `server` folder and generate server keys.

For the sake of example, we'll be using `example.org` as a domain name. Since (of course) `example.org` really exists and is reachable in the Internet, we will need to take care of that to test our certificate (later).

What does the Certificate Signing Request (CSR) file contains (`~/server/easy-rsa/easyrsa3/pki/reqs/example.req`) ?



```
-----BEGIN CERTIFICATE REQUEST-----
MIICWzCCAUMCAQAwFjEUMBIGA1UEAxMLZXhhbXBsZS5vcmcwggEiMA0GCSqGSIb3
DQEBAQUAA4IBDwAwggEKAoIBAQDuWe4JqDlbdwAJ9C1UxDtBZzb08zvk9yjsTjRQ
```

Normally, you should send this CSR to the CA with some mean (email, RFC1149 or RDC2549 or similar).

Which security service do you need to transfer the CSR to the CA ? Particularly, does the CSR payload require

- Encryption,
- Integrity protection,
- Authentication ?

1.4 Letting the CA sign the Server CSR

To make it simple, we assume sending of the certificate request has happened in a secure way (i.e., without man-in-the middle attacks or tampering in the transfer process). Thus, as previously stated in this lab we let the CA simply access the request via the VM filesystem.

Please be aware that this is not what would happen in reality! In particular, can you mention an important piece of private information that the server generated in the previous CSR and that the CA should not be aware of (but that could obtain by having access to the filesystem, as in this example?)

So go ahead importing and signing (I am not offending your intelligence by also reporting the signing command):

```
user@user-VirtualBox ~/CA/easy-rsa/easyrsa3 $ ./easyrsa import-req
~/server/easy-rsa/easyrsa3/pki/reqs/example.req  example-req
```

Now look at the signed certificate file (`~/CA/easy-rsa/easyrsa3/pki/issued/example-req.crt`). Particularly, look and comment choices EasyRSA3 has made about

- The public key information (bitwise size, modulus and exponent)
- The validity period of the certificate
- The signature algorithm and digest
- Inspect and read documentations about the X509v3 extensions you never heard before -- which source of information (that google will report you) SHOULD you use !!??

1.5 Letting the CA sign the Client CSR(s)

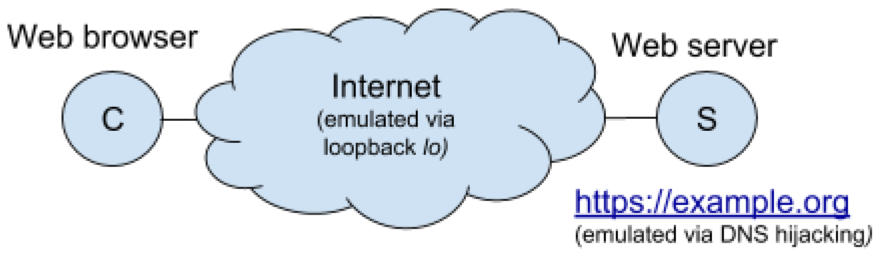
Now you have all that is needed to use the certificate for a Web browsing application (Part 2), but you still miss the client certificates for VPNs (Part 3). So please do the client certificates (points 3 - 7 of the PKI procedure in the OpenVPN EasyRSA3 tutorial) as well before moving to part2.

Instead, you can skip the last part (DH Generation) in the tutorial since that is used for VPNs in Part 3 of the Lab.

2. Certificates in Web browsing sessions

Goal: use the just generated state-of-the art certificates to secure browsing to your Website.

For the purpose of Part 2, we emulate a network in which a Web browser on client C want to access the public-facing Website of your example.org domain. Your server employs HTTPS (only TLS is enabled on the Website) and presents itself with the certificate created in Part 1.



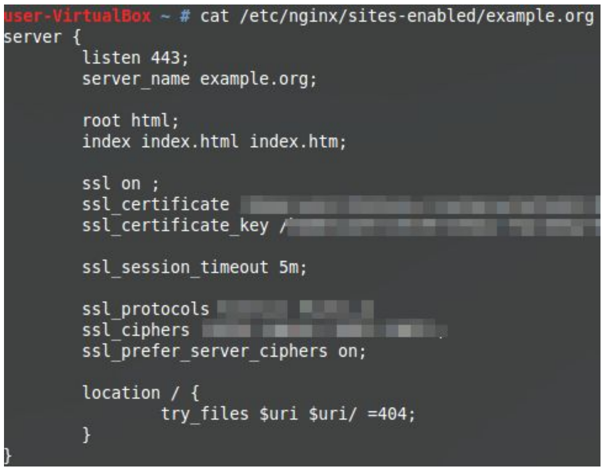
2.1 Configuring the Web server

To use the just created certificates, you need to install and configure a Web server. We will be using a `nginx` server for the purpose (which is also used by many among the most popular websites). Install the server and modify its default configuration for our website:

```
user-VirtualBox ~ # aptitude install nginx
user-VirtualBox ~ # tail -n 21 /etc/nginx/sites-enabled/default  | sed
's/^#//' > /etc/nginx/sites-enabled/example.org
```

Note that the tail "trick" may work or not depending on the verbosity of your system default (particularly maintainer of Linux-based distributions such as Ubuntu and Mint can change the defaults for one system).

However, `nginx` is very simple to configure so all the relevant configuration fits in about 10 lines. The whole configuration to solve this part of the Lab is shown (modulo some relevant parameters that are obfuscated) as in the example in the right:



```
user-VirtualBox ~ # cat /etc/nginx/sites-enabled/example.org
server {
        listen 443;
        server_name example.org;

        root html;
        index index.html index.htm;

        ssl on ;
        ssl_certificate [redacted]
        ssl_certificate_key /[redacted]

        ssl_session_timeout 5m;

        ssl_protocols [redacted]
        ssl_ciphers [redacted]
        ssl_prefer_server_ciphers on;

        location / {
                try_files $uri $uri/ =404;
        }
}
```

Note that so doing, you obtain only a templated configuration that you need to fill in with the actual values, particularly

- Which is the `ssl_certificate` ?
- Which is the `ssl_certificate_key` ?
- Do not neglect to configure the `ssl_protocols` and `ssl_ciphers`
- What simple yet dreadful attack your website would be exposed to if you also listened on port 80 and reconfigured to redirect to 443 ?

To start/stop nginx when you change the configuration, and to troubleshoot in case of errors, the following root (or sudo) commands are a useful starting point:

```
user-VirtualBox ~ # service stop nginx
user-VirtualBox ~ # service start nginx
user-VirtualBox ~ # less /var/log/nginx/error.log
```

2.1 Configuring the system

Now, we need to pretend we have an `example.org` domain up and running. This is needed since either we are using a domain name that exists (example.org case) or a domain that likely doesn't exist (fdf9dfdgmdglsdovcsmvsdifgs0dfg9sdklfgfdgfgfd.org).

In both cases, we want to be able to direct Web queries to this domain to our server. However in the first case the queries get destined to the original system, in the second system requests will be trashed. This can solved by simply hijacking the DNS resolution request. DNS hijacking can be achieved with a DNS pollution attack, or more simply by modifying the `/etc/hosts` file. In this way the `gethostbyname()` calls will be resolved from information contained in this local file instead than DNS.

```
user-VirtualBox ~ # cat /etc/hosts
127.0.0.1   localhost
127.0.1.1   user-VirtualBox
127.0.1.1   example.org
```

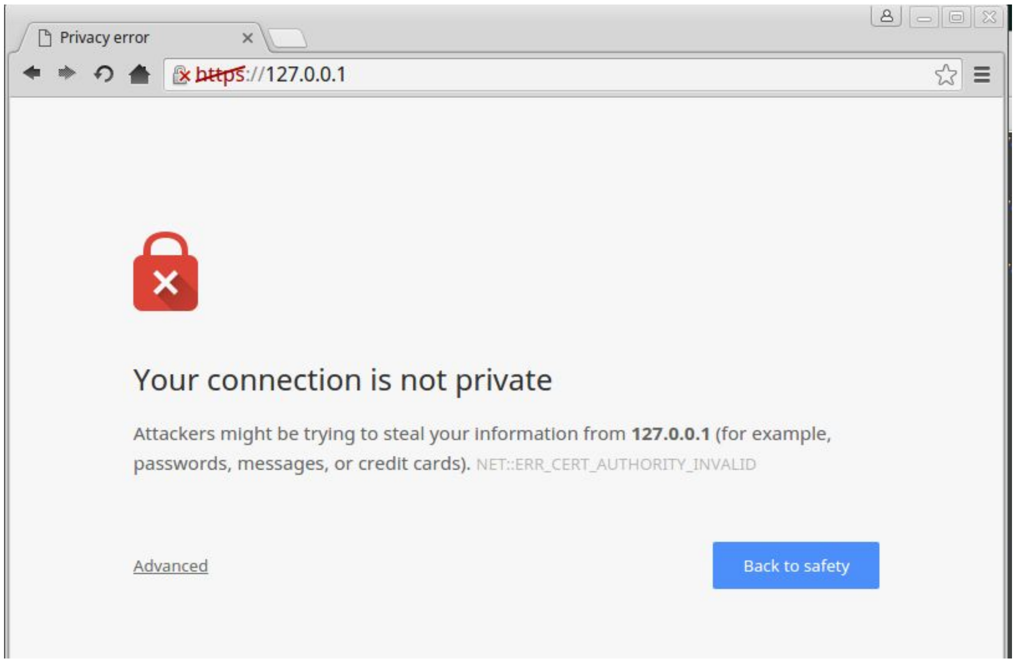
Note: alternative methods that could be used for the Lab exists. For instance `curl` supposedly support application-level resolution; however, in case you have problems (e.g., your server failed silently) then curl will attempt at redirecting example.org to 127.0.0.1, then fail (since your server failed) and direct you toward the real example.org (which is not what you want). So caveat emptor:

```
user@user-VirtualBox ~/CA/easy-rsa/easyrsa3 $ curl --resolve
example.org:443:127.0.0.1 example.org
```

2.2 Using certificates from the Web browser

Now you need to check that you can access the server with any application fully implementing the HTTP and TLS suites, e.g., Chrome (or Firefox, or…) or curl (handy HTTP implementation that is not a full browser). However, well-behaving HTTPS clients should complain about self-signed certificates:

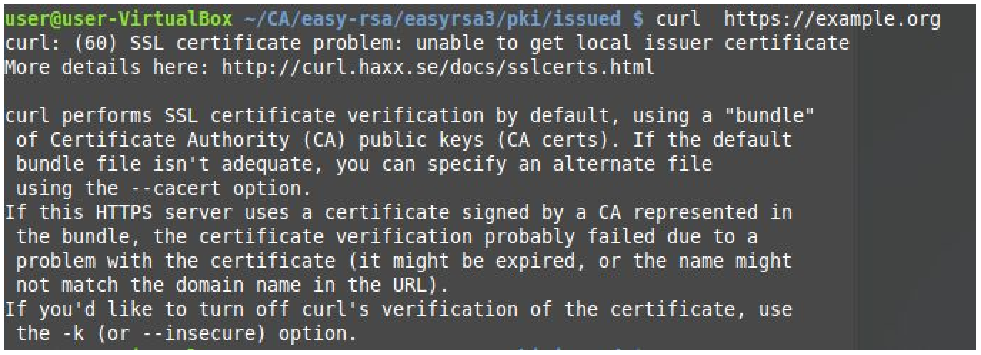
With Chrome:



Your connection is not private

Attackers might be trying to steal your information from **127.0.0.1** (for example, passwords, messages, or credit cards). NET::ERR_CERT_AUTHORITY_INVALID

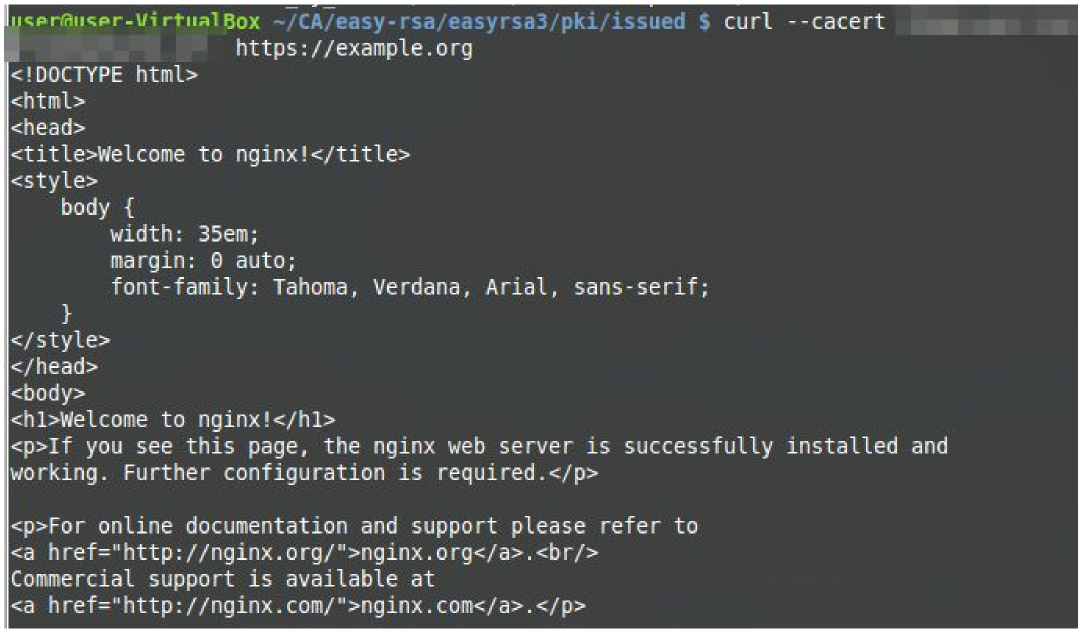
With curl:



```
user@user-VirtualBox ~/CA/easy-rsa/easyrsa3/pki/issued $ curl  https://example.org
curl: (60) SSL certificate problem: unable to get local issuer certificate
More details here: http://curl.haxx.se/docs/sslcerts.html

curl performs SSL certificate verification by default, using a "bundle"
 of Certificate Authority (CA) public keys (CA certs). If the default
 bundle file isn't adequate, you can specify an alternate file
 using the --cacert option.
If this HTTPS server uses a certificate signed by a CA represented in
 the bundle, the certificate verification probably failed due to a
 problem with the certificate (it might be expired, or the name might
 not match the domain name in the URL).
If you'd like to turn off curl's verification of the certificate, use
 the -k (or --insecure) option.
```

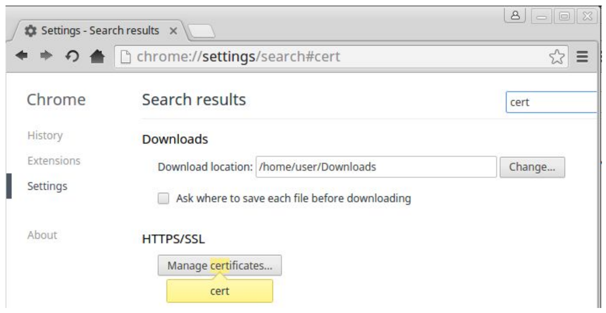
Thus, since the certificate is signed by a non-trusted CA, both browsers will refuse to connect to this suspicious website. Typically, frustrated users will click on Advanced and skip the certificate validation. However, RES212 students are technically competent and know this to be bad practice: therefore, in this lab we will go for a good practice and add the root to the set of trusted CAs.

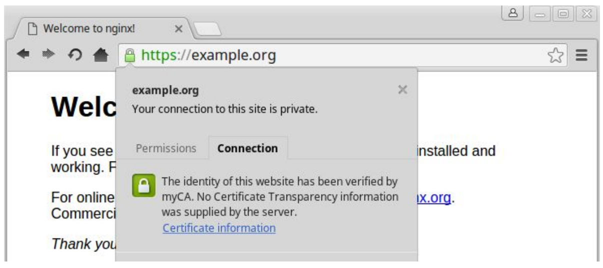
With curl, this is simply done by passing the certificate to the --cacert option from the command line:



```
user@user-VirtualBox ~/CA/easy-rsa/easyrsa3/pki/issued $ curl --cacert [redacted]
[redacted]                  https://example.org
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
```

From Chrome (or firefox), you can manage certificates from settings.


Settings - Search results
chrome://settings/search#cert
Chrome
History
Extensions
Settings
About
Search results
cert
Downloads
Download location: /home/user/Downloads
Change...
Ask where to save each file before downloading
HTTPS/SSL
Manage certificates...
cert



example.org
Your connection to this site is private.

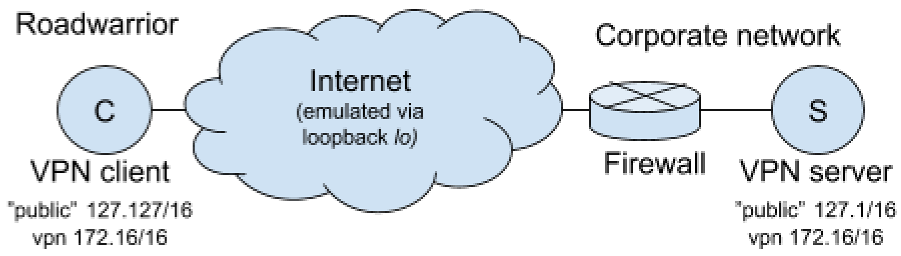
The identity of this website has been verified by myCA. No Certificate Transparency information was supplied by the server.

Now you should feel safe(r, but not safe. TLSv1.3 is not in the sever and browser yet!)

3. Certificates in Virtual Private Networks

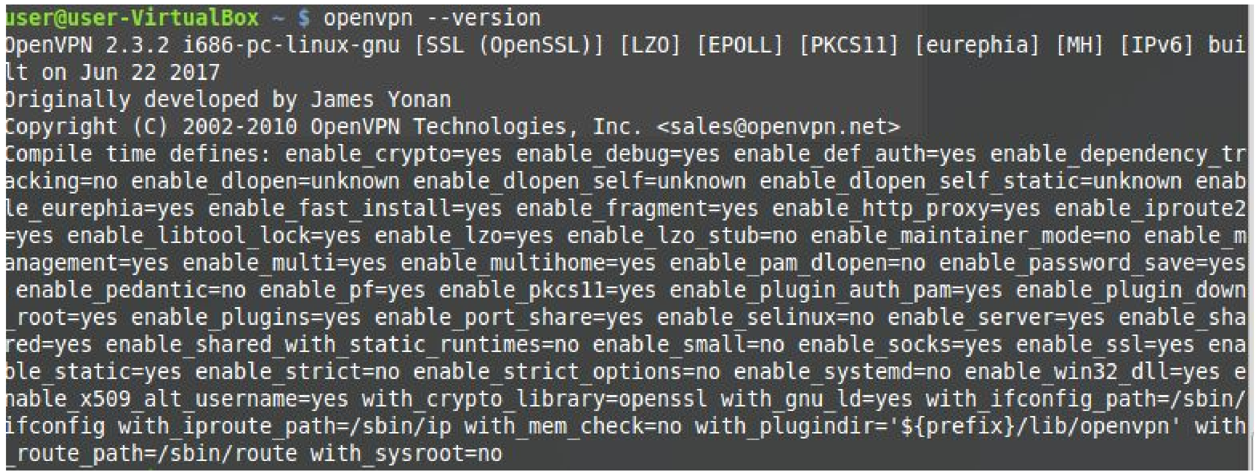
Goal: use the just generated state-of-the art certificates to extend network connectivity of a remote worker, providing him access to resources of the corporate network via a Virtual Private Network tunnel.

In this part of the lab, we will mostly deal with certificate part of setting up VPNs. Thus our setup will be overly simplistic as far as *networking* is concerned (initially, we developed the Lab over Netkit UML emulation which provides realistic network part, but with old and discouraged security practices in reason of old VPN software) but will up-to-date as far as cryptographic *security* mechanism are concerned.



3.1 Installing OpenVPN

OpenVPN is the most popular open-source VPN implementation. VMs are not equipped with OpenVPN but it shouldn't take long for you to install it:



```
user@user-VirtualBox ~ $ openvpn --version
OpenVPN 2.3.2 i686-pc-linux-gnu [SSL (OpenSSL)] [LZO] [EPOLL] [PKCS11] [eurephia] [MH] [IPv6] bui
lt on Jun 22 2017
Originally developed by James Yonan
Copyright (C) 2002-2010 OpenVPN Technologies, Inc. <sales@openvpn.net>
Compile time defines: enable_crypto=yes enable_debug=yes enable_def_auth=yes enable_dependency_tr
acking=no enable_dlopen=unknown enable_dlopen_self=unknown enable_dlopen_self_static=unknown enab
le_eurephia=yes enable_fast_install=yes enable_fragment=yes enable_http_proxy=yes enable_iproute2
=yes enable_libtool_lock=yes enable_lzo=yes enable_lzo_stub=no enable_maintainer_mode=no enable_m
anagement=yes enable_multi=yes enable_multihome=yes enable_pam_dlopen=no enable_password_save=yes
 enable_pedantic=no enable_pf=yes enable_pkcs11=yes enable_plugin_auth_pam=yes enable_plugin_down
_root=yes enable_plugins=yes enable_port_share=yes enable_selinux=no enable_server=yes enable_sha
red=yes enable_shared_with_static_runtimes=no enable_small=no enable_socks=yes enable_ssl=yes ena
ble_static=yes enable_strict=no enable_strict_options=no enable_systemd=no enable_win32_dll=yes e
nable_x509_alt_username=yes with_crypto_library=openssl with_gnu_ld=yes with_ifconfig_path=/sbin/
ifconfig with_iproute_path=/sbin/ip with_mem_check=no with_plugindir='${prefix}/lib/openvpn' with
_route_path=/sbin/route with_sysroot=no
```

Look at the compile time defines:

- In light of CRIME and BREACH attack to TLS, should `enable_lzo` worry you?
- List pros and cons of having `enable_static=yes`

3.2 Configuring OpenVPN

There are already tons of tutorials out there on configuring VPNs and OpenVPN in particular: caveat emptor, they tend to get update very quickly. A good idea is to start from documents referenced in the official documentation.

In particular, the official HOWTO is (more than) enough for our purposes.
https://openvpn.net/index.php/open-source/documentation/howto.html#examples

Note that (modulo DH setup) your certificates are ready, so you can start right away configuring the client and the server sides. Our setup is simplistic so we assume that 127.127/16 is the "public" address range of the VPN client and that 127.1/16 is the "public" address range of the VPN server. As before, we use the convention of administering in separate folders (`~/cvpn` and `~/svpn`) to imply we administer separate systems.

In particular, the example configuration described in the tutorial is already available in your system at `/usr/share/doc/openvpn/examples/sample-config-files/` (after installation of OpenVPN). Start from copying the `client.conf` to `~/cvpn` and `gunzip` the `server.conf.gz to ~/svpn.`

- Edit the relevant configuration parts, such as (but not limited to) those reported here, in alphabetical order.
- Which default cryptographic choices you find safe? Which ones debatable? and which ones broken?

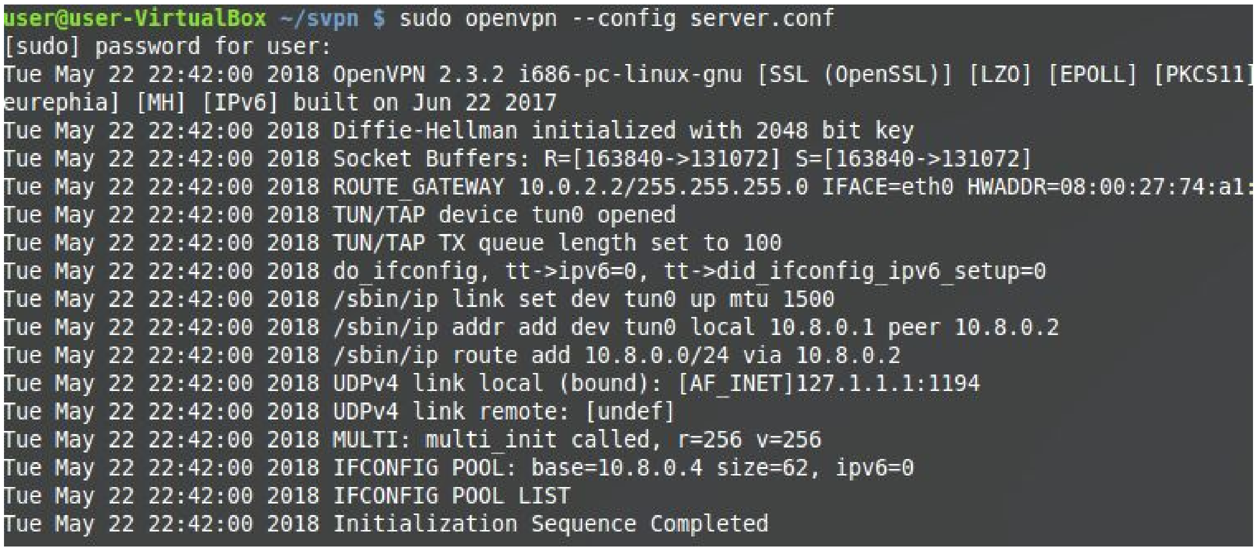
Client	Server
`ca`	`ca`
`cert`	`cert`
`cipher`	`cipher`
`key`	`dh`
`remote`	`key`
`verb`	`verb`

Remind that more interesting configuration options would be available in case we were emulating a richer network setup (eg. route `push` in particular, but not only)

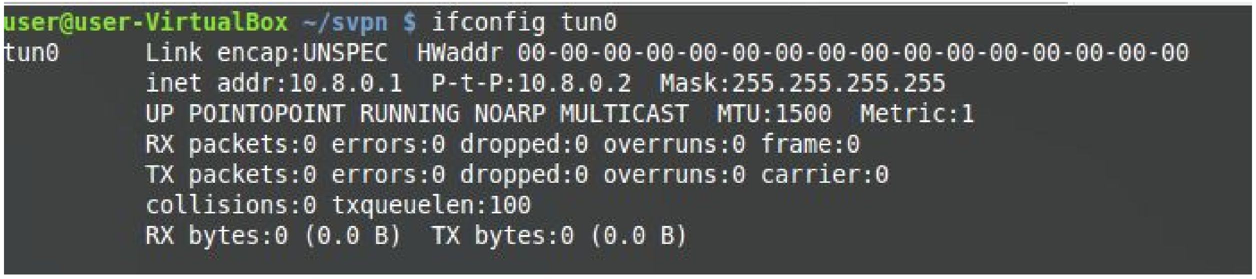
Note: check the attributes of the `server.key (ls -l).` Do they comply with the system security recommendations expressed in `server.conf` ?

3.3 Testing OpenVPN handshake

Once VPN client and server are configured, start the server, verify its initialization successfully completed and check that the virtual VPN device (tun0) has been created

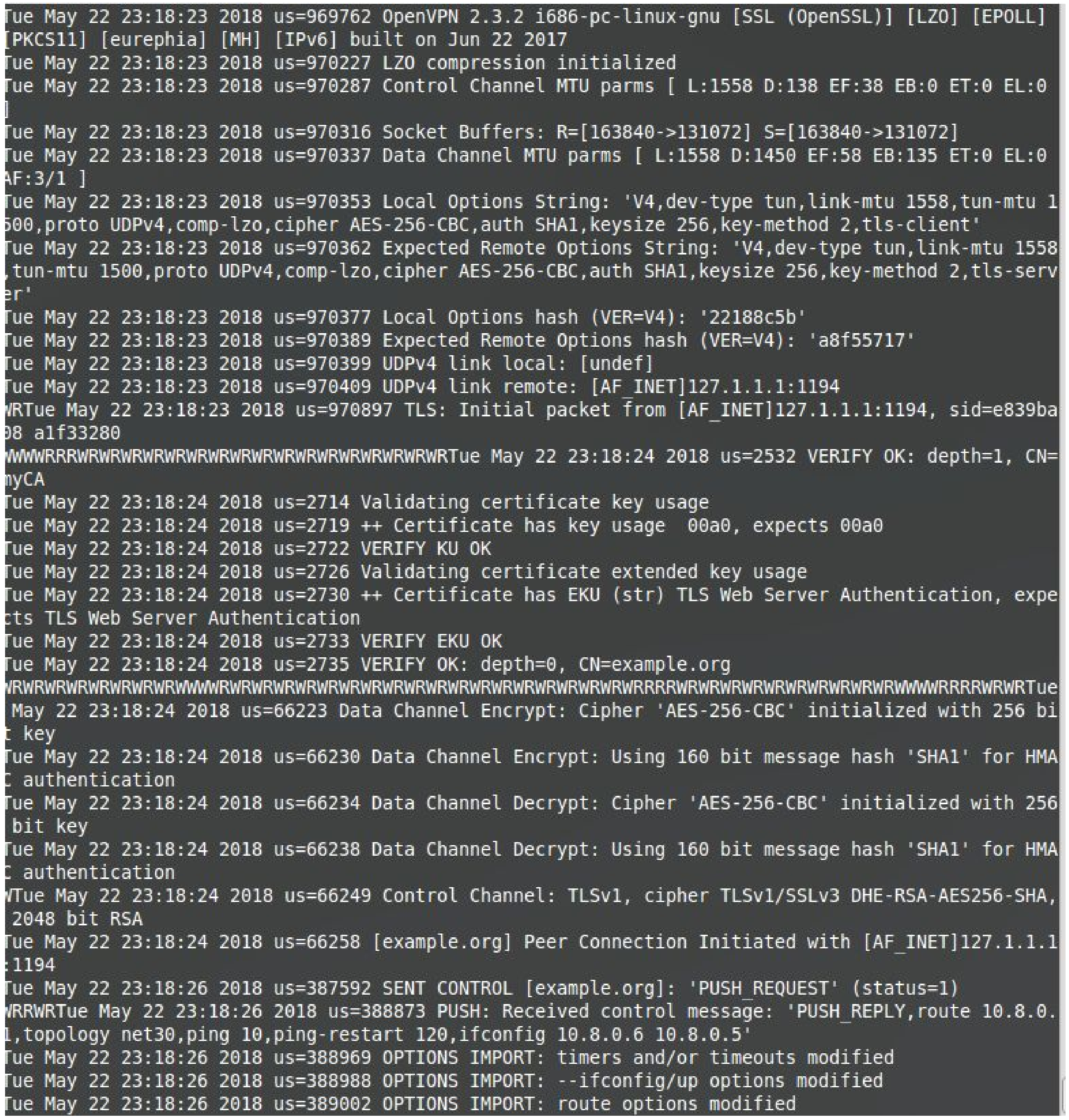


```
user@user-VirtualBox ~/svpn $ sudo openvpn --config server.conf
[sudo] password for user:
Tue May 22 22:42:00 2018 OpenVPN 2.3.2 i686-pc-linux-gnu [SSL (OpenSSL)] [LZO] [EPOLL] [PKCS11]
eurephia] [MH] [IPv6] built on Jun 22 2017
Tue May 22 22:42:00 2018 Diffie-Hellman initialized with 2048 bit key
Tue May 22 22:42:00 2018 Socket Buffers: R=[163840->131072] S=[163840->131072]
Tue May 22 22:42:00 2018 ROUTE_GATEWAY 10.0.2.2/255.255.255.0 IFACE=eth0 HWADDR=08:00:27:74:a1:
Tue May 22 22:42:00 2018 TUN/TAP device tun0 opened
Tue May 22 22:42:00 2018 TUN/TAP TX queue length set to 100
Tue May 22 22:42:00 2018 do_ifconfig, tt->ipv6=0, tt->did_ifconfig_ipv6_setup=0
Tue May 22 22:42:00 2018 /sbin/ip link set dev tun0 up mtu 1500
Tue May 22 22:42:00 2018 /sbin/ip addr add dev tun0 local 10.8.0.1 peer 10.8.0.2
Tue May 22 22:42:00 2018 /sbin/ip route add 10.8.0.0/24 via 10.8.0.2
Tue May 22 22:42:00 2018 UDPv4 link local (bound): [AF_INET]127.1.1.1:1194
Tue May 22 22:42:00 2018 UDPv4 link remote: [undef]
Tue May 22 22:42:00 2018 MULTI: multi_init called, r=256 v=256
Tue May 22 22:42:00 2018 IFCONFIG POOL: base=10.8.0.4 size=62, ipv6=0
Tue May 22 22:42:00 2018 IFCONFIG POOL LIST
Tue May 22 22:42:00 2018 Initialization Sequence Completed
```



```
user@user-VirtualBox ~/svpn $ ifconfig tun0
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:10.8.0.1  P-t-P:10.8.0.2  Mask:255.255.255.255
          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:100
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Do the same on the client to verify they could talk (but since we have no servers… they have nothing to say actually).



```
Tue May 22 23:18:23 2018 us=969762 OpenVPN 2.3.2 i686-pc-linux-gnu [SSL (OpenSSL)] [LZO] [EPOLL]
[PKCS11] [eurephia] [MH] [IPv6] built on Jun 22 2017
Tue May 22 23:18:23 2018 us=970227 LZO compression initialized
Tue May 22 23:18:23 2018 us=970287 Control Channel MTU parms [ L:1558 D:138 EF:38 EB:0 ET:0 EL:0
]
Tue May 22 23:18:23 2018 us=970316 Socket Buffers: R=[163840->131072] S=[163840->131072]
Tue May 22 23:18:23 2018 us=970337 Data Channel MTU parms [ L:1558 D:1450 EF:58 EB:135 ET:0 EL:0
AF:3/1 ]
Tue May 22 23:18:23 2018 us=970353 Local Options String: 'V4,dev-type tun,link-mtu 1558,tun-mtu 1
500,proto UDPv4,comp-lzo,cipher AES-256-CBC,auth SHA1,keysize 256,key-method 2,tls-client'
Tue May 22 23:18:23 2018 us=970362 Expected Remote Options String: 'V4,dev-type tun,link-mtu 1558
,tun-mtu 1500,proto UDPv4,comp-lzo,cipher AES-256-CBC,auth SHA1,keysize 256,key-method 2,tls-serv
er'
Tue May 22 23:18:23 2018 us=970377 Local Options hash (VER=V4): '22188c5b'
Tue May 22 23:18:23 2018 us=970389 Expected Remote Options hash (VER=V4): 'a8f55717'
Tue May 22 23:18:23 2018 us=970399 UDPv4 link local: [undef]
Tue May 22 23:18:23 2018 us=970409 UDPv4 link remote: [AF_INET]127.1.1.1:1194
WRTue May 22 23:18:23 2018 us=970897 TLS: Initial packet from [AF_INET]127.1.1.1:1194, sid=e839ba
08 a1f33280
WWWWRRRWRWRWRWRWRWRWRWRWRWRWRWRWRWRWRWRWRWRWRWRTue May 22 23:18:24 2018 us=2532 VERIFY OK: depth=1, CN=
myCA
Tue May 22 23:18:24 2018 us=2714 Validating certificate key usage
Tue May 22 23:18:24 2018 us=2719 ++ Certificate has key usage  00a0, expects 00a0
Tue May 22 23:18:24 2018 us=2722 VERIFY KU OK
Tue May 22 23:18:24 2018 us=2726 Validating certificate extended key usage
Tue May 22 23:18:24 2018 us=2730 ++ Certificate has EKU (str) TLS Web Server Authentication, expe
cts TLS Web Server Authentication
Tue May 22 23:18:24 2018 us=2733 VERIFY EKU OK
Tue May 22 23:18:24 2018 us=2735 VERIFY OK: depth=0, CN=example.org
WRWRWRWRWRWRWRWRWWWWRWRWRWRWRWRWRWRWRWRWRWRWRWRWRWRWRWRWRRRRWRWRWRWRWRWRWRWRWRWRWWWWRRRRWRWRTue
 May 22 23:18:24 2018 us=66223 Data Channel Encrypt: Cipher 'AES-256-CBC' initialized with 256 bi
t key
Tue May 22 23:18:24 2018 us=66230 Data Channel Encrypt: Using 160 bit message hash 'SHA1' for HMA
C authentication
Tue May 22 23:18:24 2018 us=66234 Data Channel Decrypt: Cipher 'AES-256-CBC' initialized with 256
 bit key
Tue May 22 23:18:24 2018 us=66238 Data Channel Decrypt: Using 160 bit message hash 'SHA1' for HMA
C authentication
WTue May 22 23:18:24 2018 us=66249 Control Channel: TLSv1, cipher TLSv1/SSLv3 DHE-RSA-AES256-SHA,
 2048 bit RSA
Tue May 22 23:18:24 2018 us=66258 [example.org] Peer Connection Initiated with [AF_INET]127.1.1.1
:1194
Tue May 22 23:18:26 2018 us=387592 SENT CONTROL [example.org]: 'PUSH_REQUEST' (status=1)
WRRWRTue May 22 23:18:26 2018 us=388873 PUSH: Received control message: 'PUSH_REPLY,route 10.8.0.
1,topology net30,ping 10,ping-restart 120,ifconfig 10.8.0.6 10.8.0.5'
Tue May 22 23:18:26 2018 us=388969 OPTIONS IMPORT: timers and/or timeouts modified
Tue May 22 23:18:26 2018 us=388988 OPTIONS IMPORT: --ifconfig/up options modified
Tue May 22 23:18:26 2018 us=389002 OPTIONS IMPORT: route options modified
```

Check the client peer connection is initiated and the client initialization sequence is successfully completed

- Given recent SHA-1 collision, are you comfortable with using SHA-1 ?
- What was the default (and what is your selected) encryption cipher?

3.4 Testing OpenVPN connectivity

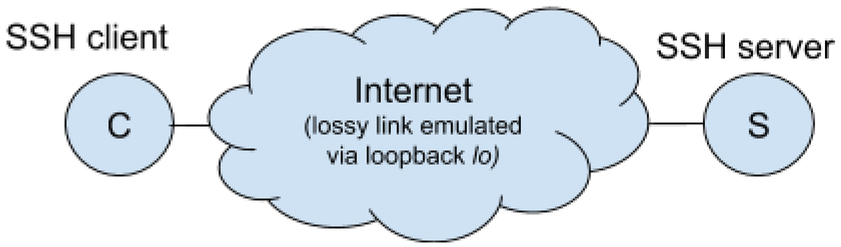
This would require a much more complex emulation environment. Setting this up is entirely in the possibility of evolution of this lab. However, that depends strictly on how much time was needed to finish the previous parts of the Lab!

Please, note down how long did it take you to succesfully complete previous assignments of this TP. You will be asked to provide this estimation in the Feedback at the end of the course. Thanks!

Also, note that a more complex setup would mean that each action in the previous part would take longer. For instance, to create and sign CSR, you would actually need to move data around multiple systems, in a secure way. This would require to setup, e.g., an SSH server on the CA. Keep in mind that the tradeoff between realism and complexity applies here.

4. Why TCP VPNs are a bad idea

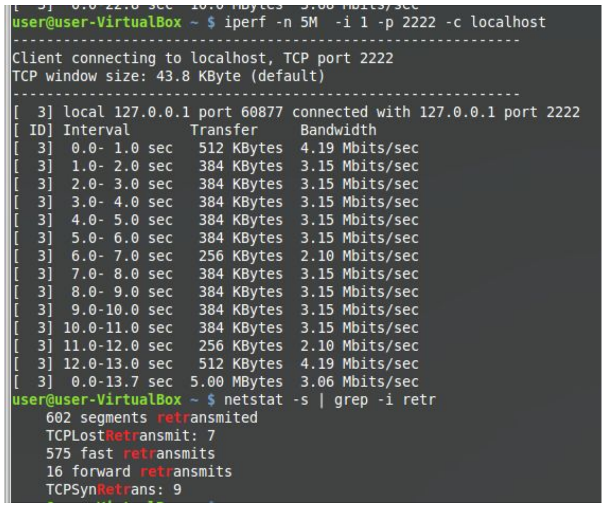
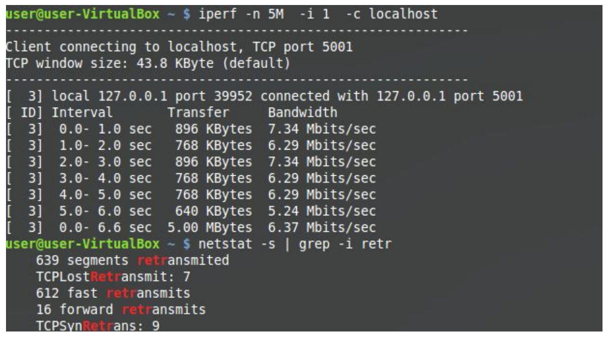
Goal: understand that performance implication can have design decision on the security architecture. It is possible that time runs out to perform this experiment. Ideally, this experiment should have been done as a demo during the lectures.



Students having followed RES203 have all instruments to setup a link with limited capacity (say 10Mbps) and moderate losses (say 1%) and delay (10ms). To measure *TCP efficiency*, one can then measure throughput of a TCP connection transferring 5MB from C to S (`iperf`), and measure the corresponding TCP retransmissions statistics (`netstat -s | grep -i retr`).

To measure *VPN TCP-over-TCP inefficiency*, one can then

- Open an SSH tunnel from C to S (install openssh, then `ssh -L2222:localhost:5001 -N localhost`)
- Open an iperf connection that will be tunneled through the SSH connection (`iperf -p 2222 localhost`)
- Measure the iperf throughput and TCP retransmissions statistics in this second case.

TCP-over-TCP with lossy link	TCP with lossy link
	 Netstat statistics are cumulative and this experiment was second: so only 37 segments were retransmitted in this case vs several more in the other case (however counters were not starting from 0)

```
user@user-VirtualBox ~ $ iperf -n 5M  -i 1 -p 2222 -c localhost
------------------------------------------------------------
Client connecting to localhost, TCP port 2222
TCP window size: 43.8 KByte (default)
------------------------------------------------------------
[  3] local 127.0.0.1 port 60877 connected with 127.0.0.1 port 2222
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec   512 KBytes  4.19 Mbits/sec
[  3]  1.0- 2.0 sec   384 KBytes  3.15 Mbits/sec
[  3]  2.0- 3.0 sec   384 KBytes  3.15 Mbits/sec
[  3]  3.0- 4.0 sec   384 KBytes  3.15 Mbits/sec
[  3]  4.0- 5.0 sec   384 KBytes  3.15 Mbits/sec
[  3]  5.0- 6.0 sec   384 KBytes  3.15 Mbits/sec
[  3]  6.0- 7.0 sec   256 KBytes  2.10 Mbits/sec
[  3]  7.0- 8.0 sec   384 KBytes  3.15 Mbits/sec
[  3]  8.0- 9.0 sec   384 KBytes  3.15 Mbits/sec
[  3]  9.0-10.0 sec   384 KBytes  3.15 Mbits/sec
[  3] 10.0-11.0 sec   384 KBytes  3.15 Mbits/sec
[  3] 11.0-12.0 sec   256 KBytes  2.10 Mbits/sec
[  3] 12.0-13.0 sec   512 KBytes  4.19 Mbits/sec
[  3]  0.0-13.7 sec  5.00 MBytes  3.06 Mbits/sec
user@user-VirtualBox ~ $ netstat -s | grep -i retr
    602 segments retransmited
    TCPLostRetransmit: 7
    575 fast retransmits
    16 forward retransmits
    TCPSynRetrans: 9
```

```
user@user-VirtualBox ~ $ iperf -n 5M  -i 1  -c localhost
------------------------------------------------------------
Client connecting to localhost, TCP port 5001
TCP window size: 43.8 KByte (default)
------------------------------------------------------------
[  3] local 127.0.0.1 port 39952 connected with 127.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec   896 KBytes  7.34 Mbits/sec
[  3]  1.0- 2.0 sec   768 KBytes  6.29 Mbits/sec
[  3]  2.0- 3.0 sec   896 KBytes  7.34 Mbits/sec
[  3]  3.0- 4.0 sec   768 KBytes  6.29 Mbits/sec
[  3]  4.0- 5.0 sec   768 KBytes  6.29 Mbits/sec
[  3]  5.0- 6.0 sec   640 KBytes  5.24 Mbits/sec
[  3]  0.0- 6.6 sec  5.00 MBytes  6.37 Mbits/sec
user@user-VirtualBox ~ $ netstat -s | grep -i retr
    639 segments retransmited
    TCPLostRetransmit: 7
    612 fast retransmits
    16 forward retransmits
    TCPSynRetrans: 9
```