Security in Computer Networks (Lab6)

Contents

Introduction

The goal of this lab is to understand symmetric and public-key encryption by setting up and analyzing TLS/SSL for secure TCP connections, as well as setting up and analyzing VPN for secure network tunneling. An additional goal is to understand the principles of sender authentication and message integrity by applying and verifying digital signatures. The lab has several **milestones**. Make sure you reach each one before advancing to the next.

For delivery, submit a PDF report where you answer **only** those steps that are marked with **REPORT(%)**. Additionally, you should submit the codes or capture files, if they were **explicitly** asked for. The percent point gives you an indication of the score of that question. Lab6 counts for **6 points** of your final score in this course.

> *HINT: The initial build of lab6 takes quite a moment, thus we recommend to run it in advance. Use that time to read through the "docker-compose.yml" file to get a deeper understanding of the system setup. Do not forget to run* `docker-compose down` *of previous labs before building this lab.*

System Setup

In this lab, you will use a system setup as shown in Figure 1. It is similar to lab5, but the servers now have private IP addresses, which can not be accessed from external networks unless through a VPN or NAT port forwarding. Your job is to first access the "webserver" from external networks using port forwarding and to setup a website with TLS/SSL support. Then disable port forwarding and setup a VPN.

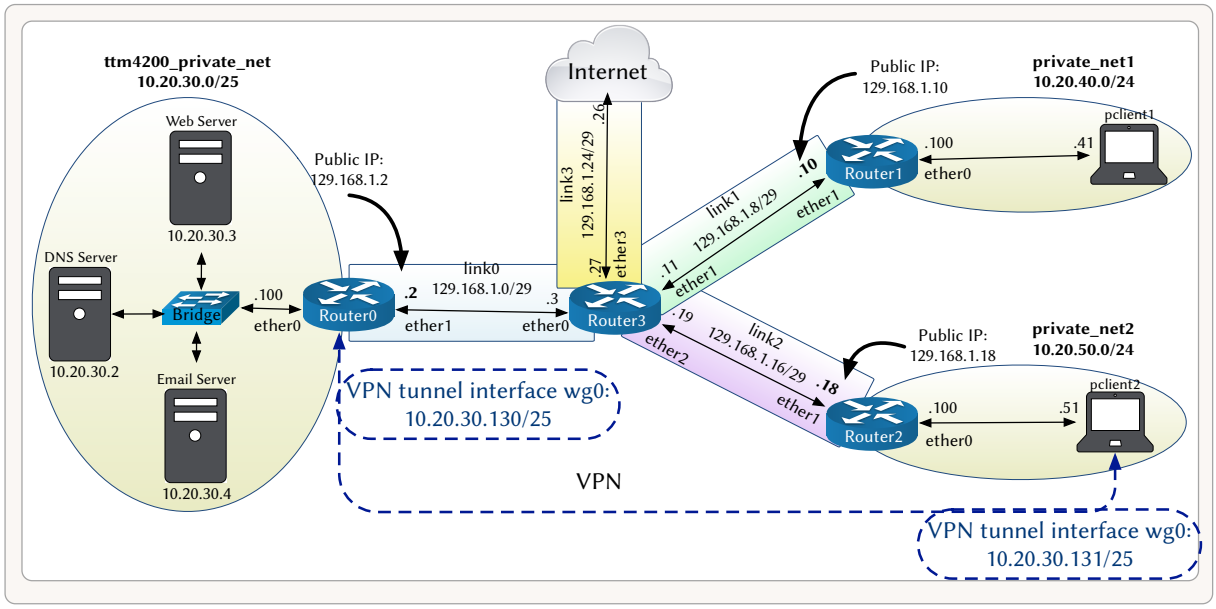

Figure 1: System Setup for Lab6

- To copy files from or to containers, use `scp` or the mounted volumes. Read through the "docker-compose.yml" to find out which directories are mounted.
- NAT is already configured in "router1" and "router2" (as in lab5).

1 Milestone 1 – Securing TCP Connection: TLS/SSL

In this section, you will setup a secure TCP connection using Transport Layer Security(TLS) commonly confused with its predecessor Secure Sockets Layer (SSL), which is nowadays considered as deprecated. Then you will analyze a trace of TLS/SSL records to investigate the various records types and the fields in these records.

As an example, you will set up a web server that supports TLS (HTTPS: HTTP over TLS). Then investigate how cryptography enhances HTTP with security (confidentiality, data integrity, and end-point authentication).

1.1 Port Forwarding

To allow access to certain services residing on a private network, we can use port forwarding to remap the public IP and port to the private IP address and port of that service.

- In **"pclient2"**, verify that you can **not** access the webserver (neither through the private IP of "webserver" nor through the public IP or "router0"):

```
wget http://10.20.30.3 --header "Host: www.ttm4200.com"
wget http://129.168.1.2 --header "Host: www.ttm4200.com"
```

- In **"router0"**, setup port forwarding for HTTP (port number = 80) and HTTPS (port number = 443):

```
#Accept new connections destined for ports 80,443:
sudo iptables -A FORWARD -i ether1 -o ether0 -p tcp --syn --match multiport \
   --dports 80,443 -m conntrack --ctstate NEW -j ACCEPT
#Allow any subsequent traffic in both directions:
sudo iptables -A FORWARD -i ether1 -o ether0 -m conntrack --ctstate \
   ESTABLISHED,RELATED -j ACCEPT
sudo iptables -A FORWARD -i ether0 -o ether1 -m conntrack --ctstate \
   ESTABLISHED,RELATED -j ACCEPT
#Alters destination address before routing:
sudo iptables -t nat -A PREROUTING -i ether1 -p tcp --match multiport --dports \
   80,443 -j DNAT --to-destination 10.20.30.3
#Modify source address after routing:
sudo iptables -t nat -A POSTROUTING -o ether0 -p tcp --match multiport --dports \
   80,443 -d 10.20.30.3 -j SNAT --to-source 10.20.30.100
```

- In **"pclient2"**, verify that you can access the webserver by retrieving the content of its HTML page:

```
wget http://129.168.1.2 --header "Host: www.ttm4200.com"
```

1.2 Eavesdropping Insecure Connection

- The "webserver" is set up with *nginx* but unsecured without TLS. Thus any established connection is susceptible to eavesdropping and tampering by an outside party. To verify that, you can start packet capturing on **"router3"** (as the outside party, or "Trudy") on **"ether0"** interface:
- In **"pclient2"**, retrieve the content from the webserver and then stop packet capturing.

Q1. **REPORT(3%):** Open the captured file in Wireshark and find the HTTP response (HTTP/1.1 200 OK) sent from the "webserver" to "pclient2". Can you read the response in cleartext (line-based text data)? Provide an annotated screenshot.

- Additionally, you can check that a web server is not secure through a browser, as in Figure 2. To do it add the following command to your **VM**, so you can access "www.ttm4200.com" through the web-browser in your host (e.g. Firefox):

```
sudo sh -c 'echo "127.0.0.1 www.ttm4200.com" >> /etc/hosts'
```

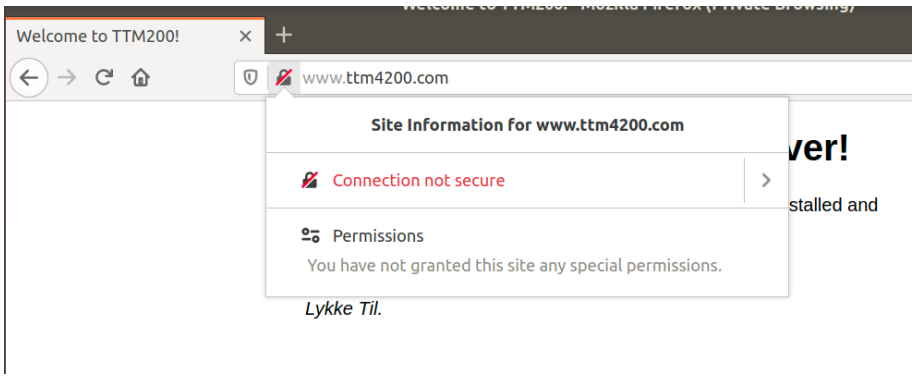

Figure 2: Figure2: Insecure TCP connection

1.3 Creating TLS/SSL Certificate

The first step in setting up a website with TLS/SSL support is to get a digital certificate signed by a certification authority(CA) (https://en.wikipedia.org/wiki/Certificate_authority). However, we can create a Self-Signed SSL Certificate using OpenSSL (https://www.openssl.org/).

> **NOTE:** Using OpenSSL to generate certificates doesn't mean we are using the deprecated SSL protocol. In fact, we can generate *certificates* that can be used by both SSL and TLS *protocols*.

- First, in the **"webserver"**, generate the server's private key with the following command:

```
sudo openssl genrsa -out /etc/ssl/private/ttm4200-selfsigned.key 2048
```

  This will generate an RSA key, 2048 bits long. This key is required to sign the certificate. Typically, a certificate is signed by a certificate authority (CA) using CA's private key, which will allow our browsers and other equipment to trust such keys (recall the importance of CAs in page 647 of the book). However, in our case, we will use the private key to sign the certificate.

  > **Extra:** If you are interested in freely generating a worldwide valid key you may try Let's Encrypt. However, take care that some of the steps may differ.

- Now you will generate the SSL/TLS certificate. To do it simply run the following command and provide the public information that will be embedded in the certificate. Feel free to change the information except for the Common Name, which must correspond to the server's domain name.

```
sudo openssl req -x509 -sha256 -nodes -days 60 \
   -key /etc/ssl/private/ttm4200-selfsigned.key \
   -out /etc/ssl/certs/ttm4200-selfsigned.crt

Country Name (2 letter code) [AU]:NO
State or Province Name (full name) [Some-State]:Trøndelag
Locality Name (eg, city) []:Trondheim
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NTNU
Organizational Unit Name (eg, section) []:IIK
Common Name (e.g. server FQDN or YOUR name) []:www.ttm4200.com
Email Address []:murad@ttm4200.com
```

  > More information about the command:
  >
  > **-x509**: define the format of public-key certificates. (https://en.wikipedia.org/wiki/X.509)
  >
  > **-sha256**: use stronger hash algorithm
  >
  > **-nodes**: skip securing the certificate with a passphrase.
  >
  > **-days** 60: specify the validity of the certificate
  >
  > **-key**: specify the private key to sign this certificate
  >
  > **-out**: specify the path and name of the certificate.
  >
  > **-Emai Address**: Use an email address with your team number in it (e.g. alex@team10A.com). We will check this in the capture file to prove that your team generated the certificate.

- Create a Diffie–Hellman (D-H) Parameter:

```
sudo openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048
```

> *"Using Diffie–Hellman key exchange will generate a random and unique session key for encryption and decryption that has the additional property of forward secrecy: if the server's private key is disclosed in future, it cannot be used to decrypt the current session, even if the session is intercepted and recorded by a third party."* (https://en.wikipedia.org/wiki/Transport_Layer_Security)

1.4 Configuring the Web Server to Support TLS/SSL

The *nginx* service installed in the "webserver" is configured at "/etc/nginx/site-available/ttm4200" to support HTTP. You need to adjust the configuration to use TLS/SSL and redirect HTTP requests to HTTPS, thus enforcing the use of the TLS/SSL certificate.

- Start by filling the following template. Keep in mind that HTTPS uses a different port number than HTTP (http://nginx.org/en/docs/http/configuring_https_servers.html).

```
server {
      #enable ssl on listening socket (IPv4, default_server)
      listen ====fill in here====;

      #enable ssl on listening socket (IPv6, default_server)
      listen ====fill in here====;

      #location of the server certificate
      ssl_certificate ====fill in here====;

      #location of the private key
      ssl_certificate_key ====fill in here====;

      #location of the dh parameter
      ssl_dhparam ====fill in here====;

      root /var/www/ttm4200;
      index ttm4200_index.html;
      server_name www.ttm4200.com;
      location / {
         try_files $uri $uri/ =404;
         }
      error_page 500 502 503 504 /50x.html;
      location = /50x.html {
            root /var/www/ttm4200;
         index 50x.html;
         }
   }

# catch-all http requests and redirect them to https
server{
```

```
        listen 80 default_server;
        listen [::]:80 default_server;
        server_name www.ttm4200.com;
        return 301 https://www.ttm4200.com$request_uri;
        }
```

- Check that the configuration's syntax (`sudo nginx -t`) and if so restart *nginx*.
- In your VM, access the website through a browser. Because we are using a self-signed certificate that is not signed by any trusted CA, the browser will be unable to verify the certificate presented by the webserver and will issue a warning, as in Figure 3.

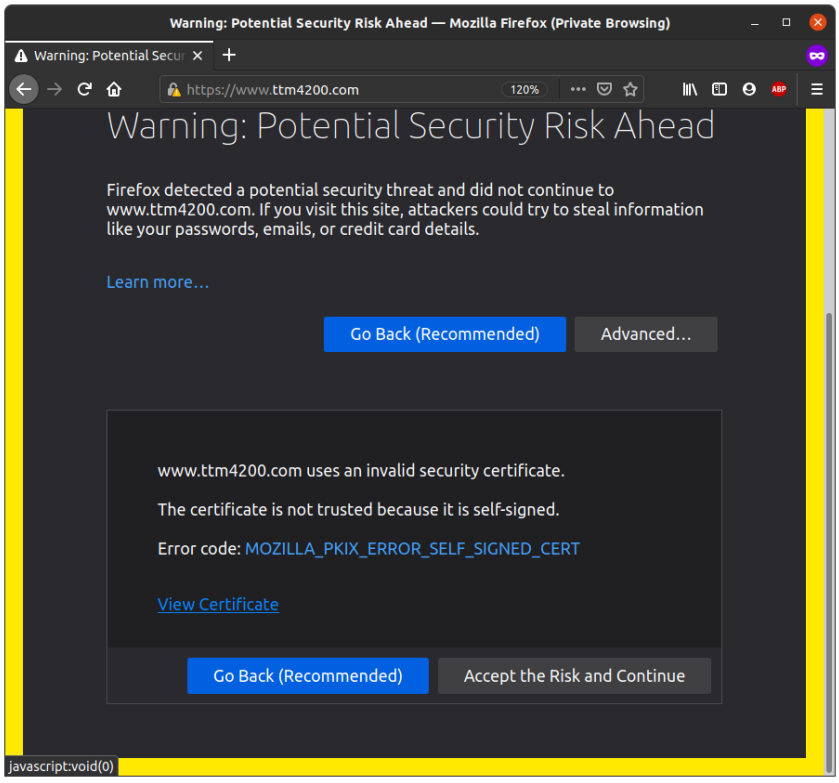

Figure 3: Figure3: Self-signed certificate warning

- Click on *View Certificate* to display certificate information. Answer the following and validate your answer with an annotated screenshot.

Q2. **REPORT(2%):** Why are the "Subject Name" and "Issuer Name" the same?

Q3. **REPORT(2%):** What is the algorithm used for public-key encryption?

Q4. **REPORT(2%):** What is the signature algorithm?

Q5. **REPORT(2%):** What is the hash function used to produce the certificate fingerprints?

> **Note:** If you already pressed *Accept the Risk and Continue* you can still view the certificate by clicking on the lock on the address bar.

1.5 Analyzing TLS/SSL protocol.

To analyze TLS/SSL, you need to capture packets in a TLS/SSL session.

- Start packet capturing in **"router3"** on "ether0" interface:

```
sudo tcpdump -i ether0 -w https_tls.pcap
```

- In **"pclient2"**, retrieve the content from the "webserver" using HTTPS and skip the certificate verification:

```
wget https://129.168.1.2 --header "Host: www.ttm4200.com" --no-check-certificate
```

- Stop packet capturing in "router3". Open the capture file in Wireshark and a display filter (`tls` or in some Wireshark versions `ssl`) to show only the frames that have TLS/SSL records. Then answer the following questions and validate your answer with an **annotated** screenshot.

> *NOTE: Keep in mind that a single Ethernet frame may contain one or more TLS/SSL records.*

Q6. **REPORT(2%):** Locate the first message in the TLS handshake (*ClientHello* record), which is sent by the client to initiate a session with the server. What is the value of the content type?

Q7. **REPORT(4%):** Locate the nonce in the *ClientHello* record (also known as a "Random"), which consists of a random number and a Unix timestamp.How long is it? Can you explain how it can defend against the "connection replay attacks", even though it is just a random number? What is the functionality of including the timestamp in the nonce?

Q8. **REPORT(4%):** Locate the cipher suite list in the *ClientHello* record, which contains the combinations of cryptographic algorithms supported by the client in order of the client's preference (favorite choice first). How many cipher suites are advertised in the Client Hello record? Select one suite, what is the public-key algorithm, the symmetric-key algorithm, and the hash algorithm?

Q9. **REPORT(4%):** Locate the *ServerHello* record and expand it. Find the chosen cipher suite (the server-chosen cipher suite from the client's advertised list) and specify the selected algorithms (public-key algorithm, the symmetric-key algorithm, and the hash algorithm).

Q10. **REPORT(4%):** Locate the Certificate record and expand it. How long is the certificate (in bytes)? Does the certificate fit into a single Ethernet frame? Expand the certificate and find the issuer and subject information. Are these entries the same as those you used when creating the certificate?

Q11. **REPORT(4%):** Which record contains a pre-master secret? What is this secret used for?

> *HINT: https://security.stackexchange.com/questions/63971/how-is-the-premaster-secret-used-in-tls-generated#answer-138559*

Q12. **REPORT(4%):** How is the application data being encrypted? Do the records containing application data include a Message Authentication Code (MAC)? Does Wireshark distinguish between the encrypted application data and the MAC?

Q13. **REPORT(3%):** Compare the content of application data to the content of HTTP (section 1.2, Q1). What is the most obvious difference?

Q14. **REPORT(5%):** Submit your capture files "http.pcap" and "https_tls.pcap" along with the report. We will check these files and make sure that they are unique and **consistent with to your answers to the previous questions**.

2 Milestone 2 – Virtual Private Network (VPN)

To establish a protected network over public internet connection, we can setup a VPN to encrypt the traffic and allow access to services residing on private networks. For example, the "Openstack" VMs are only accessible via NTNU private network, and you had to setup a VPN connection to access them.

In this lab, we will use WireGuard, which is free and open source VPN software, yet it is faster, simpler and more efficient than IPsec and OpenVPN. It is already installed in all containers of this lab. You **must** read this page (https://www.wireguard.com/) to get a conceptual overview of WireGuard.

- First, remove port forwarding from **"router0"** to prevent external access to internal resources in "ttm4200_private_net":

```
sudo iptables --table filter --flush
sudo iptables --table nat --flush
```

- In **"pclient2"**, verify that you can **not** access the webserver:

```
wget http://129.168.1.2 --header "Host: www.ttm4200.com"
wget http://10.20.30.3 --header "Host: www.ttm4200.com"
```

2.1 VPN Setup: WireGuard

In the following steps, we will establish a VPN connection from **"pclient2"** (WireGuard client) to **"router0"** (WireGuard server):

- Generate public and private keys for both the sever (**"router0"**) and the client (**"pclient2"**) (i.e. these commands need to be executed in both containers):

```
umask 077
wg genkey | tee privatekey | wg pubkey > publickey
```

> More information about the command (https://man7.org/linux/man-pages/man8/wg.8.html):
>
> **- umask 077**: Set file permission for newly created files to only the user has read, write and execute permissions (completely private).
>
> **- wg genkey**: Generates a random private key in base64.
>
> **- | tee privatekey**: Read the output of the generated private key and write it to a file, named "privatekey" .
>
> **- | wg pubkey > publickey**: Calculates a public key from the corresponding private key and write it to a file, named "publickey"

- In the server (**"router0"**), create a configuration file "/etc/wireguard/wg0.conf", and add the following lines::

```
[Interface]
PrivateKey = <Private key of "router0">
#We will use a VPN subnet of 10.20.30.128/25, use 10.20.30.130 for the sever
#and 10.20.30.131 for the client.
Address = 10.20.30.130/25
#Standard port for WireGuard (optional)
ListenPort = 51820
#Accept every packet on the tunnel interface, and masquerade outgoing
#packets with a public IP
PostUp = iptables -A FORWARD -i wg0 -j ACCEPT
PostUP = iptables -t nat -A POSTROUTING -o ether1 -j MASQUERADE
#Delete the NAT rule when removing the VPN
PostDown = iptables -D FORWARD -i wg0 -j ACCEPT
PostDown = iptables -t nat -D POSTROUTING -o ether1 -j MASQUERADE

[Peer]
PublicKey = <Public key of "pclinet2">
AllowedIPs = 10.20.30.131
#Send a keepalive packet every 25 seconds
PersistentKeepalive = 25
```

> *HINT: you can run* `cat privatekey`*, then copy the key and paste it into the config file. The same goes for the publickey.*

- In the client (**"plient2"**), create a configuration file "/etc/wireguard/wg0.conf", and add the following lines:

```
[Interface]
PrivateKey = <Private key of "pclient2">
Address = 10.20.30.131/25
```

```
[Peer]
PublicKey = <Public key of "router0">
#Public ip address of the server "router0" and ListenPort
Endpoint = 129.168.1.2:51820
# Route all traffic with destination IP address in the subnet
# of (10.20.30.0/24) through this Wireguard tunnel
AllowedIPs = 10.20.30.0/24
```

- Bring up the WireGuard tunnel interface, in **both** the server and the client:

```
sudo wg-quick up /etc/wireguard/wg0.conf
```

- Verify that there is a new interface "wg0" added:

```
ip address
```

- Verify that your VPN is setup correctly by pinging the webserver from **"pclient2"**:

```
ping 10.20.30.3
```

Q15. **REPORT(5%):** Submit a screen of the output of `sudo wg show`, in **both** the server and the client. Briefly explain the output.

2.2 Analyzing VPN

- Start packet capturing on **"router3"** (as the outside party, or "Trudy") on **"ether0"** interface:
- Bring down WireGuard tunnel interface, then bring it up to capture the handshake (in **both** the server and the client):

```
sudo wg-quick down /etc/wireguard/wg0.conf
sudo wg-quick up /etc/wireguard/wg0.conf
```

- In **"pclient2"**, retrieve the content from the "webserver, and then stop packet capturing.

```
wget https://10.20.30.3 --header "Host: www.ttm4200.com" --no-check-certificate
```

- The wireshark version installed in your VM does not support WireGuard dissection, thus you need to update it:

```
sudo add-apt-repository ppa:wireshark-dev/stable
sudo apt-get update
sudo apt install wireshark
```

> *NOTE: Traffic obfuscation is not a goal of WireGuard (i.e. Wireguard is not trying to hide the fact that a user is using a VPN), thus its packets can be sniffed and dissected. Wireshark is using heuristics to detect if the packets are from WireGuard.*

- Open "wireguard_https.pcap" in Wireshark and apply the display filter `wg` to show only WireGuard packets. Then answer the following questions and validate your answer with an **annotated** screenshot from Wireshark:

Q16. **REPORT(4%):** Locate the "Handshake Initiation" packet, which corresponds the the first message from Initiator ("pclient2" in our case) to Responder ("router0" in our case). What is the used transport protocol? What is the destination port? Does it corresponds to the "ListenPort" in the WireGuard server? What fields are included in the WirGuard protocol?

> *HINT: https://www.wireguard.com/protocol/#first-message-initiator-to-responder*

Q17. **REPORT(4%):** Locate the "Handshake Response" packet, which corresponds the second message from Responder to Initiator. What fields are included in the WireGuard protocol? Which fields has changed compared to the "Handshake Initiation"?

Q18. **REPORT(3%):** Locate a "Transport Data" packet, which corresponds to the encapsulated data. What fields are included in the WireGuard protocol?

Q19. **REPORT(3%):** Locate a "Keepalive" message. What fields are included in the WireGuard protocol?

Q20. **REPORT(6%):** What is WireGuard doing for you in this example? Confidentiality? Integrity? Authentication? Forward Secrecy? Explain why?

Q21. **REPORT(5%):** Explain the difference betwee how users can access the webserver **with** and **without** VPN.

> *HINT: Consider the use of private IP address vs Public IP address in your explanation.*

Q22. **REPORT(5%):** What do you think would happen if you use the public IP address while using VPN? What would happen if you also perform port forwarding? Explain why?

Q23. **REPORT(6%):** Submit your capture files "wireguard_https.pcap" along with the report. We will check this file and make sure that it is unique and **consistent with to your answers to the previous questions**.

Q24. **Extra Credit(7%):** WireGuard supports optional Pre-shared Symmetric key Mode, which provides sufficient protection for post-quantum cryptography. According to WireGuard protocol description: "If an additional layer of symmetric-key crypto is required (for, say, post-quantum resistance), WireGuard also supports an optional pre-shared key that is mixed into the public key cryptography. When pre-shared key mode is not in use, the pre-shared key value is assumed to be an all-zero string of 32 bytes."

(ref:https://www.wireguard.com/protocol/). Reconfigure WireGuard with a pre-shared key (use the same setup as before, but include a pre-shared key). Submit a screenshot of your configuration, screenshot of the output of `sudo wg show`, and the capture file. Briefly explain how using a Pre-shared key augments WireGuard security in case of quantum computing advances.

> *HINT: You may find this setup example helpful https://blog.ruanbekker.com/blog/2020/01/11/setup-a-wireguard-vpn-server-on-linux/*

3 Milestone 3 – Digital Signature

In this milestone, we will apply a digital signature to provide a document authentication (verifying that the document was created by a known sender) and document integrity (verifying that the document was unaltered in transit). We will use Pretty Good Privacy (PGP) protocol for creating and verifying signatures.

One common and free implementation of PGP is GNU Privacy Guard (GPG), also known as GnuPG(https://gnupg.org/). We will use the command-line tool, which can be installed in your VM (no need for `docker` in this milestone):

```
sudo apt-get install gnupg
```

3.1 Public/Private Keys

- Create a PGP public/private keypair with the following command:

  ```
  gpg --gen-key
  ```

  Then enter a name (e.g. your team's name), email address (e.g. one of your emails), and a passphrase to protect your private key (remember this passphrase!). By default, it will create an RSA key with 3072 bit. In this lab you will use the private key to sign a document, while others can use your public key to verify your signature.

- Export your public key as an ASCII file so you can share it with others:

  ```
  gpg --export --armor youremail > teamName_pubkey.asc
  ```

- Import and trust others' public keys. There is a public key that belongs to this course at lab6 directory ("lab6/pgp/ttm4200_pubkey.asc"). You should import it, and trust it:

  ```
  gpg --import ttm4200_pubkey.asc
  gpg --edit-key ttm4200
  trust
  5
  y
  ```

```
quit
```

3.2 Verifying a Digital Signature

Q25. **REPORT(3%):** This pdf document was digitally signed with the private key of ttm4200, where the signature file is at "lab6/pgp/lab6.pdf.asc". Verify the signature with the following command and provide a screenshot of the output:

```
gpg --verify lab6.pdf.asc lab6.pdf
```

Q26. **REPORT(5%):** Explain how this verification proves the document's integrity and authenticity (assuming the public-key hasn't been tampered with).

3.3 Creating a Digital Signature

Q27. **REPORT(6%):** When you are finished writing "Lab Delivery 3", digitally sign your document using the following command (adapt the filename as needed):

```
gpg --armor --detach-sig lab_delivery_3.pdf
```

This will create a detached signature file (e.g. "lab_delivery_3.pdf.asc"). **Submit the signature file along with your public key**. We will verify your document as shown before.

Q27. **Extra Credit(5%):** How can you encrypt your report such that only the owner of the private key of ttm4200 can read it?

> **Note:** if you follow this step make sure you submit both an encrypted and unencrypted version of your report, in case something was incorrect in the process.

Contributors

Abdulmajid Murad, David Palma, Stanislav Lange, Mathias Pettersen, Sebastian Fuglesang.