ENCS5121
Information Security and
Computer Networks Laboratory

EXPERIMENT #6
Public-Key Infrastructure (PKI) Lab

Slides by: Mohamad Balawi
Updated By: Tariq Odeh

Overview

- **Problem:**
  - Man-in-the-middle attacks exploit the exchange of public keys due to the lack of verification.
- **Solution:**
  - Public Key Infrastructure (PKI), establishes a trusted system to verify public key ownership.
- **Key Components:**
  - Certificate Authority (CA): Issues and verifies digital certificates.
  - X.509 Certificate: Binds public keys with identity information.
  - Root CA: The ultimate trust anchor in the PKI hierarchy.
- **Applications:**
  - Secures web communications (e.g., HTTPS).
  - Prevents MITM attacks by ensuring the authenticity of public keys.

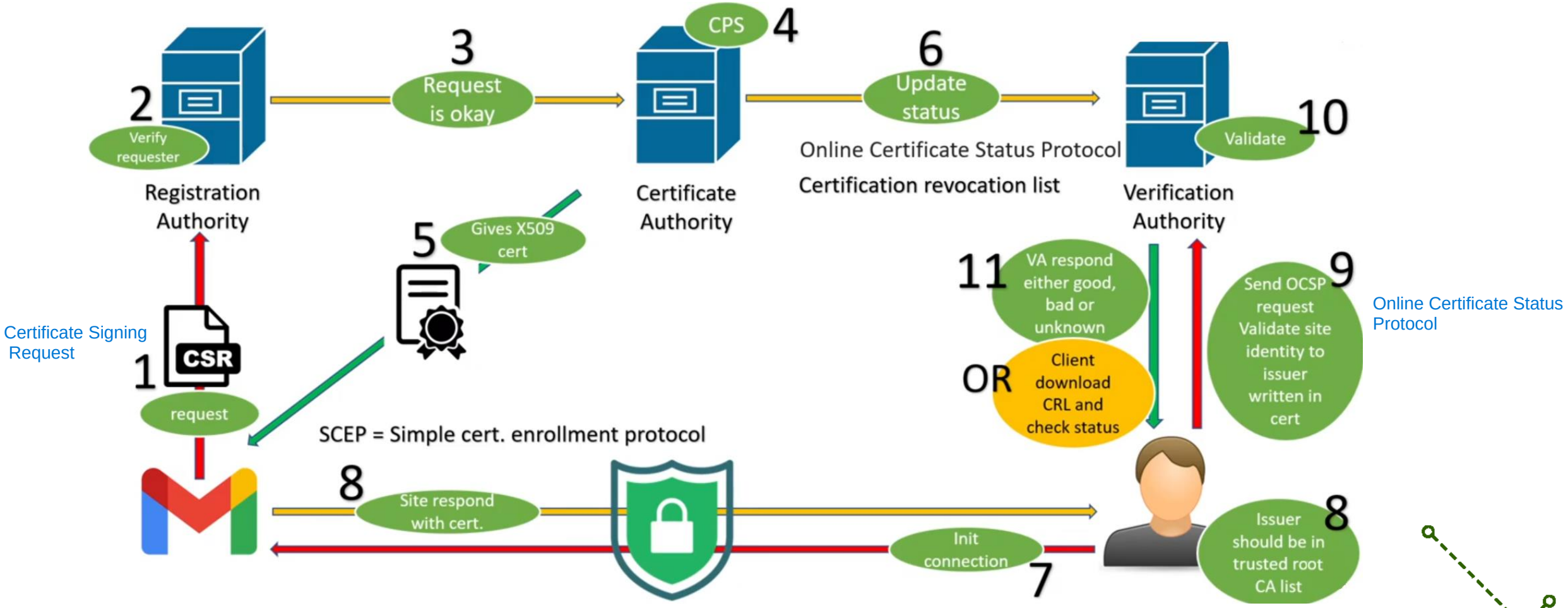
Overview (Cont.)

- **Learning Outcomes:**
  - Understand PKI, CAs, and the role of certificates.
  - Learn how HTTPS uses PKI to secure data.
  - Explore potential issues if the root trust is compromised.

Outline

- **Introduction**
- **Task 1:** Becoming a Certificate Authority (CA).
- **Task 2:** Generate a Certificate Request for a Server.
- **Task 3:** Generating a Certificate for your server.
- **Task 4:** Deploying Certificate in an Apache-Based Website.
- **Task 5:** Launching a Man-In-The-Middle Attack.
- **Task 6:** Man-In-The-Middle Attack with a Compromised CA.

Public Key Infrastructure (PKI) Overview



Apache HTTP Server Project

- The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards.

- The Apache HTTP Server ("httpd") was launched in 1995, and it has been the most popular web server on the Internet since April 1996.

SSL & TLS

- SSL (Secure Sockets Layer) and TLS (Transport Layer Security) are cryptographic protocols for securing network communication. TLS, the successor to SSL, offers improved security and is widely used for **HTTPS** and other secure connections.
- Even though TLS is the successor to SSL, but the terms are often used interchangeably. TLS 1.0 is essentially SSL 3.0, and subsequent versions of TLS have evolved independently of SSL.
- SSL versions prior to SSL 3.0 are considered insecure and deprecated due to various vulnerabilities discovered over time.

HTTP port 80
HTTPS port 443

- SSL 2.0 was first released in 1995.
- TLS 1.0 was first released in 1999.

TASK1

Becoming a Certificate Authority (CA)

Expected output of this task:

- CA Private Key (ca.key)
- CA Certificate (ca.crt)

جامعة بيرزيت BIRZEIT UNIVERSITY

Task 1: Becoming a Certificate Authority (CA)

- A Certificate Authority (CA) is a trusted entity that issues digital certificates.
- Users who want to get digital certificates issued by the commercial CAs need to pay those CAs.
- In this lab, we need to create digital certificates, but we are not going to pay any commercial CA.
- We will become a root CA ourselves and then use this CA to issue certificate for others.

Step1: Default settings to prepare

The [CA default] section of the configuration file shows the default setting that we need to prepare.

```
[ CA_default ]
Dir               = ./demoCA            # Where everything is kept.
Certs             = $dir/certs          # Where the issued certs are kept.
crl_dir           = $dir/crl            # Where the issued crl are kept.
Database          = $dir/index.txt      # database index file.
#unique_subject   = no                  # allows multiple certs with the same subject.
new_certs_dir     = $dir/newcerts       # default place for new certs.
serial            = $dir/serial         # The current serial number.
```

Step1: Default settings to prepare (Cont.)

```
seed@VM:~/.../EXP6$ mkdir ./demoCA
seed@VM:~/.../EXP6$ cd demoCA/
seed@VM:~/.../demoCA$ mkdir cert
seed@VM:~/.../demoCA$ mkdir crl
seed@VM:~/.../demoCA$ mkdir newcerts
seed@VM:~/.../demoCA$ touch index.txt
seed@VM:~/.../demoCA$ echo "1000" > serial
seed@VM:~/.../demoCA$ cd ..
```

Step2: Copy openssl.conf

- To use OpenSSL to create certificates, you must have a configuration file.
- The configuration file usually has an extension `.cnf`
- The configuration file is used by three OpenSSL commands: ca, req and x509.
- By default, OpenSSL uses the configuration file from `/usr/lib/ssl/openssl.cnf`.
- Since we need to make changes to this file, we will copy it into our current directory and instruct OpenSSL to use this copy instead.

```
seed@VM:~/.../EXP6$ cp /usr/lib/ssl/openssl.cnf myCA_openssl.cnf
```

- Later, we will use the “-config” option to use our version instead of the default one.

Step3: Certificate Authority (CA)

- As we described before, we need to generate a self-signed certificate for our CA. You can run the following command to generate the self-signed certificate for the CA:

```
seed@VM:~/.../EXP6$ openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -keyout ca.key -out ca.crt -config myCA_openssl.cnf
```

Argument	Description
req	Certificate request.
-x509	Outputs a certificate instead of a certificate request.
-newkey rsa:4096	Generate a new RSA (4096-bit) private key.
-sha256	The hashing algorithm to use for generating the certificate's fingerprint.
-days 3650	Sets the validity period of the certificate to 3650 days (approximately 10 years).
-keyout ca.key	The file where the generated private key should be saved.
-out ca.crt	The file where the generated X.509 certificate should be saved.
-config openssl.cnf	This specifies the configuration file to use for generating the certificate.

Step4: Decode X.509 Certificate and RSA Key

- To look at the decoded content of the X509 certificate and the RSA key
  - **-text** means decoding the content into plain text;
  - **-noout** means not printing out the encoded version)

```
seed@VM:~/.../EXP6$ openssl x509 -in ca.crt -text -noout
seed@VM:~/.../EXP6$ openssl rsa -in ca.key -text -noout
```

openssl rsa -in ca.key -text -noout

- The output of the "`openssl rsa -in ca.key -text -noout`" command provides comprehensive details regarding both the private and public keys.
- The table on the right correlates the output of the command with the mathematical components of the keys.
- The last three rows, (highlighted in red) are used in the Chinese Remainder Theorem (CRT) optimization, which enables efficient decryption and signing operations, particularly in RSA private key operations.

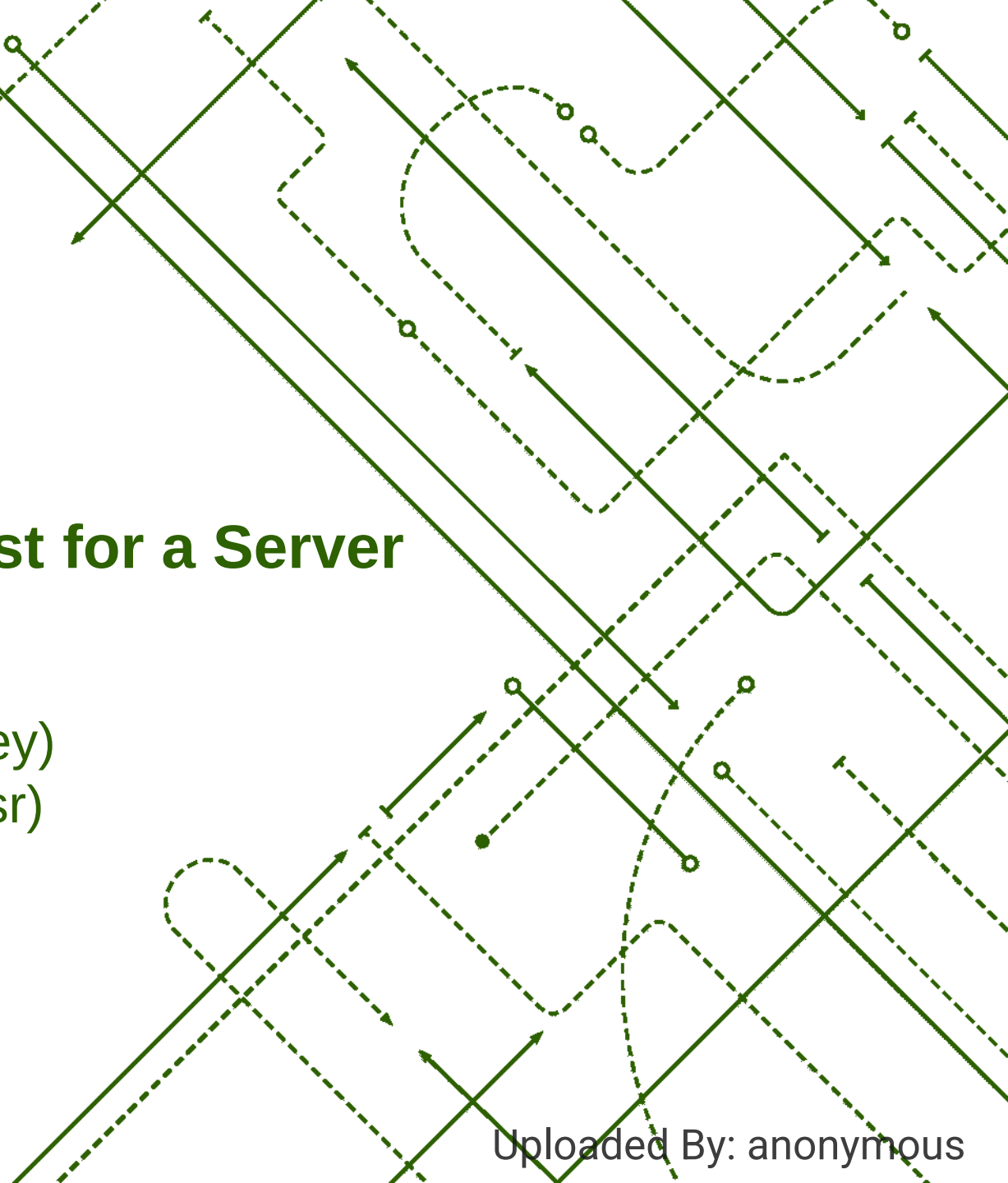
Title in the CMD output	Mathematical symbol
modulus	$n\ (pxq)$
publicExponent	e
privateExponent	d
prime1	p
prime2	q
exponent1	$d\ mod\ (p-1)$
exponent2	$d\ mod\ (q-1)$
coefficient	$q^{-1} \bmod p$

TASK2

Generate a Certificate Request for a Server

Expected output of this task:

- Server Private Key (server.key)
- -Certificate Request (server.csr)



Step 1: Edit openssl.cnf file

- Uncommenting the following from openssl.cnf file:

```
#unique_subject= no → unique_subject = no
#copy_extensions = copy → copy_extensions = copy
#Policy = policy_match → policy = policy_match
```

Step 2: Generate a Certificate Request for a Server

- A company called **www.mbalawi.com** wants to get a public-key certificate from our CA.
- First it needs to generate a Certificate Signing Request **(CSR)**, which basically includes the company's public key and identity information.
- The CSR will be sent to the CA, who will verify the identity information in the request, and then generate a certificate.
- The command to generate a CSR is similar to the one we used in creating the self-signed certificate, the only difference is the absence of the **–x509** argument.

```
seed@VM:~/.../EXP6$ openssl req -newkey rsa:2048 -sha256 -keyout server.key -out
server.csr -subj "/CN=www.mbalawi.com/O=Birzeit University/C=PS" -passout pass:dees
```

Adding Alternative names

- Many websites have different URLs. For example, www.example.com, example.com, example.net, and example.org are all pointing to the same web server.
- Due to the hostname matching policy enforced by browsers, the common name in a certificate must match with the server's hostname, or browsers will refuse to communicate with the server.
- To allow a certificate to have multiple names, the X.509 specification defines extensions to be attached to a certificate. This extension is called **Subject Alternative Name (SAN)**.
- Using the SAN extension, it's possible to specify several hostnames in the `subjectAltName` field of a certificate.

Step 3: Add SAN extension using command-line

- To use the SAN extension can add the "-addext" option to the "openssl req" command.
- It should be noted that the `subjectAltName` extension field must also include the **Common Name (CN)** field; otherwise, the common name will not be accepted as a valid name.

```
seed@VM:~/.../EXP6$ openssl req -newkey rsa:2048 -sha256 -keyout server.key -out
server.csr -subj "/CN=www.mbalawi.com/O=Birzeit University/C=PS" -passout pass:dees
-addext "subjectAltName = DNS:mbalawi.com, DNS:www.mbalawi.com, DNS:www.mb.com"
```

Additional info

- The command will generate a pair of public/private key and then create a certificate signing request from the public key.
- We can use the following command to look at the decoded content of the CSR and private key files:

```
seed@VM:~/.../EXP6$ openssl req -in server.csr -text –noout
seed@VM:~/.../EXP6$ openssl rsa -in server.key -text -noout
```

TASK3

Generating a Certificate for your server

Expected output of this task:

- Server Certificate (server.crt)

Step 1: Becoming a Certificate Authority (CA)

- The following command turns the certificate signing request (**server.csr**) into an X509 certificate (**server.crt**), using the CA's **ca.crt** and **ca.key**:

```
seed@VM:~/.../EXP6$ openssl ca -config myCA_openssl.cnf -policy policy_anything -md
sha256 -days 3650 -in server.csr -out server.crt -batch -cert ca.crt -keyfile ca.key -
passin pass:dees
```

- where **myCA_openssl.cnf** is the configuration file we copied from **/usr/lib/ ssl/openssl.cnf** (we also made changes to this file in Task 1).

Options Description

```
seed@VM:~/.../EXP6$ openssl ca -config myCA_openssl.cnf -policy policy_anything -md
sha256 -days 3650 -in server.csr -out server.crt -batch -cert ca.crt -keyfile ca.key -
passin pass:dees
```

Argument	Description
-policy policy_anything	Specifies the policy to use when signing the certificate. The "policy_anything“ allows for flexibility, the default policy has more restriction, requiring some of the subject information in the request to match those in the CA’s certificate.
-md sha256	(message digest) specifies the SHA-256 signature algorithm for OpenSSL.
-batch	batch mode, operates without user prompts, ideal for automated processes.
-passin pass:dees	Specifies the password the private key file. Here it is “dees”.

Step 2: Read the certificate

- After executing the previous certificate signing command command, execute the following to read the certificate:

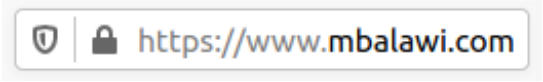
```
seed@VM:~/.../EXP6$ openssl x509 -in server.crt -text -noout
```

TASK4

Deploying Certificate in an Apache-Based HTTPS Website & add our CA to the browser's list of trusted CAs

Expected output of this task:

- Secure connection to our server



BIRZEIT UNIVERSITY

Task 4: Deploying Certificate in an HTTPS Website

- In this task, we will see how public-key certificates are used by websites to secure web browsing.
- We will set up an HTTPS website-based Apache.
- The Apache server, which is already installed in the docker container, supports the HTTPS protocol.
- You can deploy it after extracting `labsetup.zip` and executing `dcbuild && dcup` inside `labsetup` directory.
- To create an HTTPS website, we just need to configure the Apache server, so it knows where to get the private key and certificates.
- Inside our container, we have already set up an HTTPS site for bank32.com. Students can follow this example to set up their own HTTPS site.

Hosting Multiple Websites

- An Apache server can simultaneously host multiple websites.
- It needs to know the directory where a website's files are stored.
- This is done via its VirtualHost file, located in the `/etc/apache2/sites-available` directory.
- In our container, we have a file called `bank32_apache_ssl.conf`, which contains two VirtualHosts.
- Each virtual host has its own configuration settings, enabling the server to serve different content based on factors like domain name or IP address.

VirtualHost File

- In our container, we have a file called `bank32_apache_ssl.conf`, which contains the following:

```
<VirtualHost *:443>
    DocumentRoot /var/www/bank32
    ServerName www.bank32.com
    ServerAlias www.bank32A.com
    ServerAlias www.bank32B.com
    ServerAlias www.bank32W.com
    DirectoryIndex index.html
    SSLEngine On
    SSLCertificateFile /certs/bank32.crt
    SSLCertificateKeyFile /certs/bank32.key
</VirtualHost>
```

```
<VirtualHost *:80>
    DocumentRoot /var/www/bank32
    ServerName www.bank32.com
    DirectoryIndex index_red.html
</VirtualHost>
```

VirtualHost File Entries

- The following table contains the meaning of different entries in the VirtualHost file

entry	Description
<VirtualHost *:443>	Defines the port (443 is the default port for HTTPS) (80 is the default for HTTP)
DocumentRoot	Specifies where the files for the website are stored.
ServerName	Specifies the primary domain name for the website.
ServerAlias	Specifies additional domain names (aliases) for the virtual host.
DirectoryIndex index.html	Defines the default filename to be served when a directory is requested. If a directory is accessed without specifying a filename, Apache will look for index.html in that directory and serve it if found.
SSLEngine On	Enables SSL/TLS encryption for this virtual host, allowing HTTPS connections.
SSLCertificateFile	Specifies the path to the SSL certificate file. The SSL certificate file contains the public key and other details necessary for SSL/TLS encryption.
SSLCertificateKeyFile	Specifies the path to the private key file associated with the SSL certificate.

Shared Folder Between the VM and Container

- In this task, we need to copy files from the VM to the container.
- To avoid repeatedly recreating containers, we have created a shared folder between the VM and container.
- When you use the Compose file inside the `Labsetup` folder to create containers, the `volumes` sub-folder will be mounted to the container. Anything you put inside this folder will be accessible from inside of the running container.

What do we Need to Do?

1. **Open the /etc/hosts file:**

```
seed@VM:~/.../Labsetup$ sudo nano /etc/hosts
```

2. **Add the following IP-to-Hostname mapping to the end of /etc/hosts file:**

```
seed@VM:~/.../Labsetup$ 10.9.0.80      www.mbalawi.com
```

3. **Copy our server's certificate (`server.crt`) and key (`server.key`) to `Labsetup/volumes`.**

4. **Navigate to Labsetup directory and execute the following command:**

```
seed@VM:~/.../Labsetup$ dcbuild
seed@VM:~/.../Labsetup$ dcup
```

What do we Need to Do?

5. **Get the container ID from executing the following command:**

```
seed@VM:~/.../Labsetup$ dockps
```

6. **Enter the container by executing the following command after replacing <container_id>:**

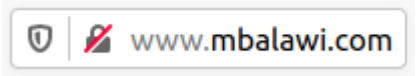
```
seed@VM:~/.../Labsetup$ docksh <container_id>
```

7. **Start the Apache server using the following command:**

```
root@<container_id>:/# service apache2 start
```

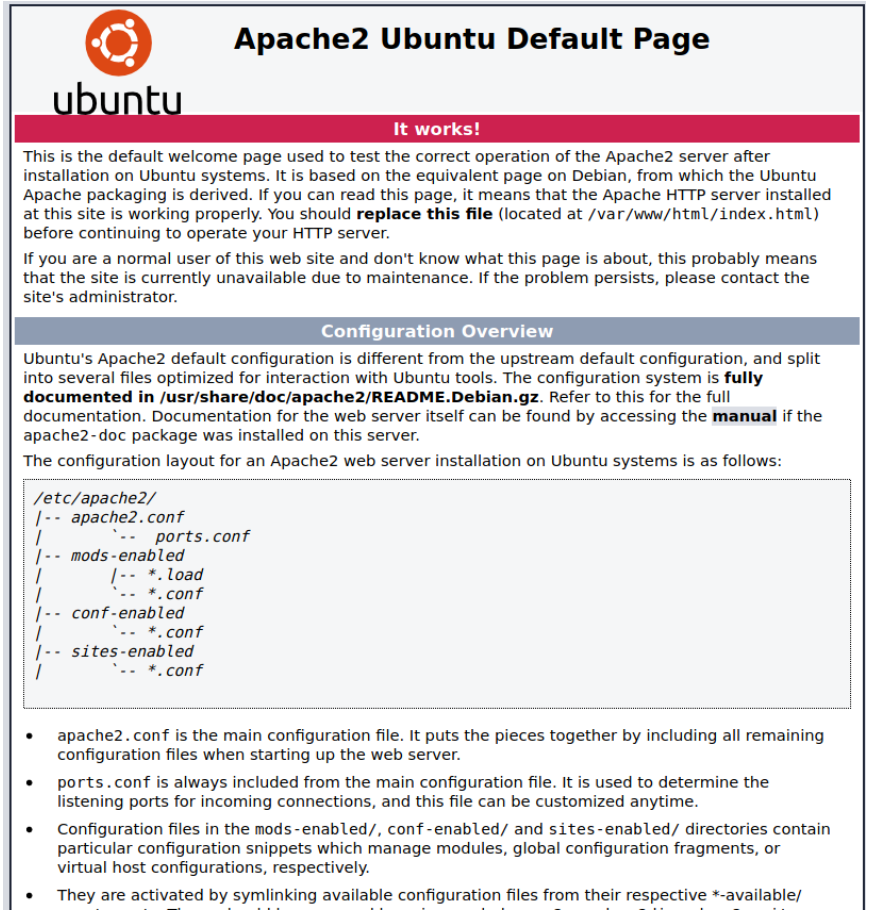
- if it asks for a password, use the one we set for our server in **Task 2**. it asks for a password because when Apache starts, it needs to load the private key for each HTTPS site, and our private key is encrypted, so Apache will ask us to type the password for decryption.

What to expect?

- If you get the Apache2 Ubuntu Default Page after accessing the HTTP port 

or the HTTPS port 

of your website, then there is something wrong with your configuration.



Apache2 Ubuntu Default Page

ubuntu

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

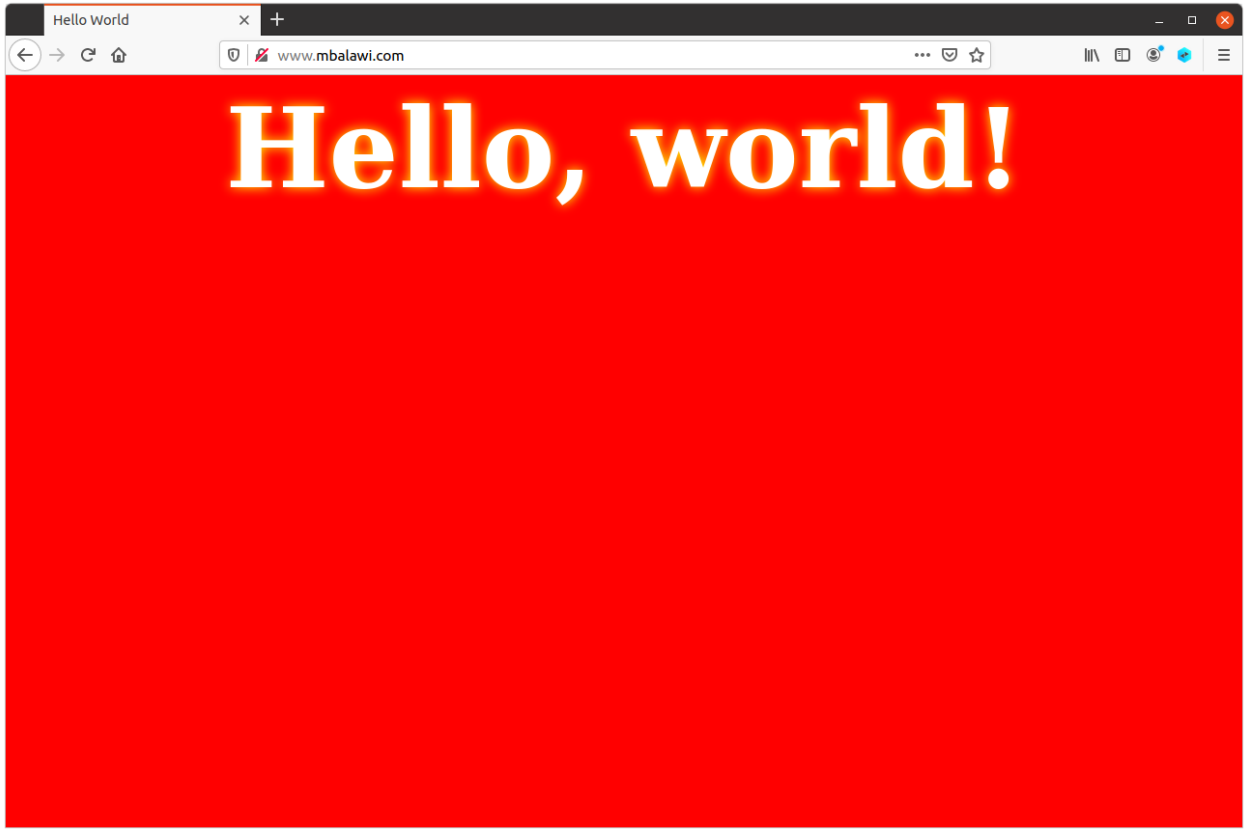
```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective *-available/

What to expect?

- When you try to access the HTTP port of your website 

you will get `index_red.html` page.
- The icon means that your connection is insecure and that is because HTTP is insecure by nature.



What to expect?

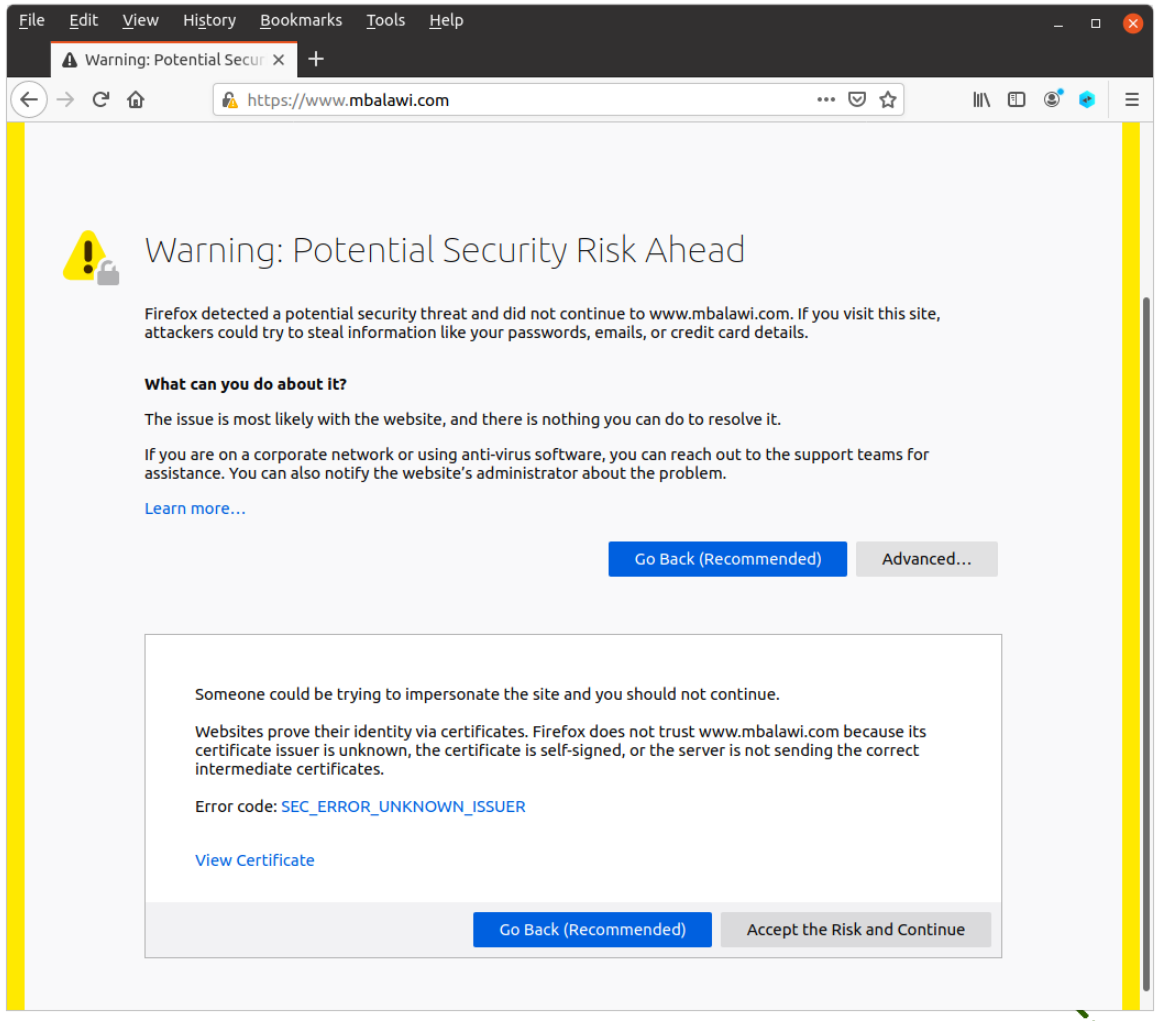
- When you try to access the HTTPS port of your website 

you will get a warning page from Firefox.
- The icon means that your connection is insecure and that is because the certificate is invalid, there are many reasons for that, in our case it is caused by unknown CA.
- You can bypass this warning by clicking 

button then 

- Read the dialoge that appears after clicking 

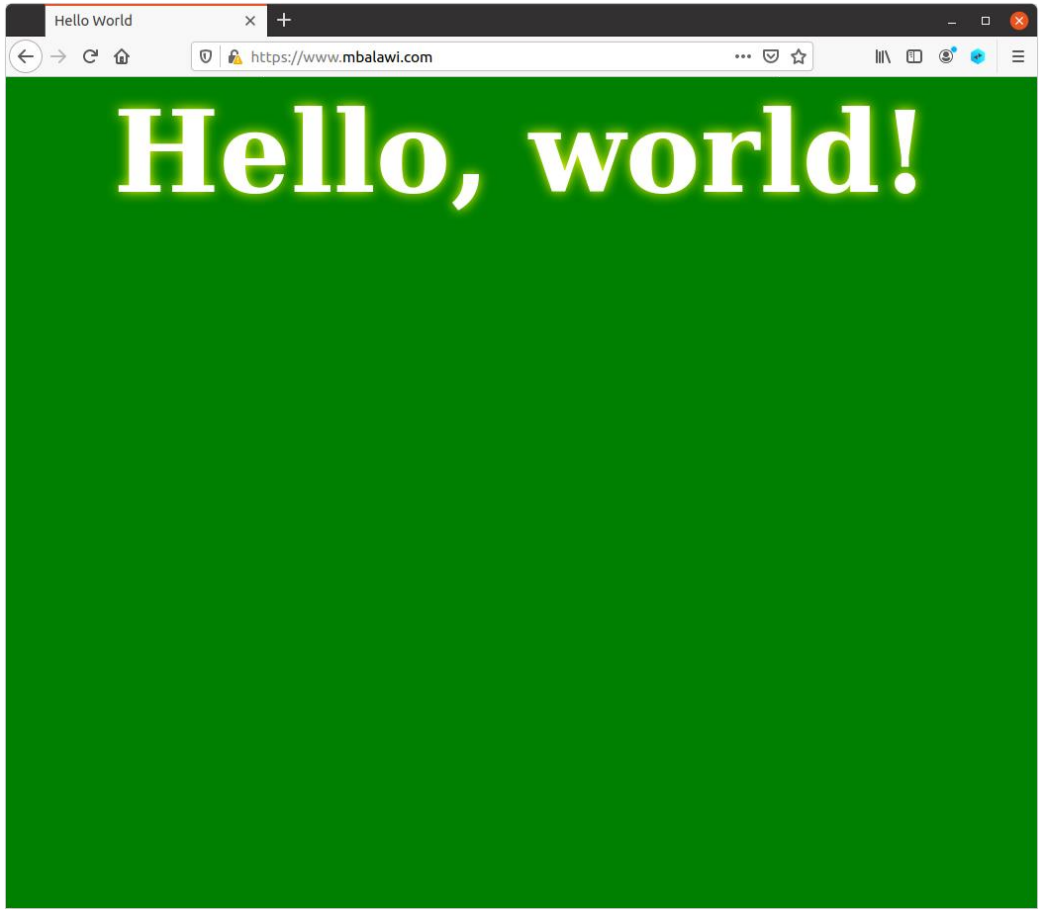
button.



What to expect?

- After bypassing the warning for the HTTPS port of your website 

you will get a `index.html` page which is a “Hello, world!” title on green background.

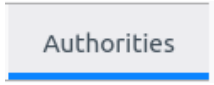


Add our CA to Firefox trusted CAs

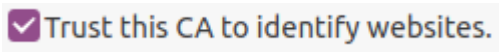
- To avoid the browser's warning, we need to add our CA to the browser's list of trusted CA.
- To do that, we need to navigate to the following URL:

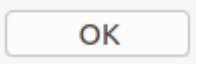
`about:preferences#privacy` 

- Then go to Certificates section and click the 

button.
- Switch to 

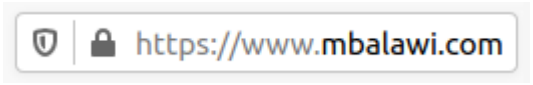
tab.
- Then click 

button.
- Choose our CA's self signed certificate (`ca.crt`).
- Tick 

checkbox.
- Click 

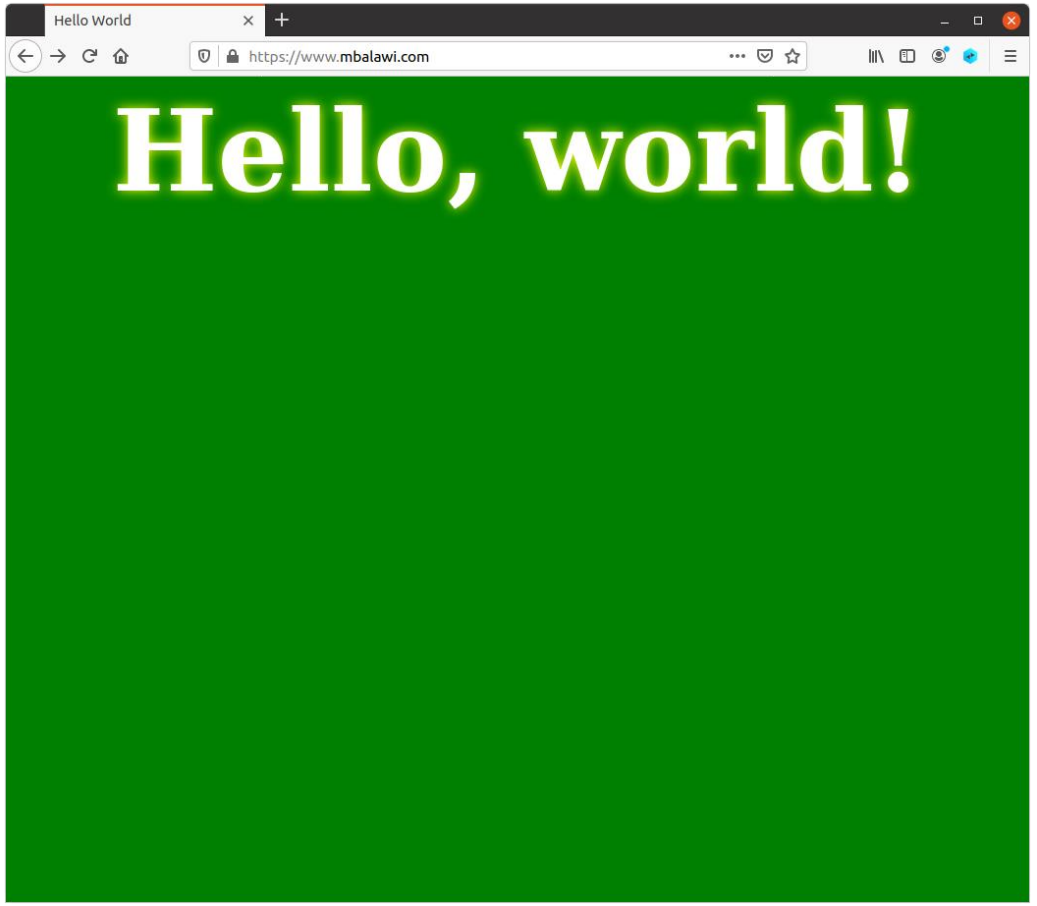
button.
- Refresh `https://www.mbalawi.com`

What to expect after adding our CA to Firefox?

- When you try to access the HTTPS port of your website 

you will get `index.html` page, but this time the padlock icon is different.

- The 🔒 icon means that your connection is secure because it provided a certificate that is signed by a valid CA (our CA).

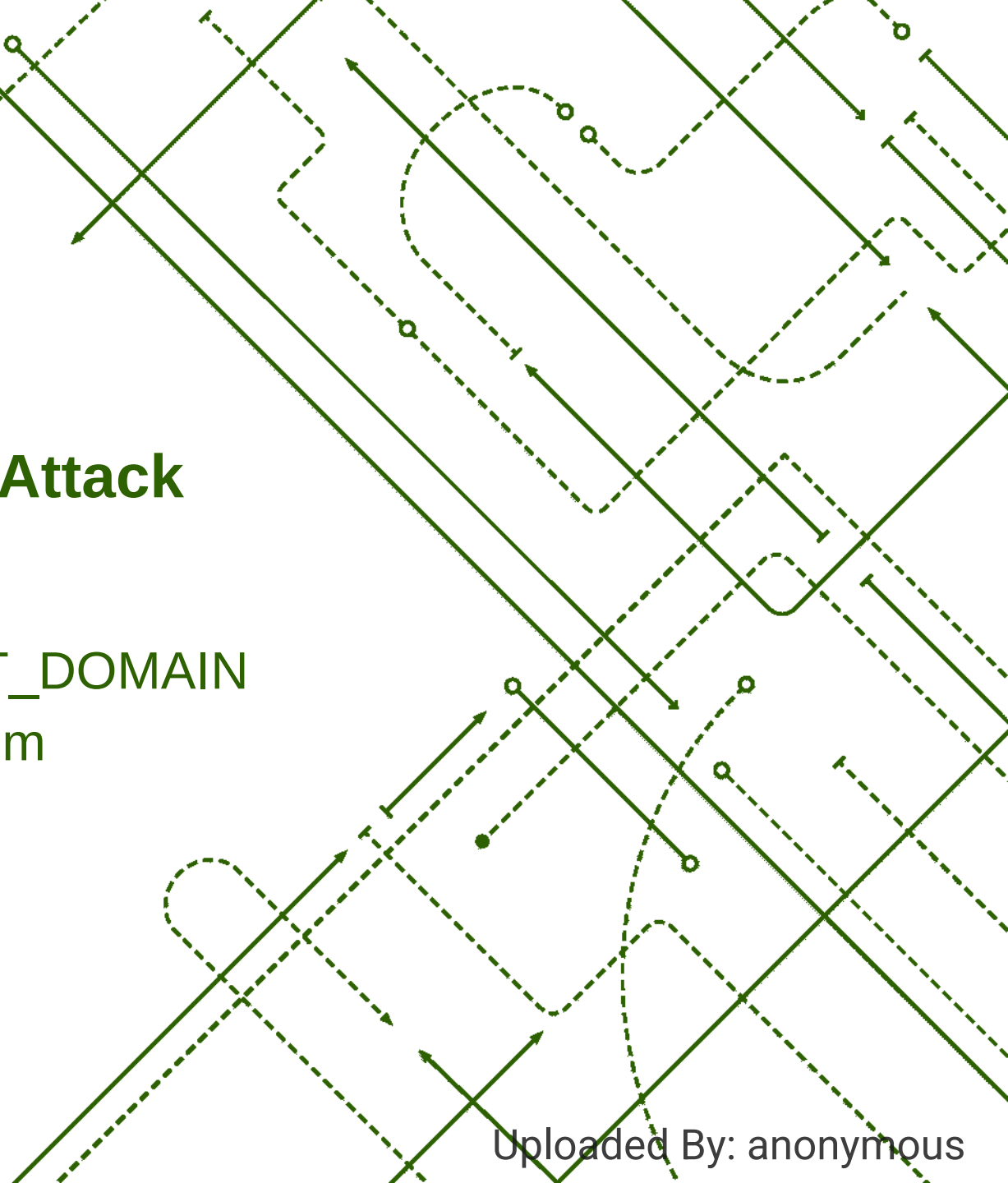


TASK5

Launching a Man-In-The-Middle Attack

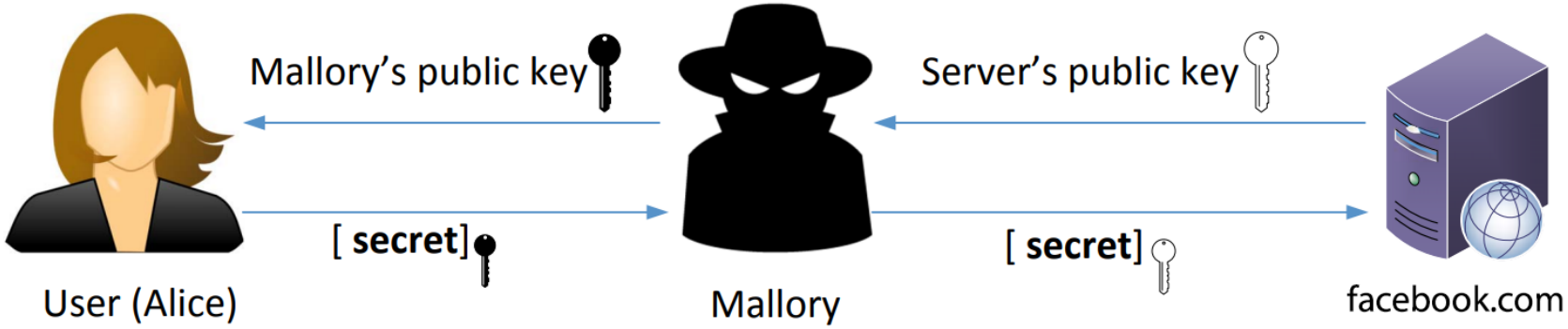
Expected output of this task:

- Getting SSL_ERROR_BAD_CERT_DOMAIN when trying to access facebook.com



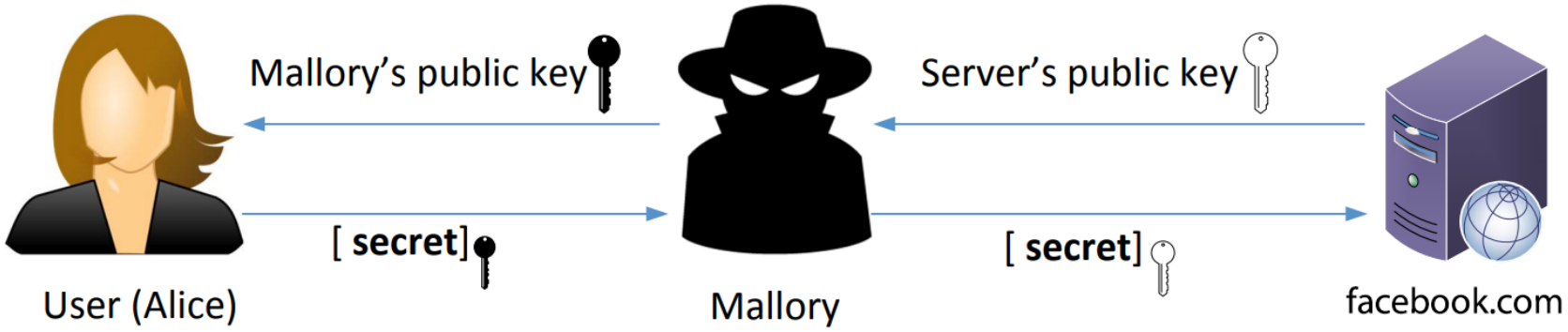
Launching a Man-In-The-Middle Attack

- In this task, we will show how PKI can defeat Man-In-The-Middle (MITM) attacks.
- Assume Alice wants to visit `facebook.com` via the HTTPS protocol. She needs to get the public key from the `facebook.com` server; Alice will generate a secret, and encrypt the secret using the server's public key, and send it to the server.
- If an attacker can intercept the communication between Alice and the server, the attacker can replace the server's public key with its own public key.



Launching a Man-In-The-Middle Attack

- Therefore, Alice's secret is encrypted with the attacker's public key, so the attacker will be able to read the secret. The attacker can forward the secret to the server using the server's public key. The secret is used to encrypt the communication between Alice and server, so the attacker can decrypt the encrypted communication.
- In the task, we will emulate an MITM attack and see how exactly PKI can defeat it.



Planning Our Attack Strategy (step 1)

- In Task 4, we have already set up an HTTPS website.
- We will use the same Apache server to impersonate `www.facebook.com`. To achieve that, we will follow the instruction in Task 4 to add a VirtualHost entry to Apache's SSL configuration file: the ServerName should be `www.facebook.com`, but the rest of the configuration can be the same as that used in Task 4. Obviously, in the real world, you won't be able to get a valid certificate for `www.facebook.com`, so we will use the same certificate that we used for our own server.
- Our goal is the following: when a user tries to visit `www.facebook.com`, we are going to get the user to land in our server, which hosts a fake website for `www.facebook.com`. The fake site can display a login page like the one in the target website. If users cannot tell the difference, they may type their account credentials in the fake webpage, essentially disclosing the credentials.

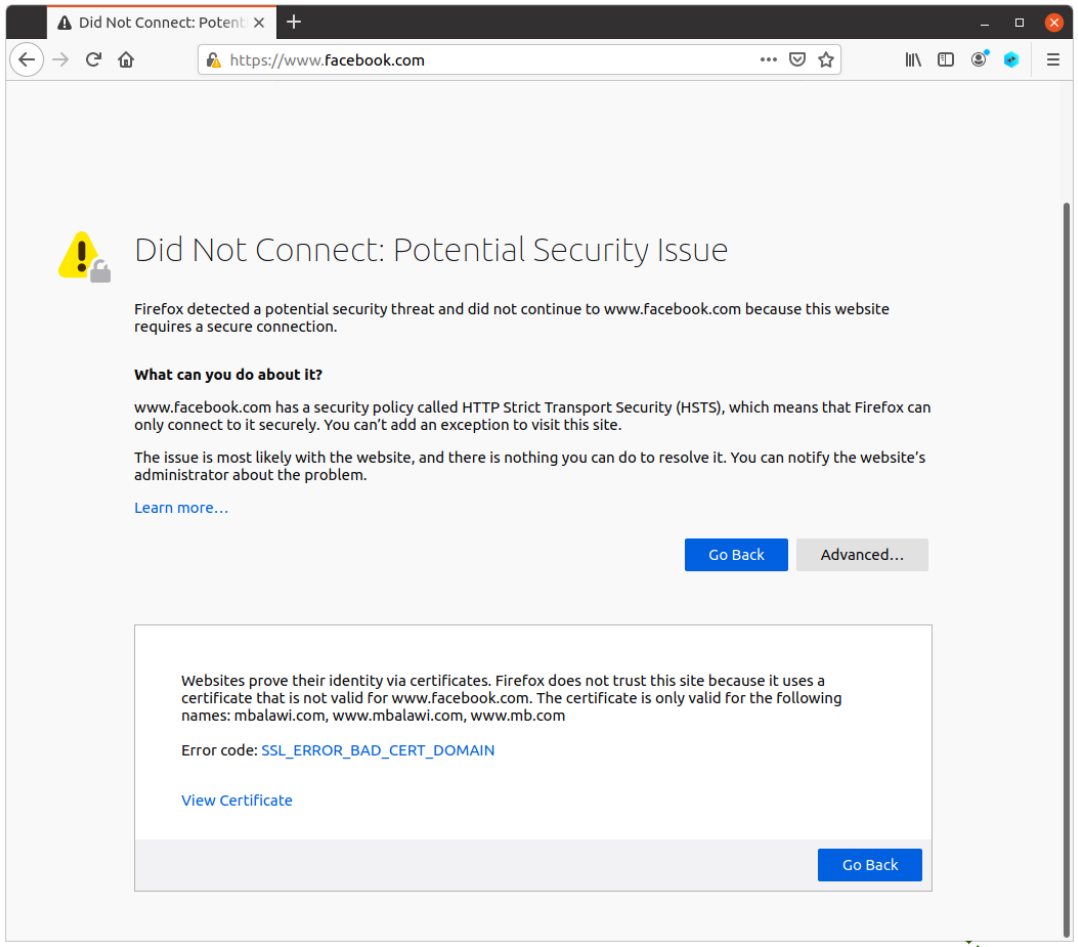
Planning Our Attack Strategy (step 2)

- There are several ways to get the user's HTTPS request to land in our web server.
- One way is to attack the routing, so the user's HTTPS request is routed to our web server.
- Another way is to attack DNS, so when the victim's machine tries to find out the IP address of the target web server, it gets the IP address of our web server.
- In this task, we simulate the attack-DNS approach. Instead of launching an actual DNS cache poisoning attack, we simply modify the victim's machine `/etc/hosts` file to emulate the result of a DNS cache poisoning attack by mapping the hostname `www.facebook.com` to our malicious web server.

```
seed@VM:~/.../Labsetup$ sudo nano /etc/hosts
seed@VM:~/.../Labsetup$ 10.9.0.80      www.facebook.com
```

What to expect?

- When attempting to visit `www.facebook.com`, we encounter a warning page denying access to the website.
- `SSL_ERROR_BAD_CERT_DOMAIN` is the displayed error message.
- This outcome is anticipated because we utilized a certificate from `www.mbalawi.com`. The browser compares the domain name on the certificate with the domain name we are attempting to access and detects a discrepancy, resulting in the display of the error page.

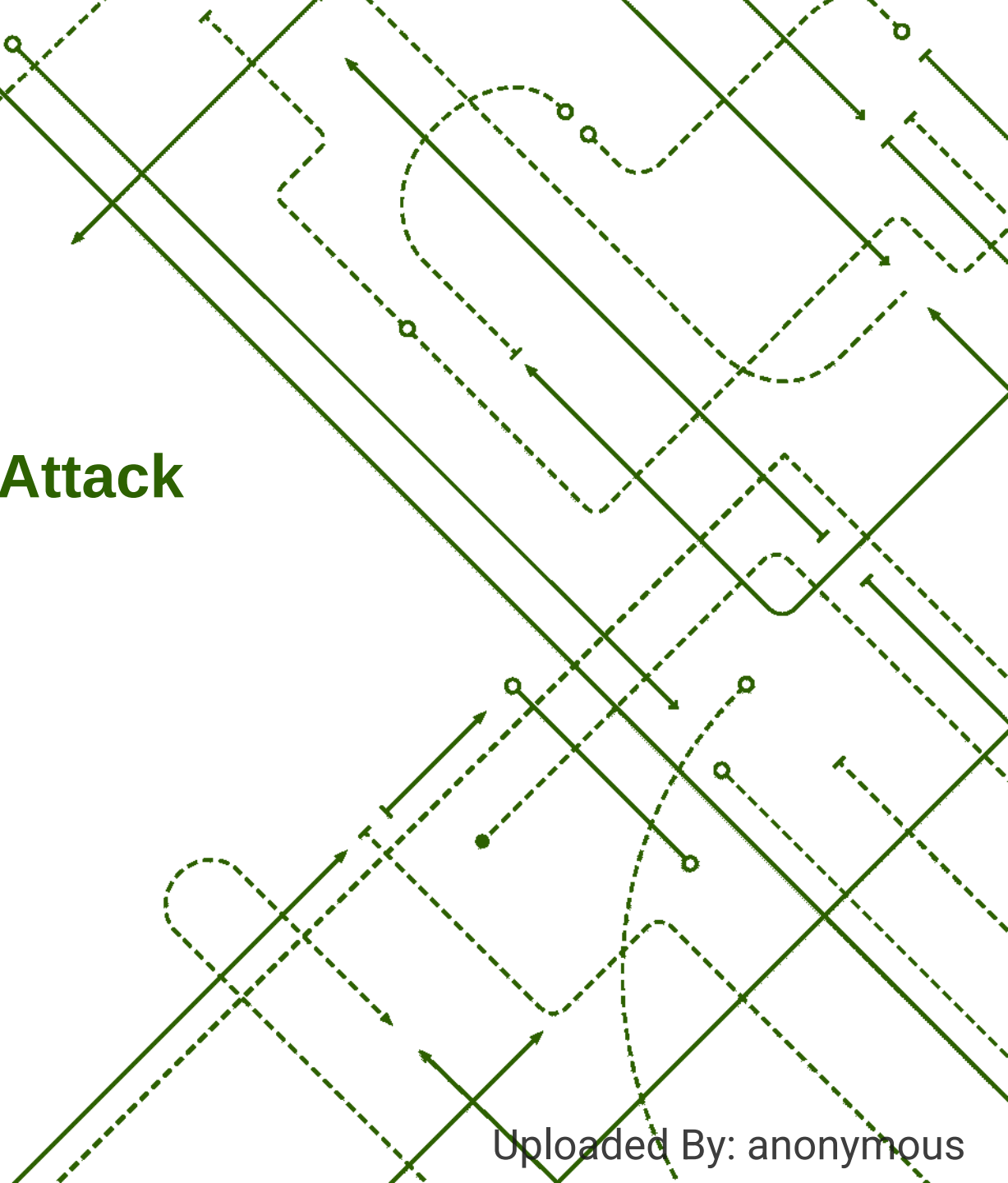


TASK6

Launching a Man-In-The-Middle Attack with a Compromised CA

Expected output of this task:

- MITM attack is successful



جامعة بيرزيت BIRZEIT UNIVERSITY

Launching a Man-In-The-Middle Attack

- In this task, we assume that the root CA created in Task 1 is compromised by an attacker, and its private key is stolen.
- Therefore, the attacker can generate any arbitrary certificate using this CA's private key.
- In this task, we will see the consequence of such a compromise.
- Please design an experiment to show that the attacker can successfully launch MITM attacks on any HTTPS website. You can use the same setting created in Task 5, but this time, you need to demonstrate that the MITM attack is successful, i.e., the browser will not raise any suspicion when the victim tries to visit a website but land in the MITM attacker's fake website.