Web Security Lab

Contents

General Information 1

1 Part I: Encrypted Email 4

2 Part II: Installing a web server and obtaining a certificate 10

3 Part III: Examining TLS Traffic 13

A Collaboration Tools 15

B Apache Configuration Structure 15

General Information

Objectives

The purpose of this lab is to provide hands-on experience with modern web security mechanisms, with a particular focus on cryptographic technologies used in practice. The lab is an assessed component of the course and contributes **20%** of the final grade.

Through the assignment, you will work with both the theoretical foundations and the practical implementation aspects of web security. In particular, the lab aims to:

- introduce secure email communication using **PGP** and its open-source implementation **GPG**,
- give you practical experience with configuring **TLS** on a standard web server,
- strengthen your understanding of how cryptographic mechanisms apply in real systems, and
- expose the practical challenges that arise when deploying and integrating security technologies.

The project involves investigation, configuration, and programming tasks related to relevant security tools and services. As in real-world engineering work, you will also need to consider practical constraints, such as interoperability issues, configuration pitfalls, and system-level behavior.

Overall, the objective is to help you develop both conceptual insight and practical proficiency in deploying and evaluating modern web security mechanisms.

Logistics

This project is carried out in groups of three or four students, with group registration handled through self-enrollment on Blackboard. The members of your group will constitute your lab team throughout the assignment.

Teaching assistants (TAs) will be available both in person during scheduled hours and online through the Ed Forum. All questions about the lab should be asked here; not through email to the course responsible. If you do not have the chance to show up physically for presenting the milestones, you can reach out to the TAs on the forum to schedule a video call. A detailed timetable for TA availability will be published on the course website.

For Parts II and III of the lab, a dedicated server is provided in the form of a virtual machine (VM) running Ubuntu. Part II explains how to activate and authenticate to this VM. Once configured, you may connect from your own computer: macOS and Linux users can use the built-in terminal, while Windows users may either use the Windows Subsystem for Linux (WSL) or install an SSH client such as PuTTY [TDHN].

If You Are Stuck

If you encounter a problem that you cannot resolve through your own investigation, consider discussing it with other groups. You may contact them directly if you know them, but we strongly encourage you to use the Ed

discussion forum. It is likely that other students have faced—or will face—the same issue, making the forum a useful place to search for existing answers or to post new questions.

The teaching assistants (TAs) also monitor the forum and will provide guidance when needed. If your issue remains unresolved after checking available resources, you are welcome to speak with a TA during the scheduled assistance hours.

Remember that while collaboration and discussion are encouraged, all solutions submitted in your lab report must be written independently and reflect your own understanding.

Evaluation

The assignment consists of two deliverables:

1. a demonstration of the completed milestones, and
2. a written lab report submitted on Blackboard.

The assignment is worth a total of 20 points. Ten points are awarded for completing the required milestones (a third of this for each), and ten points are awarded for the quality of the lab report (based on various requirements described below).

Group work is expected to be carried out collaboratively, with all members contributing meaningfully. If a group member becomes unavailable, stops participating, or consistently contributes significantly less than others, you should inform the teaching assistants or the course responsible as early as possible.

To ensure effective collaboration, we recommend establishing a clear working plan: schedule regular meetings, distribute responsibilities, and define intermediate deadlines. The final report must represent a joint effort and accurately reflect the contributions of *all* group members.

Milestones

Throughout this document, you will encounter several activities designed to guide your progress. These activities fall into three categories:

Tasks. Steps you are expected to complete while working through the lab. They are required but will not be checked individually, and are essential to be able to later respond to the milestones.

Milestones. Checkpoints that *must* be completed before the end of the lab period. A teaching assistant (TA) will verify each milestone during assistance hours, either in person or online when necessary.

Questions. Items you will later address in your lab report. You do not need to write full answers during the lab weeks, but you should collect the information needed to complete the report afterwards.

Although there are no intermediate submission deadlines, we recommend following the timeline below:

Part I (PGP). Completed during the first lab week.
Part II (Apache and Certificates). Completed during the second lab week.
Part III (Eavesdropping). Completed during the final lab week.

All **milestones** must be completed by the lab deadline, which is **Friday, 27th March, 2026 (at 17:00)**.

Note that TA availability ends at the scheduled assistance time each day, so milestones must be demonstrated before the lab closes in order to be approved.

After the milestone deadline, you will have four additional weeks to complete your lab report. Server access remains available during this period but will be removed once the report deadline has passed, so ensure that all required data is stored safely beforehand.

The Report

Your lab report will be formally assessed. While completing the tasks and milestones, you should gather sufficient documentation to support your explanations and results in the final report. All claims and statements must be supported with appropriate references. Use peer-reviewed literature or other reputable sources such as

academic publications and technical reports, and avoid relying on blogs, forum posts, or news articles whenever possible.

The report submission deadline on Blackboard is **Friday, 24th April, 2026 (end of day)**.

A LaTeX template for writing the report is available on the course website. Your report must follow the structure and formatting requirements specified in the template, including margins and font size. The template also contains comments suggesting points you may wish to address in your answers.

One point is awarded for format and style, including correct document structure and a properly formatted reference list. Your report must follow this structure:

Title page. Lists all group members, the group number, the report title, and the date.
Introduction. Summarizes the lab purpose, what you aimed to achieve, and what you accomplished.
Answers to the questions. Include each question in full, followed by your corresponding answer.
Discussion. Describe what you did in each part of the lab and explain your reasoning. Discuss any issues that did not work as expected, and reflect on the security of your implementation in Part III.
References. Include title, authors, publication venue, year, and URL when available.
Appendices. Optional. You may include screenshots, code snippets, or other supplementary material supporting the points made in the report.

Excluding the title page, references, and appendices, the report must not exceed **8 A4 pages**. Content beyond this limit will not be assessed. The final submission must be a PDF titled `TTM4135_lab_grpNN.pdf` and written in English.

Evaluation of the Lab Report

Your lab report is evaluated according to two components: **format and style** (10%) and **content** (90%). A strong submission should demonstrate clarity, structure, sound reasoning, and a solid understanding of the work you have completed.

- **Format and style (10%)**
  - Use 10 pt font size, normal line spacing, a one-column layout, and appropriate margins.
  - Ensure that your writing is grammatically correct, clear, and consistent.
  - The title page must include the names of all group members, the group number, the date, and the report title.
  - The reference list must be formatted according to established citation guidelines.

- **Content (90%)**
  - Present accurate and precise technical information.
  - Provide clear explanations and justifications for your observations, results, and decisions.
  - Demonstrate a logical and coherent understanding of the tasks carried out.
  - Include complete answers to all **Q**-questions, reproducing each question in full before answers.
  - Use reputable academic or technical references. Although the course textbooks may serve as a starting point, you are expected to consult additional high-quality sources. When using online materials, ensure their reliability. Searching through university library databases and relying on published literature is recommended.
  - Include reflections, insights, or original observations where appropriate.

1 Part I: Encrypted Email

Email communication is inherently insecure: messages may be intercepted, read, or altered during transmission, and sender addresses can easily be forged. To address these weaknesses, this part of the lab introduces the principles and practical use of encrypted email through PGP and its open-source implementation, GnuPG (GPG). Here, you will explore how the cryptographic concepts introduced in the lectures translate into real-world tools and workflows, and gain hands-on experience applying them to secure digital communication.

1.1 Historical Background

PGP (Pretty Good Privacy) is one of the earliest widely adopted tools for securing digital communication. Its history highlights both the technical challenges and the political tensions associated with early Internet cryptography.

PGP was first released in 1991 by Phil Zimmermann as a command-line program supporting symmetric encryption. Shortly afterwards, version 2 introduced asymmetric RSA functionality, which contributed to PGP's rapid spread beyond the United States. At that time, strong cryptographic software was regulated as military technology, leading to a federal investigation of Zimmermann for allegedly exporting encryption software.

To avoid prosecution, Zimmermann published the full PGP source code in printed book form. Under U.S. free-speech protections, printed material could be legally exported, effectively placing PGP outside cryptographic export restrictions. In 1996, Zimmermann and a group of collaborators formed a company to continue developing what became PGP 3.

Over time, PGP emerged as the *de facto* standard for secure email. To ensure openness and avoid proprietary control over such an essential security technology, Zimmermann transferred the PGP specification to the Internet Engineering Task Force (IETF) in 1997. Since then, the OpenPGP standard has been maintained by the IETF, with the modern version documented in RFC 9580 [For25].

1.2 PGP vs. GPG

PGP refers to the open standard for encrypted email and secure messaging, formalized today as the OpenPGP specification. In this course, we use GnuPG (GPG), a widely adopted open-source implementation of this standard.

GPG provides a complete set of tools for managing cryptographic identities. It enables users to generate and manage keypairs, sign and verify data, and encrypt and decrypt messages. Although GPG is primarily a command-line tool, it integrates with a variety of applications—particularly email clients—to support secure communication workflows.

While GPG supports symmetric encryption, its primary and most powerful mode of operation is based on public-key cryptography. Once you possess another user's public key, GPG can verify signatures created by that user and encrypt messages intended for them. After generating your own public–private keypair, you can sign outgoing messages and decrypt those sent to you.

GPG organizes keys into a keyring, which contains both public and private keys. Users can freely import and export keys, and many rely on online keyservers to distribute public keys. These servers simplify the process of locating the keys of other users and sharing your own.

GPG offers a wide range of configuration options and command-line features that extend beyond the examples presented in this section. For a more detailed overview of its capabilities, we recommend consulting the official GnuPG user manual, which provides comprehensive documentation on key management, signing, encryption, and best practices for secure usage [Pro24]. The OpenPGP standard, specified in RFC 9580, may also serve as a useful reference when exploring the underlying protocols.

If you encounter unexpected behavior or difficulties while working with GPG, consider using tools such as Microsoft Copilot or other AI-based assistants to help interpret error messages, suggest command usage, or provide troubleshooting steps. These tools can assist in resolving issues efficiently, especially when experimenting with different GPG options or working in varied system environments.

1.3 Task: Generating a PGP Keypair

In this section, you will generate a PGP public–private keypair using GnuPG (GPG). This keypair will enable you to encrypt, decrypt, and sign messages securely.

A keypair can be created by running `gpg --gen-key`. When invoked without additional parameters, GPG uses its default configuration, which on modern systems typically generates an elliptic-curve keypair based on Curve25519. This results in a primary signing key and an associated encryption subkey suitable for general use.

During key generation, GPG will prompt you for your name and email address. For this course, you must enter your real name and your official NTNU student email address (`@stud.ntnu.no`), as these form part of your key's identity.

If you wish to explore more advanced key configuration options, you may start GPG in expert mode:

```
$ gpg --full-generate-key --expert
```

Each public key is uniquely identified by its *fingerprint*, which provides an unambiguous reference. You may also create additional subkeys associated with your master key, following the traditional separation of roles such as encryption and signing, or their elliptic-curve equivalents.

After generating your keypair, verify its presence in your local keyring by running:

```
$ gpg --list-keys
```

You should now see your newly created key and its corresponding subkeys.

> **Task 1:** Ensure that GPG is installed and functioning on your computer. If it is not already available, install it using `sudo apt install gpg` on Linux, `brew install gnupg` on macOS, or download it from `https://www.gnupg.org/download` on Windows. Once installed, generate a keypair using your NTNU student email address and real name as described above.

To export your public key in ASCII-armored form, use:

```
$ gpg --armor --export <your-key-fingerprint>
```

This ASCII-formatted public key is what you will share with others when they need to verify your signatures or encrypt messages intended for you.

1.4 Web of Trust

Once you have created your public–private keypair, you can sign messages and decrypt messages encrypted for you. However, this is only meaningful if others can be confident that your public key genuinely belongs to you. OpenPGP addresses this through its *web of trust* model.

Instead of relying on a single central authority to validate identities, the web of trust distributes this responsibility among users. Each user decides how much trust to place in another user's key and may sign that key to express confidence in the key–identity binding.

Example. Suppose Bob wants to send an encrypted message to Cathy. He finds a public key associated with her name on a keyserver, but cannot be certain it is authentic. Bob does, however, trust Alice whose key he has already verified. If Alice has signed Cathy's key, Bob may consider it trustworthy. If other trusted contacts—such as Eve—have also signed Cathy's key, Bob's confidence increases further.

This decentralized model reflects real-world trust relationships: people rely on the judgments of those they already trust. The system also supports hierarchical trust by allowing users to assign different trust levels to keys. For example, Bob may decide that any key trusted by Alice should be accepted automatically.

At events such as CCC in Germany or DEF CON in the United States, participants sometimes organize *key-signing parties*, where individuals verify each other's identities in person before signing keys. While informative, key-signing and any kind of partying are out of scope for this lab exercise.

1.5 Key Servers

Key servers function as public repositories for OpenPGP public keys. By uploading your public key to such a server, you make it easier for others to obtain your key when they need to verify your signatures or encrypt messages intended for you. Likewise, you may retrieve the public keys of collaborators from these servers when initiating secure communication.

Key servers do not validate identities or assign trust to the keys they store. Their role is strictly to distribute keys, leaving it to individual users to determine how much trust to place in any given key. Many key servers operate in synchronized pools to improve reliability and ensure that uploaded keys propagate across multiple servers.

Uploading your key. A public key can be exported in ASCII-armored form using the `--armor` and `--export` options. Many key servers provide a web interface where this ASCII block can be pasted directly. Be sure to upload *only your public key*, never your private key.

In this lab, the upload process is tied explicitly to your email address. You may upload your key using:

```
$ gpg --export <your-email>@stud.ntnu.no | curl -T - https://keys.openpgp.org
```

Remark: Replace the placeholder `<your-email>` with your actual NTNU student email prefix. Some key servers require email verification before publishing your key; if so, follow the instructions in the verification email you receive.

Downloading keys. Before encrypting a message for someone else, you must obtain their public key. This can be done using:

```
$ gpg --search-keys <recipient-email>
```

After importing a key, verify that it appears in your keyring:

```
$ gpg --list-keys
```

Remark: Most key servers also provide web interfaces that allow searching by fingerprint, email address, or user ID. This can be a convenient way to confirm that you have retrieved the correct key.

> **Task 2:** Using the key server at `https://keys.openpgp.org`:
>
> 1. Upload your own PGP public key using your NTNU student email address.
> 2. Download the public key of at least one group member and verify its presence in your keyring.
> 3. Download the PGP key associated with the lab: `ttm4135-security-lab@iik.ntnu.no`.

1.6 Operations Using GPG

Now that you have experience managing keys with GPG, you can begin using them for practical cryptographic operations. This section covers the core functionality you will rely on throughout the course: signing, encrypting, verifying, and decrypting messages.

1.6.1 Signing and Encrypting Messages

PGP supports both encryption and digital signatures. Encryption protects confidentiality, while signatures ensure integrity and authenticate the sender. Although independent, these operations are often combined in practice.

Signing. Signing a file or message with your private key allows others to confirm that the content is authentic and unmodified. GPG supports three signature formats:

- *Standard signatures*: create a signed, compressed version of the original file.
- *Clearsigned signatures*: keep the content readable and place the signature in an ASCII-armored block.
- *Detached signatures*: store the signature separately from the original file.

The commands below create clearsigned and detached signatures:

```
$ gpg --output messageC.sig --clearsign <message>
$ gpg --output messageD.sig --detach-sign <message>
```

Verification. To verify a signed file:

```
$ gpg --verify messageC.sig
```

For detached signatures, provide both the message and the signature:

```
$ gpg --verify messageD.sig <message>
```

Verification confirms both integrity and authenticity, assuming the sender's public key is trusted.

> **Task 3:** Exchange a signed message with one of your group members and verify that both of you can successfully check the signature.

Remark: Avoid opening signed files in editors that modify whitespace or line endings, as this can cause signature verification to fail.

Encryption. To encrypt a file, you must already have the recipient's public key. GPG uses hybrid encryption: it first generates a temporary symmetric key to encrypt the file, then encrypts that symmetric key with the recipient's public key.

Example:

```
$ gpg --encrypt --recipient <recipient-email> --output message.gpg <file>
```

> **Question 1:** Why does GPG generate a symmetric key for encrypting the message instead of encrypting the entire message directly with the recipient's public key?

You may also combine signing and encryption of a file:

```
$ gpg --encrypt --sign --recipient <recipient-email> --output message.gpg <file>
```

Decryption. To both check the signature (if there is one) and decrypt a message addressed to you:

```
$ gpg --output message --decrypt message.gpg
```

GPG automatically selects the appropriate private key.

> **Task 4:** Exchange a file with one of your group members that is both signed and encrypted. Confirm that you can decrypt the file and verify the signature.

1.6.2 PGP Certificate Revocation

The ability to revoke a key is an essential component of secure key management. If your private key is ever lost, compromised, or otherwise no longer under your control, the corresponding public key should not be trusted by others. OpenPGP addresses this through the use of *revocation certificates*—special signed statements that indicate that a key is no longer valid.

A revocation certificate must be created using the private key it is meant to revoke. For this reason, it is best practice to generate such a certificate immediately after creating a new keypair and then store it in a safe location. As long as the private key remains accessible, you may also generate a revocation certificate at any later time.

To generate a revocation certificate and save it as `revoke.asc`, run:

```
$ gpg --output revoke.asc --gen-revoke <your-email>@stud.ntnu.no
```

To apply the revocation locally:

```
$ gpg --import revoke.asc
```

You may then inspect your keyring to confirm the revocation status:

```
$ gpg --list-keys
```

Once the key is revoked locally, you must also make the revocation known to others. This is achieved by uploading the updated key (including the revocation certificate) to a key server:

```
$ gpg --send-key --keyserver https://keys.openpgp.org <your-fingerprint>
```

Remark: Replace `<your-email>` with your NTNU student email prefix and `<your-fingerprint>` with the fingerprint of the key you intend to revoke. *Do not upload a revocation certificate unless you truly intend to invalidate the key.*

Task 5: Generate a revocation certificate for your keypair and store it securely on your machine.

1.7 Comparing Signature Efficiency in PGP

PGP supports several digital signature schemes, most commonly RSA and DSA, provided that suitable keypairs have been generated beforehand. In this exercise, you will compare their storage requirements and performance when signing and verifying messages.

To carry out these measurements, you will first create two separate signing keys: one RSA key of size 2048 bits and one DSA key of the same size. These keys are intended strictly for experimentation and must not be uploaded to any public key server.

After generating the keys, you will evaluate their behaviour by signing and verifying both a long message (approximately 1 MB) and a short message (approximately 1 KB). You will then compare the resulting private key sizes, public key sizes, signature sizes, and the time required to sign and verify messages in each case.

Task 6:

1. Generate two signing keys: one 2048-bit RSA key and one 2048-bit DSA key. These keys must be used only for this experiment and must not be uploaded to any key server.
2. For each key, sign and verify both a long message (about 1 MB) and a short message (about 1 KB).

Question 2: How many bytes does PGP require to store the private signing key and the public verification key for each of the two signature types?

Question 3: How many bytes does PGP require to store the resulting signatures in each of the four test cases (RSA/DSA applied to long and short messages)?

Question 4: What are the average signing and verification times for each of the four cases?

Question 5: Discuss your findings based on the three measurements above. To what extent do your results align with the information presented in the lectures? Explain how the different key components (such as modulus, exponents, generators, and related parameters) are represented and stored in PGP.

1.8 Email with PGP

PGP is widely used to secure email communication. This can be done either by manually signing and encrypting messages using GPG, as demonstrated in previous exercises, or by integrating PGP support directly into an email client. The latter approach is generally preferred, as it streamlines the workflow and makes secure communication more accessible.

Several tools provide direct PGP integration into email environments, including GPG Suite for Apple Mail, Enigmail [Proa] for Thunderbird and Postbox, and Mailvelope [Prob] for browser-based email in Chrome and Firefox. These tools allow users to sign, encrypt, verify, and decrypt messages directly within their existing email applications.

When using PGP in practice, it is important to remember that message metadata — such as the *To*, *From*, and *Subject* headers — is not encrypted. Only the message body and attachments are protected. Encrypted communication also requires that all communicating parties use PGP. However, digital signatures can still be verified by recipients even if they do not use PGP for encryption, providing assurance about the sender's identity and the integrity of the message.

Question 6: What assurances does a user obtain when downloading your public key from the key server? How does the role of a key server in PGP differ from the role of a certificate authority in the X.509 ecosystem with respect to security guarantees?

Milestone 1: Create a text file (.txt) containing your group number and the full names of all group members. Sign the file with your private key, then encrypt it using the public key associated with `ttm4135-security-lab@iik.ntnu.no`. Send the resulting encrypted and signed file to that address.

The teaching assistants will use your submission to verify that you are able to correctly sign and encrypt messages. Ensure that:

- you send the email from the same address associated with the key you used to sign the file,
- your public key has been uploaded to a key server, and
- your key has not been previously revoked.

This milestone should be completed only by one member on behalf of the whole group. To get this milestone approved, you need to show a TA how you did this during lab hours, and they will confirm that they have received your email.

2 Part II: Installing a web server and obtaining a certificate

Before you can do the steps in this lab, you will have to activate a virtual machine on the NTNU SkyHiGh system. Login at https://skyhigh.iik.ntnu.no using your normal NTNU username and password, and perform the following steps:

Task 7:

1. Under *Compute* → *Key Pairs*, click "Import public key" to add your personal RSA key (if you have one), or create a new one using the system. This you will later use to authenticate to the server.
2. Under *Compute* → *Instances*, click "Launch Instance" to create the server you will use. Give it a name and make sure to link it to the key pair you created above. For 'source' you need to use the image ⚠'Ubuntu Server 22.04'⚠. Choose any "Flavour" that works, e.g., "gx1.1c1r". Click then on the blue button to create the instance.
3. Go to *Network* → *Security Groups* and edit the default group. Add a rule for HTTP, for HTTPS and for SSH. You do not have to change any fields, just use "Add Rule", pick the correct one from the dropdown and click "Add".
4. Go to *Network* → *Floating IPs*. You will have one IP address here. Click "Associate" and link it to the VM you created in the second step.

Remark: Do not 'release' the one IP-address you have under *Floating IPs*, since this one is linked to your group number. If you release it you might get a different one, and we will have to update our DNS record to make sure your group URL still works.

Remark: To access the SkyHiGh web interface when you are not on campus, use the VPN provided by NTNU. Read more about this at https://i.ntnu.no/wiki/-/wiki/English/Install+vpn.

What Just Happened?

In the previous task you have created a virtual machine *in the SkyHiGh system. This VM behaves just like a real computer on a real network — except both the computer and the network are simulated somewhere on NTNU's server cluster in Gjøvik. You have specified what your VM looks like (2), given it a network address so you can reach it (4) and you have told the system that some ports in the firewall should be open (3) to make sure you can actually access the server both via the web browser and the command line. We will use the keypair to authenticate: that means you can sign a message with your private key, and the system will use the public key you uploaded to verify that it is really you.*

The server domain you were assigned is present in the DNS record to give you a subdomain, so these are now your group's details:

Login details:		
Server IP:	`129.241.150.xxx`	(from step 4)
Server domain:	`http://ttm4135-v26-groupN.iaas.iik.ntnu.no`	(`N` the group number)

Log in using the command terminal. Use the correct IP and change the path to the private key if necessary:

```
$ ssh ubuntu@129.241.150.xxx -i ~/.ssh/id_rsa
```

2.1 Installing the Apache Web-Server

Now that you have a computer running, we will install Apache. Apache is a software suite that configures your server to host websites.

```
$ sudo apt-get install apache2
```

The apt-get command downloads Apache, verifies that the package is signed and runs the installation.

Task 8: Install Apache using the instructions above.

> **Question 7:** Who typically signs a software release like Apache? What do you gain by verifying such a signature?

If all is well, you can now navigate to either Server IP address or your Server domain in a web browser, and you will get a page that says "It works!"

2.2 The Web-Server Configuration

The main configuration file for Apache is called `apache2.conf`, and found in the `/etc/apache2` directory of your server. Take a look at it and try to figure out the purpose of some of the different commands inside it. In that directory you will also find a file called `ports.conf`, which will tell you which ports the server is currently using. The default port for an unencrypted HTTP-connection is port 80. For an encrypted HTTPS-connection this is port 443.

Whenever you change something in the config file you need to restart Apache by using:

```
$ sudo apachectl restart
```

If you want to turn if off and make your site inaccessible, you can do so by using:

```
$ sudo apachectl stop
```

Remark: We are not using an encrypted connection, so you should for now use `http` and not `https`.

Remark: By default, users are able to view a list of the content of all directories via the browser. You can add `Options -Indexes` to your config file to disable that list — not doing this is bad security practice.

2.3 Certificates

2.3.1 Authority vs. Trust

In the previous part of the assignment we learned about how we can verify keys in a web of trust where users make personal decisions on the truthfulness of keys. For the next part of the lab, we will work with a different type of trust distribution, namely a trust hierarchy.

When you use your web browser to contact a web server over a secure connection, your browser validates the identity of the server by checking that it actually possesses a valid certificate. These certificates are issued by a *certificate authority (CA)*, which is in turn certified by a higher-level authority. On the top of this certificate tree or certificate hierarchy we find the *root CA*, whose keys are hardcoded into your web browser when you downloaded and installed it.

Although the technical workings are similar, we use the term *certificate* instead of *key* to signify that we are using a hierarchy instead. Intuitively this makes sense: certificates are issued by an authority in real life, too.

Buying a certificate and having it signed by a CA typically used to cost a fee, which is why you had so many websites using unencrypted `http` instead of `https` just a few years ago. After some big events like the Edward Snowden revelations made the public more aware of the issues that unencrypted connections pose, a few nonprofit organizations started the *Let's Encrypt* project in 2016: Let's Encrypt is a CA which provides certificates for free to everyone. It is also the CA we will use in this course.

2.3.2 Setting Up a Certificate

We will use the Let's Encrypt CA to provide us with certificates to authenticate our web server. After requesting and downloading that certificate, we should tell Apache to turn on the SSL protocol, so users can connect using HTTPS. Of course, Apache should also know where to find our certificate.

By now this entire process can be done in a mostly automated way using the *Certbot* tool provided by the Electronic Frontier Foundation. When running, Certbot will request your info to put on the certificate, then place a file on your webserver to verify that you actually control the domain, and then upload the certificates onto your system.

> **Task 9:** Obtain and install a certificate by following `https://certbot.eff.org/instructions`.

However, you can still choose to only obtain a certificate from Certbot and install them manually. In this case, follow the instructions so you obtain a certificate chain file and a private key. Both files have the extension `.pem`.

You do not have to move them, but we do need to tell Apache where they are by editing the config file.

We will add three lines: the first line will enable SSL. Edit the second and third line to reflect the actual location of certificate and key with the locations Certbot gave you. Example:

```
SSLEngine on
SSLCertificateFile /long/path/given/to/you/by/certbot/fullchain.pem
SSLCertificateKeyFile /long/path/given/to/you/by/certbot/privkey.pem
```

If you then reboot Apache, you should be able to access the web site through `https` and obtain a padlock (or similar) in your web browser.

Task 10: Obtain and install the certificate, edit the config file, restart Apache and verify that you can now connect to your site via a secure `https` connection.

Question 8: Why did you obtain a certificate from Let's Encrypt instead of generating a self-signed one yourself?

Question 9: What does Let's Encrypt have to verify, and how do they do it, before they issue a certificate?

Question 10: What steps does your browser take when verifying the authenticity of a web page served over `https`? Give a high-level answer.

Question 11: Have a look at the picture in Figure 1 below. What does `TLS_DHE_RSA_WITH_AES_128_CBC_SHA` say about the encryption? Address all eight parts of the string.

Question 12: The screenshot in Figure 1 is obviously from a few years ago. Do you think the encryption specified by this string is still secure right now? Motivate your answer.

Task 11: Check the rating of your server using SSL Labs: `https://www.ssllabs.com/ssltest`. Make any changes necessary to obtain an A+ rating. (Hint: you probably need to enable HSTS (strict transport security) before this works.)

Question 13: What restrictions on server TLS versions and ciphersuites are necessary in order to obtain an A rating at the SSL Labs site? Why do the majority of popular web servers not implement these restrictions?

Milestone 2: Show that you have achieved an A+ rating by the SSL Labs. To get this milestone approved, you need to demonstrate for a TA how you did this during lab hours.

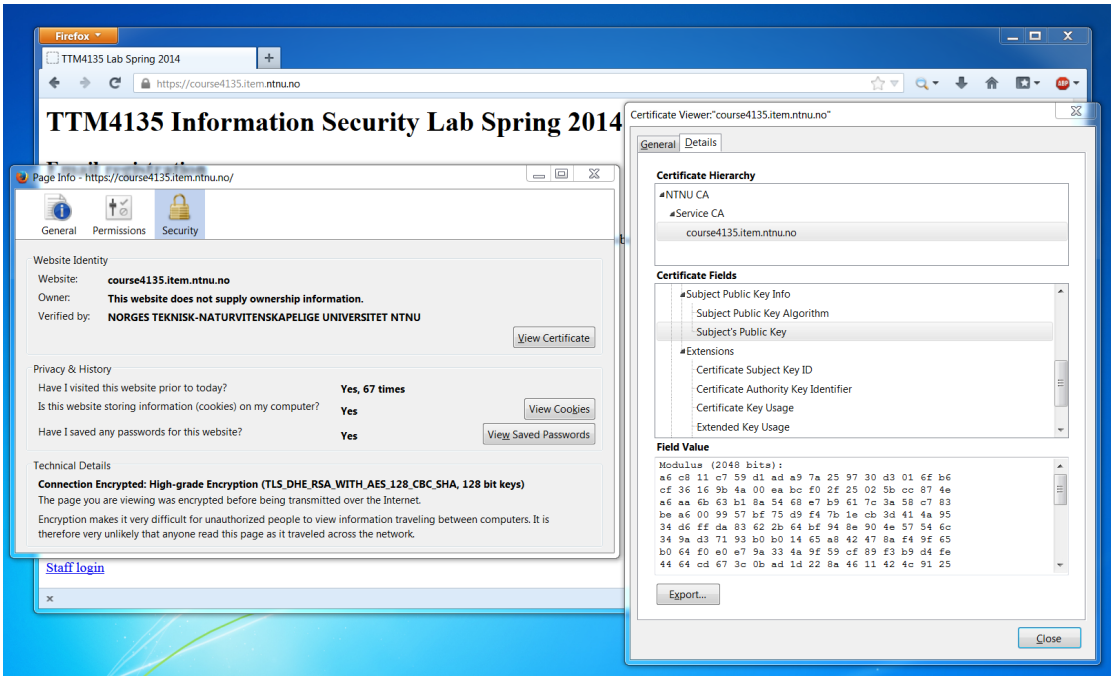

Figure 1: Information about a TLS connection to the former course site.

3 Part III: Examining TLS Traffic

In this final part you capture TLS traffic to see how it is structured and how it depends on the configuration in the server and client. You will also decrypt traffic directly from the intercepted traffic using the pre-master secret when using TLS with RSA (key transport) key exchange.

Protecting the confidentiality of user application data is one fundamental property of a secure TLS connection. Many communication networks are easy to tap so we should assume that an attacker is able to observe the traffic. In addition, servers and clients may be compromised at certain times and keys, especially long-terms keys, may become known to the attacker.

In the following task you will capture traffic which you send yourself from a client web browser to your web server and analyze it to see the details of the TLS keys and the ciphertext of application data.

You should use the open-source network protocol analyzer Wireshark [Wir26] to capture the traffic on the TLS connection that you set up. Wireshark is a powerful tool and has built-in support to allow you to filter out TLS traffic from your other network traffic. You do not need to use any complex features of Wireshark, but if you have not used it before then you should allow some time to get used to the interface. Wireshark comes with detailed documentation and support features.

The Illustrated TLS Connection [Dri] may be helpful in identifying some protocol fields. Note that there are two different versions of this resource: be sure that you use the one for TLS 1.2, not the one for TLS 1.3. However, you should also note that Illustrated TLS Connection shows a TLS 1.2 handshake with Diffie–Hellman, while we examine a connection using RSA encryption in this assignment.

Task 12: In this task you will capture traffic from a TLS 1.2 session between a browser and your server. You can check that the traffic follows the TLS protocol specification.

1. Change the acceptable ciphersuites on a client browser so that `TLS_RSA_WITH_AES_256_CBC_SHA` is negotiated between the browser and your server. If you use the Firefox browser you can edit the acceptable cipher suites by typing `about:config` in the navigation bar and searching for "ssl3.rsa_aes_256_sha". You may also have to disable TLS 1.3 by setting the maximum version to be "3". Note that you will get a warning about changing the configuration – if you are not confident about this you should not make the changes on your normal browser, or remember to reset them afterwards. You may also use any other browser in this task. Another option is to set this up on the

server side by editing your configuration file to only support TLS 1.2 and RSA for the handshake.
2. Use Wireshark [Wir26] to capture the traffic in a session between the prepared client and your server, including both the handshake and application data. The application data should include a simple text string sent by you from the client to the server. Wireshark can be used to filter out the TLS traffic and it will also recognize the different TLS message types. You can also match the bytes with the examples used in the Illustrated TLS Connection.

Question 14: What are the values of the client and server nonces used in the handshake? How many bytes are they? Is this what you expect from the TLS 1.2 specification?

What is the value of the encrypted pre-master secret in the client key exchange field sent to the server? Is this the size that you would expect given the public key of your server?

Task 13: Continue from Task 12 and now find the (pre)-master secret from which the traffic encryption keys are derived.

1. Find either the pre-master secret or the master secret used in the session. You are suggested to use the instructions on the Wireshark wiki to help you do to this: https://gitlab.com/wireshark/wireshark/-/wikis/TLS.
2. Decrypt the traffic using Wireshark and check that it is the same as was sent from the client.

Question 15: What was the value of the (pre)-master secret in the session that you captured? Write down the value, and whether it is the pre-master secret, or the master secret. Is this the size (in bytes) that you expect from the TLS specification? Explain how Task 13 shows that this session does not have forward secrecy.

Milestone 3: Show that you are able to set up a secure TLS connection with your server using ciphersuite `TLS_RSA_WITH_AES_256_CBC_SHA`. To get this milestone approved, you need to demonstrate for a TA how you did this during lab hours.

References

[Dri] Michael Driscoll. The illustrated TLS connection. https://tls12.xargs.org.

[For25] Internet Engineering Task Force. Openpgp. https://datatracker.ietf.org/doc/html/rfc9580, 2025.

[Proa] The Enigmail Project. Enigmail: A simple interface for OpenPGP email security. https://www.enigmail.net/index.php/en.

[Prob] The Mailvelope Project. Mailvelope. https://mailvelope.com/en.

[Pro24] The GnuPG Project. The gnu privacy guard manual. https://www.gnupg.org/documentation/manuals/gnupg, 2024.

[TDHN] Simon Tatham, Owen Dunn, Ben Harris, and Jacob Nevins. PuTTY download page. https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html.

[Wir26] Wireshark. Wireshark download page. https://www.wireshark.org, 2026.

A Collaboration Tools

Some students have previously experienced challenges collaborating remotely, particularly when they are unable to work on the same physical machine. This section provides a brief overview of tools that may facilitate communication and joint work during the lab. The tools listed are suggestions only; you are free to use any alternatives you prefer.

Video Communication and Screen Sharing A variety of modern communication platforms support video calls, voice chat, and screen sharing, all of which can be useful when collaborating on a single terminal or demonstrating configuration steps. Examples include: Signal, WhatsApp, Zoom, Discord, Slack, Microsoft Teams, Skype, TeamViewer.

Most of these tools support screen sharing, and some (e.g. TeamViewer) also allow remote input, enabling team members to type or control the mouse during collaboration.

Shared Terminal Environments For tasks that require multiple people to work directly in the same shell session, the following tools can be particularly effective:

Teleconsole Teleconsole provides a shared terminal environment similar to TeamViewer, but focused specifically on command-line collaboration. One team member starts a Teleconsole session and connects to the server via `ssh`. Other members can then join the same session and interact with the terminal collaboratively.

Tmux Terminal Multiplexer (tmux) allows multiple virtual terminal windows ("tabs") to run within a single terminal session. Only the server needs to have `tmux` installed. A typical workflow is:

1. One team member connects to the server via `ssh` and starts a session:
   ```
   tmux
   ```
2. Other members connect to the server and join the session:
   ```
   tmux attach
   ```

All participants can type and execute commands simultaneously. Additional features include:

- Create a new tab: `Ctrl-B, C`
- Switch tabs: `Ctrl-B, <tab number>`
- Close a tab: `exit`
- Detach from a session without closing it: `Ctrl-B, D`

Detached sessions continue running in the background. All active `tmux` sessions can be listed with:

```
tmux ls
```

To reattach to a specific session:

```
tmux attach -t <session-number>
```

If a new `tmux` session is started using only `tmux` while another session is running, an entirely separate session is created.

B Apache Configuration Structure

Students commonly encounter challenges in Part II of the lab, particularly related to understanding how Apache processes configuration files and how configuration directives interact. This section provides a concise yet structured introduction to the Apache configuration hierarchy as used on Debian- and Ubuntu-based systems. Although not required once Part II is completed, the discussion may deepen your understanding of Apache's configuration model.

Apache may be installed under different binary names depending on the operating system, such as `apache`, `apache2`, `http`, or `httpd`. On Debian- and Ubuntu-based distributions, the binary and configuration layout consistently use the `apache2` naming scheme.

All primary configuration files are located under:

```
/etc/apache2/
```

The main configuration file is:

```
/etc/apache2/apache2.conf
```

This file is well documented and provides an overview of the entire configuration structure. Two fundamental principles govern how Apache processes configuration data:

- **Sequential processing:** Configuration directives are evaluated in order; if a directive is defined multiple times, the final definition overrides earlier ones. This behavior follows Apache's sequential parsing model.
- **File inclusion:** Additional configuration files are incorporated via the `Include` and `IncludeOptional` directives. When a file is included, its contents are treated as though they were inserted directly into `apache2.conf`.

On Debian/Ubuntu systems, the configuration hierarchy is modular and structured as follows:

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

Files within each directory are processed in lexicographic order because glob expansion is performed before Apache reads the files. Thus, if two files inside the same directory define the same directive, the one whose filename comes last alphabetically takes precedence.

Near the end of the `apache2.conf` file, you will find the following include statements:

```
IncludeOptional conf-enabled/*.conf
IncludeOptional sites-enabled/*.conf
```

Any additional configuration placed *after* these lines will override directives defined in any included files.

Virtual Hosts

Apache's `VirtualHost` (VHost) mechanism enables a single server to host multiple independent websites or services. A VHost block acts similarly to an XML or HTML element: directives inside a `<VirtualHost>...</VirtualHost>` container apply exclusively to that VHost and do not affect global settings or other VHosts. This behavior is explicitly documented in the Apache HTTP Server manual.

For example, a configuration file `my-site.conf` inside `sites-enabled` may contain:

```
<VirtualHost _default_:443>
ServerName a.mysite.com
DocumentRoot /var/www/html-site-a/
... additional settings ...
</VirtualHost>

<VirtualHost _default_:443>
ServerName b.mysite.com
DocumentRoot /var/www/html-site-b/
... additional settings ...
</VirtualHost>
```

If both domains resolve to the same server, Apache will select the appropriate VHost based on the `ServerName` or `ServerAlias` directives and deliver different content depending on the requested hostname. Each VHost may therefore define its own `DocumentRoot` and its own SSL parameters, logging directives, or other configuration options.

These VHosts may reside in the same file or be separated into multiple files; the modular layout used on Debian/Ubuntu imposes no restrictions on this.

Certbot-Generated Configuration

When using Certbot with the `-apache` flag, the tool typically generates two configuration files inside `sites-enabled`:

1. A redirect configuration (often named `*-le-redirect.conf`), forwarding HTTP traffic to HTTPS.
2. A primary HTTPS VHost configuration, containing a `<VirtualHost _default_:443>` block.

Certbot also includes a Let's Encrypt SSL configuration fragment, such as:

```
Include /etc/letsencrypt/options-ssl-apache.conf
```

This file should not be modified directly. Instead, if you wish to override specific SSL parameters—such as `SSLProtocol` or `SSLCipherSuite`—you may simply specify your preferred directives *after* the include line within the VHost. Due to Apache's sequential configuration processing, your definitions will override those contained in the included file.