Tutorial 7 (Authentication and Key Distribution)

In this tutorial, we will complete some practical exercises on digital certificates.

🔐 Introduction: What Are Digital Certificates?

Digital certificates are a core component of modern internet security.

They are used to:

- ✅ Authenticate the identity of websites and servers
- ✅ Establish encrypted channels (e.g. HTTPS/TLS)
- ✅ Enable trust without sharing secrets in advance

Most secure protocols today — including HTTPS, TLS, VPNs, and code signing — rely on digital certificates to prevent impersonation and man-in-the-middle attacks.

📄 What Is a Digital Certificate?

A **digital certificate** is a structured, signed document that binds a **public key** to a verified identity (such as a domain name or an organization).

It follows the **X.509 standard** and is typically issued by a **Certificate Authority (CA)**.

🔍 What Does a Certificate Contain?

A digital certificate includes key fields required to verify identity and establish secure connections. Here's a simplified overview:

Field	Description
Subject	The identity being certified (e.g., domain)
Issuer	The Certificate Authority (CA) that issued it
Public Key	Used to establish secure communication
Validity Period	Start and end time for certificate validity
Serial Number	Unique identifier for the certificate
Signature	CA's cryptographic signature over the contents

> 🔎 **Note:** Real certificates include additional fields (e.g., extensions, key usage, algorithms) that control behavior and compatibility across systems.

🧪 Common Formats You Will Encounter

- `.pem` — Base64-encoded certificate, used by OpenSSL, Apache, Nginx
- `.cer` — Can be **PEM or DER** (binary); used in Windows and Java environments
- `.crt` — Often identical to `.pem`; varies by convention

🎯 Why This Matters

If you work with secure websites, APIs, VPNs, or authentication protocols, you must understand how certificates:

- Prove identity
- Enable encryption
- Fail when expired, revoked, or tampered with

📘 In This Tutorial

You will:

- Examine and decode real certificate files
- Understand the differences between `.pem`, `.cer`, `.crt`, and `.der`
- Use OpenSSL to parse, validate, and convert certificates
- Learn how integrity and trust are established using certificate chains

1) Exploring Digital Certificates

a) Using OpenSSL

OpenSSL is a command-line tool that can be used to display the contents of an X.509 certificate. We provide a certificate file: `whoami.pem`. Use `openssl x509 -help` to find out how to display the certificate in a human-readable format. Then, answer the following questions:

- Who is the issuer? Look up the organization.
- Who is the subject (i.e., the certified entity)?
- What is the algorithm used for the public key?
- What is the key length?
- Are Certificate Revocation Lists (CRLs) defined?
- Is Online Certificate Status Protocol (OCSP) defined?
- What hash function is used for the signature?

b) Let's try one more: `fred.cer`

Credit: Applied Crypto by Bill Buchanan

Examine the certificate and find the following information:

- Serial number
- Effective date

- Name (i.e., the Subject)
- Issuer
- What is the **CN** (Common Name) used for?
- What is the **OU** (Organizational Unit) used for?
- What is the **O** (Organization) used for?
- What is the **L** (Locality) used for?

> **Note:**
> The `fred.cer` file is in **DER** (Distinguished Encoding Rules) format, a binary encoding often used with `.cer` extensions.
> This differs from the `.pem` format we examined earlier, which is a **Base64-encoded** version of the certificate wrapped with `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` markers.
> PEM is human-readable; DER is not. Tools like `openssl` can be used to decode `.cer` files for analysis.

2) Exploring Digital Certificates in the Internet

`openssl` can also be used to download and inspect certificates from a web server. The command
`openssl s_client -connect x.com:443` will display all certificates used by `x.com`.

Run the tool to inspect the certificate configurations of the following organisations and answer the questions below:

- Which Certificate Authorities (CAs) are used by `microsoft.com`, `x.com`, and `my.gov.au`?
  Note: It is sufficient to inspect the first few lines of the output.

- Are intermediate certificates used? Speculate why.
  Note: Again, the first few lines of the output should be enough.

Tip: After running the command, you may need to stop it manually using `Ctrl+C` (in the command line) or the **Stop** button if using a Jupyter notebook cell.

3) Issues in Certificates

One possible cause of the infamous invalid certificate error is a mismatch between the domain name listed in the certificate and the actual domain name of the website.

Using your browser, try to visit:

- http://testfire.net/
- https://testfire.net/

(This is a test website maintained by IBM for demonstrating common web application vulnerabilities and website defects.)

The first connection is not encrypted, so you will be able to visit the website without any issues. However, the second attempt uses encryption, and you will receive a certificate error: `NET::ERR_CERT_COMMON_NAME_INVALID`.

a) Use openssl and try to find the problem

b) Use SSL Labs to Verify Your Answer

SSL Labs is a useful tool for troubleshooting SSL certificate issues. Try running a scan for https://testfire.net and review the results.

Then, run a scan for another site such as www.sydney.edu.au for comparison.

Verifying the Certificate Chain of Trust with OpenSSL

Before we verify the certificate chain using `openssl`, it's important to understand what the **chain of trust** means in the context of TLS/SSL certificates.

🔐 What is a Certificate Chain of Trust?

When you connect securely to a website (e.g., `https://example.com`), your browser or client needs to verify that the server's TLS certificate is **authentic** and **issued by a trusted authority**.

This verification doesn't happen in isolation. Instead, it relies on a *chain of trust* that connects the server's certificate to a **trusted root certificate authority (CA)** that your system already knows about.

The chain typically includes:

1. **Leaf certificate (Server certificate)** – belongs to the website (e.g., `x.com`)
2. **Intermediate certificate(s)** – issued by a trusted root CA, used to sign the server certificate
3. **Root certificate** – issued by a trusted Certificate Authority, pre-installed on your system

If any certificate in the chain is missing or invalid, verification will fail.

🧪 Using OpenSSL to Verify the Chain of Trust

We can use OpenSSL tools to manually inspect and verify the certificate chain returned by a server.

We'll perform the following:

1. **Fetch the full certificate chain from the server**
2. **Extract the certificates**

3. **Use OpenSSL's `verify` command to validate the chain**

We'll see this in action using the domain `x.com` in the steps below.

4) Use `openssl` to verify the chain of trust for `x.com`

You will need to locate the root certificate store on your Ubuntu system.

5) Expired Certificates

Since September 2020, major browser vendors such as Google, Apple, and Microsoft have enforced stricter rules regarding certificate validity. Publicly rooted digital certificates that expire more than 13 months (or 398 days) from their creation date are no longer accepted in modern web browsers. This marked a significant shift from earlier practices, where certificates were often valid for 8–10 years. The move was intended to enhance security across the web ecosystem.

a) Explain how this policy improves the security of websites and users.

b) What practical or usability challenges may have arisen during the early stages of this transition?

5) Homework 📚

Please read about the following two concepts:

- **Self-Signed Certificates**
- **SSL Proxies**