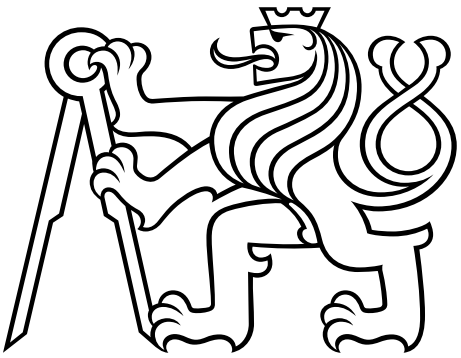
Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science

Enhancing security and privacy of computer networks with OpenBSD as network operating system on router

Bachelor thesis

Matyáš Vohralík



Supervisor: Ing. Pavel Troller, CSc.

January 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vohralík** Jméno: **Matyáš** Osobní číslo: **474536**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Zvýšení zabezpečení a ochrany soukromí počítačové sítě s pomocí OpenBSD v roli síťového operačního systému na routeru

Název bakalářské práce anglicky:

Enhancing security and privacy of computer networks with OpenBSD as a network operating system of a router

Pokyny pro vypracování:

V teoretické rovině popište využití systému OpenBSD v roli síťového operačního systému, nasazeného na směrovači počítačové sítě domácnosti či organizace a rozšiřujícího běžně dostupné funkce obdobných zařízení o technologie zvyšující úroveň zabezpečení a ochrany soukromí uživatelů.
Popište zvolené technologie a jejich vzájemnou integraci.
Porovnejte řešení založené na systému OpenBSD s jinými operačními systémy použitelnými pro uvedený účel.
Následně vytvořte funkční řešení na jednodeskovém počítači platformy arm64 a vyhodnoťte jeho parametry.

Seznam doporučené literatury:

[1] R. L. Rivest, A. Shamir, L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, 1978
[2] Michael W. Lucas. Absolute OpenBSD. 2nd Edition., 2013
[3] Peter N. M. Hansteen. Book of PF, 3rd Edition, 2014
[4] Larry Peterson, Bruce Davie. Computer Networks: A Systems Approach, 2012
[5] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone. Handbook of Applied Cryptography, 2001
[6] Brandon Palmer. Secure Architectures with OpenBSD, 2004
[7] Marshal Kirk McKusick, Keith Bostic, Michael J. Karels, John S. Quarterman. The Design and Implementation of the 4.4BSD Operating System, 1996
[8] Steve Pate. UNIX Filesystems: Evolution, Design, and Implementation, 2003
[9] Jason A. Donenfeld. WireGuard: Next Generation Kernel Network Tunnel, 2020

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Pavel Troller, CSc. katedra telekomunikační techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **07.02.2022** Termín odevzdání bakalářské práce: **10.01.2023**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Pavel Troller, CSc. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of the university thesis.

In Prague, January 10, 2023

Matyáš Vohralík

Acknowledgment

I would like to thank everyone who has supported me during the creation of this thesis. To my supervisor, Ing. Pavel Troller, CSc., for all the advice, and to my family for all the support.

Abstract

The aim of this thesis is to design complex high-level security measures for a model network of organization or family. The model network consists of three geographically dislocated subnets interconnected via VPN with unified and centralized Wi-Fi EAP-TLS authentication. A center point of the network is router built on OpenBSD with additional functions such as dynamic firewall, DNS server with AdBlock built on NSD & unbound, FreeRADIUS and OCSP responder for certificate authority. After a brief introduction to symmetric and asymmetric cryptography, the text describes the creation of robust two-tier Elliptic curve Public Key Infrastructure built on HSM module Gemalto SafeNet 5110 CC with OpenSSL toolkit. Each particular component of the solution is analyzed immediately during the configuration process. The last part of this thesis introduces single board low power arm64 platform NanoPi R4S and the installation of OpenBSD. All configuration files, demonstrative certificate authority, photos and diagrams are in the attachment.

Keywords: OpenBSD, router, security, Wi-Fi, Wireguard, DNS, adblock, RADIUS, EAP-TLS, PKI, Certificate authority, HSM, SafeNet eToken 5110

Abstrakt (in Czech)

Tato bakalářská práce se zabývá komplexním zabezpečením modelové organizační respektive rodinné počítačové sítě tvořené třemi geograficky dislokovanými podsítěmi propojenými technologií VPN s jednotnou a centralizovanou Wi-Fi autentizací protokolem EAP-TLS. Nedílnou součástí je softwarový návrh síťového routeru založeného na platformě arm64 a operačním systému OpenBSD doplněném o dynamický firewall, DNS server s funkcí AdBlock postaveném na kombinaci NSD a unbound, FreeRADIUS server a OCSP respondér pro certifikační autoritu. V další části jsou čtenáři stručně představeny základy symetrické a asymetrické kryptografie, aby se práce následně mohla v detailu věnovat zřízení dvouvrstvé infrastruktury veřejných klíčů vytvořené pomocí kryptografického toolkitu OpenSSL a vydání všech potřebných certifikátů. Je zde představen hardwarový bezpečnostní modul - USB token Gemalto SafeNet 5110 CC a jeho použití pod systémem Linux. Následně je na klíčích v něm vygenerovaných vystavěna robustní certifikační autorita založená na klíčích eliptické křivky prime256v1. V závěrečné části je představen ukázkový nízkopříkonový jednodeskový počítač NanoPi R4S disponující achitekturou arm64 a popsána instalace systému OpenBSD na tomto zařízení. Jednotlivé části celého řešení jsou analyzovány postupně v průběhu konfigurace. Součástí je také demonstrace připojení Linux, Android a Windows zařízení do sítě. Veškeré konfigurační soubory, klíče a certifikáty certifikační autority, diagramy a fotografie jsou součástí přílohy.

Klíčová slova: OpenBSD, router, zabezpečení, Wi-Fi, Wireguard, unbound, nsd, DNS, AdBlock, RADIUS, EAP-TLS, PKI, infrastruktura veřejných klíčů, Certifikační autorita, HSM, SafeNet eToken 5110

Contents

Chapter 1

Introduction

1.1 Motivation

Many contemporary Small Office/Home Office (SOHO) network devices suffer from common weakness. They stop receiving official firmware updates pretty soon after they are sold, even though its hardware is not yet outdated. Another drawback is poor logging and statistics options, which makes them hardly debuggable. Lack of Secure Shell (SSH) connection support ties the users to web Graphical User Interface (GUI) interface, Hopefully, there are few possibilities left for the owner.

1.2 Alternatives

Let's take a look at projects that follow similar goals as OpenBSD either in security or network field.

1.2.1 OpenWrt

OpenWrt is a Linux distribution well-known for being created and maintained especially for embedded network devices (Wrt stands for Wireless Router). It can run many platforms requiring at least 8MB Flash and 64MB RAM space.[1] Many SOHO devices meet these requirements and are on the official Table of Hardware. OpenWrt is secure-by-default and receives updates frequently.

1.2.2 Turris project

Turris is quite a new project maintained by CZ.NIC, the registry of the Czech national top-level domain .CZ. The project focuses both on software and hardware. They produce Turris Omnia all-in-one box router and modular Turris MOX.[2] As for software, Turris develops Turris OS, a Linux distribution adding some additional tools at the top of OpenWrt. It has many attractive features:

- Sentinel system
  - threat detection
  - dynamic firewall
  - Honeypot as a Service (HaaS) proxy application that forwards incoming WAN SSH traffic to CZ.NIC HaaS server that represents end-user operating system allowing the attacker to log in and do some job (execute commands or download malware) that is recorded and further analyzed.
- Knot DNSSEC resolver (part of Knot Domain Name System (DNS) project maintained by CZ.NIC)
- reForis router administration interface (developed by Turris project)
- LuCI configuration interface (inherited from OpenWrt)
- multiple applications that may be installed and configured easily via reForis interface:
  - Tools for network analysis
  - Nextcloud
  - OpenVPN client/server
- Wireguard is supported as well but with no Graphical User Interface (GUI) support yet.

1.2.3 Mikrotik RouterOS

Mikrotik is quite odd in this list provided by the fact that they do not distribute opensource operating system, that cannot run non-Mikrotik devices, but even the low-end devices, like Mikrotik hAP Wi-Fi routers, come with fully-featured RouterOS and receive updates as well as the high-end devices. Mikrotik is listed here because we will use Mikrotik Wi-Fi Access Point as 802.1X authenticator.

Chapter 2

OpenBSD Operating System

2.1 OpenBSD introduction

2.1.1 The OpenBSD Foundation

OpenBSD is a free, multi-platform 4.4 BSD-based UNIX-like operating system, which developers focus on security, correctness, and standardization.[3] OpenBSD developers pose high requirements on code quality so OpenBSD is sometimes called "a C language documentation". The quality and correctness of its documentation are appreciated often as well. Besides the OpenBSD operating system, the OpenBSD Foundation includes many related projects - OpenSSH, OpenBGPD, OpenNTPD, OpenSMTPD, LibreSSL[4] that are well-known in whole UNIX-like world.

2.1.2 OpenBSD security features

The OpenBSD operating system implements multiple security that makes it suitable for security-critical use cases like network router or firewall.

2.1.2.1 Address space layout randomization

OpenBSD implements randomized `malloc()`, `mmap()`, and kernel randomization, which is a special case of Address space layout randomization (ASLR). During every boot, OpenBSD relinks its kernel so every object that gets linked into the binary gets the random address. This process is signaled by the message "reordering libraries" and may slow down the boot rapidly when running on single-core CPUs or on a single processor kernel.

2.1.2.2 Privilege separation and chroot jailing

OpenBSD runs nearly all of the standard system daemons with privilege separation (each daemon has its own user/group).
Also, most of the daemons are configured by default to change root right after start.

2.1.2.3 strlcpy() and strlcat()

According to the Common Weakness Enumeration (CWE), the out-of-bounds write/read regularly tops the ranking of software weaknesses.[5] In C/C++ language, this vulnerability is often introduced by misuse of string manipulation functions `strcpy()` and `strcat()`. The OpenBSD project implemented safe versions of these functions, called `strlcpy()` and `strlcat()`.

```
size_t strlcat(char *, const char *, size_t)
size_t strlcpy(char *, const char *, size_t)
```

src/include/string.h

There are multiple safe implementations of `strcpy()` and `strcat()` from different authors. RedHat Developer article *"Efficient string copying and concatenation in C"*[1] clearly describes the differences.[6]

2.1.2.4 LibreSSL

The OpenBSD foundation develops its own fork of OpenSSL, which has been improved significantly. This topic is further described in the *Certificate authority and encryption* section.

2.1.3 OpenBSD limitations

2.1.3.1 Giant lock

OpenBSD is one of the operating systems that still use the Giant lock approach which makes each kernel call mutually exclusive so only a single kernel call may be executed at the same time. This also means only one CPU core may be used for kernel calls. Giant lock approach is implemented by `__mp_lock` in OpenBSD.

```
void __mp_lock(struct __mp_lock *);
void __mp_unlock(struct __mp_lock *);
int __mp_release_all(struct __mp_lock *);
```

sys/mplock.h

2.1.3.2 Lack of SMP support for armv7

As for version 7.2, OpenBSD does not support Symmetric Multiprocessor kernel mode (SMP) on the armv7 platform.[7] This affects some Raspberry PI-like devices (the ones with Allwinner H2 for example) so only one core works. Since these devices have multicore low-power CPUs with low core frequency, this lack is sensible and slows down especially kernel randomization (reoder of libraries) during boot.

[1] https://developers.redhat.com/blog/2019/08/12/efficient-string-copying-and-concatenation-in-c

Chapter 3

Model network

3.1 Model network

Let's consider the following scenario imitating real-world example. We have:

- 3 dislocated sites A, B, C with:
  - access to the Internet, one (A) with fixed, public IP address
  - WiFi station
  - one client device in each site
- 2 devices m_1, m_2 somewhere in the Internet

What we need:

- to connect the networks A, B, C via VPN
- to have the same WiFi SSID and 802.1x authentication in each site
- to run DNS service

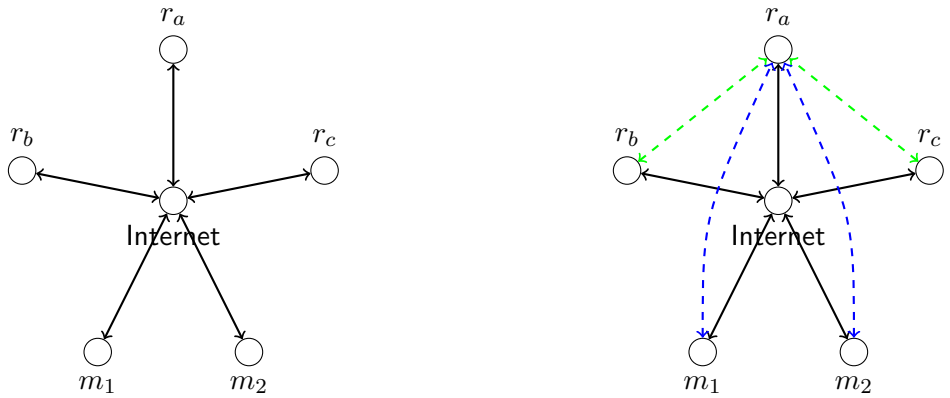


Figure 3.1: Simplified network model

3.1.1 Subnetting

IP address consists of two parts - *network prefix* and *host identifier*. A network prefix is a group of the most significat bits of an IP address, host identifier is the rest of bits. Hosts with the same network prefix are considered to be in the same network or subnetwork.[8] The network may also be characterized with a bitmask (called *netmask*) or *prefix*. Subnetting is the practice of dividing a network into two or more subnetworks. The reasons to divide networks into smaller subnetworks are numerous.

- Network performance - traffic is split which minimizes congestion probability
- Maintenance and administration responsibility distribution - computer networks must reflect organization's hierarchy with its policies
- Privacy - broadcasts are being distributed only within a subnet
- Troubleshooting - traffic monitoring and troubleshooting is easier in smaller networks
- Security - ensuring security includes the audit and network traffic monitoring

In our example network, we will divide the network to subnetworks also in favor of layer 3 VPN.

3.1.1.0.1 IPv4 private address space collision considerations Let's consider a typical situation when there is a client device (laptop) with Internet access, connected to some public IPv4 Wi-Fi network that uses IPv4 private subnet address space. Let's call this network the network X. There is a chance that X's address space will collide with the address space of either networks A, B, C or the client-to-site VPN network, and the routing will not work. Although there is no easy way to avoid such a situation by 100%, we can reduce the chance substantially when we use "random" network spaces for our networks. So instead of using 192.168.1.0/24, 192.168.2.0/24, and so on, we can pick subnets from 10.0.0.0/8 or 172.16.0.0/12 private address space - 10.237.176.0/24, 10.237.177.0/24 for instance.

Table 3.1: Subnetting the given IP range

IP range: 172.16.160.0/21, Hosts: $2^{11} = 2046$

ID	Description	Required IPs	Size 2^n	Available IPs	Prefix	Mask	Network address	Available addresses	Broadcast
A	**Site A**	500	2^9	510	/23	255.255.254.0	172.16.160.0	172.16.160.1-172.16.161.254	172.16.161.255
B	**Site B**	200	2^8	254	/24	255.255.255.0	172.16.162.0	172.16.162.1-254	172.16.162.255
C	**Site C**	200	2^8	254	/24	255.255.255.0	172.16.163.0	172.16.163.1-254	172.16.163.255
X	**VPN client-to-site**	100	2^7	126	/25	255.255.255.128	172.16.164.0	172.16.164.1-126	172.16.164.127
Y	**VPN site-to-site**	5	2^3	6	/29	255.255.255.248	172.16.164.128	172.16.164.129-134	172.16.164.135
	Free space							*172.16.164.136-172.16.167.254*	

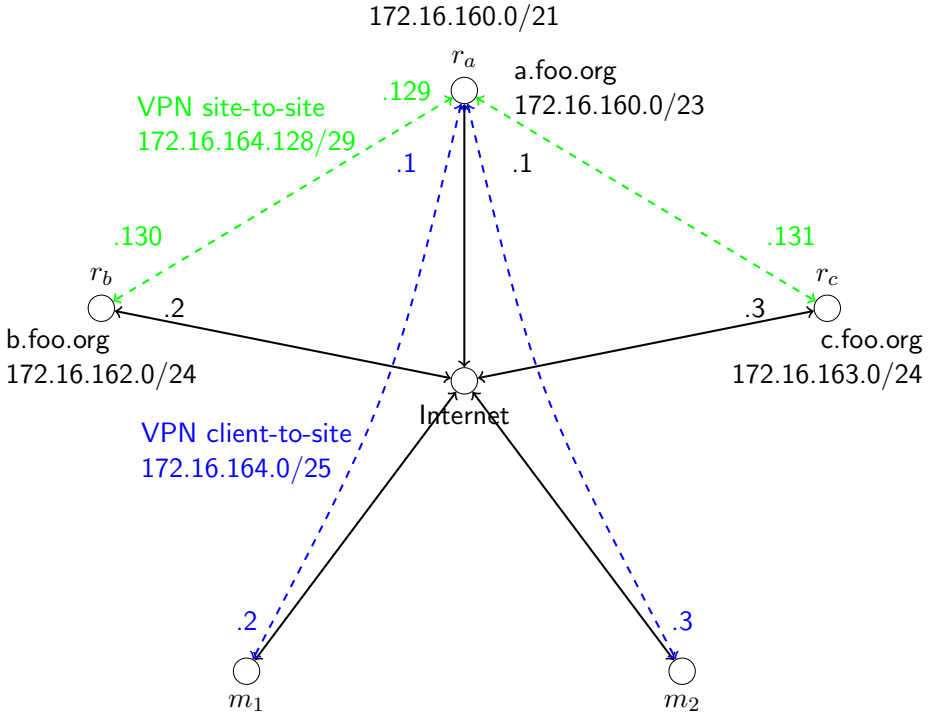


Figure 3.2: Model network

Chapter 4

Encryption

4.1 Motivation

With the rapid expansion of the Internet and computer networks in general, and its use for the exchange of private data, the risk and consequences of unauthorized access are unacceptable. Thus, the use of some form of encryption in the context of unisolated networks and/or sensitive data exchange is de-facto standard in these days. Moreover, most of the devices now have the encryption algorithms hardware implemented, so it does not affect the performance/throughput/power consumption dramatically. There are some scenarios (static websites without private content, for example), when the encryption *may* seem useless, and there is a discussion about that, but the purpose of encrypted HTTP or DNS (DNSSEC) for example, is not just to prove the validity of the data. There is also a *privacy* concern so that the unencrypted traffic, which may represent the contents of a webpage or DNS query, can be easily monitored by anyone who has the access to the network device which the traffic goes through. In 2014, the Internet Engineering Task Force (IETF) published Request for Comments (RFC) 7258 in the category Best Current Practice called "Pervasive Monitoring Is an Attack" stating *"Pervasive monitoring is a technical attack that should be mitigated in the design of IETF protocols, where possible."*[9] That is why modern versions of web browsers show disturbing warnings or pictograms when accessing insecure content. That significantly contributed to the spread of HTTPS over Internet.

4.2 Symmetric & asymmetric cryptography fundamentals

Let's take a brief look at the fundamental principles of symmetric and asymmetric cryptography, which, in cooperation, stands for the security of today's Internet. This will allow us to understand better the following process of creating a public key infrastructure and encryption settings of the services.
In 1978, the authors of the RSA algorithm, which is described below, came up with the names Alice and Bob for the communicating participants.[10] This caught on and, besides with other names, became a convention when discussing cryptography. The following text will meet this convention as well.

4.2.1 Symmetric-key algorithms

Symmetric-key algorithms use the same key to encrypt and decrypt the plaintext / ciphertext meaning all communicating subject has to know the *shared secret*. Including the Caesar Cipher, the symmetric cryptography is the oldest form. Symmetric-key algorithms are divided into

- Stream ciphers that encrypt small portion of data (typically byte) one at time, and
- Block ciphers that take a larger block of bits of constant size (256 bits for example) and encrypt the whole block at time

Most of modern symmetric-key block ciphers are based on Faistel networks.

4.2.2 Public-key algorithms

In 1874, William Stanley Jevons wrote the following sentence in his book *The principles of Science*:
"Can the reader say what two numbers multiplied together will produce the number 8616460799? I think it unlikely that anyone but myself will ever know."[11]
This is the sense of the use of one-way function in cryptography.
Public-key cryptography was invented by Clifford Cocks and James H. Ellis at the UK Government Communications Headquarters (GCHQ) in 1973, however, the research was classified by the British government until 1997.
The public discovery was done in 1976 by Whitfield Diffie and Martin Hellman, who invented Diffie-Hellman (DH) key exchange and, in 1977 by Ronald Rivest, Adi Shamir, and Leonard Adleman, who invented the RSA algorithm.
In contrast to symmetric-key algorithms, the public-key algorithms rely on key pair - the private and the public key being mathematically associated (as further described). The public key is used for encryption and the private key for decryption. Asymmetric algorithms are based on mathematical problems called one-way functions. Here is the list of mathematical problems and cryptosystems that are based on them:

- integer factorization problem - RSA (Rivest-Shamir-Adleman)
- discrete logarithm problem - DH (Diffie-Hellman), DSA (Digital Signature Algorithm)
- elliptic-curve discrete logarithm problem - ECDH & ECDSA

4.2.2.1 RSA

Let's take a look at RSA principle, which is by far the most popular public-key algorithm supporting encryption and digital signature.

4.2.2.1.1 Mathematical problem Assume three very large positive integers e, d, n.

$$(m^e)^d = \left(m^d\right)^e \equiv m \pmod{n}$$

While knowing e and n or even m, it is extremely difficult to find d. $\equiv$ denotes modular congruence relation.

4.2.2.1.2 Keypair generation

1. Pick two random prime numbers p and q,

2. Compute product $n = pq$ and the value of Euler's totient function $\varphi(n) = (p-1)(q-1)$,

3. Choose an integer e so that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$ (e and $\varphi(n)$ are coprime),

4. Determine $d \equiv e^{-1}$ (d being the modular multiplicative inverse of $e \bmod d$).
   This means to solve the congruence $de \equiv 1 \pmod{\varphi(n)}$.

the public key is touple $\{n, e\}$
the private key is touple $\{n, d\}$
Since the numbers p and q are no longer needed, it is better to forget them instead of keeping them secret.

4.2.2.1.3 Key distribution Alice and Bob publish their public key via reliable but not necessarily secret channel (there is no such one yet). When Bob wants to send information to Alice, he uses her public key to encrypt the message, Alice uses her private key to decrypt the message and vice versa.

4.2.2.1.4 Encryption Bob wants to send the plaintext message M to Alice. He converts M to padded message m such that $0 \leq m < n$. Ciphertext c is then computed using the public key e as:

$$c \equiv m^e \pmod{n}$$

... and sent to Alice

4.2.2.1.5 Decryption Alice recovers m from c using her private key exponent d by computing

$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$

... and the original message M is gained by reversing the padding scheme.

4.2.2.2 ECC

ECC stands for Elliptic Curve Cryptography which is based on the algebraic structure of elliptic curves over finite fields.
Elliptic Curve Cryptography allows us to use substantially smaller keys in comparison with RSA, while preserving the same security level. This leads to encryption and decryption speedup.[12]

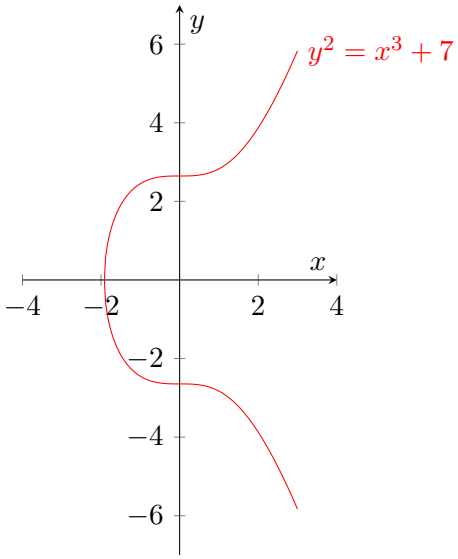


Figure 4.1: Example elliptic curve

We will use ECDSA (Elliptic Curve Digital Signature Algorithm) in our example and will generate keys using *prime256v1* curve.

4.2.3 SSL/TLS

Symmetric-key algorithms perform much better in comparison with public-key algorithms. TLS uses asymmetric encryption to establish a secure connection between the endpoints, to exchange the symmetric key, which is then used for symmetric-key encryption of the traffic exchange.

4.2.4 OpenSSL

OpenSSL is an open-source library/toolkit containing the implementation of almost every cryptographic function and TLS protocol. After the Heartbleed vulnerability was revealed in 2014[13], the OpenBSD project forked OpenSSL, pruned and modernized its codebase radically to create a project named LibreSSL. The same did Google by creating the BoringSSL fork. LibreSSL now provides TLS for multiple operating systems, not only for OpenBSD. Nevertheless, OpenSSL meanwhile resolved many issues.

4.3 Public Key Infrastructure (PKI)

4.3.1 Authorities

Formally, we distinguish *at least* four types of authorities in Public Key Infrastructure (PKI):

- **Registration Authority (RA)** - verifies requests and tells the Certificate authority to issue a certificate
- **Certificate Authority (CA)** - responsible for the issue process itself
- **Validation Authority (VA)** - maintains a list of issued certificates and keeps it publicly accessible (CRL via HTTP, OCSP)
- **Time-Stamp Authority (TSA)** - issue signs for arbitrary data which serve as evidence that a certain data blob existed at a given moment in the past. Time-stamp authorities are used in critical use cases like e-government and e-contracting and are usually not present in organizations, that rather use the services of widely recognized and established qualified certificate authorities.

In small or medium size scenarios, we do not distinguish such authorities and refer to the whole infrastructure as *Certificate authority*. This text will follow the same approach.

4.3.2 PKI hierarchy

While issuing all end certificates with a single Certificate Authority would still provide trust to the network, while it would be cryptographically as strong as a multi-tier hierarchy, this assumption erodes in real-world scenarios that include many people, many organization units, and many devices using the PKI.

Many copies of the primary key would have to be given to many people and that is where all troubles arise.
Such an organization would lose control completely and the risk of PK compromise or malicious use would be very high.
Reliability is one of the greatest requirements that we pose on PKI. It provides us the ability to build large systems and gives them trust which is essential for sensitive data manipulation.
In the case of CA's primary key compromise, the trust becomes broken completely and organization-wide which brings inconceivable troubles.

With regard to these threats, we must pay strong attention to PKI design and create a multi-level hierarchy where Root CA issues certificate for Issuing CA and the Issuing CA issues multipurpose end-entity certificates for people, personal devices and infrastructure devices.

Let's consider the following two-tier PKI model for our example organization:

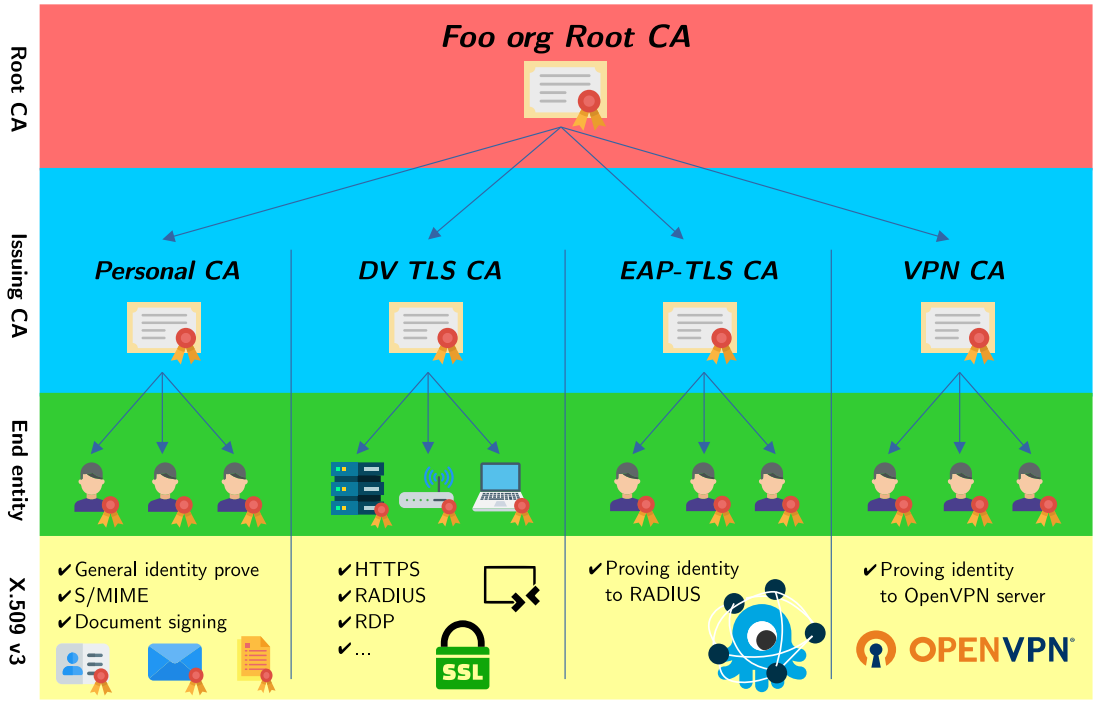


Figure 4.2: Example organizational Public Key Infrastructure hierarchy

In two-tier model, which meets most organization's needs, we recognize three entities:

- Root Certificate Authority
- Issuing Certificate Authority
- End entities - people, personal computing devices and infrastructure devices

In the context of X.509 certificates, the devil is in the detail, in certificate extensions - constraints and attributes.
The most important constraint is CA bit which can be either TRUE or FALSE. A certificate with CA:TRUE can issue (sign) a subordinate certificate and create another tier. On the other side, the end entity's certificate must always have CA set to FALSE to prevent loss of control.

Apart from CA constraint, issuing certificate authorities have *pathlen* constraint set to zero which means it is still able to issue certificates, but only end-entity certificates, not certificate authorities. Individual issuing certificate authority certificate extensions do not differ from each other.

Root CA stands on the top (root) of the hierarchy. Because there is no superior entity, Root CA always have its certificate **self-signed**. Root CA issues intermediate CA certificates (in three-tier model) or issuing CA certificates (in our two-tier model).

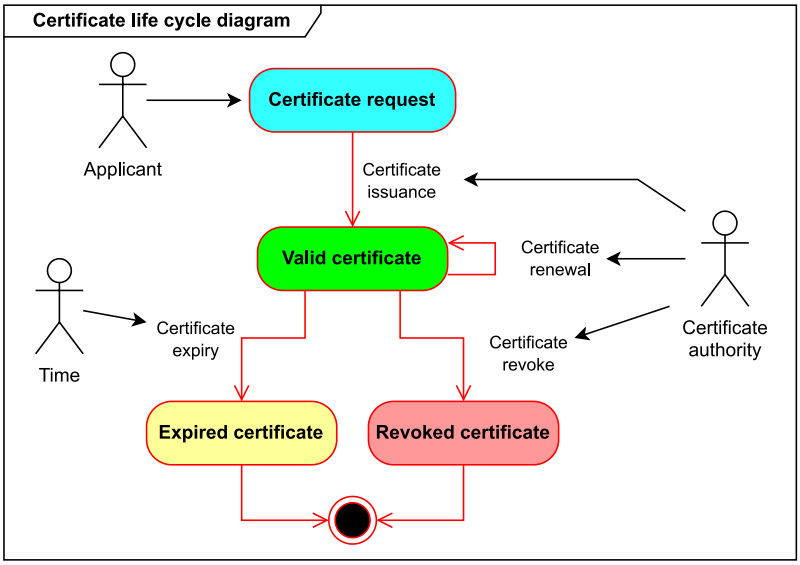


Figure 4.3: Certificate life cycle state UML diagram

4.3.2.1 Personal certificate

A personal certificate is issued to an individual person that can use it to sign documents or emails (S/MIME). The use of this kind of certificate is mostly administrative. Personal certificate links public key with an individual person. The *Common Name* attribute represents a person's full name.

4.3.2.2 Domain Validated certificate

Domain Validated certificate is issued to a device, that holds some critical function or critical data and other entities interacting with it necessarily need to verify the authenticity of the device before starting to exchange critical data with the device.
DV certificate links public key with domain name and/or IP address. Certificate for RADIUS server is a special case. End-device operating systems impose various requirements on RADIUS server certificate. Ignoring these requirements would result in an inability to connect. GÉANT list and discuss these requirements excellently on their website.[14] The *Common Name* attribute represents a domain name or IP address of the device (or both).

4.3.2.3 EAP-TLS client certificate

EAP-TLS certificate is used to authenticate a device that tries to access a network (either Ethernet or Wi-Fi). RADIUS server validates the supplied certificate against EAP-TLS certificate authority and allows or prohibits an authenticator (Wi-Fi router or switch) to connect such devices to the network.
Just like in the case of RADIUS DV certificate, EAP-TLS client certificate should also meet special requirements.

4.3.2.4 VPN certificate

VPN certificate stands for almost the same purpose as EAP-TLS certificate but in the context of a VPN server that connects devices in remote networks (typically in the Internet) to the local network. We divide Personal, EAP-TLS, and VPN certificates, all belonging to persons, due to different X.509 v3 extended key usage values and also for administrative purposes.

4.3.3 Key storage

Cryptographic keypairs can be stored on various storage devices:

1. Personal Computer/server data storage device - HDD, SSD
2. Removable device - USB stick, MMC
3. Hardware Security Module (HSM) - Smart Cards, USB tokens, Cryptocurrency wallets
4. *Unconventional storage media...*

While storing cryptographic keypairs directly in the filesystem is very convenient, straightforward, and easy to use, this approach brings many threats.
That way, keys are stored in regular files (with .pem and .key extensions) and it is completely up to the user to ensure that unauthorized entities do not have access to these files. Securing files in the local filesystem is done namely via ownership and permissions.

PKCS#11 standardizes the interaction with HSM and enables us to generate keypair directly in HSM and use it to sign arbitrary data using the supported mechanisms so that the private key never escapes the HSM itself. By using these devices, we kind of transfer the trust to manufacturers of these devices and authorities that certified the devices.

These removable HSMs may be kept in a secure place for most of the time and the PIN code may be split among multiple people. In general, this approach makes the manipulation with critical keys much more transparent and traceable.

4.3.3.1 HSM SafeNet eToken 5110 CC

Let's inspect an example USB Hardware Security Module, sometimes referenced as *Smart Card* or *Token*. The Author of this text owns SafeNet eToken 5110 CC which is certified as a Qualified Signature Creation Device (QSCD) according to eIDAS (EU regulation). That means this particular HSM enables its owner to create Qualified Electronic Signatures and Qualified seals recognized within the internal European government and market.
In 2023, this device is available for €43,56 incl. VAT.[15]
The manufacturer indicates support for Linux, MacOS, and Windows. In the context of the UNIX OS family, the OpenSC project (SC stands for Smart Card) provides drivers, tools, and middleware for such HSMs so they can be used in combination with OpenSSL. Here, the Debian Linux distribution will be used to demonstrate the use SafeNet eToken 5110 CC. Although OpenBSD has OpenSC in ports as well, the manufacturer does not provide `libeToken.so` library for OpenBSD that is necessary for pkcs11-tool to work with eToken 5110 CC. The author did not test `libeToken.so` available for MacOS that also belongs to the UNIX family.

Figure 4.4: USB Hardware Security Module - Gemalto SafeNet eToken 5110 CC

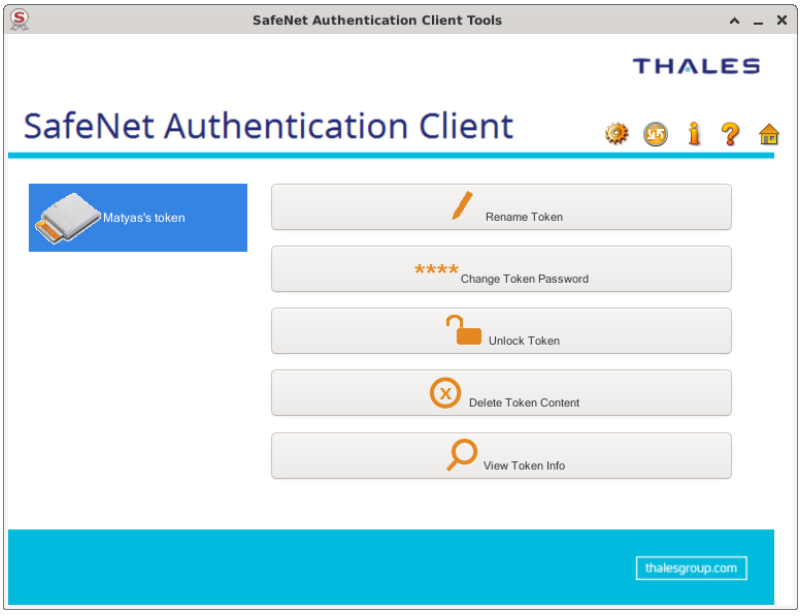


Figure 4.5: SafeNet Authentication Client - home page

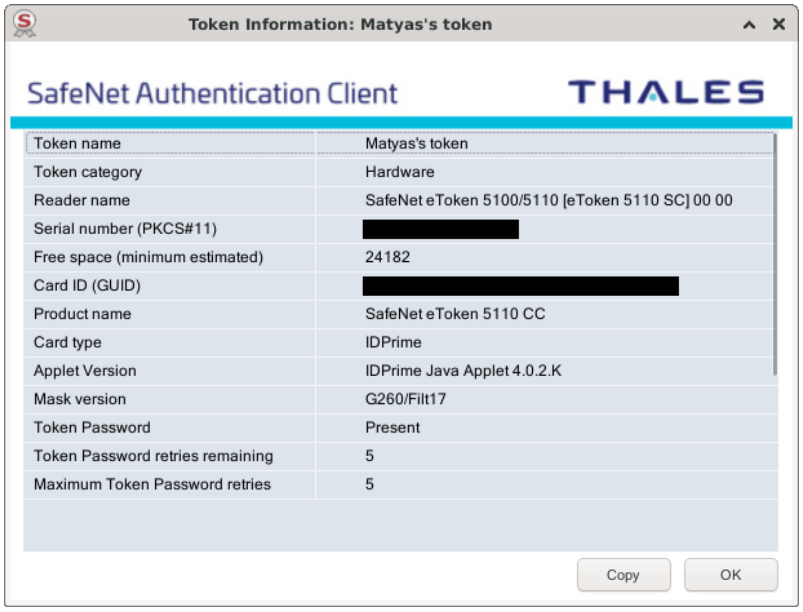


Figure 4.6: SafeNet Authentication Client - token info

Let's inspect how can we use pkcs11-tool to interact with HSM:

```
# pkcs11-tool (part of OpenSC)
pkcs11-tool --module libeToken.so --login --list-objects
pkcs11-tool --module libeToken.so --login --show-info

# Generate keypair
pkcs11-tool --module libeToken.so --login --keypairgen --id 1 --label <
    KEYPAIR_LABEL> --key-type <KEY_TYPE>

# Possible key types:
EC:prime256v1
EC:secp384r1
rsa:2048
rsa:4096

# Example output of --keypairgen:
ca@linux:~/pki/root$ pkcs11-tool --module libeToken.so --login --keypairgen
     --id 1 --label foo-root-ecc-g1 --key-type EC:prime256v1
Using slot 0 with a present token (0x0)
Logging in to "Matyas's token".
Please enter User PIN:
Key pair generated:
Private Key Object; EC
  label:      foo-root-ecc-g1
  ID:         01
  Usage:      decrypt, sign, derive
  Access:     sensitive, always sensitive, never extractable, local
Public Key Object; EC  EC_POINT 256 bits
  EC_POINT:   044104
    cdaf5ad315b7bc035bdaeb5f88c1689e426e3c66e8bc96c310f556512199 (...)
  EC_PARAMS:  06082a8648ce3d030107
  label:      foo-root-ecc-g1
  ID:         01
  Usage:      encrypt, verify, wrap, derive
  Access:     local

# Remove private and public key
pkcs11-tool --module libeToken.so --login --delete-object --id <KEYPAIR_ID>
     --type=privkey
pkcs11-tool --module libeToken.so --login --delete-object --id <KEYPAIR_ID>
     --type=pubkey
```

4.3.4 Building PKI infrastructure

In this section, we will demonstrate how to generate a whole PKI using OpenSSL. Readers searching for more detailed explanation how to use OpenSSL to create PKI can visit OpenSSL Certificate Authority from Jamie Nguyen.[1][16]

4.3.4.1 Creating a system user

At first, we create a new system user dedicated for PKI manipulation called `ca`.

```
useradd -m -c "Certificate Authority" ca
passwd ca

# Enter some strong password

# Exit and login as ‘ca‘ user
exit

```

[1] https://jamielinux.com/docs/openssl-certificate-authority/

```
mkdir /home/ca/pki
cd /home/ca/pki
vim openssl.cnf.templ
# Insert contents from openssl.cnf.templ

# Set distinguished name defaults req_distinguished_name section
sed -i 's#<DN_COUNTRY>#CZ#g' openssl.cnf.templ
sed -i 's#<DN_STATE>#Czechia#g' openssl.cnf.templ
sed -i 's#<DN_LOCALITY>#Prague#g' openssl.cnf.templ
sed -i 's#<DN_ORG>#Foo org#g' openssl.cnf.templ

# Set OCSP and CRL URL addresses
sed -i 's#<CRL_ROOT>#http://ca.foo.org/root/ca.crl#g' openssl.cnf.templ
sed -i 's#<OCSP_ROOT>#http://ca.foo.org/ocsp/root#g' openssl.cnf.templ
sed -i 's#<CRL_DVTLS>#http://ca.foo.org/dvtls/ca.crl#g' openssl.cnf.templ
sed -i 's#<OCSP_DVTLS>#http://ca.foo.org/ocsp/dvtls#g' openssl.cnf.templ
sed -i 's#<CRL_EAPTLS>#http://ca.foo.org/eaptls/ca.crl#g' openssl.cnf.templ
sed -i 's#<OCSP_EAPTLS>#http://ca.foo.org/ocsp/eaptls#g' openssl.cnf.templ
sed -i 's#<CRL_PERSONAL>#http://ca.foo.org/personal/ca.crl#g' openssl.cnf.
    templ
sed -i 's#<OCSP_PERSONAL>#http://ca.foo.org/ocsp/personal#g' openssl.cnf.
    templ

# Set RADIUS server DNS name
sed -i 's#<RADIUS_DNS>#radius.foo.org#g' openssl.cnf.templ

# Set path to HSM OpenSC library in pkcs11_section.MODULE_PATH
sed -i 's#<OpenSC_LIB>#/usr/lib/libeToken.so#g' openssl.cnf.templ
```

4.3.4.2 Root Certificate Authority

Let's now move on by creating the Root Certificate Authority.

```
mkdir root
cd root

# Create directories and files for OpenSSL ca
mkdir certs crl csr newcerts private
chmod 700 private
touch index.txt
echo 01 > serial
echo 01 > crlnumber
cp ../openssl.cnf.templ ./openssl.cnf
sed -i 's#<CA_NAME>#Root CA#g' openssl.cnf
sed -i 's#<CA_DIR>#root#g' openssl.cnf
sed -i 's#<DEFAULT_DAYS>#3650#g' openssl.cnf
sed -i 's#<POLICY>#policy_strict#g' openssl.cnf

# Generate key in HSM
pkcs11-tool --module libeToken.so \
  --login --keypairgen \
  --id 1 --label foo-root-ecc-g1 \
  --key-type EC:prime256v1

# In computer
openssl ecparam -genkey -name prime256v1 | openssl ec -aes256 -out private/
    ca.key.pem
chmod 400 private/ca.key.pem

# Create certificate on HSM
# -key accepts RFC7512 URI scheme
openssl req -config ./openssl.cnf \
      -new -x509 -days 7300 -sha256 -extensions v3_ca \
      -engine pkcs11 -keyform engine \
      -key 0:01 \
      -out certs/ca.cert.pem

# Create certificate in computer
openssl req -config ./openssl.cnf \
      -key private/ca.key.pem \
      -new -x509 -days 7300 -sha256 -extensions v3_ca \
      -out certs/ca.cert.pem

chmod 444 certs/ca.cert.pem

# Verify the root certificate
openssl x509 -noout -text -nameopt utf8 -in certs/ca.cert.pem

# Generate CRL (by following the procedure in crl.txt) ...
# Issue OCSP certificate (by following the procedure in ocsp.txt)...
```

4.3.5 DV TLS Certificate Authority

```
cd /home/ca/pki
mkdir dvtls
cd dvtls
mkdir certs crl csr newcerts private
chmod 700 private
touch index.txt
echo 01 > serial
echo 01 > crlnumber
cp ../openssl.cnf.templ ./openssl.cnf
sed -i 's#<CA_NAME>#DV TLS CA#g' openssl.cnf
sed -i 's#<CA_DIR>#dvtls#g' openssl.cnf
sed -i 's#<DEFAULT_DAYS>#730#g' openssl.cnf
sed -i 's#<POLICY>#policy_loose#g' openssl.cnf

# Generate key in HSM
pkcs11-tool --module libeToken.so \
  --login --keypairgen \
  --id 2 --label foo-dvtls-ecc-g1 \
  --key-type EC:prime256v1

# Generate key in computer
openssl ecparam -genkey -name prime256v1 | openssl ec -aes256 -out private/
    ca.key.pem
chmod 400 private/ca.key.pem

# Generate request using keypair on HSM
openssl req -config ./openssl.cnf -new -sha256 \
      -engine pkcs11 -keyform engine \
      -key 0:02 \
      -out csr/ca.csr.pem

# Generate request using keypair in computer
openssl req -config ./openssl.cnf -new -sha256 \
      -key private/ca.key.pem \
      -out csr/ca.csr.pem

# Sign dvtls certificate by root CA
cd ../root

# Sign the request using Root CA on HSM
openssl ca -config ./openssl.cnf \
      -extensions v3_issuing_ca -days 3650 -notext -md sha256 \
      -engine pkcs11 -keyform engine \
      -keyfile 0:01 \
      -in ../dvtls/csr/ca.csr.pem \
      -out ../dvtls/certs/ca.cert.pem

# Sign the request using Root CA in computer
openssl ca -config ./openssl.cnf \
      -extensions v3_issuing_ca -days 3650 -notext -md sha256 \
      -in ../dvtls/csr/ca.csr.pem \
      -out ../dvtls/certs/ca.cert.pem

chmod 444 ../dvtls/certs/ca.cert.pem

# Verify the certificate
openssl x509 -noout -text -nameopt utf8 -in ../dvtls/certs/ca.cert.pem
# Verify dvtls cert against root cert
openssl verify -CAfile certs/ca.cert.pem ../dvtls/certs/ca.cert.pem

# Create fullchain certificate
```

```
61 cat /home/ca/pki/dvtls/certs/ca.cert.pem /home/ca/pki/root/certs/ca.cert.
       pem > /home/ca/pki/dvtls/certs/ca.fchain-cert.pem
62 chmod 444 /home/ca/pki/dvtls/certs/ca.fchain-cert.pem
63
64 cd /home/ca/pki/dvtls
65 # Generate CRL (by following the procedure in crl.txt) ...
66 # Issue OCSP certificate (by following the procedure in ocsp.txt)...
```

4.3.5.1 Issuing Domain Validated certificate

Let's now use our new DV TLS certificate authority to issue a certificate for mikrotik router with 2-year validity.

```
cd /home/ca/pki/dvtls
openssl ecparam -genkey -name prime256v1 | openssl ec -aes256 -out private/mikrotik.a.foo.org.key.pem
chmod 400 private/mikrotik.a.foo.org.key.pem

vim openssl.cnf
# [ v3_server_alt_names ]
# DNS.1                   = mikrotik.a.foo.org
# #DNS.2                  = localhost
# IP.1                   = 172.16.160.2
# #IP.2                   = 192.168.1.100

openssl req -config ./openssl.cnf -new -sha256 \
  -key private/mikrotik.a.foo.org.key.pem \
  -out csr/mikrotik.a.foo.org.csr.pem

# Set CN to mikrotik.a.foo.org, leave email blank

# Transfer mikrotik.a.foo.org.csr.pem to signing machine (/home/ca/pki/dvtls/csr/)
cd /home/ca/pki/dvtls

# Sign the request using DV TLS CA keypair on HSM
openssl ca -config ./openssl.cnf \
      -extensions v3_server_cert -days 730 -notext -md sha256 \
      -engine pkcs11 -keyform engine \
      -keyfile 0:02 \
      -in csr/mikrotik.a.foo.org.csr.pem \
      -out certs/mikrotik.a.foo.org.cert.pem

# Sign the request using DV TLS CA keypair in computer
openssl ca -config ./openssl.cnf \
  -extensions v3_server_cert -days 730 -notext -md sha256 \
  -in csr/mikrotik.a.foo.org.csr.pem \
  -out certs/mikrotik.a.foo.org.cert.pem

chmod 444 certs/mikrotik.a.foo.org.cert.pem

# Verify the certificate
openssl x509 -noout -text -nameopt utf8 -in certs/mikrotik.a.foo.org.cert.pem
openssl verify -CAfile certs/ca.fchain-cert.pem certs/mikrotik.a.foo.org.cert.pem

# Create fullchain certificate
cat /home/ca/pki/dvtls/certs/mikrotik.a.foo.org.cert.pem /home/ca/pki/dvtls/certs/ca.fchain-cert.pem > /home/ca/pki/dvtls/certs/mikrotik.a.foo.org.fchain-cert.pem
chmod 444 /home/ca/pki/dvtls/certs/mikrotik.a.foo.org.fchain-cert.pem

# Create PKCS#12 bundle
openssl pkcs12 -export -out private/mikrotik.a.foo.org.p12 -inkey private/mikrotik.a.foo.org.key.pem -in certs/mikrotik.a.foo.org.fchain-cert.pem
```

4.3.6 EAP-TLS Certificate Authority

```
cd /home/ca/pki
mkdir eaptls
cd eaptls
mkdir certs crl csr newcerts private
chmod 700 private
touch index.txt
echo 01 > serial
echo 01 > crlnumber
cp ../openssl.cnf.templ ./openssl.cnf
sed -i 's#<CA_NAME>#EAP-TLS CA#g' openssl.cnf
sed -i 's#<CA_DIR>#eaptls#g' openssl.cnf
sed -i 's#<DEFAULT_DAYS>#365#g' openssl.cnf
sed -i 's#<POLICY>#policy_loose#g' openssl.cnf

# To generate key in HSM, use --id 3 --label foo-eaptls-ecc-g1

# And proceed the same way as for DV TLS CA
```

4.3.6.1 Issuing EAP-TLS certificate

```
# On client device, generate the key and request using openssl.cnf from /
    home/ca/pki/eaptls/openssl.cnf
cd /home/ca/pki/eaptls
openssl ecparam -genkey -name prime256v1 | openssl ec -aes256 -out private/
    matyas@foo.org.key.pem
chmod 400 private/matyas@foo.org.key.pem

vim openssl.cnf
# [ v3_client_alt_names ]
# email.1 = matyas@foo.org
# #email.2 = secondary@foo.org

openssl req -new -config ./openssl.cnf \
  -key private/matyas@foo.org.key.pem \
  -out csr/matyas@foo.org.csr.pem

# Transfer matyas@foo.org.csr.pem to signing machine (/home/ca/pki/eaptls/
    csr/)
cd /home/ca/pki/eaptls

# Sign the request using DV TLS CA keypair on HSM
openssl ca -config ./openssl.cnf \
      -extensions v3_eaptls_client -days 375 -notext -md sha256 \
      -engine pkcs11 -keyform engine \
      -keyfile 0:03 \
      -in csr/matyas@foo.org.csr.pem \
      -out certs/matyas@foo.org.cert.pem

# Sign the request using EAP-TLS CA keypair in computer
openssl ca -config ./openssl.cnf \
      -extensions v3_eaptls_client -days 375 -notext -md sha256 \
      -in csr/matyas@foo.org.csr.pem \
      -out certs/matyas@foo.org.cert.pem

chmod 444 certs/matyas@foo.org.cert.pem

# Verify the certificate
openssl x509 -noout -text -nameopt utf8 -in certs/matyas@foo.org.cert.pem
openssl verify -CAfile certs/ca.fchain-cert.pem certs/matyas@foo.org.cert.
    pem
```

```
37
38 # Create fullchain certificate
39 cat /home/ca/pki/eaptls/certs/matyas@foo.org.cert.pem /home/ca/pki/eaptls/
       certs/ca.fchain-cert.pem > /home/ca/pki/eaptls/certs/matyas@foo.org.
       fchain-cert.pem
40 chmod 444 /home/ca/pki/eaptls/certs/matyas@foo.org.fchain-cert.pem
41
42 # Create PKCS#12 bundle
43 openssl pkcs12 -export -out private/matyas@foo.org.p12 -inkey private/
       matyas@foo.org.key.pem -in certs/matyas@foo.org.fchain-cert.pem
```

4.3.7 Personal Certificate Authority

```
cd /home/ca/pki
mkdir personal
cd personal
mkdir certs crl csr newcerts private
chmod 700 private
touch index.txt
echo 01 > serial
echo 01 > crlnumber
cp ../openssl.cnf.templ ./openssl.cnf
sed -i 's#<CA_NAME>#Personal CA#g' openssl.cnf
sed -i 's#<CA_DIR>#personal#g' openssl.cnf
sed -i 's#<DEFAULT_DAYS>#365#g' openssl.cnf
sed -i 's#<POLICY>#policy_loose#g' openssl.cnf

# To generate key in HSM, use --id 4 --label foo-personal-ecc-g1

# And proceed the same way as for DV TLS CA
```

4.3.7.1 Issuing personal certificate

```
# On client device, generate the key and request using openssl.cnf from /
    home/ca/pki/personal/openssl.cnf
cd /home/ca/pki/personal
openssl ecparam -genkey -name prime256v1 | openssl ec -aes256 -out private/
    matyas@foo.org.key.pem
chmod 400 private/matyas@foo.org.key.pem

vim openssl.cnf
# [ v3_client_alt_names ]
# email.1 = matyas@foo.org
# #email.2 = secondary@foo.org

openssl req -new -config ./openssl.cnf \
  -key private/matyas@foo.org.key.pem \
  -out csr/matyas@foo.org.csr.pem

# Transfer matyas@foo.org.csr.pem to signing machine (/home/ca/pki/personal
    /csr/)
cd /home/ca/pki/personal

# Sign the request using DV TLS CA keypair on HSM
openssl ca -config ./openssl.cnf \
      -extensions v3_client_cert -days 375 -notext -md sha256 \
      -engine pkcs11 -keyform engine \
      -keyfile 0:04 \
      -in csr/matyas@foo.org.csr.pem \
      -out certs/matyas@foo.org.cert.pem

# Sign the request using EAP-TLS CA keypair in computer
openssl ca -config ./openssl.cnf \
      -extensions v3_client_cert -days 375 -notext -md sha256 \
      -in csr/matyas@foo.org.csr.pem \
      -out certs/matyas@foo.org.cert.pem

chmod 444 certs/matyas@foo.org.cert.pem

# Verify the certificate
openssl x509 -noout -text -nameopt utf8 -in certs/matyas@foo.org.cert.pem
openssl verify -CAfile certs/ca.fchain-cert.pem certs/matyas@foo.org.cert.
    pem
```

```
37
38 # Create fullchain certificate
39 cat /home/ca/pki/personal/certs/matyas@foo.org.cert.pem /home/ca/pki/
       personal/certs/ca.fchain-cert.pem > /home/ca/pki/personal/certs/
       matyas@foo.org.fchain-cert.pem
40 chmod 444 /home/ca/pki/personal/certs/matyas@foo.org.fchain-cert.pem
41
42 # Create PKCS#12 bundle
43 openssl pkcs12 -export -out private/matyas@foo.org.p12 -inkey private/
       matyas@foo.org.key.pem -in certs/matyas@foo.org.fchain-cert.pem
```

4.3.7.2 Trust

After creating the PKI infrastructure, we need to distribute certificate authority certificates among the network devices.
Every particular device should trust a minimal set of certificate authorities.
For example, the RADIUS server has to trust Root CA and EAP-TLS CA because clients are proving their identity using certificates issued by EAP-TLS certificate authority. In the contrary, there is no need for a client device to trust to EAP-TLS certificate authority because no one else proves its identity to the client device using EAP-TLS certificate.

As expected, the process of installing certificate authority into the operating system is specific for each one. This text is not aimed to describe how to install certificate authority into every particular operating system. Readers are kindly recommended to follow the instructions for installing CA certificate for their operating system. However, the installation process on Android and Windows is described in Wi-Fi security chapter.

In addition, it is common that some applications use their own list of trusted certificate authorities independent on the operating system list. Users of these applications need to import certificates into such applications as well.

4.3.8 Alternative ways of PKI management

When we take a look at OpenVPN GitHub, we find that it includes *easy-rsa* repository that contains a shell script-based wrapper of OpenSSL that makes PKI maintenance much more convenient and can be used for multi-tier PKI as well.[17]

FreeRADIUS project developers did something similar with the help of Makefile[18], so FreeRADIUS users are able to build a single-tier PKI with certificates, that are claimed to be compatible with most of operating systems.[19] According to commit 6678b3a[2], the developers already work on ECC PKI generation.

This text presents manual PKI management using OpenSSL library directly for demonstrative purposes. For easy use in production, the above OpenSSL calls should be wrapped by some Makefile or shell script.

[2] https://github.com/FreeRADIUS/freeradius-server/commit/6678b3a6f29d98e6bd93f783349832e325e1151f

Chapter 5

Configuration and testing

OpenBSD project claims *"Only two remote holes **in the default install**, in a heck of a long time!"*. It is important to keep in mind the fewer changes we make, the better. The fewer diffs we make in the default configuration files, the easier a future system upgrade will be, which prompts us to merge manually old configuration files with the new ones.[20] System upgrade is described in *System upgrade* section.

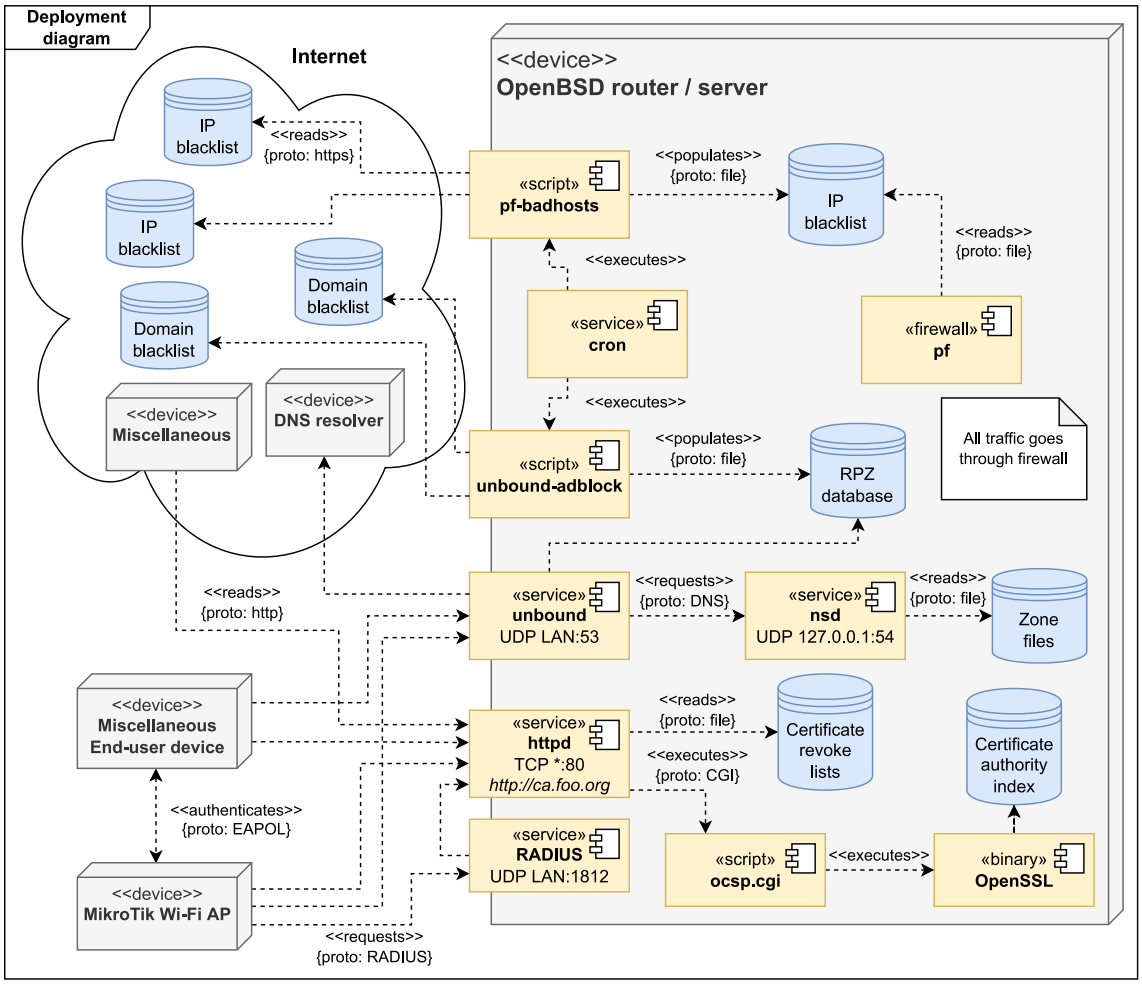


Figure 5.1: UML Deployment diagram

5.0.1 Miscellaneous initial system config

```
# Set system hostname
echo srv.a.foo.org > /etc/myname
```

5.0.2 Network configuration

Assume we have 2 network interfaces in our system:

- `re0` - WAN ethernet interface connected to ISP
- `re1` - LAN ethernet interface connected to switch

For each network interface, there should be a config file `/etc/hostname.if`, where `if` represents the interface name, `rge0` for instance.

```
# Configure DHCP on WAN interface
echo 'dhcp' > /etc/hostname.re0

# Or set static IP and mask gained from ISP
echo 'inet 10.108.271.35 255.255.255.0 description "WAN uplink"' > /etc/
    hostname.re0

# Set ISP's default gateway
echo 10.108.271.1 > /etc/mygate

# Configure the ethernet LAN interface
echo 'inet 172.16.162.1 255.255.255.0 description "LAN"' > /etc/hostname.
    re1

# Apply the changes
sh /etc/netstart
```

We have to enable IP and IPv6 forwarding

```
echo 'net.inet.ip.forwarding=1' >> /etc/sysctl.conf
```

5.0.3 DHCP

OpenBSD comes with DHCP daemon called dhcpd.

```
# Specify interfaces on which dhcpd should listen for broadcasts
rcctl set dhcpd flags re1
```

Configure DHCP daemon

```
subnet 172.16.160.0 netmask 255.255.254.0 {
  option routers 172.16.160.1;
  option domain-name-servers 172.16.160.1;
  range 172.16.160.5 172.16.161.254;

  host mikrotik {
    fixed-address 172.16.160.2;
    hardware ethernet 00:00:00:00:00:00;
  }

  host printer {
    fixed-address 172.16.160.5;
    hardware ethernet 00:00:00:00:00:00;
  }

  host pc {
    fixed-address 172.16.160.6;
    hardware ethernet 00:00:00:00:00:00;
  }
}
```

/etc/dhcpd.conf

```
# Check configuration
dhcpd -n

# Run in debug mode
dhcpd -d

# If everything seems to work correctly, start and enable the daemon
rcctl start dhcpd
rcctl enable dhcpd
```

5.0.4 Firewall

OpenBSD uses the PF (Packet Filter) which configuration file is located at `/etc/pf.conf` and `pfctl` command is used to control the firewall.

5.0.4.1 Changing configuration safely

Commonly, we are somewhere in the Internet (not present to the device) when we need to make some changes in firewall settings. In such a case, there is a high risk that we may accidentally saw off the branch on which we are sitting (kill the VPN connection). Even though we can check the new configuration for syntax mistakes, we cannot be 100% sure the new configuration will work as desired. The consequences would be catastrophic. The following procedure prevents such a situation.

1. Create a copy of the old config
   `cp /etc/pf.conf /etc/pf.conf.new`
2. Make changes in the new config
   `vim /etc/pf.conf.new`
3. Check the new config for syntax mistakes. If there are any, go to step 2.
   `pfctl -nf /etc/pf.conf.new`
4. Load the new config, wait a period and then load back the old configuration.
   `pfctl -f /etc/pf.conf.new && sleep 60 && pfctl -f /etc/pf.conf`
   The period should be long enough to allow us to check at least critical services like SSH. If we have made some mistakes, we have the guarantee that after a while, our old configuration will take effect again. If so, go to step 2.
5. If the new config works, we can overwrite the old one with the new one
   `mv /etc/pf.conf.new /etc/pf.conf`
6. ...and finally load the new configuration
   `pfctl -f /etc/pf.conf`

5.0.4.2 Router firewall configuration

The following pf configuration for router is based on the configuration posted on OpenBSD's FAQ[21].

```
lan="rge1"
adblock="172.16.160.0"
adblock6="172.16.160.0"

# Non-routable addresses
table <martians> { 0.0.0.0/8 10.0.0.0/8 127.0.0.0/8 169.254.0.0/16     \
       172.16.0.0/12 192.0.0.0/24 192.0.2.0/24 224.0.0.0/3 \
       192.168.0.0/16 198.18.0.0/15 198.51.100.0/24        \
       203.0.113.0/24 }

# Devices that are blocked from Internet access (a printer for example)
table <block_internet> { 172.16.160.5 }

# Silently drop rejected packets, do not send TCP RST
set block-policy drop

# Enable statistics collection on egress interface group
set loginterface egress

# Ignore traffic on loopback interface
set skip on lo0

# Normalize incoming packets
match in all scrub (no-df random-id max-mss 1440)

# Perform NAT between the LAN and WAN
match out on egress inet from !(egress:network) to any nat-to (egress:0)

# Antispoof
antispoof quick for { egress $lan }
block in quick on egress from <martians> to any
block return out quick on egress from any to <martians>

# Default deny.
block all

# pf-badhosts
table <pfbadhost> persist file "/etc/pf-badhost.txt"
block in quick on egress from <pfbadhost>
block out quick on egress to <pfbadhost>

# unbound-adblock
table <gooDNS4> {8.8.8.8 8.8.4.4}
table <gooDNS6> {2001:4860:4860::8888 2001:4860:4860::8844}
pass in quick to <gooDNS4> rdr-to $adblock
pass in quick to <gooDNS6> rdr-to $adblock6

# Wireguard VPN
pass in on { wg0 wg1 }
# Allow wg connection to UDP
pass in on egress proto udp to port { <PORT_SITE_TO_SITE> <
    PORT_CLIENT_TO_SITE> }
# How to NAT a traffic from wg interfaces:
#pass out on egress inet from (wg0:network) nat-to (egress:0)
#pass out on egress inet from (wg1:network) nat-to (egress:0)

# HTTP OCSP, CRL
pass in proto tcp to port http

```

```
59 # Permit RADIUS
60 pass in proto udp to port radius
61
62 # block selected devices from Internet access
63 block on egress from <block_internet> to any
64
65 # Allow outgoing IPv4 traffic
66 pass out quick inet keep state
67 pass in on { $lan } inet
68
69 # Example port forwarding of HTTP & HTTPS to 172.16.162.3 (pc.a.foo.org)
70 # pass in on egress inet proto tcp from any to (egress) port { http https }
       rdr-to 172.16.162.3
```

/etc/pf.conf

5.0.4.3 Dynamic firewall - pf-badhost

Nowadays, each device with a public IP address (not being behind NAT) exposed to the Internet and providing any kind of service faces literally a constant attack attempts. Attackers, most of the time represented by botnet machines performing mass attacks, try hard to exploit the particular software and break in our system in any way. Some attempts target poorly secured servers and try to guess SSH credentials, the others are more sophisticated and hidden.

The following extract from SSH daemon logfile shows a real example of 5 IP addresses trying to guess SSH credentials at the same time after disabling the dynamic firewall for several days:

```
1  sshd: Address 222.255.115.237 maps to static.vnpt.vn, but this does not map back to the address
       - POSSIBLE BREAK-IN ATTEMPT!
2  sshd: Invalid user ubuntu from 222.255.115.237 port 45060
3  sshd: input_userauth_request: invalid user ubuntu [preauth]
4  sshd: error: maximum authentication attempts exceeded for invalid user ubuntu from
       222.255.115.237 port 45060 ssh2 [preauth]
5  sshd: Disconnecting: Too many authentication failures [preauth]
6  sshd: Invalid user gordor from 46.101.224.184 port 48658
7  sshd: input_userauth_request: invalid user gordor [preauth]
8  sshd: error: maximum authentication attempts exceeded for invalid user gordor from
       46.101.224.184 port 48658 ssh2 [preauth]
9  sshd: Disconnecting: Too many authentication failures [preauth]
10 sshd: reverse mapping checking getaddrinfo for abts-mp-dynamic-035.241.70.182.airtelbroadband.in
       [182.70.241.35] failed - POSSIBLE BREAK-IN ATTEMPT!
11 sshd: Invalid user jordan from 182.70.241.35 port 47818
12 sshd: input_userauth_request: invalid user jordan [preauth]
13 sshd: error: maximum authentication attempts exceeded for invalid user jordan from 182.70.241.35
       port 47818 ssh2 [preauth]
```

Although there are many security measures, let's take a look at simple but effective one - a dynamic firewall.

Dynamic firewall (DynFW) is an attack prevention system based on the data provided by the threat detection system and/or public abusive IP lists. Dynamic firewall can just periodically download abusive IP addresses from public lists (acting passively) or it may include a threat detection system that inspects log files of services periodically and proactively add entries to abusive lists. While requiring critical access, such system should be restricted on a system level to very minimal set of operations needed. Luckily, the developers of this software list these requirements clearly in the documentation so there is no need to tune the permissions by reverse engineering.

Although writing a shell script that would accomplish that job would not be extremely difficult, we would rather use well-reputed script called pf-badhosts by Jordan Geoghegan.[22]
pf-badhost is a single file shell script that downloads public IP blocklists and populates local file that serves as an IP database for the system's pf firewall. It is a very simple and neat solution compatible with the UNIX OS family (including MacOS). Public blocklist URLs are shipped within the script and can be easily maintained especially when we export the list to an external file.

```
_BLOCKLISTS=$(cat <<'__EOT'
### Local File Example
# file:/path/to/local/file

### Download popular IPv4 blocklists
https://www.binarydefense.com/banlist.txt
https://rules.emergingthreats.net/blockrules/compromised-ips.txt
https://rules.emergingthreats.net/fwrules/emerging-Block-IPs.txt
https://raw.githubusercontent.com/firehol/blocklist-ipsets/master/
    firehol_level1.netset
https://raw.githubusercontent.com/firehol/blocklist-ipsets/master/
    firehol_level2.netset
### Firehol level 3 can be a little aggressive.
### Ill leave it up to users to choose to enable.
# https://raw.githubusercontent.com/firehol/blocklist-ipsets/master/
    firehol_level3.netset

### Spamhause DROP lists (Dont Route Or Peer)
https://www.spamhaus.org/drop/drop.txt
https://www.spamhaus.org/drop/edrop.txt
https://www.spamhaus.org/drop/dropv6.txt

### Block Shodan
https://isc.sans.edu/api/threatlist/shodan/?text

### Block botnets + command and control servers
https://feodotracker.abuse.ch/downloads/ipblocklist.txt
https://sslbl.abuse.ch/blacklist/sslipblacklist.txt

### Optional lists -- uncomment to enable
(...)
```

extract from pf-badhost.sh

Let's install the pf-badhost finally.

```
useradd -s /sbin/nologin -d /var/empty _pfbadhost

cd /tmp
ftp https://geoghegan.ca/pub/pf-badhost/latest/pf-badhost.sh
install -m 755 -o root -g bin pf-badhost.sh /usr/local/bin/pf-badhost

install -m 640 -o _pfbadhost -g wheel /dev/null /etc/pf-badhost.txt
install -d -m 755 -o root -g wheel /var/log/pf-badhost
install -m 640 -o _pfbadhost -g wheel /dev/null /var/log/pf-badhost/pf-
    badhost.log
install -m 640 -o _pfbadhost -g wheel /dev/null /var/log/pf-badhost/pf-
    badhost.log.0.gz

$ cat /etc/doas.conf
...
permit root
permit nopass _pfbadhost cmd /sbin/pfctl args -nf /etc/pf.conf
permit nopass _pfbadhost cmd /sbin/pfctl args -t pfbadhost -T replace -f /
    etc/pf-badhost.txt
# Optional rule for authlog scanning
permit nopass _pfbadhost cmd /usr/bin/zcat args -f /var/log/authlog /var/
    log/authlog.0.gz
...

# Run the script as _pfbadhost user with openbsd argument
doas -u _pfbadhost pf-badhost -O openbsd

# Reload pf rule set
pfctl -f /etc/pf.conf

# Edit _pfbadhost user crontab and run pf-badhost every day at random time
    between 0:00 and 1:00
crontab -u _pfbadhost -e
~ 0~1 * * * -s pf-badhost -O openbsd
```

5.0.5 OCSP responder and CRL server

When we created our certificates, we embedded an URL of the Client Revoke list and OCSP in them.
We will use OpenBDS's simple and neat httpd server as CRL publisher and OCSP validator.

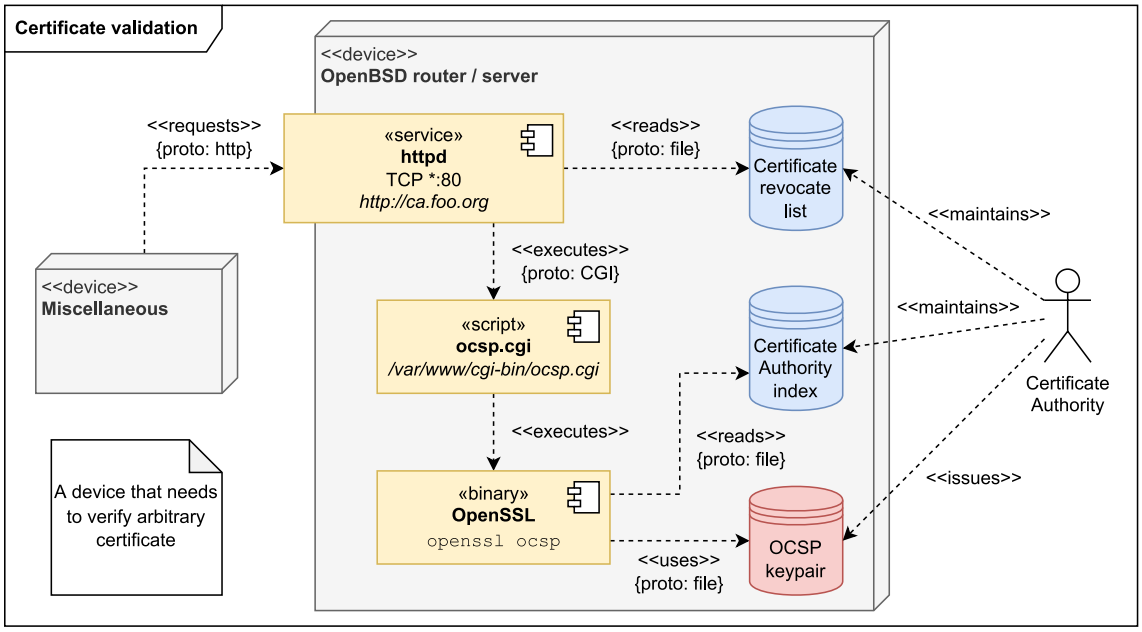


Figure 5.2: OCSP responder and CRL server UML deployment diagram

Certificate revoke lists will be served natively the same way as regular WWW files. As for OCSP, careful reader may expect that we will run an instance of `openssl ocsp` behind reverse HTTP proxy. Although it would definitelly also work, OpenSSL documentation clearly states that most of the options are for testing or debugging purposes.[23], not for regular use. Instead of that, we will create a simple shell CGI script, that will be invoked by http daemon and which will call OpenSSL binary.

```
server "ca.foo.org" {
  listen on * port 80
  root "/htdocs/ca.foo.org"

  location "/ocsp/*" {
    slowcgi
    root "/cgi-bin/ocsp.cgi"
  }
}
```

/etc/httpd.conf

```
#!/bin/sh
# Inspired by https://www.pbdigital.org/post/2019-10-28-ca-ocsp-openbsd/

if [ "$REQUEST_METHOD" == "POST" ]; then
    if [ "$CONTENT_TYPE" == "application/ocsp-request" ]; then
      CA_NAME = $(basename "$DOCUMENT_URI")
      BASEDIR="/etc/ssl/${CA_NAME}"

      if [ -d "$BASEDIR" ]; then
        INDEX="$BASEDIR/db/index"
        CA="$BASEDIR/ca.crt"
        RSIGNER="$BASEDIR/ocsp.crt"
```

```
        RKEY="$BASEDIR/private/ocsp.key"

        echo "Content-type: application/ocsp-response"
        echo ""
        cat | openssl ocsp -index $INDEX -CA $CA -rkey $RKEY -rsigner
    $RSIGNER -nmin 5 -reqin /dev/stdin -respout /dev/stdout | cat
      else
        echo "Unknown Certificate Authority."
      fi
    else
      echo "Invalid OCSP request."
    fi
fi
```

/var/www/cgi-bin/ocsp.cgi

5.0.6 DNS

Domain Name System (DNS) is responsible for translating human-friendly domain names into numerical IP addresses used in network.

OpenBSD default installation comes with nsd and unbound, which are opensource, secure, and modern DNS implementations with disjunctive functions developed by NLnet Labs.[24]

- **nsd** is strictly an authoritative DNS name server without recursive and/or caching function
- **unbound** is DNSSEC validating, recursive and caching DNS resolver without authoritative function. In this example, we will use unbound as dns-adblock as well.

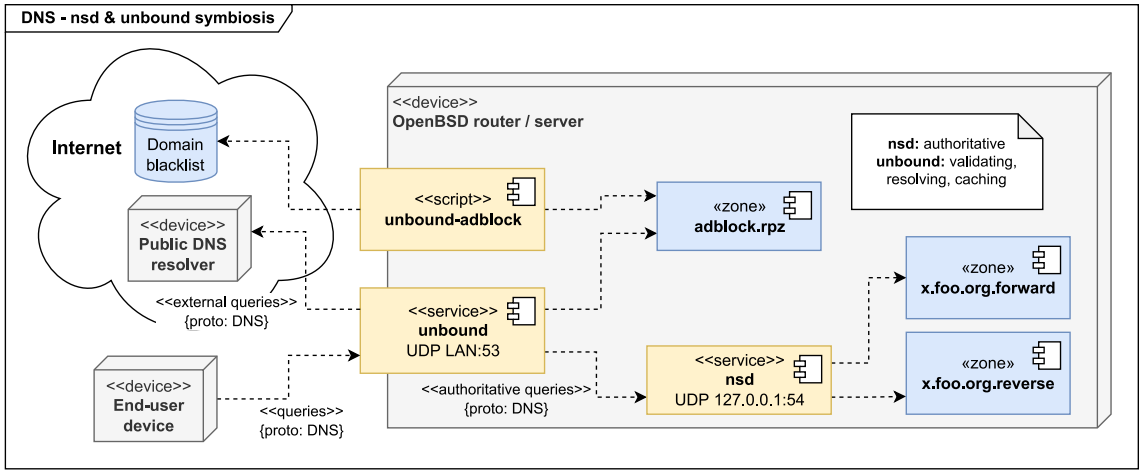


Figure 5.3: DNS - nsd & unbound symbiosis UML diagram

At first, we allow incoming DNS in server's firewall:

```
# Permit DNS and DNS over TLS
pass in proto {udp, tcp} to port {53, 853}
```

5.0.6.1 nsd

According to the Figure 5.1, we will configure nsd to run on loopback interface, because only unbound, running on the same machine, will interact with nsd. We will start nsd on

port 54 to avoid collision with unbound running on standard DNS port 53, and to ease debugging so we will be able to specify which DNS server we will query using `dig` tool. Note that nsd changes it's root to `/var/nsd` after start.

```
server:
  # 0 for regular use, 1 for detailed information, 2 for soft errors, 3
    prints more information
  verbosity: 3
  port: 54
  ip-address: 127.0.0.1

  hide-identity: yes
  hide-version: yes

  chroot: "/var/nsd"

  logfile: "/var/nsd/log/nsd.log"

  do-ip4: yes
  do-ip6: no

remote-control:
  control-enable: no

zone:
  name:     a.foo.org
  zonefile: "master/%s.forward"
zone:
  name:     160-161.16.172.in-addr.arpa
  zonefile: "master/a.foo.org.reverse"
zone:
  name:     b.foo.org
  zonefile: "master/%s.forward"
zone:
  name:     162.16.172.in-addr.arpa
  zonefile: "master/b.foo.org.reverse"
zone:
  name:     c.foo.org
  zonefile: "master/%s.forward"
zone:
  name:     163.16.172.in-addr.arpa
  zonefile: "master/c.foo.org.reverse"
```

/var/nsd/etc/nsd.conf

```
$ORIGIN a.foo.org.
$TTL 86400

@ 3600 SOA srv.a.foo.org. admin.foo.org. (
  0       ; serial
  1800    ; refresh
  7200    ; retry
  1209600 ; expire
  86400   ; min TTL
)

@ IN NS srv

srv       IN A 172.16.160.1
mikrotik  IN A 172.16.160.2
printer   IN A 172.16.160.5
pc        IN A 172.16.160.6
```

/var/nsd/zones/master/a.foo.org.forward

```
$ORIGIN a.foo.org.
$TTL 86400

160-161.247.10.in-addr.arpa. IN SOA srv.a.foo.org. admin.foo.org. (
  0         ; serial
  1800      ; refresh
  7200      ; retry
  1209600   ; expire
  86400     ; min TTL
)

1.160.16.172.in-addr.arpa. IN PTR srv
2.160.16.172.in-addr.arpa. IN PTR mikrotik
5.160.16.172.in-addr.arpa. IN PTR printer
6.160.16.172.in-addr.arpa. IN PTR pc
```

/var/nsd/zones/master/a.foo.org.reverse

```
$ORIGIN b.foo.org.
$TTL 86400

@ 3600 SOA srv.a.foo.org. admin.foo.org. (
  0         ; serial
  1800      ; refresh
  7200      ; retry
  1209600   ; expire
  86400     ; min TTL
)

router      IN A 172.16.162.1
mikrotik    IN A 172.16.162.2
printer     IN A 172.16.162.5
pc          IN A 172.16.162.6
```

/var/nsd/zones/master/b.foo.org.forward

```
$ORIGIN b.foo.org.
$TTL 86400

162.16.172.in-addr.arpa. IN SOA srv.a.foo.org. admin.foo.org. (
  0         ; serial
  1800      ; refresh
  7200      ; retry
  1209600   ; expire
  86400     ; min TTL
)

1.162.16.172.in-addr.arpa. IN PTR router
2.162.16.172.in-addr.arpa. IN PTR mikrotik
5.162.16.172.in-addr.arpa. IN PTR printer
6.162.16.172.in-addr.arpa. IN PTR pc
```

/var/nsd/zones/master/b.foo.org.reverse

```
# Check nsd service configuration
nsd-checkconf /var/nsd/etc/nsd.conf

# For each zone, check the zone files
nsd-checkzone a.foo.org /var/nsd/zones/master/a.foo.org.forward
nsd-checkzone 160-161.16.172.in-addr.arpa /var/nsd/zones/master/a.foo.org.
    reverse

# Start and enable the Name Server Daemon
rcctl start nsd
```

```
rcctl enable nsd

# Perform test query. Note the port must be specified since nsd is not
    running on the standard DNS port.
dig @127.0.0.1 -p 54 srv.a.foo.org
dig @127.0.0.1 -p 54 -x 172.16.160.1
```

5.0.6.2 unbound

Unbound will be configured to run on both loopback and external (LAN) interface and to listen on standard DNS port 53. After some security measurements, namely setting unbound to not expose its identity and version, we specify which IP addresses will be able to access unbound DNS server and receive responses. Careful reader expects it to be networks A, B, C and VPN site-to-site plus VPN client-to-site.

Response Policy Zone (RPZ) configuration is prepared there for DNS adblock which setup is covered in the following section.

At the end of the configuration file, we specify *stub zones* which tells unbound to query a specific DNS server for the specified domains. In our example, we instruct unbound to query nsd instance running on localhost:54 that we configured in the previous section.

```
server:
  verbosity: 5   # 1 for regular operation, 4 for debugging
  num-threads: 4 # 1 means no threading

  port: 53
  interface: 127.0.0.1
  interface: 172.16.160.1

  do-ip4: yes
  do-ip6: no
  do-udp: yes
  do-tcp: yes

  hide-identity: yes
  hide-version: yes
  qname-minimisation: yes

  access-control: 127.0.0.0/8 allow
  access-control: 172.16.160.0/23 allow # Site A
  access-control: 172.16.162.0/24 allow # Site B
  access-control: 172.16.163.0/24 allow # Site C
  access-control: 172.16.164.0/25 allow # VPN site-to-site
  access-control: 172.16.164.128/29 allow # VPN client-to-site

  chroot: "/var/unbound"
  directory: "/var/unbound"

  # relative path to directory variable
  logfile: "log/unbound.log"

  private-address: 10.0.0.0/8
  private-address: 172.16.0.0/12
  private-address: 192.168.0.0/16
  private-address: 169.254.0.0/16
  private-address: fd00::/8
  private-address: fe80::/10

  # Allow these domain, and all its subdomains to contain private addresses
    .
```

```
  private-domain: a.foo.org
  private-domain: b.foo.org
  private-domain: c.foo.org

  # relative path to directory variable
  root-hints: "db/root.hints"
  auto-trust-anchor-file: "db/root.key"

  insecure-lan-zones: yes
  domain-insecure: a.foo.org
  domain-insecure: 160-161.16.172.in-addr.arpa.
  domain-insecure: b.foo.org
  domain-insecure: 162.16.172.in-addr.arpa.
  domain-insecure: c.foo.org
  domain-insecure: 163.16.172.in-addr.arpa.

  # unbound by default refuses to send any DNS queries to localhost. Only
    needed if localhost (127.0.0.1@port) is used as stub-addr in stub zone
  do-not-query-localhost: no

  # Required modules for RPZ
  module-config: "respip validator iterator"
  rpz:
    name: "unbound-adblock"
    zonefile: "/var/unbound/db/adblock.rpz"
    rpz-log: yes
    rpz-log-name: "unbound-adblock"

  # Enable remote control on loopback interface for unbound-adblock
  remote-control:
    control-enable: yes
    control-interface: 127.0.0.1

  # This local-zone line will tell unbound that private addresses like
    172.16.160.5 can send queries to a stub zone authoritative server like
    NSD.
  local-zone: 160-161.247.10.in-addr.arpa. nodefault
  local-zone: a.foo.org nodefault
  local-zone: 162.247.10.in-addr.arpa. nodefault
  local-zone: b.foo.org nodefault
  local-zone: 163.247.10.in-addr.arpa. nodefault
  local-zone: c.foo.org nodefault

# Configure forward and reverse stub zones (point to NSD instance)
stub-zone:
  name: a.foo.org
  stub-addr: 127.0.0.1@54
stub-zone:
  name: b.foo.org
  stub-addr: 127.0.0.1@54
stub-zone:
  name: c.foo.org
  stub-addr: 127.0.0.1@54
```

/var/unbound/etc/unbound.conf

```
# Check configuration
unbound-checkconf /var/unbound/etc/unbound.conf

# If there are no errors, start and enable the daemon
rcctl start unbound
rcctl enable unbound

# Test local domain queries
```

```
dig @127.0.0.1 srv.a.foo.org
dig @127.0.0.1 -x 172.16.160.1

# Test DNSSEC validation
dig @127.0.0.1 com. SOA +dnssec

# Test DNSSEC validation with success expected
dig @127.0.0.1 sigok.verteiltesysteme.net

# Test DNSSEC validation with fail expected
dig @127.0.0.1 sigfail.verteiltesysteme.net
```

5.0.6.3 System resolver config

Since we have our local DNS server working, we can change the system's resolver config so primary DNS server is localhost.

```
domain a.foo.org
nameserver 127.0.0.1
nameserver 1.1.1.1
```

/etc/resolv.conf

5.0.6.4 DNS AdBlock - unbound-adblock

Users and devices in our network are targeted by *pervasive monitoring* - analytics and tracking mechanisms that send data about how we use our devices to companies that collect it. This relates to WWW browsing and affects almost every device connected to the Internet - Laptops, smartphones, smart TVs, network printers, and generic IoT devices like IP cameras or thermometers. While most of them should be blocked from Internet access entirely (as shown in `pf.conf`), because we access them via VPN, some can not by their very nature - smartphones, laptops, and smart TVs. We can effectively mitigate the effects of pervasive monitoring by applying DNS AdBlock in our network.

It must be said, that there are many opensource alternatives (ready-to-use DNS servers) as well. Here, we will use a very similar solution to pf-badhost, from the same author, Jordan Geoghegan. Again, it is a very simple and neat shell script that downloads malicious hostnames from public lists and populates local RPZ (Response Policy Zone) file that can be used by unbound, BIND, Knot, and PowerDNS resolvers.[25]

Let's see how to deploy unbound-adblock on our server by following install instructions for OpenBSD.[26] Installation process is left uncommented because it is obvious and very similar to pf-badhost.

```
cd /tmp
ftp https://www.geoghegan.ca/pub/unbound-adblock/latest/unbound-adblock.sh
useradd -s /sbin/nologin -d /var/empty _adblock
install -m 755 -o root -g bin unbound-adblock.sh /usr/local/bin/unbound-
    adblock
install -m 644 -o _adblock -g wheel /dev/null /var/unbound/db/adblock.rpz
install -d -o root -g wheel -m 755 /var/log/unbound-adblock
install -o _adblock -g wheel -m 640 /dev/null /var/log/unbound-adblock/
    unbound-adblock.log
install -o _adblock -g wheel -m 640 /dev/null /var/log/unbound-adblock/
    unbound-adblock.log.0.gz

cat /etc/doas.conf
...
```

```
permit root
permit nopass _adblock cmd /usr/sbin/unbound-control args -q status
permit nopass _adblock cmd /usr/sbin/unbound-control args -q flush_zone
    unbound-adblock
permit nopass _adblock cmd /usr/sbin/unbound-control args -q
    auth_zone_reload unbound-adblock
...

# Edit /var/unbound/etc/unbound.conf - we have already done it before
# unbound-checkconf /var/unbound/etc/unbound.conf

rcctl restart unbound
doas -u _adblock unbound-adblock -O openbsd

crontab -u _adblock -e
~ 0~1 * * * -s unbound-adblock -O openbsd
```

As Jordan Geoghegan mentions, this is not sufficient, though. Many devices and apps use hardcoded Google DNS servers. We need to add several rules in router's firewall that will redirect queries to Google DNS to our local DNS server. Luckily, router is our device itself and we have already included the rules in `pf.conf`. All routers in our model network, r_a, r_b, r_c redirect to the same local DNS server being hosted on r_a.

```
table <gooDNS4> {8.8.8.8 8.8.4.4}
table <gooDNS6> {2001:4860:4860::8888 2001:4860:4860::8844}
pass in quick to <gooDNS4> rdr-to $adblock
pass in quick to <gooDNS6> rdr-to $adblock6
```

5.0.7 VPN

The intention is to:

- Link the networks A, B, C
- Enable client devices in Internet to connect to networks A, B, C

In wireguard context, there is no client or server, there are just multiple communicating endpoints called *Peers*. Unlike IPSec or OpenVPN, Wireguard does not take advantage of X.509 certificate capabilities and Public Key Infrastructure as OpenVPN do. Instead, it works only with private and public keys as integers.

5.0.7.1 Central router configuration

Central peer (router r_a) runs OpenBSD. Although OpenBSD provides optional package `wireguard-tools` that may be used to configure wireguard interface, we can do all job with `ifconfig` command.
`man ifconfig` and `man wg` may be handy.
At first, let's modify the firewall.

```
# Allow Wireguard connection to UDP (we have already did so)
pass in on egress proto udp to port { <PORT_SITE_TO_SITE> <
    PORT_CLIENT_TO_SITE> }

# Create site-to-site and client-to-site interface
ifconfig wg0 create
ifconfig wg1 create

# Note the public keys, which will be needed for peer configuration
ifconfig wg0
ifconfig wg1
```

Next, we can create interface configuration file for both wireguard interfaces:

```
wgkey <PRIVATE_KEY>
wgport <PORT_SITE_TO_SITE>
wgpeer <RouterB_PUBKEY> wgaip 172.16.164.130/32 wgaip 172.16.162.0/24
wgpeer <RouterC_PUBKEY> wgaip 172.16.164.131/32 wgaip 172.16.163.0/24
inet 172.16.164.129/29
mtu 1500
!route add 172.16.162.0/24 172.16.164.129
!route add 172.16.163.0/24 172.16.164.129
up
```

/etc/hostname.wg0

Where the `<PRIVATE_KEY>` may be generated using `openssl rand -base64 32`

```
wgkey <PRIVATE_KEY>
wgport <PORT_CLIENT_TO_SITE>
mtu 1500
inet 172.16.164.1/25
mtu 1500
wgpeer <CLIENT1_PUBKEY> wgaip 172.16.164.2/32
wgpeer <CLIENT2_PUBKEY> wgaip 172.16.164.3/32
...
up
```

/etc/hostname.wg1

Finally, apply the changes
`sh /etc/netstart`

5.0.7.2 Peripheral router configuration

Bellow is the example of r_b configuration. There is only one, `wg0` interface. The configuration of r_c is analogous to it.

```
wgkey <PRIVATE_KEY>
wgpeer <RouterA_PUBKEY> wgendpoint <RouterA_PUBLIC_IP>:<PORT_SITE_TO_SITE>
    wgpka 25 wgaip 172.16.164.130/32 wgaip 172.16.160.0/23 172.16.162.0/24
    172.16.163.0/24
inet 172.16.164.130/29
mtu 1500
!route add 172.16.160.0/23 172.16.164.130
!route add 172.16.163.0/24 172.16.164.130
up
```

/etc/hostname.wg0

5.0.7.3 Client device configuration

In the case of a client device, we usually need to redirect all traffic to a VPN tunnel (change default route) to pretend we are in the remote network. Client devices may have different Wireguard implementations (Android, Windows), so the general idea is as follows:

- Address: `172.16.164.2/25`
- MTU: `1500`
- DNS server: `172.16.160.1`
- Peer
  - Endpoint: `<RouterA_PUBLIC_IP>:<PORT_CLIENT_TO_SITE>`
  - Persistent keepalive: 25 seconds to keep NAT table record on router
  - Allowed IPs: `0.0.0.0/0` (default route)

5.1 System upgrade

OpenBSD developer team releases a new system version twice a year.[27] Besides that, there are **system patches** released as soon as possible.[28]. Luckily, both patching and upgrading is very simple and convenient in comparison with some Linux distributions for example. OpenBSD has significant advantage of being developed as a complete operating system.

5.1.1 Updating packages

If we decide to install software from precompiled package system, we can update the packages using `pkg_add` command:

```
pkg_add -Uu
```

5.1.2 Patching

```
# We can list all available patches at first
syspatch -c

# Finally, we apply the patches
syspatch
```

5.1.3 Upgrading

Upgrades are only supported from one release to the release immediately following.[29] Since OpenBSD 6.6, system upgrade became again far more simple with `sysupgrade` tool. In addition, OpenBSD developers publish a special webpage with detailed upgrade instructions for every single release.

```
sysupgrade
# Reboot
syspatch
pkg_add -Uu
sysmerge -d
# Resolve all conflicts in configuration
```

5.1.4 Updating the Ports Tree

```
rm -rf /usr/ports
cd /tmp
ftp https://cdn.openbsd.org/pub/OpenBSD/$(uname -r)/{ports.tar.gz,SHA256.
    sig}
signify -Cp /etc/signify/openbsd-$(uname -r | cut -c 1,3)-base.pub -x
    SHA256.sig ports.tar.gz
cd /usr
tar xzf /tmp/ports.tar.gz
```

Chapter 6

WiFi security

6.1 Introduction

Securing the access to Wi-Fi networks is de-facto standard in 2023. We hardly find a private Wi-Fi network that is not secured by any type of prevention of unauthorized access. The number of portable devices that are being connected wirelessly increases for many reasons. First of all, the number of devices that are being connected to computer networks incessantly grows. IoT Analytics reports that in 2022, the number of connected IoT devices was growing 18%[30]. Besides IoT devices that are usually hidden from our eye, there are devices that we use directly - laptops, smartphones, smart watches, e-book readers, and many more. Some of the mentioned devices even do not have any wired network interface so they fully depend on wireless communication. In many use cases, it is also favorable to connect a device wirelessly instead of making expensive construction changes. Sometimes, we need to connect a device that moves permanently - robots, autonomous vehicles and moving parts in machines. Wireless communication is, by its very nature, much more vulnerable and much easier to eavesdrop. Unlike wired networks that are physical and distinguishable, wired networks are intangible for a human. Most of the time when we use public Wi-Fi networks, we have no clue about what device we connect to, what device do we supply our credentials and data. The task is to ensure maximum security in this type of networks. While there are numerous wireless standards, each suitable for a particular use case, we will focus solely on Wi-Fi networks, which are the most common at homes and in organizations. Wi-Fi is a trademark of the non-profit Wi-Fi Alliance[31], in our context, it is a family of wireless network protocols based on the IEEE 802.11 family of standards.

6.1.1 IEEE 802.11i (WPA2)

Small Wi-Fi networks at homes and small organizations typically use WPA or 802.11i security, which use preshared secret that is distributed among all users. If we take strong shared keys and it's reasonable secreting into consider, securing such networks with shared secret is acceptable.
However, a shared secret quickly becomes inapplicable when we imagine a larger-scale use case that covers a higher number of people that may fluctuate - a typical situation in a company. Would all of them share a credentials to a single mail account? Presumably no. When an employee leaves the job, it is regrettably not always to the satisfaction of both parties. Employer should always try hard to prevent any kind of sensitive information leak. A shared secret does not help with that in any way. Through it all, there still exist organizations that use such type of authentication. One of the aims of this thesis is to make the world a bit better again in this particular field.

6.1.2 IEEE 802.1X (RADIUS)

IEEE 802.1X is an IEEE standard for providing authentication mechanisms for both LAN and WLAN. 802.1X introduces the following entities:[32]

- Supplicant - end device connecting to the LAN/WLAN
- Authenticator - a network device that provides a data link between the client and network
- Authentication server - a trusted server that validates requests for networks access and tells the authenticator to either connect a client to the network or to prevent

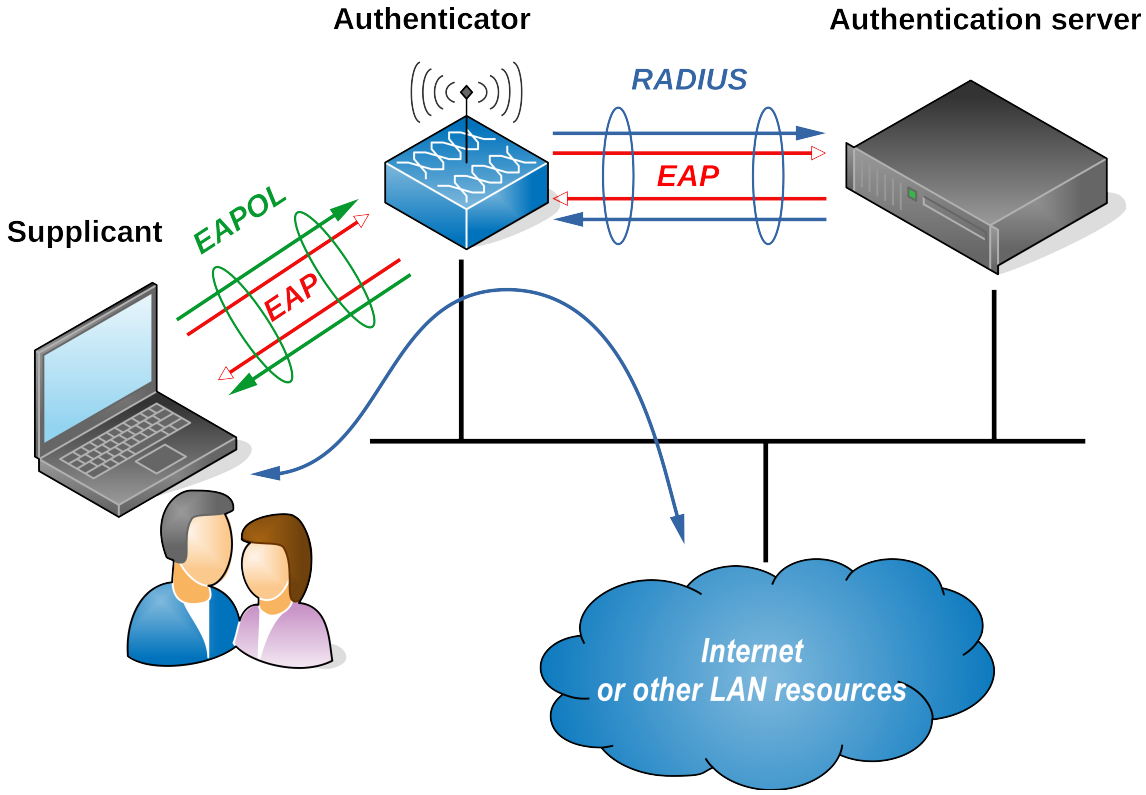


Figure 6.1: 802.1X overview

For us, the most interesting part is the Extensible Authentication Protocol (EAP) itself. It is used to pass the authentication information between the supplicant and the authentication server. In the context of EAP, the authenticator acts only as proxy and allows the two mentioned parties to communicate. EAP, represented by RFC 3748 defines multiple authentication protocols.
EAP-TLS and EAP-PEAP authentication types are the most deployed ones. EAP-PEAP provides support for password-based protocols, EAP-TLS is certificate-based.

6.1.2.1 EAP-TLS protocol

EAP-TLS protocol is considered to be the simplest, fastest and strongest among EAP authentication methods.[33]. The reason is, it relies just on mutual authentication using X.509 certificates which, being created using strong public key algorithms, provide very strong security. As a bonus, it is an IETF open standard.
Let's take a look at EAP-TLS authentication flow:

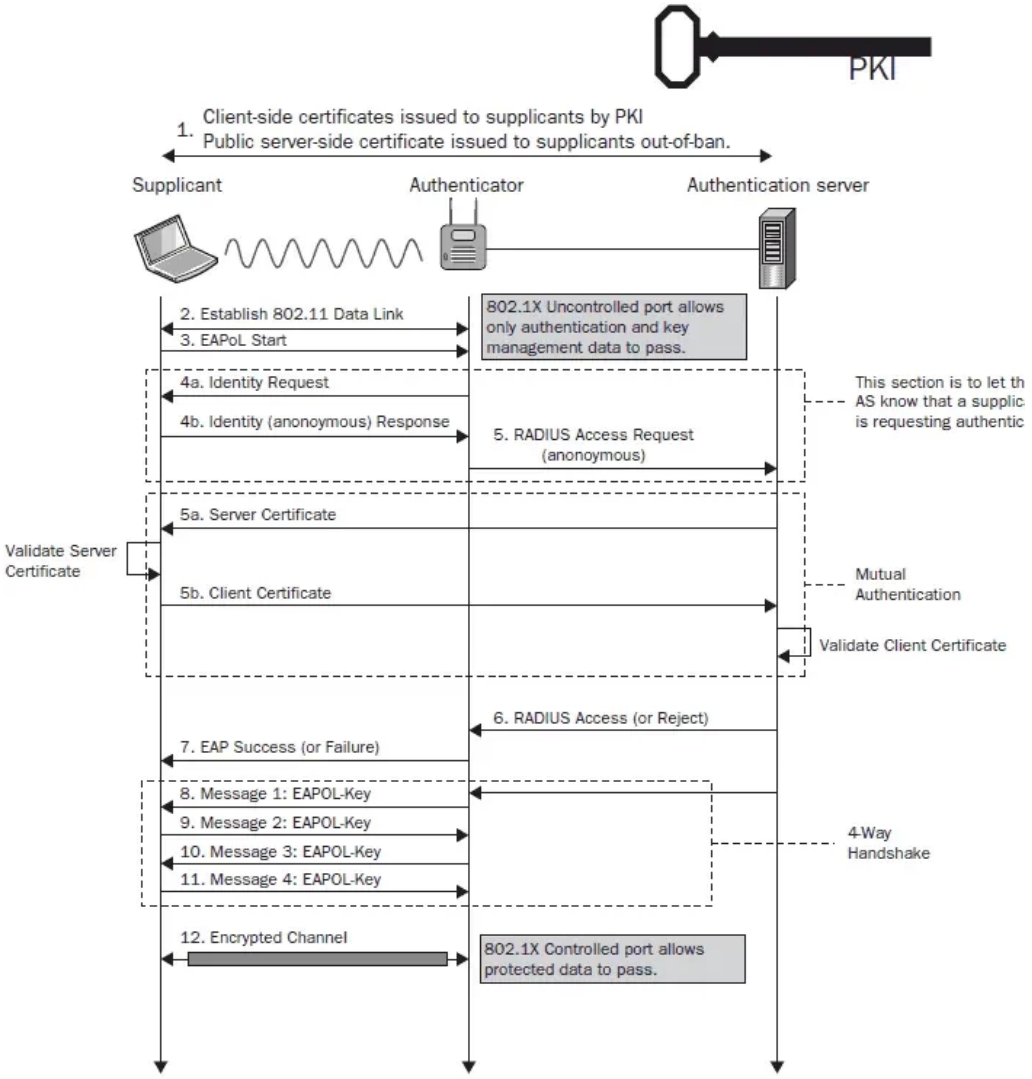


Figure 6.2: EAP-TLS authentication sequence diagram. Source: [34]

Most important is the "Mutual authentication" section. At first, authentication server proves its identity to supplicant, then the supplicant proves it's identity to the authentication server. In our scenario, server uses a certificate issued by Foo DV TLS CA, and supplicant use a certificate issued by Foo EAP-TLS certificate authority. Both parties have the certificate authorities installed mutually. After a successfull authentication, authentication server tells the authenticator to finish the connect process with the end device. Then, the access point and the device exchange Group Transient Key (GTK) and Pairwise Transient Key (PTK). Temporary keys (GTK and PTK) are then used for TKIP or CCMP block encryption protocol which encrypts the communication channel.[35]

6.2 RADIUS server configuration and tests

```
# Build FreeRADIUS ================================
cd /usr/ports/net/freeradius
make show=FLAVORS
env FLAVOR="no_freetds no_iodbc no_ldap no_memcached no_mysql no_pgsql
    no_python" make install
make clean

# Configure ========================================
cp -R /etc/raddb /etc/raddb.orig

# Remove inner-tunnel site that is used only for EAP-TTLS and PEAP
rm /etc/raddb/sites-enabled/inner-tunnel

# Disable unused modules
rm /etc/raddb/mods-enabled/pap
rm /etc/raddb/mods-enabled/chap
rm /etc/raddb/mods-enabled/mschap
rm /etc/raddb/mods-enabled/passwd
rm /etc/raddb/mods-enabled/ntlm_auth
rm /etc/raddb/mods-enabled/inner-tunnel
rm /etc/raddb/mods-enabled/realm

# Remove everything in certs directory except DH parameters and symlink to
    random device
cd /etc/raddb/certs && rm -v !("dh"|"random")

# Move keys and certificates from CA to /etc/raddb/certs directory:
cp /tmp/certs/eaptls/ca.pem /etc/raddb/certs/
cp /tmp/certs/dvtls/radius.foo.org.pem /etc/raddb/certs/server.pem
cp /tmp/certs/dvtls/radius.foo.org.key /etc/raddb/certs/server.key
chown root:wheel /etc/raddb/certs/*
chmod 440 /etc/raddb/certs/*.key /etc/raddb/certs/*.pem

# Configure FreeRADIUS ---------------------------
# Configuration files are located in /etc/raddb

# define clients = authenticators = Wi-Fi APs of the RADIUS server
vim /etc/raddb/clients.conf
# Insert contents from freeradius_conf/clients.txt

# EAP-TLS ----------------------------------------
vim /etc/raddb/mods-enabled/eap
# Insert contents from freeradius_config/eap.txt

# Sites ------------------------------------------
vim /etc/raddb/sites-enabled/default
# Insert contents from sites-enabled_default.txt

# Modify PF firewall -----------------------------
# Permit RADIUS
pass in proto udp to port radius

# Debugging --------------------------------------
# Get freeradius version
/usr/local/sbin/radiusd -v

# Check configuration
/usr/local/sbin/radiusd -XC

# Run in debug mode
/usr/local/sbin/radiusd -X
```

```
60
61 # !!! Perform tests following the eapol_tests.txt file !!!
62
63 tail -f /var/log/radius/radius.log
64
65 # Start and enable FreeRADIUS via system daemon manager
66 rcctl start freeradius
67 rcctl enable freeradius
```

6.2.1 eapol_test

After configuring and starting RADIUS server, we will use eapol_test tool to test RADIUS server function locally without need to configure a real authenticator.
This way we can test authentication using fake certificates on both sides.

```
1  # Build wpa_supplicant ============================
2  # eapol_test is part of wpa_supplicant project
3  cd /usr/ports/security/wpa_supplicant
4  make show=FLAVORS
5  make install clean
6
7  # Perform tests ====================================
8  # Login as non-root user
9  mkdir /tmp/eapol_test
10 chmod 700 /tmp/eapol_test
11 cd /tmp/eapol_test
12
13 cp /tmp/certs/dvtls/ca.pem ./ca-dvtls.pem
14 cp /tmp/certs/eaptls/matyas@foo.org.pem ./
15 cp /tmp/certs/eaptls/matyas@foo.org.key ./
16
17 vim eapol_test.conf
18 network={
19   ssid="FooNet"
20   key_mgmt=WPA-EAP
21   proto=WPA2
22   pairwise=CCMP
23   group=CCMP
24   eap=TLS
25   ca_cert="/tmp/eapol_test/ca-dvtls.pem"
26   private_key="/tmp/eapol_test/matyas@foo.org.p12"
27   private_key_passwd="matyas"
28 }
29
30 # START THE SERVER
31
32 # By default, there is localhost client config with "testing123" secret
33 eapol_test -c eapol_test.conf -s testing123
34
35 # Successfull authentication is signalled by "SUCCESS" line
```

6.3 Authenticator configuration

As we mentioned in the introduction of this text, we will use Mikrotik RB411AH as our Wi-Fi access point and 802.1X authenticator.

Let's assume we have already:

- Connected the device to our network
- Provided the following static IP configuration for ethernet interface
  - IP: 172.16.160.2/24
  - Gateway: 172.16.160.1
- Set primary DNS server to 172.16.160.1
- Configured `wlan1` interface with SSID "`FooNet`"

If we are configuring MikroTik for the first time, we can use "Quick Set" configuration wizard with "WISP AP" or "Home AP" mode.[36]

Before making any changes in the networking configuration, we first upload DV TLS certificates to the device.

1. Upload `mikrotik.a.foo.org.p12` file to "Files" section
2. Move to System → Certificates section, click on Import

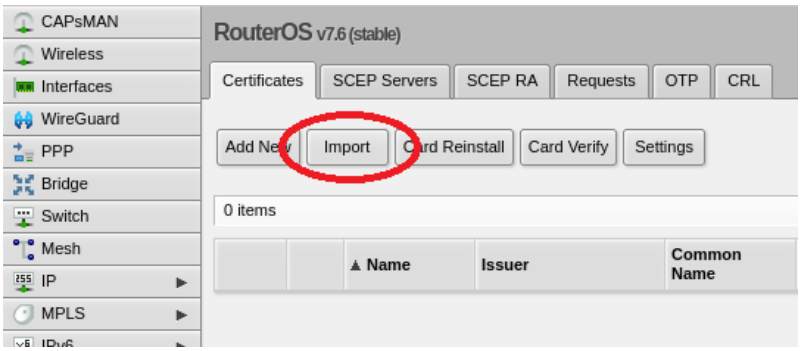


Figure 6.3: MikroTik Webfig certificate import - selecting the file

3. We will be prompted to enter the password for private key. We enter the exact password we set when we created the private key for MikroTik DV certificate.

Figure 6.4: MikroTik Webfig certificate import - entering password

We can use MikroTik Command Line Interface to achieve the same result:

```
1   certificate import file-name=mikrotik.a.foo.org.p12
2   passphrase: ********
3
```

4. After a successfull import, we can see three certificates in the list. Flag `T` denotes trusted certificate, flag `K` denotes that the device has a primary key for the certificate.

		Name	Issuer	Common Name	Subject Alt. Name	Key Size	Days Valid
-	LT	foo-dvtls-ecc-g1	C=CZ S=Czechia L=Prag	Foo DV TLS CA ECC G1		prime256v1	3650
-	T	foo-root-ecc-g1	C=CZ S=Czechia L=Prag	Foo Root CA ECC G1		prime256v1	7300
-	KLT	mikrotik-a-foo-org	C=CZ S=Czechia O=Foo	mikrotik.a.foo.org	DNS:mikrotik.a.foo.org, IP:172.16.160.2	prime256v1	730

Figure 6.5: MikroTik Webfig - certificate list

5. Next step is to enable HTTPS and assign the certificate to all TLS services. We go to IP → Services, click on https and configure the service as follows:

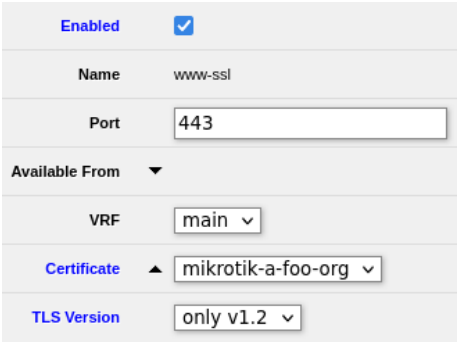


Figure 6.6: MikroTik Webfig - configuring HTTPS service

Then, when we access Mikrotik via HTTPS from a client device that trusts DV TLS Certificate Authority, we see no warning in address bar:

Figure 6.7: Mozilla Firefox - accessing MikroTik Webfig via HTTPS

After clicking on lock icon in address bar, Firefox allows us to inspect the contents of the certificate that is used by the remote server:

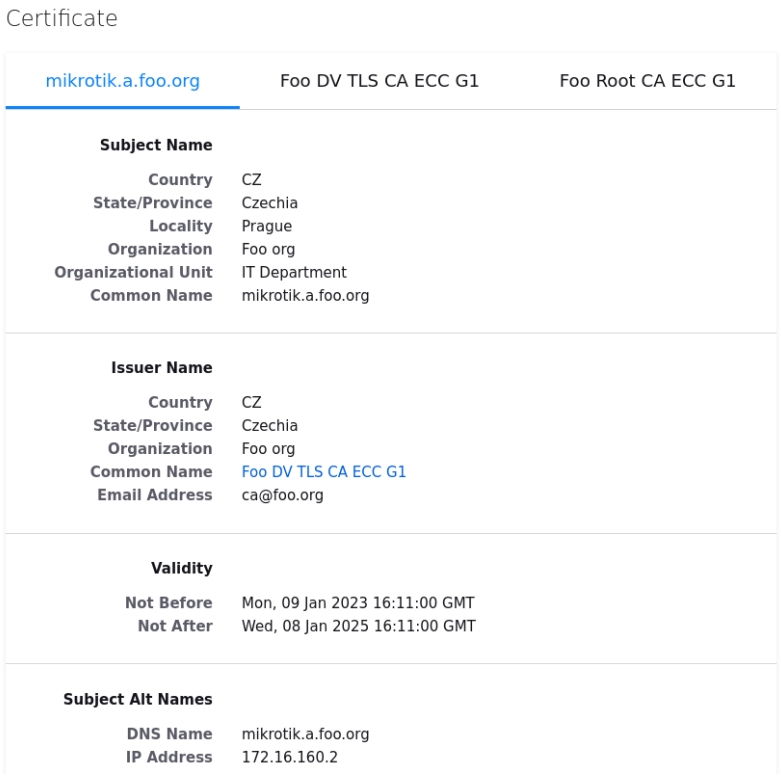


Figure 6.8: Mozilla Firefox - displaying HTTPS certificate details

Now, move on to EAP configuration:[37]

```
# Create a RADIUS profile for wireless use
/radius add address=172.16.160.1 secret=ourmikrosecret service=wireless

# Create WPA2-Enterprise security profile
/interface wireless security-profiles add authentication-types=wpa2-eap
    mode=dynamic-keys name=security_foonet

# Assign the security profile to wlan interface
/interface wireless set wlan1 disabled=no mode=ap-bridge security-profile=
    security_foonet ssid=FooNet

# Finally, we enable the wlan1 interface
/interface wireless enable wlan1
```

We have successfully configured our MikroTik Wi-Fi access point. We are ready to connect end-user devices.

6.4 Connecting devices

6.4.1 UNIX-like operating system with wpa_supplicant

Open source operating systems use wpa_supplicant, a free and open source implementation of IEEE 802.11, 802.11i and 802.1X, to authenticate for network access.[38]

```
ctrl_interface=/var/run/wpa_supplicant
network={
  ssid="FooNet"
  key_mgmt=WPA-EAP
  proto=WPA2
  pairwise=CCMP
  group=CCMP
  eap=TLS
  ca_cert="/etc/cert/ca-dvtls.pem"
  private_key="/etc/cert/user@foo.org.p12"
  private_key_passwd="<PRIVATE_KEY_PASSWORD>"
}
```

wpa_supplicant.conf

6.4.2 Android smartphone

The following procedure demonstrates how to install CA certificate into system and how to connect to Wi-Fi via EAP-TLS on Android One phone with Android version 12. Android One devices come with the original Android by Google instead of having manufacturer's customized system.[39]

6.4.2.1 Installing certificate authorities

By following the procedure below, we import Root CA, DV TLS CA and Personal CA.

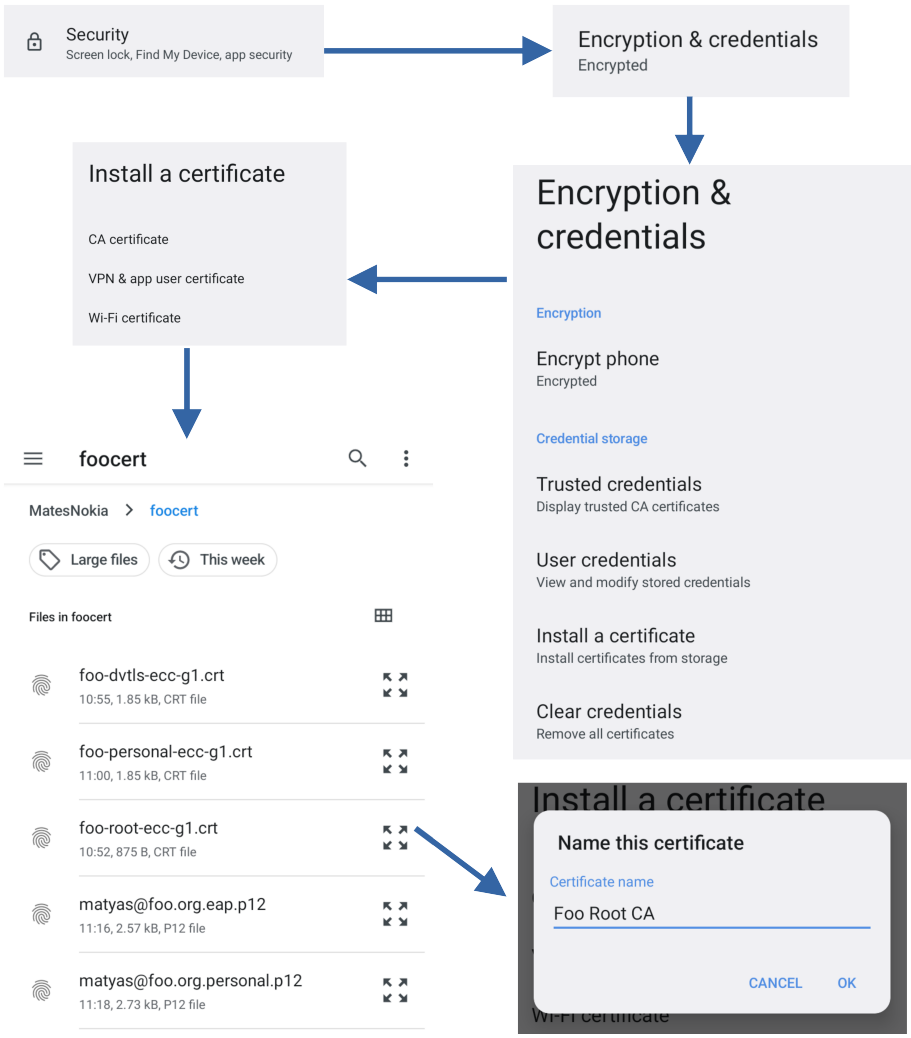


Figure 6.9: Android system configuration - importing a certificate

6.4.2.2 Importing Wi-Fi certificates

This is how we import user's keypair from PKCS#12 bundle (.p12) into system:

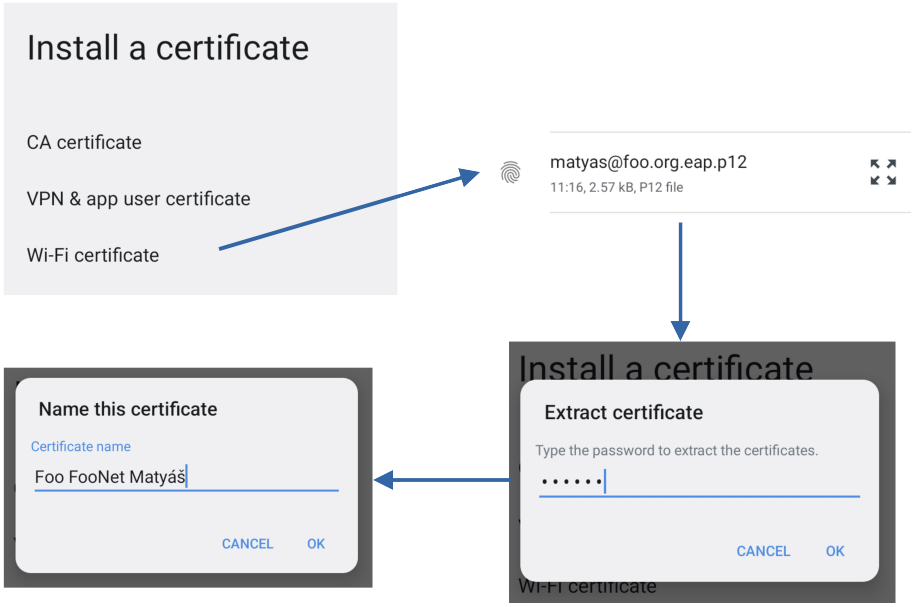


Figure 6.10: Android system configuration - importing a Wi-Fi keypair

Android distinguishes regular certificate authorities and "Wi-Fi certificate" authorities. We need to import Root CA again as an "Wi-Fi certification authority":

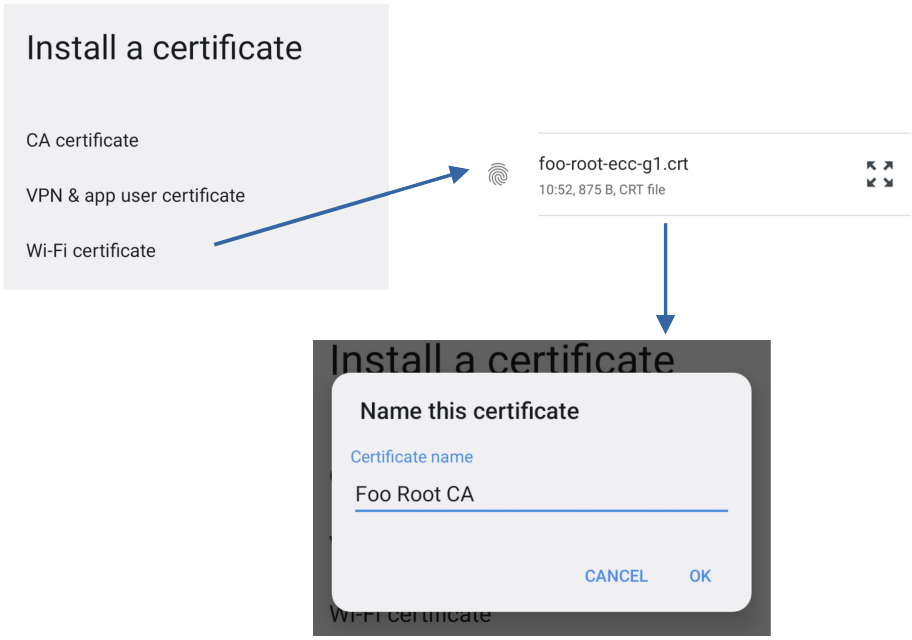


Figure 6.11: Android system configuration - importing a Wi-Fi certificate authority

Finally, we can inspect all user's certificate authorities in Trusted credentials:'

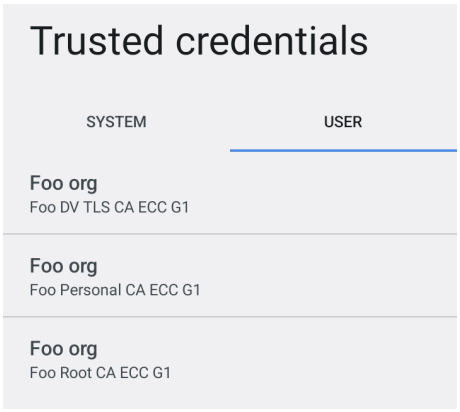


Figure 6.12: Android system configuration - trusted credentials

6.4.2.3 Connecting to Wi-Fi network

After importing all certificates in the system, we are finally ready to connect to FooNet Wi-Fi network.

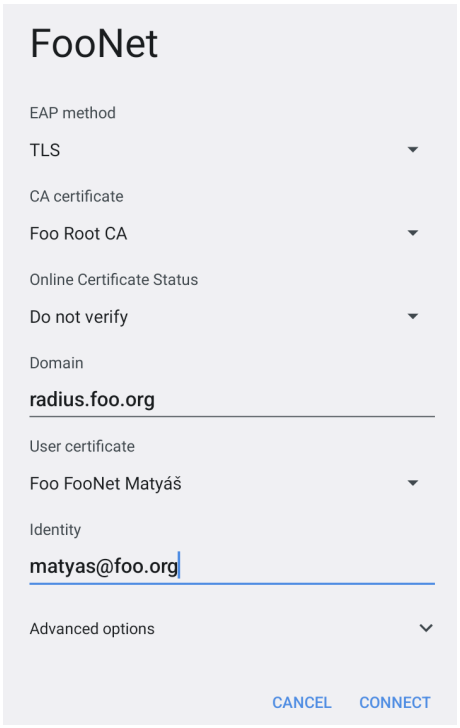


Figure 6.13: Android - configuring EAP-TLS authentication

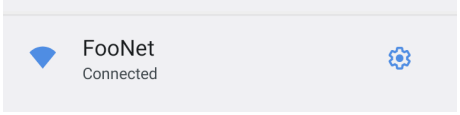


Figure 6.14: Android - a successful connection to Wi-Fi network

6.4.3 Windows 11

6.4.3.1 Certificate installation

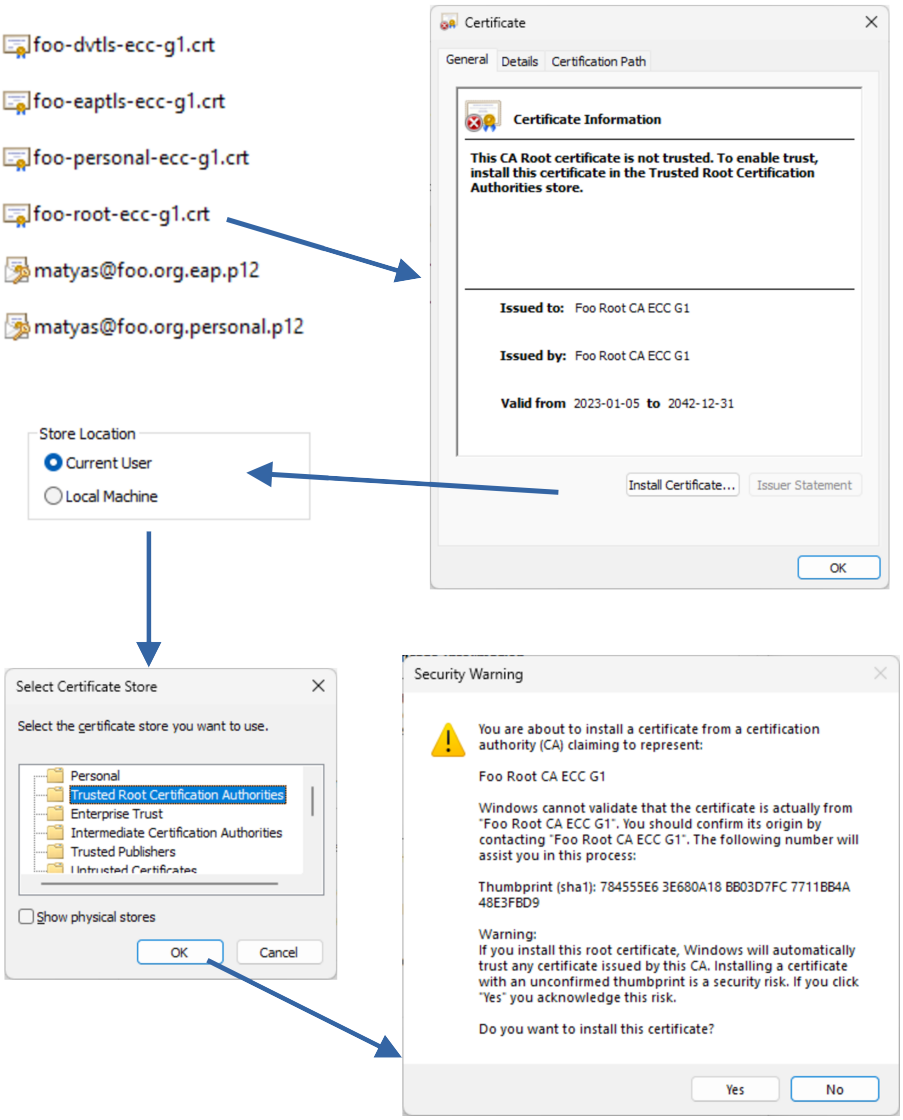


Figure 6.15: Windows - importing Root certificate authority

Issuing certificate authorities (DV TLS and Personal CA) are installed almost in the same way, but into "Intermediate Certification Authorities" store.

After certificate authorities, we follow by installing user certificates from PKCS#12 bundles - `matyas@foo.org.eap.p12` and `matyas@foo.org.personal.p12`. We are prompted to unlock the private key and can specify if we will make the private key exportable or not.

When all certificates have been installed, we see that our personal certificate is fully trusted by the operating system:

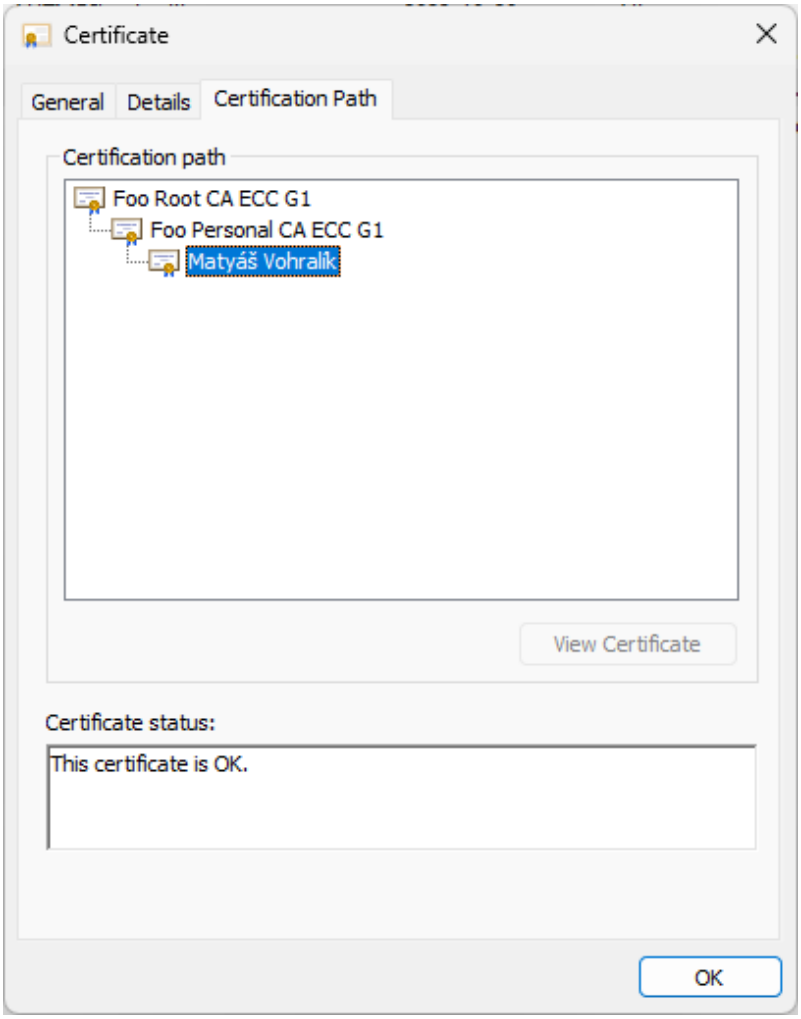


Figure 6.16: Windows - a fully trusted personal certificate

In Windows, we can manage the certificates via Microsoft Management Console `certmgr.msc` extension. For Local Machine store, there is `certlm.msc`. Certificates can be managed also via command line tool `certmgr.exe`.[40] Windows also allows organizations that use Windows Server and deployed Active Directory domain, to distribute certificates to client devices automatically by using Group Policy.[41]

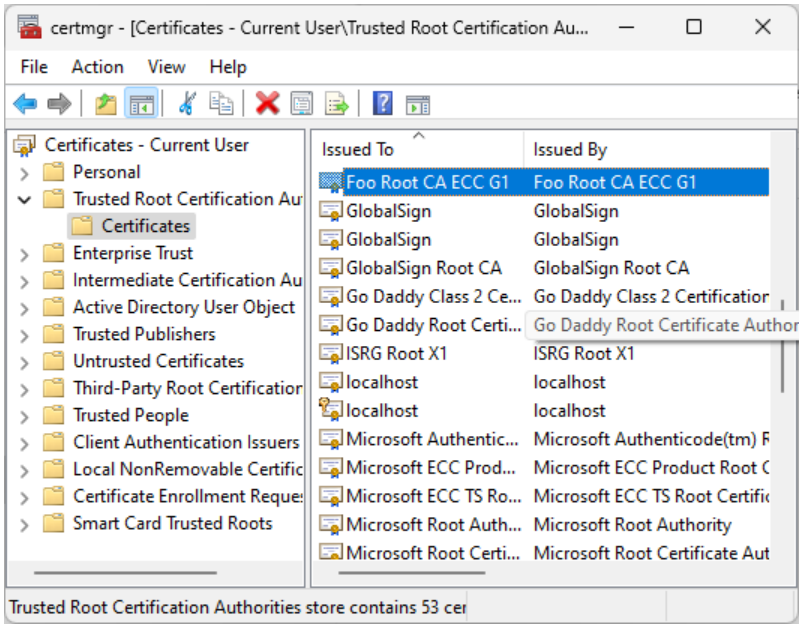


Figure 6.17: Windows Certificate Manager - list of root certificate authorities

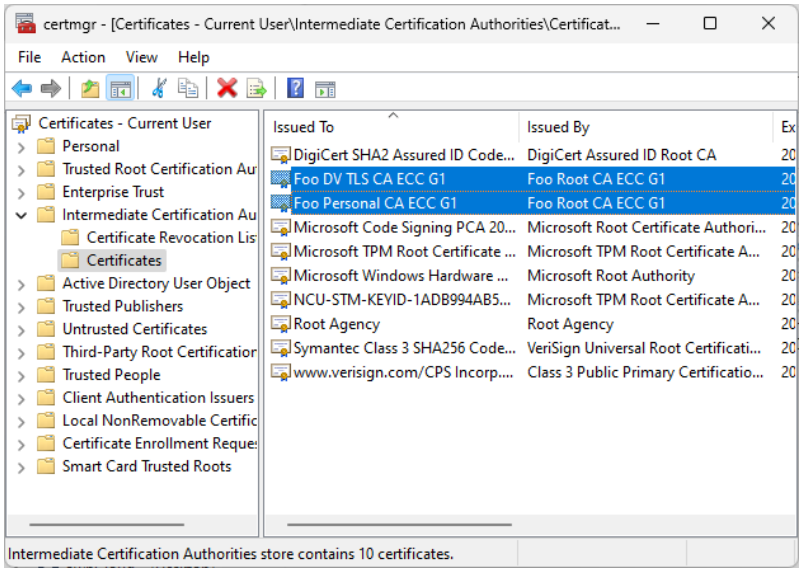


Figure 6.18: Windows Certificate Manager - list of intermediate certificate authorities

6.4.3.2 Connecting to Wi-Fi

Unfortunately, installing EAP-TLS Wi-Fi network in Windows operating system is over-complicated. We need to configure network profile via Control panel.

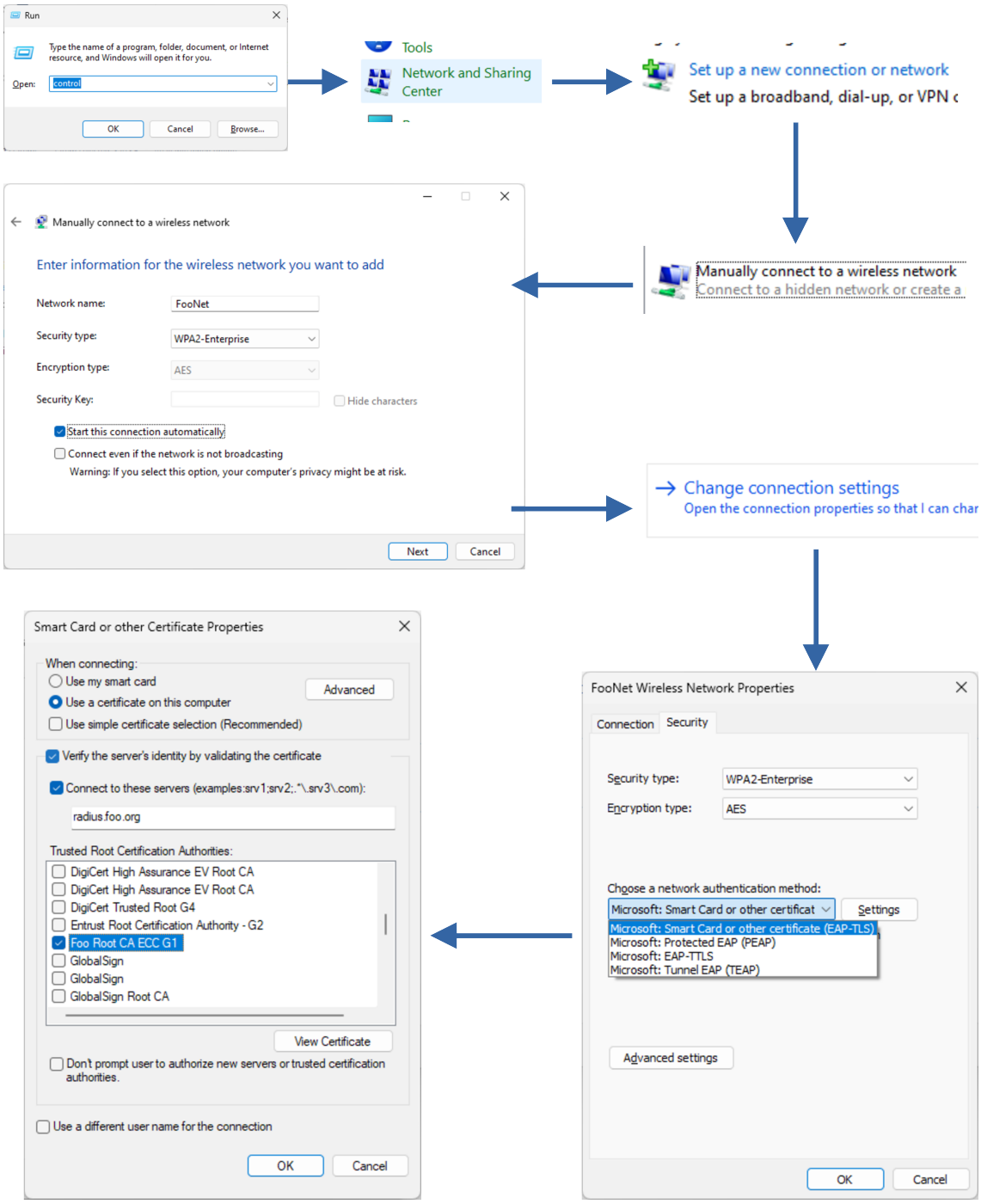
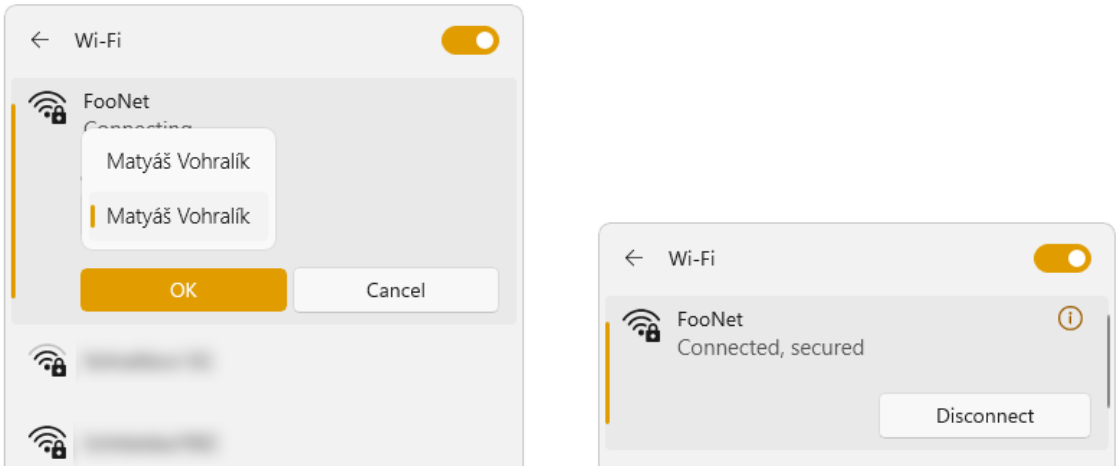


Figure 6.19: Windows - creating wireless network profile manually

After we successfully installed wireless network profile for `FooNet`, we are ready to connect via system tray network wizard.
Unfortunately, that is not entirely true. When we have multiple personal certificates issued for the same Common Name installed, then when we try to connect to `FooNet` network, operating system does not provide any additional information except Common Name to us so we have no choise but try the certificates until we succeed. Author of this text did not manage to find any way how to display more certificate information in wireless connection wizard.



(a) selecting certificate for EAP authentication (b) a successfull connection

Figure 6.20: Windows - System tray network connection wizard

Chapter 7

ARM64 platform

7.1 Introduction

Let's now introduce a single-board arm64 computer (SBC) and demonstrate how to install OpenBSD on it to create a complete solution.

These small, powerful, and low-power devices became very popular starting with the original Raspberry Pi and alongside its clones, these devices are widely used in IoT applications mainly with Linux distributions like OpenWrt, Armbian, or Raspberry Pi OS. Developers of these distributions support many devices and provide installation images that can be easily flashed on SD card[42]. Such installation images usually come with a preconfigured network so there is no need to connect to the board via a serial interface to install the system manually.

These devices used to be widely available for very attractive prices in the past, but due to Global Chip Shortage caused by strong demand and limited supply that origins in COVID-19 pandemic, it is now harder to get a suitable device.[43]
The Author of this text had to obtain an aarch64 platform equipped with two gigabit ethernet ports to use as LAN and WAN, with ethernet chips supported by OpenBSD. While for Linux distributions mentioned above, these requirements would not narrow down the selection, for OpenBSD, things got complicated. We can even find models that were changed due to chip shortage so instead of having two genuine ethernet chipsets, manufacturers decided to replace one of them with a far less known and cheaper chip. This happened to Orange Pi R1 Plus, where the manufacturer replaced one REALTEK chipset by Motorcomm YT8511C to create Orange Pi R1 Plus LTS.[44]

After being unsuccesfull to find a suitable device during the COVID-19 pandemic, the Author of this text finally managed to get the following FriendlyElec NanoPi R4S 4GB for $130.

7.2 NanoPi R4S 4GB

FriendlyElec NanoPi R4S 4GB is small and powerful single-board opensource aarch64 platform with two gigabit Ethernet ports and two USB 3.0 ports. A complete specification can be found on the producer's Wiki:[45]

- **SoC:** Rockchip RK3399 - a hexa-core heterogenous processor (MPSoC) integrating
  - Dual-Core Cortex-A72 (up to 2.0GHz) optimized for high-performance
  - Quad-Core Cortex-A53 (up to 1.5GHz) optimized for low power
  - Mali-T864 GPU with H.264, MVC and VP8 hardware encoding
- **RAM:** 4GB LPDDR4
- **Ethernet:** one Native Gigabit Ethernet, one PCIe Gigabit Ethernet with RealTek RTL8211E and R8111H chipsets
- **Connectivity:** microSD slot, two USB 3.0 Type-A ports, Debug UART
- **LEDs:** 1x power LED and 3x GPIO Controlled LEDs labelled on the case as SYS, LAN, and WAN
- **Powered by** DC 5V/3A via USB-C connector (which does not support any data transfer and Power Delivery), or 2x5-pin header
- **Outer dimensions:** 72 x 72 x 29 mm
- **Weight:** 289 g

The device has very robust case all made of aluminum which acts as thermal mass. Together with indented top side, the case serves well as passive cooler so no fan is installed. The price corresponds to high processing quality in every aspect.
From an overall perspective, it is very well made in comparison with cheaper products.

Powerful CPU, reasonably big 4GB RAM and fast USB 3.0 ports arouse ideas of additional use cases like home NAS, small HTTP server, or Asterisk IP PBX. The device meets Nextcloud system requirements for example.[46]

NanoPi R4S is produced in two versions:

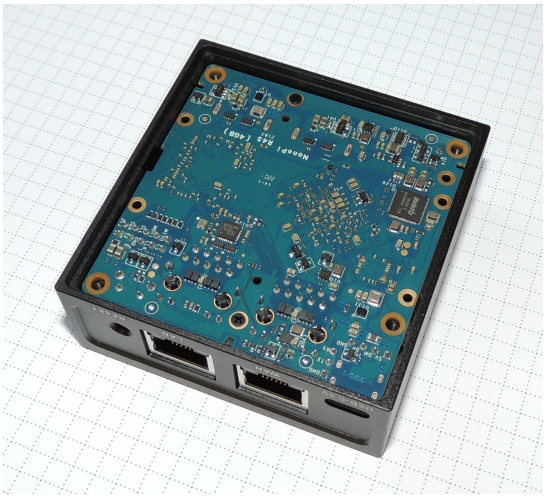
- **Enterprise version**, which comes with unique, permanent and tamper-proof MAC address stored in built-in EEPROM
- **Standard version**, which does not have EEPROM chip and MAC address is software-generated

The producer recommends enterprise users to buy the enterprise version for use in large-scale and / or complicated network scenarios.
Anyway, a MAC address can be changed from software.

(a) overall view

(b) USB 3.0 ports, MMC slot and 1/4” camera screw thread

(c) Ethernet ports, USB-C and Reset button



(d) bottom shield removed

Figure 7.1: FriendlyElec NanoPi R4S 4GB

7.2.1 Preparing installation media

We will use a generic SD/MMC as an installation media. Since we will perform only a single installation, we will choose a HTTP installation that downloads the dependencies on the way.
Install Device Tree Blob (ensure having newest version)

```
cd /usr/ports/sysutils/dtb
make install
```

Install u-boot

```
cd /usr/ports/sysutils/u-boot
env FLAVOR="aarch64" make install

cd /tmp
ftp https://cdn.openbsd.org/pub/OpenBSD/7.2/arm64/SHA256.sig
ftp https://cdn.openbsd.org/pub/OpenBSD/7.2/arm64/miniroot72.img

# Verify checksum
signify -Cp /etc/signify/openbsd-72-base.pub -x SHA256.sig miniroot*.img

```

```
# Wipe partition table
dd if=/dev/zero of=/dev/sd1 bs=2m count=1

# Write the miniroot image to an SD card:
dd if=miniroot72.img of=/dev/rsd1c bs=1m

# Add a board specific DTB file (Allwinner and Rockchip U-Boot images come
    with a default DTB):

mount /dev/sd1i /mnt
mkdir /mnt/rockchip
cp /usr/local/share/dtb/arm64/rockchip/rk3399-nanopi-r4s.dtb /mnt/rockchip/
umount /mnt

dd if=/usr/local/share/u-boot/nanopi-r4s-rk3399/idbloader.img of=/dev/sd1c
    seek=64
dd if=/usr/local/share/u-boot/nanopi-r4s-rk3399/u-boot.itb of=/dev/sd1c
    seek=16384

# Or alternativelly
dd if=/usr/local/share/u-boot/nanopi-r4s-rk3399/u-boot-rockchip.bin of=/dev
    /sd1c seek=64
```

7.2.2 Boot

In order to be able to interact with the computer, we need to connect a 3.3V TTL console via RX,TX and GND pins.

7.2.2.1 Serial console

RK3399 has default baud rate 1.5 Mbaud. OpenBSD sets the speed to 115200 baud. We will use serial to USB converter with PL2303TA chip. We can open serial terminal using a well-known `cu` command:

```
cu -l /dev/cuaU0 -s 115200
# exit with ~.
```

```
System hostname? (short form, e.g. 'foo') srv.a.foo.org
Which network interface do you wish to configure? (or 'done') [re0]
IPv4 address for re0? (or 'autoconf' or 'none') [autoconf]
```

7.2.3 Installation

7.2.3.1 Partitioning

There is a slight difference in disk partition naming convention between Linux and BSD. While in BSD, where disks are numbered and partitions labeled (typically a-h, some), in Linux, disks are labeled and partitions numbered.[47] There are three reserved labels in BSD:

- **a** for root partition
- **b** for swap partition
- **c** overlaps all partitions and describes the entire disk.

Drive partitioning should be set individually to meet specific requirements of router. Since we will not extract the whole source and ports tree, we are free to change the size of the appropriate partition. Partitioning represents an important part of the installation,

because the security of the OpenBSD system inherits the advantages of having data on separate partitions.[48] Future partitioning changes are very painfull or even impossible, large concern should be given to this part.

7.2.3.1.1 Swap When the kernel panics, it dumps the contents of the RAM to the swap partition, which should be at least of the same size as RAM. Provided by the fact that Nano Pi has non-removable on-board RAM and further upgrade is not possible, allocating the capacity of the RAM (4 GB) is fully sufficient even for the future.

7.2.3.2 File sets

- **bsd, bsd.mp, bsd.rd** Since most modern use-case scenarios include multiprocessor systems, we have to install a **bsd.mp** (multiprocessor kernel) instead of single-processor kernel, **bsd**. Ramdisk kernel, bsd.rd is pretty useless on systems with small RAM.
- **baseXX.tgz** [mandatory] core programs
- **etcXX.tgz** [mandatory] the contents of /etc and other directories.
- **manXX.tgz** [optional] man pages for the programs. May not be needed on end-platform, but handy for development and debugging.
- **compXX.tgz** [optional] C and C++ compilers, libraries and other tthe ools needed to develop or compile software from ports collection.
- **gameXX.tgz** [optional] simple games
- **xbaseXX.tgz, xetcXX.tgz, xfontXX.tgz, xservXX.tgz, xshareXX.tgz** [no] X-server specific sets. Since we will interact with the server only via SSH, omitting the installation of these sets enables us to save a considerable disk space.

7.2.3.3 MAC Address

When booting miniroot image, install script needs to download the sets from the Internet. There is one problem that breaks this process.
When we check what is the MAC address of re0 interface, we see it has zero MAC address.

```
srv# ifconfig re0
re0: flags=808843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST,AUTOCONF4> mtu
    1500
        lladdr 00:00:00:00:00:00
        llprio 3
        groups: dhcp egress
        media: Ethernet autoselect (1000baseT full-duplex,rxpause,txpause)
    (...)
```

When we try to ping a device on the same subnet, we succeed. But when we try to ping any device in the Internet, we fail. Some device in the route drops packets with zero MAC address. We need to assign a random MAC address to the re0 interface.
We can generate locally administered address for our device using any online tools like Random Locally Administered Unicast MAC Address Generator from IAN Campbell.[1]

```
ifconfig re0 lladdr de:4c:c3:08:73:7c
```

Now, we can ping openbsd.org and can continue installation. We should not forget to edit the MAC address after the installation.

[1] https://www.hellion.org.uk/cgi-bin/randmac.pl

7.2.3.4 Memory file system

OpenBSD, as well as other operating systems provides memory file system called *mfs*.[47] This type of filesystem is suitable for storing data that changes frequently so the SD card does not wear out. It is indeed important to keep in mind that RAM is volatile memory, which means that it losts its data quickly after power cut. How to deal with this specific is up to particular application.
Here is an example how to mount /tmp on memory filesystem of 1 GB size:

```
(...)
swap /tmp mfs rw,nodev,nosuid,-s=1024m 0 0
(...)
```

/etc/fstab

7.2.3.5 Problem with SD card

Author faced the following problem during installation OpenBSD on Nano Pi R4S.
After starting system for the first time, OpenBSD installation program booted successfully, but we can see the following error in `dmesg`:

```
sdmmc0: can't enable card
```

This causes that when we proceed to disk selection (before partitioning), system does not list any disk. By the date of thesis submission deadline, author did not find any way how to solve this problem. However, he had created a new thread on unix.stackexchange.com.[2] Kind reader can track this issue via HTTP link in footnote.

The Author continued the installation by inserting a USB memory stick and installing the system on it.

[2] https://unix.stackexchange.com/questions/728292/openbsd-7-2-nanopi-r4s-sdmmc0-cant-enable-card

Chapter 8

Conclusion and Future work

We have demonstrated how to use OpenBSD along with other technologies to improve security in a computer network using multiple open-source technologies and tools.
We have created a robust Public Key Infrastructure that may be used in real-world scenario and brings trust to computer intercommunication, human-machine communication and human to human communication.
The only thing that is left up to reader is the manipulation with personal certificate. It was generated to support S/MIME email signing and document signing. It is not difficult to find a manual that describes how to import these certificates into individual application. The attachment of this thesis contains printscreens how to use Personal certificate to sign a document in LibreOffice Writer.

The approach described in this text may be deemed non-user-friendly. However, the way how the Author descibed the processes targets at maximum comprehensibility and does not aim to compete with out-of-the-box (and maybe *blackbox*) solutions.
Certificate generation and PKI manipulation done by manual OpenSSL calls is not a sustainable approach as well. Future work should cover the development of some wrapping script either using shell script or Makefile.
Since BSD is a complete operating system, not just kernel, a man coming from Linux may be surprised by the quality of OpenBSD's documentation and man pages.
On the other hand, user, that is not used to such a highly secured operating system, may be paralyzed initially when dealing with security features (like chroot).
Although there are some differences when compared to Linux, we can get inspired by plenty of guides and manuals for Linux distributions since the configuration of the services is the same by most.

Further work should include a detailed benchmark of Nano Pi R4S. Testing VPN throughput, benchmarking DNS server, FreeRADIUS authentications while measuring the use of system resources and the temperature of the case to find possible overheating.
Although the Author originally intended to include this benchmark in this text, the global chip outage together with problems with SD card did not give him a chance to manage it. The Author plans to continue with research in this direction and will publish the results later.

This thesis does not target ordinary users. This indeed does not relativize the fact that the Continual User Education is one of the most powerful security measures.
Users definitely must know at least the main principles of how the technology works. The applications and wizards are quite benevolent and let user to decide. Users must know how to act in that situation or who to ask for help. The Author is of the opinion that there is still a big gap between the scale of the usage of technologies among people and the knowledge about have about how to use them safely.
Our commitment is to search for solutions that may solve these problems or moderate them at least.

Appendix A

Attachment directory structure

`dhcpd/`	DHCP daemon configuration files directory
`diagrams/`	Diagrams created for the purposes of this thesis
`eap/`	EAP-TLS related configuration files, certificates and printscreens
`libreoffice/`	Demonstrative use of personal certificate shown on printscreens
`nanopi_r4s/`	Photos and `dmesg` command output of Nano Pi R4S
`nsd/`	Name Server Daemon configuration files and DNS zone files
`pf/`	System firewall configuration
`pki/`	Public Key Infrastructure
`unbound/`	Configuration of the unbound DNS server

Appendix B

OpenSSL configuration template

```
# OpenSSL <CA_NAME> config

# PKCS11 Engine
openssl_conf = openssl_def

[ ca ]
# ‘man ca‘
default_ca = CA_default

[ CA_default ]
# Directory and file locations.
dir               = /home/ca/pki/<CA_DIR>
certs             = $dir/certs
crl_dir           = $dir/crl
new_certs_dir     = $dir/newcerts
database          = $dir/index.txt
serial            = $dir/serial
RANDFILE          = $dir/private/.rand

# The root key and root certificate.
private_key       = $dir/private/ca.key.pem
certificate       = $dir/certs/ca.cert.pem

# For certificate revocation lists.
crlnumber         = $dir/crlnumber
crl               = $dir/crl/ca.crl.pem
crl_extensions    = v3_crl_ext
default_crl_days  = 30

default_md        = sha256

name_opt          = ca_default
cert_opt          = ca_default
default_days      = <DEFAULT_DAYS>
preserve          = no
policy            = <POLICY>

[ policy_strict ]
# The root CA should only sign issuing certificates that match.
# See the POLICY FORMAT section of ‘man ca‘.
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

```

```
[ policy_loose ]
# Allow the issuing CA to sign a more diverse range of certificates.
# See the POLICY FORMAT section of ‘man ca‘.
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[ req ]
# Options for the ‘req‘ tool (‘man req‘).
default_bits        = 2048
distinguished_name  = req_distinguished_name
utf8                = yes
string_mask         = utf8only
nameopt             = multiline,utf8

# SHA-1 is deprecated, so use SHA-2 instead.
default_md          = sha256

# Extension to add when the -x509 option is used.
x509_extensions     = v3_ca

[ req_distinguished_name ]
# See <https://en.wikipedia.org/wiki/Certificate_signing_request>.
countryName                     = Country Name (2 letter code)
stateOrProvinceName             = State or Province Name
localityName                    = Locality Name
0.organizationName              = Organization Name
organizationalUnitName          = Organizational Unit Name
commonName                      = Common Name
emailAddress                    = Email Address

# Optionally, specify some defaults.
countryName_default             = <DN_COUNTRY>
stateOrProvinceName_default     = <DN_STATE>
localityName_default            = <DN_LOCALITY>
0.organizationName_default      = <DN_ORG>
organizationalUnitName_default  =
emailAddress_default            =

[ v3_ca ]
# Extensions for a typical CA (‘man x509v3_config‘).
basicConstraints       = critical, CA:TRUE
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid:always,issuer
keyUsage               = critical, digitalSignature, cRLSign, keyCertSign

[ v3_issuing_ca ]
# Extensions for a typical issuing CA (‘man x509v3_config‘).
basicConstraints       = critical, CA:TRUE, pathlen:0
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid:always,issuer
keyUsage               = critical, digitalSignature, cRLSign, keyCertSign
crlDistributionPoints  = URI:<CRL_ROOT>
authorityInfoAccess    = OCSP;URI:<OCSP_ROOT>

[ v3_eaptls_server ]
basicConstraints       = CA:FALSE
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid:always,issuer:always
```

```
keyUsage                = nonRepudiation, digitalSignature, keyEncipherment
# TLS web serverAuth and
# EAP over LAN / WAN. See RFC 4334
extendedKeyUsage        = 1.3.6.1.5.5.7.3.1, 1.3.6.1.5.5.7.3.14
subjectAltName          = @v3_eaptls_server_alt_names
# RADIUS server certificate is issued by DV TLS CA
crlDistributionPoints   = URI:<CRL_DVTLS>
authorityInfoAccess     = OCSP;URI:<OCSP_DVTLS>

[ v3_eaptls_client ]
basicConstraints       = CA:FALSE
keyUsage               = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage       = 1.3.6.1.5.5.7.3.2
subjectAltName         = @v3_client_alt_names
crlDistributionPoints  = URI:<CRL_EAPTLS>
authorityInfoAccess    = OCSP;URI:<OCSP_EAPTLS>

[ v3_server_cert ]
# Extensions for server certificates (‘man x509v3_config‘).
basicConstraints        = CA:FALSE
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid,issuer:always
keyUsage                = critical, digitalSignature, keyEncipherment
extendedKeyUsage        = serverAuth
subjectAltName          = @v3_server_alt_names
crlDistributionPoints   = URI:<CRL_DVTLS>
authorityInfoAccess     = OCSP;URI:<OCSP_DVTLS>
nsCertType              = server
nsComment               = "OpenSSL Generated Server Certificate"

[ v3_client_cert ]
# Extensions for client certificates (‘man x509v3_config‘).
basicConstraints        = CA:FALSE
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid,issuer
keyUsage                = critical, nonRepudiation, digitalSignature,
    keyEncipherment
extendedKeyUsage        = clientAuth, emailProtection,
    1.3.6.1.4.1.311.10.3.12
subjectAltName          = @v3_client_alt_names
crlDistributionPoints   = URI:<CRL_PERSONAL>
authorityInfoAccess     = OCSP;URI:<OCSP_PERSONAL>
nsCertType              = client, email
nsComment               = "OpenSSL Generated Client Certificate"

[ v3_eaptls_server_alt_names ]
DNS.1                   = <RADIUS_DNS>

# NAIRealm from RFC 7585
otherName.0             = 1.3.6.1.5.5.7.8.8;FORMAT:UTF8,UTF8:<RADIUS_DNS>

[ v3_server_alt_names ]
DNS.1                   = server.foo.org
DNS.2                   = localhost
IP.1                    = 127.0.0.1
#IP.2                   = 192.168.1.100

[ v3_client_alt_names ]
email.1 = user@foo.org
#email.2 = secondary@foo.org

[ v3_crl_ext ]
# Extension for CRLs (‘man x509v3_config‘).
```

```
172 authorityKeyIdentifier = keyid:always
173
174 [ v3_ocsp ]
175 # Extension for OCSP signing certificates (‘man ocsp‘).
176 basicConstraints       = CA:FALSE
177 subjectKeyIdentifier   = hash
178 authorityKeyIdentifier = keyid,issuer
179 keyUsage               = critical, nonRepudiation, digitalSignature,
    keyEncipherment
180 extendedKeyUsage       = critical, OCSPSigning
181
182 # PKCS11 Engine
183 [ openssl_def ]
184 engines = engine_section
185
186 [ engine_section ]
187 pkcs11 = pkcs11_section
188
189 [ pkcs11_section ]
190 engine_id = pkcs11
191 #dynamic_path = /usr/local/lib/engines/pkcs11.so
192 MODULE_PATH = <OpenSC_LIB>
193 init = 0
```

openssl.cnf.templ

List of Tables

List of Figures

Acronyms

ASLR Address space layout randomization. 3

BSD Berkeley Software Distribution. 3

CA Certificate Authority. 12

CPU Central processing unit. 3

CRL Certificate revocation list. 12

CWE According to the Common Weakness Enumeration. 3

DNS Domain Name System. 2

DNSSEC Domain Name System Security Extensions. 2

DV domain validated. 14

EAP Extensible Authentication Protocol. 14, 48

eIDAS electronic IDentification, Authentication and trust Services. 16

EU European Union. 16

GND Ground. 67

GTK Group Transient Key. 49

GUI Graphical User Interface. 1, 2

HaaS Honeypot as a Service. 2

HDD Hard disk drive. 15

HSM Hardware Security Module. 15, 16

HTTP Hypertext Transfer Protocol. 12

IEEE Institute of Electrical and Electronics Engineers. 47, 48

IETF Internet Engineering Task Force. 8, 48

IP Internet Protocol (RFC 791). 5

MMC MultiMediaCard. 15

OCSP Online Certificate Status Protocol. 12

PIN Personal identification number. 15

PK private key. 12

PKI Public Key Infrastructure. 12, 18

PTK Pair-wise Transient Key. 49

QSCD Qualified Signature Creation Device. 16

RA Registration Authority. 12

RADIUS Remote Authentication Dial-In User Service. 14

RAM Random Access Memory. 1

RFC Request for Comments. 8, 48

SMP Symmetric Multiprocessor kernel mode. 4

SOHO Small Office/Home Office. 1

SSD solid-state drive. 15

SSH Secure Shell. 1, 2

TLS Transport Layer Security. 14

TSA Time-Stamp Authority. 12

USB Universal Serial Bus. 15, 16, 67

VA Validation Authority. 12

VAT Value-added tax. 16

WAN Wide Area Network. 2

Wi-Fi . 2, 14, 47

WPA Wi-Fi Protected Access. 47

Bibliography

[1] OpenWrt Project. "General requirements for openwrt support." (), [Online]. Available: https://openwrt.org/supported_devices#general_requirements_for_openwrt_support.

[2] Turris project. "Turris routers." (2023), [Online]. Available: https://docs.turris.cz/basics/models/.

[3] The OpenBSD Foundation. "Openbsd project goals." (2023), [Online]. Available: https://www.openbsd.org/goals.html.

[4] The OpenBSD Foundation. "Funding for openbsd and related projects." (2023), [Online]. Available: https://www.openbsdfoundation.org/index.html.

[5] Common Weakness Enumeration. "Cwe top 25 most dangerous software weaknesses." (2021), [Online]. Available: https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html.

[6] Red Hat, Inc., "Efficient string copying and concatenation in c," [Online]. Available: https://developers.redhat.com/blog/2019/08/12/efficient-string-copying-and-concatenation-in-c#.

[7] The OpenBSD Foundation. "Openbsd armv7." (2022), [Online]. Available: https://www.openbsd.org/armv7.html.

[8] J. Postel, "Dod standard internet protocol," RFC, Jan. 1980. DOI: 10.17487/RFC0791. [Online]. Available: https://www.rfc-editor.org/info/rfc791.

[9] Internet Engineering Task Force. "Pervasive monitoring is an attack." (2014), [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7258.

[10] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," 1978. [Online]. Available: https://doi.org/10.1145/359340.359342.

[11] W. S. Jevons, *The Principles of Science: A Treatise on Logic and Scientific Method.* Macmillan & Co., London, 1874.

[12] N. Naziridis. "Comparing ecdsa vs rsa." (Jun. 27, 2018), [Online]. Available: https://www.ssl.com/article/comparing-ecdsa-vs-rsa/.

[13] Synopsys, Inc. "The heartbleed bug." (2014), [Online]. Available: https://heartbleed.com/.

[14] GÉANT Association. "Eap server certificate considerations." (2021), [Online]. Available: https://wiki.geant.org/display/H2eduroam/EAP+Server+Certificate+considerations.

[15] QSCD.eu. "Eidas usb token gemalto safenet etoken 5110 cc (940)." (2023), [Online]. Available: https://www.qscd.eu/eidas-usb-tokens/gemalto-safenet-etoken-5110-cc/.

[16] J. Nguyen. "Openssl certificate authority." (), [Online]. Available: https://jamielinux.com/docs/openssl-certificate-authority/.

[17] OpenVPN, Inc. "Openvpn inc." (2023), [Online]. Available: https://github.com/OpenVPN.

[18] FreeRADIUS project. "Freeradius - a multi-protocol policy server." (2023), [Online]. Available: https://github.com/FreeRADIUS/freeradius-server.

[19] FreeRADIUS. "Certificate compatibility." (Dec. 30, 2020), [Online]. Available: https://wiki.freeradius.org/guide/certificate-compatibility.

[20] The OpenBSD Foundation. "Sysmerge(8) - openbsd manual pages." (2016), [Online]. Available: https://man.openbsd.org/sysmerge.

[21] The OpenBSD Foundation. "Computer networks: A systems approach." (), [Online]. Available: https://www.openbsd.org/faq/pf/example1.html.

[22] J. Geoghegan. "Pf-badhost - stop the evil doers in their tracks!" (Jan. 10, 2021), [Online]. Available: https://www.geoghegan.ca/pfbadhost.html.

[23] The OpenSSL Project Authors. "Openssl - online certificate status protocol utility." (Dec. 21, 2021), [Online]. Available: https://www.openssl.org/docs/man1.1.1/man1/ocsp.html.

[24] Stichting NLnet Labs. "Certificate compatibility." (2023), [Online]. Available: https://www.nlnetlabs.nl/.

[25] J. Geoghegan. "Unbound-adblock - the ultimate dns firewall!" (Jan. 10, 2021), [Online]. Available: https://www.geoghegan.ca/unbound-adblock.html.

[26] J. Geoghegan. "Unbound-adblock openbsd installation instructions." (Jan. 10, 2021), [Online]. Available: https://www.geoghegan.ca/pub/unbound-adblock/latest/install/openbsd.txt.

[27] The OpenBSD Foundation. "Openbsd faq - introduction to openbsd." (2023), [Online]. Available: https://www.openbsd.org/faq/faq1.html.

[28] The OpenBSD Foundation. "Errata and patches." (2023), [Online]. Available: https://www.openbsd.org/errata.html.

[29] The OpenBSD Foundation. "Upgrade guide: 7.1 to 7.2." (2023), [Online]. Available: https://www.openbsd.org/faq/upgrade72.html.

[30] IoT Analytics GmbH. "State of iot 2022: Number of connected iot devices growing 18% to 14.4 billion globally." (May 18, 2022), [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/.

[31] Wi-Fi Alliance. "Who we are - our brands." (2023), [Online]. Available: https://www.wi-fi.org/who-we-are/our-brands.

[32] IEEE, "Ieee standard for local and metropolitan area networks–port-based network access control," *IEEE Std 802.1X-2020 (Revision of IEEE Std 802.1X-2010 Incorporating IEEE Std 802.1Xbx-2014 and IEEE Std 802.1Xck-2018)*, pp. 1–289, 2020. DOI: 10.1109/IEEESTD.2020.9018454.

[33] Intel Corporation. "802.1x overview and eap types." (Oct. 28, 2021), [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000006999/wireless/legacy-intel-wireless-products.html.

[34] E. Raphaely. "802.1x eap-tls authentication flow explained." (2023), [Online]. Available: https://www.securew2.com/blog/802-1x-eap-tls-authentication-flow-explained.

[35] A. Alabdulatif1 and X. Ma, "802.1x eap-tls authentication flow explained," *International Journal for Information Security Research*, vol. 4, 2 Jun. 2014. [Online]. Available: https://infonomics-society.org/wp-content/uploads/ijisr/published-papers/volume-4-2014/Analysing-the-EAP-TLS-Handshake-and-the-4-Way-Handshake-of-the-802-11i-Standard.pdf.

[36] SIA Mikrotīkls. "Manual:quickset." (May 23, 2022), [Online]. Available: https://wiki.mikrotik.com/index.php?title=Manual:Quickset&oldid=34546.

[37] SIA Mikrotīkls. "Manual:wireless eap-tls using routeros with freeradius." (May 19, 2019), [Online]. Available: https://wiki.mikrotik.com/index.php?title=Manual:Wireless_EAP-TLS_using_RouterOS_with_FreeRADIUS&oldid=33254.

[38] J. Malinen. "Wpa2-eap/ccmp using eap-tls." (2022), [Online]. Available: https://w1.fi/cgit/hostap/plain/wpa_supplicant/examples/wpa2-eap-ccmp.conf.

[39] C. Singh. "802.1x eap-tls authentication flow explained." (Dec. 26, 2018), [Online]. Available: https://fossbytes.com/stock-android-vs-android-one-vs-android-go-best/.

[40] Microsoft Corporation. "Certmgr.exe (certificate manager tool)." (Sep. 15, 2021), [Online]. Available: https://learn.microsoft.com/en-us/dotnet/framework/tools/certmgr-exe-certificate-manager-tool.

[41] Microsoft Corporation. "Distribute certificates to client computers by using group policy." (Jun. 18, 2021), [Online]. Available: https://learn.microsoft.com/en-us/windows-server/identity/ad-fs/deployment/distribute-certificates-to-client-computers-by-using-group-policy.

[42] Armbian. "Download." (2023), [Online]. Available: https://www.armbian.com/download/.

[43] JPMorgan Chase & Co. "Supply chain issues and autos: When will the chip shortage end?" (Aug. 11, 2022), [Online]. Available: https://www.jpmorgan.com/insights/research/supply-chain-chip-shortage.

[44] Shenzhen Xunlong Software Co., Ltd. "Orange pi r1 plus lts." (2023), [Online]. Available: http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/orange-pi-R1-Plus-LTS.html.

[45] FriendlyELEC. "Nanopi r4s." (2023), [Online]. Available: https://www.friendlyelec.com/index.php?route=product/product&product_id=284.

[46] Nextcloud GmbH. "Nextcloud system requirements." (2023), [Online]. Available: https://docs.nextcloud.com/server/latest/admin_manual/installation/system_requirements.html.

[47] S. Pate, *UNIX Filesystems: Evolution, Design, and Implementation*. Robert Ipsen, 2003, ISBN: 0-471-16483-6.

[48] M. W. Lucas. "Absolute openbsd, 2nd edition." (2013), [Online]. Available: https://nostarch.com/obenbsd2e.