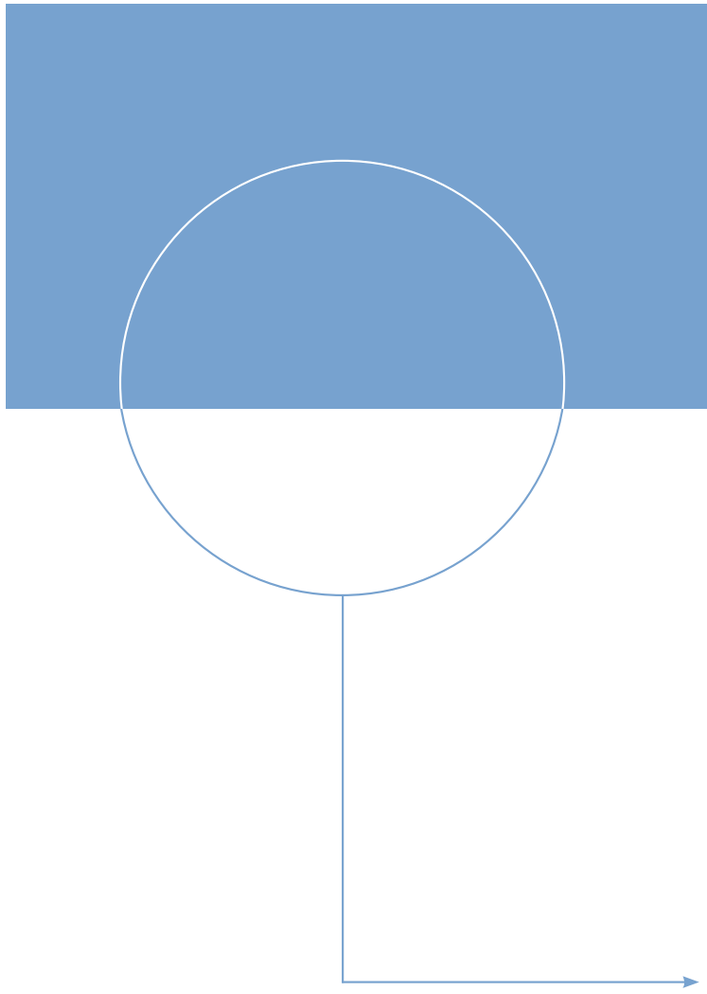

Master's thesis

2019

Sigurd Haaheim

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Information Security and Communication
Technology

Sigurd Haaheim

AC-PKI

Application-centric public-key infrastructures

June 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

AC-PKI

Application-centric public-key infrastructures

Application-centric public-key infrastructures

Information Security
Submission date: June 2019
Supervisor: Håkon Gunleifsen
Co-supervisor: Ragnar Nicolaysen
Svein-Tore Omdahl

Norwegian University of Science and Technology
Department of Information Security and Communication
Technology

Contents

Preface

This Master of Science (M.Sc.) thesis was written during the spring of 2019 for the Department of Information Security and Communication Technology at the Norwegian University of Science and Technology (NTNU) in Gjøvik, Norway. The thesis was written in collaboration with NorgesGruppen Data AS, who proposed the thesis problem upon my request during a summer internship in 2018.

The technical nature of this thesis makes it best suited for readers with at least some knowledge about computer networks and cryptography. However, the thesis is written in such a way that knowledge is required about the specific tools and frameworks used.

Acknowledgement

First, I would like to thank Ragnar Nicolaysen and Svein-Tore Omdahl from NorgesGruppen Data for suggesting the topic of this master thesis. Working with software-defined networks and Cisco ACI has been exciting and inspiring, and both are very relevant in networking today. I would also like to thank them for supervising me throughout the last year, for providing feedback and answering my questions. Without their assistance and feedback, this thesis would not have been possible.

I would also like to thank my academic supervisor, PhD candidate Håkon Gunleifsen at NTNU, for having regular supervisor meetings and providing feedback on this thesis. His positive feedback has been great motivation, and critical feedback has caused significant improvements. Finally, I want to thank Prof. Slobodan Petrovic, for helping me get started and find the right academic supervisor for my thesis.

Sigurd Haaheim

Abstract

Software-defined networking has been around for at least two decades, but been popularised in recent years by the increased complexity in computer networks and data centres. Trends like the "Internet of things" and cloud computing significantly increase the number of devices connected to some networks and data centres. Cisco ACI (Application-Centric Infrastructure) is Cisco's approach to SDN, and also includes a comprehensive policy framework.

When it comes to protecting traffic that flows through the software-defined network (the data plane), there are many unanswered questions. In this thesis, we propose a method and architecture that utilise public-key cryptography and the policies in Cisco ACI to automatically and seamlessly issue certificates to endpoints which are allowed to communicate. Each certificate pair is only valid between two endpoints and is revoked if the connection is no longer allowed. The certificates can be used to establish secure connections with public-key based protocols like TLS and state-of-the-art encryption algorithms.

The solution proposed in this thesis, Application-centric public-key infrastructures (AC-PKI), virtually eliminates the need for manual configuration of public-key infrastructures and significantly improves the network security. The solution is scalable and can be adjusted to work with other policy frameworks than Cisco ACI. We have developed a prototype that uses OpenSSL and the principles of AC-PKI to establish a TLS connection between two endpoints, only if they are allowed to communicate according to Cisco ACI. The prototype confirms the feasibility of our solution.

Sammendrag

The English summary is on the next page.

Programvaredefinerte nettverk (SDN, software-defined networks) skal gjøre komplekse nettverk smidige, skalerbare og lettere å vedlikeholde. Selv om de underliggende idéene har eksistert i minst tjue år, er det først de siste årene at SDN har fått stor oppmerksomhet på grunn av stadig mer komplekse nettverk og datasentre. Cisco ACI (applikasjonssentrisk infrastruktur) er Ciscos tilnærming til programvaredefinerte nettverk, og har samtidig bred støtte for tilgangskontroll. Når det gjelder sikkerheten rundt data som går gjennom nettverket (dataplansikkerhet) er det riktignok flere ubesvarte spørsmål.

Denne masteroppgaven har som formål å beskytte dataplanet i Cisco ACI ved hjelp av asymmetrisk kryptografi og digitale sertifikater. Ved å benytte tilgangskontrollen i Cisco ACI vil vi bedre sikkerheten både med hensyn til konfidensialitet og integritet for nettverkstrafikken, samtidig som vi integrerer konfigurasjonen av nettverks- og sikkerhetsarkitekturen i samme prosess. På denne måten vil nettverkene bli sikrere, mer oversiktlige og lettere å vedlikeholde.

Det ble først gjennomført en litteraturstudie og gjennomgang av relevant teori og dokumentasjon. Vi så også på alternative metoder for å oppnå sikker kommunikasjon på dataplanet. Vi satte opp en oversikt over muligheter, begrensninger og krav til løsningen, utviklet en metode og arkitektur, og implementerte en prototype for å demonstrere løsningen. Til slutt evaluerte vi løsningen, diskuterte resultatene og implikasjonene det har for nettverks- og dataplansikkerhet.

Ved å hente tilgangsdata fra Cisco ACI klarte vi å tildele sertifikater til de enhetene som har lov til å kommunisere med hverandre. Løsningen setter opp en sikker og effektiv forbindelse mellom en klient og en tjener, og kommuniserer effektivt med Cisco ACI ved å abonnere på endringer i relevante nettverksdata. Prototypen bekreftet at løsningen fungerer i praksis, selv om den er primitiv og lagt ifra en storskala produksjonsløsning.

Som et resultat av bakgrunn, teori og resultater som beskrevet ovenfor, konkluderer vi med at løsningen vår har et potensiale for å tilby sikker dataplankommunikasjon i Cisco ACI-nettverk. Løsningen tilbyr enklere, sikrere og mer effektiv håndtering av digitale sertifikater enn dagens alternativer. Det er riktignok flere spørsmål som må besvares før en produksjonsløsning kan implementeres.

Summary

Software-defined networking (SDN) has gained popularity in recent years as a result of the increased complexity of networks and data centres. By moving control of how traffic flows through the network to centralised controllers, SDNs aim to make networks more flexible, agile and scalable. Although the underlying ideas have been around for at least two decades, there is little work dedicated to securing the traffic that flows through SDNs (the data plane). Cisco Application-Centric Infrastructure (ACI) is Cisco's approach to SDN and includes a comprehensive policy framework.

This master's thesis aims to secure the data plane of a Cisco ACI network using a public-key infrastructure (PKI). Using policies from Cisco ACI, we generate peer-specific certificates that are only issued to pairs of endpoints with explicit permission to communicate (a contract). Doing so, we strengthen policy enforcement, validate peer identities and encrypt traffic using state-of-the-art encryption algorithms. The solution virtually eliminates the need for manual PKI configuration and significantly improves security compared to current alternatives.

First, relevant research and alternative methods were researched and evaluated. Second, we studied relevant theory and existing technologies and frameworks, and defined the constraints, requirements and superficial architecture of the solution. Third, we developed a methodology and architecture for the solution and implemented a prototype as a proof-of-concept. Finally, we discussed the results and their implications for data plane security and concluded the thesis.

Requesting data from the Cisco ACI API, we were able to process it and use it to validate whether two endpoints are permitted to communicate. The design provides efficient communication with Cisco ACI by subscribing to any changes in relevant data, automated certificate validation and revocation and simple yet secure endpoint-to-endpoint security.

Based on the background research, theory, solution and prototype results, we consider AC-PKI to have great potential in making networks more secure and efficient. However, several questions must be answered before a production system could be implemented. These questions are presented in chapter 10 "Future work".

List of Figures

List of Tables

Listings

1 Introduction

1.1 Background

The Internet of things, cloud computing and other trends have caused an increase in the complexity of networks and data centres in recent years, making traditional networks hard to manage and maintain [1, 2]. Software-defined networking (SDN) aims to solve these issues and make networks agile, scalable and flexible, by moving the control to centralised controllers, essentially making the networks programmable [3].

When it comes to network security, however, there are many unanswered questions [4] and SDN can even introduce new security problems [2]. It is possible to apply conventional security techniques to SDN, but that would result in a compromise of the agility, scalability and flexibility that the networks aim to provide. Figure 1 illustrates how an untrusted third party could eavesdrop communication on the SDN data plane. This issue could be prevented by encrypting the data plane traffic between the client and the server.

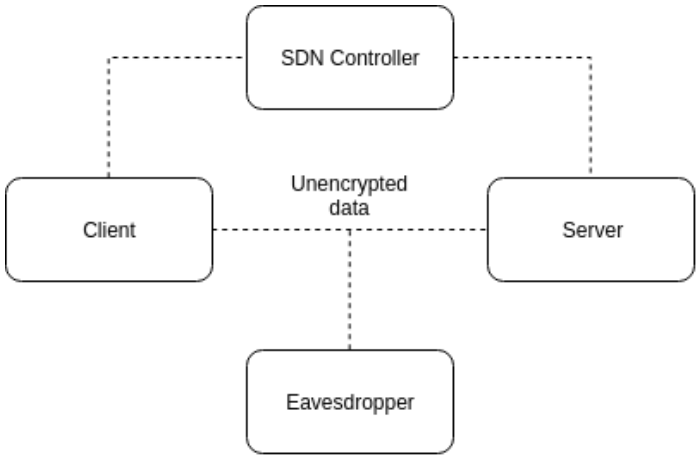

Figure 1: Simple illustration of eavesdropping in an unprotected data plane.

In this Master of Science (M.Sc.) thesis, we propose a method that provides state-of-the-art security while maintaining the named advantageous characteristics of SDNs. Specifically, we focus on Cisco Application-Centric Infrastructure (ACI), which is Cisco's approach to SDN. The solution adds little configuration overhead, is easy to understand both for network administrators and software developers, and is just as flexible, scalable and agile as the SDN itself.

1.2 Keywords

Information security, cryptography, public key, software-defined networking, networking, Cisco ACI, OpenSSL, data plane security, network security

1.3 Problem description

Cisco ACI, as the name suggests, is centred around applications rather than physical devices. It also provides a comprehensive policy framework to limit communication between devices that do not need to communicate. Policies are applied to groups of endpoints, called endpoint groups (EPGs). Some traffic isolation is provided by default, meaning that only the intended recipient can see the traffic [5]. However, it is likely that many companies want to add additional security measures to ensure the confidentiality and integrity of the network traffic. This is especially true for companies who face strong competition and the risk of industrial espionage.

Implementing a public-key infrastructure (PKI) is a popular way in which to achieve secure network communication. Confidentiality and integrity are assured by public-key cryptography and digital certificates, respectively. However, manually maintaining PKIs can be a tedious task, especially for large and complex company networks, and human errors could compromise security. By implementing a manual PKI in an SDN, the agility, scalability and flexibility provided by the network would be neglected by the PKI. It is better to issue certificates automatically and dynamically based on the policies that already exist in Cisco ACI.

1.4 Justification, motivation and benefits

Today, network policy, access control and PKIs are typically managed using solutions like Aruba ClearPass [6] or DigiCert [7]. These systems allow network administrators to manage the different types of users and devices that connect to the network, their privileges and access to various network resources, as well as the security requirements for each of these connections.

However, because Cisco ACI integrates policy into the network itself, using dedicated PKI solutions would essentially mean that network managers have to enter the same information twice. That could cause inconsistencies between the policies in Cisco ACI and those in the PKI management system. We solve this problem by using the policies in Cisco ACI to only issue certificates to devices which are allowed to communicate. The certificates are peer-specific, meaning that they only work between a specific pair of endpoints.

There are several benefits to a system like this. It would significantly decrease the time spent on security management and manual PKI configuration. Due to the decrease in manual configuration, the likelihood of misconfiguration and human errors would decrease, ultimately improving security and reducing the network downtime. Improved efficiency and security reduce costs and the likelihood of significant losses, which would benefit the company as a whole and its owners (shareholders).

This thesis presents such a system, making the security and PKI framework as flexible, scalable and agile as the network itself. As soon as an endpoint is assigned to an EPG, it should be ready to

access the PKI and receive the certificates needed for secure communication with other endpoints on the company network. Furthermore, if a policy is removed from Cisco ACI, its corresponding certificates should be revoked.

1.5 Research questions

To solve the problem described in section 1.3, there are several questions that need to be answered. The primary research questions are listed in this section.

First, we need to determine whether there are any existing or alternative solutions to the said problem. Second, we need to find a design for a PKI which reflects the architecture of the Cisco ACI network itself. This includes finding a logical model between the network and the PKI so that data can be automatically mapped and transferred between the two. Third, we need to find a way to efficiently synchronise the PKI and its certificates so that it reflects the current state of the Cisco ACI network. Finally, we need to find a method in which these certificates can be securely, dynamically and efficiently distributed to endpoints in order to communicate securely.

We have defined the main research questions as below:

1. What alternatives to PKI exist for SDN data plane security?
2. Can an OpenSSL PKI be designed so that it reflects the Cisco ACI network architecture?
3. How can the PKI be synchronised to mirror changes in the network architecture?
4. How can keys and certificates be dynamically and seamlessly issued only to endpoints that are allowed to communicate?

The first question will be answered in the literature review in chapter 3 while the others will be answered when discussing the solution in chapter 6. The research questions will be revisited at the end of the thesis, in chapter 8, to evaluate whether the research questions have been satisfactorily answered.

In order to structure this work, we applied the Design Science Research Methodology (DSRM) defined by Peffers, Tuunanen, Rothenberger & Chatterjee (2007) [8]. The methodology will be described in the next chapter.

1.6 Contributions

The primary contribution of this thesis is the architecture and design for application-centric public-key infrastructures (AC-PKI). That includes the methods for registering, issuing and validating digital certificates. The solution is generic and can be used for several PKI based security systems, such as establishing a TLS or MACsec connection or a secure IPsec tunnel. We have also conducted a solution analysis, which discusses the objectives, constraints, requirements and high-level architecture of the solution.

The solution builds on top of existing technologies like SDN, Cisco ACI and OpenSSL. By using policy data from the Cisco ACI controller, the solution processes and stores the data that is needed to issue certificates to only those endpoints which are allowed to communicate. Certificates can be

issued directly from AC-PKI or through a dedicated registration authority (RA). As far as we have found, there are currently no solutions that offer automatic certificate distribution based on Cisco ACI policies, nor any solutions that provide a dynamic and seamless PKI specifically designed for SDN.

Figure 2 illustrates how AC-PKI requests endpoint and policy data from Cisco ACI, to generate a PKI that reflects the architecture in Cisco ACI. The RA – the entity responsible for issuing new certificates – can use the PKI generated by AC-PKI to issue certificates only to pairs of endpoints which are allowed to communicate. A certificate is only valid for connections between two endpoints, to ensure compliance with the policies in Cisco ACI.

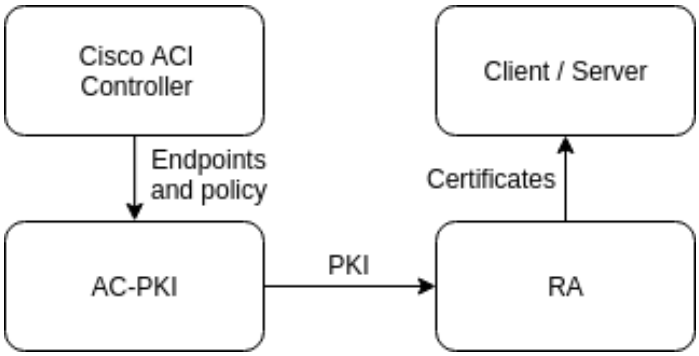

Figure 2: Simple illustration of the AC-PKI data flow.

We developed a prototype as a proof-of-concept. The prototype aims to prove that the goals presented in this chapter are indeed possible. The prototype should not be considered one of the primary contributions of this thesis, but instead a practical example of how AC-PKI would work in practice. The implementation is not designed to be secure or form the basis for any production system. The prototype includes a simple Client and Server, the latter of which merely returns any message it receives from the former, over a secure TLS connection.

The main contributions are:

1. Analysis and objectives for the solution
2. Solution architecture and design
3. Proof-of-concept prototype implementation
4. Validation and evaluation of the results

1.7 Thesis outline

Chapter 2, “Methodology”, presents the methodologies used for the thesis in detail. Chapter 3, “Literature review”, presents previous work and relevant sources, and chapter 4 discusses the theoretical principles on which the rest of this thesis is based. Chapter 5, “Solution objectives”, presents an initial analysis of objectives, constraints, requirements and limitations for the AC-PKI solution.

Chapter 6, “Solution”, presents the proposed solution and a brief overview of the prototype that was developed as a proof-of-concept.

Chapter 7 presents the results, while chapter 8 discusses the implications of the results and evaluates the solution as a whole. Finally, chapter 9 concludes the thesis and chapter 10 presents future work to be done within the topic of AC-PKI.

Most of the abbreviations used in this thesis are listed in Appendix A and the prototype is described in more technical detail in Appendix B.

2 Methodology

This chapter briefly presents the research methodology, work process and tools used throughout the thesis. The research methodology is presented in section 2.1, each stage of our work is presented in section 2.2 and the tools and frameworks used are discussed in section 2.3.

2.1 Research methodology

We applied the Design Science Research Methodology (DSRM) defined by Peffers, Tuunanen, Rothenberger and Chatterjee in 2007 [8]. The methodology includes a six-step process. Each step of the methodology, and the chapters in which they are discussed, is described in table 1.

#	**Step**	**Chapters**
1	Problem identification	1
2	Solution objectives	5
3	Design and development	6
4	Demonstration	6
5	Evaluation	8
6	Discussion	8

Table 1: The steps in DSRM and the chapters in which each step is included.

Furthermore, our research can be categorised as “constructive research” because it builds upon existing work within several topics [9], including SDN, Cisco ACI, public-key cryptography, OpenSSL and Promise Theory.

2.2 Research stages

This section presents each stage of our research and the methodology used within that stage.

2.2.1 Literature review and theory

During the literature review, we studied the technologies that are relevant to this thesis as well as alternative solutions to SDN security, as well as public-key infrastructure (PKI) based network security in general. We also studied Cisco ACI and OpenSSL specifically. The literature review refers to relevant scientific work and documentation for the technologies, protocols and frameworks used later in the thesis.

The theory chapter presents some of the topics introduced in the literature review in more detail, giving a high-level overview of relevant technologies, protocols and frameworks which are used later in the thesis.

2.2.2 Solution objectives

This stage involved defining objectives for the solution, based on the problem description, literature review and theory from the previous stages. During this stage, the solution was separated into main components, and possibilities, constraints and requirements were defined for each component.

The analysis was based on the input data (Cisco ACI policies) and the output data (certificates to the PKI). We did not focus on technical details or implementation of the system, but merely on what processing that was required and which tasks had to be done to obtain the correct output. The high-level architecture that was designed during this stage was a significant part of the thesis result.

2.2.3 Experimental

During the experimental stage, we developed both the principles and low-level architecture of AC-PKI and a prototype that implemented these in practice. The process in the experimental stage was iterative and shifted between the theoretical solution and prototype development. This emphasises the agility of our methodology, as the results and discoveries from the prototype development have shaped and improved the solution as a whole. Likewise, improvements discovered during theoretical work have been implemented in the prototype.

2.2.4 Validation and evaluation

The last stage involved validating and evaluating the discoveries from the experimental stage. That involved determining whether the solution solved the problem described in section 1.3 and answered the research questions posed in chapter 1.5. We also discussed and evaluated the overall impact of the solution, particularly concerning PKI management and network security, as well as potential issues and remaining work within the topic of AC-PKI.

2.3 Tools and frameworks

This section presents the tools and libraries that were used during the experimental stage. The tools were used not only for developing the prototype, but also for exploring the Sandbox APIC, OpenSSL and more.

2.3.1 Cisco learning labs and sandbox

Cisco Learning Labs provides several lessons related to Cisco ACI [10, 11], which are very helpful for beginners. They also provide a virtual sandbox environment, which can be used to get familiar with Cisco ACI without having to buy expensive hardware or software licences. Both the Learning Labs and the sandbox is discussed in more detail later in this thesis.

2.3.2 Programming language and operating systems

For the experimental stage, we used Python 2.7 because of its extensive documentation and support for third-party libraries. The Cisco Learning Labs also uses Python 2.7 in its examples. We primarily used the Ubuntu 18.04 long-term support (LTS) Linux operating system (OS). This OS has comprehensive online support, is compatible with a large variety of tools and comes with the terminal

tools for OpenSSL pre-installed.

2.3.3 OpenSSL

OpenSSL provides all the tools needed to generate, verify, sign and revoke digital certificates, as well as tools for setting up TLS connections for testing purposes [12, 13]. The command-line version of OpenSSL is primarily intended for testing and education and has several known bugs and other issues. This is especially true for the OpenSSL "ca" command, which according to the documentation is "quirky and at times downright unfriendly" [12]. Despite the limitations of the command-line tools, it is well suited for experimentation with simple public-key and certificate operations such as certificate inspection and validation, or generating a certificate for testing.

2.3.4 Python networking libraries

The external "requests" library provides HTTP and HTTPS support for Python, which is easy to understand, or "safe for human consumption" as they describe it [14]. We used this library for communication with the Cisco APIC.

The built-in "socket" library, is a low-level networking library [15], which enabled us to send simple strings or byte streams between two hosts using their IP address and port number. We use this library together with the PyOpenSSL library (presented in the next section) to establish a TLS connection between the client and the server.

"Websockets" [16] is a Python library that enables WebSocket subscriptions as defined by RFC 6455 [17]. The library enabled us to receive data from Cisco ACI as soon as they occurred instead of checking for any new information at a regular interval.

2.3.5 Python OpenSSL library

The Python OpenSSL library (PyOpenSSL; `https://pyopenssl.org`) provides a wide range of functions for common OpenSSL operations. It enabled us to generate certificate signing requests (CSRs), public and private keys, digital certificates, as well as signing and revoking certificates – all within Python. One drawback with PyOpenSSL is that it does not support the latest TLS version 1.3, at the time of this writing. However, as we are only developing a proof-of-concept, and TLS 1.2 is still considered a secure protocol, we did not consider this significant.

The PyOpenSSL library also enabled us to implement a functional CA and RA, capable of validating and issuing certificates, respectively. Although implementing such functionality ourselves is generally less secure than using dedicated third-party software, we consider it worthwhile for this prototype, as it provides great flexibility and enable us to customise the solution in detail.

For a production system, it might be a good idea to use a built-in library like "crypto", as they are generally considered more secure than third-party libraries like PyOpenSSL. However, as PyOpenSSL appears to be slightly more user-friendly, that was our choice for this prototype.

2.3.6 Postman API tool

Postman (`https://www.getpostman.com/`) is an essential tool for anyone working with APIs. It is available as an extension for the Google Chrome web browser or a native application for the Linux,

MacOS and Windows operating systems. The application enables developers to send HTTP requests (including GET, POST, PUT and DELETE) to the API for testing purposes. The responses can easily be displayed within the application, in the case of Cisco ACI represented as XML or JSON. Although there is a paid version, the free version was more than capable enough for our use.

3 Literature review

Limited scientific literature was found related to public-key infrastructures (PKIs) designed specifically for use in software-defined networking (SDN) or Cisco ACI. This confirms that SDN security still is a topic with more questions than answers. The following sections present the related work on which this thesis builds.

3.1 Promise Theory

The concept of Promise Theory was first proposed (in the context of computer science) by Mark Burgess in 2005 [18]. Bergstra and Burgess discuss the topic in more detail in "Promise Theory: Principles and Applications" [19]. Promise theory is based on *autonomous agents* that cooperate voluntarily, only based on their internal policy. Hence, it not possible to force an autonomous agent to do something, only to make requests for a promise to do something. For instance an agent could forward packets internally on a corporate network, but refuse to forward packets to the Internet.

The general assumption in Promise Theory is that software is unreliable and will occasionally fail, and no piece of software should, therefore, have hard dependencies on just a single external resource. Promise Theory is discussed in the context of computer networks in Borrill et al. [20]. As we will see later in this chapter, the modern interpretation of SDN is closely related to the principles of Promise Theory. These are also the underlying principles of Cisco ACI, as we will discuss in section 3.3. CFEngine is a configuration management framework that closely utilises and complies with Promise Theory [21, 22], as we will discuss in more detail in section 4.1.6.

Promise Theory is not limited to computer networks and Cisco ACI. In their book [19], Bergstra and Burgess use examples from manufacturing, transportation, society and every-day life.

3.2 SDN and security

Because SDN is an essential part of this thesis, this section presents some current research on the topic. We also discuss network function virtualisation (NFV), a concept which is closely related to SDN, as well as some security considerations and alternative approaches to SDN security.

3.2.1 Software-defined networking

SDN is all about separating the control and data planes of the network, ultimately centralising control of how traffic flows through the network [2]. By centralising network control, the controllers get a global view of the network and can make decisions based on that view. This way, the network is more agile, scalable, flexible and easy to manage than traditional "network-centric" networks [3].

Feamster et al. [1] have studied the evolution that led to the separation of the control and data planes in networks – what we now call SDN. Although it may seem like SDN is a concept that has evolved during the last few years, they track the underlying ideas back as far as the early

telephone networks and at least 20 years back for computer networks. However, the term SDN did not appear until an article about the OpenFlow project [23] at Stanford in 2009 [2]. The reason SDN is discussed so frequently now is because the complexity of traditional networks and data centres is outpacing our ability to manage them efficiently [1, 2].

3.2.2 Network function virtualisation

Network function virtualisation (NFV) is a concept which is closely related to SDN. Using the principles of cloud computing and virtualisation, in NFV network functions (NFs) that were typically implemented as "middleboxes" (i.e. hardware appliances in the data path) are moved to centralised virtual services, typically running on virtual machines (VMs) [24]. Some examples of NFs are firewalls, intrusion prevention systems (IPS), network address translation (NAT), deep packet inspection (DPI) and network monitoring. Moving NFs to centralised servers make them more flexible, scalable and agile, just as SDNs aim to do with the network itself, explaining the close relation between the two concepts.

Because virtual network functions (VNFs) are not, unlike middleboxes, physically placed within the path of the network traffic, the traffic must take a detour to reach the devices on which the VNFs run. That, of course, is easily accomplished in an SDN by modifying the forwarding rules. Also, it is easy to create filters that specify which types of traffic that need to enter a specific VNF. For instance, all external traffic (i.e. traffic from outside the company network, typically Internet traffic) could be directed through an IPS while internal traffic is not.

3.2.3 Software-defined network security

The security considerations of SDN are typically divided into control and data plane security [25]. Control plane security concerns securing the SDN infrastructure itself, including the controllers and routing-related traffic. Data plane security relates to securing the network traffic that flows through the network, such as web traffic, emails, cloud services and all other types of Internet or intranet traffic that involves transferring data between two endpoints.

From the previous section, we know that VNFs are often used to improve the security of the network. The general perception is that SDNs combined with NFV will ultimately improve network security [25]. However, there are unanswered questions within both aspects of SDN security [4].

3.2.4 Alternative approaches to data plane security

Vajaranta et al. [26] proposed a method for applying Internet Protocol security (IPsec) [27] as an Internet layer alternative for SDN security. The paper particularly focuses on OpenFlow by the Open Networking Foundation (ONF, `https://opennetworking.org`). Their proposal involved an IPsec appliance connected to the SDN switch handling both Internet Key Exchange (IKE) [28] negotiations and IPsec functionality, or multiple IPsec cryptographic appliances as well as one IKE appliance. Their solution runs on the Internet layer, unlike TLS which runs on the application layer, making it "conform nicely to the SDN paradigm" [26]. When discussing networking layers in this thesis, we refer to the IP model unless something else is specified. For a comparison of the IP and OSI network models, see table 4 on page 33.

There are two main approaches to implementing IPsec in a company network. One way is to use an IPsec tunnel from the server and all the way to the client. This is one of the most common ways in which to implement a virtual private network (VPN) [29]. VPNs enable employees to access network resources that are normally limited to devices connected to the company network, even when they are travelling or working from home. It also mitigates many of the risks typically related to connecting to public networks, such as man-in-the-middle attacks and packet capturing. The other way is to have an IPsec tunnel between a local router (near the client) and the data centre. This method is, within the realm of SDN, a software-defined wide area network (SD-WAN), which enables branch offices to connect securely to the headquarter or data centre [30].

Another solution, developed by Extreme Networks, is called Extreme Fabric Connect [31]. The solution is based on "Shortest Path Bridging" (SPB) which is specified by the IEEE 802.1Q standard [32]. Extreme Fabric Connect aims to provide a security solution that meets the demands of modern, virtualised networks. The solution can easily be integrated with existing solutions and provides fully separated virtual networks with the option to allow specific types of traffic between the virtual networks. The latter makes the solution comparable to Cisco ACI and its endpoint groups (EPGs).

3.3 Cisco ACI and OpFlex

Cisco ACI (Application-Centric Infrastructure) is Cisco's approach to SDN [33], and is ultimately based on the principles of Promise Theory [34]. The forwarding devices can be considered agents in the context of Promise Theory, and have some built-in intelligence to make decisions based on its internal controls and forwarding table, which are ultimately based on the policies received from the APIC [35]. Consequently, the APIC has some authority over the forwarding devices. However, this is fully compatible with the principles of Promise Theory as presented by Burgess and Bergstra [18, 19]. The forwarding devices comply with the instructions from the controller merely because it is their configured behaviour.

Being a relatively new framework – released in late 2013 [36] – Cisco ACI represents the modern interpretation of Promise Theory and SDN. As could be expected from such a recent framework, it has comprehensive API support for tasks such as administration, networking, monitoring and troubleshooting [37]. Administration may be the most important part of the API with respect to this thesis, as it enables us to obtain endpoints, EPGs and policies using the API. Cisco ACI and OpFlex are discussed in more detail in chapter 4.3.

3.4 Public-key infrastructures and OpenSSL

Public-key cryptography is a widely used technique to communicate securely over insecure networks operated by an untrusted third party and is the main contributor for security in the AC-PKI solution. This section discusses some of the principles, history and standards within public-key cryptography, as well as the widely used TLS protocol which utilises a public-key infrastructure (PKI) and digital certificates to establish secure connections between two endpoints.

3.4.1 Public-key infrastructures and RSA

PKIs were first proposed by Diffie and Hellman in their 1976 paper "New Directions in Cryptography" [38], motivated by the development of computer networks and the need to protect against eavesdropping and interception of network traffic. A practical implementation was proposed by Rivest, Shamir and Adleman in 1978 [39] with the RSA algorithm (named by the first letter in each of their surnames). Documents published by the British cryptographic agency CESG revealed that the concepts of public-key cryptography had been discovered by James Ellis in 1970 [40] – six years before Diffie and Hellman. However, his discovery was classified until 1997.

The principles of PKI were crucial for the development of computer networks, as they allowed anyone to distribute their public key on insecure networks, which could be used to encrypt data that only the holder of the private key could decrypt. Due to the (particularly at the time) heavy computation needed for public-key operations, it was often used to exchange symmetric cryptographic keys that could be used for more efficient communication between two endpoints. In fact, this is still widely used for secure communication over the Internet and is used for the TLS protocol which is discussed in the next section.

3.4.2 SSL and TLS

The Secure Sockets Layer (SSL) was released in 1995 [41] and replaced by the Transport Layer Security (TLS) protocol in 1999 [42]. By providing security for reliable TCP connections, SSL was originally designed to enable online payments but has later proven useful for a range of online services like the web, email, voice and video communications and more [43]. TLS provides the privacy and security that is needed in a hostile network like the Internet.

TLS is developed and maintained by the Internet Engineering Task Force (IETF). The last version of SSL (version 3.0) was deprecated in 2015 and is considered "comprehensively broken" as it relies on insecure algorithms like RC4, MD5 and SHA-1, and has a range of known security flaws [44]. The security flaws of SSL has been gradually fixed through TLS versions 1.0 [42], 1.1 [45], 1.2 [46] and the current version 1.3 [47]. Although any version of TLS is generally considered secure, each version is more secure than its predecessor, so using the latest version is always recommended. Despite being replaced two decades ago, the term "SSL" is still widely used referring to the TLS protocol and is even used in the name of the open-source framework OpenSSL [48].

RSA is often used in TLS for secure key exchange, server authentication and optionally client authentication [47]. By using RSA and the principles of PKI, symmetric keys can be securely shared between two parties, without concern about a third malicious party eavesdropping in between. The symmetric keys, as we previously implied, can then be used for more efficient encryption of the data traffic between the two endpoints.

TLS 1.3 was published in August of 2018 and has a number of improvements over version 1.2, including the removal of insecure and legacy symmetric encryption protocols and rarely used and insecure features and extensions [47, 43]. It also removes static RSA and Diffie-Hellman cypher suites, which means forward secrecy is provided by all public-key-based key exchange mechanisms (i.e. the compromise of keys in the future does not affect previously captured traffic).

Essentially, these changes make it difficult to configure TLS 1.3 insecurely, as opposed to previous versions. This is significant, as downgrade attacks are common against the TLS protocol [49]. These attacks involve persuading the server and client to communicate using legacy encryption or key exchange protocols, which have known vulnerabilities that can be utilised by an adversary.

According to Qualys' SSL Labs [50], who scan the 150,000 most popular websites based on data from Alexa [51], 8.5% of these websites still support SSL v3.0, 94.5% support TLS v1.2 and 10.7% support TLS v1.3 (numbers are from the January 2019 scan). They also provide statistics on cypher strengths and protocols supported, as well as implementation errors and invalid or incomplete certificate chains. About one in three websites has some weakness in its configuration, such as support for SSL, incomplete certificate chains or support for weak cypher suites.

3.4.3 OpenSSL

OpenSSL is an open-source framework for SSL and TLS, and the latest version 1.1.1 supports TLS version 1.3 [48]. It is available as a Linux library that provides the cryptography functions required by SSL and TLS, such as key and certificate generation, validation and revocation, certificate signing requests (CSRs), certificate authorities (CAs) and more [12]. The library also provides simple client and server functionality that can be used for testing.

Ivan Ristić has written a free e-book called OpenSSL Cookbook [13] that gives a great introduction to most of the concepts above. As with most Linux libraries, the manual ("man") pages are invaluable for reference [12].

3.4.4 Alternatives to PKI

Diffie-Hellman Key Exchange (DHE) is a popular way to exchange keys between two parties who have never before communicated [38] and is used in SSH and IPsec and often in TLS. However, in their 2018 paper, Adrian et al. [52] showed that Diffie-Hellman Key Exchange is not as secure as previously believed. They showed that implementations using 512-bit prime numbers could be compromised by relatively unsophisticated attackers and even 1024-bit primes could be compromised by state actors.

The main discovery by Adrian et al. was that DHE does not provide "perfect forward secrecy", meaning that if an adversary stores network traffic while calculating the discrete logarithms needed to derive the encryption key, they could decrypt the captured traffic when they got hold of the encryption key.

3.5 Summary

In this chapter, we have studied relevant literature related to Promise Theory, software-defined networking, Cisco ACI and public-key infrastructures. The related work presented in this chapter sets the ground for the next chapters, in which we present the theory, objectives and architecture for the solution.

4 Theory

The literature review in chapter 3 presented scientific work and documentation, which is relevant to this thesis. In this chapter, we further present the theoretical background for the solution objectives and the proposed solution. Section 4.1 presents theory related to public-key infrastructures, section 4.2 discusses software-defined networks and section 4.3 presents some theory and documentation for Cisco ACI.

4.1 Public-key infrastructures

Public-key cryptography is one of the key concepts in the experimental stage of this thesis and one of the key topics in the problem described in chapter 1. Therefore, we present the theory behind public-key infrastructures (PKIs), cryptography and digital certificates. This section discusses the main ideas and principles behind these, but not the low-level technical or mathematical principles behind them, as these are better described by the referenced sources.

4.1.1 Public-key cryptography

In public-key cryptography, keys come in pairs of public and private keys. A public key is typically used to encrypt information, whereas the corresponding private key is used for decryption. That way, Bob can send his public key over an open network to Alice, or anyone else who might want to send him something encrypted. Alice encrypts her secret message with Bob's public key, and only Bob can decrypt the message using his secret private key. The mathematics behind public-key cryptography is relatively simple, but outside the scope of this thesis. The curious reader may refer to the initial proposal by Diffie and Hellman [38], Trappe and Washington's book on cryptography [40] or several other resources that present the mathematical principles behind public-key cryptography.

There are several advantages in using a PKI over the alternatives discussed in section 3.4.4. Firstly, it is one of the most proven technologies for end-to-end security that does not require a pre-shared key (PSK). Secondly, it can provide integrity and authentication by using certificates and the chain of trust. Thirdly, since TLS runs on the application layer, it does not depend upon a specific version of IP and will work just as well on IPv4 as on IPv6.

4.1.2 Digital certificates

One key problem with public-key cryptography is that in many cases, it is difficult or impossible to know whether the public key you possess belongs to the person with whom you want to communicate. The public key could have been replaced while transmitted over the open network. Digital certificates aim to solve – or at least mitigate – this issue. X.509, which is specified by RFC 5280 [53], is the most extensively used standard for digital certificates [40].

Certificates are typically issued by a **certification authority (CA)** and signed by the CA's certifi-

cate. Each certificate contains some information about the party to whom the certificate was issued (e.g. their web address or company name), their public key and some information about the issuing CA; such as where to get its certificate and public key, which is needed for validation.

The essential part of a certificate is its signature. The signature contains the hash of every other field, which is encrypted using the issuing CA's *private key*. Notice that the private key is now used for encryption, rather than the usual decryption, hence enabling only the private key holder to encrypt but anyone to decrypt the signature. That way, any holder of the certificate can verify the validity of the certificate – assuming, of course, that the private key has not been compromised.

The signature can be verified by (1) generating the hash of the certificate using the indicated hashing algorithm, (2) decrypting the signature using the CA's *public key* and (3) compare the decrypted signature with the one generated in step 1. If the hashes match, they know that only the CA could have encrypted the signature using their private key and that no field in the certificate has been tampered with (as this would cause the hash to change).

The security of PKIs and the trust of a digital certificate ultimately rely on the CA private keys remaining secret [13]. The consequences could be disastrous if a private key was compromised, particularly that of the root CA, as the adversary in possession of the key could sign and revoke certificates at their will. The only way to stop them would be to revoke the certificate (or remove it from the clients altogether in the case of a root certificate) and issue a new certificate to everyone affected. Therefore, the root private key should be stored at an offline computer that is not used for anything other than signing the certificates of subordinate CAs. Security and secrecy of the subordinate CA private keys are also crucial, although the consequences would be less severe if their keys were compromised.

There could be several layers of subordinate CAs before (hopefully) a trusted root CA is reached, whose self-signed certificate is trusted by the client. On the web, this could be a publicly trusted root CA like Buypass or DigiCert, but in company PKIs, it could be a self-signed company certificate that is only trusted by the company devices. We will primarily consider the latter in this thesis. In the case of a company CA, it is a good idea to pre-install the root certificate on all company devices upon the initial configuration, as certificates issued over a network connection may be intercepted. If this is not possible, it is possible to sign the root certificate using the certificate of a publicly trusted CA which is already installed on the device, enabling clients to verify its authenticity.

Often a CA has one or more **registration authorities (RAs)** that can sign certificates on their behalf, or verify certificate signing requests (CSRs) before forwarding it to the CA for signing. As public-key cryptography involves relatively demanding computation, RAs decrease the load on the CA itself and can simplify the PKI architecture. If RAs sign certificates themselves, they need certificates that are signed by the CA and trusted for signing other certificates – i.e. its certificate must be signed by the CA and have the CA flag set to true. The maximum "path length" can be defined in the root certificate to prevent subordinate CAs or RAs from creating new subordinates.

As the number of direct interactions with the CA is decreased, so is the likelihood of its private key being compromised. Distributing the signing of certificates to multiple RAs also decrease the impact of one of their private keys being compromised, as only a limited number of certificates need

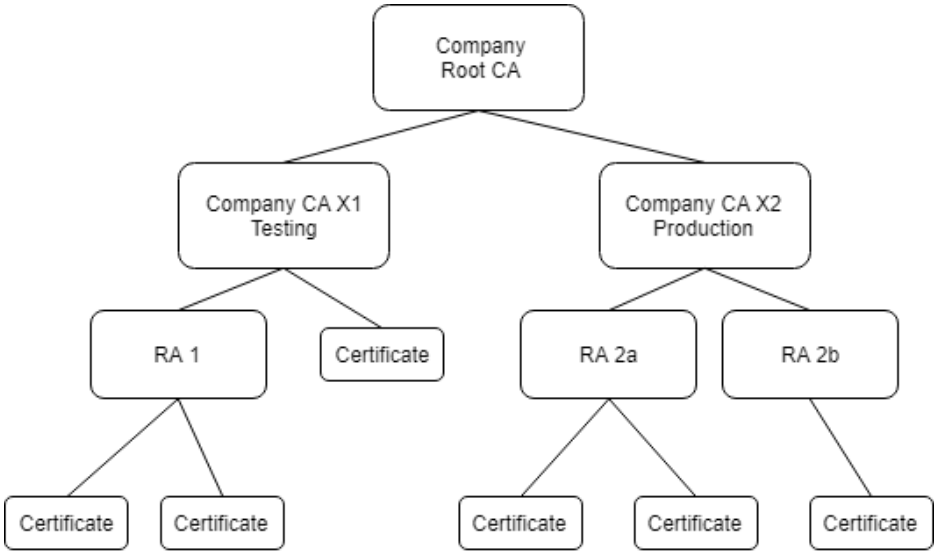

Figure 3: Example of a company CA hierarchy

to be revoked and issued from another RA. Figure 3 illustrates a relatively simple hierarchy of a root CA, two subordinate CAs and three RAs. In this example, the RAs are permitted to sign certificates using their private key, which corresponds to a certificate which is signed by the subordinate CA above.

X.509 certificates have several fields, including the subject fields presented in table 2. The fields describe the subject to whom the certificate was issued. The same fields are also given for the issuer of the certificate. Other fields include the serial number (SN), validity timestamps, version and information about the key usage and signature algorithms.

The issuing CA or RA can set pre-defined requirements to these fields in their configuration files, namely “supplied”, “optional” or “match”. The latter option means that the field should match that of the CA, overriding the value in the CSR. There is no guarantee, therefore, that the fields of the issued certificate will match those of the CSR.

Field	**Description**
C	Country
ST	State, region or municipality
L	Locality
O	Organisation or company
OU	Organisational unit
CN	Common name (typically domain name)

Table 2: Subject fields in an X.509 certificate (based on [12])

4.1.3 Certificate revocation: CRL vs. OCSP

Certificate revocation lists (CRLs) are lists that contain the serial number for revoked certificates [54]. CRLs are signed by the issuing CA and are often valid for days or months. That would, in the context of AC-PKI, mean that a certificate for a connection that has been disallowed by Cisco ACI, could still be used for a fairly long time after its revocation. In practice, Cisco ACI would block this traffic and effectively prevent the endpoints from even initiating a TLS handshake. However, as one of the main purposes of AC-PKI is to provide an additional layer of security, it would be better to have a faster way in which to revoke certificates. It is clear that a CRL based solution would not satisfy our requirements in terms of agility and delay times.

The Online Certificate Status Protocol (OCSP) enables the application to request the revocation status for a certificate by issuing a status request to an OCSP responder [55]. That way, the endpoint can acquire the most recent revocation status, omitting the delay issues associated with CRLs. It is also possible to combine the two, using positive checks on the OCSP responder and negative checks on the CRL. That way, if the OCSP is unavailable or compromised, at least the certificates which are present in the CRL are refused.

Possible OCSP responses are “good”, “revoked” and “unknown” [55]. The former means that the certificate has not been revoked, and *does not* mean the certificate is necessarily valid. A normal certificate validation process, including checking the validity interval and potentially CRLs, needs to be performed in addition to the OCSP process. The simplest OCSP implementation would be a list of revoked certificate serial numbers, and a responder which replies “revoked” if the incoming certificate’s serial number is present in the list and “good” otherwise. OCSP responses should, like CRLs, be digitally signed with a trusted certificate.

Large-volume networks can use “pre-signed” OCSP responses that are valid for some predefined amount of time [56]. As we mentioned in section 3.4.1, public-key operations are computationally demanding. Because OCSP responders often receive many requests, it would be infeasible to sign every response before returning it. By using pre-signed responses, this problem is mitigated, significantly increasing the capacity of the OCSP responder. This does not prevent the responder from checking its revocation list for the most recent changes.

4.1.4 TLS Handshake

The TLS handshake is the process in which two endpoints exchange certificates to validate the peer’s identity, agree on an encryption algorithm and exchange the key material needed to communicate securely. The handshake process for TLS 1.2 is illustrated in figure 4 and described in more technical detail in RFC 5246 [46].

4.1.5 Configuration and misconfiguration

Today, PKI configuration is typically performed using dedicated frameworks like Aruba ClearPass [6] or DigiCert [7]. These frameworks let network administrators manage device categories, access control and set requirements for encryption strength. Although these frameworks may be effective for many corporations, they would require network administrators of a Cisco ACI network to

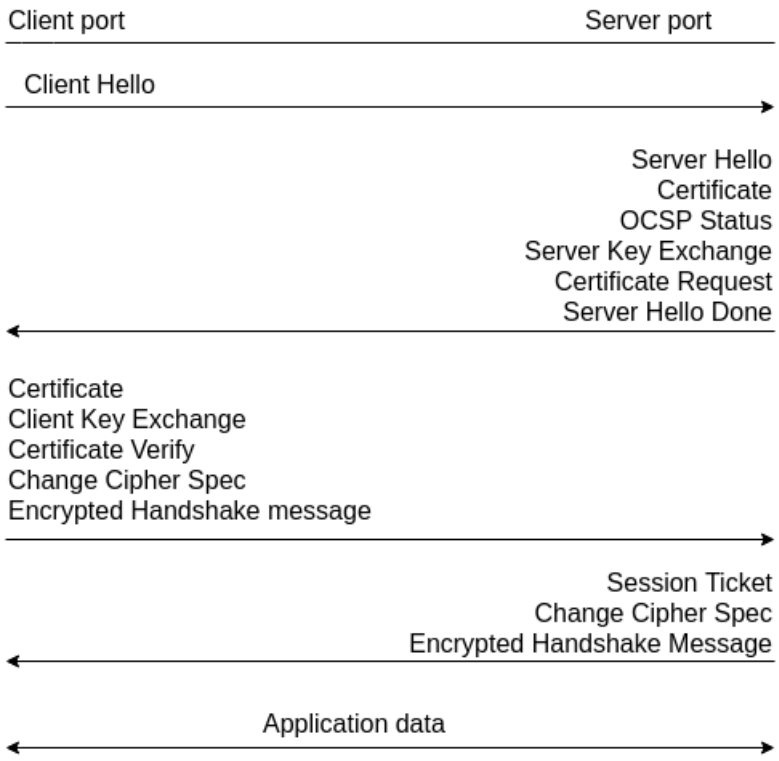

Figure 4: TLS 1.2 handshake, excluding acknowledgement packets, as observed in Wireshark.

perform policy management in two places simultaneously, both in Cisco ACI and in the PKI management framework. Some small to medium-sized businesses also decide to manage their PKI manually using OpenSSL or similar systems, which would require even more manual labour.

Misconfiguration of TLS and other PKI based systems are common vulnerabilities on modern websites and other network-based applications. According to Qualys' SSL Labs scan of the most popular websites in May of 2019 [50], 32.6% of the websites have some insecure configuration – such as incomplete certificate chains, weak cypher suites or support of the outdated SSL protocol. These vulnerabilities could be used by an adversary to access and decrypt traffic that contains valuable data, such as payment details or personal information. In the worst case scenario, the adversary could obtain the private key and use it to sign their certificate, hence enabling them to initiate sophisticated phishing or man-in-the-middle attacks.

4.1.6 CFEngine

CFEngine is a policy and configuration management framework that utilises the principles of Promise Theory introduced in section 3.1. Hosts that run the CFEngine agent aim to maintain a pre-defined "desired state" by dynamically adapting their configuration [22]. The framework is available as an

open source and an enterprise edition.

There are several use cases in the context of PKI in which CFEngine could be useful because there are many agents, with relatively simple desired states and promises. For instance, a CA could promise to offer the root certificate to any qualified client or server within its scope (e.g. corporate network), the RA could promise to process any incoming CSR, and a server could promise to accept incoming handshakes from any client with a valid certificate signed by the root CA.

We will not go into much technical detail about CFEngine in this thesis, but merely describe how it could be used in a production implementation of AC-PKI. By using CFEngine for the PKI and the endpoints' interaction with the PKI, the principles of Promise Theory extend to the whole solution, which could reduce the need for manual configuration significantly.

4.2 Software-defined networking

Software-defined networking (SDN) was briefly discussed in the literature review in chapter 3.2. This chapter will provide a more details on what SDNs do and how they work. In the next section, we study Cisco ACI specifically, to see how it compares to the general ideas behind SDN.

4.2.1 Centralised control

As we discussed in section 3.2, SDN is all about the separation of the control and data planes, by moving the control to centralised controllers. In traditional forwarding devices (i.e. routers and switches), each device had its internal forwarding table that would need to be modified if the network architecture changed. In SDN, however, it is sufficient to modify the forwarding scheme in the controller, and it will inform all forwarding devices of the changes.

Whenever a forwarding device needs to forward a packet to a device that is not present in its forwarding table, it merely requests the path from the controller before continuing. Alternatively, the packet could be forwarded to the controller itself so that the controller can forward the packet to the right device. Which method that is used depends upon the implementation. While the former is more efficient in terms of bandwidth and processing power (at least on the part of the controller), the latter is more memory efficient as the forwarding device does not need to store the packets while waiting for the reply from the controller.

The centralisation of control provides a simple, flexible, scalable and more efficient management of the network architecture. Devices can be moved, added or removed without much configuration. Figure 5 provides an example of a primitive SDN with a client that wants to communicate with a server. The client forwards the packet to the closest SDN switch, which sends a request to the SDN controller to obtain the path to the server. After receiving the path, the switch forwards the message and adds the path to its local forwarding table. As soon as the switch receives the path, it will store it in its forwarding table so it does not have to make the same request for the next packet to that server.

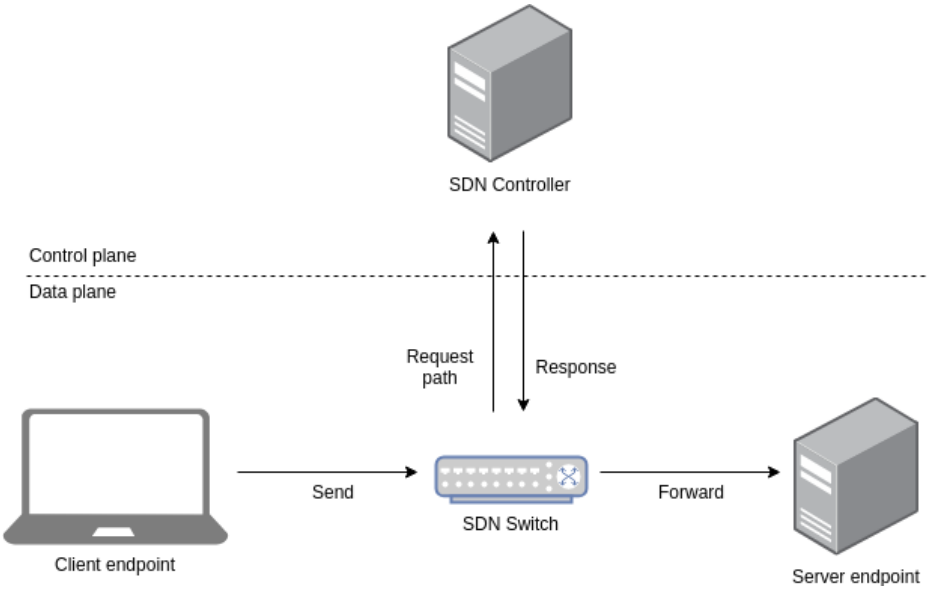

Figure 5: Simple SDN example

4.2.2 Software-defined data centres (SDDC)

The modern data centre is one of the main drivers for SDN [57]. With the increased interest in centralised servers and cloud computing, the scale and technical complexity of modern data centres have outgrown what network administrators are able to manage by traditional means. This has caused an increase in popularity for the software-defined data centre (SDDC). Although the consensus appears to be, like for SDNs, that SDDCs will become more secure, easily managed and robust than traditional data centres, there are several unanswered questions related to SDDC security.

Much of the current research on SDN security focus on "public multi-tenant" SDDCs – such as companies who sell cloud service to their customers, such as Amazon Web Services and Microsoft Azure [57]. Cisco's Application Policy Infrastructure Controller (APIC) Configuration Guide also focuses mostly on multi-tenant data centres [58]. "Private single-tenant" SDDCs, on the other hand, are operated by the company that owns the data centre and typically only serve their services. However, customers often *use* the data centre indirectly when using the company's website or applications. Companies who operate private SDDCs may also want to have multiple tenants, e.g. to separate departments to which different regulations apply. This is called a "private multi-tenant" SDDC.

Modern data centres largely consist of several virtual machines (VMs) running on each physical server. This has caused a deviation between everyday use of the term "server" and the physical servers themselves.

4.2.3 Traffic flow in SDDCs

Traffic flow in SDDCs is divided into north, south and east-west bound traffic. North and south-bound traffic flow between the servers inside and the clients outside the SDDC, while east-west traffic flows between servers within the SDDC. Several trends, like converged infrastructure [59], network virtualisation and cloud computing [60] have caused a significant increase in the amount of east-west traffic within a data centre.

This is especially true for private SDDCs. When all the servers in the data centre belong to the same company, there may be many dependencies between the services running on these servers. Imagine the invoice system in an IT consultancy company, which may require employee information from one server, timesheets from another and price information for the specific customer from yet another server. Just for this simple operation, at least three servers would need to communicate within the data centre. The only north-south traffic would be the request and response for the generated invoice.

4.2.4 Current state of SDN and SDDC security

The literature review in the previous chapter discussed SDN security and divided it into control and data plane security. Control plane security is crucial, as access to the controllers can give an adversary the ability to control the traffic flow according to his or her own desires. For instance, a rouge controller could send confidential information to an adversary's devices instead of the intended recipient. However, these issues are largely left to the infrastructure and framework providers, as well as the controller software, and is outside the scope of this thesis.

To ensure the confidentiality and integrity of the data plane traffic, a corporation may want to implement a PKI on top of their SDN. These two properties can be protected by public-key cryptography and certificates, respectively. Using a PKI-based security architecture like Transport Layer Security (TLS) would provide sufficient security for most organisations, but conventional implementations could lead to additional configuration when setting up a new device or connection, hence limiting the advantages of implementing an SDN.

4.3 Cisco ACI

Although Cisco ACI can be considered Cisco's approach to SDN, it is more than just that. In addition to the control and forwarding plane, which make up the SDN part, Cisco ACI provides a management plane, network services and a transportation plane. Also, it provides a comprehensive policy framework to prevent unintended traffic between devices on the network.

4.3.1 Physical architecture

Cisco ACI implements a two tier spine-leaf switch architecture, as illustrated in figure 6, which is typically referred to as the "ACI fabric". The two tiers consist of spine and leaf switches through which both north-south and east-west traffic flows.

Leaf switches connect servers, top of rack (ToR) or end of row (EoR) switches or external networks to the Cisco ACI network fabric [10]. They also serve as policy enforcement points, ensuring

that only endpoints that are permitted to communicate can do so. **Spine switches** serve as bridges, connecting pairs of leaf switches that need to communicate. The **APIC** provides centralised control of how traffic flows through the fabric, and stores policies that determine which endpoints that are allowed to communicate. Figure 6 provides an example of what a spine-leaf architecture could look like. In a real network, the APIC would be connected to all the fabric switches, but for simplicity this is not indicated in the figure. Large fabrics may have several APICs which are organised in a hierarchy to stay synchronised.

Endpoints in Cisco ACI can be physical devices, virtual machines (VMs) or applications, and are identified by their network interface card (NIC), virtual NIC, domain name system (DNS) name or IP address [34]. Although the term endpoint may be slightly confusing, as it can refer to a variety of physical or virtual systems, it is helpful when discussing Cisco ACI as they are treated more or less equal when it comes to policies and the forwarding of packets. That is very compatible with the highly virtualised modern SDDC. Endpoints are categorised logically into endpoint groups (EPGs) to which policies can be applied. We will get to EPGs and policies in the next section.

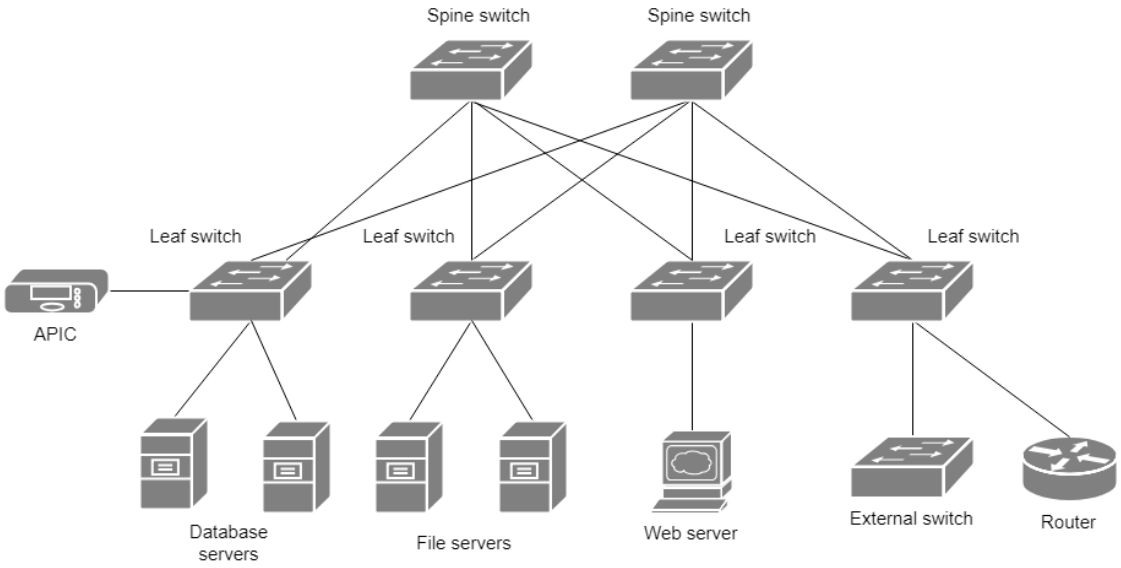

Figure 6: Physical infrastructure of the Cisco ACI fabric.

4.3.2 Logical architecture

The physical architecture in Cisco ACI is not all that different from some network-centric architectures. What is truly different about ACI is the logical architecture of the network. As we discovered in the literature review, the logical and physical network models are not as tightly connected as in traditional networks and physical locations are not as important.

The logical model is built up with a virtual routing and forwarding (VRF) node, one or more bridge domains (BD) and endpoint groups as illustrated in figure 7. BDs can be compared to tradi-

tional subnets and EPGs can be compared to virtual local area networks (VLANs). However, unlike most VLANs and subnets, groups of devices do not need to be located within the same physical area, but can instead be grouped by the function they provide and the policies that apply to them. Hence, endpoints within a single EPG can exist on different VLANs or subnets [35]. Figure 8 illustrates this point with a simple network that consists of three VLANs, each having one web server and one FTP file server, assigned to the EPGs “Web” and “Files” respectively.

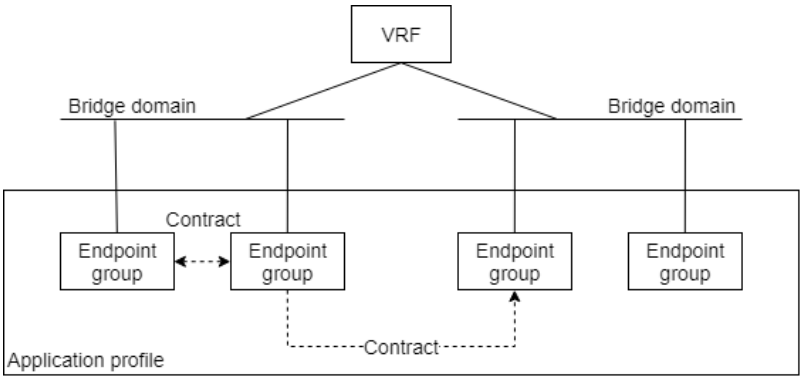

Figure 7: Fundamental components and interactions in Cisco ACI

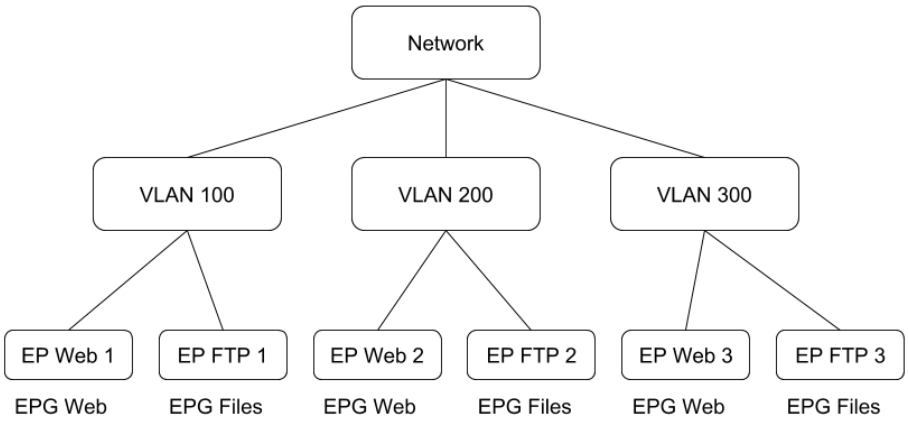

Figure 8: Illustration of the relationship between EPGs and VLANs (based on [35])

By default endpoints within the same EPG are allowed to communicate while endpoints in different EPGs are not. As is also illustrated in figure 7, communication between EPGs may be permitted if they have a contract. This contract can define specific types of traffic to permit or allow any type of traffic between two EPGs. While the former provides better security than the latter, it requires a high level of insight into how the end applications will work. A good approach is to start with an

"any-any" contract and make it more specific when the application is finished. Contracts can allow EPGs to communicate, regardless of whether they are in the same BD or not.

In traditional networks, devices of the same type are typically connected to the same leaf switch. In Cisco ACI, however, a leaf switch could be connected to a file server, a web server and a back-end API server, each belonging to different EPGs. Hence, the purpose, virtual position and policies that apply to a server can be changed remotely without moving any physical hardware or disconnecting any cables. This is one of the things that make Cisco ACI so scalable and agile.

When a new device is connected to the network, it can often be assigned to an existing EPG and hence automatically be configured with the correct policies. EPGs can also be used to enforce zone models, such as demilitarised zones (DMZs), internal and high-security zones [35]. Because devices within each zone typically have the same policies and access rules, they work well with EPGs in Cisco ACI. Security zones could also be implemented as their own tenants.

4.3.3 Tenants

Cisco ACI has built-in tenant support that can be used for different purposes depending on the company's needs [61]. A company that provides cloud services to its customers could use a multi-tenant implementation with one tenant for each customer. Each tenant could be configured individually to satisfy the customers' needs. For private data centres, a tenant could represent a business unit or there could be one tenant for development and testing and one for production. Others may want to only have a single tenant for the entire company – excluding the default tenants that ship with ACI. As mentioned in a previous chapter, we primarily consider single-tenant implementations in this thesis.

There are three default tenants coming with Cisco ACI [61]. The "common" tenant includes, as the name suggests, services that are shared between all tenants, such as DHCP, DNS, shared bridge domains and external networks. The "infra" tenant includes infrastructure such as the ACI fabric. Finally, the "mgmt" (management) tenant is used for management and configuration interfaces for ACI policies, typically interacting with the APIC API.

When configuring the APIC, tenants are the topmost management object, to which the VRF, bridge domains, application profiles, EPGs and more may be added.

4.3.4 Contracts

Contracts are essentially how policies are enforced in Cisco ACI. Without a contract, endpoints are only allowed to communicate using a few whitelisted protocols depending upon the configuration. Each contract can have one or more filters that limit the direction and protocols it allows. Filters can also specify that specific types of traffic should be logged while others should not.

Contracts in Cisco ACI follow a "provider-consumer" model, in which one entity provides a contract and another entity consumes it. Typically the providing entity is the one that offers one or more services, such as a server. For instance, a web server can provide a contract for HTTP and HTTPS traffic to a web client that consumes the same contract. This model is closely related to the principles of Promise Theory and voluntary cooperation.

Every contract has a defined scope that specifies in which parts of the network it will be available. The scope can be Global (i.e. available to all tenants), Private Network (i.e. only available within a specific VRF), or only available to the Tenant or Application Profile (AP) [61]. Any EPG within the scope can consume or provide the contract.

4.3.5 Micro-segmentation in Cisco ACI

Micro-segmentation is a popular topic in modern information security, as it provides device isolation and significantly mitigates the consequences of a compromised device, e.g. a device which is infected with a worm. Typically worms infect one device and then spreads to other connected network devices. Micro-segmentation limits the number of endpoints a worm can reach, as the device is only allowed to communicate with devices with which it has explicit permission – a contract in the case of Cisco ACI. As soon as the worm is detected, the policies can easily be updated to completely isolate the host while investigating further and avoiding more devices being infected.

The concept of micro-segmentation lacks a clear definition in terms of *how microscopic* the segments should be [62]. Some might call Cisco ACI micro-segmented by default, as only endpoints within the same EPG are allowed to communicate without a contract. A stricter definition might require each endpoint to be isolated. However, Cisco ACI offers the latter type as well by enabling "intra-EPG endpoint isolation" [63, 64]. When intra-EPG isolation is enabled, even endpoints within the same EPG need a contract to communicate. It is possible to specify exceptions to this requirement, so specific types of traffic is allowed without a contract.

4.3.6 OpFlex and declarative control

OpFlex is an open-source policy protocol which is used as the southbound interface – i.e. between the APIC and network devices – of Cisco ACI [65]. OpFlex is essentially an API which supports both JSON and XML policy formats and can consequently be compatible with many other applications. The protocol has been submitted as an informational Internet Engineering Task Force (IETF) Internet-Draft [66] which is at the time of this writing in its 3rd draft version.

OpFlex implements the principle of *declarative control*, as opposed to imperative control. Declarative control means that the agents are told what to do, but not exactly how to do it. In the OpFlex paper [33] they make a comparison with air traffic control. For instance, a pilot could be told "turn right heading 60, descend and maintain 6,000 feet" and they are expected to know how to change direction and decrease their altitude. This is possible because of the intelligent agents that know how to perform these manoeuvres. Similarly, ACI compatible switches are given declarative instructions on what to do but are expected to decide exactly how they will perform the requested action. The authors of [33] do recommend protecting the OpFlex protocol with TLS, but control plane security is outside the scope of this thesis.

4.3.7 Cisco Learning Labs and Sandbox APIC

Cisco Learning Labs (`https://developer.cisco.com/learning/`) provides learning tools for Cisco ACI and the APIC. The courses in ACI Programmability and ACI Websockets have proven particularly useful, as they give an introduction to concepts and tools that are significant for this thesis and the

prototype development.

Cisco also provides a virtual Sandbox APIC at `https://sandboxapicdc.cisco.com` (they use a self-signed certificate which may cause a warning upon entering the site in a web browser), which provides full GUI and API access to a virtual Cisco ACI environment. This way, we could set up a virtual tenant with APs, EPGs, contracts and more, without investing in expensive hardware and software to test the prototype during development. One drawback with the Sandbox APIC is that all the user-created tenants and their contents are deleted about once a day to keep things tidy. However, we propose a solution this problem in section 6.8.1.

4.3.8 Cisco APIC API

The Cisco APIC offers an application programming interface (API) which can be used for third-party applications like ours to receive or update data about the network architecture.

The Sandbox APIC, discussed in the previous section, provides an API inspector which shows all the API calls made from the Sandbox GUI to the virtual APIC. For instance, when navigating to an AP within a Tenant, the API Inspector shows the GET request with which the AP can be found. This function has proven helpful, as we had problems finding a good and up-to-date overview of the API calls.

Figure 9 on page 29 shows an example API call in Postman, which requests all the EPGs associated with the given AP. It also shows the JSON response that the APIC returned for that request. The request URL is broken down and described in table 3.

Request URL component	**Description**
https://sandboxapicdc.cisco.com/api/	Base URL for the APIC
node/mo/	Managed object
uni/tn-<tenant-name>/	Tenant domain name
ap-<ap-name>	Application profile
.json	File format, JSON or XML
?query-target=children	Target children, i.e. EPGs for tenants
?subscribe=no	Do not subscribe to changes using WebSocket

Table 3: API GET request breakdown for figure 9. The request gets all EPGs for a given tenant.

4.3.9 APIC queries and WebSocket subscriptions

The Cisco APIC supports REST API calls and replies with either an XML or a JSON response [67]. Interactions with the APIC can be made using HTTP GET, POST and DELETE requests to request, add, modify or delete data, respectively. We primarily focus on GET requests, as this is the only request that will be made from AC-PKI – except for the POST request needed for authentication. Interacting with the APIC requires a valid authentication token which can be received by providing a valid username and password. The authentication process can be automated when a “401 Unauthorized” response is received, indicating an expired token. Automatic authentication requires the credentials to be stored in a file or in memory.

The APIC REST API also allows for query subscriptions using a two-way connection with the WebSocket protocol (RFC 6455) [17]. As long as the connection is open, any changes in the results of a query will be sent to the requesting agent. It is possible to have multiple subscriptions using the same WebSocket connection [67]. Subscriptions may prove very useful when trying to maintain an updated model of the ACI architecture in our system, and will likely cause a decrease in delays and unnecessary requests between our system and the APIC. As only one WebSocket connection is needed, it is unlikely to cause a significant increase in the processing power needed by the system. How the WebSocket protocol and subscriptions work in AC-PKI is described in more detail in Appendix B.

4.3.10 Cisco ACI security

As we mentioned previously, Cisco ACI provides some traffic isolation which ensures that only the intended recipient of a packet is able to view it. However, this is not sufficient security for most modern organisations, who may want to ensure confidentiality, integrity and availability of their network traffic with a higher level of security than that provided by Cisco ACI.

Cisco ACI provides TLS for communications with the APIC management graphical user interface (GUI) and API. This ensures that traffic between the network manager (client) and the APIC is secure and cannot be intercepted.

Also, Cisco ACI traffic can be encrypted on layer 2 using MACsec (IEEE 802.1AE) [68]. This can be configured through the APIC configuration GUI. There are two modes, “must-secure” and “should-secure”, where the former requires MACsec encryption and the latter does not. MACsec encrypts everything other than the source and destination MAC addresses by default. While MACsec can be an effective way to secure communications within the data centre or software-defined wide area network (SD-WAN), it does not provide the end-to-end security between the client and the server that our solution aims to provide.

4.4 Summary

In this chapter, we have presented the theory related to public-key infrastructures, software-defined networking and Cisco ACI. This theory constructs the foundation of the solution objectives presented in the next chapter and the solution presented in chapter 6.

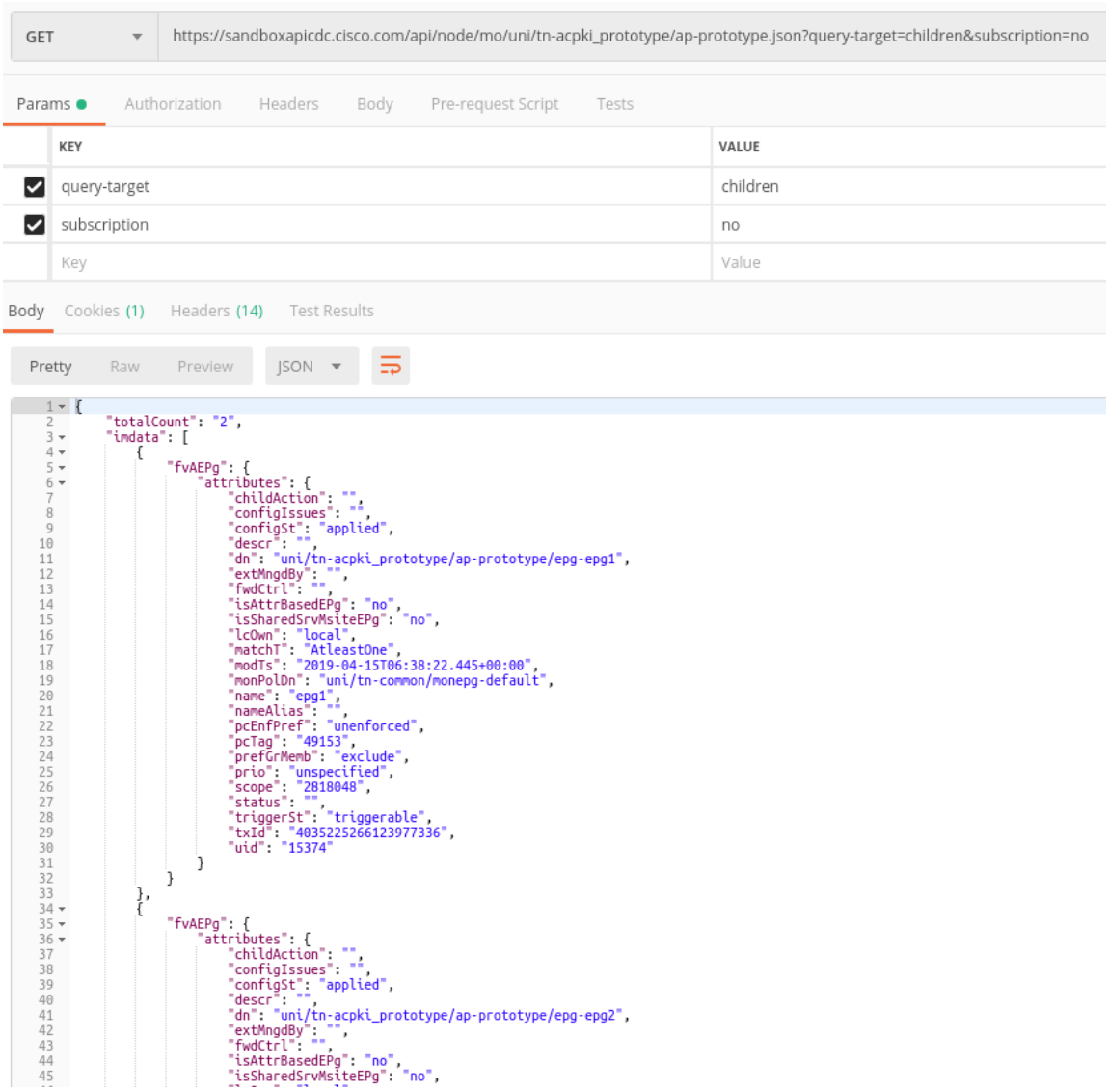

Figure 9: Example of an API call to receive EPGs for a given Tenant

5 Solution objectives

The theories from the earlier chapters imply that the policies from Cisco ACI can be used to generate a PKI and certificates automatically. However, it is apparent that some data processing is necessary between the Cisco APIC and the PKI. In this chapter, we have analysed some objectives, possibilities and constraints for the solution. We also divide the solution into four main components, which set the ground for the solution proposed in chapter 6.

Section 5.1 gives a brief overview of the components and data that flows between them. Sections 5.2 – 5.5 look at each component in more detail and section 5.6 discusses some considerations related to Promise Theory. Sections 5.7 and 5.8 discuss performance and security considerations, respectively. Finally, section 5.9 discusses how the prototype can be used to verify the feasibility and concepts of AC-PKI.

5.1 Overview

We have divided AC-PKI into four internal components; the Cisco ACI component, the core component, the PKI component and the endpoint component. In addition, there is the Cisco APIC which is considered an external component.

The Cisco ACI component requests policy and endpoint data from one or more APICs and passes that to the core component. The core component uses those data to create certificates for connections that are permitted by the ACI policies, which can be issued and validated by the PKI component, which consists of the CA, RA and OCSP responder. Ultimately, these certificates can be used for secure communication between a client and a server. Figure 10 illustrates these components as well as the data that flows between them.

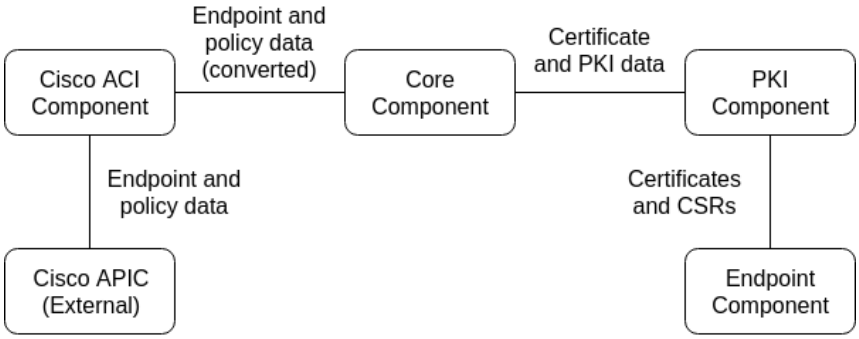

Figure 10: Overview of the main components

Here, we have introduced the Cisco ACI and Core components. The PKI and Endpoint compo-

nents are essentially present in all networks using a PKI, but require some small adjustments to be compatible with AC-PKI. The Cisco APIC is an external component, and consequently it does not require any changes in configuration.

5.2 Cisco ACI component

The Cisco ACI component of AC-PKI is the component closest to the Cisco APIC and is responsible for interactions with Cisco ACI. This component also processes the incoming data and converts it to a format that the core component can use, i.e. mapping the incoming JSON or XML data to instances of the model classes used by AC-PKI. For efficiency, only data that is required by the core component should be added to these models and anything else should be discarded. Whenever the ACI Component receives data for one of its active subscriptions, it should forward those data (using the correct model class) to the core component, using the correct callback method.

In an ideal scenario, all connections permitted by Cisco ACI are in use with TLS or similar protocols. It is likely, however, that the number of permitted connections far outgrows the number of connections used with TLS. Therefore, it is advantageous to generate certificates and check policies on-demand instead of when they arrive to the core component. Especially for large networks, the latter could be very resource intensive.

5.3 Core component

The core component requests specific data related to endpoints and policies from the ACI component and keeps this data in its internal memory. The component provides the necessary information to the RA and CA for issuing and validation of certificates, respectively. Also, the core component should be persistent, storing the data it receives in memory while running and in persistent files or databases between sessions. That way, AC-PKI can continue working even when the APIC is unavailable for a brief period. That also leads to a decrease in hard dependencies to external components, which is advantageous in terms of Promise Theory.

The core component must also be capable of accepting data from active ACI subscriptions. Incoming data must be interpreted, and the internal models and files should reflect the changes from incoming subscription data. That way, when it receives a request from, say, the RA about whether two endpoints are allowed to communicate, it can trust its internal models and need not send a request to the ACI component to verify. If one or both of the endpoints are not, however, present in the core component's internal memory, it must request the information from the ACI component.

5.4 PKI component

The PKI component consists of the CA, RA and OCSP responder. This section presents the purpose, limitations and prerequisites for each of these, as well as the interactions between them and with the core component. The PKI component is ultimately responsible for issuing and validating certificates needed for secure communication between endpoints.

5.4.1 Certification and registration authorities

Because we use an RA to issue new certificates and an OCSP responder to provide revocation statuses, endpoints can validate certificates without contacting the CA, assuming that they have the root certificate as we discussed in section 4.1.2. The only task of the CA, therefore, is essentially to keep the root private key secure and issue certificates to any new RA that is needed.

The RA is responsible for issuing certificates to any endpoint who needs to communicate securely with a new peer, and may therefore experience a much more significant load than the CA. Whenever the RA receives a certificate signing request (CSR) from an endpoint, it must send a request to the core component to determine whether the communication that the certificate represents complies with the Cisco ACI policies, i.e. whether there is a contract between the two EPGs to which the endpoints belong. This is based on a unique identifier, which is discussed in section 5.4.3.

5.4.2 OCSP Responder

As we have implied previously, OCSP can give timely status responses and decrease the delay between a policy change in Cisco ACI and the corresponding certificate revocation in AC-PKI. The OCSP responder is responsible for issuing certificate statuses to endpoints validating their peers' certificates. It should be notified by the core component whenever a connection that was previously allowed has been disallowed by Cisco ACI, e.g. because the contract has been deleted. That way, if an endpoint sends a status request to the OCSP responder, the latter knows whether the certificate has been revoked and hence whether the connection is still permitted by the ACI policies.

Pre-signed OCSP responses, which are valid for some pre-defined amount of time, will decrease the workload on the OCSP responder by reducing the frequency with which it must sign responses.

5.4.3 A mapping from certificates to EPGs

In order to validate certificates efficiently, there must be a relationship or mapping between the certificate and the connection to which it was intended – i.e. an origin and destination EPG. This can be solved by creating a unique identifier that is stored in the core component and added to a field in the certificate. Several fields could be used for this purpose, some of which will be discussed in section 6.6.

Because certificates are issued to endpoints and policies apply to EPGs, multiple certificates may have the same identifier. However, as each certificate has a unique serial number, this is unproblematic. The main advantage of associating a certificate directly with the pair of EPGs, and ultimately the contract between them, is that policies can be checked quickly and without having to find the EPG to which an endpoint belongs for every validation.

5.5 Endpoint component

The endpoint component consists of clients and servers. This section discusses considerations, constraints and requirements related to endpoints and the communication between them. We also discuss some considerations related to the network layer on which that communication occurs.

5.5.1 Server

The server is listening for incoming TLS socket connections for a specific IP address and port. When a new client wants to establish a TLS connection with the server, it must check whether a certificate exists with the client's EPG. If that is not the case, it must contact the RA and request the connection-specific certificate, which in turn validates the connection against the Cisco ACI policies using the core component. During the TLS handshake, as we described in section 4.1.4, each party sends its certificate to the other for validation.

The server must have the root certificate pre-installed, to validate that the incoming client certificate is signed by a valid RA, which in turn is signed by the root CA. It must also be able to send a status request to the OCSP responder to determine whether the certificate is revoked.

5.5.2 Client

The client must be able to initiate a TLS handshake with the server. Other than this, the requirements are very similar to those of the server. The client must, like the server, be able to request certificates from the RA and validate certificates with the CA and OCSP responder, as well as having the root certificate securely pre-installed.

5.5.3 Application vs. network layer

Section 3.2.4 touched upon an important question, namely the layer on which the solution communicates. Vajaranta et al. [26] claim that their solution works well with the SDN paradigm as it runs on the network layer. However, as this paper concerns Cisco ACI – an *application-centric* approach to SDN – we argue that it makes more sense to make an implementation running on the application layer for several reasons which are listed below. Please note that when we refer to the application layer, we refer to the application layer of the IP model, which corresponds to layers 5-7 of the OSI model. The relationship between the layers of the IP and OSI models is illustrated by table 4.

IP model	**OSI model**
Application	Application Presentation Session
Transport	Transport
Internet	Network
Network interface	Data link Physical

Table 4: Layer comparison of the IP and OSI network models

As Cisco ACI is centred around applications and not the physical infrastructure on which the applications run, it makes sense to perform encryption on the application layer instead of the network layer. There may be virtual machines (VMs) and applications running on the same physical device and sharing the same IP address, but belonging to different EPGs and hence have different policies with which they have to comply. Trying to distinguish these VMs and applications at the network

layer would be challenging, if not impossible.

Another advantage is that the application layer is generally better understood by developers and does not require a technical networking background. Because Cisco ACI requires specific knowledge about which protocols that will be used between specific endpoints and EPGs, developers are likely to be more involved in the network configuration than they were previously. Application developers may even want to configure the network devices for the applications they develop. Although AC-PKI does not require much consideration from the average developer, it is advantageous if they understand how it works to protect their applications.

5.6 Promise Theory considerations

Cisco ACI utilises the principles of Promise Theory, as forwarding devices promise the APIC that they will forward the packets according to the policies they receive at a best-effort basis. Policies and contracts dictate how traffic flows between EPGs and forwarding devices, essentially working as promises between them. Controllers have some authority over forwarding devices, which somewhat reduces autonomy but still complies with the principles of Promise Theory.

AC-PKI implementations should also, as far as possible, comply with the principles of Promise Theory. Due to voluntary cooperation, the solution ultimately decides if it will comply with an imposition (request or demand) from another agent. It also decides whether it will provide all the information requested or just parts of it. As we mentioned previously, the hard dependencies upon external components should be minimised, but some dependencies are unavoidable.

For instance, if the API to the APIC is unavailable for a while, the system keeps a simplified model of all policies, endpoint groups (EPGs) and contracts in its internal memory, and will hence be able to maintain operations until the APIC has returned. However, if any changes occur while the API is unavailable, the solution will not be aware of these changes and issues could arise if one of the affected endpoints try to request a new certificate. Another possible measure is to add several APICs, so if one is unavailable AC-PKI can simply contact the second one. Because the policies and endpoints are synchronised between APICs, it is possible to switch seamlessly between them.

5.7 Performance

This section discusses some performance considerations for the AC-PKI solution. Although performance has not been one of the key questions in this thesis, it is one of the questions that must be answered before implementing a production system.

5.7.1 Delay times

Keeping delay times minimal, relies upon frequent updates from Cisco ACI to AC-PKI as well as frequent OCSP requests for certificate validation, both of which increase the amount of traffic generated by the solution. There is ultimately a trade-off between delay times and bandwidth. Each company would have to adjust these parameters according to their requirements.

5.7.2 Availability

Availability is one of the main concerns of the AC-PKI solution. Because endpoints ultimately depend upon the framework for secure communications, the consequences could be severe if the system was compromised or unavailable even for a brief period. Table 5 presents our evaluation of the criticality of each component.

Comonent	**Criticality**
ACI Component	High
Core component	High
RA	Medium
CA	High
OCSP responder	Very high

Table 5: Criticality evaluation of each component

We have given the OCSP responder the highest priority, as it is necessary for certificate validation. Although the OCSP responder depends on the core component to stay up-to-date with the Cisco ACI policies, it can provide cached revocation statuses even if the core component is unavailable for a while. Whether a company wants to allow outdated OCSP responses is, once again, a trade-off between security and the availability of the network, and must be decided based on the company's requirements.

Because of the importance of system availability, there should be sufficient redundancy built-in to any AC-PKI implementation. For critical networks, it is possible to have two complete implementations, so endpoints have a fallback option if the one they use becomes unavailable. For efficiency, it is a good idea to use both systems by default instead of having a dedicated backup system. That way, the workload is divided between both systems.

5.7.3 Bandwidth

Bandwidth requirements for AC-PKI depend largely upon (1) the number of endpoints on a network and (2) the frequency with which each endpoint connects with new endpoints and EPGs. Although packets sent between within AC-PKI are relatively small, there could potentially be many of them in large and active networks. Because of the substantial differences between companies and implementations, we recommend evaluating the bandwidth requirements for each network before implementing AC-PKI. Test systems and calculations based on estimates can be used to set bandwidth requirements for each system.

Just as for SDN, AC-PKI can be easily scaled to satisfy requirements for bandwidth and capacity by purchasing the right hardware and distributing the work to several systems. Because packets sent within AC-PKI are relatively small and infrequent, we do not consider bandwidth a significant obstacle for a production system. If it becomes a problem, however, several measures can be taken to reduce the amount of traffic, including caching, pre-signed OCSP responses or TLS offloading. All of these do somewhat reduce security, but the reduction comes with a pay-off in terms of improved performance and reduced cost.

5.8 Security considerations

This section discusses some security considerations that are essential when finding a solution for AC-PKI. Because the network security will ultimately rely on the security AC-PKI, it is important that the security of every component and every link between two components is considered carefully.

5.8.1 Endpoint trust and security

Any endpoint on the network could send a request to AC-PKI claiming to be any other endpoint registered in Cisco ACI. Spoofing of IP and MAC addresses is trivial [69], and consequently, these fields should not be trusted. Also, endpoints in Cisco ACI are not necessarily represented by a single IP or MAC address, as multiple endpoints could share the same hardware or VM.

Therefore, for AC-PKI to trust an incoming CSR, it needs some additional proof that the requesting endpoint is whom it claims to be. Otherwise, an adversary could imitate a specific endpoint, which has a contract with its target and use that certificate to communicate with the target. The security of the endpoint certificates ultimately depends upon (1) the security that is built-in to Cisco ACI, (2) the security of the certificates and encryption between endpoints and AC-PKI, and (3) the security of the root certificate.

With each component being an autonomous agent, they can make their own decisions as to whether they want to serve a client and which ones it should prioritise in the case of an overload. Access needs to be limited by either a whitelist of endpoints or EPGs or secure client (and server) authentication. Mutual authentication is desirable for all communication within the AC-PKI solution (including endpoints), as it assures all parties of the identity of their peers. Security of the internal communication in AC-PKI is discussed further in section 5.8.3.

5.8.2 Authentication and security with the APIC

The system must be able to authenticate automatically with the Cisco APIC on start-up, as well as maintain the session as long as the system is running. It would be advantageous in terms of security to use an account which only has read access to the necessary data types within the scope of the AC-PKI solution. AC-PKI does not require write access to any fields, as it does not edit any data in Cisco ACI. The data needed by AC-PKI depends upon the specific implementation, but would at least include:

- One or more tenants
- Application profiles (APs)
- Endpoint groups (EPGs)
- Contracts
- Endpoints (EPs)

The Cisco APIC API supports TLS using the HTTPS protocol and, consequently, non-encrypted HTTP traffic is strongly discouraged. Any production system should also perform certificate validation of the X.509 certificate of the APIC. The same goes for the WebSocket Secure (WSS) protocol, which should be used instead of the unencrypted WebSocket (WS) protocol. Without sufficient se-

curity between the AC-PKI system and the Cisco APIC, there is a risk that malicious “rouge” APICs could take the place of a real APIC, or that traffic between AC-PKI and the APIC could be analysed or intercepted by an adversary.

Traffic analysis could, in turn, be used to collect information about the network architecture and ACI implementation, which could be used to discover vulnerabilities in the network itself or the systems that run on it, including AC-PKI. Interception of network traffic could create a discrepancy between Cisco ACI and AC-PKI, which would essentially enable the malicious actor to create, modify or delete endpoints on the network. This, in turn, could cause AC-PKI to revoke legitimate certificates or issue illegitimate certificates.

The latter of the two may sound more significant, but on the contrary, the former is even more significant. If a certificate was issued for a connection that is not allowed by Cisco ACI, the connection would merely be blocked by the forwarding devices, and the certificate would be useless for communication on the network. Of course, it would break the additional security that is provided by AC-PKI, but the traffic isolation provided by ACI would be a huge barrier for any attacker and essentially prevent any handshake from being initiated.

However, if an attacker was able to revoke certificates for legitimate connections, it could stop all encrypted communications between the endpoints – potentially until the system is restarted, and the adversary defused. By selectively choosing the certificates to revoke, e.g. by sending WS updates with “contract deleted” to the AC-PKI solution, this could cause a significant denial-of-service (DoS) attack from which it would be hard to recover quickly. Enabling HTTPS and WSS between AC-PKI and the APIC mitigates both of these vulnerabilities.

It is also essential that the credentials used with the Cisco APIC be stored securely, or better yet prompted upon program start-up. If credentials are stored between sessions, access to the credentials file should be limited to only the class that is responsible for authenticating with the APIC. Memory leakage should be taken into account in both cases, at least if the device on which the system operates is (1) not located in a physically secure and restricted location or (2) shares hardware with other programs that could be vulnerable.

5.8.3 Internal AC-PKI communications

The components can either be stored on the same physical device and communicate using direct method calls, for instance as several classes within the same program, or they can run on separate devices and use network-based communications such as HTTPS over TCP/IP.

Internal communication that uses direct method calls within the system should use access control for the specific programming language which is used. Due to the significant differences between languages, it is difficult to give specific recommendations about what types of access control to use. Instead, we give the general advice of using state-of-the-art security guidelines for the specific programming language that is used for the implementation, as well as the principle of least privilege [70] by using “private” and “protected” methods and attributes when possible.

When using network-based communication between the components, corresponding security measures should be used, such as TLS and X.509 certificates. Essentially, the same security mea-

sures and techniques that AC-PKI provides between endpoints, should be used for network communication between each component.

5.8.4 Memory and storage

AC-PKI relies upon some data being stored in internal files or databases. For instance, the configuration file, EPG to certificate mapping, and contract and endpoint register must be stored within the system. Access to such files or databases must be limited according to best practices for the given operating system and database software.

For Linux systems, for instance, it is recommended to have a dedicated user account for configuring and running AC-PKI and limit access to any critical files to this user. Memory management is another topic that should be considered in any production system, any weaknesses in which could be utilised by sophisticated adversaries. This topic, however, is too vast and too technical to go into in this thesis.

5.9 Verification by prototyping

As was previously mentioned, the prototype developed for this thesis is only intended as a proof-of-concept and should not form the basis for any production system. The prototype does not, and was never intended to, satisfy all the requirements presented in this chapter. That is especially true for the requirements relating to performance or security, as the prototype is only used on a small scale and under controlled circumstances.

The next chapter presents the proposed solution architecture and refers back to the requirements whenever relevant to discuss whether the requirements have been satisfied. Section 6.8 discusses the prototype implementation and whether it satisfies the properties and requirements presented in this chapter.

5.10 Summary

This chapter presented an analysis of objectives, considerations, limitations and requirements for the AC-PKI solution, and divided the solution into four main components. These components form the basis for the solution proposed in the next chapter. We have also discussed some considerations related to security, efficiency, Promise Theory, and whether the prototype needs to satisfy the requirements and characteristics presented in this chapter.

6 Solution

This chapter presents the physical and logical architecture from a bird's eye view and then go into more detail on each of the components. The solution is based on the literature review, theory and solution objectives presented in the previous chapters.

Section 6.1 gives an overview of the solution architecture. Sections 6.2 – 6.5 present each of the solution components. Section 6.6 presents some options as to how certificates can be mapped to their respective contracts and EPGs. Section 6.7 presents some considerations related to certificates exchange and validation and the handshake process. Finally, section 6.8 briefly presents the prototype.

6.1 Solution overview

Figure 11 illustrates the AC-PKI solution and the requests made between each component. The *Core component* presented in section 5.3 has been named the Policy Security Adapter (PSA). We have also named the Cisco ACI Component "ACI Adapter" in the solution. The chapter begins with the component closest to the Cisco APIC at the bottom left of the figure and moves towards the endpoints at the right side.

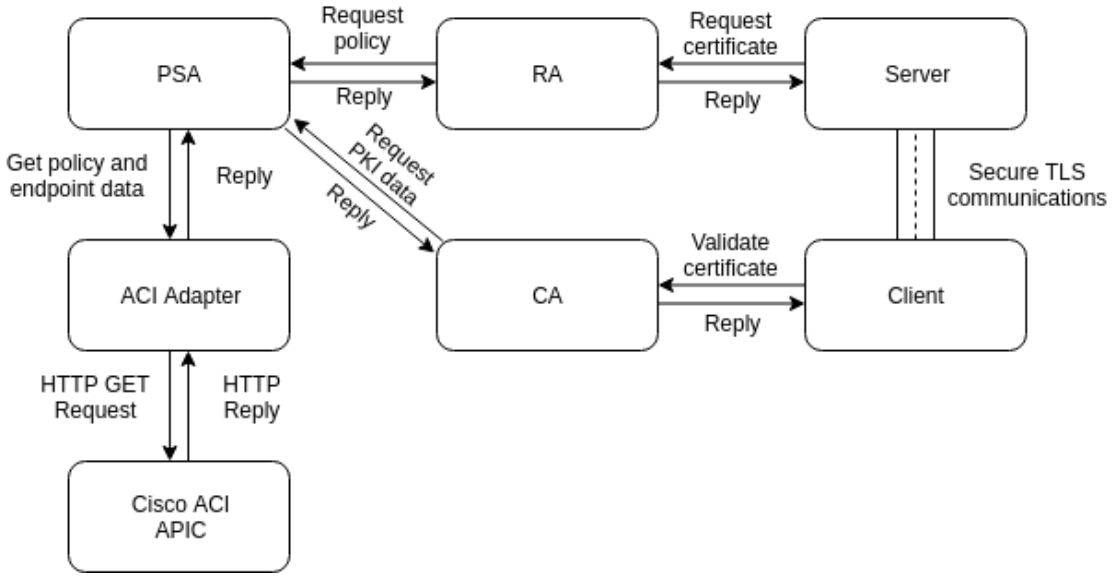

Figure 11: Overview of the solution when a client validates the server certificate. The same process applies when a server validates the client certificate.

6.2 ACI Adapter

The “ACI Adapter” is responsible for all communication with the APIC. It authenticates with a username and password and requests the necessary endpoint and policy data. It also establishes a WebSocket subscription to receive updates whenever changes occur in the requested data. For instance, if a contract is deleted or an endpoint moves between two EPGs, it is important that AC-PKI is updated to reflect those changes.

The ACI Adapter authenticates with the APIC by providing the username and password in a JSON object over the HTTPS protocol. The session token can be extracted from the HTTPS response and saved in memory for later requests, assuming that a “200 OK” response is received. Otherwise, an error message should be printed with the HTTP status code and description.

The initial plan was to use the “ACI Toolkit” (`https://github.com/datacenter/acitoolkit`) library for Python to communicate with the Cisco APIC. However, after having some problems with the WebSocket subscriptions in the library, we instead accessed the API directly using the “requests” and “websocket” libraries. While somewhat increasing the amount of manual labour, these frameworks enabled us to customise the solution in great detail, as we did not have to rely on existing framework methods.

6.3 Policy Security Adapter

The PSA receives the latest data and updates regarding the ACI architecture and policies from the ACI Adapter, and stores those data in its internal memory. It is responsible for providing the necessary policy and PKI data to the RA and CA, as well as revoking certificates via the OCSP responder when necessary. The latter can be done by sending a revocation request directly to the OCSP responder or via the CA.

That way, the PKI reflects the logical architecture of Cisco ACI and certificates are provided only for connections that comply with the ACI policies. Each component of the AC-PKI solution should, as far as possible, be an autonomous agent as defined by the principles of Promise Theory and discussed in section 5.6.

6.4 PKI component

The registration authority (RA) and certification authority (CA) are responsible for issuing and validating endpoint certificates, respectively. These certificates are used to establish a secure TLS connection between the client and the server, which is the ultimate goal of the solution. One key difference between AC-PKI and existing solutions is that certificate pairs are issued only to endpoints between which there is a valid contract. This has significant security implications, which we discuss later in this chapter.

The client and server only communicate directly with the CA and RA and have to provide only a limited amount of information in addition to the certificate signing request (CSR). The CA’s root certificate must be distributed securely to each endpoint in advance. There are several possible ways to accomplish this, including using a corporate certificate which is signed by a publicly trusted CA

(such as the one used for the company's public website), or by adding it with non-network based methods while setting up a new server, computer or company phone.

The RA can sign certificates on behalf of the CA by using its certificate, which is authorised to issue new certificates. In this sense, the RA could be considered a subordinate CA with a specific task of issuing new certificates. It would be advantageous in terms of security to set the "maximum path length" for the root certificate to one. That way, the RAs are unable to create more subordinates, reducing the impact of an RA key being compromised. Certificate validation is discussed in more detail in section 6.7.

6.5 Endpoint component

The main goal of the AC-PKI solution is to set up a secure connection between a client and a server. Although this thesis primarily considers Transport Layer Security (TLS) to provide this connection, other protocols like IPsec [26, 71, 72], MACsec [73] or MPLS-SEC [74] could be used instead. The principles presented in this thesis are applicable to any PKI based protocol. However, as TLS is a well known and widely supported protocol, we consider it the best protocol to demonstrate the principles presented in this thesis. TLS is also suitable for our use, as we argued previously because it encrypts traffic on the application layer and provides end-to-end encryption between the client and the server.

As we discussed in section 6.4, every endpoint needs to have the certificate of the CA stored locally and securely. This certificate can be used to verify the identity of the RA and any certificate issued by the RA, using the chain of trust. That way, endpoints can communicate securely with each other and with the PKI component of AC-PKI.

An endpoint cannot trust the identity of its peer solely based upon the IP address, MAC address or DNS name that is specified in the packets it receives. These fields are not sufficiently verified by the protocols themselves and address spoofing is a widespread vulnerability [69]. However, by validating the peer's certificate with the CA and comparing the address with the common name (CN) field in the certificate, the endpoint can be sure that the endpoint truly is whom they claim to be – at least if we assume that no private keys in the chain of trust have been compromised.

6.6 Connecting policies to certificates

One of the research questions in this thesis, which we presented in section 1.5, was how the PKI and certificates can be organised so that they match and mirror the Cisco ACI architecture. As we argued in the previous chapter, this requires a relationship between each certificate and its respective contract and pair of endpoints.

6.6.1 Problem

Of course, it would be possible to identify the subject endpoint for each certificate based on the CN field. Because this field contains a reference to the endpoint to which the certificate was issued, it would be possible to find its EPG. The problem, in this case, is that the certificate does not contain an identifier for the destination endpoint to which the certificate was intended. Therefore, it would

not be possible to validate the certificate purely based on its fields. It is necessary, therefore, to generate a unique identifier and add this to a field in the certificate, or in some other way, link certificates to their respective contract and source and destination EPGs.

There are several ways in which this value can be stored in a certificate. It can either be stored as a unique identifier in one of the fields in the certificate. This value would be generated and stored by the PSA and added to the certificate by the RA before it was issued. Another approach would be to use subordinate CAs to categorise certificates by to their respective connections. The following subsections discuss some advantages and disadvantages for each option, as well as our final decision for AC-PKI.

6.6.2 Certification authority (CA)

One option would be to have subordinate CAs for each pair of EPGs. However, as large corporations may have hundreds of applications and thousands of EPGs, we consider this an infeasible option. CAs are computationally demanding, particularly if each CA manages its own certificate revocations using CRLs or OCSP responders. It is better to have one or a few CAs on the organisational level that manage a larger amount of traffic, as this would be more efficient.

6.6.3 Organisational unit (OU)

Typically organisational units (OUs) in X.509 certificates are used to specify the department of the organisation or company the certificate was issued to, e.g. “IT Department” or “Economy Department”. However, with modern software development and SDN, applications often span over multiple departments and are no longer as limited to their physical locations. Therefore, the field could be considered excessive for SDNs or at least for our interpretation in this thesis.

The term “organisational unit” is ambiguous and could be interpreted as a unit used to organise the PKI and not the organisation itself. This unusual interpretation of the term could be applied to refer to pairs of EPGs and their respective contracts, as described at the beginning of this section. How OUs are used ultimately depend upon the PKI implementation. In this soltuion, we would validate certificates based on their OU field instead of the CN field, when validating a certificate against the policies.

6.6.4 Serial number (SN)

It would be possible to store all the serial numbers (SN) of issued AC-PKI certificates in the PSA along with the origin and destination EPGs for that certificate. However, as the number of individual certificates is likely to be much higher than the number of connections, and hence OU identifiers as described in the previous section, this registry could be extremely large. The large registry would cause long delays when the PSA needs to request a certificate revocation from the OCSP responder.

6.6.5 Object identifier (OID)

Object identifiers (OIDs) are specified the ISO/IEC 8842-1 / ITU-T X.680 [75] and ISO/IEC 9834-1 / ITU-T X.660 [76] standards and managed by the United Nations organ the International Telecommunications Union (ITU). The OIDs are globally unique and organised in a tree structure, for which

the Norwegian Communications Authority (Nkom) manages the Norwegian branch [77]. Nkom manages separate Norwegian branches for ISO and ITU, as well as a joint branch ("joint-iso-itu-t.country.norway"). Norwegian organisations will generally be assigned space in the joint branch.

OIDs can be used to distinctly refer to digital objects like documents, messages, profiles, object classes, organisations and departments [77]. They can also, indeed, be used to create the unique identifiers needed for our purpose. For the experimental stage of this thesis, registering OIDs with Nkom would not be necessary, as the certificates would only be used within a controlled testing environment in which maintaining unique OIDs would be trivial.

However, any production system using certificate OIDs would require registering the organisation with Nkom (or other national registration authorities) to guarantee unambiguous OIDs. The OIDs can be added as extensions to X.509 certificates [78]. Some advantages of using OIDs are that they would uniquely and unambiguously refer to the AC-PKI application, and that the OU field would be available for its traditional role of identifying a department within a company. Disadvantages include the bureaucracy of registering the OIDs with the ITU or its affiliates and the fact that not all web servers can perform matches on the OID field.

6.6.6 Solution

We used organisational units (OUs) for our solution and prototype implementation, as it requires limited computational resources and no bureaucratic work. It still provides the scalability and efficiency that is needed for our use. OIDs would be a good alternative for a production system.

The OU field works as an identifier to quickly recognise the pair of EPGs between which a certificate may be used. The identifier is symmetrical, meaning that certificate pairs have matching OU fields. Figure 12 shows two certificates with matching OU fields.

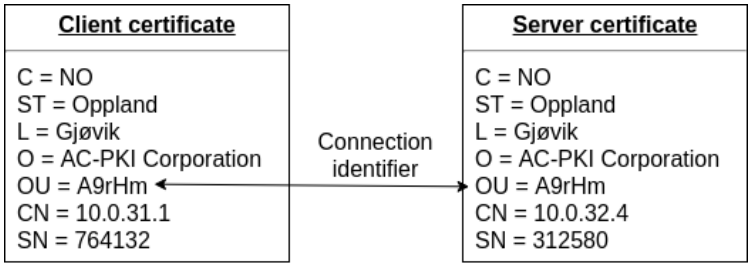

Figure 12: Two endpoint certificates with matching OU fields.

Imagine that the endpoint "EP-2" wants to establish a connection with "EP-3" as presented in table 6. As we can tell from the table, "EPG-A" has a contract with "EPG-B" which allows them to communicate. That is because EPG-A consumes a contract which is provided by EPG-B. Because we use symmetrical OU fields, it would make a difference if EPG-B consumed the contract and EPG-A provided it. In other words, the solution we propose treats any contract as "any-any" when validating certificates, and does not consider the protocols or the direction that the contract may specify.

From table 7 we can see that the certificate between EP-2 and EP-3 should have the OU field set to "A9rHm", as this refers to the pair of EPGs to which the endpoints belong. The identifiers would be longer than this in a proper implementation.

Endpoint	IPv4	EPG	Consumes	Provides
EP-1	10.0.24.1	EPG-A	A	-
EP-2	10.0.24.2	EPG-A	A	-
EP-3	10.0.24.3	EPG-B	-	A

Table 6: Overview of example endpoints and EPGs. Contract A is consumed by EPG-A and provided by EPG-B, which means that the two are allowed to communicate.

OU	EPGs
A9rHm	(EPG-A,EPG-B)

Table 7: A mapping between the OU field and its respective EPGs.

Table 8 shows an example of what the certificate of EP-2 could look like for communication with EP-3. When a connection is established between the two endpoints, they check that the OU field of the peer's certificate matches the OU field of their own. The certificates are exchanged during the TLS handshake. As both endpoints have the root certificate stored locally, they could validate the certificate without contacting the CA at all, knowing that it has been signed by the root private key. However, it is necessary to contact the OCSP responder to ensure that the certificate is still valid. The full field names are listed in table 2 on page 17 along with their abbreviations.

The OU field should be randomly generated and have a length of at least 32 characters (lower and uppercase) and numbers. With 62 distinct symbols and a length of 32, we get $N = 62^{32} \approx 2.27 \cdot 10^{57}$ different strings, and a likelihood of collision of $1/N$. With a likelihood that low, we do not find it necessary to check for collisions when generating a new OU field, even for large networks, which improves the efficiency of the process.

Field	Value
SN	10074092781033755113
C	NO
ST	Oppland
L	Gjøvik
O	Gjøvik Hat Factory
OU	Lhw1BkbXd8Bx9JAgLbNwu6L66bJJWikE
CN	10.0.24.2

Table 8: Example of what the subject fields may look like in an AC-PKI generated X.509 certificate

6.7 Certificates and handshake

This section presents the processes related to certificates and validation. The certificate hierarchy is discussed in subsection 6.7.1, the TLS handshake process in 6.7.2 and certificate validation in 6.7.3.

6.7.1 Certificate hierarchy and PKI

The certificate hierarchy in AC-PKI could be anything from the simple one illustrated in figure 13 to complex hierarchies with multiple levels of subordinate CAs for large corporations. Regardless of the hierarchy that is used, the concept is the same. The client and server certificates can be validated by following the chain of trust from the certificate itself through the RA and potential subordinate CAs, until finally reaching the trusted root certificate. As for conventional PKIs, the optimal architecture for any given company depends upon how the company, its devices and networks are organised.

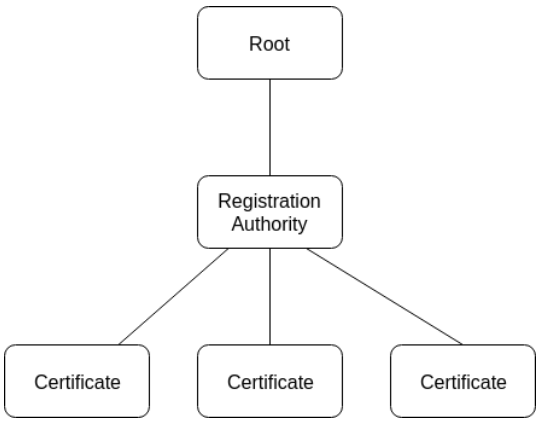

Figure 13: A simple certificate hierarchy for AC-PKI

Because certificate pairs are only issued to endpoints with a valid contract, the PKI improves not only the network security but also the policy enforcement on the network. Because certificates are validated against their corresponding contracts, security is significantly improved over existing solutions. Consequently, even if an adversary is able to obtain a certificate and its corresponding private key, it is virtually unusable on any other endpoint.

6.7.2 Handshake process

The handshake process in AC-PKI is very similar to a conventional TLS handshake, which was described in section 4.1.4. What is different about AC-PKI is the process in which the client and server must request the certificate needed for that particular connection. This process occurs before the handshake the first time two endpoints communicate. Each endpoint generates its public and private keys and create a certificate signing request (CSR). Then, they send a request to the RA which includes the CSR and the destination endpoint. If they are allowed to communicate with that endpoint, the RA signs the certificate and returns it to the requesting endpoint.

Once both parties have a valid certificate, they can initiate a conventional TLS handshake with OCSP status verification. Of course, there is more going on behind the scenes, such as the synchronisation between the OCSP responder and the PSA, which ensures that certificates are revoked once a connection is no longer permitted. However, this part of the solution does not affect the TLS handshake directly and is virtually invisible to the endpoints.

6.7.3 Certificate validation

Because all endpoints need the root certificate pre-installed, they can validate that the certificates have been signed by the root CA. With the additional step of checking whether the certificate has been revoked, by contacting the OCSP responder, the endpoints can be sure that the connection is still allowed by Cisco ACI.

Algorithm 1 presents the certificate validation in AC-PKI using pseudocode. First, it checks whether any contracts exist between the origin and destination endpoints, before checking if the OU field in the certificate is stored in the "ous" dictionary and that the value corresponds to the correct endpoints.

Algorithm 1: Certificate validation in AC-PKI

```
1   input: origin, destination, certificate, ous
2   output: boolean
3   begin
4   if get_contracts(origin,destination).length = 0
5     return false
6   end
7   cert_ou ← certificate.subject.OU
8   if cert_ou in ous.keys
9     return ous[cert_ou] equals (origin,destination)
10  else
11    return false
12  end
13  end
```

6.8 Prototype implementation

A prototype was developed as a proof-of-concept for the solution proposed in this thesis. This section gives a brief overview of the prototype, while Appendix B gives a more detailed and technical description of each component as well as how they work and interact with each other. As mentioned previously, the prototype does not provide the security or robustness expected from a production system.

6.8.1 Interacting with the Cisco APIC

The ACI Adapter is the component of AC-PKI that interacts directly with Cisco ACI and the APIC. It is responsible for authentication, requests and establishing a WebSocket connection with the APIC. As we described in section 4.3.7, Cisco provides a virtual Sandbox APIC that provides a GUI and API with admin access for testing purposes. That way, we could create a tenant, APs, EPGs and

more without investing in the hardware and software needed to set up a test network. A physical network would also require a complex configuration of the networking equipment.

The sandbox APIC can be used by anyone and does not require access to any private network using VPN. However, it should be easy to replace the sandbox APIC with a physical one by replacing the URL in the AC-PKI configuration and setting up the new APIC with the automated scripts. Another advantage with the sandbox APIC is that anyone with the source code (and some patience for the installation process) should be able to connect the system with the Sandbox APIC to see how it works.

As we also mentioned in section 4.3.7, the Sandbox APIC deletes user-created tenants at a regular interval. We have solved this problem by automatically creating a tenant and AP if they do not exist. That way, no manual setup of the APIC is required to run the prototype. These operations are performed using the same API that we use to collect endpoint and policy data. For this purpose, write access is required in the API, but as the admin credentials are provided by Cisco this did not cause any problems.

6.8.2 Policy Security Adapter

The Policy Security Adapter (PSA) class uses the ACI Adapter (ACIAdapter) class to retrieve information from Cisco ACI and maintain an internal model of the network architecture and policies. Using the internal model, it can check whether two endpoints are allowed to communicate on behalf of the RA. If the endpoints have a valid contract, the PSA generates an OU and returns it to the RA. The OU value is also stored in the PSA's internal model, so it can be revoked if the contract is deleted. The PSA does not communicate directly with the endpoints, only with the RA and OCSP responder. That way, application developers require little or no knowledge about AC-PKI and the PSA to make their applications work and communicate securely.

Furthermore, the PSA class has several callback methods that are called when the WebSocket subscriptions in the ACI Adapter receives relevant changes, e.g. in the EPGs or contracts. These callback methods assure that the internal PSA models are up-to-date with the current policy and architecture in Cisco ACI.

6.8.3 CA and RA

For the prototype, we implemented simple certification and registration authorities (CA and RA) that only contain the functionality that is strictly necessary to make the system work as intended. By implementing the classes ourselves, we could customise them to work well with AC-PKI. Because we did not prioritise security for the implementation, the classes were relatively simple.

The RA can issue certificates to the Client and Server classes by direct method calls that provide a "CertificateRequest" containing the origin and destination endpoints as well as a CSR. The origin and destination endpoints are forwarded to the PSA, which checks whether the connection is permitted. If the connection is permitted, the PSA will generate a random OU identifier and register the two endpoints in the internal dictionary using the OU as the key, and return the OU with which the endpoints were registered.

Finally, the RA will overwrite the OU (if any) that was provided in the CSR as well as some other predefined fields in the certificate, such as country, state and organisation. When the certificate is complete, it is signed and returned to the endpoint. If the certificate was denied the method returns “None” and the connection is aborted. Subsequently, when the second request comes in, the PSA will find a matching OU for that connection and return it, ensuring that the two certificates have matching OUs.

The CA keeps the root certificate, which is ultimately the shared point of trust between all parties. Anyone in possession of the root certificate can validate the signature without connecting to the CA. However, the OCSP responder must be contacted to ensure that the certificate has not been revoked.

6.8.4 Client and Server

The Client and Server classes both extend the Endpoint (EP) class and are capable of connecting to one another using the TLS 1.2 protocol. To keep the prototype implementation simple, the classes can only connect to one peer at the time. If an endpoint cannot find its certificate and key files, as defined by the configuration file, the certificate request process will be initiated.

6.9 Summary

In this chapter, we have discussed the AC-PKI solution and each of its components. While doing that, we have seen how data flows from the Cisco APIC to the client and server, ultimately establishing a secure TLS connection based upon the PSA and Cisco ACI policies. We have also looked at possible ways in which certificates can be mapped to their respective origin and destination EPG. Finally, we have briefly presented the prototype that was developed as a proof-of-concept for the AC-PKI principles.

7 Results

This chapter presents the thesis results with respect to the research questions raised in section 1.5, briefly summarising the solution presented in previous chapters. The results and their implications are discussed further in chapter 8.

7.1 Public-key infrastructure

The public-key infrastructure (PKI) presented in this thesis is simple, yet effective. The PKI consists of a root certificate authority (CA), a registration authority (RA) and several connection-specific certificates between communicating endpoints. Large corporations could extend this PKI by adding one or more layers of subordinate CAs and additional RAs. However, this would largely depend upon the specific corporation and was not a significant question in this thesis.

More significant, however, was how the specific fields in the certificates were used, especially the organisational unit (OU) field. While other fields could also be used for this purpose, we used the OU field to create a mapping between a certificate and the endpoint groups (EPGs) to which the communicating endpoints belong. That way, we were able to efficiently verify that the two endpoints which use a certificate for communication are allowed to communicate according to the Cisco ACI policies. The certificate validation process validates both the peer's identity and that both endpoints are allowed to communicate according to the Cisco ACI policies. Consequently, the certificate can only be used by the endpoint to which it was issued.

7.2 Solution architecture

The solution architecture that was proposed in chapter 6, provides a simple overview of how AC-PKI could be implemented in practice. Although there are a few components between the Cisco Application Policy Infrastructure Controller (APIC) and the endpoints, several of these are used to process and convert the data received from the APIC and store it internally for redundancy. The client and server only need to communicate with the CA, RA and OCSP responder, just as in a typical PKI.

The solution does not significantly change the way in which endpoints initiate a TLS communication and causes virtually no changes to the handshake process. That way, most of the work in AC-PKI happens "behind the scenes", meaning that each endpoint and developer does not have to know the details about how it works. The only additional task added by AC-PKI is the process in which endpoints obtain their connection certificate, by contacting the RA and providing an identifier for the endpoint with which it wants to communicate. This can be done easily using dedicated frameworks or libraries.

7.3 Prototype implementation

A simple prototype was developed as a proof-of-concept. The prototype is not secure and should not form the basis or groundwork for any production system. However, it does follow the solution architecture closely and illustrates how it could be used in practice. The prototype was developed using Python and the “request”, “socket” and “PyOpenSSL” libraries.

The prototype is capable of establishing a TLS 1.2 connection between the Client and Server classes and validating the certificate against the policies in Cisco ACI. Certificates are issued dynamically to endpoints which need them, as long as they have a valid ACI contract. The prototype demonstrates that the solution proposed in this thesis is practical and effective. The solution can be tested using the virtual Sandbox APIC provided by Cisco.

8 Discussion and evaluation

This chapter discusses the goals of the thesis and the findings during the research and experimental stages. We evaluate the results with respect to the research questions and problem description in chapter 1. The following sections discuss several aspects, implications and considerations for the AC-PKI solution.

8.1 Feasibility

The main goal of the thesis was, as described in chapter 1, to find out whether a public-key infrastructure (PKI) could be managed efficiently and seamlessly based on the policies in a Cisco ACI network. That way, we hoped to significantly reduce the amount of manual labour needed to configure the PKI and establish secure network connections, hence preserving the flexibility, agility and scalability provided by the network.

During the literature review, we did not find any existing research on this exact topic, although there were researchers trying to provide SDN data plane security using IP security (IPsec) and Internet Key Exchange (IKE). Throughout the theory chapter, we dove deeper into the theory behind, among other things, Cisco ACI and PKIs. Using this background knowledge, we provided an analysis and some requirements for the solution in chapter 5 and sketched a coarse architecture for the proposed solution.

The experimental stage, presented in chapter 6, presented some considerations and decisions made regarding the end solution. It proposed an overall architecture and some technical aspects of this solution and ended with a brief discussion of the prototype we developed. The solution looks very promising in terms of providing a simple, flexible and secure method for issuing certificates based on the network policies.

We were able to develop a prototype that satisfied the primary objectives of an AC-PKI implementation, although we had to make sacrifices due to the lack of time and resources at hand. With more researchers improving the architecture and design of AC-PKI, as well as more time devoted to a secure and efficient implementation, we think that AC-PKI could play a significant role in providing network security for Cisco ACI and software-defined networks in the future.

Regardless of what happens in the future of SDN security, the potential of AC-PKI has far been proven by the design and prototype presented in this thesis. Chapter 10 discusses the next steps towards a secure production system.

8.2 Network and PKI management

The network management does not deviate from how a Cisco ACI network would normally be managed. Endpoints and policies are organised in just the same way as they usually are. Hence,

AC-PKI makes it neither more nor less complicated to set up a Cisco ACI network architecture. This process, however, is already much more efficient than traditional network management, and we see no immediate need for improvements in this area.

When it comes to PKI management, however, AC-PKI does simplify matters significantly. When AC-PKI is fully implemented in the network, the PKI will organise itself automatically and seamlessly based on the policies in Cisco ACI. This means that the only thing left to enable secure PKI based communications is to perform some simple configurations in the endpoints.

For most implementations, we imagine this will be as simple as extending an "Endpoint" class or importing a library with the basic functions needed for secure communication. Because this is also required for communication with conventional protocols like TLS, it does not add significant complexity to the software development. When migrating to use AC-PKI some additional work is required, as existing clients and servers must be adapted to the new framework. However, we consider it worthwhile due to the significant reduction in manual labour needed after the migration.

The implications in terms of management are significant, as management of policy and security are essentially merged into one simple framework. Developers and network managers only have to decide which endpoints are allowed to communicate with each other, and AC-PKI solves the rest. The technical decisions related to key strength, algorithms, handshake and certificate validation is automatically handled by AC-PKI according to the pre-defined configuration.

8.3 Security implications

The security implications are at least as significant as those for management. As we discussed in section 4.1.5, security misconfiguration is a widespread security vulnerability on the web and other TCP/IP enabled applications today. By automating much of this process, we aim to mitigate these vulnerabilities and decrease the likelihood of insecure implementations. Because the PKI configuration is only performed once, i.e. upon the initial configuration, the process can be validated by qualified personnel.

Because one certificate pair is provided for each connection between two endpoints, and certificates are validated against the respective contract, AC-PKI provides significantly improved security over existing solutions. This is currently not provided by any existing framework. Even if an adversary was able to spoof a server certificate, they would be unable to use the certificates from any other endpoint. While the AC-PKI solution could, of course, be compromised, it would be virtually impossible to do so with existing hacking tools. The hacker would require detailed insight into the Cisco ACI network and the mechanisms of AC-PKI to compromise a good implementation.

The automatic revocation of certificates which are no longer in use, i.e. for connections which have been disallowed by Cisco ACI, means the number of certificates that are still valid but no longer in use will be close to zero – assuming that developers remember to remove the contracts or endpoint groups (EPGs) from Cisco ACI when they are no longer needed. Cisco ACI provides an excellent overview of EPGs and their respective contracts, and because AC-PKI mirrors the architecture of Cisco ACI, it is much easier to maintain a simple PKI.

Although several security aspects of AC-PKI have been considered in this thesis, it would be

unwise to assume that the system is entirely secure. This thesis should only be considered the first few steps towards a production system. Feedback and suggestions from the security community could significantly improve the system. As for the prototype, we have already discussed that the purpose is not providing a secure system, but rather proving that the main principles behind AC-PKI can be implemented in practice.

8.4 Interoperability

AC-PKI, as presented in this thesis, is adapted for Cisco ACI specifically. However, there are several other policy and access management frameworks that serve similar purposes. Examples of such services include ForgeRock's Open Access Management [79], IBM Security Access Manager [80] and Microsoft's Active Directory [81]. By replacing the ACI component of our solution with similar solutions for alternative services, AC-PKI could integrate with such solutions without major changes to the core and PKI components. For large corporations who have more than one policy framework, multiple adapters could work in parallel.

Similarly, other PKI systems than OpenSSL could be used with some changes in the PKI component. It would also be possible to provide multiple versions of the PKI component for different PKI systems. That way, AC-PKI would be compatible with a broad range of systems and apply to more companies, using different systems for policy and access management. Because a significant part of the complexity is within the core component, CA and RA, creating and maintaining different adapter and PKI components would not add a large amount of complexity to AC-PKI. It is likely that a significant part of the adapters could be generalised and shared between them.

Another approach is to implement AC-PKI as a proxy solution, leaving the task of assuring compatibility to the policy framework and PKI. That would give more flexibility but also more work to the companies who implement the solution to their networks. Such a solution could, for instance, let users connect using one API on the network side and another API on the PKI side. In this case, the policy framework and API would have to be changed to be compatible with AC-PKI. Alternatively, custom adapters could be developed for the company's specific needs, similar to the ACI and PKI adapters generated for our prototype.

8.5 Performance and Scalability

Encryption operations, and particularly those in public-key cryptography, are computationally demanding tasks which may require significant processing power for large networks. This is especially true for AC-PKI systems in which many devices frequently need to establish connections with new endpoints. In such networks, it may be unfeasible to have one certificate for each pair of EPGs, due to the processing power it would require. In such networks it could be necessary to (1) allow certificates to be used with several other EPGs, somewhat reducing security and policy enforcement or (2) change how endpoints are separated into EPGs, essentially decreasing the number of total EPGs or (3) use TLS offloading to terminate the encryption just before the endpoint is reached.

However, these problems would only arise when hundreds or even thousands of certificates are generated per minute, which would not be the case for most data centres. Because certificates only

need to be generated the first time two endpoints communicate, we consider it unlikely that this will become a significant issue. If problems do arise for any particular data centre, TLS offloading is the best alternative of the three mentioned above, as it does not add significant complexity. That means the TLS connection would be terminated at the load balancer or dedicated offloader, relieving the endpoint from the demanding public-key operations. Consequently, this solution would not provide full end-to-end security, but as long as the data centre's security (whether physical or digital) is sufficient, this is not a significant sacrifice.

A more significant performance concern is related to OCSP responders. Because OCSP responders need to be contacted whenever an uncached TLS connection is established, the frequency of these requests could be overwhelming for the OCSP responder. As we briefly discussed in section 4.1.3, pre-signed responses can reduce the workload on the responder significantly. It is also possible to use a load balancer to distribute the work between several OCSP responders or to distribute several responders throughout the network.

The flexibility and scalability of software-defined networks in general and Cisco ACI specifically, significantly simplifies the task of scaling AC-PKI. It is easy to add new AC-PKI devices when the existing ones are overloaded. Hardware can also be replaced without creating significant work in terms of configuration. Generally, we consider AC-PKI to be very scalable for anything from a small company data centre to large cloud providers, although some changes and adjustments may be necessary. However, this will remain unanswered until a large-scale prototype or production system exists.

8.6 Availability and Resilience

Any system that implements AC-PKI depends upon its availability to issue new certificates and – depending upon the implementation's security requirements – even to validate existing certificates. The consequences of a compromised or unavailable AC-PKI system would, therefore, be highly significant, possibly causing applications to be unavailable.

This problem can be mitigated by, for instance, (1) adding redundancy to the network, i.e. multiple nodes running the AC-PKI software, (2) hardening the AC-PKI system and hardware on which it runs and (3) allowing the OCSP responders to continue sending responses if the PSA has been unavailable for a brief while. Even after implementing the countermeasures above, AC-PKI should be considered a critical part of the infrastructure, to the likes of the CA, RA and network controllers.

9 Conclusion

We have presented the Application-centric public-key infrastructure (AC-PKI), a new architecture and method that issues digital certificates in a Cisco ACI software-defined network. By utilising the policies in Cisco ACI, we were able to issue certificates only to those endpoints which are allowed to communicate, virtually eliminating the need for manual configuration of the PKI. By issuing certificates to specific connections, i.e. pairs of endpoints with a valid contract between them, we significantly improve the security and minimise the impact of spoofed certificates. Because certificates are validated against their respective contracts, any spoofed certificate cannot be used by any other endpoint. Consequently, most existing hacking tools would not work against AC-PKI. The adversary would require detailed insight into Cisco ACI and AC-PKI to compromise a proper implementation.

Throughout the thesis, we conducted a literature review, studied the relevant background theory and specified the solution objectives. We developed an architecture and methodology, as well as a prototype as a proof-of-concept. The prototype demonstrates that the principles of AC-PKI can be implemented in practice. Although the prototype does not satisfy all the characteristics of an AC-PKI implementation described in this thesis, it does follow the architecture carefully and only issues certificates to endpoints who have a valid contract between them. The prototype is capable of establishing a TLS connection between a client and a server.

Based on the theory, architecture and prototype discussed above, we conclude that AC-PKI does have significant potential for the future of data plane security. The solution significantly simplifies the PKI management and improves the end-to-end security between a client and a server. In order to determine how well the solution works in practice, future work is required, and large-scale prototypes need to be implemented and tested. We discuss the next steps towards a production system in chapter 10.

10 Future work

This thesis only constitutes the first few steps towards a production AC-PKI system. This chapter will briefly discuss future work that needs to be done in order to achieve such a system and some of the challenges that lie ahead.

The solution architecture and methodology likely has some potential in terms of efficiency and security. For instance, the number of interactions between AC-PKI components can possibly be reduced. Although we consider the solution very scalable, problems may arise when large systems with thousands or even millions of simultaneous connections, each of which with its own certificate. In such a system, adjustments may be necessary for the solution architecture.

After optimising the architecture and methodology, a more realistic and complex prototype should be developed and tested in an environment closer to a production system. This is suitable for projects with more time and resources than we had for this thesis. Finally, a production system will need further security and risk analyses during development, to mitigate any vulnerability that may still be present.

The use of CFEngine was only briefly discussed in this thesis, specifically in section 4.1.6. Using the framework in the prototype would have lead to a complexity above that which one student could handle in a single semester. Nevertheless, it is our opinion that Promise Theory and CFEngine has significant potential for AC-PKI and should be further considered in future work.

Interoperability, which was discussed in section 8.4, is also an area with several possible improvements. By increasing the number of supported access and policy services, AC-PKI would be a viable alternative for many more companies.

Bibliography

[1] Feamster, N., Rexford, J., & Zegura, E. W. 2014. The road to SDN: an intellectual history of programmable networks. *Computer Communication Review*, 44, 87–98.

[2] Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. Jan 2015. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1), 14–76. doi:10.1109/JPROC.2014.2371999.

[3] Leon-Garcia, A. & Ganjali, Y. 2015. Software-defined networks. *Computer networks*, 92.

[4] Scott-Hayward, S., Natarajan, S., & Sezer, S. 2016. A survey of security in software defined networks. *IEEE Communications Surveys & Tutorials*, 18, 623–654.

[5] Balakrishnan, R. 2015. Cisco ACI – A hardened secure platform with native, built-in security. URL: https://blogs.cisco.com/datacenter/cisco-aci-a-hardened-secure-platform-with-native-built-in-security.

[6] Aruba. Aruba ClearPass Network Access Control. URL: https://www.arubanetworks.com/assets/so/SO_ClearPass.pdf.

[7] Digicert. Secure Network Access with DigiCert PKI Platform. URL: https://www.digicert.com/secure-network-access/.

[8] Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. 2007. A design science research methodology for information systems research. *J. of Management Information Systems*, 24, 45–77.

[9] Crnkovic, G. D. *Constructive Research and Info-computational Knowledge Generation*, 359–380. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. URL: https://doi.org/10.1007/978-3-642-15223-8_20, doi:10.1007/978-3-642-15223-8_20.

[10] Cisco. Cisco Learning Labs – ACI Programmability. URL: https://learninglabs.cisco.com/tracks/aci-programmability.

[11] Cisco. Cisco Learning Labs – ACI Websockets. URL: https://developer.cisco.com/learning/lab/sbx-intermediate-aci-03_websockets/step/1.

[12] OpenSSL. 2019. OpenSSL manual (man) pages (Ubuntu 16.04 LTS – OpenSSL v1.1.1). URL: https://www.openssl.org/docs/man1.1.1/man1/.

[13] Ristić, I. 2018. OpenSSL Cookbook.

[14] Various contributors. Requests: HTTP for Humans. URL: https://2.python-requests.org/en/master/.

[15] Python.org. socket - Low-level networking interface (Python 2.7). URL: https://docs.python.org/2.7/library/socket.html.

[16] Read the Docs. Websockets 7.0 documentation. URL: https://websockets.readthedocs.io/en/stable/intro.html.

[17] Fette, I. & Melnikov, A. 2011. RFC 6455: The WebSocket Protocol. URL: https://tools.ietf.org/html/rfc6455.

[18] Burgess, M. 2005. An approach to understanding policy based on autonomy and voluntary cooperation. In *Ambient Networks*, Schönwälder, J. & Serrat, J., eds, 97–108, Berlin, Heidelberg. Springer Berlin Heidelberg.

[19] Bergstra, J. & Burgess, M. 2014. *Promise Theory*. xtAxis.

[20] Borrill, P., Burgess, M., Craw, T., & Dvorkin, M. 2014. A promise theory perspective on data networks. URL: https://arxiv.org/pdf/1405.2627.pdf.

[21] CFEngine. CFEngine 3.12 Documentation. URL: https://docs.cfengine.com/docs/3.12/.

[22] What is CFEngine?

[23] Greene, K. 2009. TR10: Software-Defined Networking. URL: http://www2.technologyreview.com/news/412194/tr10-software-defined-networking.

[24] Vaezi, M. & Zhang, Y. *Virtualizing the Network Services: Network Function Virtualization", bookTitle="Cloud Mobile Networks: From RAN to EPC*, 47–56. Springer International Publishing, Cham, 2017. URL: https://doi.org/10.1007/978-3-319-54496-0_4, doi:10.1007/978-3-319-54496-0_4.

[25] Shaghaghi, A., Kaafar, M. A., Buyya, R., & Jha, S. 2018. Software-Defined Network (SDN) Data Plane Security: Issues, Solutions and Future Directions. URL: https://arxiv.org/abs/1804.00262.

[26] Vajaranta, M., Kannisto, J., & Harju, J. 2017. IPsec and IKE as Functions in SDN Controlled Network. doi:10.1007/978-3-319-64701-2_39.

[27] Kent, S. & Seo, K. 2005. RFC 4301: Security Architecture for the Internet Protocol. URL: https://tools.ietf.org/html/rfc4301#page-4.

[28] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., & Kivinen, T. 2014. RFC 7296: Internet Key Exchange Version 2 (IKEv2). URL: https://tools.ietf.org/html/rfc7296.

[29] Kotuliak, I., Rybár, P., & Trúchly, P. 2011. Performance comparison of ipsec and tls based vpn technologies. *2011 9th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 217–221.

[30] Boyd, R. 2018. What is SD-WAN? URL: https://blogs.cisco.com/cin/what-is-sd-wan.

[31] Hermansen, J. Extreme Fabric Connect / Shortest Path Bridging (Norwegian). URL: https://www.dataequipment.no/cms/files/483/shortest-path-bridging--en-kort-introduksjon_del-1.pdf.

[32] July 2018. IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks. *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, 1–1993. doi:10.1109/IEEESTD.2018.8403927.

[33] Cisco. 2015. OpFlex: An Open Policy Protocol. URL: https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.html.

[34] Cisco. 2013. Principles of Application Centric Infrastructure. URL: https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/aci-fabric-controller/white-paper-c11-729906.html.

[35] Cisco. 2014. Cisco Application Centric Infrastructure (ACI) - Endpoint Groups (EPG) Usage and Design. URL: https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731630.html.

[36] Tam, G. 2014. Happy Birthday, Cisco Application Centric Infrastructure! URL: https://blogs.cisco.com/perspectives/happy-birthday-cisco-application-centric-infrastructure.

[37] Cisco. 2018. Cisco APIC REST API Configuration Guide. URL: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/2-x/rest_cfg/2_1_x/b_Cisco_APIC_REST_API_Configuration_Guide/b_Cisco_APIC_REST_API_Configuration_Guide_preface_01001.html.

[38] Diffie, W. & Hellman, M. E. 1976. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6).

[39] Rivest, R. L., Shamir, A., & Adleman, L. February 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2), 120–126. URL: http://doi.acm.org/10.1145/359340.359342, doi:10.1145/359340.359342.

[40] Trappe, W. & Washington, L. C. 2006. *Introduciton to Cryptography with Coding Theory (2nd edition)*. Pearson.

[41] Freier, Karlton, & Kocher. 2011. RFC 6101: The Secure Sockets Layer Version 3.0 (historical record). URL: https://tools.ietf.org/html/rfc6101.

[42] Dierks & Allen. 1999. RFC 2246: The TLS Protocol Version 1.0. URL: https://tools.ietf.org/html/rfc2246.

[43] Rescorla, E. 2015. Stanford Seminar - The TLS 1.3 Protocol. URL: https://www.youtube.com/watch?v=grRi-aFrbSE.

[44] Barnes, Thomson, Pironti, & Langley. 2015. RFC 7568: Deprecating Secure Sockets Layer Version 3.0. URL: https://tools.ietf.org/html/rfc7568.

[45] Dierks & Rescorla. 2006. RFC 4346: The TLS Protocol Verison 1.1. URL: https://tools.ietf.org/html/rfc4346.

[46] Dierks & Rescorla. 2008. RFC 5246: The TLS Protocol Version 1.2. URL: https://tools.ietf.org/html/rfc5246.

[47] Rescorla. 2018. RFC 8446: The TLS Protocol Version 1.3. URL: https://tools.ietf.org/html/rfc8446.

[48] OpenSSL. OpenSSL website. URL: https://openssl.org.

[49] Alashwali, E. S. & Rasmussen, K. 2018. What's in a Downgrade? A Taxonomy of Downgrade Attacks in the TLS Protocol and Application Protocols Using TLS. *CoRR*, abs/1809.05681.

[50] Qualys SSL Labs. SSL Pulse. URL: https://www.ssllabs.com/ssl-pulse/.

[51] Alexa – An amazon.com company. Alexa Top 500 Global Sites. URL: https://www.alexa.com/topsites.

[52] Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J. A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Béguelin, S. Z., & Zimmermann, P. 2018. Imperfect forward secrecy: how diffie-hellman fails in practice. *Commun. ACM*, 62, 106–114.

[53] 2008. RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. URL: https://tools.ietf.org/html/rfc5280.

[54] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., & Polik, W. RFC 5080: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.

[55] Santsesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., & Adams, C. 2013. RFC 6960: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol. URL: https://tools.ietf.org/html/rfc6960.

[56] Hurst, A. D. R. 2007. RFC 5019: The Lightweight OCSP Profile for High-Volume Environments. URL: https://tools.ietf.org/html/rfc5019.

[57] Göransson, P., Black, C., & Culver, T. 2017. Software Defined Networks. URL: https://www.elsevier.com/books/software-defined-networks/goransson/978-0-12-804555-8.

[58] Cisco. 2018. Cisco APIC Basic Configuration Guide. URL: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/3-x/basic_config/b_APIC_Basic_Config_Guide_3_x/b_APIC_Basic_Config_Guide_3_x_chapter_010.html.

[59] Garber, L. 2012. Converged infrastructure: Addressing the efficiency challenge. *Computer*, 45, 17–20.

[60] Oppitz, M. & Tomsu, P. *Software Defined Virtual Networks*, 149–200. Springer International Publishing, Cham, 2018. URL: https://doi.org/10.1007/978-3-319-61161-7_8, doi:10.1007/978-3-319-61161-7_8.

[61] Cisco. 2018. Operating Cisco Application Centric Infrastructure. URL: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/1-x/Operating_ACI/guide/b_Cisco_Operating_ACI.html.

[62] Cisco UK. 2018. Cisco ACI (video series. URL: https://www.youtube.com/playlist?list=PLqrLLrFUWVzbVbUZ0IC9yPBC9LXRqhxAy.

[63] Lage, J. 2017. What's new with ACI Micro Segmentation. URL: https://blogs.cisco.com/datacenter/whats-new-with-aci-micro-segmentation.

[64] Cisco. Cisco ACI Fundamentals. URL: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/1-x/aci-fundamentals/b_ACI-Fundamentals.html.

[65] Cisco. 2014. Cisco ACI Security. URL: https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-732354.pdf.

[66] 2016. OpFlex Control Protocol (draft 3). URL: https://www.ietf.org/archive/id/draft-smith-opflex-03.txt.

[67] 2019. Cisco APIC REST API User Guide. URL: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/1-x/api/rest/b_APIC_RESTful_API_User_Guide.html.

[68] Cisco. 2019. Cisco APIC Layer 2 Configuration Guide. URL: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/2-x/L2_config/b_Cisco_APIC_Layer_2_Configuration_Guide.html.

[69] Heberlein, L. T. & Bishop, M. 1997. Attack class: address spoofing. URL: https://pdfs.semanticscholar.org/17a7/18292c5091d717cffa296c3814c0d9bea5dc.pdf.

[70] Rouse, M. 2017. What is principle of least privilege? URL: https://searchsecurity.techtarget.com/definition/principle-of-least-privilege-POLP.

[71] Lopez, R. & Lopez-Millan, G. Software-Defined Networking (SDN)-based IPsec Flow Protection. Internet-Draft draft-ietf-i2nsf-sdn-ipsec-flow-protection-03, Internet Engineering Task Force, October 2018. Work in Progress. URL: https://datatracker.ietf.org/doc/html/draft-ietf-i2nsf-sdn-ipsec-flow-protection-03.

[72] Frankel, S. & Krishnan, S. 2011. RFC 6071: IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. URL: https://tools.ietf.org/html/rfc6071.

[73] Group, I. . W. Dec 2018. IEEE Standard for Local and metropolitan area networks-Media Access Control (MAC) Security. *IEEE Std 802.1AE-2018 (Revision of IEEE Std 802.1AE-2006)*, 1–239. doi:10.1109/IEEESTD.2018.8585421.

[74] Fang, L. 2010. RFC 5920: Security Framework for MPLS and GMPLS Networks. URL: https://tools.ietf.org/html/rfc5920.

[75] 2011. ITU Recommendation X.680 (08/2015): Abstract Syntax Notation One (ASN.1): Specification of basic notation. URL: https://www.itu.int/rec/T-REC-X.680/en.

[76] 2011. ITU Recommendation X.660 (07/2011): Procedures for the operation of object identifier registration authorities: General procedures and top arcs of the international object identifier tree. URL: https://www.itu.int/rec/T-REC-X.660/en.

[77] Nkom. Objektidentifikatorer (Norwegian resource). URL: https://www.nkom.no/marked/nummerforvaltning/andre-nummerplaner/objektidentifikator.

[78] Redhat. Standard x.509 v3 certificate extensions. URL: https://access.redhat.com/documentation/en-US/Red_Hat_Certificate_System/8.0/html/Admin_Guide/Standard_X.509_v3_Certificate_Extensions.html.

[79] forgerock.com. Open Access Management. URL: https://backstage.forgerock.com/docs/openam/11.

[80] IBM. IMG Security Access Manager. URL: https://www.ibm.com/no-en/marketplace/access-management.

[81] Microsoft. Azure Active Directory. URL: https://azure.microsoft.com/nb-no/services/active-directory/.

A Abbreviations

Table 9 lists some of the most used abbreviations in this thesis.

Abbreviation	**Meaning**
AC-PKI	Application-centric public-key infrastructure
ACI	Application-centric infrastructure
AP	Application profile (Cisco ACI)
BD	Bridge domain (Cisco ACI)
CA	Certification authority (PKI)
EP	Endpoint (Cisco ACI)
EPG	Endpoint group (Cisco ACI)
IP	Internet protocol
IPsec	IP security
LAN	Local area network
MAC	Medium access control
MACsec	MAC security
NFV	Network function virtualisation
OCSP	Online certificate status protocol
PKI	Public-key infrastructure
RA	Registration authority (PKI)
SDDC	Software-defined data centre
SDN	Software-defined network / networking
SSL	Secure sockets layer (predecessor to TLS)
TLS	Transport layer security
VLAN	Virtual local area network
VM	Virtual machine
VNF	Virtual network functions
VRF	Virtual routing and forwarding (Cisco ACI)

Table 9: Abbreviations

B Prototype documentation

This appendix will describe the Python prototype that was developed as a proof-of-concept in more detail. Each of the main components will be described in the subsections below.

B.1 Introduction

This appendix will describe the Python prototype that was developed as a proof-of-concept in more detail. Each of the main components will be described in the subsections below.

In the following sections, each component and its main functions are described in some technical detail. However, every class and method will not be presented. However, the source code on GitHub is well commended and should be easy to understand for anyone with some programming experience and technical competence within sockets and PKIs. The Python language is easy to understand and has a relatively easy syntax for anyone with even a little programming experience. Some readers may find it useful to review parts of the source code for a more detailed understanding of how the prototype works, while others may find the descriptions and explanations in this appendix to be sufficient.

B.2 Start-up and configuration

The source code was delivered along with this report and will be published in GitHub after the report has been published (`https://github.com/sigurd120/acpki.git`). Instructions on how to download, install and configure the system can be found in the file "README.md". Please note that the prototype was developed and tested only on Linux operating systems. Different configurations and software may be needed to run on other operating systems. If you are not using Linux and have problems running the prototype on your system, we strongly recommend using an Ubuntu 18.04 virtual machine (VM). Following the instructions carefully should enable you to run the program without problems.

B.3 Component overview

The prototype uses the same component architecture that was presented in chapter 6. Each component has its own role and tasks which will be described in the following sections. Figure 14 illustrates the flow of requests and responses when setting up a secure connection between a client and server. The main purpose of each component was discussed in chapters 5 and 6 and will not be repeated here.

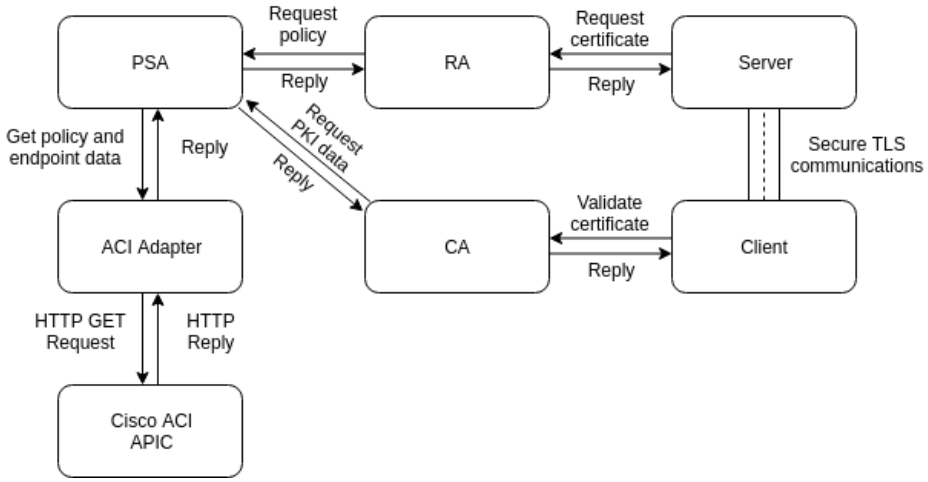

Figure 14: Component overview for the server certificate. The same procedure applies for the client certificate, with the Client and Server switching positions.

B.4 Cisco ACI component

B.4.1 Policy Security Adapter (PSA)

The Policy Security Adapter (PSA) class is core of the solution and the component which interacts with both the ACI Adapter and PKI component. This class handles the initial requests for EPGs and their corresponding contracts, and handles subscription callbacks for any changes in these data. The PSA also maintains internal models that are created upon the initial request and updated when the subscription callback methods are called. This way, the internal models are up-to-date with limited delays as long as the subscriptions are active and the APIC is available.

The CA and RA can call methods in the PSA directly to determine whether the connection between any given pair of EPGs is permitted or not. The response will be based on the PSA's internal model, and is therefore subject to minor synchronisation delays. However, we consider the advantages in terms of reduced load on the APIC makes this delay worthwhile. Changes in endpoints and contracts are also relatively infrequent.

Section 6.6 presented how the OU field can be used to map certificates to the EPGs for which the given certificate can be used. The PSA can both generate these fields, returning them to the RA when issuing new certificates, and store the values in a file for persistence. Of course, this will only happen if there exists a valid contract between the two EPGs in question. Otherwise, it will return "None". The persistent file is crucial, as it is required for the PSA to recognise certificates issued in previous sessions. If it was not for this file, new certificates would have to be issued every time the program restarted.

B.4.2 ACI Adapter

The "ACIAdapter" class is the main point of communication between the PSA class and the Cisco APIC. It can establish a connection with the APIC, send various requests and prepare the environment, i.e. create the Tenant, AP, EPGs and more if they do not already exist. This is particularly helpful while using Cisco's sandbox APIC, as it tends to delete all the user-created tenants at a regular interval. This class also handles the session with the APIC, with which all communication is performed.

As the name suggests, the ACI Adapter converts data from the APIC JSON responses to AC-PKI model objects that can be easily handled by the PSA and other classes. The class rely on some other service classes, some of which are described below. For classes not described below, the sourcecode is commented and should be relatively easy to understand for anyone with some programming and network experience.

Key service classes:

- ACISession (aci/ACISession.py)
- Subscriber (aci/Subscriper.py)
- Subscription (aci/Subscription.py)
- WSThread (aci/work_threads.py)
- RefreshThread (aci/work_threads.py)

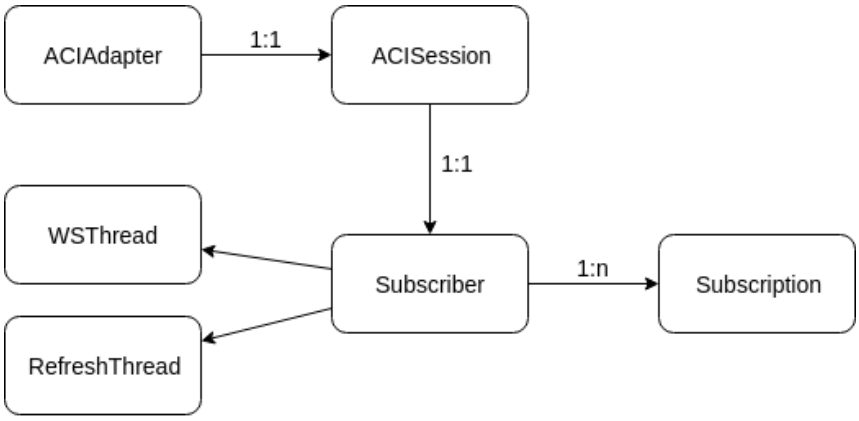

Figure 15: Overview of classes related to the ACI Adapter

The ACIAdapter has one instance object of the class *ACISession*, which represents the current session with the APIC. It has methods to authenticate with the APIC and attributes that store the username, password, session token and more. It also has the ability to resume a previous session based on a token and cookie file saved in persistent files.

The ACISession has a *Subscriber* object which handles anything related to the WebSocket subscription with the APIC. The Subscriber maintains two threads, the *WSThread* (WebSocket Thread)

and *RefreshThread*. The former is responsible for maintaining the WebSocket, listening for incoming data and forwarding that to the assigned callback method. The latter is responsible for refreshing the WebSocket and its subscriptions at a regular interval, to prevent them from being closed by the APIC (the default timeout is one minute).

The Subscriber object also maintains a list of zero or more *Subscription* objects, each of which representing an APIC subscription. This is merely a model class and does not contain any methods. Although the ACIAdapter class in the prototype has been especially developed for the Cisco Sandbox APIC, it should work with any Cisco APIC without significant changes to the source code. However, as we did not have access to physical ACI hardware this has not been done in practice.

B.4.3 APIC Sandbox (external)

Cisco ACI is typically implemented for large-scale networks and data centres, which require a lot of configuration, even for a small testing environment. Also, the hardware and licences required for testing may be hard to acquire for a master thesis. Luckily, Cisco has provided us with a sandbox APIC that is available on the Internet at `https://sandboxapicdc.cisco.com/`. Note that the APIC uses a self-signed certificate, which will cause a warning when entering the website with most browsers. It also requires the "verify=False" flag to be set when using the Python "requests" library to send HTTP requests.

The sandbox allows users to access a virtual APIC without a licence, create their own tenant and configure it as they wish with custom application profiles (APs), endpoints and EPGs. In addition to providing the GUI-based web interface you will find by entering the URL in a web browser, the sandbox APIC provides API functions for configuring or reading the current state of each tenant. Essentially, it will provide us with everything we need to test our solution, without requiring us to purchase any physical equipment or licences.

One drawback with the sandbox APIC is that user generated tenants appear to be deleted at a regular interval. However, due to the comprehensive API support, we can easily check if our tenant exists and create it again if it has been deleted.

B.4.4 WebSocket subscription

Before establishing a WebSocket connection with the APIC Sandbox, we need to authenticate by sending an authentication POST message containing the username and password. The body of the authentication request and parts of the reply are shown in listing 2 and 3, respectively.

Listing 2: Authentication request body

```
METHOD: POST
URL:    https://sandboxapicdc.cisco.com/api/aaaLogin.json
{
"aaaUser": {
"attributes": {
"name": "admin",
"pwd":  "ciscopsdt"
}
```

```
}
}
```

Listing 3: Authentication reply

```
{
"totalCount": "1",
"imdata": [{
"aaaLogin": {
"attributes": {
"token": "cLSAZ6yrP9R0KjZUhPS5QXfgYDB6Kj24[...]",
"siteFingerprint": "MiMRhU+xGHzaMF2W",
"refreshTimeoutSeconds": "600",
"maximumLifetimeSeconds": "86400",
"guiIdleTimeoutSeconds": "1200",
"restTimeoutSeconds": "90",
"creationTime": "1553590366",
"firstLoginTime": "1553590366",
"userName": "admin",
"remoteUser": "false",bash
"unixUserId": "15374",
"sessionId": "UJKpCu3NR4m2[...]",
"lastName": "",
"firstName": "",
"changePassword": "no",
"version": "3.0(2k)",
"buildTime": "Sun␣Nov␣05␣18:18:05␣PST␣2017",
"node": "topology/pod-1/node-1"
},
"children": [...]
}]
}
```

The “children” segment of the reply contains some privilege information that would have taken up too much space to include in this appendix. Parts of the token and session ID are also removed to save space, indicated with “[...]”. The token, in fact, is the most important part of the reply from the APIC, as it is what we need to establish the WebSocket subscription.

For the prototype, a secure WebSocket (WSS) connection was used. For the Sandbox APIC this is done by connecting to *wss://sandboxapicdc.cisco.com/socket[TOKEN]* using a WebSocket client, adding the token received in the authentication process above [11]. While the WebSocket connection is open, normal HTTP requests can be sent to the API with the “subscription” GET parameter set to “yes”. A normal JSON response will be returned from the server, and any changes in the queried information will be sent over the active subscription. Note that a subscription to the APIC lasts for only one minute by default, and hence need to be refreshed by sending a “subscriptionRefresh” request to the API.

B.5 PKI component

B.5.1 Certification authority (CA)

The prototype consists of a root certification authority (CA) is the root of trust in the solution. All trust and security ultimately relies on the root CA's private key being kept private. The public key of the CA must be available to all agents who need to verify other certificates (e.g. the server and clients). Because there can, essentially, be no secure connection without the root certificate, it needs to be distributed securely and not over the network. For this reason, the root CA will not issue its certificate upon request. Alternatively, the certificate, which includes the root CA's public key, could be distributed using another secure channel, such as a certificate that is signed by a publicly trusted CA.

B.5.2 Registration authority (RA)

The RA is permitted to sign certificates on behalf of the CA. Essentially, this means that the RA has a certificate which is signed by the CA and eligible to sign other certificates, i.e. with the "ca" flag set to true. In essence, this makes the RA is a subordinate CA which is allowed to issue certificates. The certificates issued from the RA *are not* allowed to distribute their own certificates, as they will have the "ca" flag set to false.

When endpoints need to establish a secure connection with another endpoint, with which they do not have a valid certificate, they will send a certificate request to the RA with the information needed to verify the connection against the Cisco ACI policies. When a valid request is received, the RA will make a decision as to whether the connection is allowed and a certificate should be issued. The RA forwards information about the requesting endpoint, the destination endpoint and both of their EPGs to the PSA, and receives a reply which states whether this connection complies with the Cisco ACI policies. If this is the case, it will sign the certificate and send it back to the client. Otherwise, it will refuse to issue the certificate.

B.5.3 Inter-component communication

In the prototype, there is no socket connection between the CA, RA, the Policy Security Adapter (PSA) or the client and server. Instead, they all communicate using direct method calls. The reason for this, is that it makes the system easy to understand and less complex when setting it up. The functionality is almost identical to one in which these classes would run on dedicated hardware and communicate using socket connections, much like the connection between the client and the server.

B.5.4 OCSP Responder

The OCSP responder is also implemented as a simple Python class, with methods to revoke, unrevoke and to check whether the certificate with the given serial number has been revoked. Essentially, the function provided by this OCSP responder is very similar to a more complex third party solution. Revoked serial numbers are stored in a text file, which is updated dynamically with the aforementioned methods. Using a text file as a database could indeed cause problems when load

on the responder is high, but as this is a controlled prototype environment, the solution is sufficient for most use cases.

When a certificate is verified by the CA, it will make a request to the OCSP responder (i.e. the OCSP callback method), to verify that the certificate has not been revoked. This is essential, as it will prevent certificates from being issued for connections that were previously allowed but have since been disallowed (e.g. due to a change in policy or endpoints being moved between two EPGs). If the OCSP responder does not reply, certificates should not be issued but instead raise an error. Otherwise, the solution would be susceptible to denial-of-service (DoS) attacks on the OCSP responder, which could be a serious vulnerability.

B.6 Client and server

The Server is a simple class that listens for incoming TLS 1.2 connections, using the libraries "socket" and "PyOpenSSL" (`https://pyopenssl.org`). At the time of this writing, PyOpenSSL did not support TLS version 1.3. Clients can connect to the Server at the dedicated port at localhost (IP 127.0.0.1).

When the client class is run, it will establish a connection with the server automatically. The handshake process is performed using functions in the PyOpenSSL library. Through the client interface, the user can type simple messages to send to the server, which will simply be sent back to the client, acknowledging that the connection works and that the message was received. If the message is received, it means that the certificates were verified and that the handshake was successful. Otherwise, the client would output an error message, explaining what went wrong.

Both the client and the server have their own private keys and certificates, which are signed by the Certification Authority (CA). They also have the CA's certificate to validate their peers' certificates. The certification and registration authorities are described in more detail in B.5.

B.7 Prototype output

Listing 4 provides output from running the prototype. The output gives a good overview of the order in which each event occurs and the API calls that are used. At the bottom, the "Hello" message sent from the client can be seen, followed by a TLS 1.2 handshake and then the "Hello" message is returned by the server.

Listing 4: Prototype output sample (verbose mode)

```
APIC Certificate verification is disabled. For improved security,
   please provide a certificate file in the configuration.
Connection status: 200 OK
Successfully resumed saved session.
Establishing websocket connection with the APIC...
Websocket connected successfully.
WS opened: wss://sandboxapicdc.cisco.com/socket[...]
GET https://sandboxapicdc.cisco.com/api/node/mo/uni/tn-
   acpki_prototype.json
```

```
Reponse: 200 OK
GET https://sandboxapicdc.cisco.com/api/node/mo/uni/tn-
    acpki_prototype/ap-prototype.json
Reponse: 200 OK
GET https://sandboxapicdc.cisco.com/api/node/mo/uni/tn-
    acpki_prototype/ap-prototype.json?target-subtree-class=fvAEPg&
    order-by=fvAEPg.name|asc&query-target=children&subscription=yes
Reponse: 200 OK
Creating new subscription
Creating new subscription
Creating new subscription
Creating new subscription
Creating new subscription
Server is listening for clients. Connect at 127.0.0.1:13151
Connected successfully to server.
You can now start typing input data to send it to the server. Send
    an empty line or type exit to end the connection politely.
    Established connection with client at 127.0.0.1:56740

Refreshing subscription 72058251184832518
GET https://sandboxapicdc.cisco.com/api/subscriptionRefresh.json?id
    =72058251184832518
Reponse: 200 OK
Hello
OCSP callback: server-endpoint
SSL Client verify callback
SSL Client verify callback
OCSP callback: client-endpoint
Server serial number: 17875320950502593001
SSL Server verify callback
SSL Server verify callback
Hello
```