

**FH Aachen**

**Department of Electrical Engineering and Information  
Technology**

**IT Security Subject Area**

Prof. Dr. Georg Neugebauer

## **Bachelor's Thesis**

# Zero Trust Security Architectures in Smart Home Environments

Submitted by:

Hristomir Dimov

Matriculation number: 3536320

Study Program: Computer Science

April 14, 2025

Supervisor: Prof. Dr. Georg Neugebauer

Co-supervisor: M.Eng. Sacha Hack

# Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Aachen, April 14, 2025

# Abstract

With the growing adoption of Internet of Things (IoT) technology and its essential role in the realization of the smart home, the threat of the end user privacy being compromised is consequently ever more prevalent due to the expanded network surface. The common network architecture ensures the security of the network perimeters, while implicitly trusting that communication within the smart home, typically between IoT devices and the users, is secure, which serves to extend the attack surface of the home network.

This bachelor's thesis focuses on the integration of Zero Trust security principles in the smart home network architecture with the intention of providing better security for users. This is achieved by introducing two smart home architectures — one with an on-premise hosting concept and another that relies on the AWS cloud service provider for its hosting process. In that regard, with the combination of the features from both the Zero Trust (multi-factor authentication, monitoring, and analytics) and perimeter-based (IP-based traffic filtering) models, two smart home solutions established on the architectures have been realized.

Furthermore, this work evaluates the two smart home environments with the help of predefined primary and secondary metrics, aiming to assess and formulate an answer to the question on whether a Zero Trust smart home network should be hosted locally or in the cloud, depending on the user's needs for security and technical expertise.

**Keywords:** Zero Trust, IoT, Smart Home, Architectures, Security, On-premise, Cloud, Multi-factor Authentication, Metrics

# Contents

<b>1</b>	<b>Introduction</b>	<b>11</b>
1.1	Motivation and Objectives . . . . .	11
1.2	Structure . . . . .	12
<b>2</b>	<b>Fundamentals</b>	<b>14</b>
2.1	Smart Homes . . . . .	14
2.1.1	Internet of Things (IoT) . . . . .	14
2.1.2	Home Automation . . . . .	15
2.1.3	Edge Computing . . . . .	16
2.2	Zero Trust Security Model . . . . .	17
2.2.1	Origins . . . . .	17
2.2.2	Core Components . . . . .	18
2.3	Fundamental Technologies and Concepts . . . . .	21
2.3.1	Home Assistant . . . . .	21
2.3.2	Raspberry Pi . . . . .	22
2.3.3	Communication Protocols . . . . .	22
2.3.4	Infrastructure-as-a-Service . . . . .	24
2.3.5	Infrastructure-as-Code . . . . .	25
2.3.6	Containerization . . . . .	27
<b>3</b>	<b>Theoretical Part</b>	<b>29</b>
3.1	Proposed Architectures for Smart Home . . . . .	30
3.1.1	On-Premise Architecture . . . . .	31
3.1.2	Cloud Architecture . . . . .	34
3.2	Evaluation Metrics . . . . .	36
3.3	Limitations . . . . .	38
3.3.1	On-Premise Architecture . . . . .	38

3.3.2	Cloud Architecture . . . . .	39
<b>4</b>	<b>Practical Part</b>	<b>40</b>
4.1	On-Premise Architecture Implementation . . . . .	40
4.1.1	Configuration . . . . .	44
4.1.2	Structure & Device Integration . . . . .	48
4.2	Cloud-based Architecture Implementation . . . . .	50
4.2.1	Configuration . . . . .	51
4.3	Conclusions . . . . .	57
<b>5</b>	<b>Evaluation</b>	<b>59</b>
5.1	Overview . . . . .	59
5.2	Implementation Assessment . . . . .	60
5.3	Metric Evaluation . . . . .	62
<b>6</b>	<b>Summary and Outlook</b>	<b>65</b>
6.1	Summary . . . . .	65
6.2	Outlook . . . . .	67
<b>A</b>	<b>Appendix</b>	<b>69</b>
	<b>References</b>	<b>77</b>

# List of Tables

3.1 Metrics Overview Table . . . . .	37
5.1 Security Metrics Evaluation Table . . . . .	63
5.2 Secondary Metrics Evaluation Table . . . . .	64

# List of Figures

2.1	A smart home with several examples of HA . . . . .	16
2.2	Multi-Factor Authentication Workflow . . . . .	19
2.3	Home Assistant architecture overview . . . . .	21
2.4	Architecture of a Raspberry Pi . . . . .	22
2.5	Main blocks of a Raspberry Pi . . . . .	22
2.6	Zigbee mesh network . . . . .	23
2.7	MQTT Publish/Subscribe Architecture . . . . .	24
2.8	Terraform Workflow . . . . .	26
2.9	Docker Compose Workflow . . . . .	28
3.1	Smart Home Environment Concept Diagram . . . . .	31
3.2	Main Components of a Smart Home Architecture . . . . .	32
4.1	Main Components of a Raspberry Pi Starter Set . . . . .	41
4.2	Raspberry Pi Imager . . . . .	42
4.3	Home Assistant Start Page . . . . .	42
4.4	User Profile Creation in Home Assistant . . . . .	43
4.5	Home Assistant Default Dashboard . . . . .	43
4.6	Backup Instance Details . . . . .	44
4.7	Home Assistant MFA YAML Configuration . . . . .	44
4.8	Home Assistant MFA QR Code Example . . . . .	45
4.9	Home Assistant IP Ban Configuration . . . . .	45
4.10	Home Assistant IP Ban Blocklist . . . . .	46
4.11	Home Assistant User Ban . . . . .	46
4.12	Home Assistant Remote Tunnel YAML Configuration . . . . .	46
4.13	Remote Tunnel Dashboard . . . . .	47
4.14	Cloudflared Tunnel Configuration . . . . .	47
4.15	Remote Tunnel Access Error . . . . .	48

## *List of Figures*

---

4.16	Architecture Diagram of On-Premise Environment . . . . .	49
4.17	Home Assistant Devices Dashboard . . . . .	50
4.18	Architecture diagram of cloud-based environment . . . . .	51
4.19	main.tf of cloud environment configuration . . . . .	52
4.20	Terminal output after applying script changes . . . . .	52
4.21	variables.tf . . . . .	53
4.22	terraform.tfvars . . . . .	53
4.23	HomeAssistant-SG Security Group . . . . .	54
4.24	HomeAssistant-SG in the AWS Dashboard . . . . .	55
4.25	EC2 Instance Setup Script . . . . .	56
A.1	Local Home Assistant configuration.yaml File . . . . .	69
A.2	Cloud Home Assistant configuration.yaml File . . . . .	70
A.3	variables.tf Terraform File . . . . .	71
A.4	terraform.tfvars Terraform File . . . . .	72
A.5	main.tf Terraform File . . . . .	73
A.6	security.tf Terraform File . . . . .	74
A.7	network.tf Terraform File . . . . .	75
A.8	EC2 Instance Setup Script . . . . .	76



# List of Abbreviations

<b>AMI</b>	Amazon Machine Image
<b>AWS</b>	Amazon Web Services
<b>BLE</b>	Bluetooth Low Energy
<b>CLI</b>	Command-line interface
<b>CSP</b>	Cloud Service Provider
<b>EC2</b>	Elastic Compute Cloud
<b>HA</b>	Home Automation
<b>IaC</b>	Infrastructure-as-Code
<b>IaaS</b>	Infrastructure-as-a-Service
<b>IIoT</b>	Industrial Internet of Things
<b>IoT</b>	Internet of Things
<b>KPI</b>	Key Performance Indicator
<b>LAN</b>	Local Area Network
<b>LR-WPAN</b>	Low-Rate Wireless Personal Area Networks
<b>MAC</b>	Media Access Control
<b>MFA</b>	Multi-Factor Authentication
<b>MQTT</b>	Message Queuing Telemetry Transport
<b>NIST</b>	National Institute of Standards and Technology

*List of Abbreviations*

---

<b>OS</b>	Operating System
<b>QR</b>	Quick-repsonse
<b>RBAC</b>	Role-based Access Control
<b>RFID</b>	Radio-Frequency Identification
<b>RPI</b>	Raspberry Pi
<b>RTT</b>	Round-trip Time
<b>SBC</b>	Single-board Computer
<b>SHS</b>	Smart Home System
<b>SIEM</b>	Security Information and Event Management
<b>SSH</b>	Secure Shell
<b>TOTP</b>	Time-based one-time password
<b>TVOC</b>	Total Volatile Organic Compounds
<b>VM</b>	Virtual Machine
<b>VPC</b>	Virtual Private Cloud
<b>UEBA</b>	User and Entity Behavior Analytics
<b>ZHA</b>	Zigbee Home Automation
<b>ZT</b>	Zero Trust
<b>ZTA</b>	Zero Trust Architecture
<b>ZTNA</b>	Zero Trust Network Access

# 1 Introduction

The rise of smart homes related to the surge of Internet of Things (IoT) devices, promoting convenience and control in everyday life [1] has seen a substantial development in the last decade. One of the chief problems that impact smart homes and their networks is the process of ensuring not only the security of external communication, but also within the network itself, with respect to the user’s privacy. In the context of this work, the Zero Trust security model, based on the principle of “never trust, always verify”, is intended to oppose the implicit trust placed on the network perimeters [2] and thus provide more efficient security measures for smart homes. This paper is intended for readers to get acquainted with the topic of integrating Zero Trust for smart homes and home automation, as well as ones researching a way to alleviate the security of a home network.

## 1.1 Motivation and Objectives

Given the fact that in the current age of information technology, the interconnectedness of all kinds of devices from smartphones to different appliances through the Internet has become an expected commodity in people’s homes. This phenomenon makes it convenient for consumers to transform parts of or whole living quarters into an IoT-based Smart Home System (SHS), allowing users to automate, remote monitor and control their IoT-integrated living space with different goals in mind ranging from better privacy to higher quality of life through a more intelligent environment [3].

However, as the home itself is being regarded as a network, this integration of IoT devices is marked by a latent need for their network security. In light of this, a common example is that once a smart device has been paired within the home network, it is assumed to be secure and without any need for authentication of its identity or status. This, in turn, opens a possible backdoor for threat agents to compromise the security of the device and of the smart home itself.

To tackle the potential vulnerabilities of a perimeter-oriented network topology, this work proposes and explores the integration process of the Zero Trust security model in a smart home environment. More notably, this thesis aims to give insight, from both a security and a usability point of view, on whether such an environment is more appropriately hosted locally or by utilizing a cloud service provider. Therefore, the main research questions this bachelor's thesis aims to answer can be formulated, as follows:

- How well is the Zero Trust security paradigm integrated into a smart home environment?
- Should a Zero Trust smart home network be hosted locally or in the cloud?

## 1.2 Structure

The first chapter introduces the reader to the main topic of the thesis. It also outlines the motivation behind delving into the subject, as well as clarify the research questions and objectives of the bachelor's thesis.

The second chapter opens with a comprehensive overview of the origins and core principles behind smart homes and Zero Trust, while later moving on to describe the fundamental technologies and concepts with regard to the implementation of the two smart home architectures and their practical implementations.

Following this, the third chapter provides an overview of the theoretical work with the Zero Trust model in the context of smart homes, while subsequently describing the conception of the on-premise and cloud-based ZT security architectures. The chapter also introduces primary and secondary metrics, relating to the security and overall user experience respectively, which have been defined for the context-based comparison of both implementation scenarios. Furthermore, this chapter identifies the limitations of these architectures in a brief overview.

The fourth chapter provides a short summary of the features of the implemented SHS solutions, while later explaining in more detail the process for each of the practical implementations. Even though both environments have an initially identical Zero Trust configuration, as part of this work, they deliver unique insights into the setup and device integration process, as outlined in the last section of this chapter.

Regarding the analysis of the work, the fifth chapter begins by reminding the user of the main research questions of this thesis and comprises a more in-depth synthesis of the

results of this work and a collection of the insights gathered from working on the practical implementation of both environments. Following up on that, this paper provides a general assessment of the implementation of both the locally operated and cloud-operated smart home environment, providing an answer to the aforementioned research directions. Subsequently, concluding the chapter, an assessment based on the evaluation metrics from Chapter 3 is made, estimating the success of both SHS environments.

Finally, the last chapter of this work starts with a thorough summary of the whole thesis and its results. Both of the theoretical and the practical findings from each chapter are recapitulated, pointing out the positive and negative aspects. Moreover, the last parts of the chapter provide an outlook on different directions the work could take, regarding the idea of another ZT security architecture based on the concept of edge computing, as well as the further development of the cloud-based SHS solution using a collection of AWS IoT services.

## 2 Fundamentals

This chapter of the bachelor’s thesis serves to introduce the fundamental concepts for the inception of smart homes, such as the Internet of Things, home automation and also edge computing. Moreover, it delves into the origins and main principles of the Zero Trust security model, while later describing the essential concepts and technologies necessary for understanding the thesis work.

### 2.1 Smart Homes

In today’s digital world, the practice of devices and sensors being connected over the Internet to integrate real-time monitoring and a degree of smart assistance with appliances in a given environment is becoming more prevalent, especially into private residences. This trend marks the rise of the smart home, also known as a smart home system (SHS), with the goal to automate certain mundane tasks and improve the quality of life in a household. By the end of the 2020s, the number of smart homes is projected to reach 1.9 billion people [4], indicating a growing market for the integration of smart appliances at home.

With regard to these numbers, the attack surface of home networks is consequently bound to expand as well, making them a prime target for malicious actors, usually hackers to collect user data and thus compromise user privacy. Before diving deeper into the topic of smart home security, this section provides a brief overview on the topics of IoT and Home Automation as foundational concepts for smart homes, to allow the reader to understand the bigger picture.

#### 2.1.1 Internet of Things (IoT)

At first, the term “Internet of Things” (IoT) referred to the effort to combine radio-frequency identification (RFID) and different types of sensors to provide additional

functionality to generic objects [5]. However, over the years the concept of IoT has proceeded to expand on the idea of connectivity and evolved into a network of interoperable devices, which function as an amalgamation of household, transport, and monitoring devices with sensors, which contributes to the tremendous exchange known as big data, thus defining the modern foundations for the IoT [6]. A typical example for an IoT device in the context of smart homes is a mobile TVOC [7] air quality sensor, which gathers the raw data about the temperature, humidity and TVOC levels in the air and transmits it to a designated network gateway connected to the home network [8].

### 2.1.2 Home Automation

Home Automation (HA) is broadly defined as the use of sensors and other smart devices together with predetermined behaviour with the objective to enable a domestic environment to autonomically and remotely accomplish different sets of operations [9]. This notion, in conjunction with IoT technology and HA-enabling software, e.g. Home Assistant [10], is what constitutes the basic structure of a smart home system. Figure 2.1 depicts a high-level diagram of a smart home with different IoT-based home automations. It should also be noted that while each node can be treated as a separate operation, HA is most successfully realized with a set of various automations.

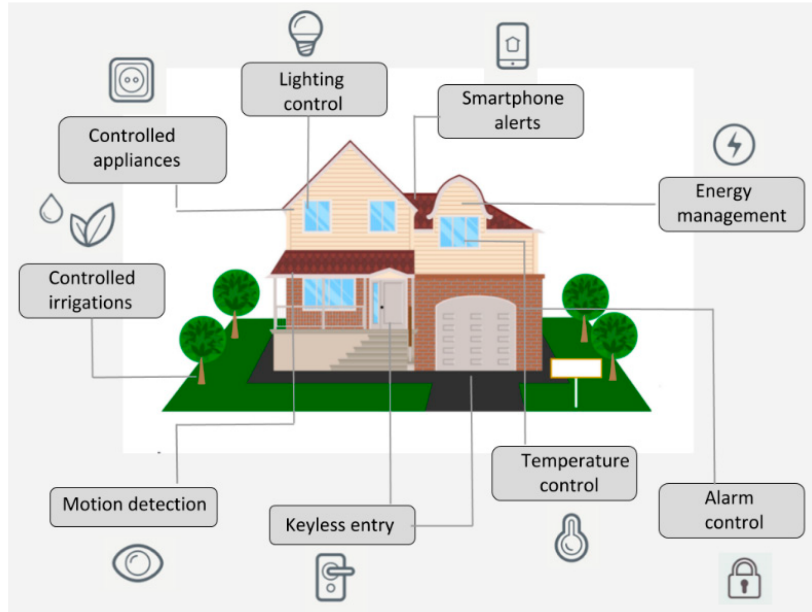


Figure 2.1: A smart home with several examples of HA [11]

For instance, temperature control can be implemented via a smart thermostat device, which regulates the internal temperature of the home according to the information gathered from motion sensors. Moreover, a simple energy management automation can monitor all controlled smart lights and appliances in the house, sending a weekly notification from the HA software with the estimated electricity costs. These are two basic examples, and yet the depth of home automation is in actuality merely limited by the amount of smart devices in the home and the creativity of end users.

### 2.1.3 Edge Computing

Being often mentioned together with IoT, Edge computing involves the implementation of all computing tasks on the “edge” of the network, which can be defined as any computing and network resource between the source of the data and a cloud data centre [12]. The premise of this approach is that, as the majority of data is produced on the edge of the network, it would be more efficient for it to be processed there. As stated in [12], the results based on a proof-of-concept platform for face recognition are a significant reduction of response times and energy consumption. In this context, the idea to implement edge computing in a smart home security architecture offers interesting



insights in regard to security and safety [13], however as this thesis primarily focuses on the Zero Trust security model for a smart home network these are only mentioned in the “Outlook” section of the last chapter.

## **2.2 Zero Trust Security Model**

The zero trust security model is defined by the assumption that a network has already been breached and as a result, trust granted to users within the network must continuously be subject to reevaluation. The function of this security paradigm is to limit the implicit trust, which comes with perimeter-based security measures, and thus adjust the network’s security to the constant analysis of the risks associated with its users and resources.

A typical zero trust architecture (ZTA) would include the provision of resources (e.g. data, services, applications) for subjects in the network on an as-needed basis, as well as the constant authentication of the identity and security of each of those subjects. In stark contrast to that, traditional network architectures, focused on the defence of their perimeters, provide authorized subjects with a broad array of resources once they are in the network, making way for lateral movement within [14].

### **2.2.1 Origins**

The concept of Zero Trust security was first devised to mitigate the limitations coming from perimeter-based security models. The term was first formalized by the analyst John Kindervag in 2010, as he came to the following conclusion – it is possible for attackers to take advantage of and subsequently infiltrate an organization’s network to access the company’s sensitive data, all because of the implicit trust placed on the internal network traffic [15]. This, in turn, prompted the paradigm shift from placing trust in both the outside and inside perimeters of a network into making sure that the separate entities, e.g. users, applications, etc., are secure. Since its inception, the Zero Trust model has become widely recognized, with many organizations aiming to overview and adopt its principles with the intention of strengthening their own security posture. In 2020, the National Institute of Standards and Technology (NIST) published the NIST Special Publication 800-207, which provides a detailed overview of ZT architectures as a whole, including their logical components, possible deployment scenarios and related threats,

along with a framework for Zero Trust architecture design [16].

### 2.2.2 Core Components

The Zero Trust model integrates several core components to achieve the task of strengthening network security and reduce the number of unauthorized access occurrences and the size of attack surface. As thoroughly described in [15], the main components of the ZT model are as follows:

#### Multi-Factor Authentication (MFA)

Considered foundational for the implementation of Zero Trust, Multi-Factor Authentication is the practice of utilizing two or more authentication factors to verify the identity of users and thus grant access to certain resources. As part of this process, authentication factors are divided into one of three groups to identify users, outlined in [17]:

- Something the user alone knows (e.g., a password or PIN)
- A physical credential only the user can provide (e.g., a one-time passcode)
- An inherent trait of the user (e.g., a fingerprint or face scan)

Figure 2.2 depicts the overall workflow of MFA in its most common configuration with two authentication factors. Should one of the factors not be verified correctly, this means that the identity of the user cannot be properly confirmed, in turn making it invalid. MFA benefits the security of the Zero Trust model by introducing an additional factor, making it harder for attackers to replicate and assume the identity of users. Moreover, should one of the factors be compromised, by guessing the password for instance, MFA lowers the likelihood of a user's account being taken control of.

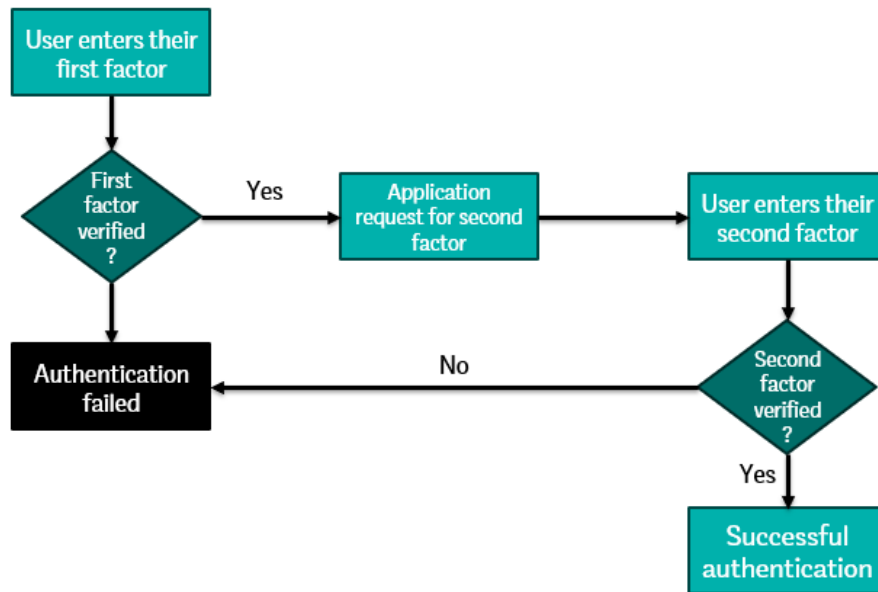


Figure 2.2: Multi-Factor Authentication Workflow

Commonly used as a complementary authentication factor to passwords, time-based one-time passwords, or TOTP, is an authentication standard where users receive a random constantly changing code, either through a mobile application or a hardware device, which confirms the user's identity, as long as the secret code and the current time between the user device and the authentication server are roughly the same [17]. This authentication factor is specifically introduced, since it plays a crucial role in the MFA process of the practical implementation in Chapter 4.

### Micro-Segmentation

Micro-segmentation is the process of dividing a network into smaller and isolated network segments, which are protected by their own security settings. With the view to restrict the lateral movement inside a network, micro-segmentation divides the network into zones based on separate policies. Access to a segment of the network is only granted predicated on the identity and predefined user behaviour.

This serves to minimize the attack surface for each segment of the network, however, more importantly, to contain eventual breaches in a single network segment. By dividing the network into several small segments, this also makes anomalies in the network much more visible.

### **Identity and Access Management (IAM)**

The IAM component in the Zero Trust model controls which specific resources only authorized users can access in the network. It ensures the user's identity is verified correctly and utilizes role-based access control (RBAC) to ensure that the access a user has been granted is well within their role in the organization.

The benefits of such a process vary from centralizing the control over users' access to resources to enhancing the synergy between the other ZT model components by enforcing least privilege access and allowing straightforward monitoring of user behaviour.

### **Least Privilege Access**

The principle of least privilege is defined by the provision of the minimum permissions to users required to finish a certain task. This granular access is adjusted based on a user's role within the organization and is usually enforced by the IAM system. Permissions within the network are dynamically adjusted, meaning that they are updated in real-time, according to user behaviour and their security status. This ensures that no standing user privileges can compromise sensitive information inside the network.

This aspect of the Zero Trust framework serves to minimize sensitive data exposure, in the case of user credentials being compromised.

### **Continuous Monitoring and Analytics**

While not considered a security feature in itself, the act of continuous monitoring is just as essential as the rest of the other components by providing real-time information for any unusual network activity, such as change of user location, revoked access, change in device health, etc. By utilizing this component, any potentially malicious anomaly in the network is quickly detected and managed accordingly – usually by way of User and Entity Behavior Analytics (UEBA) and Security Information and Event Management (SIEM) tools.

This anomaly prevention process stands to improve the visibility of user activity within the network, allowing for the early detection of threats through logging and alerting for anomalous network activity.

## 2.3 Fundamental Technologies and Concepts

This section aims to provide additional information about a collection of concepts and technologies, which are not a part of any specific category, however are just as relevant for this thesis work as the aforementioned topics around smart homes and Zero Trust.

### 2.3.1 Home Assistant

Originally developed by Paulus Schoutsen in 2013, “Home Assistant” is a free and open-source platform for home automation [18]. It serves as a central management hub for IoT devices and various services, where they are connected through one interface. Home Assistant supports all major smart home solutions, including but not limited to Amazon Alexa, IKEA TRÅDFRI, Zigbee Home Automation (ZHA), etc.

It consists of 3 main components – Operating System, Supervisor, and Core, as depicted in Figure 2.3. The operating system is the base Linux environment, which runs both Core and Supervisor. On the other hand, Supervisor is responsible for managing the operating system, while Core interacts with users and the variety of IoT devices and services. [19]

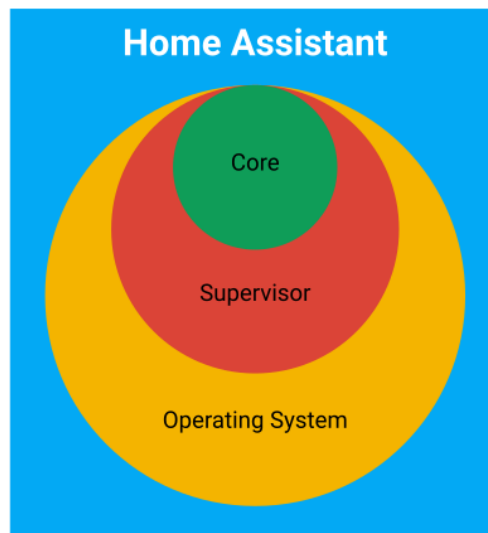


Figure 2.3: Home Assistant architecture overview  
[19]

### 2.3.2 Raspberry Pi

Originally intended for students to further develop their computer science skills, the “Raspberry Pi” (RPI) is a series of single-board computers (SBCs) with over 60 million devices sold [20], [21]. The goal of this project is to provide a credit card-sized and mobile computing solution (Figure 2.4) with all relevant building blocks (Figure 2.5) and enough processing power to act as a standalone minicomputer workstation [22].

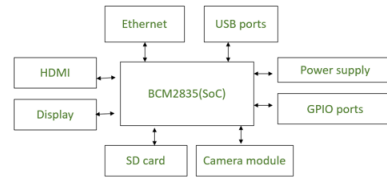
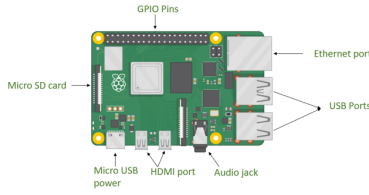


Figure 2.4: Architecture of a Raspberry Pi [22]      Figure 2.5: Main blocks of a Raspberry Pi [22]

Due to its large support community and ubiquitous usage in the field of smart homes, the RPI in the context of this thesis work is used to run the Home Assistant Operating System [23]. It is also responsible for the gateway-based communication with IoT devices and services, assuming the role of a central server for the on-premise smart home setup.

### 2.3.3 Communication Protocols

#### Zigbee

Based on the IEEE 802.15.4 standard for Low-Rate Wireless Personal Area Networks (LR-WPAN) [24], Zigbee is a wireless communication protocol, which has been specifically designed for small-scale personal area networks with low-power consumption devices in mind. In more detail, the Zigbee protocol is widely utilized in the integration of light bulbs, sockets, thermostats, and other smart devices, having established itself as a standard in home automation networks.

The devices within a Zigbee network are all dependent on a Zigbee gateway for their communication, which is connected to the Internet and serves as their controller node. However, they form a so-called mesh network, as shown in Figure 2.6 – where the devices

themselves act as routers, receiving and forwarding network requests – thus ensuring the stability and reliability of the network. For a Zigbee setup to be regarded as complete, it must include 3 core components: the Zigbee gateway, which acts as an intermediary between the Internet and the smart devices, a coordinator device, usually either a smartphone or a tablet, to control and monitor the network remotely through a coordinator app (e.g. Home Assistant), and at least one end device as a component of the network itself [25].

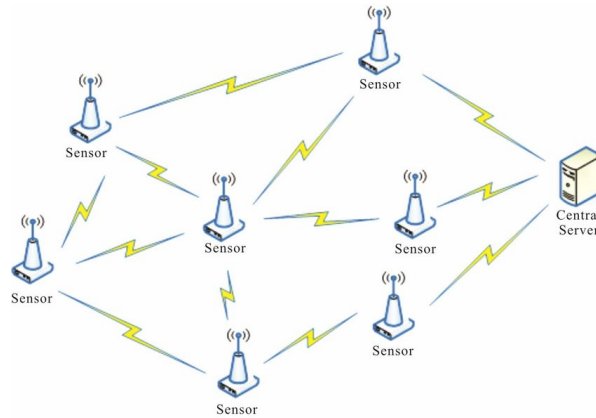


Figure 2.6: Zigbee mesh network  
[26]

### Bluetooth Low Energy

Developed as an upgrade over Bluetooth 4.0 and above, Bluetooth Low Energy (BLE) has become the de facto Bluetooth technology for low-power devices [27], also being adopted as a communication protocol for IoT devices as a result.

In contrast to Zigbee, BLE does not require a separate gateway to establish the connection to smart devices, since the user device (usually a smartphone or an RPI) can already communicate per Bluetooth. This small difference makes the use of BLE devices, especially for its wide compatibility and straightforwardness, more convenient to set up and use than Zigbee. However, as the Zigbee protocol was designed with home automation in mind, BLE devices are best suited for infrequent communication between battery-powered devices, such as wireless headphones or smartwatches, which incorporate a wide variety of sensors [28]. Conversely, Zigbee devices most commonly consist of one or several sensors with a specific niche application, e.g. room environment metrics.

## MQTT

The MQTT (Message Queuing Telemetry Transport) protocol is a Publish/Subscribe messaging protocol. Due to it being a lightweight protocol with a simple implementation, it is possible to run on devices without an operating system (OS), making it ideal for IoT applications. The central part of the MQTT network is the MQTT *broker*, which acts as the server, receiving and distributing the messages between MQTT clients (either mobile devices or other backend systems) [29].

The client communication is based on the concept of *topics* – as the data is organized in a tree-based topology, a topic is essentially the leaf node, which saves the data transmitted by the device. IoT devices then *publish* their data over a corresponding topic for the broker to store, which in turn would forward it to all devices currently *subscribed* to the given topic.

Figure 2.7 depicts an example of the MQTT Publish/Subscribe architecture, where a temperature sensor publishes its updated temperature data to the ‘*temperature*’ topic. The broker then publishes the changed value of the data for the other two clients, which are subscribed to that particular topic.

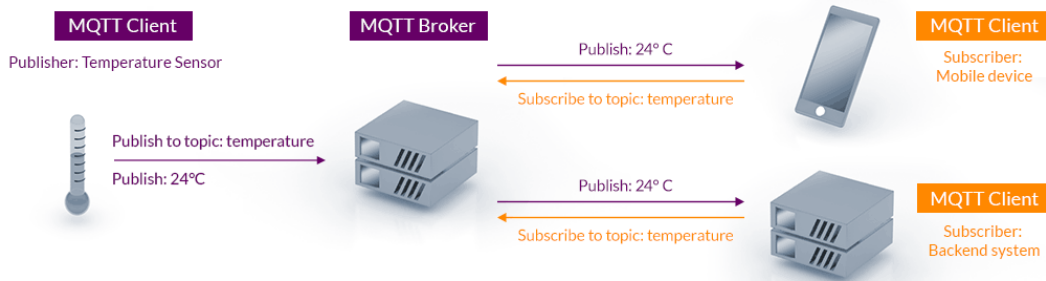


Figure 2.7: MQTT Publish/Subscribe Architecture [30]

### 2.3.4 Infrastructure-as-a-Service

Infrastructure-as-a-Service (IaaS) is a business model, which delivers hardware resources, such as processing power, storage and network resources, on virtual servers over the Internet. By employing the services of an IaaS provider, users benefit by not having to provide and manage physical IT infrastructure and instead work on the deployment and



support of their own applications. This makes clients solely in charge of the software stack. [31]

### **Elastic Compute Cloud**

Elastic Compute Cloud (EC2) is the IaaS solution of Amazon Web Services (AWS), setting the norm for public cloud computing solutions in 2006. EC2 offers virtual servers that allow users to host a Linux or Windows OS of their choice, with all compatible software accessible as well. According to the users' needs, EC2 offers a plethora of instance types [32] through Amazon Machine Images (AMI), which serve as OS images for the virtual machines; the deployment of virtual machines (VMs) and virtual networks can then be finetuned and expanded upon with storage devices, load balancing, an autoscaling configuration, etc. [33]

### **2.3.5 Infrastructure-as-Code**

The Infrastructure-as-Code (IaC) paradigm contributes to the IaaS model by enabling the allocation and management of resources for in the cloud using template source files. This involves the automation of the infrastructure configuration with a view to minimize errors and establish a convention for the resources of virtual servers to be restored and reproduced identically. As the adoption of those principles is based on configuration scripts, practices from the field of software development, such as version control and syntax checking, have been integrated into the workflow of IaC tools. [33]

### **Terraform**

Terraform, developed by HashiCorp Inc., is an IaC tool that can be used to define both cloud and on-premise resources in human-readable configuration files for managing IT infrastructure throughout its whole development lifecycle [34].

Since this is a command-line interface (CLI) application, it can be directly embedded into the IaC workflow, enabling the flexibility of provisioned infrastructure with a few script lines. Terraform supports all major IaaS services of cloud providers, including Amazon Web Services, owing to its abundance of both official and community plugins to make compatibility one of its most appealing features. [33]

Utilizing a declarative programming style, the Terraform framework is divided in 3 main

stages: write, plan, and apply, as shown in Figure 2.8. Terraform scripts are written to describe the desired end state of the infrastructure in configuration files, with the code itself partially serving as documentation for the configuration settings. The writing stage defines the resources to be provisioned by the service provider, ranging from the low-level aspects, including, but not limited to the number of virtual CPUs, memory, and storage to the network architecture – CIDR blocks, subnets, inbound/outbound traffic rules, etc. Subsequently, the planning stage provides a preview of the changes for the user.

The last stage of applying the changes provisions the infrastructure to the IaaS provider and updates the state file [35], which is primarily used for storing the bindings between objects in a remote system and resource instances declared in the user’s configuration. The purpose of this is to enable Terraform to know which resources are under its control, when to update and when to destroy them, streamlining the provisioning process [36].

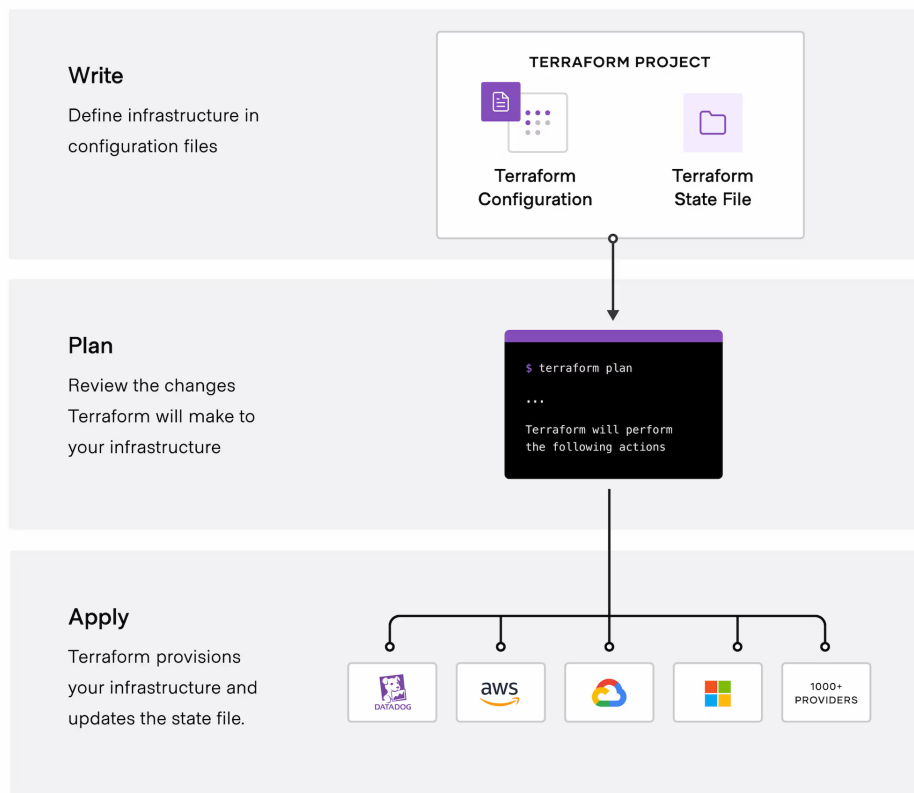


Figure 2.8: Terraform Workflow

[34]

### 2.3.6 Containerization

Containerization is the process of “packaging” a software application with all libraries and dependencies needed to run it into an executable called a *container*. This method is widely adopted to universally deploy applications more swiftly and securely on any IT infrastructure as a single entity. [37]

The entire container entity then runs encapsulated in a separate runtime environment with its own file system, isolated from the host system. A container is launched from a so-called *container image*, which is the amalgamation of the software application with its runtime environment, the OS, all library and file dependencies in a lightweight base system. This means that a *container* is simply an instance of an image, although the two terms are usually used interchangeably. The individual container images are usually distributed over a *container registry*, e.g. Docker Hub, allowing them to be imported locally on the host. [33]

#### Docker Compose

As part of the Docker platform, Docker Compose is the tool for defining and running applications, comprised of several containers. With the goal to simplify the control over these multi-container applications, all configuration is done in a single YAML configuration file, most commonly known as *docker-compose.yml*, where the desired environment for the application has been specified. [38]

Docker Compose interacts with the applications in the Docker CLI, by using the *docker compose* command and its subcommands, allowing users to effortlessly start, stop and configure their applications [39].

Figure 2.9 outlines the overall workflow for Docker Compose. After the configuration file has been defined, Docker Compose will then build and run all the needed images based on it, and then create the environment for the application, including the container setup, any network settings required for the application’s services and occasionally data volumes for persistent data storage provided that containers are restarted [40].

## 2 Fundamentals

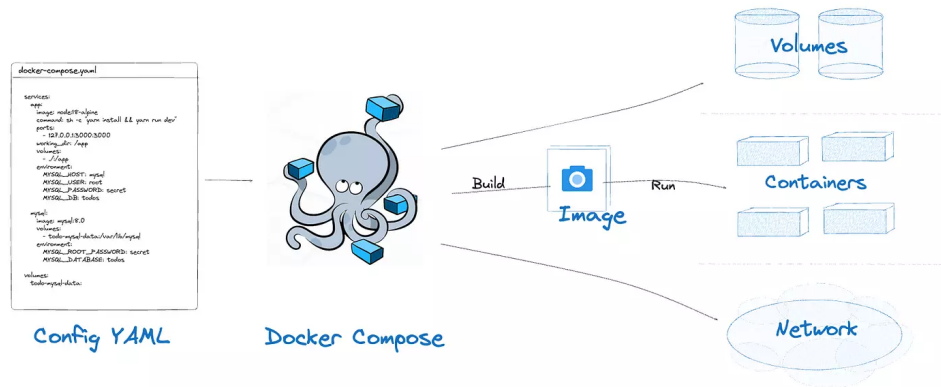


Figure 2.9: Docker Compose Workflow  
[40]

## 3 Theoretical Part

After having introduced the concepts of smart homes and Zero Trust security to the reader, together with the other fundamental concepts for this work, the next chapter will primarily focus on the theoretical aspects of the thesis. The chapter itself is divided into three sections.

The opening section introduces the security concepts for both network architectures, together with their adoption of Zero Trust principles in smart homes. Subsequently, the reader is provided with a detailed overview of each of how these setups should operate and serve to enhance the security of a smart home setup.

The next section of the chapter is dedicated to the introduction of two categories of evaluation metrics for both architectures. With regard to the research questions, the first set of metrics, also known as the primary metrics, is centered around the security factors, derived from the aforementioned Zero Trust components in Chapter 2 and the CIA triad[41] with a view to evaluate the security of the smart home environments. The secondary set of metrics focuses on the subjective matter around the implementation and ease of use of the smart home architectures, introducing factors such as the duration of the start-up phase, platform support and complexity.

The final parts of this chapter are dedicated to the limitations and challenges of the aforementioned smart home environments. It criticizes their flaws from an implementation and security standpoint, pinpointing the particular challenges of implementing Zero Trust practices in smart homes.

### 3.1 Proposed Architectures for Smart Home

With the aim to provide a more robust security model for smart home networks and for its measures against implicit trust, the Zero Trust security model has been chosen to be integrated in said smart home architectures. Both the locally hosted and the cloud-hosted environments have a similar setup configuration, as they essentially depend on a host device (either an RPI or a VM) to run a home automation software as per the user's choice, e.g. Home Assistant, which serves to facilitate a default smart home environment. After the initial setup, both environments have been conceptualized to first adopt their Zero Trust security configuration before integrating an IoT device in their SHS. The user then accesses the home automation instance by means of using the client page of the HA software through a Multi-factor Authentication (MFA) login mechanism with a TOTP token as a second factor or directly connect to the host device, establishing an SSH connection. While the former approach is the default way for users to interact with the SHS and the IoT devices within, the SSH tunnel is meant as a way to manage the host file system and extend configuration.

Following this, the user integrates their IoT devices into the smart home network, as the process for said integration deviates, depending on whether the environment is on-premise or in the cloud. Figure 3.1 depicts a concept diagram of the template for both smart home environments, as previously described.

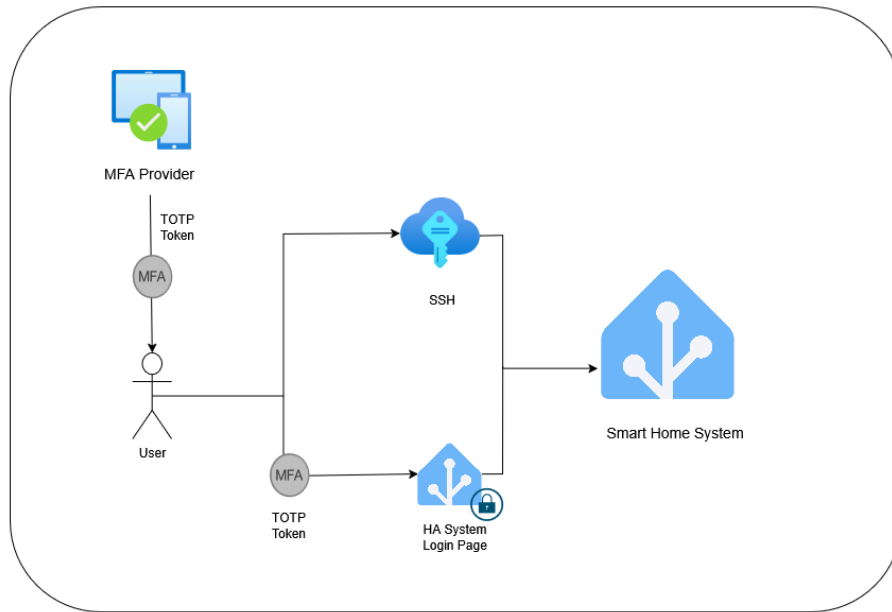


Figure 3.1: Smart Home Environment Concept Diagram

### 3.1.1 On-Premise Architecture

When considering the structure of an on-premise smart home architecture, it should be noted that its hardware comprises three synergizing main parts: the host device, gateway, and client device, as depicted in Figure 3.2.

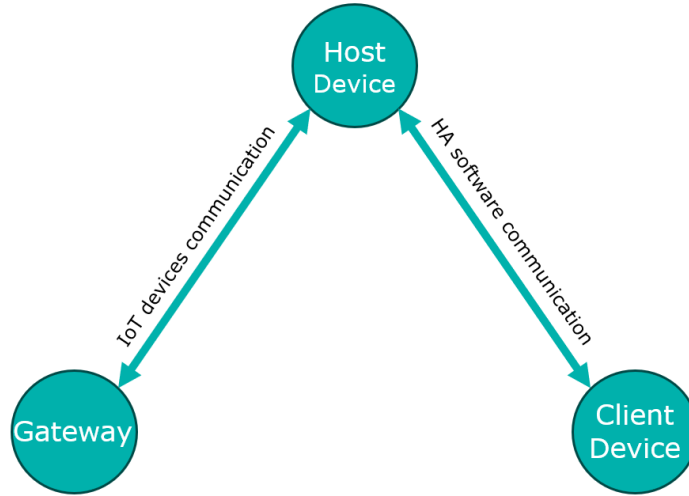


Figure 3.2: Main Components of a Smart Home Architecture

The host device is the one to operate the HA software, either as an OS in the case of Home Assistant[10] or as a separate application and communicate with the user device. In the entire setup, it plays the role of the central hub, over which the communication between the IoT devices and the client device is established. Secondly, the gateway is responsible for the communication between the IoT devices and the host device, having no communication with the client device. Depending on the communication protocol over which the smart devices communicate, e.g. Zigbee, the gateway type may vary, as there is no universal gateway for all IoT devices. Together with the host device, they are the additional hardware required to get started with a SHS. As for the client device, it is usually a smartphone or a laptop, which connects to the personal area network at home. It allows the user to connect to the HA software instance running on the host device and either initiate actions with the IoT devices, e.g. adjust the colour temperature of a smart light bulb, or access information from their sensors, for instance room temperature from a temperature sensor.

Collectively, these three components create the basic smart home architecture, which, when integrated into the home network, provide the necessary environment for IoT devices to connect to. Integrating the IoT devices is usually done via a pairing process, depending on the protocol. The general principle is that a smart device will broadcast its presence and characteristics over the communication protocol (e.g. Zigbee) in the proximity before being paired. The pairing process is then initiated by the host device



by sending a broadcast over the required gateway that it is available to pair. Once it picks up the IoT device's broadcast in a close radius, the process is then finished with both the IoT and the host device being successfully paired. Some IoT devices require the entering of a unique identifier pairing code for cases where there are several instances of the same device, as described in [42].

#### **On-Premise Concept**

With regard to the concept of a Zero Trust architecture hosted locally, it has to be said that when conceptualizing the architecture, not all implementations of implicit trust principles in the network should be omitted. Instead, a more stable and user-friendly approach would be to combine the best aspects of both the Zero Trust and perimeter-based security principles for easier integration into the home network. Considering this, the core features of this architecture can be defined, as follows:

- **Multi-Factor Authentication**

Multi-Factor Authentication is the authentication method of providing 2 or more credential types for user authentication. The credential types are based on information only the user should know (e.g. a password), own (e.g. a confirmation of identity over a client device) or identify as (e.g. fingerprints). Being one of the key principles of Zero Trust, the continuous authentication can be implemented using MFA for all users of the HA software. With this approach, each time a user logs in to manage or check the status of an IoT device over the HA software, e.g. Home Assistant, their identity can be proven with an additional factor, without implicitly trusting the entity logging in with a correct password as the user. When considering what additional factors are to be added to the authentication process, a more straightforward combination is the tuple of a password and a TOTP through an MFA service provider, as shown in Chapter 4.

- **Least Privilege Access**

With Least Privilege Access, users and entities are granted the minimum level of access within a network to complete their tasks. Implementing it for users in the HA platform can reduce the impact an attacker's breach has on the smart home network by reducing the amount of information and control a user profile has for certain IoT devices. In this context, the implementation of said feature depends on the features of the HA platform.

- Firewall Rules over a Remote Tunnel

A remote tunnel is in its entirety a proxy name for the default address of the HA instance in the user local area network (LAN). While the access to the HA platform instance should mostly be achieved from the LAN in an on-premise architecture, integrating a remote tunnel over the encrypted HTTPS protocol and enforcing a set of firewall rules provides a trade-off for being able to access the HA software. Depending on its capabilities, these firewall rules can range from restricting the access to the remote tunnel to a certain range of IP addresses to implementing an IP address ban on users after a number of login attempts has been exceeded. To achieve this in a local environment, it is crucial that the HA platform supports the integration of additional applications, for instance through add-ons, making it that more straightforward for users to configure.

- Tunnel Analytics & Alarms

Moreover, the implementation of a remote tunnel should allow users to track general and web traffic analytics, such as number of requests sent, unique visitors over IP, etc. How this is beneficial for the security of the network is not that it directly prevents, but tracks, patterns of malicious activity. Combined with a set of alarm rules which either further restrict the access to the remote connection or simply shut off the connection, should a malicious attack pattern becomes more prominent over the tunnel, this makes the tunnel an important asset for the security of a local smart home network.

#### 3.1.2 Cloud Architecture

While the on-premise architecture requires several pieces of hardware as a base, setting up a smart home cloud architecture is more straightforward, as it requires just a client device with a stable enough internet connection. The hardware part which acts as a host device comprises a virtual machine provided by a cloud service provider, e.g. Amazon Web Services. However, a more complex aspect of this setup is the provisioning of the hardware and network resources for the VM as a host device.

Additionally, the establishing of a connection of a IoT device with the host device is not as easily done as when pairing in an on-premise architecture. For this purpose, either a virtual home gateway is needed, tailored to the specific needs of the cloud architecture

or, as a broader approach, utilizing a proprietary gateway, which depends on the cloud service provider.

#### Cloud Concept

Regarding the Zero Trust cloud architecture in the cloud, the major conceptualization prerequisite is the same one followed by the on-premise solution – it is by combining the best aspects of the Zero Trust and perimeter-based security, that this smart home cloud concept can be considered constructive. As for the core features of this architecture, they remain largely the same, but are implemented differently from the on-premise architecture:

- *Multi-Factor Authentication*

In the case of using a cloud service provider (CSP) for hosting the smart home environment, it is possible to implement MFA on several instances. For starters, MFA can first be a component of logging into the CSP account of the user. Moreover, MFA can also be applied in the same manner as in the on-premise architecture – by requiring each user profile in the hosted HA software to be authenticated using a TOTP. This multi-layered utilization of MFA should be implemented to serve as a deterrent for unauthorized access to the user's profile. Additionally, the second factor for an MFA process, e.g. the TOTP, should not be stored or delivered over third-party channels, only being accessible using an MFA provider service.

- *Least Privilege Infrastructure Provision*

When designing the cloud architecture for hosting a smart home, the principle of Least Privilege can be applied in the same way as it is in the local HA platform environment. Nevertheless, in the case of AWS as a CSP, the user is allowed to finetune the hardware specifications of the VM as provisioned infrastructure. Using an IaC tool like Terraform, it is possible to allocate just enough resources for the provisioned virtual machine to run effectively, according to the number of IoT devices and their function. Additionally, as the IaC concept was already introduced, it is possible to restrict the incoming and outgoing traffic both over specific ports, implementing a traffic filter of sort, and for certain IP address ranges only, by modifying the configuration scripts for the provisioned infrastructure.

## 3.2 Evaluation Metrics

In the process of working on both of the described architectures, there is an inherent need for performance indicator metrics through which to determine their utility and provide some insights into the following research questions:

- How well is the Zero Trust paradigm integrated into a smart home environment?
- Should a Zero Trust smart home network be hosted locally or in the cloud?

For this purpose, the following group of important KPI measurements related to the research questions have been defined, as depicted in Table 3.1. The metrics themselves are divided into three categories: primary (also security) metrics and secondary metrics. Starting with the security metrics, the first two represent the integrity and availability of devices, as part of the CIA triad [41], since it is absolutely essential that sensor data over the smart home network is immutable and not serve as a way to sabotage the SHS. Moreover, the user should be able to fully access and modify their SHS, including but not limited to sensor data, configurations, automations, etc. The following two metrics relate to the ability of the smart home architecture to automatically update its system, and any services or applications currently operating on it, as well as provide users with the ability to create either partial or complete backups of the SHS configuration. The most important aspects of these two requirements are to be seamless and straightforward, regardless of the experience level of users, along with a well-documented setup process. In this context, the Home Assistant HA software introduced in Chapter 2 excels at fulfilling the aforementioned metric criteria [43], [44], [45]. The last of the security metrics is derived from the set of ZT principles defined in Chapter 2. While not directly contributing to the security integrity of the environment, the ability of a SHS to monitor device activity and trigger alarms for certain anomaly patterns, e.g. temperature sensor data fluctuating too rapidly or being outside the typical range, contributes to creating a privacy-conscious environment for users.

With respect to the last three metrics in the table, they are designated as secondary metrics and thus cannot be measured due to them being subjective. The reasons for that are, these measurements depend on a variety of subjective factors, mainly user experience and the functionality of both the smart home and hosting platforms, which are nonetheless just as influential for the adoption of the Zero Trust paradigm in smart homes.

The first one, measuring the duration until a simple ZT smart home instance is operational, is significant in the context of users being able to readily scale their smart home architectures and is most heavily influenced by the level of experience the user has. The latter two relate to the smart home (e.g. Home Assistant) and the hosting (e.g. Amazon Web Services) platforms utilized in creating the HA instance. Despite the fact that these services may offer a great amount of assistance for implementing ZT security, it should not be overlooked that an excess of functionality may cause confusion amongst users. Consequently, it is critical for smart home architectures in the context of Zero Trust to be balanced in terms of platform support and complexity.

Metric	Unit	Description
Device Integrity	%	The ratio of secure and functional devices without known vulnerabilities.
Device Availability	%	The ratio of fully operational devices.
Automatic Updates	Yes/No	The ability for services to be automatically updates without user guidance.
Automatic Backups	Yes/No	The ability for services to provide an automatic backup of the user's configuration.
Device Alarms and Monitoring	Yes/No	The ability of the smart home system to monitor device and user activity and alarm users for anomalous behaviour.
Duration of start-up phase	nil	This metric refers to the time it takes to start from an empty original state to a simple smart home instance implementing ZT.
Platform Support	nil	This metric refers to the amount of mechanisms, tools, and tips for overall and ZT security principles the user has at their disposal.
Platform Complexity	nil	This metric refers to the amount of functionality and features the platforms offers, an overabundance of which may cause difficulties for the end user.

Table 3.1: Metrics Overview Table

### 3.3 Limitations

When considering the limitations of the introduced smart home architectures, it should be noted that the conceptualized security measures are primarily intended to be deployed in a consumer IoT environment and are therefore unfit for environments as part of the Industrial Internet of Things (IIoT) [46]. This notion is in view of the fact that the security challenges as part of industrial environments are different to those in consumer ones on account of the longer lifetime of industrial-grade devices, the larger scale of IIoT device networks and their dynamic interconnection between field, controller devices and servers [47].

As most security approaches for consumer IoT place the emphasis on the security of individual devices and services within the IoT environment, the need to secure connections between devices and their automated tasks is not directly addressed. This is also true for the current iterations of both ZT smart home architectures submitted as part of this work. Whereas these architectures focus on user-centric security measures for these environments, e.g. MFA, Principle of Least Privilege, firewall-based remote access, etc., they do not take into consideration the specific IIoT demands for security to be customized to the resource constraints of devices. [47]

Consequently, this section provides concise insights on the limitations and challenges for smart home implementations, established on the aforementioned on-premise and cloud ZT network architectures, which serve as a base for the implementation and findings assessment in Chapter 5 of this work.

#### 3.3.1 On-Premise Architecture

##### **Plugins as a security risk**

While the on-premise architecture can definitely benefit from the adoption of security-related plugins, it is entirely possible that the plugins themselves contain zero-day vulnerabilities, which can compromise the smart home network. In the context of Home Assistant, these so-called add-ons are primarily open-source software without any security guarantees and could pose a security risk for the smart home network.

#### **Instability of services**

The services used to enforce security measures through plugins can also be slow or unstable due to resource constraints. As the local smart home environment is conceptualized to operate on an SBC, the connection to the remote tunnel may prove unstable or even inactive, prompting a restart of the system. If such events occur one too many times, the boundaries between security and user-friendliness of the on-premise architecture become ambiguous.

### **3.3.2 Cloud Architecture**

#### **IoT Device connection**

As connecting an IoT device with the HA platform in the cloud is not that straightforward of a process, requiring either a virtual or a proprietary gateway, this introduces additional difficulty for new users. From a security perspective, this also expands the attack surface of the network in comparison with the on-premise architecture.

#### **Virtual Machine Setup**

When first setting up the virtual machine in the cloud, a starter script is required to run for it to host the HA software, provided it is operated as a container. If a similar way to the local architecture is chosen to set up the environment through an OS for home automation, this prompts the creation of a specific virtual machine image with the desired configuration, further complicating the cloud environment setup process.

## 4 Practical Part

With the view to implement both Zero Trust smart home architectures, which were previously introduced, this chapter describes their step by step implementation, while also aiming to answer the practical research questions on whether the Zero Trust paradigm is suitable to be integrated into smart homes, and whether a smart home network should be hosted locally or in the cloud.

Both implementations offer a similar security configuration, utilizing multi-factor authentication through the Cisco Duo MFA provider, which offers a simplistic and user-friendly solution [48]. Additionally, the introduction of filtering of the incoming traffic – locally based on the user IP address over the remote VPN tunnel, as part of the Cloudflare Zero Trust service [49] and in the cloud through exclusively allowing traffic over specific ports, as well as the integration of an IP ban further serve as security measures.

### 4.1 On-Premise Architecture Implementation

Beginning with the on-premises implementation, its smart home architecture has been designed to be a purely local one, integrated it into the university network of the FH Aachen and exclusively within its IP address range. Hardware wise, it has been implemented using a Raspberry Pi SBC as a host device, together with the necessary accessories, as depicted in Figure 4.1.





Figure 4.1: Main Components of a Raspberry Pi Starter Set  
[50]

### Installation

Firstly, the first step of the project was utilizing the Raspberry Pi Imager [51] (Figure 4.2.) to load an image of the Home Assistant OS (HAOS) onto an SD card and inserting it into the Raspberry Pi. After RPI Imager had finished loading the OS, the RPI itself was booted with the modified SD card. After a short duration of ca. 10 minutes, the RPI was fully operational running HAOS and hosting the Home Assistant starting page to begin the onboarding, as depicted in Figure 4.3. The HA instance was accessible under the *homeassistant.local:8123* URL under the 8123 default port.



Figure 4.2: Raspberry Pi Imager  
[52]

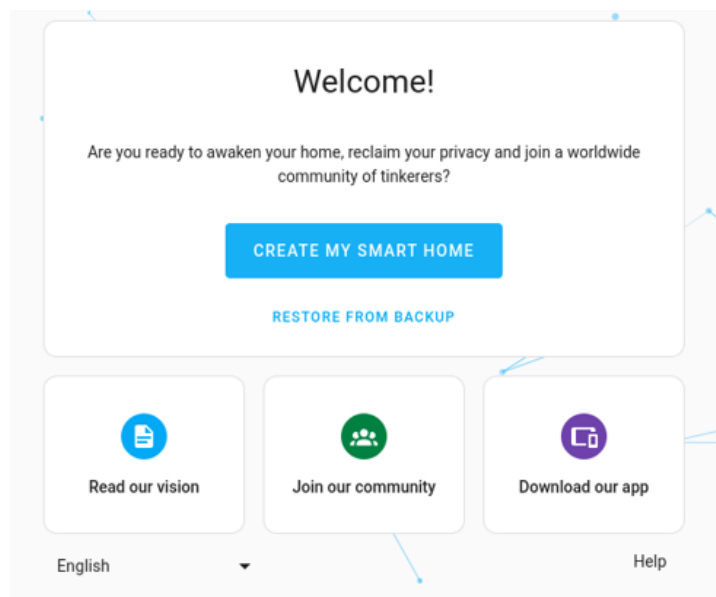


Figure 4.3: Home Assistant Start Page  
[53]

After the installation is complete, the user is walked through the creation of their first profile by being required to enter their username and password, as depicted in Figure 4.4. It should also be noted that while the onboarding process is very straightforward

and easy to follow, it can lack security-conscious messages for passwords. Beyond a simple text under the text field, there is no indication that the user is creating a secure password, as defined in [54]. After a short loading time, the user is greeted by the default Home Assistant dashboard (Figure 4.5.).

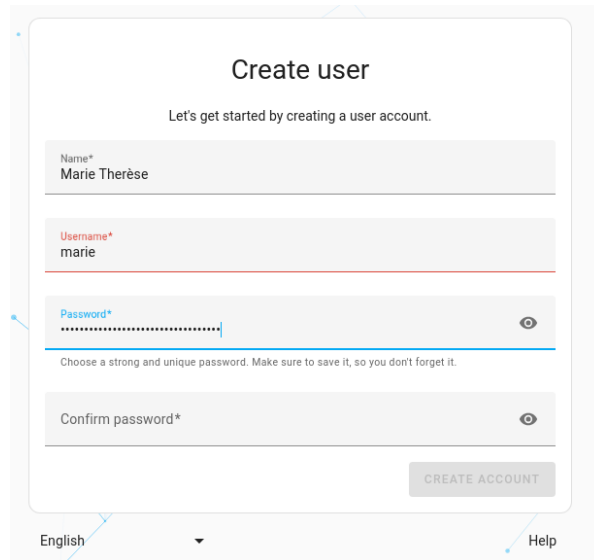
The image shows a 'Create user' form in a light gray box. At the top, it says 'Create user' and 'Let's get started by creating a user account.' Below this are four input fields: 'Name\*' with the value 'Marie Therèse', 'Username\*' with the value 'marie', 'Password\*' with a masked password and a hint 'Choose a strong and unique password. Make sure to save it, so you don't forget it.', and 'Confirm password\*'. Each field has a small eye icon to toggle visibility. At the bottom right is a 'CREATE ACCOUNT' button. At the very bottom, there is a language selector showing 'English' and a 'Help' link.

Figure 4.4: User Profile Creation in Home Assistant [53]

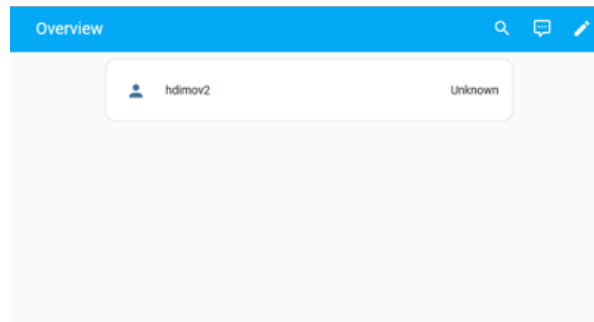


Figure 4.5: Home Assistant Default Dashboard

By delving into the settings, it is apparent that upon the installation completion, there has automatically been a full backup of the system created (Figure 4.6.), establishing an initially straightforward and automatic backup process.

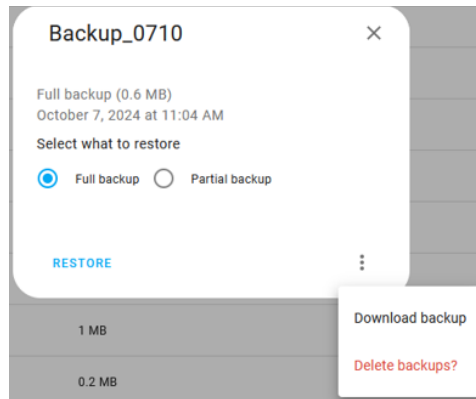


Figure 4.6: Backup Instance Details

### 4.1.1 Configuration

With the view to introduce the configuration of the security features for this environment in more detail, the first one to be configured was the MFA process for the user login. After integrating the MFA provider Cisco Duo for the distribution of a TOTP for each login with a 30-second time limit from the client device, the login process can be described in three simple steps:

1. *Enter username and password.*
2. *Enter TOTP from client device.*
3. *Access login page.*

As the configuration under Home Assistant is concentrated in a single *configuration.yaml* file [55], the integration of an MFA process is carried out in a centralized manner. To describe the process, firstly, the user enters the following lines in the config file, as shown in Figure 4.7. It is noteworthy that the configuration code may differ, depending on the type of second factor added and MFA provider [56].

```
13 homeassistant:~
14   .auth_mfa_modules:~
15     .- type: totp~
16     .- name: "Duo Mobile"~
```

Figure 4.7: Home Assistant MFA YAML Configuration

Subsequently, the user links the MFA provider application (e.g. Duo Mobile) on the client device by scanning a temporary quick-response (QR) code, provided in the Home Assistant settings for the introduction of a second authentication factor, as depicted in Figure 4.8.

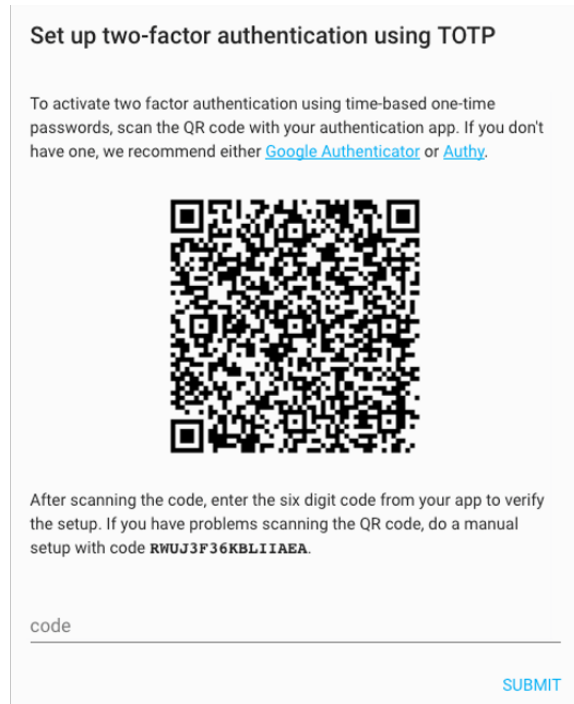


Abbildung 2.11. TOTP QR-Code Einrichtung in Home Assistant

Figure 4.8: Home Assistant MFA QR Code Example

The next feature to be implemented is an IP ban on all IP addresses, which have failed to authenticate themselves properly, exceeding a certain login attempt threshold. This feature is implemented the same way, by entering the lines, depicted in the Figure 4.9.

```
http:~
  ip_ban_enabled: true~
  login_attempts_threshold: 3~
```

Figure 4.9: Home Assistant IP Ban Configuration

After a successful IP address ban a so-called block list file, *ip\_bans.yaml*, has been created in the configuration folder containing the banned client IP address with a timestamp (Figure 4.10.), being updated over time with banned IP addresses. After being

banned, the user receives a 403 status code, forbidding them from visiting the page, as depicted in Figure 4.11.

```
149.201.182.39:~
- banned_at: '2025-01-07T13:19:21.632856+00:00'
```

Figure 4.10: Home Assistant IP Ban Blocklist

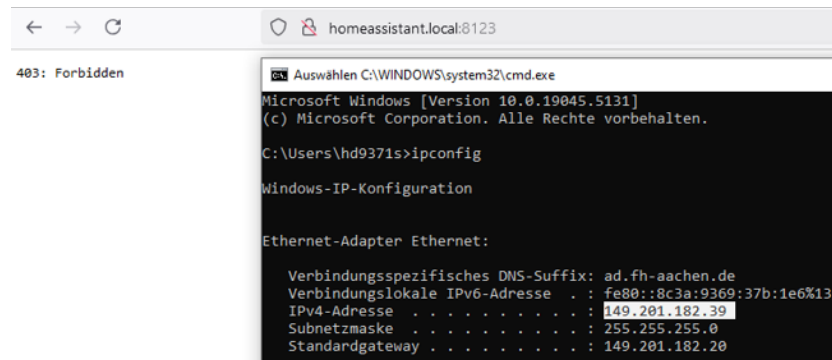


Figure 4.11: Home Assistant User Ban

Another feature implemented in the on-premise environment is the remote tunnel over a VPN connection, connecting the Home Assistant instance with the Internet. Figure 4.12. depicts the added lines to the *configuration.yaml* file as the first step to implement the tunnel. The configuration file should include the list of trusted proxy IP addresses, allowed to access the remote tunnel [57].

```
19 http:~
20   ip_ban_enabled: true~
21   login_attempts_threshold: 3~
22   use_x_forwarded_for: true~
23   trusted_proxies:~
24     - 149.201.182.0/24~
25     - 172.30.33.0/24~
```

Figure 4.12: Home Assistant Remote Tunnel YAML Configuration

By creating the remotely managed tunnel connection using Cloudflare Zero Trust [58], the user receives a Cloudflare tunnel token. Together with the name of the tunnel, they are saved in the Cloudflare tunnel client, utilized by Home Assistant as an add-on [59]. Figure 4.13. represents the details about the tunnel from the Cloudflare dashboard, while

figure 4.14. shows the add-on configuration window with the tunnel URL, Cloudflared tunnel name and token.

**Your tunnels** *Showing 1 - 1*  
Manage the configurations of your existing tunnels.

[+ Create a tunnel](#)

Tunnel name <span>↑</span>	Connector type	Connector ID	Tunnel ID	Routes	Status	Uptime
<a href="#">ha-remote-tunnel</a>	cloudflared	<a href="#">68e0b171-42ee-417f-97e1-be5158dba2cc</a>	ccb5063e-4d71-43f4-9d17-921ad01f7221	--	HEALTHY	10 days

1 - 1 | Items per page: 10 < 1 of 1 page >

Figure 4.13: Remote Tunnel Dashboard

Cloudflared

Options

External Home Assistant Hostname

ha.cyberseclab.xyz

Set this to your domain name or subdomain that you want to use to access Home Assistant.

Additional Hosts

1

Define a list of additional hosts to be routed by the Cloudflare Tunnel.

Cloudflare Tunnel Name

ha-remote-tunnel

Defines the name of the tunnel created for the communication between this service and the Cloudflare edge server. The default value should be fine in most use cases.

Enable Catch-All Nginx-Proxy-Manager

Sets the catch-all service to the "Nginx-Proxy-Manager Community Add-Ons" Add-on.

Cloudflare Tunnel Token

eyJhIjoiM2UyTg2YTU2ZjM2NDZlZTYyOWQwZWY5NGE3NWYyYjciLCJ0IjoiY2NNTA2M2U0NGQ3MS00M2Y0LTlkMTctOTIxYVQwMWY3MjIiwicyY6Iik4y1

When set all other options will be ignored. Use this option if you set up the tunnel with the Cloudflare Dashboard.

Show unused optional configuration options

SAVE

Figure 4.14: Cloudflared Tunnel Configuration

After the tunnel has been configured, the following two firewall rules have been implemented with special syntax over the Cloudflare UI:

- (*ip.geoip.country ne "DE"*): All IP Addresses not registered in Germany are blocked from accessing the tunnel.
- (*not ip.src in uni\_network*): All IP Addresses not part of the university network are blocked from accessing the tunnel.

Figure 4.15. depicts an example of a user not being able to access the tunnel, since they fulfil either of the criteria of the aforementioned blocklists.

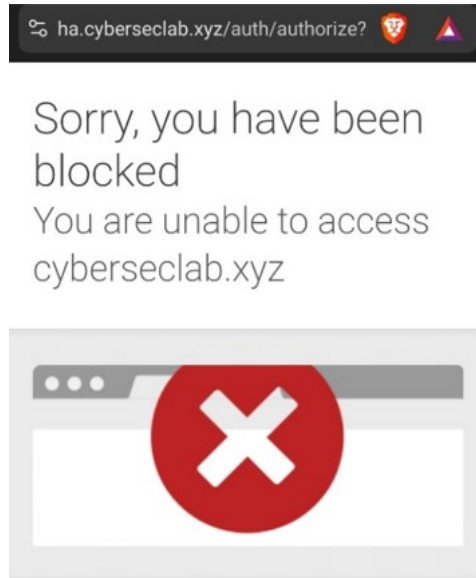


Figure 4.15: Remote Tunnel Access Error

### 4.1.2 Structure & Device Integration

In its entirety, the on-premise architecture is depicted in Figure 4.16., which provides a high-level view of the user and design workflow within the environment. The user has access to the local smart home environment directly either by an SSH tunnel connection or using the default local connection, or the remote tunnel connection by way of a VPN. Inside the local area network, the Home Assistant instance utilizes a Zigbee gateway connection to connect to the set of IoT devices, paired within the network.



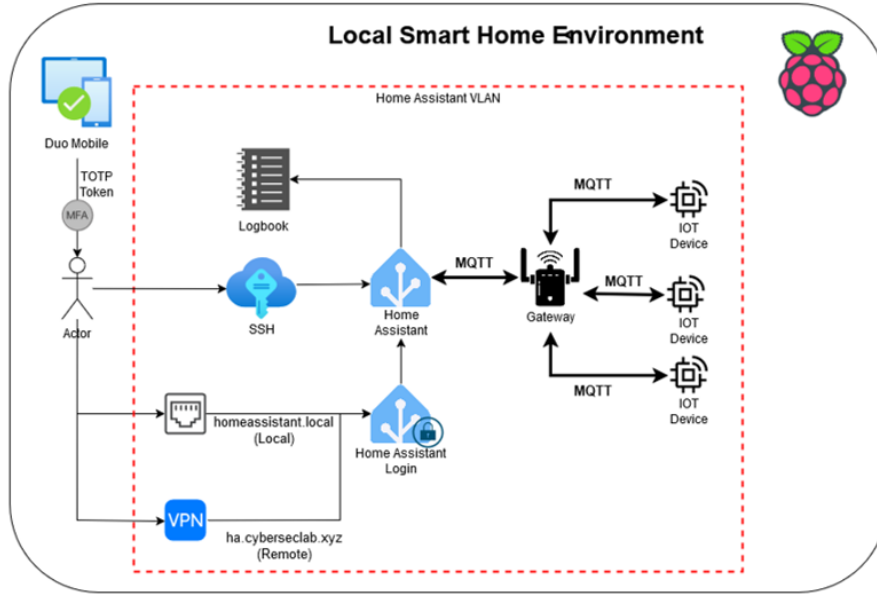


Figure 4.16: Architecture Diagram of On-Premise Environment

The integration of the following IoT devices has been carried out over the Zigbee and BLE protocols:

- 1 TVOC air quality sensor[8]
- 2 IKEA Tradfri LED bulbs[60]
- 1 IKEA Tradfri power outlet[61]
- 1 Flower care plant sensor[62]

All IoT devices except the Flower care plant sensor communicate over the Zigbee with the gateway, while the flower sensor uses the integrated BLE protocol.

With regards to the functionality of the devices, the IKEA Tradfri devices are switched on and off from the Home Assistant dashboard. In addition, 2 automations have been created with this setup. One of the Tradfri light bulbs flashes shortly after the Tradfri socket is switched on, while the other automation has been implemented as a troubleshooting mechanism, flashing when the Zigbee gateway is connected to Home Assistant. Figure 4.17. depicts the aforementioned devices within the local environment dashboard.

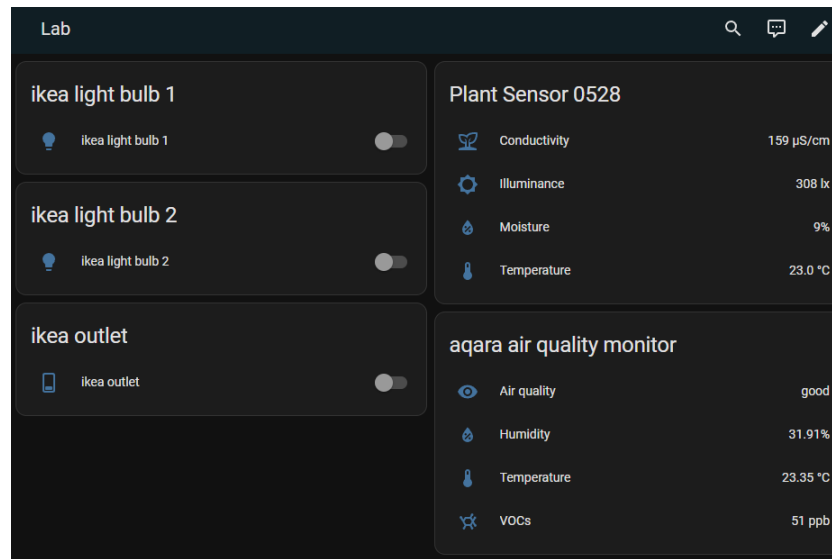


Figure 4.17: Home Assistant Devices Dashboard

## 4.2 Cloud-based Architecture Implementation

Regarding the cloud-based architecture, it has been implemented by utilizing the AWS Elastic Compute Cloud service as a VM, together with Terraform to provision the needed infrastructure and Docker Compose to start the Home Assistant application in a container. Figure 4.18. depicts the concept diagram for the environment's structure. The user has two ways of accessing the HA software instance, either by the MFA login process or over an SSH session.

What separates this environment from the local one in terms of functionality, is the lack of integrated IoT in the environment, since the process of pairing a device with the host device is more complicated for the cloud-based SHS. The "Outlook" section of Chapter 6 addresses how this process could be realized by using AWS IoT services.

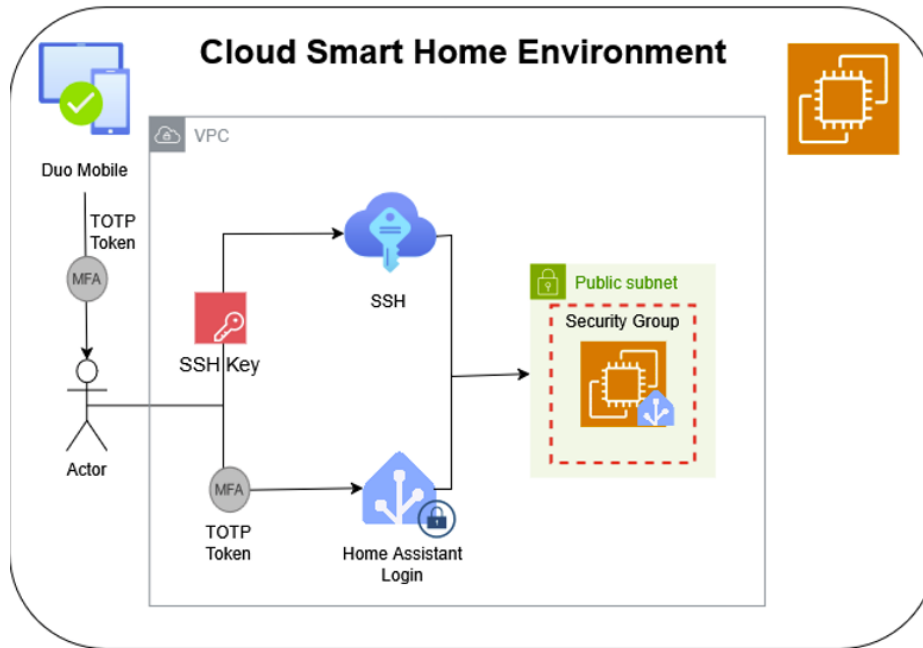


Figure 4.18: Architecture diagram of cloud-based environment

### 4.2.1 Configuration

To review the configuration process, as already mentioned, the configuring the cloud environment is mostly managed over Terraform as an IaC tool. This is achieved by writing configuration files for the instance's provisioned infrastructure – defining the desired state of the system. A simple example is depicted in the *main.tf* file (Figure 4.19) as part of the cloud setup, depicting the definition of the EC2 instance under the *"aws\_instance" "ec2\_vm"* resource and an output of its public IP address every time changes to the terraform scripts are applied [63], as Figure 4.20 depicts.

```
provider "aws" {
  region = var.region
}

// Definition of the EC2 instance as resource with all the r
resource "aws_instance" "ec2_vm" {
  ami           = var.ami
  instance_type = var.instance_type
  key_name      = var.key_name
  subnet_id     = aws_subnet.main_subnet.id
  vpc_security_group_ids = [aws_security_group.ha_sg.id]

  user_data = "${file("ec2setup.sh")}"

  tags = {
    Name = "HomeAssistantCloudInstance"
  }
}

// Terraform output value for the public IP of the instance
output "ec2_public_ip" {
  description = "Public IP for ssh access"
  value       = aws_instance.ec2_vm.public_ip
}
```

Figure 4.19: main.tf of cloud environment configuration

```
Outputs:

ec2_public_ip = "35.159.195.176"
```

Figure 4.20: Terminal output after applying script changes

All the specific information about the EC2 instance and its configuration is accessible from the variables, defined in two separate variable files. The first file, *variables.tf* (Figure 4.21.) declares the variables and provides individual descriptions. On the other hand, the *terraform.tfvars* file (Figure 4.22.) is where all variable values are set, making it the more crucial file of the two. It specifically defines the network as part of the Virtual Private Cloud (VPC) of an EC2 instance and the VPC default components [64], containing the essential data, e.g. the subnet and CIDR distribution, availability zones, AMI image ID and the instance type.

```
// Variable declarations in Terraform (see descriptions)
variable "region" {
  description = "AWS region"
  type        = string
}
variable "vpc_cidr" {
  description = "CIDR block for VPC"
  type        = string
}
```

Figure 4.21: variables.tf

```
// AWS region where resources will be deployed
region = "eu-central-1"

// Default CIDR block for the VPC
vpc_cidr = "172.31.0.0/16"

// List of default CIDR blocks for subnets
subnet_cidrs = [
  "172.31.0.0/20", // eu-central-1a
  "172.31.16.0/20", // eu-central-1b
  "172.31.32.0/20" // eu-central-1c
]

// List of availability zones for subnets
az = [
  "eu-central-1a",
  "eu-central-1b",
  "eu-central-1c"
]

// Public CIDR block (uni network)
public_cidr = "149.201.0.0/16"

// Defined SSH key
key_name = "ha-cloud"

// AMI with OS of the EC2 instance
ami = "ami-0584590e5f0e97daa"

// EC2 instance type
instance_type = "t2.micro"
```

Figure 4.22: terraform.tfvars

As regards the security settings, they are most commonly allocated in terraform through resources, in the same way as the EC2 instance. For instance, to configure the incom-

ing and outgoing traffic for the EC2 instance the user defines the *aws\_security\_group* resource, as depicted in Figure 4.23. In this instance, the security group is connected to the CIDR block of the current VPC to allow the incoming traffic over the ports 22 (SSH) and 8123 (Home Assistant) from a specific public CIDR group of IP addresses through the definition of these rules as additional resources, while allowing outgoing traffic to the constraints of the local network. These settings are also visible from the AWS EC2 instance dashboard, as Figure 4.24. depicts.

```
// Define AWS security group for the ports 22 and 8123
resource "aws_security_group" "ha_sg" {
  name      = "HomeAssistant-SG"
  description = "Allow port 22 and 8123 inbound traffic and all outbound"
  vpc_id    = aws_vpc.main_vpc.id

  tags = {
    Name = "HomeAssistant-SG"
  }
}

// Define incoming traffic rules for ports 22 and 8123
resource "aws_vpc_security_group_ingress_rule" "allow_ssh" {
  security_group_id = aws_security_group.ha_sg.id
  cidr_ipv4         = aws_vpc.main_vpc.cidr_block
  from_port         = 22
  ip_protocol       = "tcp"
  to_port           = 22
}
resource "aws_vpc_security_group_ingress_rule" "allow_ha_client" {
  security_group_id = aws_security_group.ha_sg.id
  cidr_ipv4         = aws_vpc.main_vpc.cidr_block
  from_port         = 8123
  ip_protocol       = "tcp"
  to_port           = 8123
}

// Define outgoing traffic rule for all ports
resource "aws_vpc_security_group_egress_rule" "allow_all_traffic_ipv4" {
  security_group_id = aws_security_group.ha_sg.id
  cidr_ipv4         = var.public_cidr
  ip_protocol       = "-1" # semantically equivalent to all ports
}
```

Figure 4.23: HomeAssistant-SG Security Group

**i-0c29fa3255dc82955 (HomeAssistantCloud)**

▼ Inbound rules

Filter rules

Name	Security group rule ID	Port range	Protocol	Source	Security groups
-	sgr-0c7ac64bba7da0466	22	TCP	149.201.0.0/16	<a href="#">HomeAssistant-SG</a>
-	sgr-0996ba1d25995d0d5	1883	TCP	0.0.0.0/0	<a href="#">HomeAssistant-SG</a>
-	sgr-06081bd4cd1cb98f6	8123	TCP	149.201.0.0/16	<a href="#">HomeAssistant-SG</a>

▼ Outbound rules

Filter rules

Name	Security group rule ID	Port range	Protocol	Destination	Security groups
-	sgr-05160a5b1d3a5e1d0	All	All	0.0.0.0/0	<a href="#">HomeAssistant-SG</a>

Figure 4.24: HomeAssistant-SG in the AWS Dashboard

With the aim to set up the smart home environment, a starter script can be passed over the `user_data` attribute in Terraform, as if the user itself has entered commands in the CLI. The purpose of this script is to automatically install the default packages needed for the VM to run Home Assistant in a Docker Compose container setup [65], as well as provide the starter configuration for the `docker-compose.yml` file, on which Docker Compose bases the HA application deployment. The container is then started at the end of the script, allowing the user to access the Home Assistant instance over its public IP address and the 8123 port through the FH Aachen network. According to Figure 4.20. this is `35.159.195.176:8123`, however the IP address may vary. Figure 4.25. depicts the starter configuration script for the EC2 instance in AWS.

```

1  #!/bin/bash
2  # Install required packages for Docker Compose in EC2 virtual machine
   ↪  noninteractively
3  sudo apt-get -y update
4  sudo apt-get -y install ca-certificates curl openssl
5  sudo install -m 0755 -d /etc/apt/keyrings
6  sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o
   ↪  /etc/apt/keyrings/docker.asc
7  sudo chmod a+r /etc/apt/keyrings/docker.asc
8  echo \

```

```
9  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
   ↪ https://download.docker.com/linux/debian \
10  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
11  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
12  sudo apt-get -y update
13  sudo apt-get -y install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
   ↪ docker-compose-plugin
14
15  # Create Home Assistant container and encryption folder
16  sudo mkdir -p /home/admin/homeassistant
17  sudo mkdir -p /home/admin/homeassistant/ssl
18  # Create encryption key
19  sudo openssl req -x509 -nodes -newkey rsa:2048 -keyout privkey.pem -out cert.pem
   ↪ -days 365
20  sudo mv privkey.pem cert.pem home/admin/homeassistant/ssl/
21
22  # Create Docker Compose config file for HA container part of the network
23  cat > /home/admin/homeassistant/docker-compose.yml << EOF
24  services:
25    homeassistant:
26      container_name: home-assistant
27      image: ghcr.io/home-assistant/home-assistant:stable
28      volumes:
29        - ./config:/config
30        - ./ssl:/ssl
31      ports:
32        - "8123:8123"
33      network_mode: host
34  EOF
35
36  # Start Home Assistant service
37  cd /home/admin/homeassistant
38  sudo docker compose down
39  sudo docker compose up -d
```

Figure 4.25: EC2 Instance Setup Script



## 4.3 Conclusions

To complete the chapter outlining the implementation of both security smart home architectures, there are several key conclusions that have been drawn from their practical realization as ZT starting configurations. This section provides a brief overview of each of those aspects, which are then further expanded upon in the evaluation of this work in the overall assessment of this work according to the evaluation metrics defined in Chapter 3.

### Environment Setup

When assessing either the on-premise or the cloud-hosted environment, it should be kept in mind that while they offer a starting configuration for a Zero Trust architecture in smart homes, their setup process is different.

With regard to the local environment, the duration of the start-up phase (defined in Chapter 3) can be longer, as the user will need to obtain their own host device (e.g. Raspberry Pi), as well as install the HA software, which will manage the IoT devices within the environment. Moreover, one of the central components of the on-premise environment are the Home Assistant add-on applications [66], which even though they are easy to install and contribute to the integration of security measures, may cause a fundamental security risk. This is because the vast majority of them are open-source and cannot be subject to security guarantees.

In contrast to that, the cloud environment is relatively swift to set up. The combination of AWS and Terraform as an IaC tool offers a way for users to organize every aspect of the environment in a straightforward way. The provisioned VM negates the need to acquire a separate host device and utilizing a one-time setup script together with Terraform, with its approach to define the desired end state of the configuration, is a way to minimize the time it takes to have a starter smart home environment.

### Device Integration

As regards the topic of device integrations for both environments, they also differ in their complexity. Integrating devices locally is a straightforward process of the device

being in proximity to the host device. This offers an advantage when it comes to the ease of use of the SHS.

When it comes to the cloud-hosted environment, the process of pairing an IoT device is by all means a more complex one and has not been practically realized. Since the central part of the SHS is a host device, which is remotely provisioned by an IaaS provider, it is not plausible to integrate a device by the traditional means of closeness. The concept of a virtual gateway hosted by the host device (in this case a Raspberry Pi) offers a potential solution to that problem, however from a security perspective, it should be further investigated. As it stands, the current configuration of the cloud environment cannot integrate an IoT device, the implications of which are further explored in the next chapter.

## 5 Evaluation

To start the evaluation of this work, the initial research questions need to be taken into account, namely:

- How well is the Zero Trust security paradigm integrated into a smart home environment?
- Should a Zero Trust smart home network be hosted locally or in the cloud?

This thesis outlines the conceptualization and implementation methodology of two ZT smart home architectures — one in a local environment and the other hosted in the cloud. Despite the fact that both approaches to Zero Trust in smart homes had similar security configurations, the research yielded vastly different insights into the two smart home systems.

### 5.1 Overview

The practical implementation of both environments has proven useful in understanding the nature and the constraints of the on-premise and cloud-based concepts of the Zero Trust architectures. Concerning the locally operated solution, the environment setup has proven to be the more complex part, because it requires the acquisition of additional specific hardware from users. Additionally, the entire environment depends on the HA software add-ons [66] to deliver the desired functionality to the SHS. Nonetheless, the on-premise architecture solution is characterized by a straightforward approach to integrating devices.

On the other hand, in the process of examining the implementation of the cloud architecture, organizing and launching the SHS is more uncomplicated compared to the local one. It constitutes writing a lightweight Terraform script to launch the virtual machine in the cloud service provider (e.g. AWS) and then an additional Shell script to launch

the HA software. While adjusting the security configuration is a separate process, the bare minimum setup is still simpler in a cloud environment. However, with regard to the device integration, the implementation of the cloud smart home environment could not establish a pairing procedure with IoT devices. Although the prospect of a virtual gateway has been taken into account, it could not be examined further as part of this work.

## 5.2 Implementation Assessment

The implementation of the ZT architectures has proven useful in answering the aforementioned research questions, on top of offering further insight into their applicability. In regard to the topic of whether the Zero Trust security paradigm is applicable in a smart home environment and to what extent, the conclusions differ based on the type of environment. What unites the implementations of both the on-premise and cloud solutions is that they opt out for a more balanced approach, aiming to adopt ZT security principles together with the perimeter-based ones.

The on-premise implementation successfully adopted the principle of Multi-factor Authentication by means of modifying the configuration of the HA software (Figure 4.7). Moreover, by realizing a remote VPN tunnel connection to the local environment, the goal of having the environment continuously monitored through traffic analytics has been achieved. With respect to the traditional security features, implementing an IP address ban and a set of firewall rules for the remote VPN connection has also proven to be a success, serving to strengthen the security of the environment. What also gives the on-premise implementation an edge over the cloud-based one is the feature of automatic backups implemented in the Home Assistant operating system, as shown in Figure 4.6. Be that as it may, the on-premise architecture realization suffers from several drawbacks. A considerable portion of the functionality in this implementation is based on the widely available and open-source Home Assistant add-ons. While this makes the first environment more user-friendly and intuitive to configure, the source code of these plugins cannot be subject to any security guarantees and may pose security vulnerabilities for the smart home environment. In addition to this, even if one ignores the costs for the hardware acquisition and setup process, the connection to the remote tunnel has been inconstant on several occasions, defeating the purpose of its firewall and monitoring

features and going against the ease of use of the architecture. In these conditions, the adoption of ZT principles in a smart home environment has proven successful to some extent when combined with perimeter-based security principles. Nevertheless, relative to the core principles of the Zero Trust model outlined in Chapter 2, this implementation has yet to include IAM policies and network segmentation mechanisms for a fully realized ZT network environment.

On the other side of the spectrum, the solution derived from the cloud-based architecture is subject to improvement. On the positive side, the environment is nevertheless able to incorporate the MFA and IP ban mechanisms through the utilized HA software. Furthermore, the combination of Terraform and AWS empowers users to deploy the smart home environment in an EC2 virtual machine, proving to be the most efficient way to modify and apply configuration settings by way of IaC script files. What is achieved differently in this environment is the control over the allowed incoming and outgoing traffic, specifying the ports over which these rules can be enforced (Figure 4.20). In addition, the setup of the environment is realized by a minimal and adjustable setup script, as shown in Figure 4.23, marking this solution as one achieving scalability and hardware independence.

In spite of everything, the core functionalities of a smart home environment could not be realized – integration of IoT devices into the network and the ability to control them remotely. The exclusion of these features undermines the current setup of the environment, although it can be less difficult to launch at first. In this context, the HA software has been hosted as a container in contrast to the local setup, due to the lack of a proprietary AMI for this use case. Prompting users to create their own tailored OS image for the EC2 instance further complicates the cloud-based implementation and serves to counteract the aforementioned benefits of the cloud architecture. With regard to all these details, the verdict for this environment in the overall context of Zero Trust is that it is still in need of improvement. Although one of the four main ZT principles has been implemented, the dependence on perimeter-based configuration persists within this implementation.

To provide an answer to the second research question, the locally hosted smart home environment is better suited to hosting a Zero Trust smart home network, based on the aforementioned analysis of both environments. Not only does it implement more of the ZT framework’s principles, but it also has a better resolution as a standalone smart home environment compared to its cloud variant.

### 5.3 Metric Evaluation

As a conclusion for the evaluation of this thesis and its results, a short assessment of both categories of metrics is presented. The first table defines the security goals of the smart home architectures as metrics, which are then individually assessed on whether and to what extent the target metric was realized. Regarding the second table, it provides an assessment of how the secondary metrics relate to both the on-premise and cloud Zero Trust smart home architectures and their implementations.

Metric	Assessment
Device Integrity	In the on-premise environment, there has not been a registered instance of changes in the stored information as a form of sabotage for the collection of IoT devices.
Device Availability	In the on-premise environment, there has not been a registered instance of the user not being able to modify the information and configuration for the collection of IoT devices.
Automatic Updates	The system and add-on settings as part of the Home Assistant operating system provide users with the ability to enable automatic updates for the collection of services and add-ons in the on-premise environment. The update of the cloud environment is done by the users.
Automatic Backups	In the on-premise environment, an automatic backup of the system configuration is made after installation. Any further backups of the systems in both environments is done by the users.
Device Alarms and Monitoring	The ability to monitor the data from IoT devices is achieved through a dashboard in the HA software. More details about the state of single devices can be discovered from the device settings in Home Assistant. Device alarms have not been implemented.

Table 5.1: Security Metrics Evaluation Table

Metric	Assessment
Duration of start-up phase	Both implementations of the smart home architectures had a relatively short start-up phase as a result of excellent documentation of the employed services. In the comparison of both environments however, the cloud-based one has a shorter start-up phase on account of the script-based configuration process of the EC2 virtual machine through Terraform.
Platform Support	The local environment's central platform is Home Assistant, which gives users the ability to easily adjust and backup their system configuration, and also offered a way to easily implement multi-factor authentication. The cloud-based environment's main platform being AWS, means that users have access to a plethora of tools to finetune and enhance the management and security of the EC2 environment.
Platform Complexity	Home Assistant offers a wide functionality and support for services and add-ons, most of which incredibly well documented to make their integration into the on-premise environment straightforward. With regard to AWS, while it offers a larger functionality than Home Assistant, the overabundance of those features could cause difficulties to realize a comprehensive smart home implementation.

Table 5.2: Secondary Metrics Evaluation Table



## 6 Summary and Outlook

### 6.1 Summary

The purpose of this work was to examine the integration of the Zero Trust security paradigm in two types of smart home environments. As outlined in the first chapter, the two main objectives are derived from the research questions for this topic: first and foremost, to examine how can the ZT security model be integrated into a smart home environment; and secondarily, to provide an answer on whether the superior way to host a Zero smart home system is on premises or in the cloud. In hindsight, both of these objectives have been fulfilled successfully, with regard to addressing the pre-established research questions and providing valuable insights into the topic.

Regarding the integration of Zero Trust as part of the first research question, the proposed architectures are based on the setup of three main components: a host device, device gateway and a client device, as depicted in Figure 3.2. Additionally, the concepts for both the on-premise and cloud architectures have outlined as starting configurations with the view to incorporate the most prominent aspects of both the ZT and perimeter-based security principles. In that regard, the local architecture has been conceptualized with the principles of MFA, principle of least privilege, firewall-based remote connection and the tunnel analytics. In contrast to that, the cloud concept incorporates a shorter set of features: MFA, least privilege infrastructure provision and IP-based traffic filtering. Furthermore, for the purpose of evaluating both architecture concepts and their implementations described in Chapter 4, there was a latent need for KPI measurements specifically created for that. Summarized in Table 3.1, they are separated into primary (relating to security) and secondary metrics. After providing a short insight into the limitations and challenges of the architectures in the context of IIoT, as well as for their individual features, the process of practically setting up the smart home solutions derived from them has been described in detail.

The on-premise architecture implementation has utilized a Raspberry Pi booted with the Home Assistant OS, establishing this environment as part of a larger local network. In terms of the configuration settings, they are managed in a central YAML configuration file as part of the file system, used to implement an MFA mechanism using TOTP, IP address banning and a remote tunnel connection. On the subject of the devices within the smart home environment, both Zigbee and BLE devices have been integrated with two separate automations, according to their functionality. Figure 4.14 depicts the full architecture of the on-premise environment, summarizing all its main features.

Pertaining to the cloud-based architecture implementation, it has been actualized using an EC2 virtual machine as part of AWS and Terraform to manage its configuration. In addition, the Docker Compose software is used to run the Home Assistant software in a container environment, as opposed to an explicit OS. Figure 4.15 depicts the full architecture of the cloud-based environment, which provides a summary of its integrated features. The configuration settings as well as the fundamental Terraform concepts have been outlined, related to their role in the virtual machine setup and utilized to provision the needed network infrastructure. In regard to the security features, MFA has been implemented through the Home Assistant container application, with the IP-based traffic filtering being based on the definition of security groups as part of the VPC settings. The final steps in the setup of the environment have been achieved by simulating user input through a Shell script, aiming to install all the prerequisite packages for the EC2 instance and start the Home Assistant application.

Before evaluating the results from their respective Zero Trust smart home architectures, the last part of Chapter 4 provides several key conclusions derived from their practical implementation with reference to the setup processes of both smart home environments and the complexity of their device integration.

The evaluation of the thesis results is composed of three parts. It begins with reminding the reader of both research questions and provides a brief overview of the insights in the practical implementations of both smart home environments. Subsequently, the chapter reviews the implementations of both smart home environments in the context of the first research question on whether Zero Trust principles are applicable in a SHS, including the effectiveness of the introduced security measures, usability, and their shortcomings. Moreover, a final verdict for both SHS solutions has been included, referring to their advantages and disadvantages, together with an answer to the second research pointing out the on-premise implementation as more suitable for a Zero Trust SHS. Conclusively,

the last part of the evaluation is focused on the primary and secondary evaluation metrics, defined as part of Section 3.2. The estimated results in relation to each of the metrics are summarized in two tables.

To summarize, this thesis work explored the adoption of the Zero Trust security model and its core principles within the context of smart home environments. As part of this process, two smart home architectures have been conceptualized with a combination of Zero Trust and perimeter-based security measures. Additionally, the overall and specific limitations of these architectures have been described, together with evaluation metrics to estimate the security of both environments. Considering the practical implementation of SHS solutions on the basis of the aforementioned architectures, both yielded tangible results with the cloud-based environment being more unsophisticated in its setup, yet the on-premise one has proven more refined in its implementation. This distinction persists in the evaluation of the work, as it not only adopts more of the Zero Trust model's principles, but also functions more effectively as a separate smart home environment, taking into account the metric-based analysis.

## 6.2 Outlook

In bringing this thesis work to a close, the conceptualized SHS architectures and their respective implementations yielded intriguing insights into the adoption of Zero Trust security into smart home environments. Consequently, two main focuses with research potential arise from the subject of this work, which could pave the way for the continued development of solutions in this academic niche.

The first research direction places emphasis on the concept of a Zero Trust security architecture based on edge computing principles. As described in [12], the case study of edge computing for smart homes sets a clear boundary pertaining to the processing of data, mostly limiting this task to the home network with a view towards data privacy. Therefore, this line of research could serve to refine the originally introduced on-premise architecture to keep the access to and control over the data from IoT devices exclusively in the local environment, while also giving users the opportunity to restrict accessibility to highly sensitive data, should it be used by service providers [12]. In the context of Zero Trust, this process could include implementing RBAC for different user

profiles and enforcing it through IAM policies as part of the edge operating system in this architecture, intended to breach the gap between the smart home system and its security measures.

Secondly, since the implementations of the introduced architectures amount to starting configurations for ZT in smart homes, they both offer many opportunities for improvement. This is especially true for the cloud-based smart home solution, which, when compared to its on-premise counterpart, lacks in the device integration process. In that regard, AWS offers additional services that can be worth investigating for the integration and monitoring of IoT devices in a ZTA. For instance, AWS IoT Core is responsible for connecting IoT devices and provides authentication measures to other AWS resources through device certificates, whereas the AWS Device Defender service could be used to establish secure baseline behaviour for each of the IoT devices within the network and monitor for anomalous activity [67]. It would also be beneficial to explore the mitigation approaches these AWS IoT services can adopt for non-compliant device behaviour.

# A Appendix

```
1 # Loads default set of integrations. Do not remove.
2 default_config:
3
4 # Load frontend themes from the themes folder
5 frontend:
6   themes: !include_dir_merge_named themes
7
8 automation: !include automations.yaml
9 script: !include scripts.yaml
10 scene: !include scenes.yaml
11
12 # Security settings
13 homeassistant:
14   auth_mfa_modules:
15     type: totp
16     name: "Duo Mobile"
17
18 http:
19   ip_ban_enabled: true
20   login_attempts_threshold: 3
21   use_x_forwarded_for: true
22   trusted_proxies:
23     - 149.201.182.0/24
24     - 172.30.33.0/24
25
```

Figure A.1: Local Home Assistant configuration.yaml File

```
1 # Loads default set of integrations. Do not remove.
2 default_config:
3
4 # Load frontend themes from the themes folder
5 frontend:
6   themes: !include_dir_merge_named themes
7
8 automation: !include automations.yaml
9 script: !include scripts.yaml
10 scene: !include scenes.yaml
11
12 # Security settings
13 http:
14   ssl_certificate: /ssl/fullchain.pem
15   ssl_key: /ssl/privkey.pem
16   ip_ban_enabled: true
17   login_attempts_threshold: 3
18
19 homeassistant:
20   auth_mfa_modules:
21     type: totp
22     name: "Duo Mobile"
```

Figure A.2: Cloud Home Assistant configuration.yaml File

```
1 // Variable declarations in Terraform (see descriptions)
2 variable "region" {
3   description = "AWS region"
4   type        = string
5 }
6 variable "vpc_cidr" {
7   description = "CIDR block for VPC"
8   type        = string
9 }
10 variable "subnet_cidrs" {
11   description = "CIDR blocks for the subnets"
12   type        = list(string)
```

```
13 }
14 variable "az" {
15     description = "Availability zones for the subnets"
16     type        = list(string)
17 }
18 variable "public_cidr" {
19     description = "CIDR block for public access (SSH)"
20     type        = string
21 }
22 variable "key_name" {
23     description = "SSH key name for the EC2 instance"
24     type        = string
25 }
26 variable "ami" {
27     description = "AMI of EC2 instance"
28     type        = string
29 }
30 variable "instance_type" {
31     description = "EC2 instance type"
32     type        = string
33 }
```

Figure A.3: variables.tf Terraform File

```
1 // AWS region where resources will be deployed
2 region = "eu-central-1"
3
4 // Default CIDR block for the VPC
5 vpc_cidr = "172.31.0.0/16"
6
7 // List of default CIDR blocks for subnets
8 subnet_cidrs = [
9     "172.31.0.0/20", // eu-central-1a
10    "172.31.16.0/20", // eu-central-1b
11    "172.31.32.0/20" // eu-central-1c
12 ]
13
```

```
14 // List of availability zones for subnets
15 az = [
16     "eu-central-1a",
17     "eu-central-1b",
18     "eu-central-1c"
19 ]
20
21 // Public CIDR block (uni network)
22 public_cidr = "149.201.0.0/16"
23
24 // Defined SSH key
25 key_name = "ha-cloud"
26
27 // AMI with OS of the EC2 instance
28 ami = "ami-0584590e5f0e97daa"
29
30 // EC2 instance type
31 instance_type = "t2.micro"
```

Figure A.4: terraform.tfvars Terraform File

```
1 // Prerequisites for Terraform in main.tf
2 terraform {
3     required_providers {
4         aws = {
5             source = "hashicorp/aws"
6             version = "~> 5.77"
7         }
8     }
9     required_version = ">= 1.2.0"
10 }
11 provider "aws" {
12     region = var.region
13 }
14
15 // Definition of the EC2 instance as resource with all the needed infrastructure
16 resource "aws_instance" "ec2_vm" {
```



```
17     ami                = var.ami
18     instance_type      = var.instance_type
19     key_name           = var.key_name
20     subnet_id          = aws_subnet.main_subnet.id
21     vpc_security_group_ids = [aws_security_group.ha_sg.id]
22
23     user_data = "${file("ec2setup.sh")}"
24
25     tags = {
26       Name = "HomeAssistantCloudInstance"
27     }
28   }
29   // Terraform output value for the public IP of the instance
30   output "ec2_public_ip" {
31     description = "Public IP for ssh access"
32     value       = aws_instance.ec2_vm.public_ip
33   }
```

Figure A.5: main.tf Terraform File

```
1 // Define AWS security group for the ports 22 and 8123
2 resource "aws_security_group" "ha_sg" {
3   name     = "HomeAssistant-SG"
4   description = "Allow port 22 and 8123 inbound traffic and all outbound traffic"
5   vpc_id    = aws_vpc.main_vpc.id
6
7   tags = {
8     Name = "HomeAssistant-SG"
9   }
10 }
11
12 // Define incoming traffic rules for ports 22 and 8123
13 resource "aws_vpc_security_group_ingress_rule" "allow_ssh" {
14   security_group_id = aws_security_group.ha_sg.id
15   cidr_ipv4         = aws_vpc.main_vpc.cidr_block
16   from_port         = 22
17   ip_protocol       = "tcp"
```

```
18     to_port          = 22
19 }
20 resource "aws_vpc_security_group_ingress_rule" "allow_ha_client" {
21     security_group_id = aws_security_group.ha_sg.id
22     cidr_ipv4         = aws_vpc.main_vpc.cidr_block
23     from_port         = 8123
24     ip_protocol       = "tcp"
25     to_port          = 8123
26 }
27
28 // Define outgoing traffic rule for all ports
29 resource "aws_vpc_security_group_egress_rule" "allow_all_traffic_ipv4" {
30     security_group_id = aws_security_group.ha_sg.id
31     cidr_ipv4         = var.public_cidr
32     ip_protocol       = "-1" # semantically equivalent to all ports
33 }
```

Figure A.6: security.tf Terraform File

```
1 // Define AWS VPC with the 172.31.0.0/16 CIDR
2 resource "aws_vpc" "main_vpc" {
3     cidr_block = var.vpc_cidr
4
5     tags = {
6         Name = "HA-VPC"
7     }
8 }
9 // Define VPC subnet with availability zone
10 resource "aws_subnet" "main_subnet" {
11     vpc_id = aws_vpc.main_vpc.id
12     cidr_block = element(var.subnet_cidrs, 0) // reserve first subnet for the instance
13     ↵
14     availability_zone = element(var.az, 2) // "eu-central-1c"
15     tags = {
16         Name = "HA-Subnet"
17     }
18 }
```

```
18 // Define the internet gateway for the VPC
19 resource "aws_internet_gateway" "main_gw" {
20     vpc_id = aws_vpc.main_vpc.id
21
22     tags = {
23         Name = "HA-Gateway"
24     }
25 }
26 // Define the routing table for the VPC and subnets
27 resource "aws_route_table" "main_rt" {
28     vpc_id = aws_vpc.main_vpc.id
29     route {
30         cidr_block = var.public_cidr
31         gateway_id = aws_internet_gateway.main_gw.id
32     }
33
34     tags = {
35         Name = "HA-RT"
36     }
37 }
38 // Define table association for the subnet
39 resource "aws_route_table_association" "a" {
40     subnet_id = aws_subnet.main_subnet.id
41     route_table_id = aws_route_table.main_rt.id
42 }
```

Figure A.7: network.tf Terraform File

```
1 #!/bin/bash
2 # Install required packages for Docker Compose in EC2 virtual machine
3 ↪ noninteractively
4 sudo apt-get -y update
5sudo apt-get -y install ca-certificates curl openssl
6sudo install -m 0755 -d /etc/apt/keyrings
7sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o
8↪ /etc/apt/keyrings/docker.asc
9sudo chmod a+r /etc/apt/keyrings/docker.asc
```

```

8 echo \
9   "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
   ↪ https://download.docker.com/linux/debian \
10  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
11  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
12 sudo apt-get -y update
13 sudo apt-get -y install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
   ↪ docker-compose-plugin
14
15 # Create Home Assistant container and encryption folder
16 sudo mkdir -p /home/admin/homeassistant
17 sudo mkdir -p /home/admin/homeassistant/ssl
18 # Create encryption key
19 sudo openssl req -x509 -nodes -newkey rsa:2048 -keyout privkey.pem -out cert.pem
   ↪ -days 365
20 sudo mv privkey.pem cert.pem home/admin/homeassistant/ssl/
21
22 # Create Docker Compose config file for HA container part of the network
23 cat > /home/admin/homeassistant/docker-compose.yml << EOF
24 services:
25   homeassistant:
26     container_name: home-assistant
27     image: ghcr.io/home-assistant/home-assistant:stable
28     volumes:
29       - ./config:/config
30       - ./ssl:/ssl
31     ports:
32       - "8123:8123"
33     network_mode: host
34 EOF
35
36 # Start Home Assistant service
37 cd /home/admin/homeassistant
38 sudo docker compose down
39 sudo docker compose up -d

```

Figure A.8: EC2 Instance Setup Script

# References

- [1] T. Chaurasia and P. K. Jain,  
“Enhanced Smart Home Automation System based on Internet of Things,”  
in *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Dec. 2019, p. 709.  
DOI: 10.1109/I-SMAC47947.2019.9032685. [Online]. Available:  
<https://ieeexplore.ieee.org/document/9032685> (visited on 04/13/2025).
- [2] H. Kang, G. Liu, Q. Wang, L. Meng, and J. Liu, “Theory and Application of Zero Trust Security: A Brief Survey,”  
*Entropy*, vol. 25, no. 12, p. 1595, Nov. 2023, ISSN: 1099-4300.  
DOI: 10.3390/e25121595. [Online]. Available:  
<https://www.mdpi.com/1099-4300/25/12/1595> (visited on 04/13/2025).
- [3] A. Chakraborty, M. Islam, F. Shahriyar, S. Islam, H. U. Zaman, and M. Hasan,  
“Smart Home System: A Comprehensive Review,”  
*Journal of Electrical and Computer Engineering*, vol. 2023, no. 1, pp. 1, 7, 2023,  
ISSN: 2090-0155. DOI: 10.1155/2023/7616683. [Online]. Available:  
<https://onlinelibrary.wiley.com/doi/abs/10.1155/2023/7616683> (visited on 04/13/2025).
- [4] Y. Han, *Smart home: Market data & analysis*. [Online]. Available:  
<https://www.statista.com/study/42112/smart-home-report/> (visited on 04/13/2025).
- [5] I. C. L. Ng and S. Y. L. Wakenshaw, “The Internet-of-Things: Review and research directions,” *International Journal of Research in Marketing*, vol. 34, no. 1, pp. 3–21, Mar. 2017, ISSN: 0167-8116.  
DOI: 10.1016/j.ijresmar.2016.11.003. [Online]. Available:  
<https://www.sciencedirect.com/science/article/pii/S0167811615301890> (visited on 04/13/2025).

- [6] D. C. Khedekar, A. C. Truco, D. A. Oteyza, and G. F. Huertas, “Home Automation—A Fast - Expanding Market,” en, *Thunderbird International Business Review*, vol. 59, no. 1, pp. 79–91, 2017, \_eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/tie.21829>, ISSN: 1520-6874. DOI: 10.1002/tie.21829. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/tie.21829> (visited on 04/13/2025).
- [7] D. Smith, *Understanding tvoc: What you need to know about volatile organic compounds*. [Online]. Available: <https://learn.kaiterra.com/en/resources/understanding-tvoc-volatile-organic-compounds> (visited on 04/13/2025).
- [8] Aqara, *Tvoc air quality monitor - aqara*. [Online]. Available: <https://www.aqara.com/eu/product/tvoc-air-quality-monitor/> (visited on 04/13/2025).
- [9] Z. Sharif, T. Low, M. Ayaz, M. Yahya, and D. Khan, “Smart home automation by internet-of-things edge computing platform,” *International Journal of Advanced Computer Science and Applications*, vol. 13, p. 474, Jan. 2022. DOI: 10.14569/IJACSA.2022.0130455. (visited on 04/13/2025).
- [10] D. Griffin, *Home assistant*. [Online]. Available: <https://github.com/home-assistant> (visited on 04/13/2025).
- [11] C. Stolojescu-Crisan, C. Crisan, and B.-P. Butunoi, “An IoT-Based Smart Home Automation System,” en, *Sensors*, vol. 21, no. 11, p. 3784, Jan. 2021, Number: 11 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 1424-8220. DOI: 10.3390/s21113784. [Online]. Available: <https://www.mdpi.com/1424-8220/21/11/3784> (visited on 04/13/2025).
- [12] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016. DOI: 10.1109/JIOT.2016.2579198. (visited on 04/13/2025).
- [13] G. Sharkov, W. Asif, and I. Rehman, “Securing smart home environment using edge computing,” in *2022 IEEE International Smart Cities Conference (ISC2)*, 2022, pp. 1–7. DOI: 10.1109/ISC255366.2022.9921912. (visited on 04/13/2025).

## References

---

- [14] W. Williamson, *Lateral movement: When cyber attacks go sideways*. [Online]. Available: <https://www.securityweek.com/lateral-movement-when-cyber-attacks-go-sideways/> (visited on 04/13/2025).
- [15] F. Jimmy, “Zero trust security: Reimagining cyber defense for modern organizations,” *International Journal of Scientific Research and Management (IJSRM)*, vol. 10, pp. 887–905, Nov. 2024. DOI: 10.18535/ijssrm/v10i4.ec11. (visited on 04/13/2025).
- [16] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, “Zero trust architecture,” Tech. Rep., Aug. 2020. DOI: 10.6028/nist.sp.800-207. [Online]. Available: <https://doi.org/10.6028/nist.sp.800-207> (visited on 04/13/2025).
- [17] Gilman, Evan and Barth, Doug, *Zero trust networks: building secure systems in untrusted networks*, First edition. Beijing ; Boston ; Farnham ; Sebastopol ; Tokyo: O’Reilly, July 2017, pp. 101, 103, ISBN: 1491962194; 9781491962190. (visited on 04/13/2025).
- [18] W. contributors, *Home assistant*. [Online]. Available: [https://en.wikipedia.org/wiki/Home\\_Assistant](https://en.wikipedia.org/wiki/Home_Assistant) (visited on 04/13/2025).
- [19] H. Assistant, *Architecture overview*. [Online]. Available: [https://developers.home-assistant.io/docs/architecture\\_index/](https://developers.home-assistant.io/docs/architecture_index/) (visited on 04/13/2025).
- [20] W. contributors, *Raspberry pi*. [Online]. Available: [https://en.wikipedia.org/wiki/Raspberry\\_Pi](https://en.wikipedia.org/wiki/Raspberry_Pi) (visited on 04/13/2025).
- [21] A. Cooban, *Why investors are going gaga over a tiny, 35computer*. [Online]. Available: <https://edition.cnn.com/2024/06/11/tech/raspberry-pi-ipo-london-stock-exchange/index.html> (visited on 04/13/2025).
- [22] GeeksforGeeks, *Architecture of raspberry pi*. [Online]. Available: <https://www.geeksforgeeks.org/architecture-of-raspberry-pi/> (visited on 04/13/2025).
- [23] D. Griffin, *Home assistant operating system*. [Online]. Available: <https://github.com/home-assistant/operating-system> (visited on 04/13/2025).

- [24] W. Wiki, *Architecture of raspberry pi*. [Online]. Available: [https://wiki.wireshark.org/IEEE\\_802.15.4](https://wiki.wireshark.org/IEEE_802.15.4) (visited on 04/13/2025).
- [25] watt24 GmbH, *Zigbee - all you need to know*. [Online]. Available: <https://www.watt24.com/en/guide/everything-you-need-to-know-about-zigbee/> (visited on 04/13/2025).
- [26] H. Tung, K. Tsang, H. Tung, V. Rakocovic, K. Chui, and Y. W. Leung, “A wifi-zigbee building area network design of high traffics ami for smart grid,” *Smart Grid and Renewable Energy*, vol. 03, pp. 324–333, Jan. 2012. DOI: 10.4236/sgre.2012.34043. (visited on 04/13/2025).
- [27] N. Kajikawa, Y. Minami, E. Kohno, and Y. Kakuda, “On availability and energy consumption of the fast connection establishment method by using bluetooth classic and bluetooth low energy,” in *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, 2016, p. 287. DOI: 10.1109/CANDAR.2016.0058. (visited on 04/13/2025).
- [28] Sparkleo, *Bluetooth vs wifi vs zigbee: Which one is the best for smart home?* [Online]. Available: <https://medium.com/@sparkleo/bluetooth-vs-wifi-vs-zigbee-which-one-is-the-best-for-smart-home-1bbabbbf5c4c6> (visited on 04/13/2025).
- [29] R. Aigner, M. Kofler, K. Gebeshuber, *et al.*, *Hacking & Security: das umfassende Handbuch* (Rheinwerk Computing). Rheinwerk Verlag, 2018, pp. 1081–1083, ISBN: 978-3-8362-4548-7. [Online]. Available: <https://books.google.de/books?id=tb0JtAEACAAJ> (visited on 04/13/2025).
- [30] MQTT.org, *Mqtt: The standard for iot messaging*. [Online]. Available: <https://mqtt.org/> (visited on 04/13/2025).
- [31] A. W. Services, *What is iaas (infrastructure as a service)?* [Online]. Available: <https://aws.amazon.com/what-is/iaas/> (visited on 04/13/2025).
- [32] A. W. Services, *Amazon ec2 instance types*. [Online]. Available: <https://aws.amazon.com/ec2/instance-types/> (visited on 04/13/2025).



- [33] Stender, Daniel,  
*Cloud-Infrastrukturen: das Handbuch für DevOps-Teams und Administratoren*,  
1. Auflage. Bonn: Rheinwerk Verlag, 2020,  
pp. 30, 42, 54–56, 151–152, 194, 364–365, ISBN: 9783836269483.  
[Online]. Available: [https://digitale-objekte.hbz-nrw.de/storage/2/2020/07/09/file\\_18/8883567.pdf](https://digitale-objekte.hbz-nrw.de/storage/2/2020/07/09/file_18/8883567.pdf) (visited on 04/13/2025).
- [34] I. HashiCorp, *How does terraform work?*  
[Online]. Available: <https://developer.hashicorp.com/terraform/intro>  
(visited on 04/13/2025).
- [35] I. HashiCorp, *State*. [Online]. Available:  
<https://developer.hashicorp.com/terraform/language/state> (visited on  
04/13/2025).
- [36] J. R. Flavius Dinu, *Managing terraform state – best practices examples*.  
[Online]. Available: <https://spacelift.io/blog/terraform-state> (visited on  
04/13/2025).
- [37] *What is containerization?*  
[Online]. Available: <https://www.ibm.com/think/topics/containerization>  
(visited on 04/13/2025).
- [38] *Docker compose*. [Online]. Available: <https://docs.docker.com/compose/>  
(visited on 04/13/2025).
- [39] *How compose works*. [Online]. Available:  
<https://docs.docker.com/compose/intro/compose-application-model/>  
(visited on 04/13/2025).
- [40] V. Antoseac, *Why should you go with docker and docker compose?*  
[Online]. Available: <https://try.direct/blog/why-should-you-go-with-docker-and-docker-compose-> (visited on 04/13/2025).
- [41] D. Coss, “The cia strikes back: Redefining confidentiality, integrity and availability in security,”  
*Journal of Information System Security*, vol. 10, pp. 21, 24, Jan. 2014.  
(visited on 04/13/2025).

- [42] O. H. Foundation, *Homekit device*. [Online]. Available: [https://www.home-assistant.io/integrations/homekit\\_controller/#adding-a-homekit-device-through-bluetooth](https://www.home-assistant.io/integrations/homekit_controller/#adding-a-homekit-device-through-bluetooth) (visited on 04/13/2025).
- [43] O. H. Foundation, *Updating home assistant*. [Online]. Available: <https://www.home-assistant.io/common-tasks/os/#updating-home-assistant> (visited on 04/13/2025).
- [44] O. H. Foundation, *Backups*. [Online]. Available: <https://www.home-assistant.io/common-tasks/general/#backups> (visited on 04/13/2025).
- [45] O. H. Foundation, *Backup*. [Online]. Available: <https://www.home-assistant.io/integrations/backup/> (visited on 04/13/2025).
- [46] L. D. Xu, W. He, and S. Li, “Internet of things in industries: A survey,” *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014. DOI: 10.1109/TII.2014.2300753. (visited on 04/13/2025).
- [47] M. Serror, S. Hack, M. Henze, M. Schuba, and K. Wehrle, “Challenges and opportunities in securing the industrial internet of things,” *IEEE Transactions on Industrial Informatics*, vol. 17, no. 5, pp. 2985–2996, 2021. DOI: 10.1109/TII.2020.3023507. (visited on 04/13/2025).
- [48] C. Duo, *Multi-factor authentication (mfa)*. [Online]. Available: <https://duo.com/product/multi-factor-authentication-mfa> (visited on 04/13/2025).
- [49] I. Cloudflare, *Cloudflare zero trust docs*. [Online]. Available: <https://developers.cloudflare.com/cloudflare-one/> (visited on 04/13/2025).
- [50] smart-home-komponente, *Raspberry pi 4 4gb starter-set / 64gb sd karte / usb-c netzteile 15w / gehäuse / 4k micro hdmi kabel / kühlkörper set / raspberry pi 4 model b 4gb ram*. [Online]. Available: [https://www.amazon.de/Raspberry-Model-4GB-Desktop-Starter-Kit-wei%C3%9F/dp/B07WHWR4LH/ref=sr\\_1\\_5?\\_\\_mk\\_de\\_DE=%C3%85M%C3%85C5BD%C3%95%C3%91&crid=U8BOY005FJPU&dib=eyJ2IjoimSJ9.MjsAGAS8\\_trWJoKpwhUilaPCpo1XC47Asn3FMp24sHtIiqltebPKzThpzgBkleoIU0um3CqQnD](https://www.amazon.de/Raspberry-Model-4GB-Desktop-Starter-Kit-wei%C3%9F/dp/B07WHWR4LH/ref=sr_1_5?__mk_de_DE=%C3%85M%C3%85C5BD%C3%95%C3%91&crid=U8BOY005FJPU&dib=eyJ2IjoimSJ9.MjsAGAS8_trWJoKpwhUilaPCpo1XC47Asn3FMp24sHtIiqltebPKzThpzgBkleoIU0um3CqQnD)

- tY1XFD7h6grU\_BzAGhGsvLBQMGwx4XfR657\_nBL9KJofyVf4YhL-i2FuTjol2pw08sF  
EyRfOIJaB3TSYPxI4miRnV0\_qDOWeVkRYacy5z7Rho28RipKZJgxShfWIaW4oyuAhac  
H3RkRIC5hK7Kxeq-81WZlLcP8xKwE8cBcoPvSszyljSBBHGXIJj\_comhX0fA6ZVpicz  
OGyze2e774FWMJLSh9wLaK593KFqqvmcYqNPQo-sl27MvtP5AQAMOR2np\_wgC3Ve-kg  
hM\_ilTmoR0JmJvm5aRT\_8.vY2lFRim6-Kp7SLKm4TocoJPDyEZv8wrteHB1xluqsg&d  
ib\_tag=se&keywords=raspberrypi+starter-set&qid=1743400680&s=ce-de  
&sprefix=raspberrypi+starter-set%2Celectronics%2C85&sr=1-5 (visited  
on 04/13/2025).
- [51] R. Pi, *Raspberrypi/rpi-imager: The home of raspberry pi imager, a user-friendly tool for creating bootable media for raspberry pi devices*. [Online]. Available: <https://github.com/raspberrypi/rpi-imager> (visited on 04/13/2025).
- [52] O. H. Foundation, *Raspberry pi*. [Online]. Available: <https://www.home-assistant.io/installation/raspberrypi> (visited on 04/13/2025).
- [53] O. H. Foundation, *Onboarding home assistant*. [Online]. Available: <https://www.home-assistant.io/getting-started/onboarding/> (visited on 04/13/2025).
- [54] F. O. for Information Security, *Creating secure passwords*. [Online]. Available: [https://www.bsi.bund.de/EN/Themen/Verbraucherinnen-und-Verbraucher/Informationen-und-Empfehlungen/Cyber-Sicherheitsempfehlungen/Accountschutz/Sichere-Passwoerter-erstellen/sichere-passwoerter-erstellen\\_node.html](https://www.bsi.bund.de/EN/Themen/Verbraucherinnen-und-Verbraucher/Informationen-und-Empfehlungen/Cyber-Sicherheitsempfehlungen/Accountschutz/Sichere-Passwoerter-erstellen/sichere-passwoerter-erstellen_node.html) (visited on 04/13/2025).
- [55] O. H. Foundation, *Configuration.yaml*. [Online]. Available: <https://www.home-assistant.io/docs/configuration/> (visited on 04/13/2025).
- [56] O. H. Foundation, *Raspberry pi*. [Online]. Available: <https://www.home-assistant.io/docs/authentication/multi-factor-auth/#notify-multi-factor-authentication-module> (visited on 04/13/2025).
- [57] O. H. Foundation, *Http*. [Online]. Available: <https://www.home-assistant.io/integrations/http/#reverse-proxies> (visited on 04/13/2025).

- [58] I. Cloudflare, *Create a tunnel (dashboard)*. [Online]. Available: <https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/get-started/create-remote-tunnel/> (visited on 04/13/2025).
- [59] T. Brenner, *Home assistant add-on: Cloudflared*. [Online]. Available: <https://github.com/brenner-tobias/addon-cloudflared> (visited on 04/13/2025).
- [60] Z. D. C. Repository, *Tradfri led bulb e27 1055 lumen, dimmable, white spectrum, opal white*. [Online]. Available: [https://zigbee.blakadder.com/Ikea\\_LED2003G10.html](https://zigbee.blakadder.com/Ikea_LED2003G10.html) (visited on 04/13/2025).
- [61] Z. D. C. Repository, *Tradfri control outlet*. [Online]. Available: [https://zigbee.blakadder.com/Ikea\\_E1603.html](https://zigbee.blakadder.com/Ikea_E1603.html) (visited on 04/13/2025).
- [62] A. Inc., *Flower care smart plant monitor official european version real time sensors for soil fertility humidity level light temperature ios and android app*. [Online]. Available: [https://www.amazon.de/-/en/Official-European-Fertility-Humidity-Temperature/dp/B01MUDQD8I/ref=sr\\_1\\_6?sr=8-6](https://www.amazon.de/-/en/Official-European-Fertility-Humidity-Temperature/dp/B01MUDQD8I/ref=sr_1_6?sr=8-6) (visited on 04/13/2025).
- [63] I. HashiCorp, *Output values*. [Online]. Available: <https://developer.hashicorp.com/terraform/language/values/outputs> (visited on 04/13/2025).
- [64] I. Amazon Web Services, *Default vpc components*. [Online]. Available: <https://docs.aws.amazon.com/vpc/latest/userguide/default-vpc-components.html> (visited on 04/13/2025).
- [65] *Install docker engine on debian*. [Online]. Available: <https://docs.docker.com/engine/install/debian/#install-using-the-repository> (visited on 04/13/2025).
- [66] O. H. Foundation, *Home assistant add-ons*. [Online]. Available: <https://www.home-assistant.io/addons/> (visited on 04/13/2025).
- [67] R. Dsouza, *How to implement zero trust iot solutions with aws iot*. [Online]. Available: <https://aws.amazon.com/blogs/iot/how-to-implement-zero-trust-iot-solutions-with-aws-iot-3/> (visited on 04/13/2025).