Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Evaluation and Implementation of Zero-Touch Onboarding Solutions for IIoT

Vishwas Avinash Jadhav

Course of Study: M.Sc. Information Technology

Examiner: Dr. Ilche Georgievski

Supervisor: Dr. Wilfried Wessner

Commenced: December 1, 2022

Completed: May 9, 2023

Abstract

The Internet of Things (IoT) has provided numerous opportunities across various industries, and its technological advancements are rapidly progressing. The process of Device Onboarding refers to the task of initially registering a device onto an IoT cloud. Onboarding can be very challenging while doing a large-scale IoT deployment. A recent report by IoT analytics predicts that the number of IoT devices will increase twofold, with an expected total of Twenty-seven billion devices by 2025. As the number of IoT devices proliferates, establishing a streamlined process for onboarding multiple devices has become more imperative than ever.

While doing this research, we first evaluated the state-of-the-art solutions for onboarding IoT edge devices. The methods were Fido (Fast Identity Online) Device Onboard (FDO) from the Linux Foundation, Keylime Solutions from the cloud-native computing foundation (CNCF), and OPCUA device onboard by the OPC foundation. The evaluation is based on 22 various factors. The assessment led us toward the best possible solution, i.e., FDO. to do zero-touch onboarding and register the device to an IoT device management service.

In addition to addressing the existing inconsistencies in the device manufacturing process using FDO standards, this research has also implemented these standards in a novel way. Many zero-touch onboarding solutions today demand special software and hardware to be integrated into the IoT device during manufacturing. The proposed solution using FDO standards with a "Late Binding"feature is an open, cloud-agnostic solution that allows users to choose their preferred cloud provider during the initial power-up. It eliminates the need for Original Device Manufacturers to manufacture unique device Stock Keeping Units (SKUs) for each customer and cloud combination.

To summarize, This research thoroughly evaluated and chose a suitable onboarding method and successfully demonstrated the implementation. We used the raspberry pi compute module 4 as an IoT edge device with a custom-embedded Linux OS. Developing the novel hawkBit onboarding modules on both the client and server sides has facilitated this implementation. The proposed software-based solution is capable of onboarding 43200 devices per day. Utilizing the FDO 1.1 standards, this approach has proven to be a highly effective solution for the zero-touch onboarding of a large number of IoT devices.

Contents

1 Introduction **9**
- 1.1 Motivation of Research . . . 10
- 1.2 Research Questions . . . 11
- 1.3 Research Methodology . . . 12
- 1.4 Organisation of Report . . . 12

2 Background **14**
- 2.1 IoT Device Onboarding . . . 14
- 2.2 Zero Touch IoT Device Onboarding . . . 14
- 2.3 FIDO Device Onboarding (FDO) [**2**] . . . 15
- 2.4 Keylime[**3**] . . . 18
- 2.5 OPCUA Device Onboard . . . 20
- 2.6 Embedded Linux . . . 22
- 2.7 The Yocto Project . . . 22
- 2.8 Basic Hardware Terms . . . 26
- 2.9 Software Tools . . . 28
- 2.10 Device Management Service(DMS) . . . 29

3 State of the Art **30**
- 3.1 Literature Survey . . . 30

4 Evaluation of Onboarding Methods **33**
- 4.1 Evaluation Criteria . . . 33
- 4.2 Evaluation . . . 34
- 4.3 Advantages and Disadvantages . . . 46

5 Software Architecture Design for zero-touch onboarding to hawkBit using FDO standards **51**
- 5.1 Manual Onboarding to hawkBit Server . . . 51
- 5.2 Architecture Design . . . 55

6 Software Solution Implementation **60**
- 6.1 Device manufacturing . . . 60
- 6.2 FDO hawkBit Device Module . . . 60
- 6.3 FDO hawkBit Owner Module . . . 63

7 Validation and Testing of Prototype **67**
- 7.1 Quick Overview of FDO . . . 67
- 7.2 Server and Device Setup . . . 68
- 7.3 FDO Demonstration and Validdation . . . 70

7.4 Testing & Evaluation 73
7.5 Onboarding of Smart Energy Meter 76

8 Conclusion and Future Scope **77**

List of Figures

1.1 Number of Internet of Things (IoT) connected devices worldwide from 2015 to 2022, with forecasts from 2023 to 2025, from [**1**] 10

2.1 FIDO Device Onboard Entities and Entity Interconnection, from [**2**] 17
2.2 Graphical Representation of the FIDO Device Onboard Protocols, from [**2**] . . . 18
2.3 Kelyme Workflow (High level), from [**5**] 19
2.4 OPCUA Device Lifecycle, from [**7**] 22
2.5 Yocto Project Overview, from [**11**] 23
2.6 Yocto Project Workflow, from [**11**] 25
2.7 Raspberry Pi Compute Module 4 26
2.8 Compute Module 4 IoT Router Carrier Board Mini 27
2.9 IoT Edge Device complete hardware setup 27

4.1 Three Party Bootstrap Key Derivation Protocol [**3**] 35
4.2 Keylime Latency [**3**] 37
4.3 Scaling the Keylime Verifier (CV) on bare metal [**3**] 39
4.4 FDO Ownership Voucher [**2**] 40

5.1 Step 1: Log in to hawkBit Server 52
5.2 Step 2: Create a target(device) on hawkBit server 52
5.3 Step 3: Security Token generated 53
5.4 Step 4: Configure and register the device 54
5.5 Step 5: Confirm device status as registered 54
5.6 Management Service - Agent Interactions via ServiceInfo 55
5.7 Intel FDO Client Block Diagram [**27**] 56
5.8 Intel FDO Client Execution Flow [**27**] 57
5.9 FDO Device architecture with hawkBit device module 58
5.10 Zero touch onboarding to hawkBit using FDO Architecture 59

7.1 FIDO Device Onboard Entities and Entity Interconnection, [**2**] 67
7.2 database server 68
7.3 Manufacturer server 68
7.4 Rendezvous server 69
7.5 Owner server 69
7.6 hawkBit server 69
7.7 Adding all required data to the device 70
7.8 DI log from the Device, Device got its unique GUID and Serial number 70
7.9 Manufacturing Server Log, msg 13 means DI protocol complete 70
7.10 Sharing of Ownership voucher and triggering TO0 protocol 71
7.11 Manufacturer Log: Manufactured voucher for serial no lxfdo005 71

7.12 Owner Log: TO0 completed by Owner server 71
7.13 RV Log: RV server confirming TO0 completion 71
7.14 Device log : validates the TO2 protocol completed 72
7.15 hawkbit.config: validates that correct configurations are sent using T02 protocol . 72
7.16 hawkbit.log: validates the device is registered to hawkBit server 72
7.17 RV Log: RV server confirming TO1 completion 73
7.18 Owner server Log: Owner server confirming TO2 completion. msg71 means T02 completed 73
7.19 Latency of FDO to hawkBit server 74

List of Tables

2.1 Steps for FIDO Device Onboard 17

4.1 Keys used by keylime and their purpose [**3**] 35
4.2 Average TPM Operation Latency (ms)[**3**] 38
4.3 **Evaluation of Onboarding Methods** 44
4.4 Advantages and disadvantages IoT device onboarding methods 50

7.1 FDO Protocols and Messages 68
7.2 FDO Device latency 74

Listings

4.1 keylime_tenant command to provision keylime agent 34

5.1 swupdate suricatta daemon 53

6.1 fdolinuxclient.service 60
6.2 get_device_serial_number function 61
6.3 hawkbitOnboarding function 62
6.4 Receive serial number from device 63
6.5 createHawkbitTarget method 64
6.6 createHawkbitConfig and writeHawkbitConfig methods 64
6.7 confirmTargetRegistration method 65

1 Introduction

The world is at the threshold of a significant change that will revolutionize the way we live and work, specifically in smart cities, logistics, supply chains, things to do with transportation, and even the way we do agriculture and the way we manage precious resources such as water and energy. This transformation will be powered by the Internet of Things (IoT), which involves deploying numerous devices and collecting data to make better decisions. Linutronix GmbH is enthusiastic about the potential of IoT to benefit our customers, industry, and, eventually, citizens worldwide. However, to realize this potential, we must deploy and connect billions of devices safely, securely, efficiently, and cost-effectively. This thesis focuses on developing a key enabler to facilitate this transformation and automatically provide improved services to onboard millions of devices without human intervention. Traditional methods of onboarding devices, such as manual provisioning, can be time-consuming, error-prone, and impractical for these future-oriented large-scale IoT deployments.

To proceed, we need to understand the onboarding process for an IoT edge device. This process involves various steps to connect and configure the device to an IoT cloud platform, like Azure IoT, AWS IoT, hawkBit, and others, referred to as IoT Device Management Services (DMS). This research will consider an example of an IoT-enabled smart energy meter that needs to connect to the hawkBit Device Management Service. Here is a typical manual onboarding process for an IoT edge device or a smart energy meter. First, the commissioning technician installs the device physically and connect it to the internet. After that, he configures the device, such as the name, ID, and authentication token. These details are then provided to the device, which will authenticate to the IoT DMS. Ultimately, the technician will check if the device is functioning correctly. If yes, an IoT edge device is successfully onboarded onto an IoT DMS, enabling it to collect and transmit data and participate in the larger IoT ecosystem.

This thesis will evaluate state-of-the-art IoT device onboarding solutions and explore scalability, latency, cost, compatibility concerns, and other requirements and specifications for a successful IoT device onboarding technique. A combination of simulations, real-world use cases, and experimental testing will be used to design and assess the suggested solution. The findings of this study will open the door for more successful and scalable IoT deployments by offering insightful information about the viability and efficacy of the designed and existing solutions. The primary objective of this thesis is to develop and demonstrate a cutting-edge zero-touch IoT device onboarding solution that advances IoT technology and has the potential to enhance how we connect to and interact with IoT devices.

1.1 Motivation of Research

As mentioned earlier, The Internet of Things (IoT) is growing at a spectacular rate. We are on the verge of an explosion of IoT-related products and services, as per a report from IoT Analytics [1] The number of IoT devices worldwide is forecast to almost double from 14.4 billion in 2023 to more than 27 billion IoT devices in 2025. The graph of this projection can be visualized in Figure 1.1.

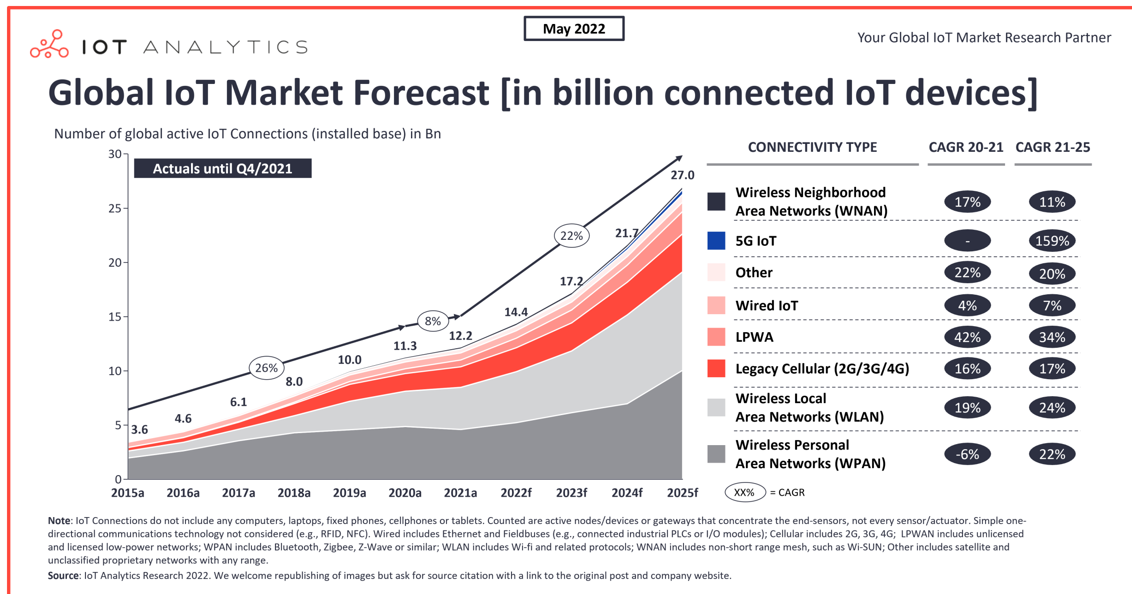


Figure 1.1: Number of Internet of Things (IoT) connected devices worldwide from 2015 to 2022, with forecasts from 2023 to 2025, from [**1**]

However, are we prepared for this huge growth? Onboarding these many devices one by one the first time or after IoT Hub reset (for security reasons) will be a cumbersome task. The current manual onboarding process has several drawbacks as given below:

1. Time-consuming: Manual onboarding requires manual configuration, and testing, which can be time-consuming and delay the deployment of IoT devices. As per this research[**21**] manual onboarding may take a minimum of 5 to 20 minutes.

2. Human Error: Manual configuration is prone to human error, which can lead to incorrect settings, security vulnerabilities, and malfunctioning devices.

3. Inconsistent Configuration: Manual configuration can result in inconsistent settings across different devices, making it harder to manage and maintain the network.

4. Security Risks: Manual configuration can increase security risks, as credentials and security settings can be mishandled, forgotten, or misconfigured, leading to potential security breaches.

5. Costly: Manual onboarding can be costly, as it requires human resources to perform the installation and configuration, increasing the deployment and operational costs.

6. Outdated software: IIoT devices are often equipped with outdated software at the time of commissioning, so they require an update.

These drawbacks can cause serious hurdles in achieving this growth. To address these challenges, a state-of-the-art zero-touch IoT device onboarding (ZTO) solution is needed to enable seamless and automated device provisioning. The IoT device only needs to be drop shipped to the point of installation, connected to the network and powered up. ZTO should do the rest.

1.2 Research Questions

To find out a suitable method for onboarding, we evaluate three different methods of IoT device onboarding: FIDO IoT device onboard from the Linux Foundation, Keylime solutions from MIT, and OPCUA device onboard from OPC foundation.

FIDO IoT device onboard is a method that utilizes the FIDO (Fast Identity Online) authentication protocol to securely and efficiently onboard IoT devices. This method aims to address the security concerns that arise during onboarding by providing a secure and streamlined authentication process.

Keylime solutions from MIT is another method that focuses on security during onboarding. This method uses Trusted Platform Modules (TPMs) to verify the integrity of IoT devices before allowing them onto a network. This ensures that only trustworthy devices are connected, thereby minimizing the risk of security breaches.

Finally, OPCUA device onboard from OPC foundation is a method that aims to improve the scalability and ease of use of IoT device onboarding. This method uses the OPC Unified Architecture (OPC UA) standard to provide a standardized and efficient process for onboarding a large number of devices onto a network.

The research questions are given below.

Research Question 1: What are the key differences between all these IoT Device onboarding solutions? Set up evaluation criteria and compare each method against them.

Research Question 2: What are the advantages and disadvantages of each of these solutions?

By evaluating the effectiveness of these three methods using various evaluation criteria (such as scalability, ease of use, and reliability), we aim to provide organizations with valuable insights into the strengths and weaknesses of each approach, enabling them to make informed decisions regarding onboarding their own IoT devices.

Research Question 3: How can we develop a zero-touch onboarding solution using FDO standards to efficiently onboard a large number of IoT devices to a hawkBit Device Management server?

Research Question 4: Which onboarding method is more suitable for smart energy meters (IoT-enabled energy meters) based on the merits and demerits of each method, and how does the suitability depend on the specific use cases of the smart energy meters?

1.3 Research Methodology

The research was built upon standard industry practices. To answer the first two research questions, we used "ISO/IEC/IEEE 42030 INTERNATIONAL STANDARD: Software, systems, and enterprise — Architecture evaluation framework" to set the evaluation criteria. We came up with 22 evaluation criteria. The comparison was theoretical and practical; we used sources such as official documentation, research papers, and feasibility studies to evaluate the methods. We implemented the Keylime Solutions and FDO onboarding to check evaluation criteria such as development efforts, required programming skills, and maintainability. The OPCUA device onboard standards were released in November 2022; therefore, the implementation was not yet available, so we could only evaluate them theoretically.

A prototype was created to answer the third research question, which comprises Raspberry Pi Compute Module 4 and IoT Router Carrier Board Mini. This prototype board is considered an IoT edge device, referred to as a device from here onward. A software solution is developed using FDO 1.1 standards such that the device will have a Linux client responsible for device onboarding. On the other hand, the owner server will have an onboarding module that provides the device with the correct DMS credentials. Apart from that, We used a docker container environment to create virtual servers, running the manufacturer, rendezvous, owner, and hawkBit servers in containers. We also created a docker image of the IoT edge device to simulate and test various evaluation criteria. We tested the scalability and latency by simulating a virtual onboarding process. We also tested the effects when multiple devices were simultaneously trying to connect to the owner onboarding server and documented the results.

The prototype device uses an embedded Linux Operating System built using the Yocto project. When we add a new feature to the OS, we have to change the Linux distribution, build the image, and flash it to the device. We did this to add the Linux client. The device has a serial console, which we can use to test and debug the newly added feature.

The fourth question was answered based on our evaluation so far, considering the merits and demerits of our use case.

1.4 Organisation of Report

The research project report is structured as follows.

- Chapter 2 we will learn about theories and concepts that readers should be familiar with, such as Device Onboarding, Zero Touch, Embedded Linux, The Yocto Project, FDO, keylime, and OPCUA. We will also make users familiar with the Hardware of devices, and software such as the hawkBit server,usbboot. It will also make readers well acquainted with various terms in IoT device onboarding.
- Chapter 3 presents the reader with the necessary state-of-the-art research in the field of IoT device onboarding. It will outline every method currently used to onboard IoT edge devices. It will also describe how we defined IoT device onboarding and zero-touch onboarding. It will also highlight what guidelines we have referred to come up with the evaluation criteria.

- Chapter 4 discusses What are the key differences between all these IoT Device onboarding solutions? It will make the reader well-informed about the advantages and disadvantages of each onboarding method.
- Chapter 5 will describe the architecture design of the software solution to perform zero-touch onboarding to the hawkBit server using fdo 1.1 standards.
- Chapter 6 is all about the realization of this software solution and how we have implemented it.
- Chapter 7 will validate our software solution. We will inform the readers how the prototype is following each subprotocol in the fdo 1.1 standard.
- Chapter 8 Concludes the thesis, It will interpret the findings of our research and highlight the future scope.

2 Background

2.1 IoT Device Onboarding

Based on our literature survey and industry practice, we can say that IoT device onboarding involves below steps:

1. Physical Installation: The device must be physically installed in its intended location and connected to a power source and the network.
2. Device Configuration: The device needs to be configured with basic settings such as device name, network settings, and security settings.
3. Authentication and Authorization: The device must be authenticated and authorized to access the Device Management Service, which may involve setting up security certificates and credentials.
4. Firmware and Software Updates: The device's firmware and software must be updated to the latest version to ensure it has the necessary features and security patches.
5. Testing and Validation: The device must be tested and validated to ensure it is correctly connected to the DMS and can communicate with the cloud.The device must be integrated with cloud services to enable data storage, analytics, and remote management.

2.2 Zero Touch IoT Device Onboarding

A Zero Touch IoT device onboarding requires no human intervention during installation. A technician will physically install the device and connect it to the Internet, but after that, the device will automatically configure itself and authenticate to the IoT Cloud. This means that the device can start performing its intended function right away without any additional input from humans.

We assume that the internet connection will be available to the device when it plugs in the LAN cable. The network or proxy setup is not included in the definition of Zero Touch IoT Device Onboarding.

2.3 FIDO Device Onboarding (FDO) [2]

The FIDO Device Onboard protocol is a recently established standard that ensures IoT devices' security and automatic onboarding. Its main aim is to simplify the provisioning and management of a large number of devices while providing high security. The protocol employs the FIDO (Fast Identity Online) authentication standards, widely used for secure online authentication.

It enables IoT devices to authenticate themselves to a new owner or network in a standardized way, thus making it possible to provision and configure devices without manual intervention. This flexible and extensible protocol supports various cryptographic mechanisms for device attestation. It can be implemented in hardware or software and supports different types of ownership transfer, including transfers between individuals, organizations, or cloud services. The essential advantage is it uses late binding. Traditional methods, such as manual configuration or pre-provisioning with keys or certificates, can be time-consuming and error-prone. FDO uses late binding such that the end owner will be able to choose on which cloud his device should be onboarded.

Another benefit of the FIDO Device Onboard protocol is that it provides robust security guarantees. The protocol employs cryptographic device attestation based on signed Entity Attestation Tokens (EATs), which provides strong evidence that the device is authentic and has not been tampered with. It is also designed to work with different networks and cloud services, enabling integration into existing IoT infrastructures.

The protocol involves interaction between various entities. The entities and commonly used terms are explained below:

- **Manufacturer (Mfg)**: Device manufacturer. A FIDO Device Onboard application runs in the factory, which implements the initial communications with the Device ROE, as part of the Device Initialize Protocol (DI) or appropriate substitute.
- **Device**: The device being manufactured, later the device being provisioned. This device has hardware and software configured on it, including a Device ROE and a Device to Manager Agent.
- **ROE**: A Restricted Operating Environment (ROE) refers to a specialized system consisting of both hardware and firmware components that are designed to create a secure and isolated environment for executing the essential functions and applications of a FIDO (Fast Identity Online) Device Onboard.
- **Owner Onboarding Service**: This is an entity constructed to perform FIDO Device Onboard protocols on behalf of the Owner. The Owner Onboarding Service is an application that executes on some platform already controlled by the Owner. After the protocols are completed, the Owner Onboarding Service transfers control of the device to the Owner's Manager, and never interacts with the device again. In FIDO Device Onboard, the Owner Onboarding Service is a component of the Manager, rather than a separate network service.
- **Rendezvous Server**: A network server or service (e.g., on the Internet) that acts as a rendezvous point between a newly powered on Device and the Owner Onboarding Service. It is expected that Internet versions of the Rendezvous Server will comprise multiple actual servers and service points; the reader will understand that Rendezvous Server in this document applies to the aggregate service.

- **Device Management Service**: The entity that uses the Owner Onboarding Service to take ownership of the Device, so that it can manage the device remotely using its own management techniques (protocols, etc.). During FIDO Device Onboard operation, the Management Service interacts with the Management Agent via the ServiceInfo key-value pairs. A common industry term for "Management Service"is "Device Management Service"(DMS). We have used hawkBit Server as Device Management Service for implementation purposes.
- **Management Agent**: The entity that uses the FDO Device software to allow the device ownership to be transferred using FIDO Device Onboard protocols. During FIDO Device Onboard operation, the Management Agent interacts with the Management Service via the ServiceInfo key-value pairs. We have created a Novel hawkBit onboarding module, which is an integral part of the linux-client that we have built. Here the linux-client is our FDO application and the swupdate suricatta daemon is our management agent.
- **Ownership Voucher**: The Ownership Voucher is a structured digital document that links the Manufacturer with the Owner. It is formed as a chain of signed public keys, each signature of a public key authorizing the possessor of the corresponding private key to take ownership of the Device or pass ownership through another link in the chain.

The protocol is further divided into 4 sub-protocols, These protocols are explained below.

- **Device Initialize Protocol (DI)**: The non-normative Device Initialize Protocol (DI) provides an example of a protocol that runs within the factory when a new device is completed. The protocol's function is to embed the ownership and manufacturing credentials into the newly created device's ROE. This prepares the device and establishes the first in a chain for creating an Ownership Voucher with which to transfer ownership of the device. The Device Initialize Protocol assumes that the protocol will be run in a safe environment. The trust model is Trust on First Use (TOFU).
- **Transfer Ownership Protocol 0 (TO0)**: Transfer Ownership Protocol 0 (TO0) serves to connect the Owner Onboarding Service with the Rendezvous Server. In this protocol, the Owner Onboarding Service indicates its intention and proves it is capable of taking control of a specific Device, based on the Device's current GUID.
- **Transfer Ownership Protocol 1 (TO1)**: Transfer Ownership Protocol 1 (TO1) is an interaction between the Device ROE and the Rendezvous Server that points the Device ROE at its intended Owner Onboarding Service, which has recently completed Transfer Ownership Protocol 0. The TO1 Protocol is the mirror image of the TO0 Protocol, on the Device side.
- **Transfer Ownership Protocol 2 (TO2)**: Transfer Ownership Protocol 2 (TO2) is an interaction between the Device ROE and the Owner Onboarding Service where the transfer of ownership to the new Owner actually happens.

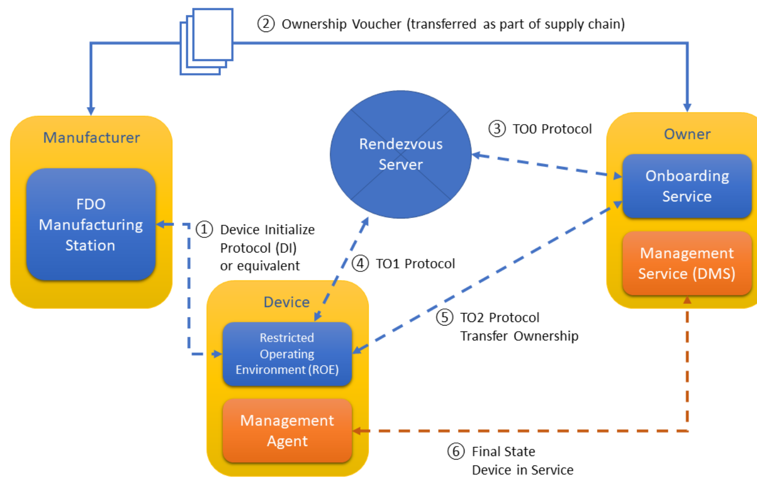


Figure 2.1: FIDO Device Onboard Entities and Entity Interconnection, from [**2**]

The overall working of the FDO can be understood using the above Figure 2.1 and the details about each step are given in Table 2.1

Serial Number	**Step Name**	**Function**
1	Device Initialize Protocol (DI)	Insertion of FIDO Device Onboard credentials into the device during the manufacturing process.
2	Ownership Voucher transfer	The Ownership Voucher is transferred to the device Owner From the manufacturer as a part of the supply chain.
3	Transfer Ownership Protocol 0 (TO0)	FDO Owner identifies itself to Rendezvous Server. Establishes the mapping of GUID to the Owner's IP address.
4	Transfer Ownership Protocol 1 (TO1)	Device identifies itself to the Rendezvous Server. Obtains mapping to connect to the Owner's IP address.
5	Transfer Ownership Protocol 2 (TO2)	Device contacts Owner. Establishes trust and then performs Ownership Transfer.
6	Final State: Device in Service	The Device authenticates itself to the Device Management Service and starts with its intended function.

Table 2.1: Steps for FIDO Device Onboard

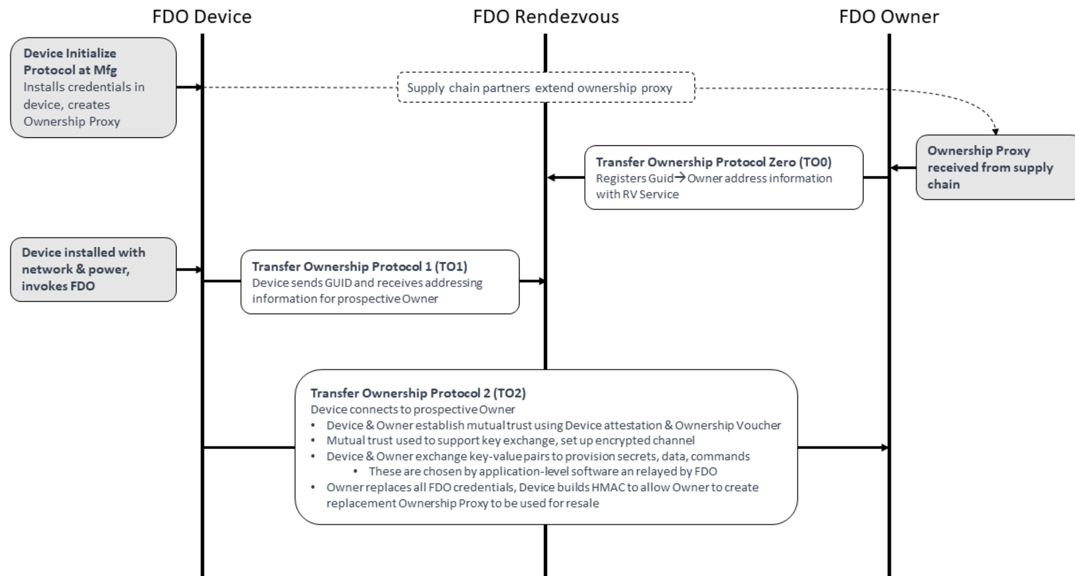


Figure 2.2: Graphical Representation of the FIDO Device Onboard Protocols, from [**2**]

2.4 Keylime[3]

Keylime is a cutting-edge CNCF-hosted project that offers a remote boot attestation and runtime integrity measurement solution. This state-of-the-art solution empowers users to easily monitor remote nodes with the help of a hardware-based cryptographic root of trust. Keylime originates in the pioneering security research team at MIT's Lincoln Laboratory.

keylime, a scalable, trusted cloud key management system. keylime provides an end-to-end solution for bootstrapping hardware-rooted cryptographic identities for IaaS nodes and system integrity monitoring of those nodes via periodic attestation. keylime offers a clean interface that allows higher-level security services like disk encryption or configuration management to leverage trusted computing without being trusted computing aware.

Regarding securing the cloud, Keylime is a reliable solution that offers Secure Bootstrapping, System Integrity Monitoring, Secure Layering with Virtualization Support, and Compatibility features. With Keylime, you can install an initial root secret into each cloud node safely and securely, monitor cloud nodes as they operate, leverage hardware-rooted cryptographic keys in software to secure services you already use and scale up quickly.

- **Keylime Tenant**: is a cloud resource user. Keylime's trusted key management system allows tenants to securely access and manage their data and applications in the cloud. The tenant is nothing but owner of the device.[**4**]
- **Keylime Registrar**: manages cryptographic identities for nodes in an IaaS cloud environment. The registrar bootstraps hardware-rooted cryptographic identities into physical and virtual cloud nodes, enabling organizations to maintain strong security controls over their data and applications while taking advantage of the benefits of cloud computing.[**4**]

- **Keylime Verifier** (or CV: Cloud Verifier): verifies the integrity of a tenant's virtualized infrastructure in an IaaS cloud environment. It provides integrity measurement, allowlists, and policy centralization for all integrity measurement activities, simplifying deployment and enhancing security. [**4**]
- **Keylime Agent** (Cloud Node): refers to a virtual machine or physical server instance running in an IaaS cloud environment. It is a computing resource that tenants provision to execute their applications in the cloud. Simply these are the IoT edge devices.[**5**]
- **TPM**: TPM stands for Trusted Platform Module. It's a security module designed to store cryptographic keys, system integrity measurements, and other security data in a secure location. TPMs are typically installed directly onto the motherboard of a computer or electronic device, providing a root of trust for the system. They're designed to protect against unauthorized access, tampering, and theft of sensitive information. TPMs have a range of applications, including secure boot, disk encryption, and digital rights management. In cloud computing, TPMs can provide hardware-rooted cryptographic identities for cloud nodes, increasing security and reducing insider threats.[**6**]

Keylime follows a simple three-step process: registration, measurement, and attestation.

The platform is given a key pair and associated credentials during registration using a trusted platform module (TPM) or similar hardware security module. This ensures the identification and authentication of the platform to the Keylime Registrar. In the measurement step, Keylime measures the platform's software stack and creates a cryptographic hash of the measurements. This hash is then signed with the platform's private key and stored on the Keylime Verifier. Finally, in the attestation step, the Keylime server remotely verifies the measurements and credentials of the platform. If they are valid, the Keylime server confirms that the platform is running trusted software that hasn't been tampered with. Cloud providers can use this attestation to enforce security policies. These steps can be easily understood from the below Figure. Remember that this is a very high-level overview. A detailed explanation can be found in the research paper "Bootstrapping and Maintaining Trust in the Cloud."

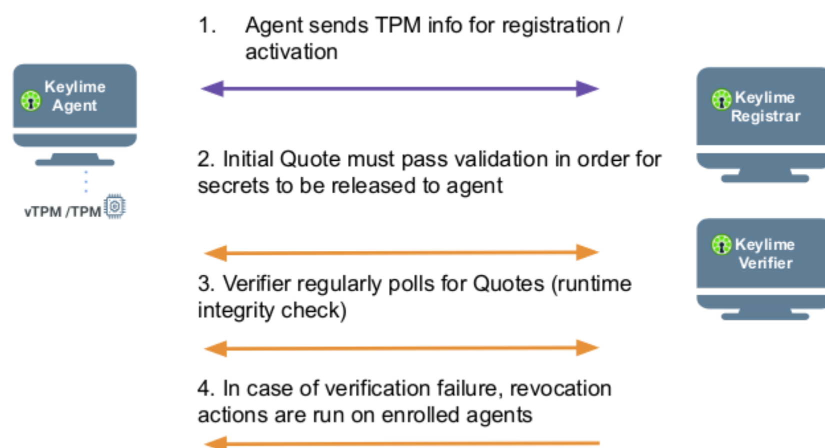


Figure 2.3: Kelyme Workflow (High level), from [**5**]

2.5 OPCUA Device Onboard

The OPC UA Specification for Device Onboarding is an essential industry standard that outlines the complete life cycle of devices and composites. It also establishes mechanisms to verify their authenticity, set up their security, and maintain their configuration. The main goal of this specification is to make it easier for multiple vendors to develop applications that can interoperate seamlessly.

The onboarding model is designed to ensure that the configuration of a device can be managed over its entire life cycle, from when it is manufactured to when it is decommissioned. This is particularly important because devices, unlike PC-class computers, are typically shipped with automation software pre-installed and are connected directly to sensitive networks. Therefore, it is crucial to have a process to authenticate devices before they are given access to a sensitive network.

The complete life cycle of a device involves several stages, each with its requirements and challenges that must be addressed to ensure secure device onboarding. These stages include manufacturing, shipping, installation, commissioning, operation, maintenance, and decommissioning.

Below are some definitions for the essential terms used in these onboarding guidelines. We will use these terms to compare them with other onboarding methods theoretically.

- **Device**: A computer that can communicate via a network. A Device has a unique identifier and may have one or more Applications
- **Composite**: A collection of Devices or Composites assembled into a single unit. Each Composite has a unique identifier and may appear as a single Device on a network or multiple Devices.
- **Application**: A program that runs on a Device. Each Application has a unique identifier and communicates with other Applications on the network.
- **OwnerOperator**: An organization deploying and operating a system comprising devices, Composites, or other computers connected via a network.
- **Manufacturer**: An organization that creates Devices.
- **CompositeBuilder**: An organization that creates Composites.
- **Distributor**: An organization that re-sells Devices and/or Composites. A Distributor enhances Devices and Composites by adding customized products or services before resale.
- **SystemIntegrator**: An organization that installs and configures a system for an OwnerOperator that comprises Devices, Composites, or other computers connected via a network.
- **Registrar**: an OPC UA Application that registers and authenticates Devices added to the network.
- **RegistrarAdmin**: A user authorized to change the configuration of the Registrar.
- **SoftwareUpdateAdmin**: A user authorized to update the firmware running on a Device.
- **SecurityAdmin**: A user authorized to change security configuration for Clients and Servers running on the network.

- **Device Manufacture**: A Device is created and a DeviceIdentity Certificate is assigned. This Certificate is provided when the Device is transferred to other actors. During Device Manufacture, Applications may be installed on the Device. A Ticket describing the Device is created and signed by the Manufacturer.
- **Composite Assembly**: A Composite is created from Devices and a unique identity is assigned to the Composite. This identity is provided when the Composite is transferred to other actors. During Composite Assembly, Applications may be installed on the Devices contained in the Composite. A Ticket describing the Composite is created and signed by the CompositeBuilder.
- **Distribution**: The Device or Composite is stored until it is delivered to a CompositeBuilder, SystemIntegrator, OwnerOperator, or another Distributor.
- **Onboarding**: The SystemIntegrator connects a Device to the network and verifies that the identity reported by the Device matches the identity in a Ticket provided by the Manufacturer or CompositeBuilder.
- **Application Setup**: The SystemIntegrator configures the Applications running on the Device or Composite so they can communicate with other Applications running in the system. This process includes distributing TrustLists and issuing Certificates.
- **Configuration**: The OwnerOperator performs tasks that are not done while the Device is in full operation, such as updating firmware, installing new Applications, or changing Application configuration.
- **Operation**: The Device does the tasks it was deployed to do.
- **Decommissioning**: The Device has all access revoked and, if the Device is still functional, then it is reset to the default factory settings.

The OPCUA provides a holistic approach to Device Onboarding, The complete workflow of the device lifecycle can be visualized by the below Figure. This is just a high-level overview to make readers familiar with the concepts, the technical details can be found in the ÖPC Unified Architecture Part 21: Device Onboarding".[**7**]

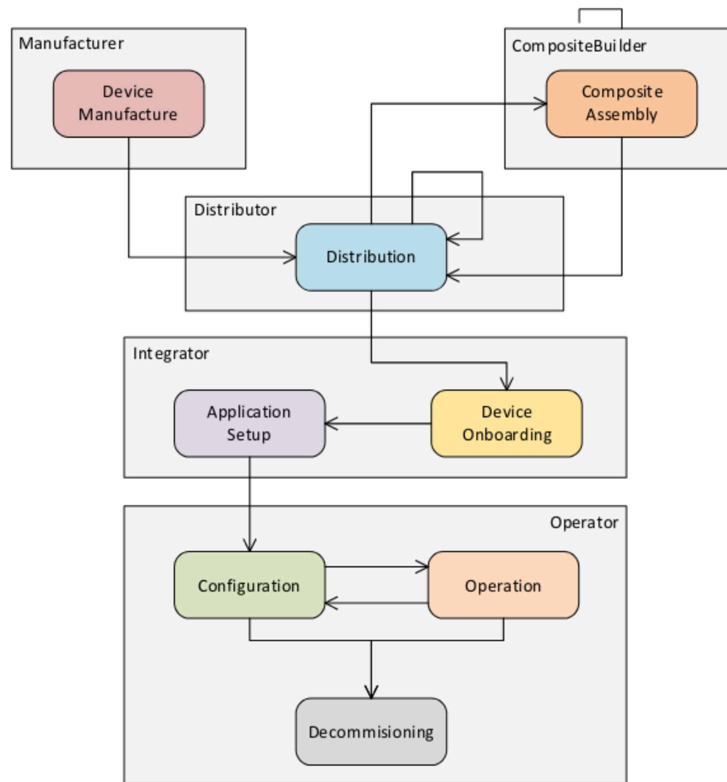


Figure 2.4: OPCUA Device Lifecycle, from [7]

2.6 Embedded Linux

"An embedded device is a microprocessor-based system that is built to control a function or range of functions and is not designed to be programmed by the end user in the same way a PC is" [80]. This definition indicates that embedded devices are often implemented on non-PC platforms, which also results in changes in the available computing resources – often those are constrained in some way(s). The definition also shows the modification of or insight into embedded devices is more difficult to achieve than for PC platforms.[**8**]

The term "Embedded Linux" is used to describe an embedded operating system that is running a Linux kernel inside. Embedded Linux devices are special-purpose computers running the Linux kernel and just the necessary GNU utilities to help achieve the specific purpose the device is built for.[**10**]

2.7 The Yocto Project

The Yocto Project is a Linux Foundation collaborative open-source project whose goal is to produce tools and processes that enable the creation of Linux distributions for embedded and IoT software that are independent of the underlying architecture of the embedded hardware. The project was

announced by the Linux Foundation in 2010 and launched in March, 2011, in collaboration with 22 organizations, including OpenEmbedded. To make readers the yocto project easy to understand this complete section is referred from [**11**]

The Yocto Project's focus is on improving the software development process for embedded Linux distributions. The Yocto Project provides interoperable tools, metadata, and processes that enable the rapid, repeatable development of Linux-based embedded systems in which every aspect of the development process can be customized.

The Yocto Project has the aim and objective of attempting to improve the lives of developers of customized Linux systems supporting the ARM, MIPS, PowerPC and x86/x86-64 architectures. A key part of this is the OpenEmbedded build system, which enables developers to create their own Linux distribution specific to their environment. The Yocto Project and OpenEmbedded Project share maintainership of the main parts of the OpenEmbedded build system: the build engine, BitBake, and the core metadata, OpenEmbedded-Core. The Yocto Project provides a reference implementation called Poky, which contains the OpenEmbedded build system plus a large set of recipes, arranged in a hierarchical system of layers, that can be used as a fully functional template for a customized embedded operating system.

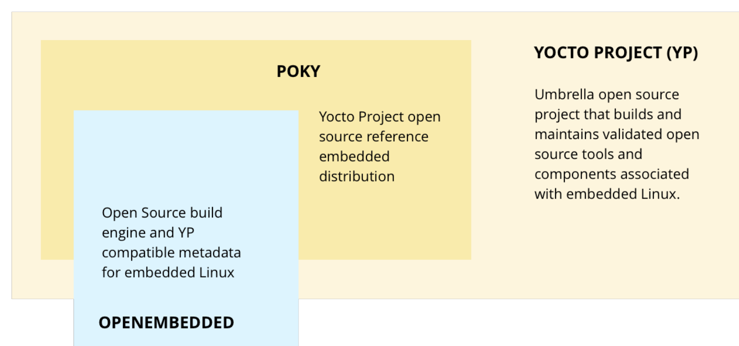


Figure 2.5: Yocto Project Overview, from [**11**]

2.7.1 The Layer model

Yocto Project has a development model for embedded Linux creation which distinguishes it from other simple build systems. It is called the Layer Model.

The Layer Model is designed to support both collaboration and customization at the same time. Layers are repositories containing related sets of instructions which tell the build system what to do. Users can collaborate, share, and reuse layers. Layers can contain changes to previous instructions or settings at any time.[**11**]

This powerful override capability is what allows you to customize previous collaborative or community supplied layers to suit your product requirements.

2.7.2 Terms Of Reference

- **Configuration Files**: Files which hold global definitions of variables, user defined variables and hardware configuration information. They tell the build system what to build and put into the image to support a particular platform.
- **Recipe**: The most common form of metadata. A recipe will contain a list of settings and tasks (instructions) for building packages which are then used to build the binary image. A recipe describes where you get source code and which patches to apply. Recipes describe dependencies for libraries or for other recipes, as well as configuration and compilation options. They are stored in layers.
- **Layer**: A collection of related recipes. Layers allow you to consolidate related metadata to customize your build, and isolate information for multiple architecture builds. Layers are hierarchical in their ability to override previous specifications. You can include any number of available layers from the Yocto Project and customize the build by adding your layers after them. The Layer Index is searchable for layers within Yocto Project.
- **Metadata**: A key element of the Yocto Project is the meta-data which is used to construct a Linux distribution, contained in the files that the build system parses when building an image. In general, Metadata includes recipes, configuration files and other information refering to the build instructions themselves, as well as the data used to control what things get built and to affect how they are built. The meta-data also includes commands and data used to indicate what versions of software are used, and where they are obtained from, as well as changes or additions to the software itself (patches or auxiliary files) which are used to fix bugs or customize the software for use in a particular situation. OpenEmbedded Core is an important set of validated metadata.
- **OpenEmbedded-Core**: oe-core is meta-data comprised of foundation recipes, classes and associated files that are meant to be common among many different OpenEmbedded-derived systems, including the Yocto Project. It is a curated subset of an original repository developed by the OpenEmbedded community which has been pared down into a smaller, core set of continuously validated recipes resulting in a tightly controlled and an quality-assured core set of recipes.
- **Poky**: A reference embedded distribution and a reference test configuration created to 1) provide a base level functional distro which can be used to illustrate how to customize a distribution, 2) to test the Yocto Project components, Poky is used to validate Yocto Project, and 3) as a vehicle for users to download Yocto Project. Poky is not a product level distro, but a good starting point for customization. Poky is an integration layer on top of oe-core.
- **Build System - "Bitbake"**: a scheduler and execution engine which parses instructions (recipes) and configuration data. It then creates a dependency tree to order the compilation, schedules the compilation of the included code, and finally, executes the building of the specified, custom Linux image (distribution). BitBake is a make-like build tool. BitBake recipes specify how a particular package is built. They include all the package dependencies, source code locations, configuration, compilation, build, install and remove instructions. Recipes also store the metadata for the package in standard variables. Related recipes are consolidated into a layer. During the build process dependencies are tracked and native or

cross-compilation of the package is performed. As a first step in a cross-build setup, the framework will attempt to create a cross-compiler toolchain (Extensible SDK) suited for the target platform.

- **Packages**: The output of the build system used to create your final image.
- **Extensible Software Development Kit (ESDK)**: A custom SDK for application developers that allows them to incorporate their library and programming changes back into the image to make their code available to other apps developers.
- **Image**: A binary form of a Linux distribution (operating system) intended to be loaded onto a device.

2.7.3 The General Workflow

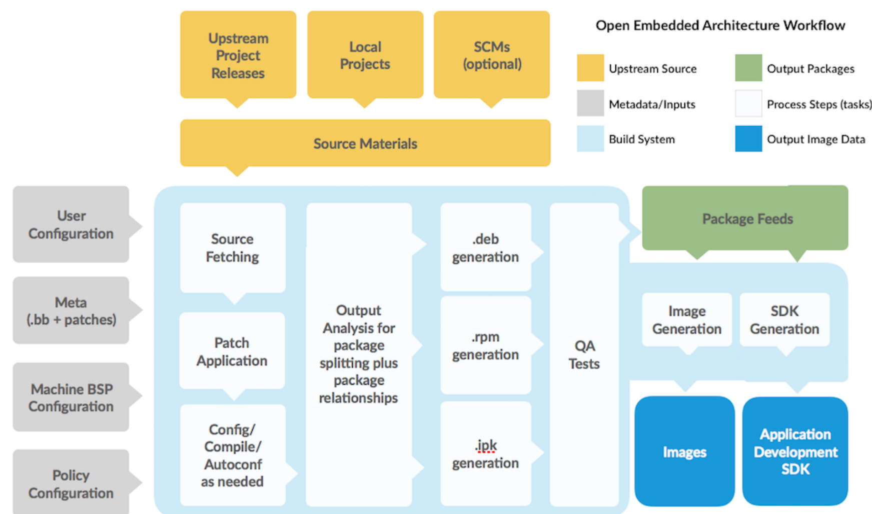


Figure 2.6: Yocto Project Workflow, from [**11**]

- To begin, developers specify architecture, policies, patches and configuration details.
- The build system then fetches and downloads the source code from where ever specified. The project supports standard methods such as tarballs or source code repositories systems such as git.
- Once downloaded, the sources are extracted into a local work area where patches are applied and common steps for configuring and compiling the software will be run.
- The software is then installed into a temporary staging area where the binary package format you select (deb, rpm, or ipk) will be used to roll up the software.
- Different QA and sanity checks are run throughout entire build process.
- After the binaries are created, a binary package feed is generated which is then used to create the final root file image.

- The file system image is generated.

2.8 Basic Hardware Terms

- The **Raspberry Pi Compute Module 4** is specifically designed for industrial and embedded systems use. As such, it can be used in digital signage, thin clients, and process automation. It's built around the same processor as the Raspberry Pi 4 and, as a result, delivers increased performance compared to its predecessors.[**12**]

  We are using Raspberry Pi Compute Module with following specs:

  1. Broadcom BCM2711 quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
  2. 2GB LPDDR4-3200 SDRAM
  3. 16GB eMMC Flash memory
  4. Bluetooth 5.0, BLE
  5. 2.4 GHz, 5.0 GHz IEEE 802.11 b/g/n/ac wireless
  6. On-board electronic switch to select either external or PCB trace antenna

Figure 2.7: Raspberry Pi Compute Module 4

- **Compute Module 4 IoT Router Carrier Board Mini** is an internet expansion board based on the Raspberry Pi Compute Module 4. When connecting with a gigabit network card via PCle, it brings Raspberry Pi CM4 two full-speed gigabit network ports and offers better performance, lower CPU usage, and higher stability for a long time work compared with a USB network card. Besides, with a mini size of 55 x62mm, the board still retains the characteristic GPIO pin header of Raspberry Pi, which makes it applicable for connecting other actuators, sensors, and smart cooling fan. Furthermore, there is also a USB2.0 interface that can be connected to mobile hard drives, printers, WIFI modules, LTF modules, etc. The powerful performance of BCM2711 4 core 1.5GHz Cortex-A72 and the rich software support in the Raspberry Pi community make this board a solid foundation for building high-performance gateways, smart routers, and home IoT platforms. It can also be connected to peripherals and used as a mini-NAS, wireless network bridge, or LTE Internet terminal.[**13**]

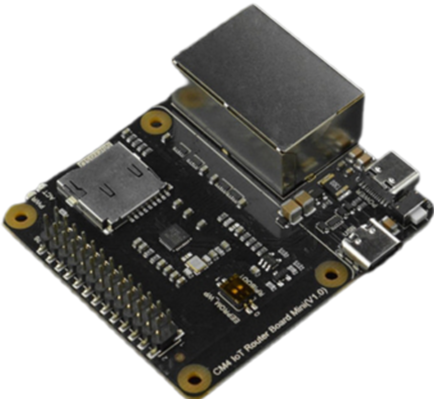


Figure 2.8: Compute Module 4 IoT Router Carrier Board Mini

- The prototype of IoT Edge Device is made up of these two boards and can be visualised from Figure. 2.9

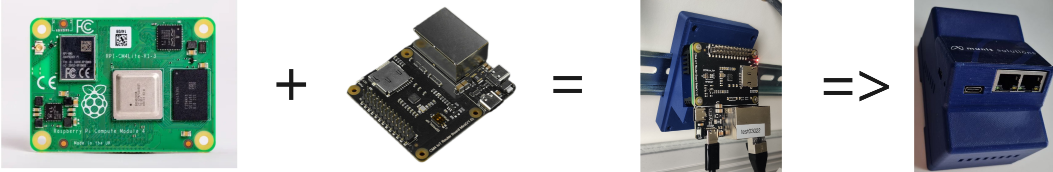


Figure 2.9: IoT Edge Device complete hardware setup

2.9 Software Tools

Now we have got an idea about how the embedded Linux image is generated using the Yocto project and logic unit hardware which we need to flash with that image. This section will give brief idea about what are the tools required for flashing the hardware. First we need to connect our logic unit with host device which will be used to flash the image. This can be done using 'USBBOOT' tool from raspberry-pi. Once the device has established the connection we can flash it using 'bmap-tools' which is developed by Intel.

2.9.1 usbboot

source code:https://github.com/raspberrypi/usbboot

This is the USB MSD boot code which supports the Raspberry Pi 1A, 3A+, Compute Module, Compute Module 3, 3+ and 4, Raspberry Pi Zero and Zero 2 W.

The default behaviour when run with no arguments is to boot the Raspberry Pi with special firmware so that it emulates USB Mass Storage Device (MSD). The host OS will treat this as a normal USB mass storage device allowing the file system to be accessed. If the storage has not been formatted yet (default for Compute Module) then the Raspberry Pi Imager App can be used to install a new operating system.

Since RPIBOOT is a generic firmware loading interface, it is possible to load other versions of the firmware by passing the -d flag to specify the directory where the firmware should be loaded from. E.g. The firmware in the msd can be replaced with newer/older versions.[**14**]

2.9.2 bmap-tools

source code:https://github.com/intel/bmap-tools

This is also know as 'the better dd for embedded projects, based on block maps.' bmaptool is a generic tool for creating the block map (bmap) for a file and copying files using the block map. The idea is that large files, like raw system image files, can be copied or flashed a lot faster and more reliably with bmaptool than with traditional tools, like dd or cp.[**15**]

- Faster. Depending on various factors, like write speed, image size, how full is the image, and so on, bmaptool was 5-7 times faster than dd in the Tizen IVI project.
- Integrity. bmaptool verifies data integrity while flashing, which means that possible data corruptions will be noticed immediately.
- Usability. bmaptool can read images directly from the remote server, so users do not have to download images and save them locally.
- Protects user's data. Unlike dd, if you make a mistake and specify a wrong block device name, bmaptool will less likely destroy your data because it has protection mechanisms which, for example, prevent bmaptool from writing to a mounted block device.

2.9.3 Docker-Container

We have used Docker-Container to perform simulated testing of novel software solutions. Hence you must get familiar with it to understand our testing and validation section.

- A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.[**9**]
- Docker Engine is an open-source containerization technology for building and containerizing your application
- Docker Compose is a tool that was developed to help define and share multi-container applications. With Compose, we can create a YAML file to define the services and with a single command, can spin everything up or tear it all down.

2.10 Device Management Service(DMS)

2.10.1 hawkBit Server

Eclipse hawkBit™ is an open-source,domain-independent back-end framework for rolling out software updates to constrained edge devices as well as more powerful controllers and gateways connected to IP-based networking infrastructure [**16**]. hawkBit is a cloud-ready infrastructure that we are going to use as DMS.

2.10.2 SWupdate Suricatta Daemon

Swupdate Suricatta Daemon and hawkBit server work together to provide a complete software update management system. hawkBit acts as a central hub for updates while Suricatta Daemon is responsible for executing the update process on target devices. The integration between the two enables centralized monitoring and control of updates across multiple devices, with advanced features such as staged rollouts for testing and validation of updates before wider distribution. Suricatta Daemon manages the download, verification, and installation of updates, ensuring optimal device performance, security, and functionality.

3 State of the Art

3.1 Literature Survey

3.1.1 Evaluation Criteria

The thesis aims to evaluate a suitable solution for large-scale IoT deployments. To do so, evaluating all the state-of-the-art methods available for device onboarding becomes crucial.

We wanted to set evaluation criteria that would give us a fair evaluation. The evaluation criteria were selected based on the software industry's best practices and standards. We have referred to ISO/IEC/IEEE 42030 International Standards. These standards are specifically designed to provide a framework for software architecture evaluation. According to this standard, Architecture evaluation judges how well architecture objectives have been or will be achieved. It can provide answers to an identified set of questions too, for example, provide inputs to strategic decision-making (such as whether it would be cheaper, in the long run, to modify an existing architecture to close value gaps) or to produce a new architecture that better addresses current and future stakeholder needs. An architecture evaluation can also provide inputs to decisions made at the operational and tactical levels. For example, the evaluation may provide helpful information regarding the capability limitations of the entity in question. [**17**]

In their paper-A Framework for evaluating software technology published in IEEE Software in 1996, Alan W Brown and Kurt C. Wallnau propose a novel experimental framework for helping organizations make informed decisions when investing in new software technologies. The authors argue that their systematic approach, which involves modeling and experiments, can assist companies in evaluating a new technology's features and comparing them with those of their peers and competitors. Companies can use this framework to make more informed decisions about which technologies to invest in and how to maximize their benefits. Although it is old, this paper has also guided us in setting the evaluation criteria[**18**]. We have also referred to-Analysis of Alternatives (AoA) Handbook A Practical Guide to Analysis of Alternatives. By US Air Force, This handbook helped us identify how to choose a particular technology when alternatives are available.[**19**]

Ultimately, we devised 22 criteria to compare and evaluate FDO, Keylime, and OPCUA methods for IoT Device Onboarding, which are discussed in detail in the next topic.

3.1.2 IoT Device Onboarding

IoT device onboarding is a part of the IoT deployment process. It refers to configuring a new device to an IoT Device Management Server (DMS). Microsoft published a whitepaper in collaboration with Globalsign, Eurotech, and Infineon. They mentioned that there are two main parts to the process when it comes to onboarding a new device to a cloud-served IoT network. The first step is to establish the initial connection to the cloud-accessed network. Once that connection is established, the next task is configuring the Device to perform its intended task.[**20**]

According to research carried out at Samsung in collaboration with Purdue Univerisity, "Device Onboarding refers to the first-time registration of a device into an Internet of Things (IoT) network. It involves securely sharing authentication-related information between the already onboarded Device and the Device to be onboarded. It can have multiple non-trivial steps." [**21**]

As per the OPC foundation, Onboarding is the process where a Device or Composite is connected to the network managed by an organization. When this happens, the Device's authenticity is verified via interactions with a Registrar running on the network. The commonly understood concept of "Commissioning" is represented by the Onboarding, Application Setup, and Configuration stages. Another aspect of onboarding is that if the Device is commissioned after a long period from the manufacturing date. The software and OS of the Device may be outdated and should be updated to avoid any security risks and device malfunctioning.[**7**]

Based upon this literature survey and our own understanding, we came up with steps involved in IoT Device Onboarding, which are mentioned in section 2.1

3.1.3 Zero Touch IoT Device Onboarding

To overcome the drawbacks of manual device onboarding, various organizations have worked to develop a zero-touch onboarding (ZTO) solution. We will refer to their definitions of Zero-touch and ultimately come up with our understanding.

According to Microsoft, The goal is Zero Touch Provisioning, where a handshake triggered when a device is powered on initiates onboarding and a subsequent automated provisioning process. [**22**], the following steps should be included in ZTO:

Automatic and secure onboarding to a production certificate provider. Receipt of device operational certificate credentials. Automatic provisioning to cloud application services. Automation of credentials renewal and lifecycle management.

Intel is working on its product FIDO Device Onboard, which is also based on FDO 1.1 standards. It says, "Zero-Touch. Zero-Worries. Intel® FIDO Device Onboard (FDO) is an automated "Zero-Touch" onboarding service. To more securely and automatically onboard and provision a device, it only needs to be drop shipped to the point of installation, connected to the network, and powered up. SDO does the rest." This means the IoT devices should be plug-and-play. [**23**]

As per this study(“Secure & Zero Touch Device Onboarding,” p. 1), “The solution facilitates a process where when an IoT device is installed and establishes its first connection; it is registered and connected automatically and securely, as a bonafide and fully trusted device, into the cloud-based IoT platform. The platform then adopts the device as a secure device that it monitors and maintains over the air (OTA). This overall process is known as “Zero-Touch Device Onboarding (ZDO).”[**24**]

These definitions suggest the device should be connected to the Internet once plugged in the network cable or connected to the WiFi. For our research purpose, we will also assume that the device does not need any network configuration setup or proxy setup. We plug in the Ethernet cable, and the Internet will be there. Hence, We will define Zero Touch Device onboarding as given below:

A Zero Touch IoT device onboarding requires no human intervention during installation. A technician will physically install the device and connect it to the Internet, but after that, the device will automatically configure itself and authenticate to the IoT Cloud. This means that the device can start performing its intended function right away without any additional input from humans.

3.1.4 Device Onboarding to hawkBit Server

To set the scope of our thesis, we initially selected three onboarding methods. Each method has its merits, demerits, and specific use case.

The Keylime solution is based on the research paper "Bootstrapping and Maintaining Trust in the Cloud."[**3**] We have also referred to the documentation of keylime for implementation [**4**]. The OPC Foundation has just released standards for Device onboarding in ÖPC Unified Architecture Part 21: Device Onboarding"[**7**]. The documentation number is OPC 10000:21. The method FIDO Device Onboard was more suitable for our application and large-scale IoT deployments. To understand this method, we referred FIDO Device Onboard Specification given by FIDO Alliance. The complete documentation is sufficient to understand the protocol. We have also referred to documentation from the Linux Foundation, which provided a practical overview and implementation guidelines.[**25**]

The existing FDO implementation has no module that can onboard the IoT device to the hawkBit server. The FIDO alliance expects the industry to work on its own DMS and develop a module that can onboard the device to the DMS. Such as, Microsoft will create an Azure IoT onboarding module and contribute to the source code of FDO. Similarly, We have implemented a cutting-edge method and created a unique software solution for IoT devices to be onboarded to hawkBit DMS. It is the first time IoT devices are utilizing the Late Binding feature of FDO to onboard to the hawkBit server. As part of our research, we developed a hawkBit onboarding module to help expand our knowledge and contribute to the open-source code of FDO. Our research has resulted in a revolutionary Zero Touch IoT Device Onboarding solution that eliminates the need for pre-configuration during manufacturing. All we have to do is power on the device and connect it to the internet, and our solution will onboard the IoT device to the Hawkbit server automatically. Our testing has shown that it is highly scalable and can onboard multiple devices simultaneously.

4 Evaluation of Onboarding Methods

We will review each criterion and compare FDO, Keylime, and OPCUA onboarding. We aim to compare each method on fair grounds so that this evaluation will be helpful for organizations to select suitable onboarding methods. Research Question 1 seeks to identify the critical differences between these IoT device onboarding solutions by setting up evaluation criteria and comparing each method against them. By doing so, this research aims to provide a comprehensive understanding of each solution's strengths and limitations, which can assist organizations in selecting the most suitable approach for their specific needs. Section 4.2 will answer research question 1.

Research Question 2 complements this analysis by exploring the advantages and disadvantages of each solution, allowing organizations to make informed decisions regarding onboarding their own IoT devices based on a thorough understanding of the strengths and limitations of each method. Section 4.3 will answer research question 2.

4.1 Evaluation Criteria

As discussed in section 3.1, we have developed the following evaluation criteria.

1. Zero Touch: Is it a complete zero-touch onboarding solution?
2. Hardware requirement: Basic hardware components required for device onboarding
3. Computing Requirements: What is the minimum computing architecture it supports?
4. Time/latency: Time taken for each device to complete the onboarding
5. Efforts: Efforts required to onboard a device
6. Scalability: How many devices can be onboarded to a server
7. Reliability: Can the device connect to the server reliably in case of network failure/ power failure during onboarding.
8. Maintainability: Can we maintain and provide service for this method, solution maintainability
9. Ease of manufacturing: Is the method provides ease of manufacturing? How?
10. Ease of distribution/life cycle management: Is the method provides a secure distribution, such as the device can have a chain of trust?
11. Expertise required: What level of expertise the technician needs to have to onboard the device

12. Development efforts: How much effort a developer needs to spend to implement the provided method
13. Programming skills: Which programming language and skills needed as a developer
14. User Friendliness: does the end user finds this mechanism valuable and easy?
15. Overall cost: Overall cost, including hardware+software+on-boarding
16. Security: Does this method provide proper security guidelines for each stakeholder?
17. Runtime integrity: Can this method provide runtime integrity(trust) measurement?
18. Software update: Does this method takes care of software update?
19. Certificate management: Does this method takes care of certificate management
20. Device Attestation: How does this method take care of device attestation
21. Proof of ownership: How does this method take care of proof of ownership
22. Correlation Attack Concerns: Are the device onboarding credentials used for application provisioning?

4.2 Evaluation

- **Criteria 1:** Zero Touch: Is it a complete zero-touch onboarding solution?

  FDO: FDO is a complete zero-touch onboarding solution. It does not need any manual input from the installation/commissioning technician for device onboarding.

  Keylime: Keylime is not a zero-touch solution. It uses the "Three Party Bootstrap Key Derivation Protocol". It can be explained using the below Figure 4.1. Here The Commissioning technician must share the Keylime agent's IP Address, UUID, and Port With the tenant and eventually to the cloud verifier. It means the technician must manually enter the input and run a CLI command to proceed with onboarding. The keylime_tenant utility can be used to provision the keylime agent. For example, the following command tells Keylime to provision a new agent at 127.0.0.1 with UUID d432fbb3-d2f1-4a97-9ef7-75bd81c00000 and talk to a verifier at 127.0.0.1. Finally, it will encrypt a file called file to send and send it to the agent, allowing it to decrypt it only if the configured TPM policy is satisfied[**26**]:

```
keylime_tenant -c add -t 127.0.0.1 -v 127.0.0.1 -u D432fbb3-d2f1-4a97-9ef7-75bd81c00000
-f filetosend
```

  Listing 4.1: keylime_tenant command to provision keylime agent

  OPCUA Onboarding: The OPCUA Onboarding documentation never mentions a word such as zero touch. However, it uses Automatic, which eventually means without human intervention. When a CompositeBuilder or Integrator receives a shipment of Devices, it needs to connect them to its network and verify their authenticity. This process is automated using a Registrar that detects new Devices added to the network, inspects their DeviceIdentity Certificates, and finds the corresponding DeviceIdentityTicket. If a match is found, the Device is accepted and can be provisioned for use on the network.

Key	Type	Purpose
EK	RSA 2048	Permanent TPM credential that identifies the TPM hardware.
SRK	RSA 2048	TPM key that protects TPM created private keys when they are stored outside the TPM.
AIK	RSA 2048	TPM key used to sign quotes.
K_e	AES-256	Enrollment key created by the registrar and used to activate the AIK.
K_b	AES-256	Bootstrap key the tenant creates. keylime securely delivers to the node.
U, V	256-bit random	Trivial secret shares of Kb, derived with random 256-bit V: U = Kb ⊕ V.
NK	RSA 2048	Non-TPM software key used to protect secret shares U, V in transit.

Table 4.1: Keys used by keylime and their purpose [**3**]

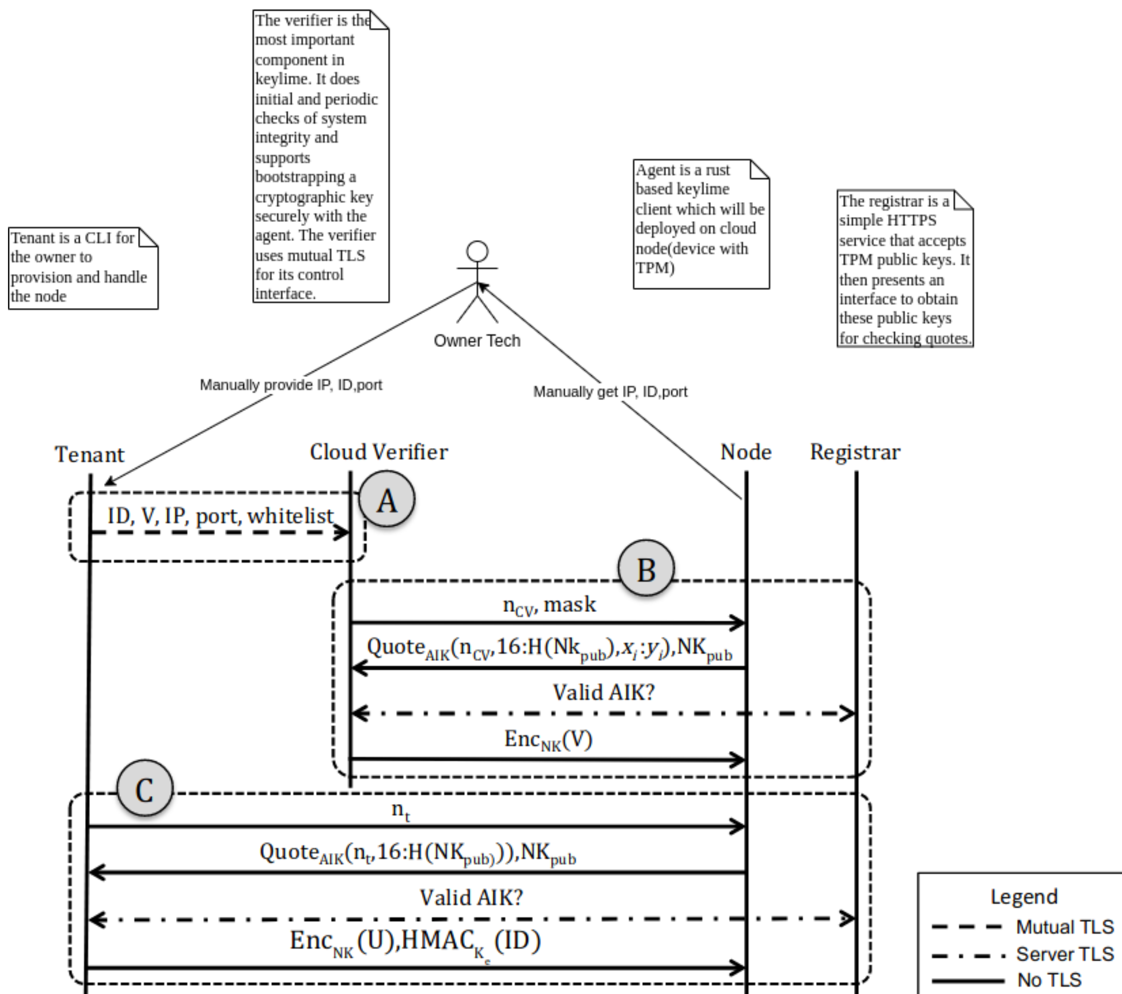


Figure 4.1: Three Party Bootstrap Key Derivation Protocol [**3**]

The 'Three Party Key Derivation Protocol' used in keylime is explained below in simplified words:

1. The tenant generates a random encryption key (Kb) and uses it to encrypt sensitive data that will be sent to the cloud node.
2. The tenant splits Kb into two parts: U and V. The tenant keeps U and shares V with a component called the Cloud Verifier (CV), which is responsible for ensuring the integrity of the cloud node.
3. The tenant requests the cloud service provider to create a new cloud node and sends the encrypted data (EncKb(d)) to the provider as part of the node creation process.
4. The provider/manufacturer/installation technician assigns a unique identifier (UUID) and an IP address to the newly created node and shares this information with the tenant.
5. The tenant notifies the Cloud Verifier (CV) about their intent to boot the cloud node. The tenant provides information about the node, such as UUID, IP address, and a TPM policy (which specifies acceptable values for certain security measures).
6. The attestation protocol begins between the CV, the cloud node, and the tenant. It involves exchanging messages to verify the integrity of the cloud node's Trusted Platform Module (TPM) and establish a secure connection.
7. The tenant and the CV request a TPM quote from the cloud node. The quote contains information about the TPM's identity and the values of certain measurements.
8. The CV validates the TPM quote to ensure the integrity of the cloud node according to the tenant's whitelist policy. The tenant validates the TPM quote to verify the identity of the cloud node's TPM.
9. Once the TPM quote is validated, the tenant and the CV can securely transmit their shares of the encryption key (Kb) to the cloud node. The tenant sends EncNK(U), and the CV sends EncNK(V).
10. The cloud node decrypts the encrypted data using the received shares of Kb (U and V) and can proceed with the boot or startup process.
11. The cloud node stores the share U in its Trusted Platform Module's non-volatile memory (NVRAM) to avoid needing the tenant's interaction in case of node reboot or migration. If rebooting or migrating, the node must be verified by the CV again to obtain the share V and re-derive Kb.

- **Criteria 2:** Hardware requirement: Basic hardware components required for onboarding

  FDO: As per FDO 1.1 standards the IoT Edge device must have a Restricted Operating Environment(ROE), This can be a TPM or TEE or Intel EPID. The allowed list of ROE can be found here 'https://fidoalliance.org/specs/fido-security-requirements/fido-authenticator-allowed-restricted-operating-environments-list-v1.3-fd-20211102.pdf'

  Keylime: Keylime Solely depends upon TPM for its operation.

OPCUA Onboarding: The OPCUA needs to have SecureElements. SecureElements are hardware-based storage for cryptographic secrets that protect them against authorized access and disclosure. The mechanisms defined for Device authentication depend on PrivateKeys that are stored in SecureElements. Private keys stored on Devices without SecureElements can be stolen and reused on counterfeit Devices. OwnerOperators may provision Devices without SecureElements if they have other ways to ensure their authenticity.

- **Criteria 3:** Computing Requirements: What is the minimum computing architecture it supports?

  FDO: The FDO 1.1 specifications are drafted to support the FDO on Microprocessors as well as MCU. However, as of now, the implementation is developed only for 64-bit microprocessors.

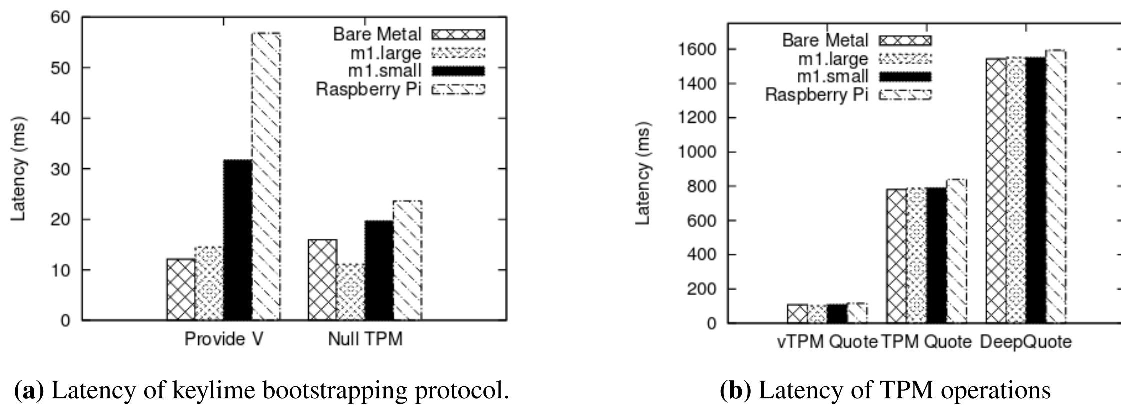
  Keylime: Keylime depends on TPM 2.0 library and Its main objective is to bootstrap and maintain trust in the cloud so it is specifically built for microprocessors.

  OPCUA Onboarding: In OPCUA Every Device has multiple layers of hardware and software that are installed and managed at different stages in the lifecycle by different actors. So OPCUA onboarding does support microprocessor-based IoT devices.

- **Criteria 4:** Time/latency: Time taken for each device to complete the onboarding.

  FDO: Intel claims that the FDO takes less than a minute to onboard a device. For our implementation, the latency increases as the number of devices trying to connect to the server simultaneously increases—the min. Latency is 13.167 seconds, and max. is 24.936 seconds. This data is for simulated devices in a container. The actual time on our prototype is less than a minute. We will discuss this in detail in the Testing and Validation section.

  Keylime: As per the evaluation carried out in this [**3**] research paper, The latency can be visualized from the below charts.



(a) Latency of keylime bootstrapping protocol.

(b) Latency of TPM operations

Figure 4.2: Keylime Latency [**3**]

So the avg. Latency for TPM operations can be seen in the below table. During our implementation with virtual TPM, we also got similar results. Keylime also has latency in an acceptable range.

Table 4.2: Average TPM Operation Latency (ms)[3]

	TPM	vTPM	Deep Quote
Create Quote	725	68.5	1390
Check Quote	4.64	4.64	5.33

OPCUA Onboarding: We do not have any data to compare the latency of OPCUA onboarding.

- **Criteria 5:** Efforts: Efforts required to onboard a device

  FDO: FDO is a complete Zero Touch solution. The installation technician has to plug the Device into the internet and power it on. FDO will take complete responsibility to onboard the Device to a specific IoT cloud. So FDO needs very minimal effort to onboard.

  Keylime: The commissioning agent has to install the Device. Log into the Device and check its IP address. Then this IP address needs to be provided to Keylime Verifier manually using the Keylime tenant. Compared to FDO, This process increases the efforts required to onboard the Device.

  OPCUA Onboarding: As OPCUA Device Onboarding is drafted to be the Automatic or zero-touch device onboarding, the efforts required to onboard the Device shall be equal to FDO.

- **Criteria 6:** Scalability: How many devices can be onboarded to a server

  Scalability is the ability of a system to handle an increasing amount of work or traffic without compromising its performance or availability.

  FDO: During our testing of FDO, we observed that we could onboard at least 30 simulated devices in less than a minute. Considering this data for physical devices, We can onboard 1800 devices per hour and 43200 devices per day. It means that the FDO is highly scalable.

  The scalability also depends upon the hardware of the owner onboarding server. The primary factor affecting FDO scalability is the number of cores and RAM available, which can be increased to scale up the system. The system is also shown to provide linear speedup by adding more cores and parallelism until the parallelism of the host CPU is exhausted.

  Keylime: Based on the information provided in this research [3], the scalability of the Keylime cloud verifier is truly impressive. It can handle a large number of cloud nodes while maintaining reasonable detection latency and response time. It is possible through parallelization and test nodes with zero latency TPM emulation. With the available resources, the system can quickly scale up to handle hundreds or thousands of virtual machines. Similar to other servers, the scalability also depends upon hardware of the server.

  OPCUA Onboarding: We do not have any data to compare the scalability of OPCUA onboarding.

- **Criteria 7:**Reliability: Can the device be onboarded reliably in case of failure during onboarding.

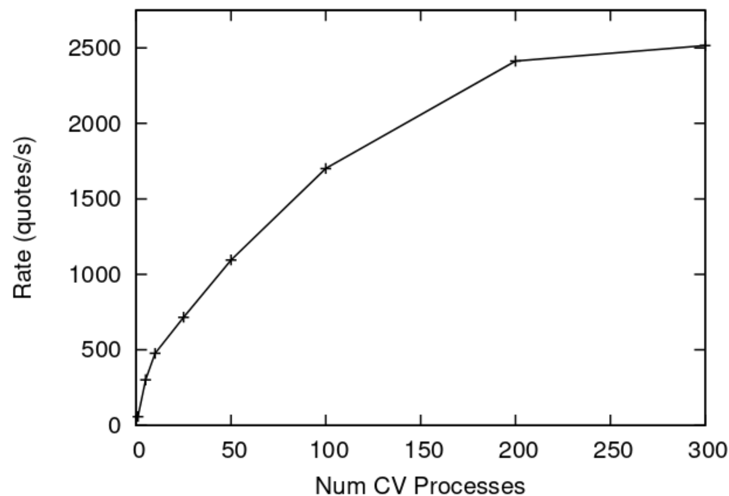


Figure 4.3: Scaling the Keylime Verifier (CV) on bare metal [**3**]

Reliability is the ability of a software system to perform its intended function under specific conditions for a specified period.The reliability depends upon Stability, Error handling, Exception handling and Robustness of the soliton.

FDO: We have tested FDO for reliability. FDO is designed in such a way that we can rely on it to handle any failure, such as power and network failure.

Keylime: As the keylime needs its tenant to enter the command to add the device to the cloud. If the device is not connected to the internet or in case of power failure the tenant again has to enter the command to onboard the device. So when onboarding a device using keylime the technician has to ensure that the device must not have any exception cases such as power or network failure.

OPCUA Onboarding: We do not have any data to compare the reliability of OPCUA onboarding.

- **Criteria 8:** Maintainability: Can we maintain and provide additional service for this method?

  Maintainability is an important aspect of software solutions, which refers to the ease with which a program can be modified, improved, or fixed over time, without causing any new issues or impacting the existing features.

  FDO: FDO is a Linux Foundation Open-source project which follows all standard practices making it maintainable.

  Keylime: Keylime is also Open-Source and maintainable as good as FDO.

  OPCUA Onboarding: OPCUA onboarding has no central code maintaining authority, specifically for onboarding. We have to refer to multiple sources to implement the onboarding. As of now, there is yet to be a complete solution available.

- **Criteria 9:** Ease of manufacturing: Is the method provide ease of manufacturing, How?

  FDO: FDO provides a Late binding feature. The end owner of the IoT device can choose any cloud provider of his choice. The manufacturer can manufacture identical devices without pre-configured data for specific customers. The manufacturer does not have to maintain any SKU numbers because all the produced devices are identical. It eventually saves time in manufacturing, which means it can manufacture more devices, reducing the cost of manufacturing.

  Ease of manufacturing is the most significant advantage of choosing FDO for large-scale IoT deployments.

  Keylime: keylime is initially designed to measure runtime integrity and improve the Device's security. It provides no inputs to guide the manufacturer or how the supply chain will work. So, FDO and OPCUA onboarding is a far better solution in case of ease of manufacturing.

  OPCUA Onboarding: OPCUA onboarding has complete guidelines on how to manufacture the devices and how they will be transferred through the supply chain. However, it does not specify whether it will ease the manufacturing process.

- **Criteria 10:** Ease of distribution/life cycle management: Is the method provide a secure distribution, such as the device can have a chain of trust?

  FDO: FDO is designed to provide secure supply chain management. The threat model of FDO is so mature that it does not allow any of the supply chain members to get access to the credentials of the Device.

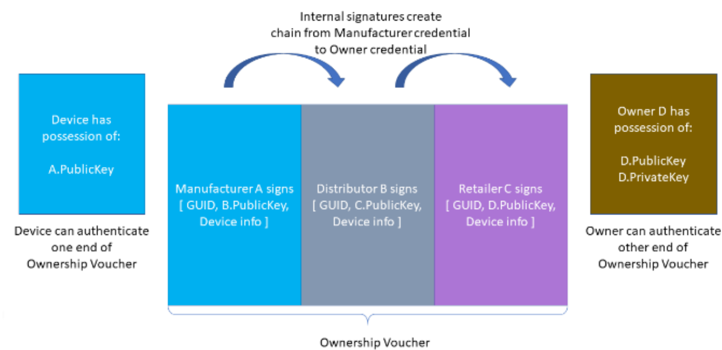


Figure 4.4: FDO Ownership Voucher [**2**]

The Ownership Voucher is a structured digital document that links the Manufacturer with the Owner. It is formed as a chain of signed public keys, each signature of a public key authorizing the possessor of the corresponding private key to take ownership of the Device or pass ownership through another link in the chain.

The above diagram 4.4 illustrates an Ownership Voucher with 3 entries. In the first entry, Manufacturer A, signs the public key of Distributor B. In the second entry, Distributor B signs the public key of Retailer C. In the third entry, Retailer C signs the public key of Owner D. The entries also contain a description of the GUID or GUIDs to which they apply, and a description of the make and model of the device.

The signatures in the Ownership Voucher create a chain of trust from the manufacturer to the Owner. The Device is preprovisioned (e.g., in the Device Initialize Protocol (DI)) with a crypto-hash of A.PublicKey, which it can verify against A.PublicKey in the Ownership Voucher header transmitted in the TO2 protocol. The owner can prove his connection with the Ownership Voucher (and thus his right to take ownership of the Device) by proving its ownership of D.PrivateKey. It can do this by signing a nonce (sometimes called a challenge), and the signature may be verified using D.PublicKey from the Ownership Voucher.

The last entry in the Ownership Voucher belongs to the current owner. The public key signed in that entry is the owner's public key, signed by the previous owner. We call this public key the "Owner Key."

In the TO2 Protocol, the Owner proves his ownership to the device using a signature (as above) and an Ownership Voucher that is rooted in A.PublicKey. The Device verifies that the hash of A.PublicKey stored in its ROE matches A.PublicKey in the Ownership Voucher, then verifies the signatures of the Ownership Voucher in sequence, until it comes to D.PublicKey. The Owner provides the Device separate proof of D.PublicKey (the "owner key"), completing the chain of trust. The only private key needed to verify the Owner's assertion of ownership is the key of the Owner itself. The public keys in the Ownership Voucher (and the public key hash in the Device) are sufficient to verify the chain of signatures.

The public keys in the Ownership Voucher are just public keys. They do not include other ownership info, such as the name of the entity that owns the public key, what other keys they might own, where they are, etc. The Ownership Voucher is maintained only for the purposes of connecting a particular device with its particular first owner. The entities involved can switch the key pairs they use to sign the Ownership Voucher from time to time, make it more difficult for potential attackers to use the Ownership Voucher as a means to map out the flow of devices from factory to implementation. Conversely, it is conceivable that a private data structure might contain supply chain identities, allowing the Ownership Voucher to specifically map the identities who signed it.[**2**]

Keylime: Keylime has no life cycle management guidelines or discusses supply chain management.

OPCUA Onboarding: OPCUA onboarding has complete guidelines on how to manufacture the devices and how they will be transferred through the supply chain. However, it does not specify whether it will ease the distribution process.

- **Criteria 11:** Expertise required :What level of expertise the technician needs to have to on-boarding the device

  FDO: FDO is completely zero touch, so no expertise are required to commission or onboard the devices.

  Keylime: The commissioning technician should have a basic understanding of using computer , such that he can enter the information required to onboard the device.

  OPCUA Onboarding: OPCUA onboarding shall be automatic, so the technician does not need any specific skills to onboard the devices.

- **Criteria 12:** Development efforts : How much efforts a developer needs to spend to implement provided method

  FDO: The Intel has developed a client-SDK and Protocol Reference Implementation for FDO. This code is production ready. The developers can use it as a baseline to start developing their applications. So the development efforts are moderate.

  Keylime: Similarly, keylime has a well-established reference implementation, which developers can use.

  OPCUA Onboarding: There is no reference implementation provided for OPCUA onboarding. The development efforts are high if one chooses to use OPCUA onboarding.

- **Criteria 13:** Programming skills: Which programming language and skills needed as developer

  FDO: The reference implementation from Intel uses c and JAVA as their programming languages. Red Hat has also come up with its reference implementation in RUST.

  Keylime: Keylime uses Python and RUST as their programming languages.

  OPCUA Onboarding: Since OPCUA onboarding does not have any reference implementation. If one wants to refer open62541 stack then you must be skilled in c programming language.

- **Criteria 14:** User Friendliness: does the end user finds this mechanism useful and easy.

  FDO: The FDO is user-friendly. The end owner of the Device can decide which cloud provider he wants to choose for onboarding the Device. It also provides a feature to securely decommission the Device and resale it to the next owner without sharing any credentials used by him. FDO is completely zero-touch making it more user-friendly for smart home devices.

  Keylime: keylime is not zero-touch or discusses anything about the Device's resale, making it less user-friendly.

  OPCUA Onboarding: OPCUA onboarding has similar features to FDO. Hence it is also user-friendly.

- **Criteria 15:** Overall cost : Overall cost including hardware+software+on-boarding

  FDO: FDO needs to have a ROE on the Device, similar to TPM. So the device cost is similar for all three methods. It requires a rendezvous server, owner server, and manufacturer server. The software development and maintenance cost will be less than OPCUA and keylime.

  FDO saves much time during onboarding, reducing the cost of person-hours required to onboard the Device.

  Keylime: Keylime has a Device cost similar to FDO and OPCUA. Since it takes more time to onboard the Device using Keylime, the onboarding cost is more. Keylime continuously monitors the integrity of its nodes, so it requires more computing power on the server side, which adds to the cost.

  OPCUA Onboarding: In OPCUA onboarding, the device cost remains the same. Since there is no implementation, there will be an additional cost required to develop the complete onboarding solution and maintain it.

- **Criteria 16:** Security: Does this method provide proper security guidelines for each stakeholder?

  FDO: Yes, The FDO has a very well thought threat model and security guidelines, which shows the solution has been considered to provide security by design. The details can be found in the FDO documentation here [**27**].

  Keylime: There are no security guidelines, or best practices provided for the implementation of keylime.

  OPCUA Onboarding: OPC foundation has very detailed documentation for security guidelines. The details can be found in OPC 10000-2: UA Part 2: Security.

- **Criteria 17:** Runtime integrity : Can this method provide runtime integrity(trust) measurement

  FDO: No, FDO is solely responsible for onboarding the Device. If required, the user is free to add a DMS that takes care of the Runtime security of the Device.

  Keylime: Keylime has a Cloud Verifier, which constantly checks the integrity of the deployed software and reacts if there is a breach in security.

  OPCUA Onboarding: No. OPCUA does not have anything for runtime integrity measurement.

- **Criteria 18:**Software update : Does this method takes care for automatic software update.

  FDO: No, FDO is solely responsible for onboarding the Device. If required, the user is free to add a DMS that takes care of the Software Update at the time of onboarding the Device.

  Keylime: No, Keylime does not consider to do software update at the time of provisioning.

  OPCUA Onboarding: Yes, OPCUA Onboarding has a Software manager server which will take care for doing software updates at the time of Onboarding.

- **Criteria 19:** Certificate management : Does this method takes care for certificate management

  FDO: No, FDO is solely responsible for onboarding the Device. If required, the user is free to add a DMS that takes care of the Certificate Management of the Device.

  Keylime: yes, keylime has a mechanism to do certificate management.

  OPCUA Onboarding: OPCUA uses a Certificate Manager that monitors a certificate's validity on the client or server.

- **Criteria 20:** Device Attestation : How does this method takes care for device attestation

  FDO: FDO uses Entity Attestation Tokens to do Device Attestation,

  Keylime: Keylime uses Three party key derivation protocol to identify the device.

  OPCUA Onboarding: OPCUA onboarding method has defined a Ticket. Which will be used for device attestation.

- **Criteria 21:** Proof of ownership : How does this method takes care for proof of ownership

  FDO: FDO uses an Ownership voucher as proof of Ownership.

  Keylime: Keylime uses Three party key derivation protocol to prove the ownership.

  OPCUA Onboarding: OPCUA Onboarding uses DeviceIdentity Tickets as proof of Ownership.

- **Criteria 22:** Correlation Attack Concerns: Are the device onboarding credentials used for application provisioning?

  FDO: No, All keys exposed by protocol entities in FIDO Device Onboard can be limited to be used only in FIDO Device Onboard. The Transfer Ownership Protocol 2 (TO2) allows the onboarding of additional device credentials so that the äpplication keysüsed during device operation are distinct from the keys used in FIDO Device Onboard.

  Keylime: We use a hardware TPM, which is not only limited to Device onboarding but also to device management.

  OPCUA Onboarding: The OPCUA onboarding has two distinguished certificates for device onboarding and application provisioning. The onboarding process uses DeviceIdentity Certificates, once the device has been authenticated the registrar issues a DCA Application Instance Certificate to the Device that indicates that it has been authenticated. This Application Instance Certificate is used for further operations.

Table 4.3: Evaluation of Onboarding Methods

Serial Number	Evaluation Criteria	FDO	Keylime	OPCUA Onboarding
1	Zero Touch	Yes	No	Depends
2	Hardware Requirement	ROE	TPM	SecureElement
3	Computing Requirements	Supports MCU and MPU	Only for MPU	Only for MPU
4	Time/Latency	< 1 minute	< 1 minute	No data
5	Efforts	Easy	Moderate	Depends
6	Scalability	Highly Scalable	Moderately Scalable	No data
7	Reliability	Reliably handles failures during onboarding.	Needs to repeat the process	No data
8	Maintainability	Yes,Maintained by open-source community	Yes,Maintained by open-source community	No central code maintaining authority.

Table 4.3 Evaluation of Onboarding Methods (continued)

Serial Number	Evaluation Criteria	FDO	Keylime	OPCUA Onboarding
9	Ease of Manufacturing	Late binding feature for device provisioning, reduces manufacturing cost.	No guidelines for manufacturer	Complete guidelines for manufacturing, no mention of ease.
10	Ease of Distribution/Life Cycle Management	Provides secure supply chain management.	No life cycle management guidelines.	Complete guidelines for supply chain, no mention of ease.
11	Expertise Required	Zero-touch, minimal skills are required	Needs skilled technician	Depends if the onboarding is zero touch or not
12	Development Efforts	Moderate	Moderate	Very High
13	Programming Skills	C, JAVA, RUST	Python, RUST	C
14	User Friendliness	User-friendly with secure decommissioning and resale.	Not zero-touch, no mention of resale.	User-friendly with similar features to FDO.
15	Overall Cost	Time-saving during onboarding, similar software development cost, additional cost for RV server	Onboarding cost is more, increased server computing cost.	Additional development and maintenance cost required, needs additional servers
16	Security	Well-thought threat model and security guidelines.	Developed considering security in mind, however, no security guidelines provided.	Detailed security guidelines are available.
17	Runtime Integrity	No runtime integrity measurement.	Cloud Verifier for integrity checks.	No runtime integrity measurement.
18	Software Update	No automatic software update at provisioning.	No software update at provisioning time.	Software manager server for software updates.
19	Certificate Management	No built-in certificate management.	Provides certificate management mechanism.	Uses Certificate Manager for monitoring certificate validity.

Table 4.3 Evaluation of Onboarding Methods (continued)

Serial Number	Evaluation Criteria	FDO	Keylime	OPCUA Onboarding
20	Device Attestation	Uses Entity Attestation Tokens for device attestation.	Three-party key derivation protocol for device identification.	Defined Ticket for device attestation.
21	Proof of Ownership	Uses Ownership voucher as proof of ownership.	Three-party key derivation protocol to prove ownership.	Uses DeviceIdentity Tickets as proof of ownership.
22	Correlation Attack Concerns	All keys used in FDO are distinct from application keys.	Limited to device onboarding and management.	Uses separate certificate for DeviceIdentity and ApplicationIdentity

4.3 Advantages and Disadvantages

4.3.1 FIDO Device Onboard

- **Advantages:**
  1. FDO is a complete zero-touch solution, making it easy and convenient to use.
  2. FDO specifications support both microprocessors and MCU, although the implementation is only available for 64-bit microprocessors.
  3. FDO is highly scalable, allowing up to 1800 devices to be onboarded per hour, with linear speedup by adding more cores and parallelism until the parallelism of the host CPU is exhausted.
  4. Provides reliable onboarding even in case of power and network failure.
  5. Follows standard practices and is an open-source project, making it easy to maintain.
  6. Provides a late binding feature, which makes it easy to manufacture identical devices without pre-configured data for specific customers, reducing the cost of manufacturing.
  7. Provides secure supply chain management, with a mature threat model and an ownership voucher that creates a chain of trust from the manufacturer to the owner.
  8. Moderate development efforts due to a well-established client-SDK and protocol reference implementation provided by Intel.
  9. User-friendly with the ability for the end owner to choose the cloud provider and securely decommission/resale the device.
  10. Saves time during onboarding, reducing the cost of person-hours required to onboard the device.

11. FDO provides well-defined security guidelines and a threat model for the secure onboarding of devices.
12. FDO uses Entity Attestation Tokens for device attestation to verify the authenticity of devices.
13. FDO allows the user to add a DMS to take care of runtime security, software updates, and certificate management of devices.

- **Disadvantages:**
  1. FDO requires a Restricted Operating Environment (ROE) in the IoT Edge device, which can be a TPM or TEE or Intel EPID, limiting the compatibility of the devices.
  2. FDO Latency increases with simultaneous device connections
  3. Similar device cost as OPCUA and Keylime, as it requires a ROE on the device and additional servers (rendezvous, owner, manufacturer) to implement the protocol.
  4. FDO does not provide any runtime integrity measurement for devices during operation.
  5. FDO does not have any built-in mechanism for software updates or certificate management.

4.3.2 Keylime

- **Advantages:**
  1. Keylime is built specifically for cloud nodes, making it optimized for such devices.
  2. Keylime's cloud verifier can handle a large number of cloud nodes while maintaining reasonable detection latency and response time.
  3. Keylime can perform parallelization and test nodes with zero latency TPM emulation, making it scalable to handle hundreds or thousands of virtual machines.
  4. Open-source and maintainable, similar to FDO.
  5. Designed to measure runtime integrity and improve device security.
  6. Well-established reference implementation, which developers can use.
  7. Keylime uses a Three-Party Key Derivation Protocol for device attestation and ownership verification.
  8. Keylime provides a Cloud Verifier for continuous runtime security monitoring of deployed software.

- **Disadvantages:**
  1. Keylime is not a complete zero-touch solution. Keylime requires the commissioning agent to manually provide the IP address of the Device to Keylime Verifier, which may increase the efforts required to onboard the Device.
  2. Keylime depends solely on TPM for its operation, which may limit its compatibility with devices that do not have TPM.
  3. Does not provide inputs to guide the manufacturer or supply chain, making it difficult to use for ease of manufacturing.
  4. Higher onboarding cost due to it taking more time to onboard the device compared to FDO and OPCUA.
  5. Not zero-touch and does not discuss anything about device resale, making it less user-friendly.
  6. Requires more computing power on the server side due to continuous monitoring of node integrity, adding to the cost.
  7. Keylime does not consider automatic software updates during provisioning.
  8. There are no security guidelines or best practices provided for the implementation of Keylime.

4.3.3 OPCUA Onboarding

- **Advantages:**
  1. OPCUA onboarding is drafted to be automatic or zero-touch, making it easy to use.
  2. OPCUA onboarding supports microprocessor-based IoT devices.
  3. Provides complete guidelines on how to manufacture the devices and transfer them through the supply chain.
  4. Provides a secure distribution method with a chain of trust.
  5. Similar features to FDO, making it user-friendly.
  6. OPCUA Onboarding has a well-defined Certificate Manager for certificate management of devices.
  7. OPCUA Onboarding also takes care of software updates at the time of onboarding using a software manager.
  8. OPCUA Onboarding uses DeviceIdentity Tickets for proof of ownership.

- **Disadvantages:**
  1. OPCUA onboarding requires SecureElements, making it incompatible with devices that do not have SecureElements.
  2. There is no central code maintaining authority specifically for onboarding, which can make it difficult to maintain.
  3. No information on whether it eases the manufacturing process.
  4. High development efforts as there is no reference implementation provided, and one must be skilled in c programming language if referring to the open62541 stack.
  5. Additional cost required to develop the complete onboarding solution and maintain it, as there is no implementation provided.
  6. Similar device cost to FDO and Keylime, requiring a ROE on the device and additional servers.
  7. OPCUA Onboarding does not provide any runtime integrity measurement for devices during operation.

Method	Advantages	Disadvantages
FDO	- Complete zero-touch solution - Supports both microprocessors and MCU - Highly scalable - Reliable onboarding - Follows standard practices and is open-source - Late binding feature for cost reduction - Secure supply chain management - Moderate development efforts - User-friendly - Saves time during onboarding - Well-defined security guidelines and threat model - Uses Entity Attestation Tokens for device attestation - Allows the use of DMS for runtime security, software updates, and certificate management	- Requires a Restricted Operating Environment - Latency increases with simultaneous device connections - Similar device cost to other methods - No runtime integrity measurement - No built-in mechanism for software updates or certificate management
Keylime	- Optimized for cloud nodes - Can handle a large number of cloud nodes - Scalable through parallelization - Open-source and maintainable - Measures runtime integrity - Well-established reference implementation - Uses Three-Party Key Derivation Protocol for device attestation and ownership verification - Provides Cloud Verifier for continuous runtime security monitoring	- Depends solely on TPM - Commissioning agent needs to manually provide device IP - No inputs to guide manufacturer or supply chain - Higher onboarding cost compared to FDO and OPCUA - Not zero-touch and no device resale support - Requires more computing power on server side - No automatic software updates during provisioning - No security guidelines or best practices provided
OPCUA Onboarding	- Manual or zero-touch - Supports microprocessor-based IoT devices - Provides guidelines for device manufacturing and secure distribution, decommissioning - User-friendly - Well-defined Certificate Manager for certificate management - Uses DeviceIdentity Tickets for proof of ownership - No correlation attack concerns	- No data on latency and reliability - Requires SecureElements - No central code maintaining authority - No information on manufacturing ease - High development efforts - Additional cost required for complete onboarding solution and maintenance - No runtime integrity measurement

Table 4.4: Advantages and disadvantages IoT device onboarding methods

5 Software Architecture Design for zero-touch onboarding to hawkBit using FDO standards

In this section of the report, we will explore the development of a zero-touch onboarding solution using FDO 1.1 standards for the efficient onboarding of a large number of IoT devices to a hawkBit Device Management Server. The process of onboarding a large number of devices manually can be time-consuming and error-prone, which is why automating this process is crucial for managing IoT devices effectively. We will discuss the key aspects of manual onboarding to the Hawkbit server. We will analyze each step and then come up with our software architecture which will help us to achieve automation of the onboarding process. Chapter 5,6,7 answers our Research Question 3, by providing a software solution to facilitate zero-touch onboarding to the hawkBit server.

5.1 Manual Onboarding to hawkBit Server

The manual onboarding of the IoT edge device to the hawkBit server starts with the commissioning technician physically installing the device. For our implementation, we will assume that the device is an embedded Linux platform with an SW update library configured.

He will also need to carry a list of all the devices he is going to onboard, and where the list contain a serial number of the device, a controller id, make and model, and other such details. The technician will create a target on the hawkBit server using this list. Sometimes, the owner or the organization will have a tool, or software application that is responsible for commissioning the device. This application will automate a few steps in manual onboarding such as having a list of devices and generating security tokens automatically. However in the end the commissioning technician needs to log in to the device and configure the device using this security token, so still it is time-consuming. The process of creating a target using the UI of the hawkBit server is described below.

Step 1: He will log in to the Hawkbit server using the credentials provided to him, as shown in Figure 5.1.

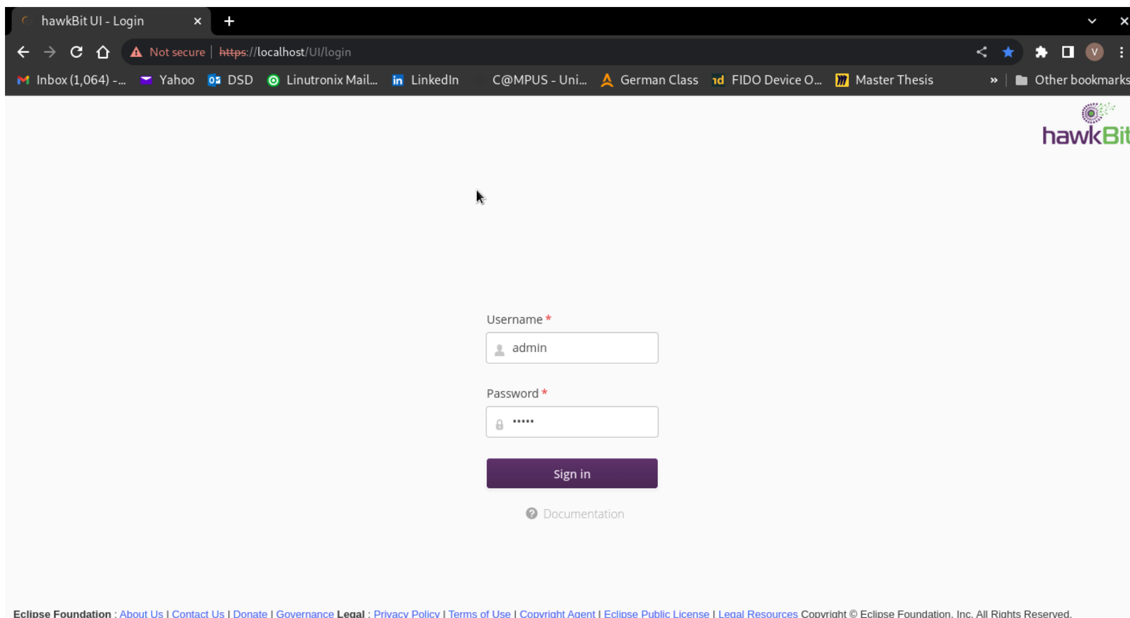


Figure 5.1: Step 1: Log in to hawkBit Server

Step 2: Once logged in, he will then create a device using hawkBit UI. He must enter the Controller ID, Name, and device description here, As shown in the below Figure 5.2.

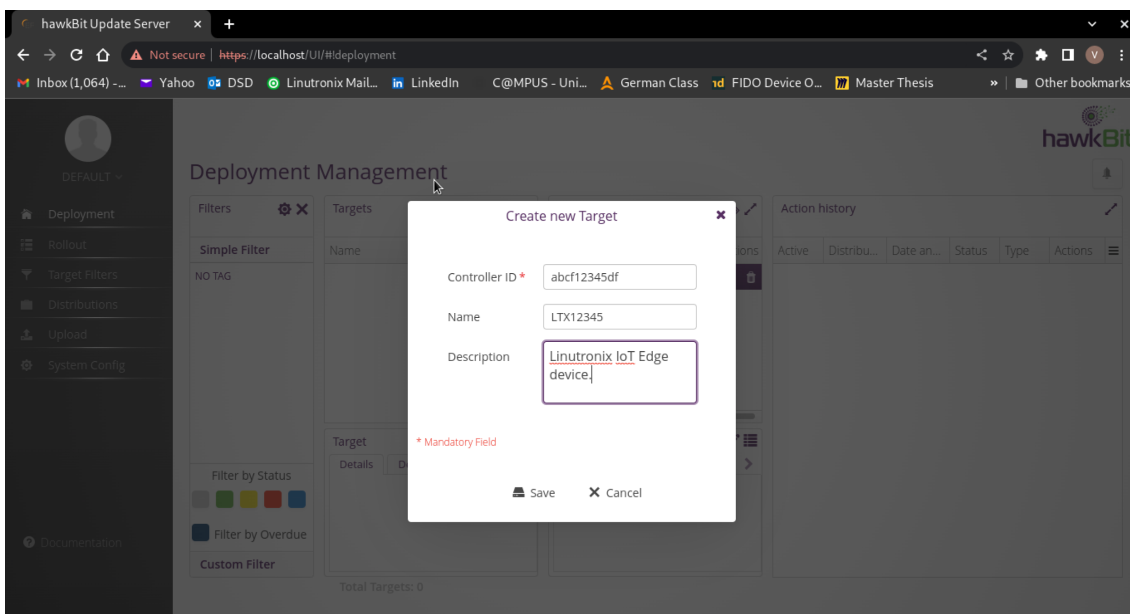


Figure 5.2: Step 2: Create a target(device) on hawkBit server

Step 3: The target or device is now created on the hawkBit server, and a Security token is created for that device. It can be seen below Figure5.3.

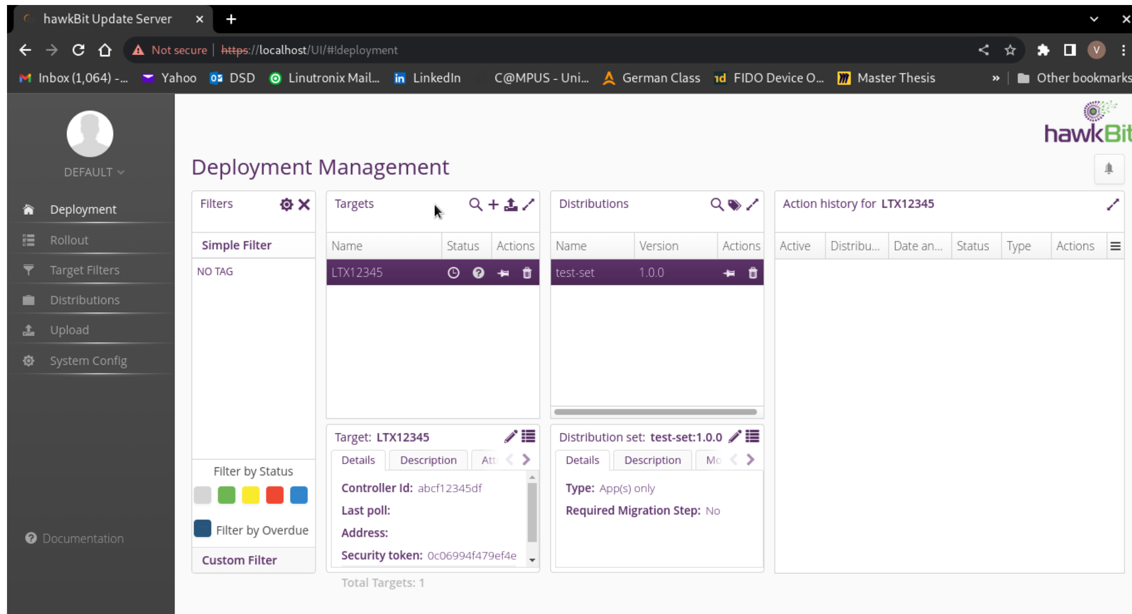


Figure 5.3: Step 3: Security Token generated

Step 4: The technician will now have to configure the device using this Security token. To do so, he will log into the IoT Edge device using the login credentials provided in the list. It is also a security concern because most devices are shipped with default IDs and passwords. Once logged in, he will run a shell command to activate the suricatta daemon of swupdate. This daemon is responsible for working as a hawkBit client. It will use the security token to authenticate the device to the hawkBit server. The authentication will happen as shown in the below Figure 5.4.

```
swupdate -u '-t DEFAULT -x -u https://localhost -i abcf12345df -k 0
c06994f479ef4e5e3b8fb6e670591a0'

-u, --suricatta [OPTIONS]      : Parameters to be passed to suricatta
    -t, --tenant      * Set hawkBit tenant ID for this device.
    -u, --url         * Host and port of the hawkBit instance, e.g., localhost:8080
    -i, --id          * The device ID to communicate to hawkBit.
    -k, --targettoken   Set target token.
```

Listing 5.1: swupdate suricatta daemon

Figure 5.4: Step 4: Configure and register the device

Step 5: Once the device is registered with hawkBit, the hawkBit UI will update the status of the device. We can see the updated status in the below Figure. This complete procedure takes approximately 5-10 mins and is time-consuming. On average, a skilled technician can only onboard or deploy a maximum of 15-20 devices per hour. Along with that, he needs to enter all the details manually, which can lead to errors in configuration. Automating this onboarding process is crucial to provide seamless large-scale IoT deployment if we would like to achieve massive growth in IoT technology.

Figure 5.5: Step 5: Confirm device status as registered

5.2 Architecture Design

We are using FDO 1.1 standards to design our software architecture. As seen in the previous section, we need to automate all five steps of manual onboarding. We will divide this design into two parts. The first part will be a combination of steps 1,2,4,5. We will call it cloud-side architecture design. Furthermore, the second part will be step 3, called device-side architecture design. We did this segregation because the first part is carried out independently of the device being onboarded, and the second part is executed on the device to be onboarded.

5.2.1 FDO Device architecture

Before going ahead, we will get an idea of the FDO Device. It has developed a unique feature of late binding using its state-of-the-art ServiceInfo module. In simple terms, ServiceInfo is a type that contains key-value pairs used to communicate between the Management Service on the cloud side, and Management Agent functions on the device side. Each pair represents a message between two modules, one on the owner's side and the other on the device's side, identified by a module name and a message name separated by a colon.

As discussed in section 2.3, Once a device is onboarded to the owner server, a secure channel is established between the device and the Owner Server. It ensures that the device and the Owner Server authenticate each other during the TO2 protocol. After this step, the Owner Server can query the device's information, which is Device ServiceInfo. In addition, the Owner Server can also send down configuration information to the device, referred to as Owner ServiceInfo. This secure channel uses encryption and integrity protection to ensure that data in transit from/to the device is secure. It can be visualized in the below image.

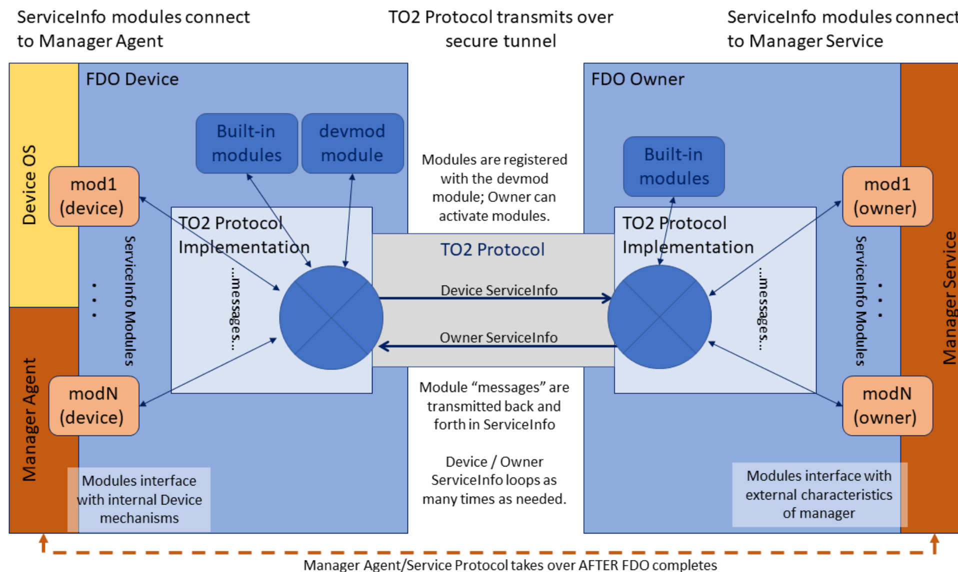


Figure 5.6: Management Service - Agent Interactions via ServiceInfo

Messages sent to a module on the FDO Device may interact with the Device OS to install software components. Another message might use those components, in combination with a cryptographic key, to establish communications. In some systems, the Management Agent might be installed by cooperating modules before it is active by others, allowing an 'off-the-shelf' device to be customized by FDO. The intention is that modules will implement common or standardized IOT provisioning functions and will be reused for different IOT solutions provisioned by FDO. In some cases, modules on the FDO Owner and FDO Device will be designed to cooperate directly. For example, a module that implements a particular device management client on the FDO Device and its counterpart that feeds it exactly the right credentials on the FDO Owner. In other cases, modules may implement IOT or OS primitives so that the FDO Owner or FDO Device picks and chooses among them. For example, allocating a key pair on the FDO Device; signing a certificate on the FDO Owner; transferring a file into the OS; upgrading software; and so on.

We have used client SDK developed by Intel. The block diagram of SDK is given below.

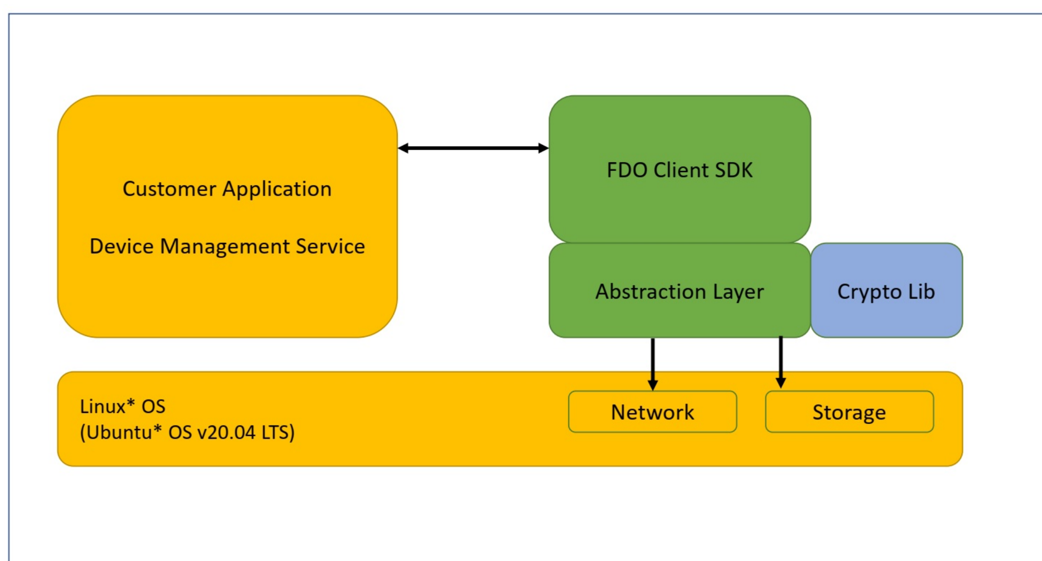


Figure 5.7: Intel FDO Client Block Diagram [**27**]

The integrated image and execution flow from the system boot are shown above and each step is described below[**27**]:

1. On reset, the Board Support Package (BSP) and RTOS initialize the hardware and pass the control to the Application. On Linux* systems, this is the normal OS boot flow and is complete by the time the Application executes.

2. The Application initializes all FDO modules if required. The Application also initializes the SDK by calling the fdo_sdk_init() API and registers each module with the SDK by passing the module's name and callback address to the SDK (in the fdo_sdk_service_info_module structure).

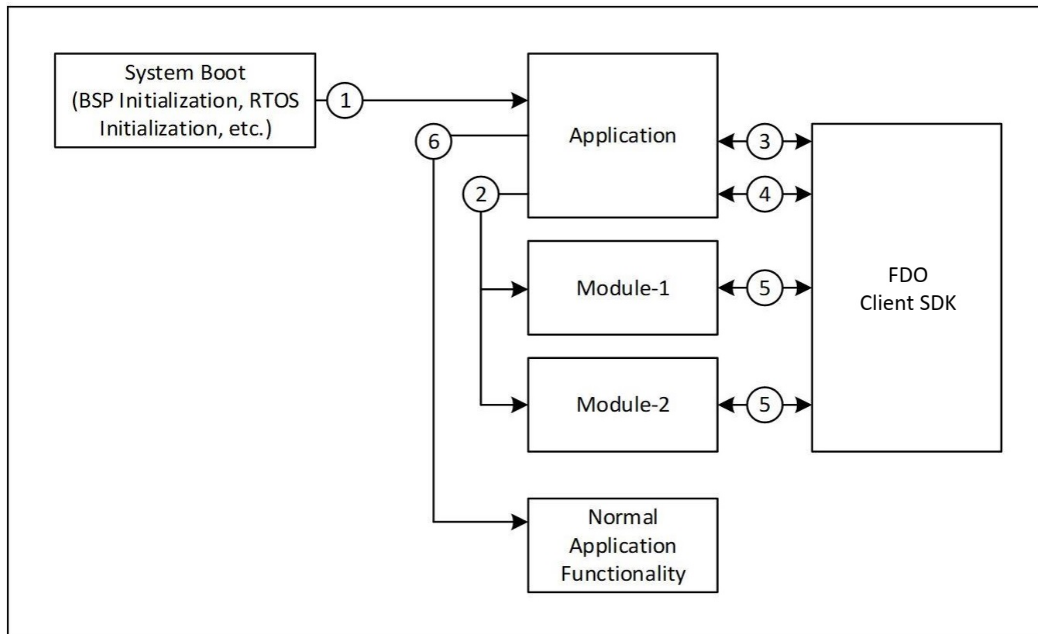


Figure 5.8: Intel FDO Client Execution Flow [27]

3. The Application checks if FDO onboarding has been completed by calling the fdo_sdk_get_status() API. If the status FDO_SDK_STATE_IDLE is returned, onboarding has been completed and the Application goes to step 6. If not, the Application goes to step 4.
4. The Application initiates FDO onboarding by calling the fdo_sdk_run() API. This call returns either a successful completion of onboarding or an error. If an error occurs, the Application will reset the device and retry the sequence, with some delay. On successful completion of onboarding, the Application goes to step 6.
5. During the onboarding process, the SDK will call registered modules during the ServiceInfo stage of the protocol. This is done by calling the registered module callback. The onboarding process will succeed only if all module interactions at this stage are successful.
6. The Application has successfully completed onboarding and continues the normal operation of the device.

The Application continues operating until the system is powered off or reset. On System restart, the preceding steps are re-executed.

Based upon the above architecture we have created our own module for hawkBit onboarding. This module is responsible for sending the device GUID, and Serial Number to the Owner server. It is also responsible for receiving Controller ID, URL to the hawkBit server, and Security Token. After receiving this data our hawkbitonboarding module will use these parameters to execute the swupdate suricatta daemon and will register the device to the hawkBit server. The final architecture of our device can be seen in the below Figure. This will cover step 4 from the manual device onboarding.

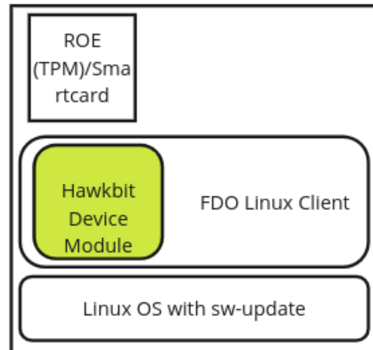


Figure 5.9: FDO Device architecture with hawkBit device module

5.2.2 FDO Owner Server architecture

Similar to the Device module, we must create an owner module on the owner server. This module will be responsible for coordinating the device and device management server. In our case, the DMS is the hawkBit server.

The Owner server will first receive the device service info from the device using the devMod module. It will be the serial number of the device, GUID. The owner module will create a target on the hawkBit server using these details. To create the target, we have used hawkBit server management APIs.Once the target is created, we will query the security token for the newly created target device using the same management API. It automated step 1,2,3 from manual onboarding.

The Owner module now sends the hawkBit client configuration details to the device. It will send a hawkBit configuration file. It will contain the Controller id of the device, URL to the hawkBit server, and security token. The hawkBit Device Module will further read this file and execute step 4.

Now, the owner module needs to verify that the device is registered and, if registered, update the FDO device onboarding status as successful. It again uses management API to check the status of the device. If the device is not registered, then the status will be unsuccessful. Step 4 will be executed again until the status changes to the device are registered and the FDO device onboard is successful. It automates step 5 of manual onboarding. The figure below can visualize the complete architecture.

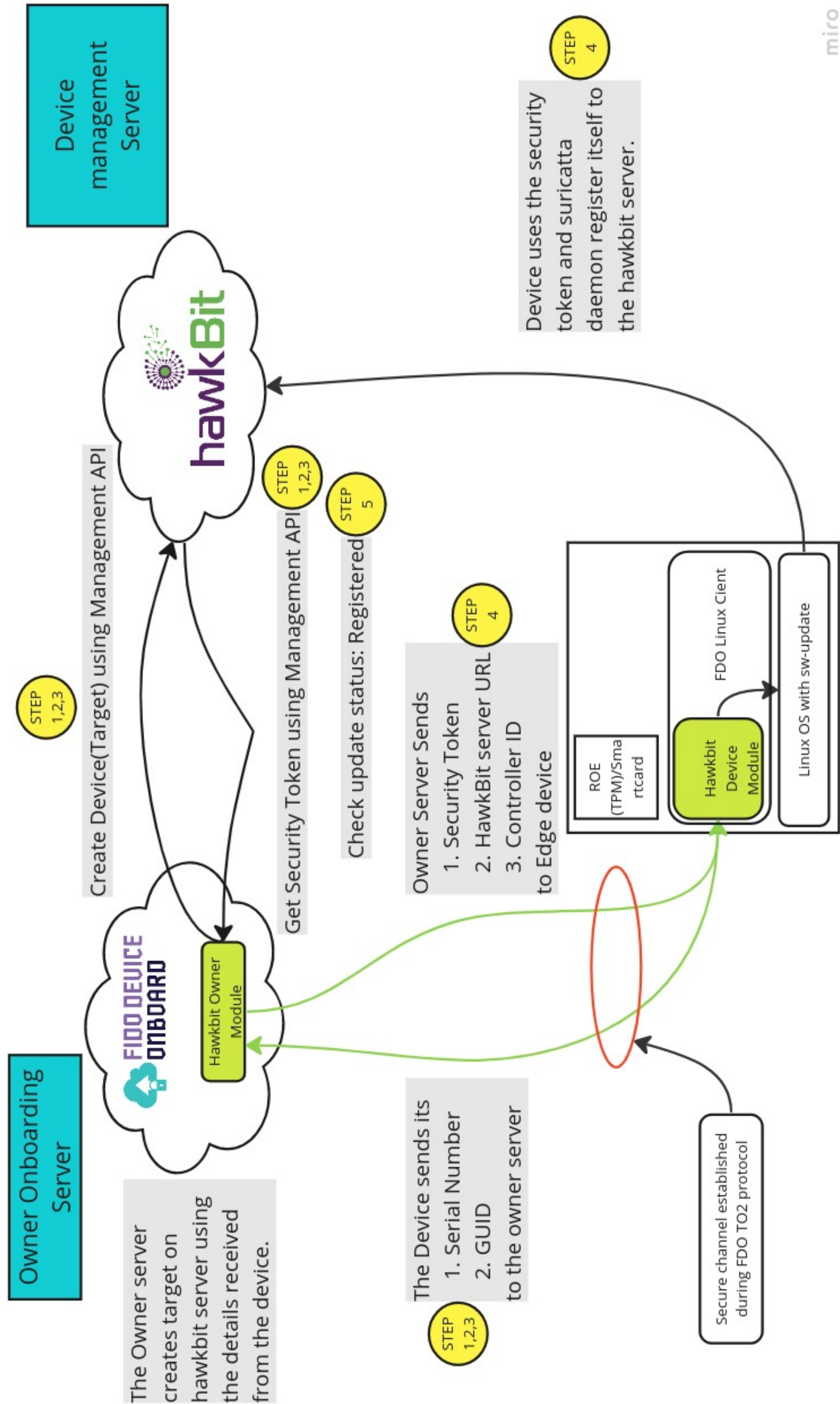


Figure 5.10: Zero touch onboarding to hawkBit using FDO Architecture

6 Software Solution Implementation

6.1 Device manufacturing

We have used C client SDK provided by Intel, and on top of it, created our hawkBit device module.

The one significant advantage of FDO is that it provides complete solutions from manufacturing to decommissioning the device. During Device manufacturing, the manufacturer needs to execute the Device Initialization protocol. It is nothing but the Insertion of FIDO Device Onboard credentials into the device during manufacturing. To implement this process, we have to insert the manufacturer serial number, manufacturer address, and private keys for the device into a specific folder, /opt/fdo/data.Önce they are inserted, we run the Linux client. It will complete the DI protocol and create a DIStatus file if the Device Initialization is successful.

If we run the linux-client for a second time and the DI and TO0 protocols are already completed, the device will start with TO1 and TO2 protocols. Therefore we have created a systemd service that will run whenever the device boots. It checks if the DI is complete by parsing the DIStatus file. If DI is completed, it will immediately execute the linux-client so that the device executes TO1 and TO2 protocol immediately after it gets powered at installation, making our solution zero-touch. The fdolinuxclient.service is given below.

```
[Unit]
Description=Run FDO Linux Client at boot if DI is complete
After=network.target

[Service]
Type=simple
ExecStart=/usr/bin/bash -c 'if [ -f /opt/fdo/DIStatus ]; then cd /opt/fdo && ./linux-
client; fi'

[Install]
WantedBy=multi-user.target
```

Listing 6.1: fdolinuxclient.service

6.2 FDO hawkBit Device Module

The FDO hawkBit Device module is responsible for the following part of device onboarding.

1. Sending the details of the device to the owner server.

2. Receive the configuration file from the owner server.
3. Configure and register the device to the hawkBit DMS.

We have used C client SDK provided by intel, and on top of it, created our hawkBit device module. The implementation of the above steps is given below.

6.2.1 Sending the details of the device to the owner server

As per FDO standards, the DevMod module sends the device information to the owner's server. We created a c function to read the device's serial number. The serial number is given to the device in the manufacturing process. It is stored in a file name "manufacturer_sn.bin."The c function get_device_serial_number is given below.

```
/**
 *  get device serial number
 *
 * @return
 *        returns device serial number as string.
 */
const char *get_device_serial_number(void)
{
  FILE *fp;
    char *line = NULL;
    size_t len = 0;
    ssize_t read;

   fp = fopen("/opt/fdo/data/manufacturer_sn.bin", "r");
    if (fp == NULL)
        exit(EXIT_FAILURE);
    LOG(LOG_ERROR, "manufacturer_sn.bin: No such file or the file is empty \n");

   while ((read = getline(&line, &len, fp)) != -1) {
    LOG(LOG_DEBUG, "Retrieved line of length %zu :\n", read);
    LOG(LOG_DEBUG, "The Serial Number is : %s", line);

    }
  return line;
  free(line);
    exit(EXIT_SUCCESS);
}
```

Listing 6.2: get_device_serial_number function

6.2.2 Receive the configuration file from the owner server

We have used original functions from FDO System Module to create a hawkbit.config file and write the configuration details from the server to the file.

6.2.3 Configure and register the device to the hawkBit DMS.

The configuration and registration of the device to the hawkBit server are taken care of by the the hawkbitOnboarding function(see listing 6.3). Once the hawkBit.config file is written successfully, and the hawkBit onboarding function will be called. This function will parse the configuration file, read the URL, controller ID, and security token and store them as variables. It will then execute the suricatta daemon CLI. Before that, it will also ensure that the configuration details used are valid. It will also create a hawkbit.log file to store the logs.

```
void hawkbitOnboarding()
{
    FILE *configFile;
    char buffer[256];
    char url_copy[256], controllerid_copy[256], securitytoken_copy[256];
    char *url, *controllerid, *securitytoken, *timestamp;

    configFile = fopen("/opt/fdo/hawkbit.config", "r");
    if (configFile == NULL) {
        perror("Error opening file");
        exit(1);
    }

    while (fgets(buffer, sizeof(buffer), configFile)) {
        if (strstr(buffer, "URL:") != NULL) {
            strcpy(url_copy, buffer);
            url = strtok(url_copy, ":");
            url = strtok(NULL, " \n");
        } else if (strstr(buffer, "ControllerId:") != NULL) {
            strcpy(controllerid_copy, buffer);
            controllerid = strtok(controllerid_copy, ":");
            controllerid = strtok(NULL, " \n");
        } else if (strstr(buffer, "SecurityToken:") != NULL) {
            strcpy(securitytoken_copy, buffer);
            securitytoken = strtok(securitytoken_copy, ":");
            securitytoken = strtok(NULL, " \n");
        }
    }

    fclose(configFile);

    printf("URL: %s\n", url);
    printf("ControllerId: %s\n", controllerid);
    printf("SecurityToken: %s\n", securitytoken);


  // check if all values are non-empty and valid
  if (strlen(url) > 0 && strlen(controllerid) > 0 && strlen(securitytoken) > 0 &&
      strspn(securitytoken, "abcdefghijklmnopqrstuvwxyz0123456789") == 32) {
    // get the current timestamp
    time_t current_time = time(NULL);
    struct tm *tm = localtime(&current_time);
    strftime(timestamp, 20, "%Y-%m-%d_%H:%M:%S", tm);
```

```
    // write the log message to the file
    FILE *log_file = fopen("/opt/fdo/hawkbit.log", "a");
    if (log_file == NULL) {
      printf("Error: could not open log file\n");
      exit(1);
    }
    fprintf(log_file, "Hawkbit config changed at %s\n", timestamp);
    fclose(log_file);

  // execute the swupdate command
  char command[2048];
  sprintf(command, "/usr/bin/swupdate -v -u \"-t DEFAULT -x -u %s -i %s -k %s\" >> /opt/
fdo/hawkbit.log 2>&1 &", url, controllerid, securitytoken);
  popen(command, "r");
  }
}
```

Listing 6.3: hawkbitOnboarding function

6.3 FDO hawkBit Owner Module

The FDO hawkBit Owner module is responsible for the following part of device onboarding.

1. Receive the details of the device.
2. Create a hawkBit target on the hawkBit server and Receive the security Token of the device from the hawkBit server.
3. Send the configuration details to the device.
4. Confirm the device is onboarded to the hawkBit server.

6.3.1 Receive the details of the device

The hawkBit owner module receives the device's serial number by reading data received from DevMod.KEY_SN. it maps every serial number with GUID using Hashmaps, such that whenever multiple client requests come to the owner server, it should not mix up the serial number of one GUID with the others.

```
case DevMod.KEY_SN:
guid = state.getGuid().toString();
serialnumber =  Mapper.INSTANCE.readValue(kvPair.getValue(), String.class);
serialNumbers.put(guid, serialnumber);
break;
```

Listing 6.4: Receive serial number from device

6.3.2 Create a hawkBit target on the hawkBit server and get security token

We have created the createHawkbitTarget(ServiceInfoModuleState state, String guid) method to create a hawkBit target on the hawkBit server. This function uses the serial number as the device name and guid as controllerID. It then uses management API to create the device. Once the device is created, it will immediately retrieve the security token for the device.

```
/*create hawkbit target device and get security token */
  protected String createHawkbitTarget(ServiceInfoModuleState state, String guid) throws
 IOException{
    try {
      ProcessBuilder processBuilder = new ProcessBuilder();
      processBuilder.command("bash", "-c", "HISTCONTROL=ignoreboth; " +
              "curl -k -u admin:admin \"https://" + hawkbitserver + "/rest/v1/targets\"
-X POST -H \"Content-Type: application/json;charset=UTF-8\" -d '[{\"controllerId\":\"" +
 guid + "\",\"name\":\"" + serialnumber + "\",\"description\":\"Linutronix FDO_device
\"}]' < /dev/null; " +
              "get_securityToken=$(curl -k -u admin:admin \"https://" + hawkbitserver +
"/rest/v1/targets/\"" + guid + "\"\" -X GET | jq '.securityToken'); " +
              "ST=$(echo \"$get_securityToken\" | tr -d '\"'); " +
              "echo $ST" );
      processBuilder.redirectErrorStream(true);
      Process process = processBuilder.start();

      // read the output from the command
      InputStream inputStream = process.getInputStream();
      BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
      String line;
      while ((line = reader.readLine()) != null) {
          ST = line; // reads the value of the security Token
          System.out.println(line);
      }
  } catch (Exception e) {
      e.printStackTrace();
  }
    return ST; //returns the security token
}
```

Listing 6.5: createHawkbitTarget method

6.3.3 Send the configuration details to the device

The owner module will now send the configuration details to the device. It is divided into two functions. The first one is to create a hawkbit.config file on the device, done by the createHawkbitConfig(FdoSysModuleExtra extra) method. The second one is to write the configuration details to this file, which is carried out by the writeHawkbitConfig(ServiceInfoModuleState state, FdoSysModuleExtra extra, String securityToken, String guid) method.

```
/*create hawkbit.config file */
protected void createHawkbitConfig(FdoSysModuleExtra extra) throws IOException {
  ServiceInfoKeyValuePair kv = new ServiceInfoKeyValuePair();
```

```
  kv.setKeyName(FdoSys.FILEDESC);
  String filename = "hawkbit.config";
  kv.setValue(Mapper.INSTANCE.writeValue(filename));
  extra.getQueue().add(kv);
}

/*write hawkbit.config file */
protected void writeHawkbitConfig(ServiceInfoModuleState state,FdoSysModuleExtra extra,
String securityToken, String guid ) throws IOException {
  guid = state.getGuid().toString();
  String CFG = "URL:https://"+ hawkbitserver +"\n"
  + "ControllerId:"+ guid +" \n"
  + "SecurityToken:"+ securityToken +"\n";

    InputStream targetStream = new ByteArrayInputStream(CFG.getBytes());
      try (InputStream input = targetStream) {
        for (; ; ) {
          byte[] data = new byte[state.getMtu() - 26];
          int br = input.read(data);
          if (br == -1) {
            break;
          }
          ServiceInfoKeyValuePair kv = new ServiceInfoKeyValuePair();
          kv.setKeyName(FdoSys.WRITE);

          if (br < data.length) {
            byte[] temp = data;
            data = new byte[br];
            System.arraycopy(temp, 0, data, 0, br);
          }
          kv.setValue(Mapper.INSTANCE.writeValue(data));
          extra.getQueue().add(kv);
        }
      }
  }
```

Listing 6.6: createHawkbitConfig and writeHawkbitConfig methods

6.3.4 Confirm the device is onboarded to the hawkBit server.

The final step for the owner module is to confirm that the device registration to the hawkBit server is successful. It uses the management APIs to check the hawkBit target status. If the status is registered, it will complete the FDO TO2 protocol. Otherwise, it will just repeat the process until the device gets registered.

```
protected String confirmTargetRegistration(ServiceInfoModuleState state, String guid)
throws IOException{
  try {
    guid = state.getGuid().toString();
    ProcessBuilder processBuilder = new ProcessBuilder();
    processBuilder.command("bash", "-c", "HISTCONTROL=ignoreboth; " +
```

```
            "get_updateStatus=$(curl -k -u admin:admin \"https://" + hawkbitserver + "/
rest/v1/targets/\"" + guid + "\"\" -X GET | jq '.updateStatus'); " +
            "updateStatus=$(echo \"$get_updateStatus\" | tr -d '\"'); " +
            "echo $updateStatus" );
    processBuilder.redirectErrorStream(true);
    Process process = processBuilder.start();

    // read the output from the command
    InputStream inputStream = process.getInputStream();
    BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
    String line;
    while ((line = reader.readLine()) != null) {
        updateStatus = line; // reads the value of updateStatus
        System.out.println(line);
    }

} catch (Exception e) {
    e.printStackTrace();
}
return updateStatus;
}
```

Listing 6.7: confirmTargetRegistration method

7 Validation and Testing of Prototype

7.1 Quick Overview of FDO

The FDO framework comprises three main server-side components and one client-side component. On the server side, we have the Manufacturer, RV (Rendezvous), and Owner Service, while on the client side, we have the device implementation in C (Client-sdk-fidoiot). We have created a prototype IoT device that will be onboarded to the hawkBit Server using FDO 1.1 standards. The hardware for the IoT device is explained in section 2.8. The flow chart given below highlights the complete process of FDO.

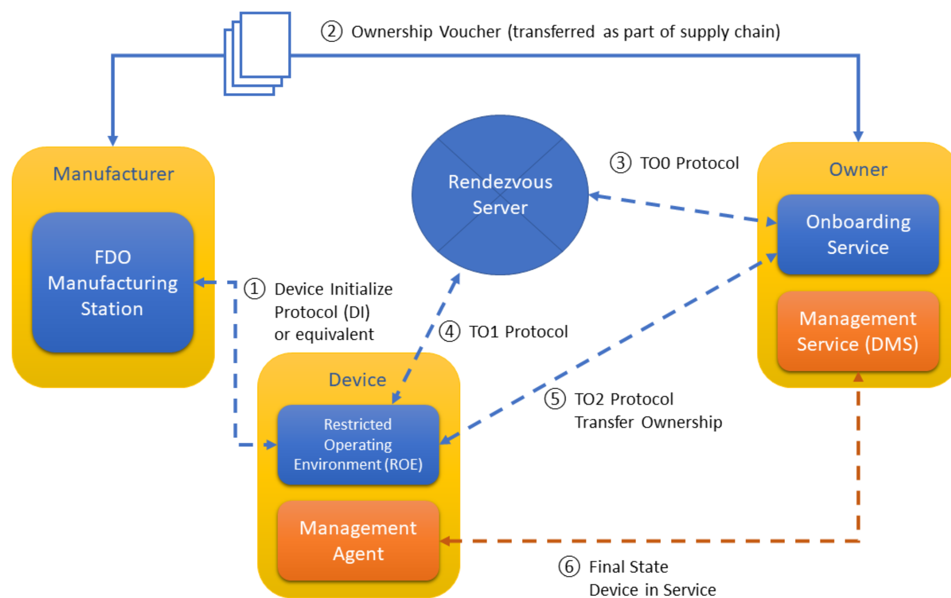


Figure 7.1: FIDO Device Onboard Entities and Entity Interconnection, [**2**]

Four sets of protocols make up FDO: DI, TO0, TO1, and TO2. The DI protocol is used for Device Initialization and involves communication between the Device and the Manufacturer (msg 10-13). The TO0 protocol, or Transfer of Ownership 0, occurs between the Owner and the RV server (msg 20-23), while the T01 protocol, or Transfer of Ownership 1, occurs between the Device and the RV server (msg 30-33). Lastly, the T02 protocol, or Transfer of Ownership 2, occurs between the Device and the Owner Server (msg 60-71). The table below provides a quick overview of each protocol.

Protocol	Parties Involved	Messages	Description
DI (Device Initialization)	Device, Manufacturer	Msg 10-13	Device initializes contact with Manufacturer's service. FDO credentials, including RVInfo, are created and inserted into the device during this process.
TO0 (Transfer of Ownership 0)	Owner, Rendezvous (RV) server	Msg 20-23	Owner initiates contact with RV server using the Ownership voucher. Mapping between GUID and Owner address (DNS/IP) is created and stored in the RV server's database.
T01 (Transfer of Ownership 1)	Device, Rendezvous (RV) server	Msg 30-33	Device contacts RV server using rvInfo collected during DI. Device identifies itself and collects the mapping of Owner address based on its GUID from the RV server.
T02 (Transfer of Ownership 2)	Device, Owner Server	Msg 60-71	Device uses the OwnerAddress collected during T01 to contact the Owner Server. Trust is established, and Ownership Transfer is performed.

Table 7.1: FDO Protocols and Messages

7.2 Server and Device Setup

7.2.1 Server Setup

We have used protocol reference implementation created by Intel. On top of it, we have modified the owner server to support our hawkBit owner module. The protocol reference implementation can be found here: https://github.com/Vishwasrao1/pri-fidoiot. We have built the Java application and run each server in a containerized environment. We have also generated the necessary self-signed certificates and passwords for each server.

First of all, we need to run the MariaDB Database server. This server is responsible for storing all the related data of the FDO protocol.

```
fdo-db_1  | 2023-05-07 18:59:08 0 [Note] InnoDB: Buffer pool(s) load completed at 230507 18:59:08
fdo-db_1  | 2023-05-07 18:59:08 0 [Note] Server socket created on IP: '0.0.0.0'.
fdo-db_1  | 2023-05-07 18:59:08 0 [Note] Server socket created on IP: '::'.
fdo-db_1  | 2023-05-07 18:59:08 0 [Note] mariadbd: ready for connections.
fdo-db_1  | Version: '10.10.2-MariaDB-1:10.10.2+maria~ubu2204-log'  socket: '/tmp/mysql.sock'  port: 3306  mariadb.org binary distribution
```

Figure 7.2: database server

Next, we will start the Device manufacturer server responsible for manufacturing the Device and running the FIDO Device Onboard application during the factory stage to perform Device Initialization.

```
pri-fdo-mfg    | May 07, 2023 7:02:02 PM org.apache.coyote.AbstractProtocol start
pri-fdo-mfg    | INFO: Starting ProtocolHandler ["http-nio-8039"]
pri-fdo-mfg    | May 07, 2023 7:02:02 PM org.apache.coyote.AbstractProtocol start
pri-fdo-mfg    | INFO: Starting ProtocolHandler ["https-jsse-nio-8038"]
pri-fdo-mfg    | 19:02:02.193 [INFO ] Started Manufacturer Service.
```

Figure 7.3: Manufacturer server

After that, we will start Rendezvous Server. It acts as a Network server or service as a rendezvous point between a newly powered-on Device and the Owner.

```
pri-fdo-rv     | May 07, 2023 7:03:20 PM org.apache.coyote.AbstractProtocol start
pri-fdo-rv     | INFO: Starting ProtocolHandler ["http-nio-8040"]
pri-fdo-rv     | May 07, 2023 7:03:20 PM org.apache.coyote.AbstractProtocol start
pri-fdo-rv     | INFO: Starting ProtocolHandler ["https-jsse-nio-8041"]
pri-fdo-rv     | 19:03:20.633 [INFO ] Started Rendezvous Service.
```

Figure 7.4: Rendezvous server

We will now start the Owner server. It is an entity constructed to perform FIDO Device Onboard protocols on behalf of the Owner.

```
pri-fdo-owner     | May 07, 2023 7:05:14 PM org.apache.coyote.AbstractProtocol start
pri-fdo-owner     | INFO: Starting ProtocolHandler ["http-nio-8042"]
pri-fdo-owner     | May 07, 2023 7:05:14 PM org.apache.coyote.AbstractProtocol start
pri-fdo-owner     | INFO: Starting ProtocolHandler ["https-jsse-nio-8043"]
pri-fdo-owner     | 19:05:14.401 [INFO ] Started Owner Service.
```

Figure 7.5: Owner server

Ultimately, we will start Device Management Server, our hawkBit server. We have created our own containerized version of the hawkBit server. It can be found in this repository: https://github.com/Vishwasrao1/hawkbit-server

```
hawkbit_1  |
hawkbit_1  | Eclipse hawkBit Update Server  (v0.3.0M7)
hawkbit_1  | using Spring Boot  (v2.3.7.RELEASE)
hawkbit_1  |
hawkbit_1  | Go to https://www.eclipse.org/hawkbit for more information.
```

Figure 7.6: hawkBit server

7.2.2 Device Setup

We have created our meta-layer to add fdo functionality to the raspberry pi board. We have written a recipe fdoclientsdk.bb to build the FDO client SDK, install the linux-client to the embedded Linux device, i.e., raspberry pi CM4, and make it FDO capable. The same meta layer can be found here https://github.com/Vishwasrao1/lx-fidoiot-rpi. We used usbboot and bmaptools to flash the prototype IoT device.

7.3 FDO Demonstration and Validdation

7.3.1 Manufacturing of Device

As discussed in section 6.1, The manufacturing process involves below mentioned steps:

- We flash the Device and add a private key. We provide the Device with the FDO Manufacturer station address and serial number.

```
root@raspberrypi4-64:/opt/fdo/data# ls
Normal.blob   ecdsa384privkey.dat   manufacturer_addr.bin   max_serviceinfo_sz.bin   owner_proxy.dat        platform_hmac_key.bin   raw.blob
Secure.blob   ecdsa384privkey.pem   manufacturer_sn.bin     mfg_proxy.dat            platform_aes_key.bin   platform_iv.bin         rv_proxy.dat
```

Figure 7.7: Adding all required data to the device

- Run the FDO application, which runs the linux-client for the first time, which will eventually complete the DI protocol. The DI protocol is now complete. Logs from both Devices and the manufacturer server can validate it. The Device is now manufactured and ready for sale.

```
12:44:32:568 FDOProtDI: Received message type 13 : 1 bytes
12:44:32:568 Writing to Normal.blob blob
12:44:32:568 Hash write completed
12:44:32:569 HMAC computed successfully!
12:44:32:569 Writing to Secure.blob blob
12:44:32:569 Generating platform IV of length: 12
12:44:32:570 Generating platform AES Key of length: 32
12:44:32:571 Device credentials successfully written!!
(Current) GUID after DI: 86192538-27ce-4453-9e87-1bb539f0b95a
12:44:32:572 DIDone completed
12:44:32:572
------------------------------------ DI Successful -----------
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@FIDO Device Initialization Complete@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
root@raspberrypi4-64:/opt/fdo#
```

Figure 7.8: DI log from the Device, Device got its unique GUID and Serial number

```
10   19:02:01.327 [INFO ] HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProv
11   19:02:02.193 [INFO ] Started Manufacturer Service.
12   20:13:44.848 [INFO ] Type 10 [h'850B01686C7866646F303035653132333435590136308201323081B80201003039310B300906035504061302494E310C300A06035504030C0366646F310C300
13   20:13:45.668 [INFO ] Type 11 [h'861865508619253827CE44539E871BB539F0B95A81858205557468 6F73742E646F636B65722E696E7465726E616C820343191F68820C410182024544C0A8006
14   20:13:45.686 [INFO ] Type 12 [[6, h'61A7DF6600BF336EB37363E083A658A80530BD25BF11D269DFDF39C3BFF4785AC14A8F9F9B11E20824CC6014BA64DCDC']]
15   20:13:45.810 [INFO ] Type 13 []
```

Figure 7.9: Manufacturing Server Log, msg 13 means DI protocol complete

7.3.2 Sale of the Device: Transfer of Ownership

The ownership voucher ensures the transfer of ownership in the supply chain. As discussed, the current Owner will send the ownership voucher to the next Owner. This transfer happens through the HTTP protocol and the APIs provided by the manufacturer and owner server. We have used a shell script to generate the ownership voucher, transfer it to the next Owner and Initiate the TO0 protocol. The below Figures can validate the transfer of ownership. Once the TO0 protocol is completed, the Device is ready to be commissioned.

```
vishwas@t560-vishwas:~/fdo-5/pri-fidoiot/component-samples/demo/scripts$ bash extend_upload.sh -e mtls -c ./secrets -s lxfdo005
Client Certificate authentication mode is being used
Success in downloading SECP256R1 owner certificate to owner_cert_SECP256R1.txt
Success in downloading extended voucher for device with serial number lxfdo005
Success in uploading voucher to owner for device with serial number lxfdo005
GUID of the device is 86192538-27ce-4453-9e87-1bb539f0b95a
Success in triggering TO0 for lxfdo005 with GUID 86192538-27ce-4453-9e87-1bb539f0b95a
```

Figure 7.10: Sharing of Ownership voucher and triggering TO0 protocol

```
19:02:02.193 [INFO ] Started Manufacturer Service.
20:13:44.848 [INFO ] Type 10 [h'850B01686C7866646F3030356531323334355901363082013230818802010030393108300906035504061302494E310C300A06035504030C0366646F310C300
20:13:45.668 [INFO ] Type 11 [h'861865508619253827CE44539E871BB539F0B95A818582055574686F73742E646F636B65722E696E7465726E616C820343191F68820C410182024544C0A8006
20:13:45.686 [INFO ] Type 12 [[6, h'61A7DF6600BF336EB37363E083A658A80530BD25BF11D269DFDF39C3BFF4785AC14A8F9F9B11E20824CC6014BA64DCDC']]
20:13:45.810 [INFO ] Type 13 []
20:45:29.776 [INFO ] Manufacturing Voucher serialNo : lxfdo005
```

Figure 7.11: Manufacturer Log: Manufactured voucher for serial no lxfdo005

```
19:05:14.401 [INFO ] Started Owner Service.
20:45:30.192 [INFO ] GUID is 86192538-27ce-4453-9e87-1bb539f0b95a
20:45:30.283 [INFO ] Triggering TO0 for GUID: 86192538-27ce-4453-9e87-1bb539f0b95a
20:45:30.383 [INFO ] TO0 URL is http://host.docker.internal:8040
20:45:30.394 [INFO ] Type 20 []
20:45:31.784 [INFO ] Type 21 [h'B245EE57D6BA44D082DA68AC54562ECF']
20:45:32.248 [INFO ] Type 22 [[[101, h'861865508619253827CE44539E871BB539F0B95A8185
20:45:32.438 [INFO ] Type 23 [86400]
20:45:32.453 [INFO ] TO0 completed for GUID: 86192538-27ce-4453-9e87-1bb539f0b95a
```

Figure 7.12: Owner Log: TO0 completed by Owner server

```
19:03:20.633 [INFO ] Started Rendezvous Service.
20:45:30.766 [INFO ] Type 20 []
20:45:31.764 [INFO ] Type 21 [h'B245EE57D6BA44D082DA68AC54562ECF']
20:45:32.262 [INFO ] Type 22 [[[101, h'861865508619253827CE44539E871
20:45:32.436 [INFO ] Type 23 [86400]
```

Figure 7.13: RV Log: RV server confirming TO0 completion

7.3.3 Provisioning of the Device: Installation and Onboarding

Now, the TO0 protocol is completed, and the Device is ready to be installed. We will power on the Device and connect it to the internet. The Device will then contact the RV server to get the Owner server IP details, completing the TO1 protocol. It will then initiate the T02 protocol and transfer the hawkbit.config file to the Device, and using this config details Device will register itself to the hawkBit server. Everything will happen without any human intervention. The below Figures can validate the zero-touch onboarding to the hawkBit server achieved successfully using FDO 1.1 standards.

```
13:42:00:102 TO2.Done2 started
13:42:00:102 FDOProtTO2: Received message type 71 : 57 bytes
13:42:00:102 Encrypted Message Read: Encrypted Message parsed successfully
13:42:00:102 Encrypted Message (decrypt): Decryption done
13:42:00:102 TO2.Done2 completed successfully
13:42:00:102
------------------------------------ TO2 Successful ----------------------

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@FIDO Device Onboard Complete@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
root@raspberrypi4-64:/opt/fdo#
```

Figure 7.14: Device log : validates the TO2 protocol completed

```
root@raspberrypi4-64:/opt/fdo# ls
DIStatus         data             data_bkp         hawkbit.config  hawkbit.log     linux-client
root@raspberrypi4-64:/opt/fdo# cat hawkbit.config
URL:https://192.168.0.102
ControllerId:86192538-27ce-4453-9e87-1bb539f0b95a
SecurityToken:d88c92af6d9524627429dcfebdb92bfa
root@raspberrypi4-64:/opt/fdo#
```

Figure 7.15: hawkbit.config: validates that correct configurations are sent using T02 protocol

```
Hawkbit config changed at (null)
SWUpdate v2022.12-dirty

Licensed under GPLv2. See source distribution for detailed copyright notices.

[TRACE] : SWUPDATE running :  [print_registered_bootloaders] : Registered bootloaders:
[TRACE] : SWUPDATE running :  [print_registered_bootloaders] :  uboot   loaded.
[INFO ] : SWUPDATE running :  [main] : Using default bootloader interface: uboot
[INFO ] : SWUPDATE running :  [print_registered_handlers] : Registered handlers:
[INFO ] : SWUPDATE running :  [print_registered_handlers] :     dummy
[INFO ] : SWUPDATE running :  [print_registered_handlers] :     archive
[INFO ] : SWUPDATE running :  [print_registered_handlers] :     tar
[INFO ] : SWUPDATE running :  [print_registered_handlers] :     uboot
[INFO ] : SWUPDATE running :  [print_registered_handlers] :     bootloader
[INFO ] : SWUPDATE running :  [print_registered_handlers] :     lua
[INFO ] : SWUPDATE running :  [print_registered_handlers] :     raw
[INFO ] : SWUPDATE running :  [print_registered_handlers] :     rawfile
[INFO ] : SWUPDATE running :  [print_registered_handlers] :     remote
[INFO ] : SWUPDATE running :  [print_registered_handlers] :     shellscript
[INFO ] : SWUPDATE running :  [print_registered_handlers] :     preinstall
[INFO ] : SWUPDATE running :  [print_registered_handlers] :     postinstall
[TRACE] : SWUPDATE running :  [listener_create] : creating socket at /tmp/swupdateprog
[TRACE] : SWUPDATE running :  [network_initializer] : Main loop daemon
[TRACE] : SWUPDATE running :  [listener_create] : creating socket at /tmp/sockinstctrl
[TRACE] : SWUPDATE running :  [start_swupdate_subprocess] : Started suricatta with pid 478 and fd 9
[ERROR] : SWUPDATE failed [0] ERROR : Configuration file /etc/fw_env.config wrong or corrupted
[INFO ] : SWUPDATE running :  [read_state] : Key 'none' not found in Bootloader's environment.
[ERROR] : SWUPDATE failed [0] ERROR : Cannot read stored update state.
[TRACE] : SWUPDATE running :  [start_suricatta] : Server initialized, entering suricatta main loop.
[TRACE] : SWUPDATE running :  [channel_log_effective_url] : Channel's effective URL resolved to https://192.168.0.102/DEFAULT/controller/v1/86192538-27ce-4453-9e87-1bb539f0b95a
[TRACE] : SWUPDATE running :  [server_set_config_data] : ConfigData: http://hawkbit:8080/DEFAULT/controller/v1/86192538-27ce-4453-9e87-1bb539f0b95a/configData
[TRACE] : SWUPDATE running :  [server_get_deployment_info] : No pending action on server.
```

Figure 7.16: hawkbit.log: validates the device is registered to hawkBit server

```
21:10:39.616 [INFO ] Type 30 [h'8619253827CE44539E871BB539F0B95A', [-35, h'']]
21:10:39.782 [INFO ] Type 31 [h'BE82AC6C751E4963B626D67DEAD11FEB', [-35, h'']]
21:10:39.804 [INFO ] Type 32 [h'A1013822', {}, h'A219010051018619253827CE44539E
21:10:39.885 [INFO ] Type 33 [h'A1013822', {}, h'82818444C0A8006674686F73742E64
```

Figure 7.17: RV Log: RV server confirming TO1 completion

```
21:11:00.019 [INFO ] Type 60 [2000, h'8619253827CE44539E871BB539F0B95A', h'A18EDB86F9B35DA01822B06C8A62A458', "ECDH384", 3, [-35, h'']]
21:11:00.185 [INFO ] Type 61 [h'A1013822', {256: h'FFB8DC205E064B9480A9936B0947BF8F', 257: [11, 1, h'3059301306072A8648CE3D020106082A86
21:11:00.222 [INFO ] Type 62 [0]
21:11:00.250 [INFO ] Type 63 [0, [h'A1013822', {}, h'8482382A58308E342552ED95D563EA4C8E7CC3F500FB0566E81D18A854ABDF953F5E54F5FA65F1EA42
21:11:00.627 [INFO ] Type 64 [h'A1013822', {-259: h'DF19C90F1AC7CDF1BD5BF0B027F7CF43'}, h'A319010051018619253827CE44539E871BB539F0B95A0
21:11:00.900 [INFO ] Type 65 [h'A10103', {5: h'EFCF4C00FB0CA91B20E30B5B'}, h'C2365DC5F56326FDE12E67383E3D6653B4F2ED491578E2D141CABA216A
21:11:00.925 [INFO ] Type 66 [h'A10103', {5: h'E73AC7A79F2EA9EB9AEAD8AA'}, h'F523285B5002F2EE790C248667FD261EF2698ED077C20B545774DAEA20
21:11:00.990 [INFO ] Type 67 [h'A10103', {5: h'5F524EBA3C4F09C6E66A02B7'}, h'A18ECF03F2ED0B6D19B18A0E0FC7A17A11AA']
21:11:01.341 [INFO ] Type 68 [h'A10103', {5: h'061444775E076DA7241849D8'}, h'2A00B2C011387D9D61485361610F833D76B198386FA8640ED202F4ECCC
21:11:01.393 [INFO ] Type 69 [h'A10103', {5: h'AB0F06850366C354A3945D1D'}, h'67BB996D9D255BDABC11857A7C88731A740492DB']
21:11:01.416 [INFO ] Type 68 [h'A10103', {5: h'145FCC010FEBA2EDB66A1A64'}, h'8F66815BE754BD4D1B5586C377474C88FE7CB3CADDB988DA5F4222C6CF
21:11:07.811 [INFO ] Type 69 [h'A10103', {5: h'F0AEA6ADD7BE2CA9CC0586D0'}, h'9214743B10941690F429FF7F84235B7027599E56ED8E4551C548D4BABF
21:11:07.827 [INFO ] Type 68 [h'A10103', {5: h'B1AB4E87A438DAFF277698F1'}, h'BD34F7B91A5748BB6D84B2B8096957EB8B266B']
21:11:13.235 [INFO ] Type 69 [h'A10103', {5: h'EAE5CD0697E688E9690E7097'}, h'846831CADE78038F41C01CC7DC63684B7CA5BD055846A28C691E02919D
21:11:13.254 [INFO ] Type 70 [h'A10103', {5: h'58AC56F423E25B23E6B6D10A'}, h'F9FE9592D1E13D9A282EE1F9893A1F89724401B370DB640212149DAFD5
21:11:13.295 [INFO ] Type 71 [h'A10103', {5: h'C24EF80308B1624E2E2CD716'}, h'355F0CD8BAB8D406907716669C5D2D86FEC19521E48B7D14D1FB195F5F
```

Figure 7.18: Owner server Log: Owner server confirming TO2 completion. msg71 means T02 completed

7.4 Testing & Evaluation

We have performed various tests to evaluate our software solution's reliability, scalability, and latency. We wanted to check how our servers will react when multiple clients try to connect with them and query for getting hawkbit.config file. To test this, we created a simulated environment, creating a docker image of our IoT edge device. The rest of the servers were also in a containerized environment. The demonstration and the source code can be found here: https://github.com/Vishwasrao1/fidoiot-edge-device.

7.4.1 Scalability

We have tested scalability by using simulating multiple devices trying to connect to the server. Initially, the server was getting jumbled between the serial number, GUI, and security Token of the device. However, modifying our source code has improved the scalability.

We simulated 1,10,20,30 devices at a time and our solution was perfectly fine to onboard 30 devices within less than a minute. We tried going beyond 30 but due to computing power restrictions, the host machine was not capable of doing so.

We must note that these are virtual devices, which don't have any other high-priority processes, the onboarding is quick such that 30 devices are onboarded in half a minute. However, our prototype device has other high-priority processes which must run at the time of boot so the approximate time to onboard the device is less than a minute. Considering this scenario, if we can onboard 30 devices per minute, that means 1800 devices per hour, and eventually 43200 devices can be onboarded in 24 hours. This makes our application highly scalable for large-scale IoT deployments.

7.4.2 Latency

We have observed the latency of the devices when multiple devices try to communicate with the server. The latency increases as the number of parallel processes required increases. The data is shown in below table and graph.

Sr No	Device Name	Actual Time (s)	User Space Time (s)	Kernel Space Time (s)
1	1 Device	13.167	0.1	0.044
2	10 Devices in parallel	14.95	0.116	0.046
3	20 Devices in parallel	20.627	0.126	0.047
4	30 Devices in parallel	24.936	0.133	0.0557

Table 7.2: FDO Device latency

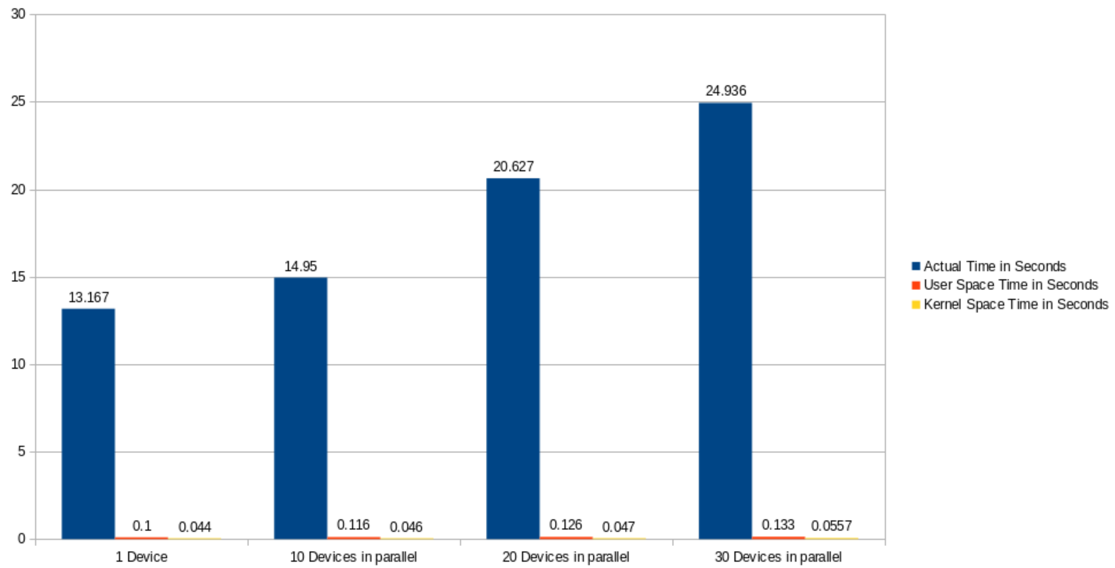


Figure 7.19: Latency of FDO to hawkBit server

As mentioned earlier, this data is for a simulated environment with virtual devices running as containers. on actual hardware, it took approximately 60 seconds to onboard the device considering the boot time.

7.4.3 Reliability

We tested our prototype device in case of power and network failure. Our solution is highly reliable and takes care of such errors and exceptions.

Power Failure Testing

The device functions reliably when power failure happens at the time of onboarding. We started the device onboarding and unplugged the power cable to simulate a power failure.

Manufacturing: During manufacturing, if a power failure occurs at the time of the device initialization. The Device Initialization status will be stored as incomplete, and when we retry the device initialization again, the DI protocol will complete successfully.

Provisioning: During provisioning, if a power failure occurs and Device onboarding is incomplete. We need to power up the device again. The systemd service fdolinuxclient.service will take care to start the onboarding process again and onboard the device. If the TO1 protocol is already completed before the power failure, our software solution will start the TO1 protocol again at the next retry. Similarly, if any of the onboarding steps still need to be completed, the complete onboarding process will run again at the next boot.

Network Failure Testing

The device is highly reliable if any network failure exception occurs during onboarding. We started the device onboarding and unplugged the network cable to simulate a network failure.

Manufacturing: During manufacturing, if a network failure occurs at the time of the device initialization. The device will retry to connect to the manufacturer's server. The DI status will be stored as failed if the connection does not happen during three retries. Whenever the network is restored, the manufacturer should try device initialization again.

Provisioning: A network failure during device onboarding does not cause any severe breakdown. Instead, it will retry to connect the server, and if the network is restored, it will continue the process and onboard the device. If the network is not restored, the device will store the onboarding status as failed. We need to reboot the device and ensure the network is available to start the onboarding process again.

7.5 Onboarding of Smart Energy Meter

Research question 4 was about choosing a suitable method for onboarding smart meters. As we develop smart cities and buildings, we inevitably need to deploy IoT on a large scale. When considering onboarding solutions for smart energy meters, several important factors must be remembered. These include the level of security needed to protect the device and data, the ability to scale the method to handle a large number of devices, compatibility with the device's hardware and software, cost and time required for implementation and maintenance, and the ability to support specific use cases such as remote monitoring, meter data analytics, billing, and demand response management.

To ensure efficiency and avoid delays, the onboarding process of the smart meters needs to be zero-touch, especially considering the large-scale deployment. FDO simplifies the installation and configuration process by having plug-and-play devices. FDO makes sure that the installation is quick and worry-free, especially when we have to install a large number of devices. The smart meters are shipped with the default login id and password. If we use FDO, this can be avoided, reducing the device's vulnerability to cybersecurity attacks. There will not be any need for skilled installation technicians as the onboarding is zero-touch.

Moreover, we can integrate various DMS to provide runtime integrity checks, software updates, and certificate management which is important in the case of smart energy meters to handle the overall lifecycle of the device. Smart energy meters are mostly used for remote monitoring, meter data analytics, billing, and demand response management. Each energy provider may have developed its cloud solution to satisfy these use cases. Each energy provider may have different preferences regarding cloud providers. It means the manufacturer has to take care of each device and cloud combination when manufacturing the device. However, this can be resolved through the late binding capabilities of FDO. Customization can be done at the end of the supply chain by using identical devices for all providers. The energy provider is free to choose the cloud provider of his choice at the time of onboarding; therefore, the energy meters need not be preconfigured.

Considering all the merits of FDO, it is recommended to choose FDO for the onboarding process of smart energy meters. However, it may not be cost-effective for small-scale deployment.

8 Conclusion and Future Scope

This study has found that large-scale IoT deployments can benefit significantly from zero-touch onboarding. Three different methods were evaluated: FIDO Device Onboard (FDO), Keylime, and OPCUA Onboarding. Each approach had its pros and cons, but it is clear that zero-touch onboarding is future of IoT deployments. This method offers numerous benefits, such as increased security, scalability, efficiency, and cost-effectiveness. By removing the weaknesses associated with traditional manual onboarding methods, zero-touch onboarding can simplify the process while maintaining strong security measures. It also automates device management and provisioning, improving interoperability and reducing operational expenses.

In conclusion, We choose FIDO Device Onboard (FDO) as the preferred solution for IoT device onboarding over Keylime and OPCUA Onboarding. FDO's complete zero-touch approach ensures ease of use and automation, saving time and reducing costs. It supports a wide range of devices, provides scalability, and ensures reliable onboarding despite failures. FDO follows standard practices as an open-source project, offers a late binding feature for cost-effective manufacturing, and provides a user-friendly experience with options for cloud provider selection and secure device decommissioning/resale. These advantages make FDO the ideal choice for organizations seeking a versatile, efficient, and user-centric onboarding solution for their IoT deployments. In addition, zero-touch onboarding aids organizations in complying with regulatory requirements, safeguarding sensitive data, and adhering to regulatory standards such as IEC 62443.

We developed a novel software solution to facilitate zero-touch device onboarding to the hawkBit server using FDO 1.1 standards. The significant finding was that we could onboard the device in less than a minute and onboard up to 43200 devices per day, eventually making it more than a million per month. The scalability of our solution is promising. The FDO did not provide any specific mechanism for the certificate or runtime integrity management; however, these things can be easily integrated using various Device Management Services. One possible approach could be integrating FDO and Keylime. The Keylime does not have any automatic method to get the device's IP address. FDO can provide this to keylime, and from there, Keylime would take care of the certificate and runtime integrity measurement. Right now, we have integrated the software update manager with FDO. If we integrate all three management services into FDO, it will be a significant achievement. There is also scope for improvement in the implementation of the solution.

To sum up, this thesis represents a significant contribution to large-scale IoT deployments using zero-touch onboarding. Through extensive research and analysis, we have provided valuable insights into the benefits and challenges of implementing such deployments. This thesis has led to a solid foundation for future research and advancements. The findings of this research serve as a valuable guide for organizations and stakeholders looking to leverage the potential of IoT at scale while ensuring secure and efficient device onboarding.

Bibliography

[1] Hasan, M. (2022, June 14). State of IOT 2022: Number of connected IOT devices growing 18% to 14.4 billion globally. IoT Analytics. Retrieved May 8, 2023, from https://iot-analytics.com/number-connected-iot-devices/

[2] Fido device onboard specification. FIDO Alliance. (2021, December 14). Retrieved May 8, 2023, from https://fidoalliance.org/specs/FDO/FIDO-Device-Onboard-RD-v1.1-20211214/FIDO-device-onboard-spec-v1.1-rd-20211214.pdf

[3] Schear, N., Cable, P. T., Moyer, T. M., Richard, B., & Rudd, R. (2016). Bootstrapping and maintaining trust in the cloud. Proceedings of the 32nd Annual Conference on Computer Security Applications. https://doi.org/10.1145/2991079.2991104

[4] Keylime documentation. Keylime Documentation - Keylime Documentation 7.0.0 documentation. (n.d.). Retrieved May 8, 2023, from https://keylime.readthedocs.io/en/latest/

[5] Open source summit north america 2022. (2022, June 1). Keylime: Bootstrap and Maintain Trust on the Edge, Cloud, and IoT - Lily Sturmann & Michael Peters, Red Hat. ossna2022. Retrieved May 8, 2023, from https://ossna2022.sched.com/event/11Nme/keylime-bootstrap-and-maintain-trust-on-the-edge-cloud-and-iot-lily-sturmann-michael-peters-red-hat

[6] Arthur, W., & Challener, D. (2015). A practical guide to Tpm 2.0 using the Trusted Platform Module in the New Age of security. Apress.

[7] Ua Part 21: Device onboarding. OPC UA Online Reference - Released Specifications. (2022, November 1). Retrieved May 8, 2023, from https://reference.opcfoundation.org/Onboarding/v105/docs/

[8] Hardware threat landscape and good practice guide. ENISA. (2021, August 26). Retrieved May 8, 2023, from https://www.enisa.europa.eu/publications/hardware-threat-landscape

[9] What is a container? Docker. (2023, February 21). Retrieved May 8, 2023, from https://www.docker.com/resources/what-container/

[10] Ei. (2019, December 18). The field of ëmbedded linuxëxplained! Embedded Inventor. Retrieved May 8, 2023, from https://embeddedinventor.com/a-clear-cut-explanation-to-embedded-linux/

[11] Software. Yocto Project. (n.d.). Retrieved May 8, 2023, from https://www.yoctoproject.org/software-overview/

[12] Raspberry pi compute module vs Raspberry Pi 4 - jfrog connect (formerly UPSWIFT). JFrog Connect. (n.d.). Retrieved May 8, 2023, from https://jfrog.com/connect/post/raspberry-pi-compute-module-vs-raspberry-pi-4/

[13] Mixos. (2021, March 31). Raspberry pi compute module 4 IOT Router Carrier Board Mini. Electronics. Retrieved May 8, 2023, from https://www.electronics-lab.com/raspberry-pi-compute-module-4-iot-router-carrier-board-mini/

[14] Raspberrypi. (2022, May 6). Usbboot/readme.md at master · raspberrypi/USBBOOT. GitHub. Retrieved May 8, 2023, from https://github.com/raspberrypi/usbboot/blob/master/secure-boot-recovery/README.md

[15] Intel. (n.d.). Intel/bmap-tools: BMAP tools. GitHub. Retrieved October 7, 2022, from https://github.com/intel/bmap-tools

[16] Project, T. E. hawkB. (n.d.). Eclipse hawkbit. The Community for Open Innovation and Collaboration. Retrieved May 8, 2023, from https://www.eclipse.org/hawkbit/

[17] ISO/IEC/IEEE 42030:2019. ISO. (2019, July 24). Retrieved May 8, 2023, from https://www.iso.org/standard/73436.html

[18] Brown, Alan & Wallnau, Kurt. (1996). Framework for evaluating software technology. Software, IEEE. 13. 39 - 49. 10.1109/52.536457.

[19] Office of Aerospace Studies Air Force Materiel Command (AFMC) OAS/A9. (n.d.). Aoa Handbook - Afacpo.com. Retrieved May 8, 2023, from https://www.afacpo.com/AQDocs/AoAHandbook.pdf

[20] Secure IOT begins with zero-touch provisioning at scale. IoT Security. (2021, April 1). Retrieved May 8, 2023, from https://azure.microsoft.com/en-us/resources/secure-iot-begins-with-zero-touch-provisioning-at-scale/

[21] Kumar, V., Mohan, S., & Kumar, R. (2019). A voice based one step solution for bulk IOT device onboarding. 2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC). https://doi.org/10.1109/ccnc.2019.8651724

[22] Eustace Asanghanwa Principal Program Manager. (n.d.). The blueprint to securely solve the elusive zero-touch provisioning of IOT devices at scale: Azure blog and updates: Microsoft Azure. Azure Blog and Updates | Microsoft Azure. Retrieved May 8, 2023, from https://azure.microsoft.com/en-gb/blog/the-blueprint-to-securely-solve-the-elusive-zerotouch-provisioning-of-iot-devices-at-scale/

[23] Intel® secure device onboard simplifies the supply chain. Intel. (n.d.). Retrieved May 8, 2023, from https://www.intel.com/content/www/us/en/internet-of-things/secure-device-onboard.html

[24] Zoualfaghari, M. H., & Reeves, A. (2019). Secure & zero touch device onboarding. Living in the Internet of Things (IoT 2019). https://doi.org/10.1049/cp.2019.0133

[25] Fido device onboard. LF Edge. (2023, April 19). Retrieved May 8, 2023, from https://www.lfedge.org/projects/fidodeviceonboard/

[26] Keylime. (n.d.). Keylime/keylime: A CNCF project to Bootstrap & Maintain Trust on the edge / cloud and IOT. GitHub. Retrieved May 8, 2023, from https://github.com/keylime/keylime

[27] Contributers, F. I. D. O. D. O. (n.d.). Home - Fido device onboard. Retrieved May 8, 2023, from https://fido-device-onboard.github.io/docs-fidoiot/1.1.4/security-best-practices/

All links were last followed on May 8, 2023.