University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

# DAA & PUF-based Onboarding of IoT Devices

*Julia Kostadinova*
*Zurich, Switzerland*
*Student ID: 17-741-174*

Supervisor: Katharina Müller, Dr. Eryk Schiller, Dr. Bruno Rodrigues,
Date of Submission: December 20, 2023

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

# Abstract

Das Ziel der Arbeit ist es Direct Anonymous Attestation (DAA) und Physically Unclonable Functions (PUF) zu untersuchen um diese Bescheinigungsmethoden mit dem Extensible Authentication Protocol zu kombinieren. IoT-Geräteonboarding ist zusammengesetzt aus einigen manuellen Schritten, was in langer Hinsicht nicht nachhaltig ist. Eine Lösung ist es die Bescheinigung des Gerätes mittels DAA oder PUF zu automatisieren bevor das Gerät beginnt EAP zu nutzen um Zugriff zu einem Netzwerk zu erlangen. Ein Ziel wares eine Struktur für beide Bescheinigungsvarianten auszuarbeitn. Ein weiteres Ziel bestand daraus einen Prototyp mit einer der beiden Varianten zu implementieren.

The goal of this thesis is to investigate Direct Anonymous Attestation (DAA) and Physically Unclonable Functions (PUF) to combine these attestation methods with the Extensible Authentication Protocol. IoT device onboarding is comprised of many manual steps which is not sustainable in the long run. One Solution is to automate attestation using either DAA or PUF before the device starts using EAP to access a network. One Goal was to map out a structure for both attestation methods. Another Goal was to implement a prototype using one of these methods.

# Acknowledgments

I am grateful to everyone who supported me while working on this project. My Gratitude especially goes to Katharina Müller as my supervisor. Her support and guidance were critical and have impacted the outcome of this work. Further, I want to thank Prof. Dr. Burkhard Stiller for allowing me to write my thesis at the Communication Systems Group (CSG) of the UZH as well his guidance concerning the progression of my work. Special thanks also go to Dr. Eryk Schiller for introducing me to this topic.

Finally, I also want to thank my partner, my family and my friends for their support and patience during this time and not to forget my cats for providing emotional support.

# Contents

# Chapter 1

# Introduction

The usage of the IoT scales rapidly, with an estimation by the National Cybersecurity Center of Excellence (NCCoE) expecting that by 2025 there will be around 40 billion devices connected globally [1]. This expansion however also increases the need to protect the IoT lifecycle. Especially the addition of new edge devices shall be secured because this onboarding also exposes the existing network to potential threats from outside. Currently, a team of researchers is working on a project automating and simplifying the integration and maintenance of the IoT lifecycle named CERTIFY [2]. Within the CERTIFY project, this thesis is going to work on addressing one aspect of the IoT lifecycle automation.

## 1.1 Motivation

IoT devices need credentials and a policy so that they can join a network. This procedure is called network-layer-onboarding [1]. Before joining a network however devices and networks need to be able to attest themselves. Attestation in this context means that an IoT node needs to show proof that its identity and security values can be trusted[1]. This is crucial for reducing the risk of potentially communicating with a malicious entity [1].

Executing the entire onboarding process and then maintaining every single IoT device manually is neither sustainable nor manageable [1]. A manufacturer could provide these credentials locally, but this leads to a manufacturer needing to customise the network-layer-onboarding for every specific customer [1]. As previously implied many times the onboarding process is done by hand where several problems can arise. Often devices are onboarded without verification or credentials are sent using an unprotected connection. Another source of danger described in [1] is all devices connecting to a specific network sharing a password, without consideration of a platform's identity. One compromised device would suffice to gain access to the network. Underlined by[1] again providing unique credentials to every platform manually would be error-prone and would need too many resources.

All of these issues could be automated to reduce manual labour. Promising candidates for such an automated onboarding process are direct anonymous attestation (DAA) [3] and

physically unclonable functions (PUF) [4]. Evaluating these methods' feasibility for IoT device onboarding by combining them with the extensible authentication protocol (EAP) will be the main question this thesis will try to answer. Based on the choice of method another goal in this context will be to implement this automation in a test environment. With the previous points in mind, the research questions for this thesis are the following:

- **RQ 1:** Which use cases result in an IoT device needing to authenticate to a network?
- **RQ 2:** What are the advantages and disadvantages of DAA and PUFs and how do these methods compare?
- **RQ 3:** How could EAP be extended to support DAA and PUF while respecting the anonymity of a device?
- **RQ 4:** How would an EAP extension be implemented concretely to attest an IoT device?

## 1.2 Description of Work

In this thesis, the starting step of the IoT lifecycle was studied from different angles. First literature needed to be consulted to understand the requirements, specialities and steps of DAA, PUF and EAP. Following this step use cases could be found and described.

In the subsequent step, a potential procedure was formulated based on the information and knowledge gained in the previous step. Since consolidating and designing a procedure is tied to extensively investigating both attestation methods and the authentication protocol. Based on this investigation strengths and shortcomings of both methods could be documented. After mapping out a potential structure in the design chapter the available hardware needed to be checked for their properties, and configured. Another important aspect is how their configuration is handled.

It is important to note that because of this work being done concerning the CERTIFY project, certain implementation decisions and hardware choices are based on predefined aspects. Influences of CERTIFY are most noticeable in Chapter 5. Later after the decision was made to focus the implementation only on DAA more configuration work needed to be done to install the dependencies and commands for the TPM. After the installation, the TPM was used to generate all necessary attestation components and prepare them to be sent to the laptop acting as a server. To implement the connection to the server the library variant of cURL was used. Once the platform was set up as a client, communication on the server side had to be implemented. First, the server needed to generate its keys and provide an access point to the public keys and parameters. The server side also was implemented to accept the files sent by the board and process the inputs. Lastly, the endorsement key and certificate received from the board were compared.

The implementation finally set the ground for the continuation of the DAA scheme in future developments which resulted in a detailed proposition for future work.

## 1.3 Thesis Outline

Following the introduction the background will be discussed in chapter 2. The background will go into more detail about DAA and PUFs based on reviewed literature. After reviewing both DAA and PUF the background will continue to discuss the topic of EAP concerning the attestation methods. Chapter 3 will cover the related work regarding DAA, PUFs and EAP. Following this in chapter 4 first the topic will be defining use cases where device attestation needs to occur. The next section of the design will go into more detail about specific plans on how to extend EAP with the respective Attestation method. Chapter 5 will first discuss what hardware is used in context with regards to the CERTIFY project on one hand and which devices are available for the scope of this work on the other hand. The following section will cover implementation steps and explain them in more detail. The evaluation in chapter 6 will cover findings during the entire scope of this thesis and finally, the summary and conclusion are discussed in chapter 7 including future work that could be made based on the results of this thesis.

# Chapter 2

# Background

This chapter will elaborate more on the state of the potential attestation methods DAA and PUFs by reviewing the available literature. Followed by introducing EAP and its extensions [5] that will be important to the context of the current setup.

## 2.1 State of Literature for DAA and PUF

This section will discuss both attestation methods, and evaluate which of these methods will be more suitable authentication for this project. At first, the thesis is going to discuss further research that was done on these methods. The following two sections will discuss both methods in more detail starting with DAA and then PUFs. The Goal is to Find the advantages and disadvantages of each method. CERTIFY is implementing DAA for their test setup, therefore the focus will rely more heavily on DAA as well. To also review potential alternatives PUFs will also be taken into consideration.

### 2.1.1 Methods

The starting points for reviewing the literature are the papers written by Brickell, Camenisch and Chen for DAA [3] and Devadas and Suh for [4]. From both starting points to investigate further material forward snowballing was used to get a more recent state of both methods.

DAA in the ACM library yielded 565 results using this method. To cover the current state of the art only the years 2020-2023 are considered which reduced the results to 67. The remaining results were then filtered with different criteria. One goal was to gather papers that try to improve DAA and discuss possible improvements. Another goal was to gather papers that implement DAA in their project to have examples. The last filtering measure was the accessibility of papers, some had to be removed due to paywalls. With those goals screening the abstracts resulted in 20 remaining candidates for DAA. Reading the remaining papers in more detail reduces the number of sources for this review to seven.

From these seven results, backward snowballing revealed more relevant literature which can deliver valuable insight into the topic.

For the PUF-based authentication forward snowballing resulted in 749 citations of the initial starting source. Limiting the accepted results again only to the years 2020-2023 reduced the number of papers for reviewing to 137. From these 137 sources, 104 were discarded. The reasons for that were either that the solution was using additional hardware like FPGA or the PUF was applied to components that were not in the scope of this work. Another reason for excluding a paper was its availability, so paywalls that were not avoidable through the institution also meant excluding the source. These conditions reduce the number of documents to six potential sources.

### 2.1.2 Direct Anonymous Attestation

Brickell et al. propose DAA [3] to authenticate devices while maintaining the privacy of that platform user trying to connect. This scheme considers four actors which are specified by the authors as a trusted platform module (TPM), a host, an issuer and a verifier [3].

As described previously the first that needs to happen before a device can join a network is to establish trust between the parties. For that to be possible Brickell et al. [3] begin with the step to set up the issuer, which will be the party that provides the platform with its certificate. The setup of the issuer is described as generating its private and public key as well as proof that the generation was done correctly. In the end, the public key, and the proof are published.

The IoT device verifies the issuer's public key using the proof that was published along it before starting the join process [3]. The join protocol described in [3] is the point where the TPM computes the necessary credentials, which after the issuer determines that the credentials are valid results in the platform receiving a certificate from the issuer.

A platform with the issued certificate can now sign and also authenticate its messages to a verifier. As stated in [3] both the host and the TPM must compute the signature, this also plays into account that a corrupted host cannot sign messages alone.

The final step in the process description in [3] explains how a verifier uses the information it receives to allow access and thus link the device if a verification was successful.

This initial version of DAA requires devices to compute higher orders of exponents in combination with modulus as can be seen in [3]. Therefore newer implementations based on elliptic curve cryptography (ECC) such as the ones in [6]–[8] target to reduce the resources that are needed for computation given that a TPM is a small chip which is limited in its computational power.

ECC-DAA uses the endorsement key hierarchy that is specified by the TCG [9]. An endorsement key (EK) consists of an asymmetric key pair where only the public key can be read out of the TPM while the private part remains inaccessible, crucial here is to note that in the TPM 2.0 specification, a TPM can have multiple different kinds of endorsement keys, which are all generated from a secret seed that is inserted by the manufacturer and

unique to each TPM[9]. Furthermore, as specified by the TCG [9] changing the secret seed devalues all certificates and keys previously generated, effectively meaning that the device will have to freshly re-attest its credentials with new keys and certificates.

### 2.1.3 Physically Unclonable Functions

Physically unclonable functions (PUF) use the physical properties of the hardware itself instead of using cryptographic primitives which require many calculations and also storage of secrets in memory which can become compromised as Suh and Devadas [4] describe. Their implementation uses a field programmable gate array (FPGA) to construct two different kinds of PUFs, an arbiter PUF and a ring oscillator PUF. The arbiter PUF is based on a row of multiplexer pairs and sends a signal through both rows and in the end, an arbiter decides which of these signals is faster [4]. On the other hand ring oscillator PUFs use different frequencies generated by an array of ring oscillators from which a pair is chosen to compare against each other [4].

Other PUF variants which do not necessarily need an FPGA are those reading out values from RAM. Depending on which kind of RAM either dynamic or static RAM a PUF will use slightly different properties for a unique hardware key. For example Sutar, Raha and Raghunathan [10], [11] implemented D-PUF which uses the refresh-pause interval of a dynamic RAM. However, their implementation [10] still uses an FPGA. Alternatively, for devices with static RAM other PUFs are relevant such as an SRAM PUF that uses the bias of SRAM cells towards 0 or 1 and at startup as a unique key as was done in the work of Niya, Jeffery and Stiller in [12].

A challenge when using PUFs, however, is that reading out physical values of integrated circuits tends to be noisy, therefore the output of the PUF needs to be error corrected and usually this entails implementing threshold-based ”fuzzy” logic as it is described in [4], [10], [12].

## 2.2 Extensible Authentication Protocol

To authenticate, parties can use the extensible authentication protocol (EAP). The protocol specified by Aboba, Blunk, Vollbrecht, Carlson and Lekowetz [5] is a framework that does not restrict specifically which authentication methods need to be used.

EAP is mainly used for network access authentication, where devices trying to access a network are asked to authenticate themselves by a server managing network access. As specified in [5] EAP selects a specific authentication mechanism depending on the authentication method the peer requires more information to. This can be done without needing to update the authenticator itself but by having the authenticator relay the information to a different backend server which provides all functionality needed for verification [5].

RADIUS, as described by Rigney, Willens, Rubens and Simpson [13] designates a Network Access Server (NAS) as the client communicating to a RADIUS server. The RADIUS

server provides authentication methods to verify the user via the NAS [13]. In the context of the EAP the NAS would be the authenticator and the RADIUS server is the mentioned AAA server.

Fajardo, Arkko, Loughney, and Zorn [14] describe the Diameter protocol as an improvement to the RADIUS protocol where they argue that RADIUS comes with a larger administrative burden.

The EAP type targeted in this work is EAP with transport layer security (EAP-TLS) as described in [15]. The reason why EAP-TLS is a candidate for implementation is the fact that it requires the peer and the authenticator to verify each other [15], [16].

# Chapter 3

# Related Work

Intending to improve the DAA scheme proposed in[3] in 2004, Chen and Feng 2008 in [6] propose a version that uses elliptic curve cryptography, shortening the length of keys and signatures and improving the computational performance compared to the initial version that is based on RSA cryptography. The continuous improvement of the scheme proceeds with Brickell and Li [7] where they, based on Chan and Feng's previously mentioned work, also introduce a pairing-based DAA scheme that even further reduces the need for TPM resources and downsizes the written code needed. Two years later Zhang and Liu [17] use Brickell and Li's DAA to implement an anonymous authentication scheme that combines DAA with EAP-TLS for WLAN. Chen and Li[18] demonstrate methods to implement different signature schemes using the TPM version 2.0 functionality DAA is included. Regarding security Camenisch, Drijvers and Anja in [19] further propose a security definition for DAA where the TPM and its Host are separate entities with each entity taken into consideration for potential corruption. Also addressing security shortcomings Camenisch, Chen, Drijvers, Lehmann Novick and Urian [20] proposed small changes to the current TPM 2.0 interfaces to mitigate shortcomings like missing security proofs or Diffie-Hellmann oracles that are directly used with the TPMs secret key. Whitefield, Chen, Sasse, Schneider, Treharne and Wesemeyer in [21] analyze the ECC-DAA scheme resulting in a new prover model called TAMARIN Prover and find that one compromised TPM can result in uncovering secrecy properties for other TPMs.

Following the explanation in [4] by Suh and Devadas many potential circuits can serve as PUFs given they are stable enough. Sutar, Raha and Raghunathan propose D-PUF, which uses the refresh pausing of a dynamic RAM. In a later work Sutar, Raha, Kulkami, Shory, Tew and Raghunatan were able to develop a true random number generator using the previously developed D-PUF. Using a PUF for static RAM Niya, Jeffery and Stiller in [12] derive a unique identifying key for IoT Devices which are then registered on the Etherium blockchain. Nimmy, Sankaran and Achuthan in their work [22]discuss the risks of storing challenge-response pairs as it is done conventionally and propose a protocol for IoT with geometric threshold secret sharing to mitigate the vulnerability. A different approach to PUFs is shown by Vaidya, Praha and Manju [23] who researched a GPIO-based PUF and evaluated 12 devices for a month.

EAP is adapted to various usages depending on what is needed. Intel provides documentation [16] on the different EAP types, such as EAP-MD-5 Challenge-based EAP. Other types presented in [16] are lightweight EAP (LEAP) formerly a proprietary by Cisco or EAP-SIM which is the standard to authenticate devices participating in mobile communication in other words at one point every mobile phone with a SIM card has gone through authentication using EAP-SIM. A variation of EAP-TLS specified in [16] adds tunnelling to transport-layer-security eliminating, called EAP-TTLS, the need for client certificates. Another EAP type that does not require a client certificate is protected EAP (PEAP).

# Chapter 4

# Design

The goal of this chapter is to discuss what the implementation will need to complete the entire authentication process. The first section will discuss potential use cases where authentication to a network is required. After discussing use cases, the second section will cover EAP. DAA will be discussed in the third section followed by PUF which will be the topic for the last section before the different components are combined.

## 4.1 Use-cases

This section will discuss scenarios where network layer device onboarding is happening, and go into detail about current actors and specialties of that setting. Several outcomes can cause devices to need onboarding to a new network.

As previously described in the introduction it is expected that the speed at which new IoT Infrastructure needs to be set up is growing rapidly to the point where human interaction will not be sufficient. [1] describe the possibility of providing the necessary credentials during production. So the first scenario of onboarding starts with a factory of fresh IoT devices where credentials are issued for the first time.

The Guidelines of the Trusted Computing Group[24] also explain the cases of reselling or decommissioning a used IoT device. The TCG Guidelines [24] advise that there should be a method to erase all sensitive Data from the previous owner of the device and reset itself to factory settings. Devices equipped with a trusted platform module (TPM) can execute the necessary steps by applying *TPM2_Clear*". The next step of the process is to clear the master encryption key (MEK) of the device. An MEK is used to encrypt other keys that are stored [25]. After those steps, the new owner can verify that the reset has been done correctly and set up their credentials as explained in the Guidelines. [24]. Often IoT devices have keys that were integrated by the manufacturer, depending on how this was done those keys either need to be re-provisioned or need to be set such that a change of owner is possible [24]. As previously mentioned in the background for DAA such a revocation can be done by [9] regenerating the seed on which the endorsement

keys and therefore all following credentials of the TPM are based on using the command *TPM2_ChangeEPS*.

The third case where an IoT device may need to be onboarded again is after recovery from being compromised. Again here depending on the kind of state the device was the previously explained regeneration would be a viable approach.

Regular re-attestation could also be a possible mitigation to the risk. Suh et al. [4] describe where memory can be read out after storing the same key for too long. It is possible that regular changes would render the memory not so easily readable since some parts would be overwritten regularly and reading out memory would not reveal the information reliably.

## 4.2 Extensible Authentication Protocol

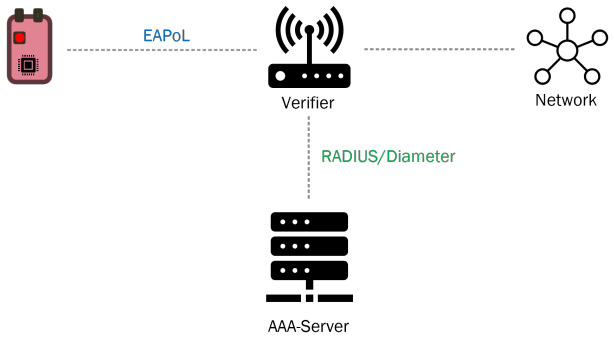


Figure 4.1: EAP components and connections

The EAP, in general, describes three actors. These actors are the supplicant also called the peer, the authenticator and the authentication server also known as AAA-server as explained in [5]. The peer in this context is the IoT device that tries to access the network. The verifier acting as the gatekeeper of the network begins the EAP Process by requesting the device to authenticate itself [5]. This implies that the device previously needs to have completed all necessary steps to gain attestation that it can be trusted. All components are shown in Figure 4.1. The peer and the authenticator communicate via a local area network (LAN), hence the connection is labelled EAP over LAN (EAPoL) both actors are at the other end of the link between them.

The authenticator according to the task description for this thesis is meant to communicate to an AAA-Server. From the information in [5] an authentication server implementing AAA methods only allows the authenticator to be used as a pass-through connection, this means the verification process is executed by the server while the authenticator only redirects messages between the peer and the server. The protocols used for communication between authenticators and AAA-servers as explained in [5] and shown in Figure 4.1 are either RADIUS or the newer Diameter protocol.

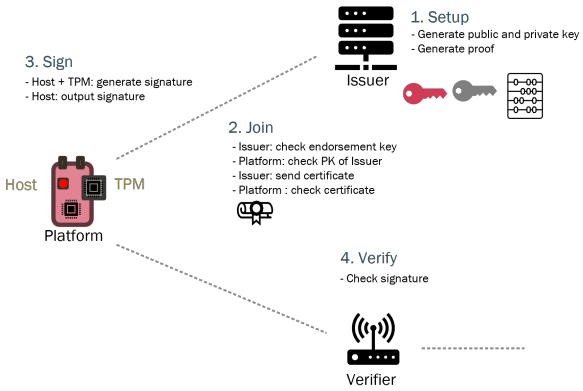


Figure 4.2: Direct Anonymous Attestation roughly sketched out

## 4.3 Direct Anonymous Attestation

DAA as previously explained has four actors, the TPM and its hosting platform, the issuer-server and the authentication server. DAA as described in the initial work by Brickell et al.[3] and other authors [6], [26] begins with the process of setting up the issuer. As described in [3], [26] This step involves generating the asymmetric key-pair for the issuer and subsequently publishing the public key and its public parameters.

Subsequently, the join operation can be executed by following the steps below.

1. The TPM generate its attestation key pair (AIK) and sends that to the host [26].
2. The host forwards the public parameters of the AIK as well as the public endorsement key (EK) and its EK certificate to the issuer [26].
3. The issuer then checks if the certificate matches with the public EK [3], [26] before it challenges the platform to confirm that the TPM can be trusted.
4. The Host and the TPM module bot take part in the computation to respond to the challenge by the issuer [3], [26] After this step, the issuer trusts that the EK and AIK both come from the same TPM and that the platform can be trusted.
5. To complete the join the issuer creates an attestation key credential and sends that to the platform [26]. Finally, The platform and the TPM follow similar steps as they did to respond to the challenge to activate the credential [26].

The next operation in the DAA scheme is ”sign”. Since the platform after completing the join operation now has its credentials it can use it to attest that the messages that are output by this platform were all computed by the same device as explained in [26].

On the receiving end the verifier will need to ”verify” the signed message by computing various sets of comparisons: in the newer implementation in [26] this contains the attestation key credential, the signature, the attestation key itself and the state of the platform

that sends the message using its PCR-values. The PCR-values are platform configuration registers only modifiable by concatenating the existing value of that register and then hashing this concatenation which creates a history of hashes and also is a way to counter physical tampering [27]. With this step, the DAA scheme extends into the EAP, completing the process.

## 4.4 Physically Unclonable Function

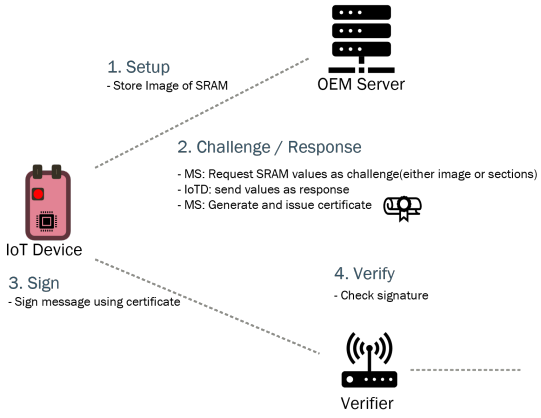


Figure 4.3: Attestation with a SRAM PUF

The other potential method to attest trust in an IoT device is the more hardware-based method of PUFs. In the context of CERTIFY, the prototype discussed has an SRAM. Therefore, the design will also be based on SRAM-PUFs. Another point arguing for using an SRAM-based PUF is that many embedded devices are also equipped with SRAM since this memory type has lower power consumption and does not need to constantly refresh its memory cells to store its data [28], [29].

As proposed in [28] the first step to set up an authentication scheme with SRAM-PUF is to read out the SRAM values while the device is in its startup sequence several times to map out the stability of the SRAM cells. This information is saved on an authentication server after reducing the noise of the data with error correction algorithms [28].

Using this way to read out SRAM however implies that the authentication entity knows the identity of the device since it has to store a challenge-response table for every peer trying to access the network. Instead, the Idea is to generate a key pair as proposed in [4] with the difference that the SRAM-PUF is used to attest the device identity instead of an EK stored separately on a TPM. As shown in Figure 4.3 The remaining operations such as signing and verifying could potentially be implemented similarly as they were in DAA, but since there are few resources this would need to be tested first.

## 4.5 Big Picture and Implementation Decision

Combining the attestation methods with EAP constructs two potential authentication designs that could be pursued further.

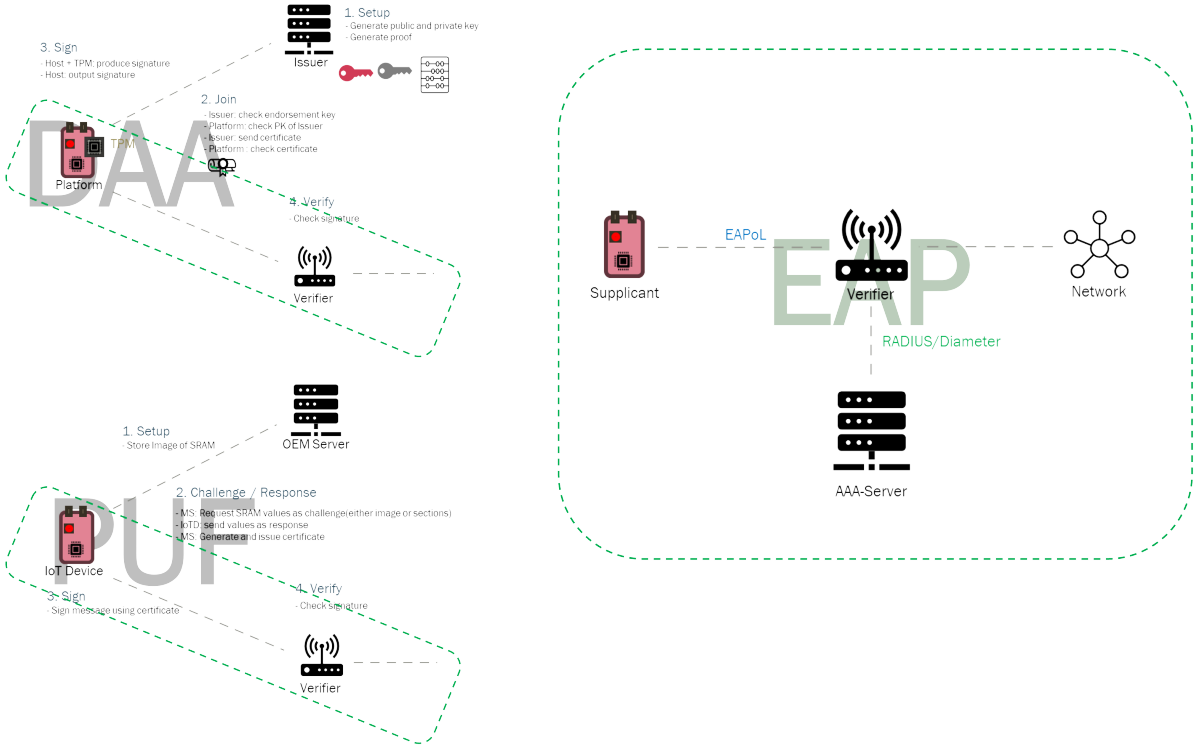


Figure 4.4: All components and how they can be combined

Before moving to the next chapter which will focus on hardware decisions and implementation steps, it is important to note that the initial plan to implement both authentication constructions had to be halted to narrow the scope of this thesis. Instead, the decision was made to implement DAA according to the planned models in the CERTIFY project [2], [30].

# Chapter 5

# Implementation

The first idea in this chapter was to decide on a suitable authentication method. After the choice of attestation method, the idea was to start the authentication process in a very simple setup and then extend from there. But as previously mentioned the initial idea was scratched to focus on one implementation alone since the scope would have been too broad.

Despite this change discussing the hardware remains valuable nonetheless. As mentioned in the introduction this thesis was defined within the CERTIFY project therefore in earlier stages hardware decisions were influenced from the beginning. The first section will discuss the hardware proposed during prototype discussions within the project. This will be followed by introducing what hardware was available in the next section. The last section will cover the concrete implementation.

## 5.1 Hardware used in CERTIFY

Reviewing documentation of the project revealed that CERTIFY will use the B-L462E-CELL1 Cellular IoT Discovery kit (see Figure 5.1 ) from ST-Microelectronics [31] [2], [32] as their nodes. These nodes will consist of:

- Secure MCU STM32L462RE [33]
- LTE Cat M/NBIoT modem (LBAD0XX1SC-DM) [34]
- ST4SIM-200M GSMA-certified embedded SIM (usable as a secure element) [33]
- STMod+ for extension capabilities [31]
- connectivity USB + LTE, UART and I2C

The STM32L462RE [35] microcontroller is equipped with 160 K bytes of SRAM. The embedded Sim acts in this context as the secure element of the node. It implements

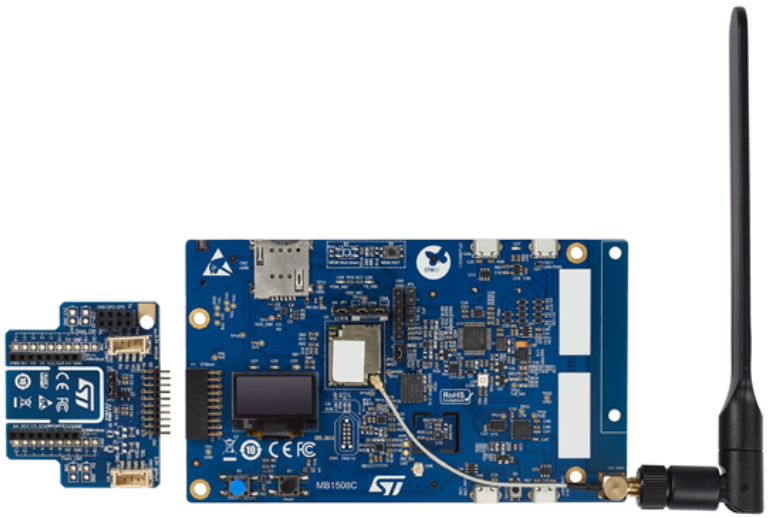

Figure 5.1: The B-L462E-CELL1

Asymmetric RSA up to 2018 bits and secure element access control [33] these properties fulfil the hardware requirements to implement DAA on one hand and at the same time SRAM- based attestation would also be possible.

## 5.2 Hardware scope of the thesis

In the scope of the thesis the hardware used will consist of different boards than the ones shown before, one of the reasons for this was that the boards are available and therefore there is less delay in setting up a working test bed.

### 5.2.1 Raspberry PI 3 Model B V1.2

[h] The Raspberry PI 3 Model B (Figure 5.2) is described as a single-board computer, it is capable of connecting to a LAN either using an Ethernet cable or wireless. It can also communicate using Bluetooth. Furthermore, its specification found in [36] describes among other things four USB 2 ports, 1GB RAM, a Quad Core 1.2 GHz Broadcom BCM2837 64-bit CPU and a HDMI port in addition to being powered by Micro USB

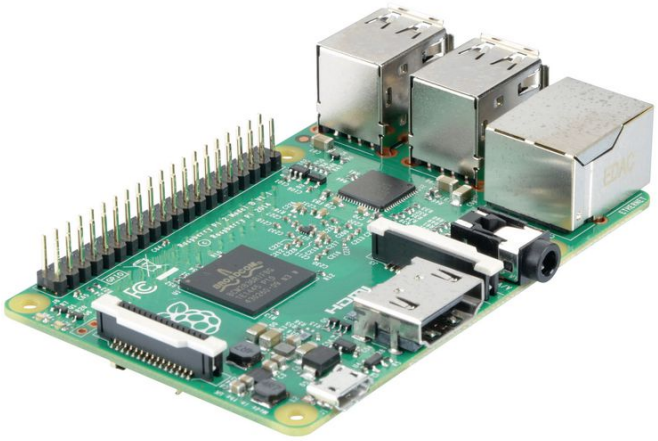

Figure 5.2: Raspberry PI 3

power source with up to 2.5A. The Raspberry PI also provides a micro SD port where the operating system is loaded and also to be able to store information.

Since the RAM is also included in the Broadcom chip it is not directly visible what type it is. The documentation on the raspberry PI [37] leads to a document mentioning the previous model including synchronous dynamic RAM (SDRAM). This is a type of dynamic RAM that is synchronized with a computer's internal clock as described in [38].

The PUF taken into consideration is meant for SRAM, the Raspberry PI cannot be used for this type of attestation to simulate an IoT device. Instead, the board would be suitable to be set up as an authenticator for EAP.

The device is suitable to be programmed to do authentication using DAA, once it is equipped with a TPM to perform actions to gain a certificate and calculate its cryptographic credentials.

For setup, the Raspberry PI Imager is installed on a computer with an SD card reader [39]. The application then can be used to select the SD card that is in the card reader and to write the officially supported Raspberry Pi OS on it. Alternatively, the OS can be set up manually, both versions are available in [39]. The OS formerly known as Raspbian is based on Debian version 11, so it has a Linux kernel.

### 5.2.2 LetsTrust TPM

[h] The LetsTrust TPM [40], shown in Figure 5.3, that will be used in this work is compatible with all Raspberry PI boards. This component enables the Raspberry PI to authenticate using DAA since it is a core component of the attestation process.

For PUFs, it could be used for additional computations where randomness is a topic or simply as an aid.

Figure 5.3: LetsTrust TPM

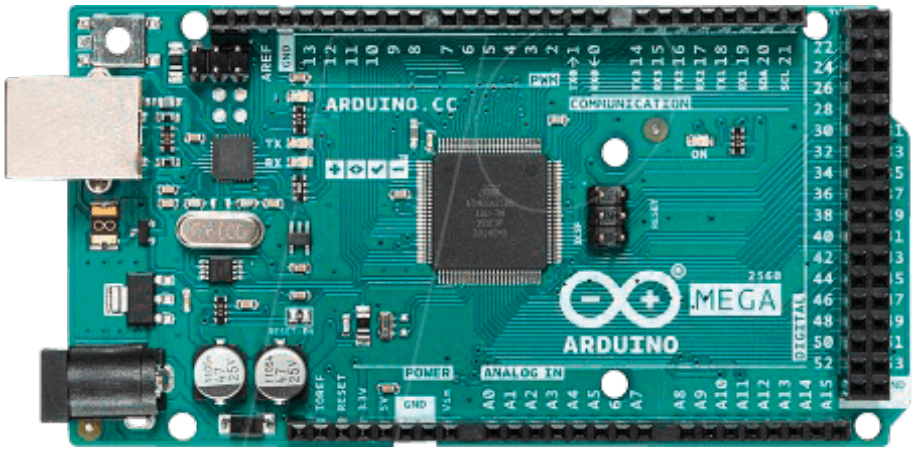


Figure 5.4: Arduino Mega

The module arrives with a female header help to install it correctly onto the target board's GP-IO header. Included are also Instructions for activation of the TPM on the Raspberry PI and links to the necessary software. The TPM is compliant with the TPM2.0 specifications which are needed to implement DAA as shown in [26].

The Homepage of LetsTrust [40] also provides links to software packages on GitHub for the TPM to use. How the chip is set up in more detail will be explained in the following section where the implementation of the prototype will be discussed.

### 5.2.3 Arduino MEGA 2560

[h] The Arduino Mega shown in figure 5.4 is a microcontroller board with 8KB of SRAM and a USB B Slot for connection and power. The device can also be powered separately by a power jack. It also provides pins for several inputs and outputs as described in [41].

Figure 5.5: ESP32

Since this device uses SRAM it is a candidate to authenticate using SRAM PUFs. But it does not have connectivity for accessing LAN so this device cannot be considered.

Arduino on their page [41] also provides different possibilities to program the Device, in this thesis the Arduino IDE is chosen to program the device.

### 5.2.4 ESP32-DevKitC

[h] The Espressif ESP32 Board houses the ESP32-WROOM-32D MCU which unlike the Arduino board can communicate using Wi-Fi or Bluetooth [42]. It is equipped with 520 KB of SRAM for data and instruction storage as well as 8 KB of SRAM in the real-time clock (RTC) [43]. The SRAM in RTC can be accessed during RTC boot which the datasheet [43] describes as RTC FAST. Additionally, the ESP32 has 8 KB of SRAM in the RTC which can be accessed while the module is in sleep mode called RTC SLOW [43].

These specifications make the board suitable for SRAM-PUFs where for example the 8 KB RAM of the RTC FAST can be read out at boot. But since the choice was made to implement DAA instead of PUF this board also was not used.

## 5.3 Implementation of a Prototype based on the DAA design

The prototype for a DAA-based authentication process has three components. One component is the PC acting as a server, particularly as a credential issuer. The other two components are the Raspberry PI acting as the host and the LetsTrust TPM installed on the board as shown in Figure 5.8. The goal is to implement the DAA as it was described in Chapter 4 by following the implementation made in [26]. First, all components need to be prepared so that they can be programmed. The first component to be addressed will be the issuer. Once all the details are discussed concerning the issuer the next step will explain how the Raspberry PI and TPM need to be configured.

The entire preparation setup will be summarized in the installation guidelines.

### 5.3.1 Issuer setup

For the initial setup, all actors are within the same network. Since the computer needs to act as a server, there are some steps necessary to allow the computer to communicate with other devices. Since the implementation was done using Python the libraries ”http.server” [44] and ”socketserver” [45] were available options program a bare bones server.

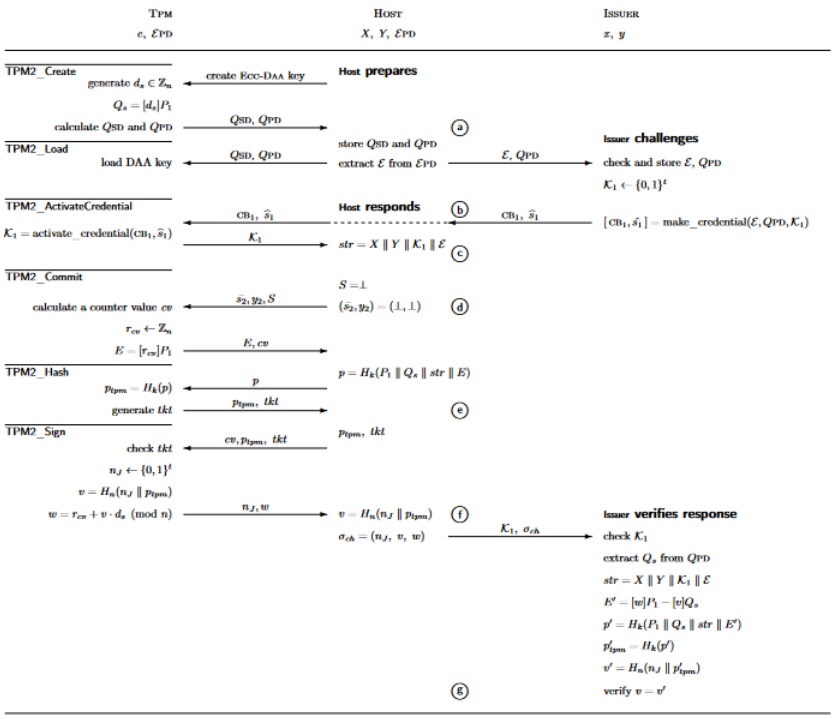


Figure 5.6: Beginning of the Join Process as shown in [26]

The PC is set up to become a host using its private IPv4 address for incoming requests and designating a specific port which in this example is port 8000 to send and receive data. An important aspect to keep in mind here is to enable the port to forward incoming and outgoing messages on the firewall. On Windows, it is done by defining incoming and outgoing rules which allows the connection. Otherwise, the platform will not be able to send requests.

DAA mostly starts with the setup of the issuer. The personal computer should first generate its key pair before it can publish its credentials following [26]. To achieve that the Python library pycryptodome [46] was used. The package can be installed by running *pip install pycryptodome* in the console.

This package provides low-level authentication mechanisms and it supports elliptic curve cryptography. Using this package the project is set up to generate its key pair based on ECC in PEM (Privacy Enhanced Mail) format if none has been generated yet. Otherwise, it should fetch from the existing files the one storing the public key and forward that to the Raspberry PI 3 if it requests the keys via GET.

The issuer in its current state mostly is comprised of REST response methods. However, key and certificate files sent by the platform arrive in MIME format. The MIME structure as described in [47] also known as Multipurpose Internet Mail Extensions is a type which indicates information about the document such as its type, its format, the document name and other properties. To read the key or certificate from a MIME form the ”email.parser” [48] needed to be included. The issuer first stores the raw MIME data from the request body in a temporary file. This file is passed as an argument to the helper function below:

```
def _save_files_from_mime(self, path):
```

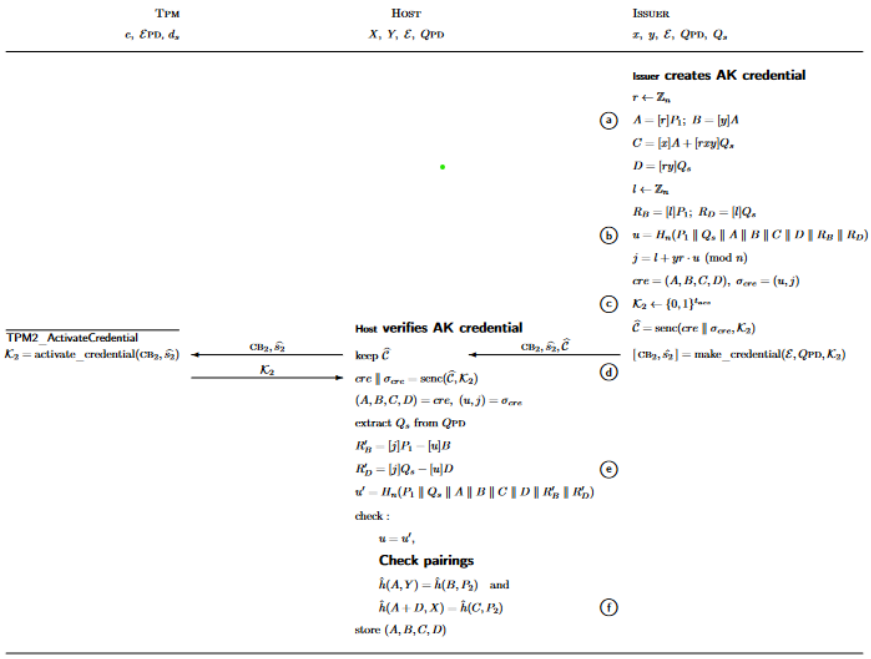


Figure 5.7: Finsihing of the Join Process as shown in [26]

```
        with open(path) as fp:
            msg = parser.Parser().parse(fp)
            check_mp = msg.is_multipart()
            sections = msg.walk()
            for part in sections:

                if part.get_content_type() == "application/octet-stream" or
                    part.get_content_type() == "base64":
                    filename = str(part.get_filename())
                    pot_key = part.get_payload()
                    key = open("files/" + filename, "w")
                    key.write(pot_key)
                    key.close()
        os.remove(path)
```

What this method does is use the temporary file created previously to convert it into an EmailMessage Object [48] and use this object to iterate over the entire content until the field containing the key or certificate is found. From this field, the filename can be extracted and using *get_payload()* will extract the key string.

Once the public EK key and the EK certificate were stored, the certificate was decoded to compare it against the public endorsement key to satisfy the first action from the issuer side in the join process in [26]. The following methods were used to perform the comparison:

```
def get_pk_from_cert(path):
    certfile = open(path, "rb")
    cert = certfile.read()
    der = base64.b64decode(cert)
```

```
    seq = DerSequence()
    seq.decode(der)
    subseq = DerSequence()
    subseq.decode(seq[0])
    subject_PK_info = subseq[6]
    key = RSA.import_key(subject_PK_info)
    certfile.close()
    return key

def get_pk_from_pem(path):
    pub_file = open(path, "r")
    puk = pub_file.read()
    pem_key = RSA.import_key(puk)
    pub_file.close()
    return pem_key

def validate_EK(key_path, cert_path):
    ek_key = get_pk_from_pem(key_path)
    cert_key = get_pk_from_cert(cert_path)
    return ek_key == cert_key
```

the first helper method extracts the public key information from the certificate as described in [49] and creates an RSA Key which is returned. A future solution should apply a more sophisticated method to get the public key information. The second helper method loads the public EK key into an RSA Key Object and returns the key. Finally, the EK is validated by comparing both keys. As explained in more detail below the implementation of the stops at this stage because of time constraints.

### 5.3.2 Platform

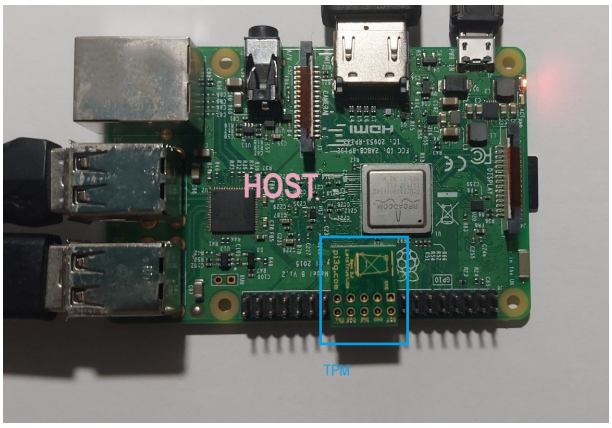


Figure 5.8: The Platform: Raspberry PI with a TPM mounted

In this part, the goal is to explain all actions that need to happen such that the Raspberry PI can operate as a platform in a DAA authentication setting. Additionally, the EK, EK certificate and AIK are generated from files and sent. The necessary commands are explained in more detail to give context and a basis for future implementations.

**Preparation**

Before the Raspberry PI 3 can be used to function as a trusted device it has to be set up. As previously mentioned the Raspberry PI Operating System needs to be loaded onto an SD-card.

This is done using an SD card reader. If no reader is built into the PC an external one can be purchased which connects via USB ports. From the official website [39] the application Raspberry Pi Imager is downloaded depending on the OS that it will be installed on.

In the current case, the Windows version will be used. From the variety of options that will be listed in the Imager, this device will be using the recommended Raspberry Pi OS. The second step is to choose the SD card that the image should be written on. After completing the writing the SD card can be inserted into the designated slot and the board is ready to boot.

The next step will be to extend the board with the LetsTrustTPM. According to the instructions in [40], the TPM is installed by first inserting the female helper header into the GP-IO header starting from pin 1 on the outermost corner. Directly next to the female header, the TPM is placed facing towards the HDMI port and the chip facing downwards. The female header can be removed afterwards. In the end, the extension of the Raspberry PI should look like figure 5.8.

Before starting the preparation, the swap file of the Raspberry PI should be adjusted so that the system can run smoothly [50]. This should be done before anything else so nothing is lost if this procedure should fail. Steps according to [50] consist of:

1. Turn off swapping with *sudo swapoff -a*
2. Use *sudo dd if=/dev/zero of=/swapfile bs=1G count=2* to adjust the size to 2 Gigabytes and save the change into a new file.
3. Change the permissions of the generated swap file: *sudo chmod 600 /swapfile*
4. Enable the file to be used as swap: *sudo mkswap /swapfile*
5. Activate the swap file using *sudo swapon /swapfile*
6. If the file is not available on /etc/fstab it can be added with */swapfile none swap sw 0 0*
7. To check the swap memory after all steps were taken *grep SwapTotal /proc/meminfo* can be used.

After completing the OS's initial setup, the board can be configured to use the TPM. The instructions [40] first require the user to update the boot configuration of the Raspberry Pi. This can be accomplished by executing the following commands in the terminal:

1. To have the newest running packages it is necessary to run the commands *sudo apt-get update && sudo apt-get upgrade*

2. Then we access the boot configuration by executing *sudo nano /boot/config.txt*.
3. After the file is open in the terminal the instructions state that *dtparam=spi=on* and *dtoverlay=tpm-slb9670* need to be added to the file at the bottom. After doing this the file can be saved and closed.
4. Reboot the board using *sudo reboot*.

After completing all steps, the folder dev should contain a file named ”tpm0”. Before the TPM can be used, however, the corresponding software needs to be added first. To accomplish the installation as smoothly as possible according to the descriptions in [51], [52] the repositories Tpm2-tss, Tpm2-abrmd and Tpm2-tools will be installed in the order listed. Before starting the packages general dependencies used by all libraries need to be installed or updated if they already exist. As listed in [51]–[53] the general dependencies are:

- GNU Autoconf Archive (minimum version 2019.01.06)
- GNU Automake
- GNU Libtool
- C compiler
- C library development libraries and header files
- pkg-config
- doxygen
- OpenSSL development libraries and header files (minimum version 1.1.0)
- libcurl development libraries
- Access Control List utility (acl)
- JSON C Development library
- Package libusb-1.0-0-dev

Another dependency for Tpm2-abrmd is the GLib library of C routines [53]. The terminal command to install these libraries as per [52] is as follows:

*sudo apt -y install autoconf-archive libcmocka0 libcmocka-dev procps iproute2 build-essential git pkg-config gcc libtool automake libssl-dev uthash-dev autoconf doxygen libjson-c-dev libini-config-dev libcurl4-openssl-dev uuid-dev libltdl-dev libusb-1.0-0-dev libftdi-dev libglib2.0-dev*

Continuing from the dependencies the source code of the TCG Software Stack [54], Tpm2-tss, can now be added next. The software stack contains five layers.

- Starting from the top there is the Feature API (FAPI) for simplified usage of a TPM.
- The Enhanced System API (ESAPI) provides one-to-one mapping and asynchronous calls to the TPM2 [54]. The ESAPI additionally provides functionality for session handling and tracking of metadata for TPM objects.[54]
- The System API (SAPI) [54] only provides one-to-one mapping and asynchronous calls to the TPM2 commands
- The MarshallingUnmarshalling (MU) layer handles the different data types [54]
- The last layer is the TPM Command Transmission Interface (TCTI) transmits and receives TPM commands and their responses [54].

These libraries will potentially become interesting in developing the prototype in the future, however currently the Tpm2-tss package is only a mandatory dependency for Tpm2-abrmd and Tpm2-tools as specified in [51], [53].

The installation of Tpm2-tss by terminal is done as explained in [51]:

1. Download the repository from git: *git clone https://github.com/tpm2-software/tpm2-tss.git*
2. Change into the directory using *cd tpm2-tss*
3. Execute *./bootstrap* to generate a list of source files and the configure script.
4. Execute *./configure –prefix=/usr* to generate the makefiles.
5. The generated makefiles are compiled using *make -j5*
6. The installation is completed by running the command *sudo make install*.

Although Tpm2-abrmd is optional the installation guide recommends adding this library as well [51]. Therefore following the instructions [51], [53] leaves the following installation procedure:

1. The First step consists of adding a system user named tss as it is recommended in [53] by executing in the terminal command *sudo useradd –system –user-group tss*.
2. After adding the system user the installation process works very similar to the process for Tpm2-tss by downloading the repository using *git clone https://github.com/tpm2-software/tpm2-abrmd.git*.
3. The procedure continues by changing into the directory using *cd tpm2-abrmd* and then running *./bootstrap*.
4. Once bootstrapping generated the files the configuration can be started by executing *./configure –with-dbuspolicydir=/etc/dbus-1/system.d –with-udevrulesdir=/usr/lib/udev/rules.*
   *–with-systemdsystemunitdir=/usr/lib/systemd/system –libdir=/usr/lib64 –prefix=/usr*.

5. The installation is finalized by first compiling with *make -j5* and then executing *sudo make install*

Now that the dependencies are installed the main library used for the prototype, Tpm2-tools, can be added. The process is almost identical to the previous ones and is also described in [51]:

1. As for the libraries before the process starts with *git clone https://github.com/tpm2-software/tpm2-tools.git*.
2. After moving into the directory with *cd tpm2-tools*, *./bootstrap* and *./configure –prefix=/usr* are executed subsequently.
3. Running *make -j5* and *sudo make install* complete this installation as well.

A successful installation can be tested by executing the command *tpm2 getrandom 8* as demonstrated in [51]. The command returns eight random hex-encoded bytes to the output stream. Raspberry PI can now be programmed to operate within the DAA scheme.

### Key Creation and Certificate Preparation

The implementation continues to follow closely the instructions given in [26] and the implementation example in [27]. A distinct advantage in [26] is its recency and the fact that the TPM commands needed are explicitly stated for an implementation of a DAA scheme based on elliptic curves.

A precondition before the device can start the join process is the existence of a primary key hierarchy as stated in [26], [51] specifically an endorsement key hierarchy. The Endorsement key (EK) used is generated with RSA algorithm [26]. Only after generating the EK an attestation identity key (AIK) can be created with the EK as its parent [51]. The prototype uses RSA for the EK following the example in[26].

```
tpm2 createek -G rsa -c daa/rsa_ek.ctx -f pem  -u daa/ekpublic.pem
tpm2 createak -C daa/rsa_ek.ctx -c daa/ecc_ak.ctx -G ecc -g sha256 -s ecdaa
    -f pem -u daa/ecc_ak.pem
    -r daa/ecc_ak.priv -n daa/ecc_ak.name
```

The code snippet above shows how first the EK is created followed by the AIK. First, the EK is generated with the tpm2_createek by setting RSA as the key algorithm and saving the EK context object to a file named rsa_ek.ctx and additionally outputting the public key part in PEM format [27], [51]. The EK can also be generated by using ECC starting with the TCG specification version 2.0 for the TPM [9], initially only RSA 2048-bit keys were supported. After the creation of the EK, the AIK creation is executed by calling tpm2_createak. In the code shown above the first argument is used to link the AIK to the EK context object file generated before, followed by specifying again the path to save the key context as specified in [51]. After choosing the key generation algorithm for the

AIK, in this case, ECC the next arguments set the choices for the hash algorithm with -g sha256 and the signing algorithm with -s ecdaa as explained in [51].

This implementation uses the default curve NIST_P256 but if a TPM supports another curve it can be chosen instead by specifying the curve in a more detailed manner e.g. using "ecc384" to base the ECC key generation on the NIST_P384 as it is shown in [51]. This becomes relevant for cases where the security of the standard curve is questioned as it was mentioned in [26].

To check among other properties which elliptic curves are integrated, Tpm2-tools provides the command tpm2_getcap which in combination with one of the arguments in [51] will print the properties of this TPM, specifically *tpm2 getcap ecc-curves* provides information about all supported ECC-curves. The LetsTrust TPM used for this project supports the nist-p256 and bn-p256 curves for ECC key generation.

Another detail to note is that for AIKs, instead of context object files it is possible to save the object on a vacant handle, a reference to a memory space, directly on the TPM by applying the context argument -c - [51].

```
RSA_EK_CERT_NV_INDEX=0x01c00002
NV_SIZE=`tpm2_nvreadpublic $RSA_EK_CERT_NV_INDEX | grep size | awk '{print␣
    $2}'`
echo $NV_SIZE
tpm2 nvread -C o -s $NV_SIZE -o daa/rsa_ek_cert.bin $RSA_EK_CERT_NV_INDEX
```

After the keys were generated the next step was to prepare the EK certificate to send it to the issuer as shown in [26]. As explained in [27] the certificate is either read from non-volatile memory by executing the commands above or it is provided by the manufacturer for download.

The code snippet above first defines a variable storing the handle where the RSA certificate for the EK is stored, the location is also specified by the TCG [9], [27]. In [27] the next step is to use the handle in combination with tpm2_readpublic to get the metadata of the handle [51], then search for the pattern size with grep in the result followed by storing the value after the keyword size using the output of awk 'print $2'. The size value is needed for tpm2_nvread. As shown above as well as in [27] tpm2_nvread first is authorised against the owner hierarchy using *-C o* then the parameter *-s $NV_SIZE* defines how many bytes need to be read, *-o* and the handle *$RSA_EK_CERT_NV_INDEX* is used to point to the start of the section that shall be read [51]. The result is a file containing the EK certificate.

Alternatively following the instructions in [51] allows the certificate file to be generated by applying the command tpm2_getekcertificate by only passing the parameter -o followed by the filename. The documentation [51] further explains the first -o will output the RSA certificate and applying the parameter -o a second time also outputs the EK certificate for ECC-based keys. The implementation reading out non-volatile memory is mostly useful if only the certificate needed shall be output, for this reason, the prototype also applies the previous method.

```
tpm2 getekcertificate -o daa/rsa_ek_cert.bin
```

Before the certificate is usable, however, [27] first manipulates the content of the rsa_ek_cert.bin file to change its encoding to Base64. A precondition to achieve this is installing with *sudo apt-get install -y cl-base64* cl-base64 on the Raspberry PI so the encoding can be altered since this library is missing on the current distro. Based on [27] the code snippet below shows how these changes are done.

```
sed 's/-/+/g;s/_/\//g;s/%3D/=/g;s/^{.*certificate":"//g;s/"}$//g' daa/
    rsa_ek_cert.bin
base64 daa/rsa_ek_cert.bin > daa/rsa_ek_cert_b64.bin
```

The command *sed* as shown above combines five regex pattern replacements divided by semicolons applied to the recently generated certificate file. The following characters are replaced:

- "-" is replaced by "+"
- "_" is replaced by /
- "%3D" is replaced by "="
- In the line starting with an opening bracket everything before and the word certificate including the colon is removed
- the closing bracket at the end of the line is removed

These replacements are important to allow changing the encoding of the certificate to base64. Base64 only allows the Symbols "A-Z", "a-z", "0-9", + and "/" as explained in [55] which is a potential source for problems if undefined characters are not removed before encoding 64file to Base. With this step complete, all files can be sent to the issuer.

### Communication to the Issuer

The platform communication is implemented in C/C++ build and compilation handling using cmake in Visual Studio Code [56] based mainly on the library variant of cURL LibcURL which is already installed due to it being a required dependency for the TPM Software. Visual Studio Code is not installed from the start, by downloading and running the deb file from [57]. Following this, the C, C++ and Cmake extensions can be installed directly via the extensions window as shown in [56].

Sending the files to the server is implemented as a POST request as visible below in the postEKCertificate method used as an example. A similar method was created to send both the public EK and AIK. The implementation is based on the example called postit2.c [58] found on the API documentation of LibcURL [59]. According to the implementation shown in the example files are sent using the MIME structure. The example in [58] and the implementation below send the file as multipart/form-data.

The method begins by declaring and initialising all relevant variables. This is followed by building the MIME structure using the curl_mime commands and sending the POST

request containing the HTML form as detailed in [58] by configuring the request using curl_easy_setopt with the respective parameters. Certain points need to be considered during this step. On one hand, before the file can be sent, its access rights need to be set such that the file can be opened to read from the input stream. Access rights can be changed by executing the command *sudo chown <user>:<user> /path/to/file* while replacing the <user> with the owner and adding the file path. Additionally, cURL is a C library so any string needs be be either defined as a char array or if C++ strings are used, the value needs to be converted to a C-style string using c_str() as shown below.

The Platform code also contains some helper functions for faster debugging. Such as a Helper function printing file content to verify that the communication process was sending correctly. Otherwise, the functionality of the methods is mostly here to handle communication over LAN.

```
bool postEKCertificate(){
            bool retval = false;
            CURL *curl;
            CURLcode res;
            curl_mime *form = NULL;
            curl_mimepart *field = NULL;
            struct curl_slist *headerlist = NULL;
            static const char buf[] = "Expect:";
            std::string url = "http://192.168.178.26:8000/joincert";
            char sourceParams[FILENAME_MAX] = "/home/jkosta/daa/
                rsa_ek_cert_b64.bin";

            curl = curl_easy_init();
    if(curl){
            form = curl_mime_init(curl);

            // Add file for EK Certificate here
            field = curl_mime_addpart(form);
            curl_mime_name(field, "sendfile");
            curl_mime_filedata(field, sourceParams);

            // Add filename for  EK here
            field = curl_mime_addpart(form);
            curl_mime_name(field, "filename");
            curl_mime_data(field, "rsa_ek_cert_b64.bin",
                CURL_ZERO_TERMINATED);
            // Add filename for  EK Certificate here

            static const char buf[] = "Expect:";
            headerlist = curl_slist_append(headerlist, buf);
            curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
            curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headerlist);
            curl_easy_setopt(curl, CURLOPT_MIMEPOST, form);
            curl_easy_setopt(curl, CURLOPT_VERBOSE, 1L);


            res = curl_easy_perform(curl);
            if(!res){
                        retval = true;
                    }
```

```
            curl_easy_cleanup(curl);
            curl_mime_free(form);
            curl_slist_free_all(headerlist);
        }

        return retval;
    }
```

The implementation of the prototype had to stop at this point. While the implementation is still at the beginning of the join process, continuing this prototype with the remaining time would prove infeasible. The next implementation step is at the issuer is to create a challenge for the device. This step, however, was not described completely in [26] and left open investigations with would have broken the time frame of this project. In general, the implementation is tied to extensive investigations since many different cryptographic elements need to be applied to progress.

# Chapter 6

# Evaluation

In this chapter, the discussion section will evaluate the differences, advantages and disadvantages between the different authentication methods. Based on the hardware and architecture decisions of CERTIFY, this thesis specifically also follows suit to progress DAA instead of PUFs. Therefore this section evaluates DAA and PUF detached from the decision that was made. The succeeding section will go over the challenges that were present in this work.

## 6.1 Discussion on DAA and PUF

From the start what became apparent while reviewing the literature was how different the questions asked were when developing the attestation methods. For DAA the progression focuses heavily on reducing computational costs on the platform side while maintaining security, as can be observed in the related work. DAA as visible in [3] and progressing works such as [18] often involves applied number theory used in cryptography, leading to validation needing exact matches. One example of this is the verification result of the EK certificate having to match the public EK. On one hand, this eliminates the need for thresholds as in [10] but on the other hand, it opens questions about key storage and requires extensive mechanisms to establish trust as seen in [26].

As previously established DAA needs additional external hardware such as the TPM to establish trust. Depending on which TPM is used a different number of RSA and ECC algorithms are supported. Before extending the platform with a TPM one Question asked concerns the compatibility of an edge device with the TPM. Another criterion could also be which cryptographic algorithms are integrated into the module. What was not always clearly mentioned in the research papers is the fact that elliptic curves are usually already defined and integrated on the TPM, so calculating the values is not necessary. Another risk that has to be accounted for is the secret storage itself as it was also previously mentioned in [4] where it is mentioned that after a while storing information in nonvolatile memory could leave afterimages compromising the secret.

The Research surrounding PUFs is more oriented towards finding new potential circuitry that could be the basis for a PUF or further develop existing ones as it can be interpreted from the related work. Since this solution is closely tied to the hardware many questions asked are regarding the stability of the PUF depending on what temperature the devices are exposed to or how the use and progressing age of the boards influence the reliability of a PUF as it can be seen in [4]. One important aspect contrasting DAA here is that device identity can be attested using simple challenge-response mechanisms with error correction algorithms shown in [4], [28]. This reduces the computational workload on the side of the edge device considerably and potentially allows the omission of the TPM. However, one potential pitfall of PUF implementations is as previously implied their probabilistic nature. Although in [28] the authentication is considered stable, there is little information about testing authentication in a more large-scale setting where the PUF is tested in regular but intense intervals to determine how the authentication holds for a large number of devices. Performing multiple attestations in a larger experiment could be beneficial to review if false negatives or positives for attestation occur and what the consequences are if a device is wrongfully not allowed to authenticate.

Niya et al. [12] state that a risk that can occur with PUF using SRAM is if a user has physical access to a device that is authenticated using SRAM. The authors state that a person can read out the entire RAM and submit the correct section while registering a malicious device. As per the example mentioned by Niya et al. [12] a malicious user can implement a simple function with the Arduino IDE by reading the values, which can simply be done by using a pointer, setting its address to the start and then iterating over the entire memory space. Another drawback of an SRAM-PUF is pointed out in [28] where the challenge occurs at the start-up of the device, so the SRAM-PUF can only be executed during the booting phase of the platform. Additionally, the discussion in [28] mentions that to be able to join a network during runtime the platform would need to store its memory in a designated memory space.

Although the choice to implement DAA was more feasible for the context of this thesis, this preference does not apply to every situation especially if the anonymity of a platform is not in the foreground. Excluding the requirement that an attestation has to occur anonymously, EAP could be extended to authenticate devices similarly as it was proposed in [28].

## 6.2 Challenges of this Thesis

The progression of this thesis faced a variety of challenges. It starts with the amount and density of knowledge needed to understand the topic and form an understanding of the procedure. The information density and variety were most noticeable while configuring the device to communicate with its TPM. Before the software for the TPM could be used it had to be configured. To install all packages while Navigating the documentation's installation steps is tricky. On one hand, what needs to be done is summarized at the bottom in [51]. On the other hand, the summary at the bottom is written with the assumption that all dependencies already were satisfied. Only after through the pages of the accompanying packages, the process became more clear. The complexity was only highlighted more when

it came to applying the TPM commands. Understanding the parameters and arguments of the specific TPM commands are based on specifications according to the TCG such as [9], [54], however, these documents were written to accommodate developers with a certain high level of knowledge on the topic of cryptography.

Interpreting both DAA and PUF required learning the principles of the sources quite intensively. DAA implementations were mostly described in such a detailed manner that it was broken down into the respective algorithms as best seen in [3], [6]. Again much Information was given under the assumption that certain cryptographic structures were clear such as the EK and AIK hierarchy. Although PUFs in contrast are not coupled to extensive cryptographic primitives the complexity starts with the requirements of the hardware depending on which PUFs are executable. Another difficulty with PUFs was also mostly the focus on how a specific function is performed and how good it holds. However, there are sparsely any resources implementing PUFs to authenticate against a verifier while maintaining their anonymity.

During the implementation of the attestation scheme, certain challenges need to be taken into account. First of all, before the Raspberry PI could programmed its swap file needed to be increased. In Linux, swap files use a defined amount of memory space on the SD card as additional working memory. The lack of swap memory first caused the board to freeze indefinitely or react after a long delay. One mistake that should never be made in this case is forcefully turning off the power. Doing this poses a risk of damaging the SD card to such an extent that not even reformatting was possible. This happened for the first configuration of the Raspberry PI. Implementing the attestation scheme based on DAA also showcased certain aspects that were not highlighted in sources like [3], [26]. The first aspect concerns the assumption that the EK and its corresponding certificate are a given. Another aspect was certain implementation steps were described in text only. Sometimes the the information needed to complete the step was not always clear. Specific topics that were challenging while programming the board stem from handling the files generated by the TPM. These topics are the inconsistent file endings in [51] where context files are denoted with the endings ".handle", ".out", and ".ctx" interchangeably between the different key creation functions.

It is also important to know that if a file has restricted access rights libcurl might not be able to open it to read its content for POST requests. The process will fail during the phase where the program is trying to open the file to import its content.

From the sources regarding DAA, some vital specifics are also not quite clear like which encoding to use to send keys. The certificate output by the TPM used in this work was encoded in such a manner, that some sequences of its string need to be replaced such that an encoding in DER format is possible. Additionally, while the TPM2 Tools can output the keys in PEM or DER format, the same thing is not possible for the certificate corresponding to the Endorsement key. Trying to send the certificate without first encoding causes libcurl to send an empty string due the the read function not recognizing some byte sequences. Also to complete multiple subsequent requests the server side has to immediately send a response header before handling the input received via POST request otherwise the process gets stuck and subsequent requests do not pass through.

# Chapter 7

# Future Work Summary and Conclusions

This chapter concludes this Thesis and will go into topics like potential future research, summarize and conclude this work by addressing the research questions at the beginning. Since the potential future work is quite extensive, it will be covered separately in its section.

## 7.1 Future Work

Due to this thesis being written over a broad range of topics and the context given by CERTIFY still being mostly described on paper while designing both attestation methods, it was decided to focus on one implementation only. Since the prototype was implemented up to the point where the issuer has to check that the AIK (ECC-DAA key) has properties as specified in [26]. The steps left to complete the process are described in [26]. Potentially if available their implementation could be studied in more detail and integrated to complete the remaining join protocol. Future implementations may adapt the communication to respect transfer protocols and security aspects.

Tied to the previous points, many arguments needed to operate the TPM could be implemented to be more accessible. One idea would be to consolidate installation steps and subsequent operations by creating an interface which allows the setup of arguments like for example the preferred key generation methods based on the TPM's capabilities in a readable and more automated format. such that the extension of a board with a TPM does not require the number of steps it does currently.

Since the current scope focused mainly on the implementation of DAA authentication, in the future it would be interesting to implement the attestation prototype with PUFs as proposed in the design section. Especially implementing a PUF-based attestation where the device anonymity is respected is interesting. Questions that could be addressed about key storage, and how message signing with verification would work since no external chip is installed on the platform.

## 7.2 Summary and Conclusions

Starting from the literature review in the background and after formulating the research questions this work was able to lay a foundation from which authentication of IoT devices can be pursued further. While working on this thesis it was possible to define use cases for device onboarding.

To address **RQ 1** it can be summarized that device onboarding happens on one hand if a new node is purchased from a manufacturer and installed to an existing network and on the other hand it also is a necessary step if the device is resold to a new owner and thus has to access a new network. The last use case specified was recovering after a node was corrupted or other external influences forced the node to be recovered first. It can be concluded that any device trying to join a new network needs to be able to attest itself.

**RQ 2** asked bout the Advantages and Disadvantages of DAA and PUF respectively and how they compare. This was covered mainly in the evaluation section. Both methods work very differently and have advantages for some uses and disadvantages for others. Currently, in the scope of this context, SRAM-PUFs do not match the goal of anonymity and therefore DAA was preferred.

The third Question **RQ 3** asked how EAP needs to be extended to use the proposed attestation methods. Potential solutions are described in the design chapter. Both suggested architectures try to maintain the anonymity of the node towards the network. While describing both potential architectures one apparent speciality is that DAA and PUF have different approaches to authentication and for DAA the process to obtain a root of trust without compromising its identity is documented and a central aspect. The same thing cannot be said for PUFs currently. Most literature covering authentication via PUF did use the authenticator directly to manage the challenge-response mechanisms which is why the design for PUF-based attestation is more experimental. Since at some point the decision was made to only start the implementation of the DAA, it remains open if the suggested design holds. Nevertheless, both DAA and PUF can extend EAP if the device does not need to stay anonymous.

*RQ 4* remains open for future progress.

Finally considering the broad Topic and organizational aspects It can be said however that despite the prototype remaining in its unfinished state, the thesis has laid out the groundwork necessary to develop more advanced test beds and it has opened opportunities for more defined future research.

# Chapter 8

# Bibliography


[1] M. Fagan, "Trusted internet of things (IoT) device network-layer onboarding and lifecycle management",

[2] "CERTIFY-general.pptx". (), [Online]. Available: https://umu-projects.inf.um.es/products/files/doceditor.aspx?fileid=7473 (visited on 12/20/2023).

[3] E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation", 2004, pp. 132–145. [Online]. Available: https://eprint.iacr.org/undefined/undefined (visited on 06/20/2023).

[4] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation", in *Proceedings of the 44th annual conference on Design automation - DAC '07*, ISSN: 0738100X, San Diego, California: ACM Press, 2007, p. 9, ISBN: 978-1-59593-627-1. DOI: 10.1145/1278480.1278484. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1278480.1278484 (visited on 07/14/2023).

[5] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. "Extensible authentication protocol (EAP)". (), [Online]. Available: https://www.ietf.org/rfc/rfc3748.txt (visited on 07/15/2023).

[6] X. Chen and D. Feng, "A new direct anonymous attestation scheme from bilinear maps", in *2008 The 9th International Conference for Young Computer Scientists*, Nov. 2008, pp. 2308–2313. DOI: 10.1109/ICYCS.2008.396.

[7] E. Brickell and J. Li, *A pairing-based DAA scheme further reducing TPM resources*, Publication info: Published elsewhere. Unknown where it was published, 2010. [Online]. Available: https://eprint.iacr.org/2010/067 (visited on 08/30/2023).

[8] K. Yang, L. Chen, Z. Zhang, C. J. P. Newton, B. Yang, and L. Xi, "Direct anonymous attestation with optimal TPM signing efficiency", *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2260–2275, 2021, Conference Name: IEEE Transactions on Information Forensics and Security, ISSN: 1556-6021. DOI: 10.1109/TIFS.2021.3051801.

[9] "TCG EK credential profile for TPM family 2.0", Trusted Computing Group. (), [Online]. Available: https://trustedcomputinggroup.org/resource/http-trustedcomputinggroup-org-wp-content-uploads-tcg-ek-credential-profile-v-2-5-r2_published-pdf/ (visited on 11/29/2023).

[10] S. Sutar, A. Raha, and V. Raghunathan, "D-PUF: An intrinsically reconfigurable DRAM PUF for device authentication in embedded systems", in *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, Pittsburgh Pennsylvania: ACM, Oct. 2016, pp. 1–10, ISBN: 978-1-4503-4482-1. DOI: 10.1145/2968455.2968519. [Online]. Available: https://dl.acm.org/doi/10.1145/2968455.2968519 (visited on 07/15/2023).

[11] S. Sutar, A. Raha, D. Kulkarni, R. Shorey, J. Tew, and V. Raghunathan, "D-PUF: An intrinsically reconfigurable DRAM PUF for device authentication and random number generation", *ACM Transactions on Embedded Computing Systems*, vol. 17, no. 1, pp. 1–31, Jan. 31, 2018, ISSN: 1539-9087, 1558-3465. DOI: 10.1145/3105915. [Online]. Available: https://dl.acm.org/doi/10.1145/3105915 (visited on 07/15/2023).

[12] S. R. Niya, B. Jeffrey, and B. Stiller, "KYoT: Self-sovereign IoT identification with a physically unclonable function", in *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, ISSN: 0742-1303, Nov. 2020, pp. 485–490. DOI: 10.1109/LCN48667.2020.9314816.

[13] A. Rubens, C. Rigney, S. Willens, and W. A. Simpson, "Remote authentication dial in user service (RADIUS)", Internet Engineering Task Force, Request for Comments RFC 2865, Jun. 2000, Num Pages: 76. DOI: 10.17487/RFC2865. [Online]. Available: https://datatracker.ietf.org/doc/rfc2865 (visited on 09/03/2023).

[14] V. Fajardo, J. Arkko, J. A. Loughney, and G. Zorn, "Diameter base protocol", Internet Engineering Task Force, Request for Comments RFC 6733, Oct. 2012, Num Pages: 152. DOI: 10.17487/RFC6733. [Online]. Available: https://datatracker.ietf.org/doc/rfc6733 (visited on 09/03/2023).

[15] D. Simon and B. D. Aboba, "PPP EAP TLS authentication protocol", Internet Engineering Task Force, Request for Comments RFC 2716, Oct. 1999, Num Pages: 24. DOI: 10.17487/RFC2716. [Online]. Available: https://datatracker.ietf.org/doc/rfc2716 (visited on 12/19/2023).

[16] "802.1X – Übersicht und EAP-Typen", Intel. (), [Online]. Available: https://www.intel.com/content/www/de/de/support/articles/000006999/wireless/legacy-intel-wireless-products.html (visited on 11/29/2023).

[17] S. Zhang and J. Liu, "A WLAN anonymous authentication scheme combining EAP-TLS and DAA", in *2012 Second International Conference on Instrumentation, Measurement, Computer, Communication and Control*, Dec. 2012, pp. 1232–1234. DOI: 10.1109/IMCCC.2012.289.

[18] L. Chen and J. Li, "Flexible and scalable digital signatures in TPM 2.0", in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*, Berlin, Germany: ACM Press, 2013, pp. 37–48, ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516729. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2508859.2516729 (visited on 08/30/2023).

[19] J. Camenisch, M. Drijvers, and A. Lehmann, "Universally composable direct anonymous attestation", in *Public-Key Cryptography – PKC 2016*, C.-M. Cheng, K.-M. Chung, G. Persiano, and B.-Y. Yang, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2016, pp. 234–264, ISBN: 978-3-662-49387-8. DOI: 10.1007/978-3-662-49387-8_10.

[20] J. Camenisch, L. Chen, M. Drijvers, A. Lehmann, D. Novick, and R. Urian, "One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation",

in *2017 IEEE Symposium on Security and Privacy (SP)*, ISSN: 2375-1207, May 2017, pp. 901–920. DOI: 10.1109/SP.2017.22.

[21] J. Whitefield, L. Chen, R. Sasse, S. Schneider, H. Treharne, and S. Wesemeyer, "A symbolic analysis of ECC-based direct anonymous attestation", in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, Jun. 2019, pp. 127–141. DOI: 10.1109/EuroSP.2019.00019.

[22] K. Nimmy, S. Sankaran, and K. Achuthan, "A novel lightweight PUF based authentication protocol for IoT without explicit CRPs in verifier database", *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 5, pp. 6227–6242, May 1, 2023, ISSN: 1868-5145. DOI: 10.1007/s12652-021-03421-4. [Online]. Available: https://doi.org/10.1007/s12652-021-03421-4 (visited on 07/15/2023).

[23] G. Vaidya, T. V. Prabhakar, and L. Manjunath, "GPIO PUF for IoT devices", in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, ISSN: 2576-6813, Dec. 2020, pp. 1–6. DOI: 10.1109/GLOBECOM42002.2020.9322590. [Online]. Available: https://ieeexplore.ieee.org/document/9322590 (visited on 11/29/2023).

[24] "TCG_guidance_for_securing_iot_1_0r21.pdf", Trusted Computing Group. (), [Online]. Available: https://www.trustedcomputinggroup.org/wp-content/uploads/TCG_Guidance_for_Securing_IoT_1_0r21.pdf (visited on 06/27/2023).

[25] "IBM documentation". (Mar. 22, 2021), [Online]. Available: https://www.ibm.com/docs/en/zos/2.1.0?topic=keys-rsa-private-public (visited on 07/16/2023).

[26] S. Wesemeyer, C. J. Newton, H. Treharne, L. Chen, R. Sasse, and J. Whitefield, "Formal analysis and implementation of a TPM 2.0-based direct anonymous attestation scheme", in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, Taipei Taiwan: ACM, Oct. 5, 2020, pp. 784–798, ISBN: 978-1-4503-6750-9. DOI: 10.1145/3320269.3372197. [Online]. Available: https://dl.acm.org/doi/10.1145/3320269.3372197 (visited on 07/30/2023).

[27] "Remote attestation with tpm2 tools", tpm2-software community. (Jun. 12, 2020), [Online]. Available: https://tpm2-software.github.io/2020/06/12/Remote-Attestation-With-tpm2-tools.html (visited on 12/10/2023).

[28] F. Farha, H. Ning, K. Ali, L. Chen, and C. Nugent, "SRAM-PUF-based entities authentication scheme for resource-constrained IoT devices", *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5904–5913, Apr. 2021, Conference Name: IEEE Internet of Things Journal, ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.3032518.

[29] "Difference between SRAM and DRAM", GeeksforGeeks. Section: Computer Organization & Architecture. (Apr. 13, 2020), [Online]. Available: https://www.geeksforgeeks.org/difference-between-sram-and-dram/ (visited on 12/10/2023).

[30] "CERTIFY-UBITECH presentation 1". (), [Online]. Available: https://umu-projects.inf.um.es/products/files/doceditor.aspx?fileid=7467 (visited on 12/20/2023).

[31] "B-l462e-CELL1". (), [Online]. Available: https://estore.st.com/en/b-l462e-cell1-cpn.html (visited on 07/16/2023).

[32] V. Senni, F. Federici, and A. Cilardo, *WP4/5 - IoT platform working group*. [Online]. Available: https://umu-projects.inf.um.es/products/files/doceditor.aspx?fileid=8806.

[33] "ST4sim-200m - GSMA eSIM system-on-chip for secure m2m industrial applications - STMicroelectronics". (), [Online]. Available: https://www.st.com/en/secure-mcus/st4sim-200m.html (visited on 07/18/2023).

[34] "Type 1sc-DM(LBAD0xx1sc) | LPWA products", Murata Manufacturing Co., Ltd. (), [Online]. Available: https://www.murata.com/en-global/products/connectivitymodule/lpwa/overview/lineup/type-1sc-dm (visited on 07/18/2023).

[35] "STM32stepbystep:step1 tools installation - stm32mcu". (), [Online]. Available: https://wiki.st.com/stm32mcu/wiki/STM32StepByStep:Step1_Tools_installation#Download_STM32CubeL4_Firmware_package (visited on 07/19/2023).

[36] R. P. Ltd. "Buy a raspberry pi 3 model b", Raspberry Pi. (), [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b/ (visited on 09/07/2023).

[37] R. P. Ltd. "Raspberry pi 4 model b specifications", Raspberry Pi. (), [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/ (visited on 07/19/2023).

[38] "SDRAM full form", GeeksforGeeks. Section: Full Form. (Apr. 11, 2020), [Online]. Available: https://www.geeksforgeeks.org/sdram-full-form/ (visited on 09/08/2023).

[39] R. P. Ltd. "Raspberry pi OS", Raspberry Pi. (), [Online]. Available: https://www.raspberrypi.com/software/ (visited on 09/08/2023).

[40] "LetsTrust TPM", pi3g.com. (), [Online]. Available: https://pi3g.com/products/industrial/letstrust-tpm/ (visited on 09/08/2023).

[41] "Installing classic AVR boards. | arduino documentation". (), [Online]. Available: https://docs.arduino.cc/software/ide-v1/tutorials/getting-started/cores/arduino-avr (visited on 08/26/2023).

[42] "ESP product selector". (), [Online]. Available: https://products.espressif.com/#/product-selector?language=en&names= (visited on 09/09/2023).

[43] *ESP32WROOM32d & ESP32WROOM32u datasheet*. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf (visited on 09/09/2023).

[44] "Http.server — HTTP servers", Python documentation. (), [Online]. Available: https://docs.python.org/3/library/http.server.html (visited on 12/20/2023).

[45] "Socketserver — a framework for network servers", Python documentation. (), [Online]. Available: https://docs.python.org/3/library/socketserver.html (visited on 12/20/2023).

[46] "ECC — PyCryptodome 3.19.0 documentation". (), [Online]. Available: https://www.pycryptodome.org/src/public_key/ecc (visited on 10/30/2023).

[47] "MIME types (IANA media types) - HTTP | MDN". (Dec. 18, 2023), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types (visited on 12/19/2023).

[48] "Email.parser: Parsing email messages", Python documentation. (), [Online]. Available: https://docs.python.org/3/library/email.parser.html (visited on 12/20/2023).

[49] SquareRootOfTwentyThree. "Answer to "how do i use a x509 certificate with PyCrypto?"", Stack Overflow. (Oct. 16, 2012), [Online]. Available: https://stackoverflow.com/a/12921889 (visited on 12/20/2023).

[50] B. Cornianu. “Change swap size in ubuntu 18.04 or newer”, Bogdan Cornianu. (Aug. 11, 2018), [Online]. Available: https://bogdancornianu.com/change-swap-size-in-ubuntu/ (visited on 12/18/2023).

[51] “Tpm2-tools”. (), [Online]. Available: https://tpm2-tools.readthedocs.io/en/latest/ (visited on 11/14/2023).

[52] “Tpm2-tss/INSTALL.md at master · tpm2-software/tpm2-tss”. (), [Online]. Available: https://github.com/tpm2-software/tpm2-tss (visited on 12/15/2023).

[53] “Tpm2-abrmd/INSTALL.md at master · tpm2-software/tpm2-abrmd”, GitHub. (), [Online]. Available: https://github.com/tpm2-software/tpm2-abrmd/ (visited on 12/15/2023).

[54] “TCG software stack (TSS) specification”, Trusted Computing Group. (), [Online]. Available: https://trustedcomputinggroup.org/resource/tcg-software-stack-tss-specification/ (visited on 11/29/2023).

[55] dotnet-bot. “Convert.FromBase64string(string) method (system)”. (), [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/system.convert.frombase64string?view=net-8.0 (visited on 12/17/2023).

[56] “Get started with CMake tools on linux”. (), [Online]. Available: https://code.visualstudio.com/docs/cpp/cmake-linux (visited on 12/18/2023).

[57] “Download visual studio code - mac, linux, windows”. (), [Online]. Available: https://code.visualstudio.com/Download (visited on 12/18/2023).

[58] “Libcurl example - postit2.c”. (), [Online]. Available: https://curl.se/libcurl/c/postit2.html (visited on 12/18/2023).

[59] “Libcurl - API”. (), [Online]. Available: https://curl.se/libcurl/c/ (visited on 12/17/2023).

# Abbreviations

| | |
|---|---|
| AAA | Authentication, Authorization, and Accounting |
| AIK | Attestation Identity Key |
| DAA | Direct Anonymous Attestation |
| DER | Distinguished Encoding Rules (Encoding) |
| DRAM | Dynamic RAM |
| DSP | Digital Signal Processing |
| EAP | Extensible Authentication Protocol |
| ECC | Elliptic Curve Cryptography |
| EK | Endorsement Key |
| FPGA | Field Programmable Gate Array |
| IC | Integrated Circuit |
| IoT | Internet of Things |
| LTE | Long Term Evolution standard |
| MCU | Microcontroller (unit) |
| MEK | Master Encryption Key |
| NAS | Network Access Server |
| OEM | Original Equipment Manufacturer |
| OS | Operating System |
| PCR | Platform Configuration Registers |
| PEM | Privacy Enhanced Mail |
| PUF | Physical Unclonable Functions |
| RISC | Reduced Instruction Set Computing |
| SIM | Subscriber identity module |
| SDRAM | Synchronous Dynamic RAM |
| SRAM | Static RAM |
| TCG | Trusted Computing Group |
| TLS | Transport Layer Security |
| TPM | Trusted Party Module |
| TTLS | Tunneled Transport Layer Security |
| USB | Universal Serial Bus |
| UART | Universal Asynchronous Receiver / Transmitter |

# List of Figures

# List of Tables

# Appendix A

# Installation Guidelines

## A.1 Installation on Raspberry PI

Here the installation detailed in the implementation is summarized again starting from the point where the TPM module is mounted on the board. To activate the module:

1. *sudo apt-get update && sudo apt-get upgrade*
2. *sudo nano /boot/config.txt* edit file by adding *dtparam=spi=on dtoverlay=tpm-slb9670* at the bottom then save (Ctrl + S ) and close (Ctrl + X).
3. *sudo reboot*

Add dependencies for TPM Software as well as other libraries that are needed but missing:

1. *sudo apt -y install autoconf-archive libcmocka0 libcmocka-dev procps iproute2 build-essential git pkg-config gcc libtool automake libssl-dev uthash-dev autoconf doxygen libjson-c-dev libini-config-dev libcurl4-openssl-dev uuid-dev libltdl-dev libusb-1.0-0-dev libftdi-dev libglib2.0-dev*
2. *sudo apt-get install -y cl-base64*

Install Tpm2-tss:

1. *git clone https://github.com/tpm2-software/tpm2-tss.git*
2. *cd tpm2-tss*
3. *./bootstrap*
4. *./configure –prefix=/usr*
5. *make -j5*

6. *sudo make install*

Install TPM2-abrmd:

1. Add a system user named tss *sudo useradd –system –user-group tss*
2. *git clone https://github.com/tpm2-software/tpm2-abrmd.git.*
3. *cd tpm2-abrmd*
4. *./bootstrap.*
5. *./configure –with-dbuspolicydir=/etc/dbus-1/system.d –with-udevrulesdir=/usr/lib/udev/rules.d –with-systemdsystemunitdir=/usr/lib/systemd/system –libdir=/usr/lib64 –prefix=/usr*
6. *make -j5*
7. *sudo make install*

Install TPM2-Tools:

1. *git clone https://github.com/tpm2-software/tpm2-tools.git.*
2. *cd tpm2-tools*,
3. *./bootstrap*
4. *./configure –prefix=/usr*
5. *make -j5 sudo make install*

Installation of visual studio code:

1. Download from[57]
2. Execute downloaded file
3. Using the Extension Window download the Extensions for C/C++ and CMake.

## A.2 Installation on the PC

*pip install pycryptodome.*