Bachelor Thesis

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Remote Attestation for IoT with Smart Verifier

Lukas Petzi
Department of Computer Science
Chair of Computer Science II (Software Engineering)

Prof. Dr.-Ing. Alexandra Dmitrienko
First Reviewer

Ala Ben Yahya
First Advisor

Submission
21. January 2021

www.uni-wuerzburg.de

Contents

Abstract

The Internet is evolving. The Internet of Things (IoT) is a disruptive technology, billions of small and smart devices are encompassing every aspect of our lives, which leads us to a new era of the Internet. IoT connects billions of heterogeneous embedded devices in large networks. Restricted capabilities of the devices in combination with various mutually mistrusting parties throughout the network pose new challenges to security. As IoT devices become increasingly important, so does trust in their secure and reliable operation. But before we can trust them, we need to establish the trust. In the past, remote attestation has emerged into a very powerful tool in order to establish trust between devices. Traditional remote attestation establishes a static trust between two devices. This approach suffers badly heterogeneity of IoT network and purely scales to a large number of devices. Furthermore, it is very vulnerable to Denial of Service attacks.
Within this thesis, we present a new approach to Remote Attestation in IoT networks. Our design combines the core strength of Remote Attestation with the advantages blockchain technology in order to enable attestation and validation of attestation evidences in a scalable manner. At first, we develop a remote attestation scheme on a low end device serving as an exemplary IoT device. Second, we design our smart verifier, a flexible system capable of validating and storing attestation evidence. The system is build on top of blockchain technology and solves several issues accompanied by Traditional Remote Attestation in IoT such as scalability and device heterogeneity. In the end, we build and evaluate a proof of concept implementation based on ARM TrustZone and Sawtooth Hyperledger. The proposed system architecture enables trust establishment in a scalable manner even in heterogeneous networks while preventing denial of service attacks.

Zusammenfassung

Das Internet verändert sich. Das Internet der Dinge (IoT) ist eine disruptive Technologie, Milliarden kleiner und intelligenter Geräte dringen in jeden Bereich unseres Lebens vor, und führt uns damit in ein neues Zeitalter des Internets. Das Internet der Dinge verbindet Milliarden heterogener eingebetteter System innerhalb rießiger Netzwerke. Die beschränkten Möglichkeiten der Geräte in Verbindung mit zahlreichen sich gegenseitig Misstrauenden Parteien innerhalb des Netwerkes stellt Sicherheit vor neue Herausforderungen. Dadurch, dass IoT immer wichtiger wird, ist unser Vertrauen in die sichere und verlässliche Arbeitsweise immer wichtiger. Aber bevor wir dieses Vertrauen aussprechen können, muss es aufgebaut werden. In der Vergangenheit hat sich gezeigt, dass Remote Attestation ein verlässliches Werkzeug ist um Vertrauen zwischen Geräten zu etablieren. Traditionelles Remote Attestation sorgt für Vertrauen zwischen zwei Geräten. Dieser Ansatz leidet stark unter der Heterogenität von IoT Netwerken und der großen Anzahl der Geräte. Außerdem ist es sehr Anfällig für Denial of Service Angriffe.
In dieser Arbeit präsentieren wie einen neuen Ansatz für Remote Attestation im Internet der Dinge. Unser Design verbindet die Stärken von Remote Attestation mit den Vorteilen von Blockchain Technologien um Attestierung und Validierung der Resultat in einer Skalierbaren Form zu ermöglichen. Zu Beginn entwerfen wir ein Remote Attestation Schema für ein simples Endgeräte, das exemplarisch als IoT Geräte fungiert. Im Anschluss entwerfen wir unseren Smart Verifier, ein flexibles System mit der Möglichkeit die Resultate von Attestierungsprozessen zu validieren und zu speichern. Das System wurde basieren auf Blockchain Technologien entwickelt und löst zahlreiche Probleme einhergehend mit traditionallem Remote Attestation in IoT wie die Skalierbarkeit und Heterogenität der Geräte. Abschließen wurde eine Proof of Concept Implementierung durchgeführt und evaluiert basieren auf ARM TrustZone und Sawtooth Hyperledger. Die vorgestellte Systemarchitektur ermöglicht es auch in großen und heterogenen Netwerken vertrauen zwischen Geräten aufzubauen und verhindert gleichzeitig Denial of Service Angriffe.

1. Introduction

The Internet of Things impacts our society more then ever before. There are seemingly endless use cases for IoT systems and therefore more and more embedded devices are being deployed. These devices are used in various settings ranging from wearables and smart home solutions all the way up to industrial usage in smart factories or smart cities. Billions of IoT devices are already deployed and numbers are constantly rising [1]. The increasing integration into our daily life paired with connectivity, collection of sensitive data and execution of security critical operations make embedded devices an attractive target for attackers. To protect sensitive data in an IoT network it is necessary to ensure the integrity of all used devices.
In the future there will be millions of IoT devices within the same network. Due to the fact that all these devices usually communicate with each other, IoT networks are very vulnerable to common attacks like sinkhole [2] or man-in-the-middle attacks [3]. With sinkhole attacks, for example, an intruder can easily compromise the security of a network by only compromising a single device. But malicious IoT devices or even whole malicious networks can form more complex large scaling attacks like an example of 2018 shows where GitHub was victim of the until now biggest denial of service attack where at peak 1.35 Tbps traffic tried to bring down GitHub servers [4].
Despite all concerns, security is not a priority for most low end IoT device manufacturers, due to cost, size and power constrains. Thereby, most IoT devices lack on security features. Their simple design and restricted capabilities leave most IoT devices very vulnerable to attackers as many strong security mechanisms like for example firewalls cannot be applied to a simple IoT device. But it is unrealistic to expect low end devices to have the means to prevent malware attacks. Because of that, it is crucial to deploy a reliable mechanism that detects malicious devices. Therefore, it can help to establish trust between different devices. Establishing trust using a unified scheme is complicated due to the large number of devices and the heterogeneity of IoT networks. Additionally, IoT networks usually interconnect devices form different parties and vendors which are usually mutually mistrusting. In order to establish trust within an IoT nework, we need a procedure that can be easily applied to every device and provides information about the internal state of the device. This procedure is called, Remote Attestation (RA). The term RA was introduced by Trusted Computing Group [5] and describes the process of one device, usually called *verifer*, attesting the internal state of another device, called *prover*, in order to identify whether the *prover* is compromised or not.
Attestation techniques can be divided in three different categories [6]: (1) software-based, (2) hardware-based and (3) hybrid-based. Software-based attestation schemes usually ex-

ploit the computational limit of embedded devices. This means they use the fact that the execution of specific algorithms takes a specific amount of time using the device's computational limit. Software-based attestation techniques usually rely on strong assumptions like the adversery being passive during the whole attestation process and require detailed knowledge about network delays to prevent false positives and is thereby not suitable for large scale and dynamic IoT networks. The hardware-based attestation techniques require specific secure hardware such as Trusted Platform Module (TPM) [5]. Most of these secure hardware solutions scale poorly to IoT networks due to their high complexity and costs. But there are some very promising options especially for industrial IoT where security is very important in order to protect intellectual property as well as other sensitive data. The most promising one of these solutions is ARM TrustZone. It does not provided remote attestation by itself, but the hardware architecture allows the developer to implement an own attestation scheme. With TrustZone, ARM developed a technology that separates memory space of a single core device into secure and non-secure. The so-called secure world can run cryptographic operations and store private data like cryptographic keys. These boards are built quite small and are available of the shelf while not being very expensive. Finally, there are hybrid Remote Attestation Schemes which are a mixture out of both techniques. They usually rely on software based attestation combined with some minor secure hardware modification capabilities, such as Read Only Memory (ROM) to store cryptographic keys. This combination does not rely on strong assumptions like software based attestation and is also not dependent on significant hardware modification, but can be deployed with only a few small changes to common off the shelf IoT devices. These features make hybrid attestation schemes suitable for most low end IoT devices who have the appropriate hardware.
Traditional RA was not developed targeting a large number of heterogeneous IoT devices but targeting PCs. This leaves traditional remote attestation schemes with some challenges to overcome when applying it to IoT. Apart from the attestation scheme itself, validating and storing attestation evidences is a defiance to solve when talking about remote attestation in IoT. Most schemes either have device to device communication to exchange attestation messages or use a central authority like a server to store and verify the evidences. But unfortunately, both approaches offer some major drawbacks. Device to device communication for attestation must tackle a key management challenge as it requires the devices to have a shared secret in order to verify the authenticity of the received evidence. This is not applicable to large IoT networks as this approach does not scale. That is because, a single device requires a unique shared secret for every other device within the network in order to be able to attest them. Another downfall of this approach is the possibility of denial of service attacks. A malicious device can just ask another device for attestation over and over again. This will keep the device from doing anything but calculating attestation evidences. In order to mitigate this issue, one could introduced a central authority that ask for attestation and validates received evidences. This introduces a single point of failure as the system is highly dependent on the central authority and attackers can easily run denial of service attacks by shutting down the central server. Another downside of this approach is that one must introduce a trusted party into a highly dynamic and heterogeneous network. IoT devices are usually controlled by different parties and this approach requires all of them to trust the central authority in order to run remote attestation. This is also a problem with swarm attestation schemes [7][8]. These schemes are developed to attest a large number of devices but usually require the devices to have similar hardware and software configurations and are thereby not applicable to heterogeneous networks.
In order to solve the issues discussed above, our idee is to use a blockchain as decentralized authority and a smart contract to handle attestation requests and validate incoming attestation evidences. Additionally, we made our attestation results publicly verifiable

and use transactions in order to exchange attestation evidences and requests. This comes with the benefit, that we are able to drastically reduce the number of saved keys on a device as we do not need any shared secrets anymore but can use public key cryptography. Additionally, we have a decentralized authority and all data is publicly verifiable for every network participant. In addition to that, by using a smart contract as a verifier, we solve the problem with denial of service attacks. With introduction of the blockchain technology, we remove the direct device to device communication during the attestation process and mitigate denial of service attacks made possible by flaws in traditional RA. Even submitting a large amount of transactions requesting attestation is not feasible anymore.
The following thesis is structured as follows. The first upcoming chapter is the background chapter. This chapter provides all necessary information required to understand this thesis. Chapter 3 discusses the current related work in this area of research. It first comprehends current traditional remote attestation schemes before heading further towards remote attestation in combination with blockchain technology. In the following chapter we discuss our approach. In the beginning, some overall challenges with remote attestation in IoT are explained before presenting our approach of using a blockchain as a central authority to solve this challenges. Additionally, the used prover device and the attacker model is described. Chapter 5 is the implementation section. This section explains the implementation required to realize the presented approach and all used components. After the implementation section comes the evaluation in chapter 6. The Evaluation section is divided in performance evaluation of the system and a security evaluation. In the end, a conclusion is drown summarizing the overall thesis.

Contributions and Outline The contributions of this bachelor thesis can be summarized as follows:

- **Hardware-based attestation scheme utilizing ARM TrustZone.** We propose a hardware-based remote attestation scheme build on top of ARM TrustZone. It has a secure and non-secure application and is able to measure its internal state as well as communicating with the blockchain.
- **Decoupling Prover and Verifier.** The analysis of traditional remote attestation schemes in chapter 2 discovers that trust establishment based on remote attestation will not scale to larger networks due to single device-to-device communication patterns. With decoupling the prover and verifier pairs in our system, we enable scalability for remote attestation.
- **Public Verifiability of Remote Attestation.** In chapter 4 the introduced smart verification authority is designed. The usage of distributed ledger technology enables the system to make remote attestation publicly verifiable.
- **Remote Attestation in Publish/Subscribe Environment.** The publish and subscribe environment of IoT networks brings properties with it that complicate the remote attestation process, such as lack of time synchronization between devices and sleeping devices. Our approach does not only enable remote attestation in a publish and subscribe environment, but additionally fully removes the communication between two devices in order to get attestation evidence.
- **Evaluation.** Finally, we supply an evaluation of our system. The overview shows that the smart verifier operates very fast even with up too 100 transactions at once. In addition to that, we illustrate that our system effectively protects against the defined attacker model.

Overall, this thesis proposes a solution to trust establishment in large scale heterogeneous networks.

2. Background

2.1. Remote Attestation

The term Remote Attestation(RA) was introduced and disseminated from Trusted Computing Group (TCG) as an important concept of their Trusted Platform Module (TPM). The principal goal of RA is to provide a user with reliable knowledge about a platform's internal state. Usually, Remote Attestation is a process between two parties namely prover and verifier. An exemplary RA process is shown in Figure 2.1. In the first step the verifier submits a pseudo-random challenge to the prover. As a reaction, the prover measures its internal state. Once step two is done, the prover links the measurement to the challenge and returns it to the verifier. As a last step, the verifier compares the received evidence against a database of pre-known good values to determine whether the prover can be considered trusted or not. To ensure the authenticity of the evidence prover and verifier usually use some sort of shared secret.

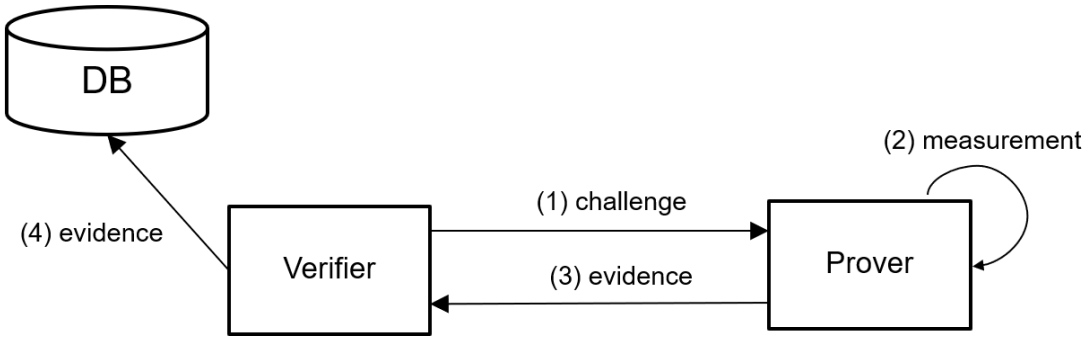

Figure 2.1.: Remote Attestation

2.2. Blockchain and Smart Contract

Bitcoin: A Peer-to-Peer Electronic Cash System [9] was the first decentralized cash system. Bitcoin was introduced in 2009 by Satoshi Nakamoto who was the first one to use a distributed ledger to store transactions in order to establish a cryptocurrency platform. In principle, a cryptocurrency is a decentralised system for interacting with virtual money in a shared global ledger [10]. That distributed ledger is called blockchain, because the ledger consists of a sequence of blocks which holds a complete list of transaction records

that were made within the network and it is not stored in a single location but on multiple network nodes. An exemplary overview is provided in figure 2.2.

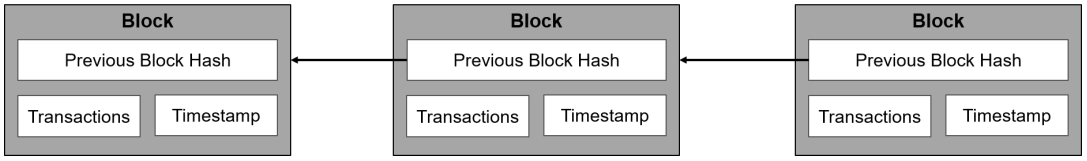

Figure 2.2.: Overview of a blockchain architecture

Each block consists of some transactions, a time stamp, and the hash of the previous block [11]. In order to determine, which block is added to the blockchain next, the network participants, also called miners, have to solve complex computational problems [12]. Different consensus mechanisms are used to determine what block is added to the blockchain. Bitcoin, the biggest blockchain platform uses proof-of-work (POW). The POW mechanism assumes that each node votes with his computing power by solving proof of work instances and constructing the appropriate blocks [13]. The node that solved the computation first is then allowed to add its block to the blockchain. Other blockchains use the proof-of-stake (POS) mechanism. With POS the competition of POW is replaced in order to reduce the computational requirement. Instead of solving complex computations to be able to add a block, POS chooses nodes based on stakes that they are holding or fully random [12] [14]. Since 2009 multiple blockchain platforms emerged with different features like varying consensus mechanisms, support of smart contract or the possibility to be deployed privately. In 1997 Nick Szabo [15] proposed the idea of smart contracts, a software having self-verifying, self-executing and tamper-resistant properties. A smart contract usually consists of a unique address, a set of executable functions, state variables and value [11]. Figure 2.3 displays a high level overview of a smart contract.

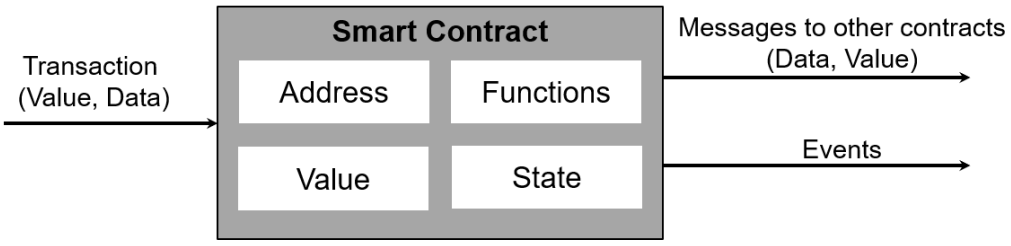

Figure 2.3.: Smart Contract

In order to trigger a function, a transaction targeting one of the functions has to be submitted. A transaction is a signed message published by a user to the network. The transaction includes input parameter which are required by the function in the contract. Once a function is triggered it executes the corresponding code and triggers an output event. Depending on the internal logic, the state may change or other smart contracts are triggered. In general, a smart contract can be very similar to a real world contract capturing agreements in a fully digital manner. Apart from being fully digital, smart contracts offer some obvious advantages [16]. In the first place, they offer increased efficiency as a transactions facilitated through smart contract does not require a trusted third party to validate it but the consensus of the network. This can also result in reduced transaction and legal costs due to the absence of a central authority. A final advantage is the greater transparency and anonymity. Transparency comes with the decentralisation of data through distributed ledger and anonymity is provided because no proof of identity is required but public-key cryptography instead [16]. The usage of smart contracts has

some downfalls as well. First of all, there is a big trust issue as you are fully reliant on computer software. Software can have bugs or errors in the code that can cause unpredictable malfunction of the smart contract or can be exploited by attackers [17]. Because smart contracts are immutable it is nearly impossible to fix bugs once the contract is deployed. Another issue is privacy, smart contracts are unable to store secrets as data saved in blockchain is publicly available [18]. As already mentioned, smart contracts are usually constructed upon an underlying distributed ledger, mostly a cryptocurrency platform.

2.3. Sawtooth Hyperledger

In this thesis, we use Sawtooth Hyperledger as our underlying distributed ledger platform. Sawtooth Hyperledger [19] is a project of the Linux Foundation and distributed under the Hyperledger umbrella like multiple other projects. It offers a modular and open source system for building and running a distributed ledger that also supports smart contracts. Figure 2.4 provided a high level overview of Sawtooth Hyperledger architecture.

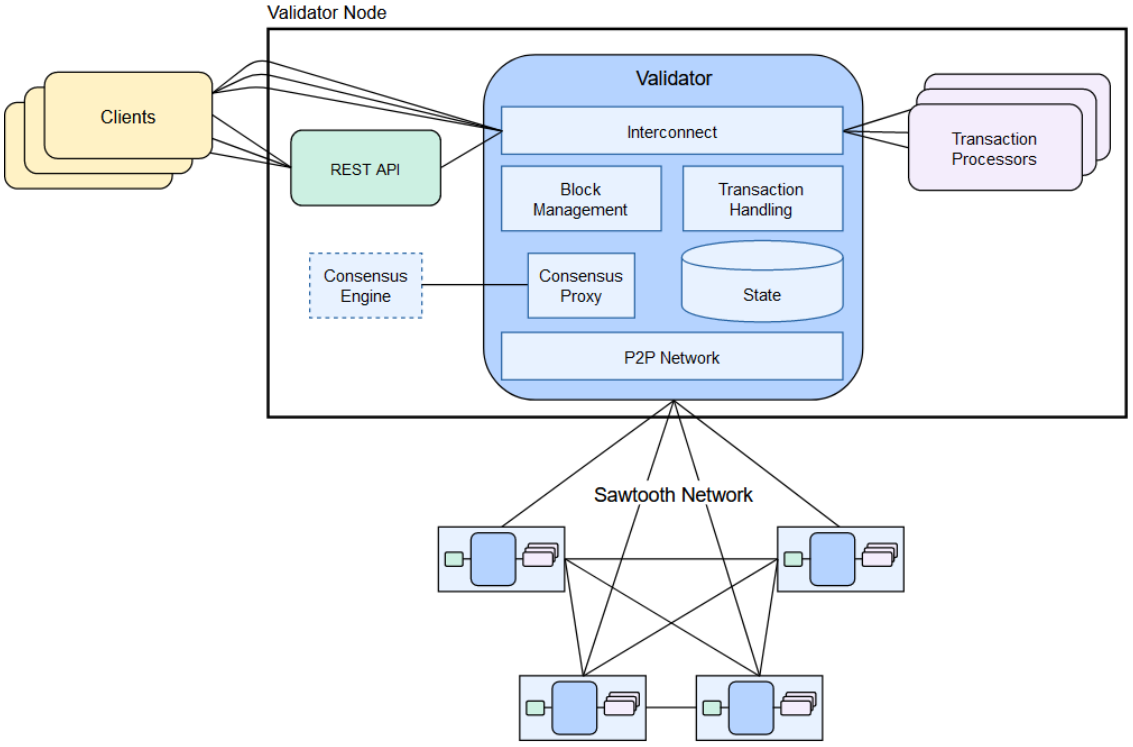

Figure 2.4.: Sawtooth Hyperledger High Level Architectural Overview [19]

In Sawtooth there are existing two different type of nodes, clients and validators. The client is submitting transactions and uses the provided functionality of the system. Validator nodes are required to run the blockchain and provide the application interfaces to the clients. The functionality of validator nodes in Sawtooth is equivalent to a miner in a public distributed ledger. The task of validator nodes is to validate transactions, create the corresponding blocks and attach them. In contrast to other distributed ledger platforms like Ethereum, the biggest distributed ledger supporting smart contracts, Sawtooth is deployed privately and offers some additional features like different consensus mechanisms. In addition, Sawtooth supports transaction families, an abstraction of a smart contract, that allows the developer to write smart contract logic in several languages. An additional feature called Sawtooth Seth also enables Sawtooth to include Ethereum smart contracts written in Solidity into its environment. Every application in Sawtooth is realized by a

transaction family. Each transaction family consists of three components. The first one is the transaction processor. The transaction processor defines the business logic of the application and provides the required functionality. This is what is considered a smart contract in a public blockchain. The code of the transaction processor runs on the validator nodes. The second component is the data model. We need to define a data model as it is responsible for recording and storing data for a transaction family. The third component consists of the client. The client logic defined the functionality and the format of transaction submitted to validator. Unlike many other blockchain platforms, Sawtooth Hyperledger does not have a cryptocurrency, which is obvious because there is no publicly available instance but every user can deploy his very own instance fully private. Sawtooth supports different consensus algorithms. We chose Practical Byzantine Fault Tolerance (PBFT) as consensus algorithm for our system. PBFT is a voting based consensus algorithm that guarantees liveness and safety of the network as long as a minimum amount of nodes function properly. Unlike many other consensus algorithms, PBFT does not treat every node equally but introduces two types of nodes, primary and secondary nodes. The methodology of PBFT is described in [20] and the algorithm operates as follows:

- Client sends a request to invoke a operation to the primary
- The primary multicasts the request to all secondary nodes
- Secondary nodes execute the operation and send a reply to the client
- Client waits until he has received f+1 replies with the same result from different secondary nodes
- This is the result of the operation

Sawtooth PBFT is based on the methodology described above. The implementation differs a bit from original PBFT as it was not developed targeting distributed ledger technology, but as a way to generally reach consensus within a network. Therefore, the actual implementation has been modified to work in a blockchain context but the principles of operation are still the same. In order to generate a new block, a primary node creates candidate blocks and secondary blocks vote on them. The primary node changes in a round robin order.

3. Related Work

In this chapter, different categories of remote attestation are presented. Additionally, the current related work for this categories is shown, starting with traditional remote attestation and following up with the provers point of view on remote attestation as well as presenting some more advances schemes using blockchain in combination with RA. As already mentioned, the traditional remote attestation schemes can be divided in three categories namely hardware-, software- and hybrid-based.

3.1. Hardware-based

Hardware-based attestation scheme is an umbrella term for every remote attestation scheme utilizing special secure hardware in order to operate. This secure hardware can for example include protected storage for secure keys or cryptographic processors.

Trusted Platform Modules. Trusted Platform Modules (TPMs) [5] are one of the most popular secure hardware attestation architectures. TPMs are discrete co-processors equipped with special Platform Configuration Registers (PCRs). These PCRs are special registers inside the TPM and thus a secure location for integrity measurements. TPM does not only provide attestation but also various other security features like secure storage and a cryptographic processor. Due to its size and complexity, TPM is intended for larger platforms like computers and not for IoT devices.

Intel Software Guard Extension. Another hardware-based security architecture is Intel Software Guard Extension (Intel SGX) [21]. Intel SGX adds a set of new instructions and memory access changes to common Intel architecture. This extension enables the device to instantiate a protected container called *enclave*. It also provides a build-in mechanism for attestation. But similar to TPM, the usage of Intel SGX is more towards high end devices like computers and not IoT devices.

M-Shield. M-Shield [22] is a hardware-based mobile security architecture. It was developed by Texas Instruments and offers secure hardware for small devices like smartphones. M-Shield offers a complete security infrastructure like secure storage and a secure execution environment. In addition to that, a secure middleware component is provided to support the interoperability with third-party software to support other security features like attestation.

ARM TrustZone. The use of integrated trusted execution environments within the main processor is a third way to realize a hardware-based security. An example for such an architecture is ARM TrustZone [23]. As shown in figure 3.1, ARM TrustZone separates

all hardware and software resources into two worlds. A security world where the security subsystem is deployed together with all security relevant data and keys and a normal world for every other application. Software running in the secure world have a completely different view of the whole system then software running in normal world. In this way, cryptographic credentials and security functions can be hidden from software running in normal world. TrustZone's functionality is defined by software which makes it more flexible then other hardware-based approaches where functionality is hard-wired. TrustZone can be used to develop simple secure applications but even more complex security architectures can be implemented. Such an example is fTPM [24]. The authors used TrustZone to create a fully operating firmware-based TPM. The major downside of hardware-based security architectures is the mandatory use of secure hardware which results in extra costs. But in order to establish a secure system some trusted hardware components seem to be necessary. ARM TrustZone appears as a well-rounded solution that, unlike TPM or Intel SGX, can be used in IoT devices.

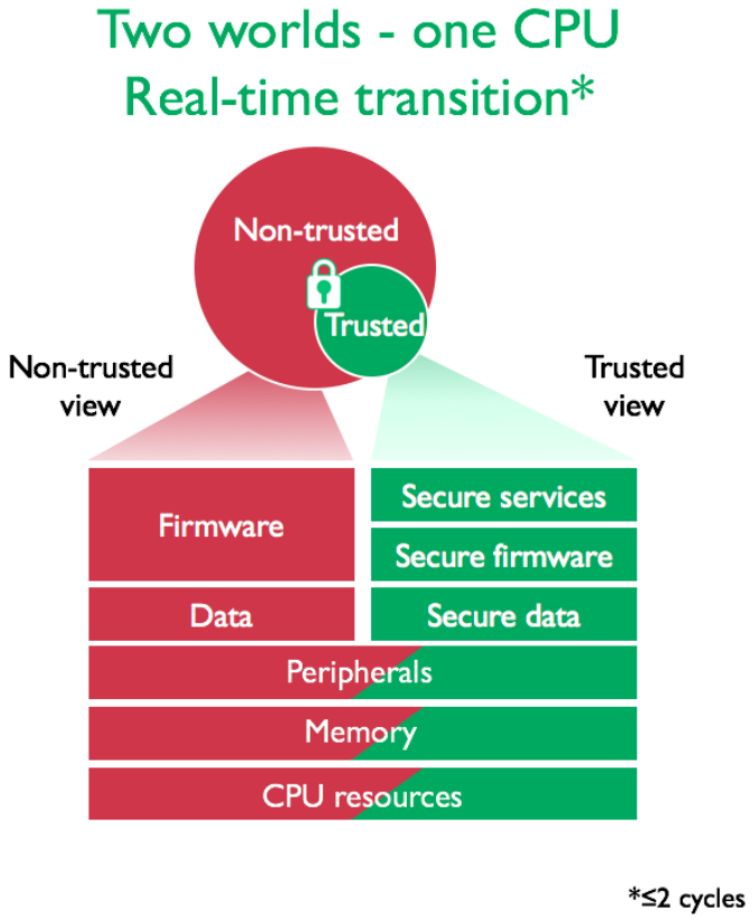

Figure 3.1.: ARM TrustZone [25]

3.2. Software-based

Software-based attestation techniques are not reliant on any hardware support and therefore cheap and easy to deploy on any IoT device. Software-based attestation makes use of side channel information like the execution time to detect abnormal behavior of software. Any modification of the executed function or algorithm would be reflected in a increased execution time. SWATT [26] is a software-based attestation technique using

pseudo-random memory traversal to attest the software state of a device. The *prover* can only execute specific algorithms within a certain time period. SWATT calculates a checksum over the whole memory of the *prover* using a pseudo-random number generator to determine the next address being attested. The *verifier* calculates a reference checksum using a reference software state. If both checksums are equal and the *prover* calculated the checksum within a certain time period, the *prover* is considered trusted and otherwise not.

3.3. Hybrid-based

Hybrid-based attestation schemes evolved as a combination of software and hardware-based schemes. They cannot be considered software-based because they required some sort of hardware support or small hardware modifications. Still, these schemes are not considered hardware-based because in contrast to hardware-based schemes, the required hardware for hybrid schemes is very simple. Most of the time Read Only Memory (ROM) or a Memory Protection Unit is sufficient in order to run a hybrid-based scheme. In literature many different hybrid remote attestation architectures can be found.

SMART. This solution [27] was one of the first hybrid remote attestation architectures proposed. The authors used standard Memory Control Unit (MCU) enabled platforms with an additional hardwired Read Only Memory (ROM). The additional ROM is used to store a secret key K and the attestation code. K and the attestation code are guarded by hard wired MCU access rules. The success of SMART is based on three security objectives: (1) *Prover Authentication*, (2) *External Verification* and (3) *Guaranteed Execution*. Therefore SMART uses four major security components namely, *Attestation ROM*, *Secure Key Storage*,*Access Control* and *Reset and Memory Erasure*. These components are considered necessary and sufficient for establishing a dynamic root of trust in a low-end IoT device and are located at the *prover*. SMART calculates a HMAC over a given memory region [a,b] before passing control to a specified memory address x and returning HMAC([a,b]) to the *verifier* who checks the correctness. To ensure the attestation process will not be interrupted, interrupts are globally disabled during the execution of the attestation code. The SMART architecture seems promising with the only downside that the whole memory region is attested at once, which makes it vulnerable to roving malware, and the attestation process cannot be interrupted which adversely affects the availability of the devices.

TrustLite. This scheme [28] architecture relies on ROM and an Execution-Aware Memory Protection Unit (EA-MPU), which is similar to a normal MPU, but also considers the address of the currently executing instruction when validating code or data access. While the EA-MPU is responsible for controlling every data access including regular memory, the ROM stores the platform key K and a secure boot loader which has exclusive access to K. The secure loader allows TrustLite to enable isolation of critical software components by initiating critical components via secure boot which sets up necessary memory access rules for each component in the MPU.

TrustLite enables different security features by the use of *trustlets*. A *trustlet* or *trusted task* is a program implementing a specific security feature. During attestation process interrupts can happen which are then managed by the TrustLite Exception Engine. The result is that attestation can be interrupted and continued later on.

SWARM. On top of SMART architecture builds SWARM [29]. Unlike SMART, SWARM allows interruption of the attestation process. In order to do so, SWARM does not attest the whole memory region at once, but divides it in n equal parts. After a block got attested it is possible to interrupt the process. The next block n_i being attested is determined randomly to detect roving malware. Additionally, the block is chosen in privat so that the malware remains unaware of which blocks are already measured and which will be measured next. This technique reduces the impact of the attestation process on the

prover while detecting malware with high probability.

HEALED. Another aspect to remote attestation is added by HEALED [30]. The attestation process in general remains unchanged, that means that HEALED does not modify the attestation process of its used platform. In the case that the *prover* turns out to be malicious, HEALED enables the *verifer* to heal infected memory regions. To heal the infected device HEALED aims to identify the infected memory region using a Merkle Hash Tree. Once the infected memory region is found, the $verifier$ looks for a healer device having the same reference software configuration as the infected device. As soon as a healer is found the infected device gets healed. This obviously relies on the assumption that at least one device within the network can heal the infected device, which means it has the same software configuration while not being infected.

TyTAN. This approach [31] uses nearly the same architecture as TrustLite. Additionally to a EA-MPU and a secure bootloader to securely load critical tasks into the platform, each device needs to run TyTAN OS, which is an extended version of FreeRTOS. This enables TyTAN to dynamically load an unload multiple tasks at runtime and establish secure communication between the tasks. A downside of this improvement is the fact that every device needs to run TyTAN OS which increases the overall complexity of the setup and might require more advanced hardware.

HYDRA. HYDRA [32] requires only very basic secure hardware. To compensate the lack of advanced hardware the formally verified seL4 microkernel is used. This kernel offers guarantees like process isolation and access control that most of the other architectures could only realize with secure hardware. HYDRA fully satisfies these requirements with software. The only hardware requirement is ROM to immutably store the seL4 kernel. One downside of HYDRA is the complexity of the kernel which results in a dependency on more advanced hardware suitable to implement HYDRA. The kernel also does not allow the interruption of the attestation process.

HAtt. A different approach is HAtt [33]. The authors used Physical Unclonable Functions (PUFs) embedded into IoT devices. The usage of PUFs is an effective way to protect IoT devices from physical attacks by eliminating the need of storing secret keys. This makes HAtt also viable against attacker models using hardware attacks. Therefore, HAtt differs from all the other already featured remote attestation schemes as they keep hardware attacks out of scope. HAtt still needs some sort of secure hardware as it is necessary to store the attestation code immutable and insure its immutability. HAtt divides the memory space of the device into n blocks with m words in every block. The attestation process attests the blocks in a pseudo-random manner using the PUF to determine which one is used next. Furthermore, only a random number of bits from every word in a block are attested. To increase the detection probability, the *verifier* might run the attestation process multiple times. This attestation technique does not require the whole memory space of the device to be attested, but attests only random parts of it. This reduces the computational complexity and still detects all malware including roving malware with high probability.

ATT-Auth. Just like HAtt, ATT-Auth [34] uses PUFs in its attestation process. The attestation protocols are build upon SWATT, a software-based attestation technique. The usage of a software-based attestation scheme reduces the cost of IoT devices as it does not require any secure hardware. To run Att-Auth every device needs to install SWATT and have its own PUF. No further secure hardware is required. For attestation, Att-Auth supports two different techniques, individual attestation and multiple node attestation. Individual attestation is based on the usage of SWATT in combination with a PUF to introduce a hardware root of trust to the attestation process. Additionally the verifier has a challenge response pair CRP (C,R) for every device it tries to attest. Due to the usage of SWATT, both single and multiple node attestation with Att-Auth rely on timing information which is not suitable for all types of IoT networks.

Att-Auth and HAtt seem to be very promising solutions for hybrid-based attestation. The

biggest downside of both schemes is the usage of PUFs. PUFs in general are very susceptible to environmental influences like temperature or humidity [35] and are thereby not reliable enough for huge IoT networks.

VRASED. The last hybrid remote attestation scheme is VRASED [36]. The authors of VRASED used the same architecture as the authors of SMART did. They divided their design into software and hardware components and formally verified all used components as well as the interaction between them. Since, VRASED uses the same architecture as SMART, it requires atomic execution of the attestation process and is by that not interruptible.

APEX and PURE. In order to further increase the capabilities of VRASED [36], APEX [37] and PURE [38] were proposed. Both are extensions of VRASED adding some additional features while not affecting VRASED in any way. APEX adds a proof of execution on top of VRASED while PURE enables provably secure and verified proofs of software update, erasure and system-wide reset. They cannot be deployed without VRASED as they are both fully reliant on VRASED secure architecture. Table 3.1 shows the adversarial model targeting each presented hybrid-based attestation scheme as well as the interruptibility of the attestation process. The first column shows the name of the attestation scheme while the following three columns display the attacker model of each scheme. Every attestation scheme considers remote and local adversaries, but only two schemes also include a physical adversary. The fifth column tells if the attestation process can be interrupted or not and in the last column can be seen that only the source code of VRASED is public.

Attestation scheme	Remote Adversaries	Local Adversaries	Physical Adversaries	Attestation can be interrupted	Public Source Code
SMART	√	√	×	×	×
TrustLite	√	√	×	√	×
SWARM	√	√	×	√	×
HEALED	√	√	×	Platform dependent	×
TyTAN	√	√	×	√	×
HYDRA	√	√	×	×	×
HAtt	√	√	√	√	×
Att-Auth	√	√	√	√	×
VRASED	√	√	×	×	√

Table 3.1.: Comparison of properties of hybrid attestation scheme

3.4. Attestation from the Provers Point of View

Most of the above presented attestation schemes focus on the scenario where there is a trusted verifier and a possibly malicious prover. The opposite setting with a compromised verifier and a honest prover is not considered in most attestation schemes and might offer some attack vectors. Tsudik et al showed in [39] that a malicious verifier can use the attestation protocol in order to run certain attacks against a prover device. As the computation of the attestation evidence takes some time, a malicious verifier can easily run a Denial-of-Service attack by requesting attestation over and over again. To run this attack, it is not even necessary to have a malicious verifier but the attack can be executed by any network participant that is able to impersonate the verifier. To mitigate these attacks, two additional countermeasures are required. Authentication of the attestation requests

published by the verifier is the first one. This prevents malicious devices from impersonating the verifier and can be achieved by using either a pre-shared key to establish a secure communication channel or public key cryptography to sign the messages. Unfortunately, this does not prevent DoS attacks as a attacker can just intercept and replay attestation requests. In order to solve this, we need a way to detect replayed attestation request. There are several standard ways proposed like nonces, counter or timestamps to ensure the freshness of the attestation request. Obviously, the prover still has to check the counter for example but this results in a minimal computational effort then calculating a whole attestation.

3.5. Remote Attestation in combined with Blockchain Technology

BARRETT. BlockchAin Regulated REmote aTTestation [40] attends the problem that in a RA setting a malicious verifier can repeatedly send attestation request in order to prevent the target device form performing its usual tasks or draining its battery. This attack is also called Computational Denial of Service (CDoS). To resolve the problem, BARRETT combines RA with the Public Ethereum Network (PEN). The proposed architecture can be seen in figure 3.2. In order to send an attestation request the verifier needs to submit a transaction to the PEN. For every transaction, the submitting account is forced to pay transaction fees. These fees would naturally alleviate the feasibility of a CDoS attacks as it might result in high costs for the attacker. Additionally, they used a smart contract that limits the maximum amount of incoming attestation requests within a time period for a single device.

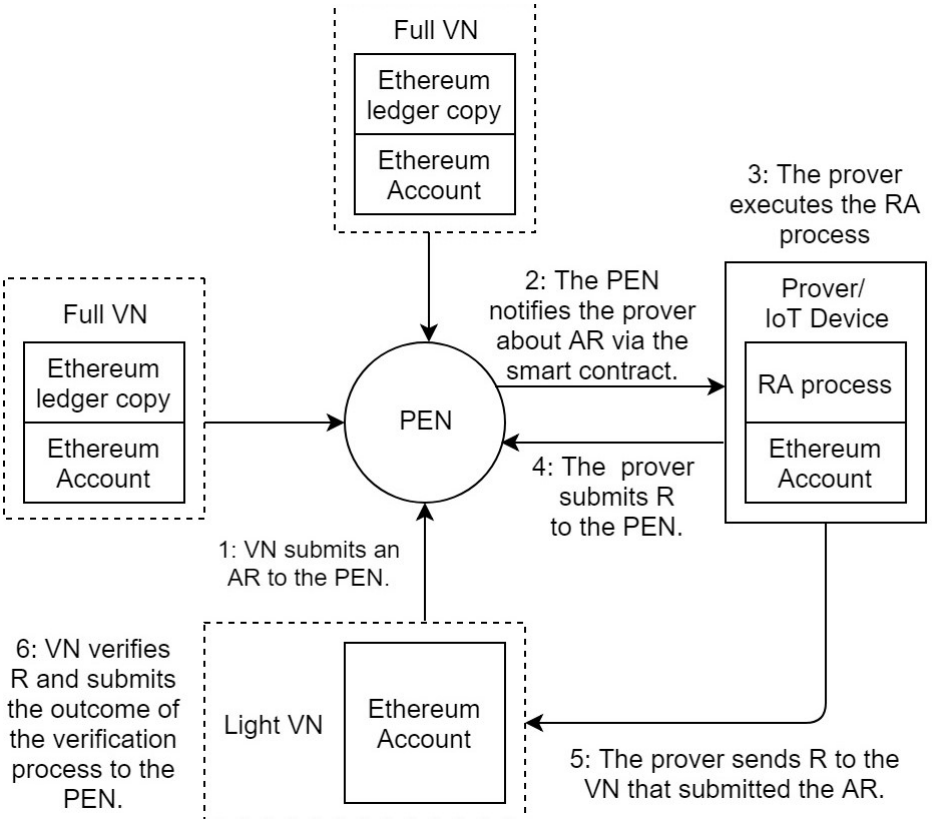

Figure 3.2.: BATTETT Architecture [40]

TM-Coin. This solution [41] uses the blockchain to store and manage the attestation results of a device. Figure 3.3 provides a detailed overview of the system architecture of

TM-Coin. The blockchain is used as a trustworthy decentralized database to store the attestation results and make them publicly accessible for every verifier. TM-Coin uses the miner to perform RA on a target device and publish the result to the blockchain. Later on, a verifier can request the attestation result directly from the blockchain and does not have to request the result from the prover device. This method makes the attestation process more efficient because attestation does not have to be calculated every time for every verifier. This means that once the attestation result is calculated and published to the blockchain every verifier device can request it from the blockchain and verify it. Therefore, the verifier does not need to trigger attestation of the prover device again but can just request the already calculated results from the blockchain. This process assumes the trustworthiness of the miners as they must act as a verifier and publish the attestation result to the blockchain.

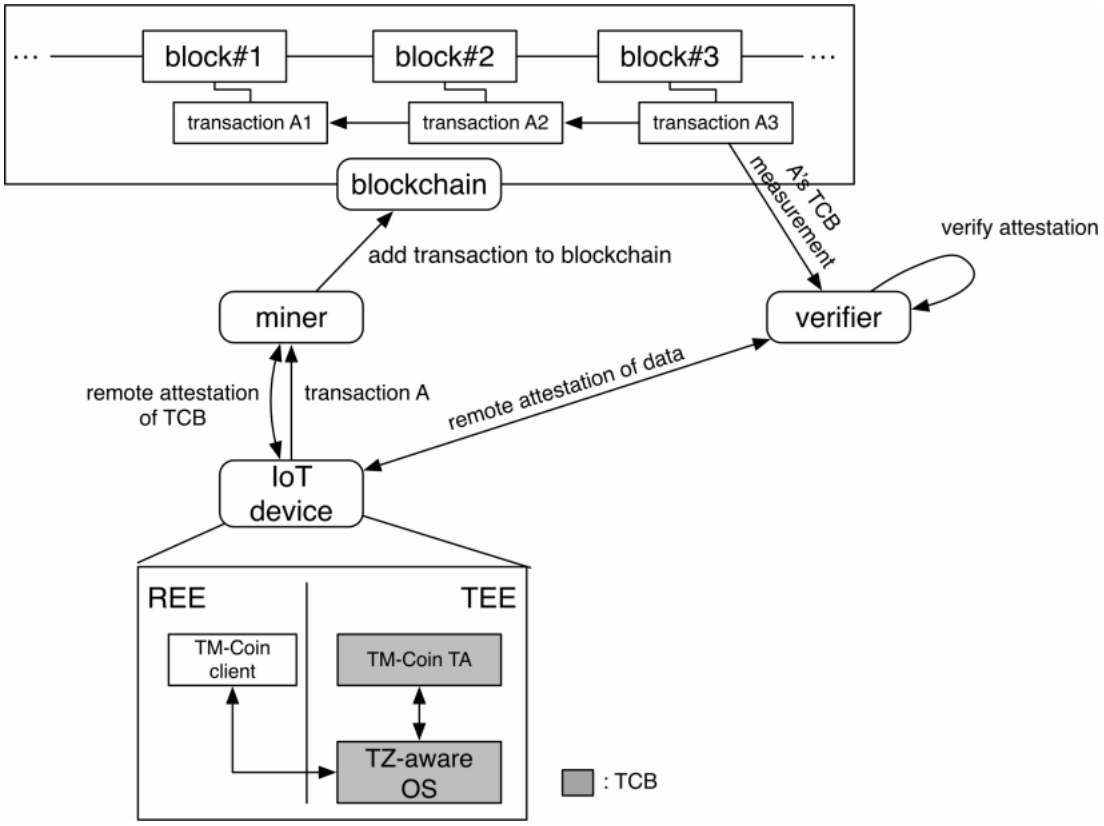

Figure 3.3.: TM Coin Architecture [41]

LegIoT. In Ledgered Trust Management Platform for IoT [42] J. Neureither et al use a graph-based representations of trust relationships. The system builds and maintains system-wide trust information by applying trust chains across the network. A trust chain is a edge of the graph and implies that the nodes that are connected by an edge trust each other. LegIoT uses indirect trust relationships via a trust graph between devices. This means that in order to establish trust, the devices do not necessarily need to access each other directly but can follow the trusted path along the edges of the graph. This approach significantly reduces the number of trust assessments in the network. The graph is calculated and maintained by a distributed ledger, this offers the benefit that all parties reach consensus about the trust graph without relying on a central authority. This additionally removes the single point of failure usually coming with the introduction of a central authority. The core functionality of LegIoT is the trust query. This query, allows every

device within the network to query its trust relationship to another device and operates as follows: A devices sends a transaction containing his own ID and the ID of another device. The smart contract receives the message and queries the stored trust graph. If a path between both devices exists, the trust score of this found path is returned. The authors of LegIoT proved that the usage of a distributed ledger as replacement for a central authority and the usage of indirect trust chains can be a very effective tool to overcome challenges of traditional remote attestation in IoT.

4. Approach

This section provides an overview of the approach used in this thesis. The section is divided into two major subsections namely blockchain as verifier and the capabilities of the prover. In the first subsection the task of the blockchain as well as the way of interacting with it is described. The functionalities of the prover are described within the second subsection. Figure 4.1 provides an architectural overview of the whole system. It consists of four main parties:

- Device A
- The prover device P
- Smart Contract
- Underlying distributed ledger

The device A can be any IoT device connected to the network. Its part is to request attestation evidence. The prover device is responsible for checking his pending requests and calculating and submitting fresh attestation evidence whenever required. All these information are stored in a distributed ledger with a smart contract on top of it. The smart contract provides the required application interface for all devices including the prover devices. But before going into details about every component's functionality we need to discuss some challenges that have to be solved. These challenges are a result of two different techniques used in this approach and solving them is crucial for the whole system. The usage of remote attestation in an IoT setting poses several challenges. Remote attestation is originally designed targeting complex machines like PCs and requires the machines to establish a synchronous end-to-end communication. In order to run remote attestation over IoT devices, there are five challenges that have to be solved:

Heterogeneity of devices The usage of unified schemes for trust establishment is complicated due to heterogeneity of IoT networks.

Large number of devices The large number of devices causes scalability issues because of required key management and the direct device to device communication of traditional RA.

Networks interconnect devices controlled by different parties IoT networks interconnect devices produced, owned and controlled by many different parties which are typically mutually mistrusting.

Difficulty to establish trust in a scalable manner Without a central authority it is difficult to establish trust in a scalable manner. But central authorities provide a target for denial of service attacks.

Asynchronous communication mediated by brokers IoT networks usually do not support synchronous communication but rely on a asynchronous communication mediated by brokers.

Sleeping IoT devices IoT devices are sleeping from time to time and therefore will not receive any requests.

The idea to master those challenges is to use a smart contract as a verification authority. This makes remote attestation publicly verifiable and visible for every network participant. In addition, it introduces trust to every network participant. Using a smart contract as a verification authority solves several problems of remote attestation in IoT but is not really straightforward to apply and raises some new challenges.

No support of synchronous communication A smart contract does not support synchronous communication.

Cannot keep secrets Remote attestation usually utilizes some sort of shared secret to establish a secure communication channel. This is not possible when communicating with a smart contract as they are public and obviously cannot store secrets.

Unable to generate random challenges Freshness is a problem because the blockchain cannot generate random challenges.

In order to solve the first challenge, we do not run attestation over an established session. This means that messages are exchanged by querying the blockchain and sending data to the smart contract using transactions. The second challenge is solved by switching to public key cryptography from pre-shared symmetric keys and use signatures to ensure authenticity of the attestation evidence. By using time stamps we introduce a weaker notion of freshness to solve the last challenge. This notion is still sufficient for attestation, as it is not a process that is run every second and requires a very precisive time stamp but is executed every few minutes or hours.

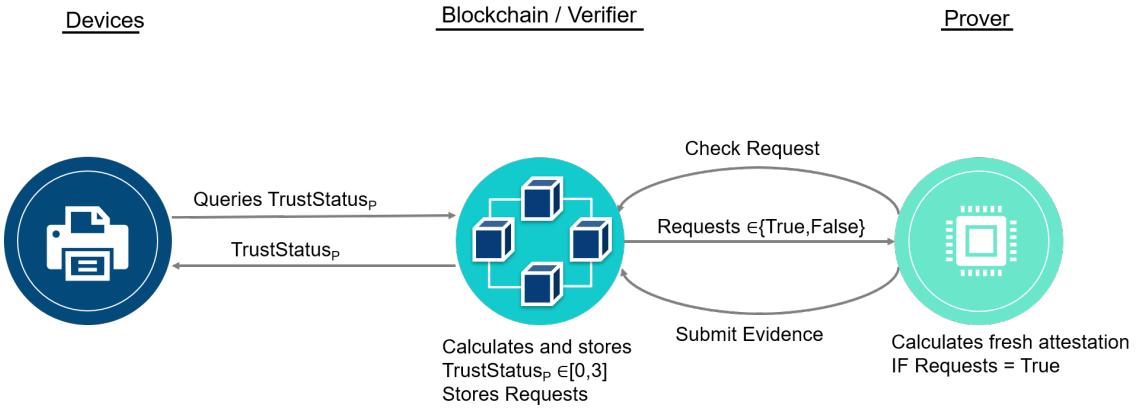

Figure 4.1.: Architectural Overview

4.1. Blockchain as Verifier

In this thesis, a blockchain in combination with a smart contract is used to verify the attestation results of a prover device. Therefore, the blockchain needs to store the expected

attestation result $Attest_P$ as well as the public key pk_P of every prover device P. This can be realized during an enrolement phase. In the following, the operation of the blockchain is described. Once device A wants to know whether a device P is malicious or not, A submits a trust query. The smart contract will take this message and query the blockchain for the latest attestation result of P. If there is no result available or the latest one is older then a certain threshold, the smart contract stores an attestation request (AR). This AR is realized by a smart contract that sets a status bit which indicates that the prover needs to calculate a fresh attestation. The request cannot be directly forwarded to P but needs to be stored. That is because in an IoT scenario devices can be sleeping at some time and therefore not reachable for downlink messages. From time to time the prover requests the smart contract and checks if it is required to calculate a new attestation result. If the AR bit is set to 1, the smart contract will tell the prover device to calculate a fresh attestation evidence and additionally add a timestamp to the reply message. This timestamp is then used as a nonce of freshness and included in the attestation evidence computed by the prover device.
Telling the prover device its current ARs as well as processing incoming attestation results and comparing it with the stored reference results are the main tasks handled by the smart contract.

4.2. The Prover Device

To realize the above explained approach, we decided to implement a hardware based attestation scheme. The biggest downside of hardware based approaches usually taken to point is the price of the hardware. But boards with secure hardware support are becoming cheaper as new technologies like ARM TrustZone emerge. The big advantage of a secure hardware architecture when implementing attestation schemes is that the scheme usually require less assumptions. Especially, the reliance on any timing assumptions that are usually part of software based attestation schemes is a huge problem when applying it to IoT networks. The communication in IoT networks is usually indirect and mediated by brokers. The reliance on any timing information is thereby not applicably to IoT networks as the transmission delay can fluctuate widely.

4.3. Communication

The following section provides a detailed overview about the communication between all parts of the system. The presented system consists of three communication parties.

The Device Submitting trust queries to get the trust status of a requested prover device.

The Prover Checking pending request, computing and submitting attestation evidence.

The Smart Contract Answering trust queries, calculating trust status, storing requests and validating attestation evidence.

Figure 4.2 provides a system sequence diagram showing the system communication step by step.

In the first step a device publishing a trust query, requesting the current trust status of a prover device P. The smart contract receives the trust query and checks if an attestation evidence is available for the requested prover device. If evidence is available, the smart contract will additionally check the timestamp of this result. The timestamp is compared with the current time of the system. If the timestamp is older then a certain threshold, the smart contract will declare the evidence as invalid, delete the evidence and store

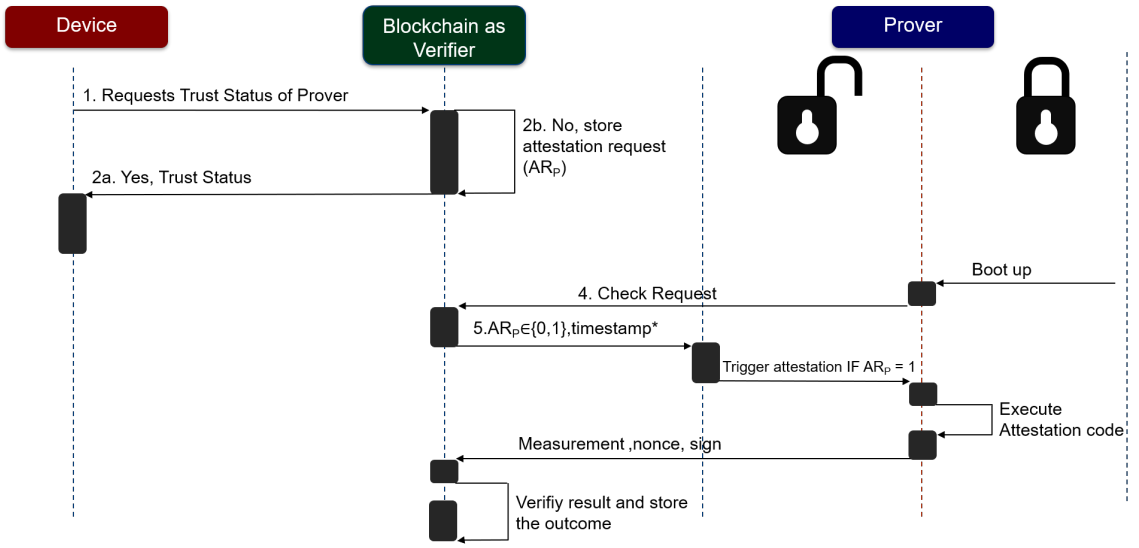

Figure 4.2.: Sequence Diagramm of whole System

a new attestation request. If the timestamp is valid, the smart contracts returns the corresponding trust status. After every reboot and after a certain amount of time, the prover device requests its pending requests AR_P from the smart contract. The smart contracts answers the request by sending AR_P. Additionally, the smart contract adds a timestamp to the reply message if $AR_P = 1$. If $AR_P = 1$ the normal world of prover device triggers attestation and passes the received timestamp to the attestation message. By that, the attestation code in secure world is executed and a new attestation result $Attest'_P$ is calculated. Once the computation is finished, the non-secure world publishes the attestation evidence to the smart contract. The attestation result $Attest'_P$ is a SHA-256 hash of the whole untrusted flash memory of the prover device P. Before transmission, the prover signes the whole message to ensure integrity of the attestation result. The smart contract first checks the signature to see if the received message is valid. If this is the case, it checks $Attest'_P$ and the timestamp. If the timestamp is valid, the smart contract calculates and stores the trust status of the prover device as well as the timestamp.

4.4. Attacker Model

In order to define an attacker model for our approach we must take two different possible attack vectors into account. One possible attack vector are attacks against the blockchain or blockchain related parties like validators. The other one combines all possible attacks against the device itself. We use Sawtooth PBFT as consensus algorithm, because of this we have to assume that at least f+1 validator nodes are trustworthy and working as expected with the total amount of nodes being $R = 3f + 1$. So a maximum of f nodes in our network can be malicious. Additionally we assume, that the attacker has limited computational power so he cannot derive any private keys from public keys and cannot forge any signatures. For the prover device we consider an adversary that can control the entire software state, code and data of the non-secure world. Every for the non-secure application readable memory region can be read by the attacker and every writable memory region can be written. The attacker can also execute every function provided by the veneer table. Every memory region protected by TrustZone cannot be accessed. We also assume that an attacker can intercept, delay and replay any messages transferred within the network. We do not consider any physical attacks against the device.

5. Implementation

The implementation is spilt up into four major components. The first component is setting up and configuring TrustZone and the memory assignment on our device. Once TrustZone is configured and all access rules are defined, the second component is developing the secure application. The secure world application is responsible for calculating the attestation result as well as building and signing the transactions, batches and batch lists required to communicate with the blockchain, in order to submit query or publish attestation evidence. The last device related component of our implementation is the normal world or non-secure application. This application initializes the wifi board and connects to a provided wifi. An additional part of the non-secure application is the MQTT Client that is running in the non-secure environment and is required to send data to and get data from the blockchain. The other part of the implementation is the blockchain and the corresponding smart contract. We use the blockchain to verify submitted attestation evidences, store attestation requests and answer check request transactions. Additionally, the blockchain has to response to trust queries and publish the trust status of the requested prover device. To realize this functionality a smart contract was developed handling all required features. The implementation of the prover device is written in standard C language, while the blockchain related component is implemented in python. But before going into details of the implementation, the components used to realized our approach are described.

5.1. The Prover Device

We chose an LPC55S69-EVK evaluation board from NXP as a platform to implement prover device. The LPC55S69 is equipped with an Arm Cortex - M33 processor with TrustZone and should be used for every prover device. The Cortex M33 provides a security foundation offering isolation to protect valuable data. TrustZone offers a system-wide approach to security by isolating critical security firmware and private information such as cryptographic keys from the rest of the application. The board provides 640KB on-chip flash memory and can run at a frequency of up to 150MHz. Due to TrustZone, every prover device has a secure hardware architecture with hardware-enforced isolation built into the CPU. This secure hardware architecture allows us to implement a hardware based attestation scheme running on the device. The device function as a

5.2. Choice of Blockchain Platform

We decided to use Sawtooth Hyperledger as the blockchain platform to realize our approach. Sawtooth is an open-source blockchain platform for building distributed ledger

applications and networks. This decision is based on several reasons. First of all, Sawtooth Hyperledger offers the support of smart contracts which is obviously required. In addition to that, Sawtooth provides separation of the core system from the application domain. This allows application developers to develop applications and specify business rules without deep knowledge of the underlying core system. Another advantage is that Sawtooth is an open source project what simplifies development. Unlike Bitcoin, Ethereum or many other blockchain platforms, Sawtooth does not a publicly deployed instance but a private one that can be deployed separately by every individual and does thereby not have a cryptocurrency and transaction fees.
In Sawtooth, the developer can define a unique transaction family for every application. The transaction family then consists of three components: an transaction processor, a data model to define stored and recorded data and a client. The core of every transaction family is the transaction processor. It is Sawtooth's equivalent to smart contract of other blockchain platforms as it defines the internal business logic of the application. Its counterpart is the client, as he is the one running the client application and sending transactions to the processor.

5.3. Implementation of the Prover Device

As already mentioned, ARM TrustZone splits the system-architecture in two separate worlds, the secure and the normal one. The secure world is used to store the private key pk_P as well as the attestation code and everything required to create the transaction messages. The normal world stores every other software, especially a client that is required to communicate with the blockchain. We use MQTT to communicate with the smart contract. The client cannot be store in secure world as it will not be remote accessible. When awake, the device handles its normal tasks and periodically asks the blockchain if there are any new unprocessed attestation requests. In case of a new AR, the client triggers the attestation process. Once the attestation process is finished it will be published and the device resumes its normal tasks.
To realize the described functionality, two different applications need to be developed, a secure and a non-secure one. Above it is described that the board is equipped with a secure hardware architecture called TrustZone. During startup the TrustZone device executes the secure application first because it is responsible for configuring the Security Attribution Unit(SAU) and Implementation specific Device Attribution Unit (IDAU) to enable memory protection, access to peripheral components and interrupts. Once the security components are configured, the secure application passes the control to the non-secure application.

5.3.1. Secure Application

5.3.1.1. Configuring TrustZone

In our implementation, the first step is to properly initialize TrustZone, enable the memory protection, configure the peripheral access and the interrupt security and establish a secure gateway between secure and non-secure application. Memory and access protection is handled by SAU and IDAU as shown in figure 5.1.

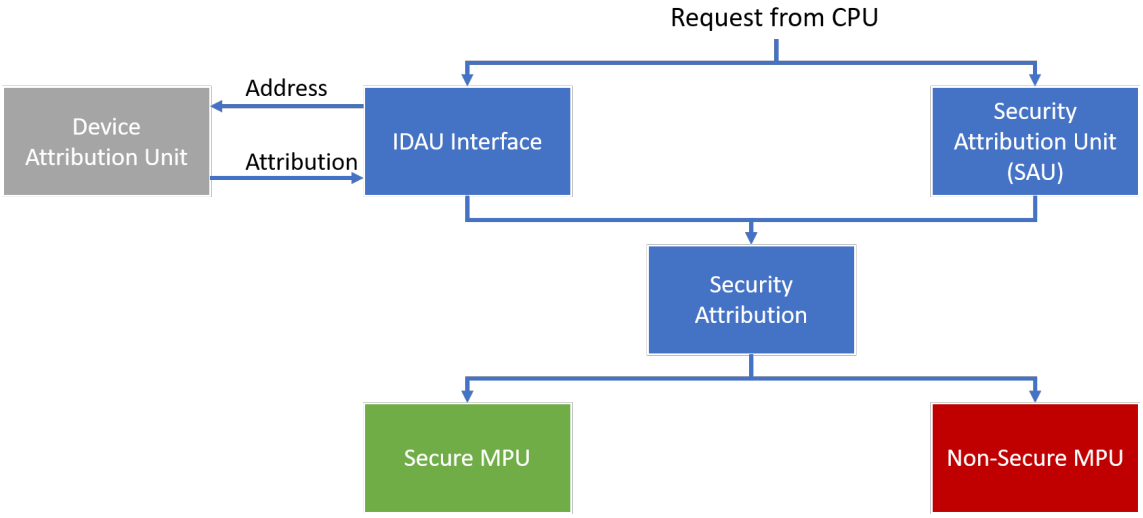

Figure 5.1.: TrustZone Access Control

Everytime the CPU requests a memory address, the security attribution of the address is checked. Every memory address has a security access configuration from IDAU and SAU. The final security attribution of a memory address is then defined by combining the security access configuration of both SAU and IDAU following the rules expressed in figure 5.2. The NXP IDAU has a rather simple design and configures every memory address with bit $28 = 0$ as non-secure and secure with bit $28 = 1$. SAU allows the developer to override the memory map of IDAU in order to define additional non-secure memory areas or a special memory area called non-secure callable (NSC). The access request is then delegated to secure or non-secure MPU. For peripheral access TrustZone has two different memory regions for non-secure an secure peripheral access. By default all non-secure peripheral memory regions are accessible for the non secure application as well as for the secure application. Secure peripheral access instead is locked for non-secure applications. TrustZone provides a Peripheral Protection Controller (PPC) and a Memory Protection Controller (MPC) controlling the accessibility of a memory page or peripheral. These controllers can also lock the non-secure peripherals so that the non-secure application cannot initialize any peripherals or handle and trigger interrupts.

By default, the secure application restricts every peripheral access as well as the interrupt handling so only the secure application has access to peripherals and can handle and trigger interrupts. For our application it is required to initialize the WiFi board in the non-secure world. To make this possible, it is necessary to change the security configuration and allow the IOCON and SYSCON controller to access the non-secure peripheral location. We faced the problem, that even when SYSCON and IOCON where configured correctly and execution of the WiFi initialization function did not trigger any BUS Faults or Memory Access Violations, the function did still not work properly from non-secure world while execution in the secure world worked perfectly fine. We realized, that the execution in non-secure environment stopped at a point where the the board was waiting on a wmi_ready event from the WiFI board. This event was never propagated to non-secure application what resulted in the function timing out and the program crashing. The problem why it was working fine in the secure environment but crashing in the non-secure was the fact, that interrupts have a security configuration in TrustZone and a secure interrupt cannot be received or handled in the non-secure environment. To solve this we had to adjust the Nested Vector Interrupt Controller (NVIC). Like peripheral access, TrustZone supports two separate interrupt vector tables one for secure and the other one for non

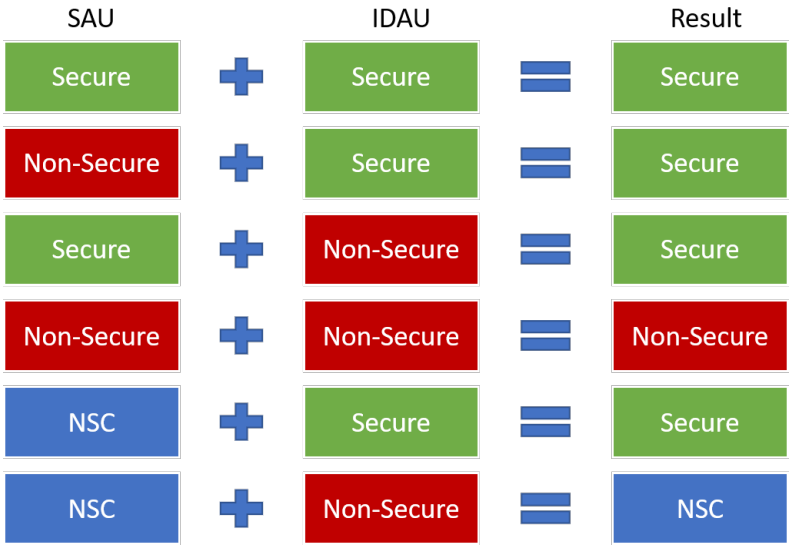

Figure 5.2.: Memory Access Rules

secure execution. The NVIC is part of the secure application and initialized during TrustZone initialization. It controls the interrupt assignment during execution and configures every interrupt as secure by default. Once the corresponding interrupts where configured right and non-secure access was allowed, the WiFi board did work as expected.
After the peripheral access and the interrupt access is configured, we started with the second step: enabling a secure communication between the non-secure and the secure application so that the non-secure application can call some functions that are specially declared. To enable a secure communication between secure and non secure application we need to implement an entry point from non-secure to secure world. Every function from secure world that is callable from non-secure must be marked with $cmse_nonsecure_entry$ function attribute. Additionally, these functions must be located in a special memory region, the so called veneer table. Figure 5.3 provides an exemplary overview of the communication flow between the secure and non-secure world in a system utilizing ARM TrustZone. The veneer table defines all functions that are located in the secure world and are callable from the non-secure world. Thereby, the veneer table functions as a defined entry point from the non-secure world into the secure one. In our project, the veneer table exports three functions, one to check the pending request for our prover device, another one to calculate the attestation evidence and the third one to submit a trust query. Implementing those functions as part of the secure application is mandatory. That is because, all three functions require access to the secure key that must not be leaked to non-secure world. The secure application can just access data and memory from the non-secure application but the non-secure application cannot. To allow the non-secure application to access the veneer table it must not be placed in secure memory nor non-secure memory but needs to be placed in a specific memory region called non-secure callable. This memory region must be defined in SAU and assigned to a memory region marked as non-secure in IDAU. Only this memory region is accessible from non-secure memory and is able to branch to secure memory. Additionally, every veneer table function must check the location of every handover parameter as well as the location of the callback to ensure no secure data is leaked or the non-secure application is able branch to secure memory.

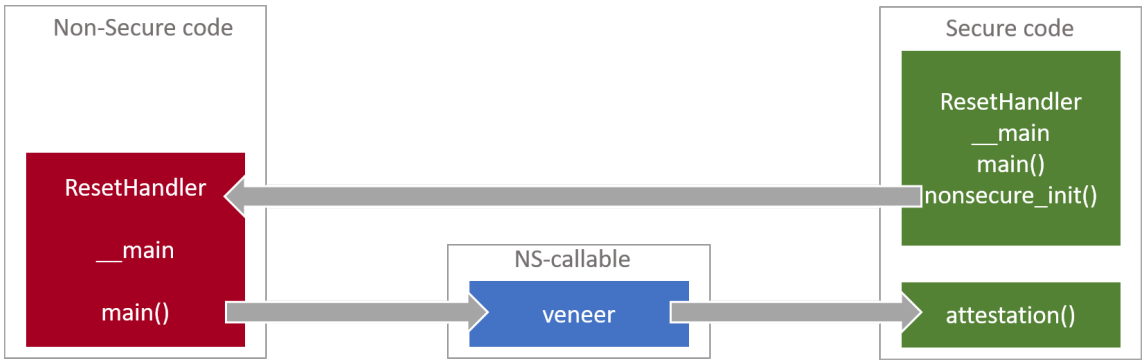

Figure 5.3.: Program Control Flow

5.3.1.2. Calculation of Attestation Evidence

The second part of the secure application is the implementation of the attestation code. The attestation code exports a function to the veneer table, that can be called from non-secure application in order to trigger the attestation process. This function requires a nonce or a timestamp as parameter as well as an output buffer to store the transaction and the size of this output buffer. Before calling the secure world the veneer table function checks the validity of the provided parameters. The output buffer must be located in non-secure memory as a whole and large enough to store the whole batch. Once all the parameters are checked and valid the function will call the attestation function in secure world. The main part of attestation is calculating the evidence. In order to calculate the attestation evidence the application iterates over every memory region classified as non-secure, reads the stored values and calculates a SHA-256 hash of all read values. In order to read the memory multiple factors must be considered. The device provides 640K of flash memory but not the whole memory needs to be attested, only the area marked as non-secure. Figure 5.4 shows our used memory assignment. The green memory area from 0x0 up to 0x40000 is secure and cannot be altered. All the other memory is assigned non-secure and has to be attested. On this device all flash operations must be done when

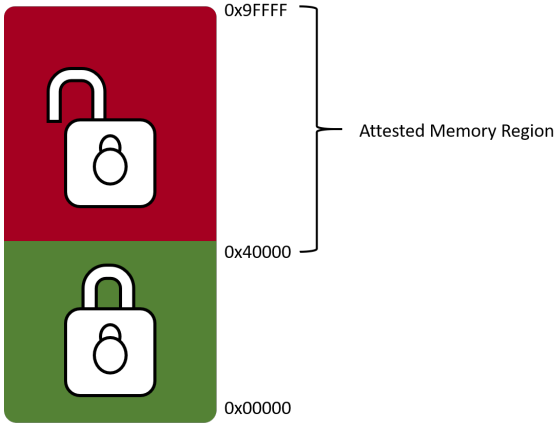

Figure 5.4.: Memory Configuration of our device

the clock is under 100MHz. So in order to make any flash operations we are required to reduce the clock speed from its maximum of 150MHz down to 100MHz or below. But even with reduced clock speed reading flash memory might fail if we just use flash_read() operation or standart C pointer. This will happen most likely through the fact, that the accessed memory location is not storing any data, but instead the memory is erased. If

the application tries to access a memory page that is erased or blank, this will trigger a hardfault what causes the program execution to fail. To solve this issue, we need to check if the accessed memory page is blank using Flash_VerifyErase() to confirm that the page is erased or not. If we have an erased page, we first write 0xFF to this memory before we read it so no malicious code can hide in erased marked sections.
We use the SHA-256 hash function to hash all values read. We do not read the whole flash memory at once to RAM but read a single memory page before updating the hash and go on to the next one. Once the calculation is finished, some additional steps are required in order to be able the submit the calculated attestation result to the blockchain.

5.3.1.3. Building the Attestation Evidence Message

As already mentioned, we use Sawtooth Hyperledger as our blockchain platform. In order to submit the attestation evidence to Sawtooth Hyperledger we need to encapsulate the measurement into a transaction and this transaction must be wrapped inside of a batch and a batch list. The overall structure of a transaction and batch includes four different parts, the transaction itself, a transaction header, the batch and a batch header. The final construction is illustrated in Figure5.5 below.

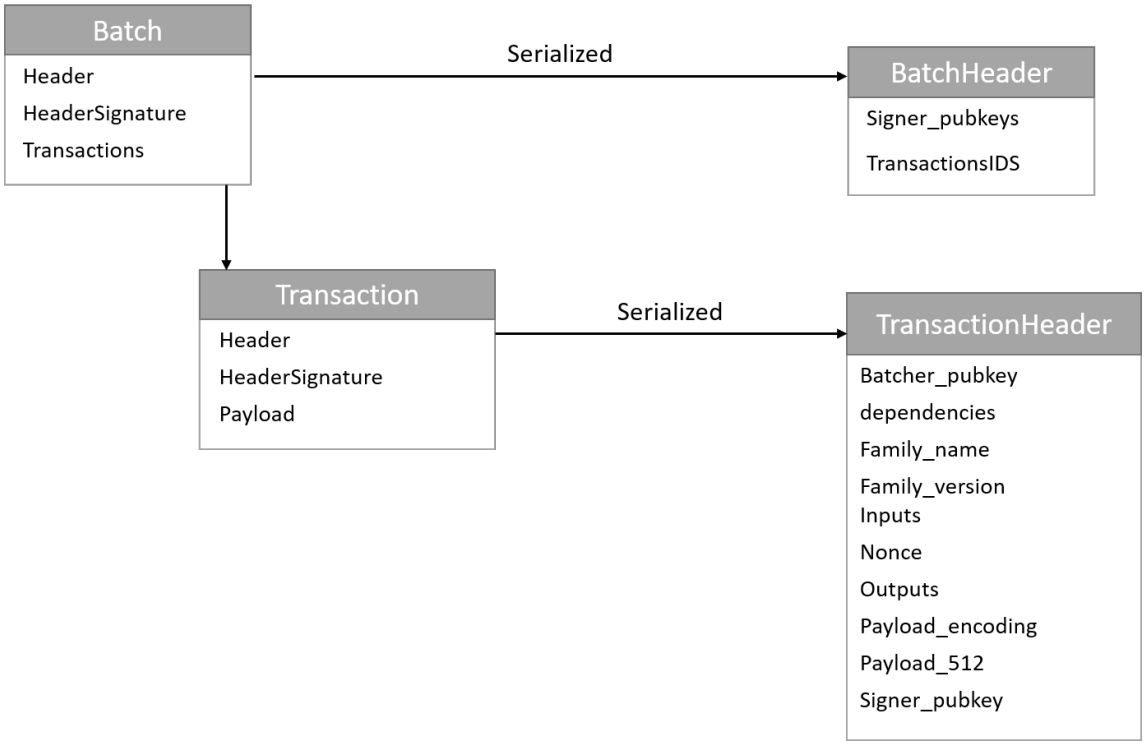

Figure 5.5.: Batch Construction in Sawtooth Hyperledger

In order to create the final batch, the first step is to create a transaction. Every transaction is serialized using protocol buffers and consist of two fields, the transaction header and the transaction itself. Every transaction contains a serialized form of its header as well as the header signature and a payload. The transaction header field is a serialized version of the TransactionHeader. This has the advantage, that the exact same bytes can be used to verify against the signature. The header is signed using the signers private key and the signature is stored in the HeaderSignature field of the transaction. The same applies for the batch apart from having its own payload which stores one up to multiple hundred transaction depending on configuration. In our application we only have one transaction that is wrapped inside of a batch. For every signature, Sawtooth relies on the use of

Elliptic Curve Cryptography (ECC). As a curve we use secp256k1, the same curve is used in Bitcoin for public key cryptography. We use libsecp256k1 libary, an optimized C libary for ECDSA signatures and secret/public key operations on curve secp256k1. This library is an open source cryptographic library that provides a small memory footprint what makes it suitable for embedded programming. Additionally, signatures are already provided in lover S format. This is required, as a validator in Sawtooth expects a 64 byte compact signature and some libraries add additional informations to the signature like recovery fields for example. This signature would be rejected by the validator or require additional parsing overhead during verification. To use this library we first need to create a secp256k1 context that stores large precomputed data tables that are expensive to construct and required for calculation. The construction of the context is very time consuming but it can be reused for multiple calculations and is necessary for both, signature verification and calculation. Once the transaction is finished, the transaction is wrapped in a batch. The batch is also signed using the private key stored in secure application. Once the batch is finished, we create a batch list from the created batch and build a json object of the batch list. This json object is then written to the provided output buffer so the completed attestation evidence message is now available to the non-secure application and can be transmitted. With this, the execution of the secure function is finished and program execution is passed back to the non-secure application.

5.3.1.4. Building the Blockchain Query

The last functionality of the secure application is querying the blockchain. As described in the approach, the device periodically publishes requests to the blockchain to check if attestation is required or not. If the queried information tells the device that attestation is required, the attestation code will be executed. In order to submit a query the device needs to build a transaction storing the request. As already mentioned above, it is mandatory to sign the transaction. And even if no attestation is calculated it is required to access the secure key in order to build a transaction. Therefore, it is necessary to make this functionality also part of the secure application. This function is also callable from non-secure application as it returns the message that needs to be transmitted. Therefore, the non-secure application has to provide a large enough buffer to store the output and passes the buffer as well as the size of it to the veneer table function. This function checks the location and size of the provided buffer and executes the buildQuery function if the provided parameters are valid. The operation of building the query message is nearly the same as building the attestation message apart from the calculation of the hash of the non-secure memory. Instead the message contains a inquiry requesting whether the device must calculate and submit a new attestation or not. This transaction is also wrapped in a batch and stored in a batch list as already described above. Finally, the batch list is serialized and returned back to the provided output buffer. Once this is done, the execution of the secure application is finished and program control flow is passed back to non-secure application.

5.3.2. Non-Secure Application

Unlike the secure application, the non secure application runs FreeRTOS a Real Time Operating System (RTOS). It is a real-time operating system for microcontrollers and microprocessors and functions as an operation system of our non-secure application. FreeRTOS provides some already available and easy to integrate libraries and files to step up the implementation process. The main responsibilities of the non-secure application are the communication with the MQTT broker and calling the functions provided by the secure application. For the communication with the secure application we use the veneer table which is implemented as part of the secure application and only the header file is

exported to the non-secure application. The non-secure application can call every function marked as non-secure callable and exposed by the veneer table like any other function of the project. The only difference is that the non-secure application only has the header file but the corresponding source file is part of the secure application and thereby inaccessible for the non-secure application like every other secure memory region and unchangeable.

5.3.2.1. Setting up the WiFi board

For the communication with the MQTT broker we need to implement a MQTT client. But before we can start with implementing the MQTT client we need to establish a WiFi connection and therefore initializing the WiFI board. As WiFi is not available on the LPC55S69 itself, we use the WiFi 10 Click board which is clicked in the mikro BUS of the board. The WiFi 10 Click board comes equipped with the QCA400x WiFi module manufactured by Qualcomm. It is just clicked in the board and will be powered by the LPC55S69 so no additional power supply is required.
To communicate with the module we use WiFi QCA400x drivers provided by the MCUXpresso SDK. Additionally we need to run FreeRTOS as it is required to properly run the WiFi. We already mentioned, that initializing the WiFi board is not straight forward, because the initialization of pins and peripherals as well as the interrupts are usually handled as part of the secure application and by default not accessible within non-secure environment. For our approach we do not want the WiFi to run as a part of the secure application but want the attestation code and all cryptographic operations to be fully separated from other program logic. Because of that we changed the access configuration during the initialization of TrustZone. Once the non-secure application boots, it at first initializes pins and clock of the system. Once this is done, the application creates a new task. This task initializes the socket required for communication before turning on the WiFi. The WiFi 10 click uses the SPI interface and GPIO to communicate with the board. To use them we had to adjust the pin multiplexing of our board to configure the pins according to the WiFi board. We use the Flexcomm SPI interface and WiFi SPI drivers from FreeRTOS to communicate with the board. Additionally we use the iot wifi library for QCA400x to power on the board and to connect to a predefined network and to get an IP address using DHCP. This library is also part of FreeRTOS and is provided by the MCUXpresso SDK. To connect to a WiFi network, one usually needs some form of cryptographic library as most WiFi networks use WPA2 to securely connect to their network. Therefore we use Mbed TLS library. Mbed TLS is an open source cryptographic library that was published by ARM and is now maintained by TrustedFirmware. FreeRTOS relies on Mbed TLS as an underlying cryptographic library. Additionally, Mbed TLS provides a small memory foodprint what makes it very suitable for embedded projects. Once the device has booted the WiFi board properly and is connected to the WiFi network, the current task creates a new one before deleting itself. It is necessary to delete every task created once they are not required anymore to free allocated resources.

5.3.2.2. Setting up the MQTT Client

After the WiFi connection is established, next step is setting up the MQTT client and connecting it to the broker. For this we delete the old task and create a new one with greater heap and stack size as the MQTT requires much more memory then the initialization process. For our application, we require a MQTT client that subscribes to a specified topic, publishes messages and respond on incoming messages sent under the subscription topic. For the MQTT we use AWS IoT MQTT which is included in FreeRTOS C SDK and thereby fits perfectly in our scenario. In order to use FreeRTOS C SDK we first

need to initialize the SDK itself before calling any of its API functions or it will trigger a hardfault. After C SDK is initialized we create a MQTT client and connect to the broker. Therefore, we use the MQTT agents provided by the MQTT library. There is an agent provided for every core functionality of the MQTT protocol. These agent function as an abstraction layer of the underlying MQTT logic and simplify the implementation as the developer does not need to care about underlying program logic like data serialization or used TCP/IP stack. Once the device established a connection to the broker, we subscribe to the specified topic and publish a check_request transaction. This message is published to the blockchain that will check for pending attestation requests and reply.

5.3.2.3. Communication with Blockchain and Secure World

In order to publish a transaction message successfully to Sawtooth, the publisher must sign the transaction using his private key. As the private key is only available to a secure world function, non-secure application has to call the veneer table function check_request first and pass the control to the secure function where the batch message is constructed and handed over to the non secure application where the message is published. This method is not trivial as the batches require a huge amount of memory space. This memory space must be allocated in the non-secure memory area as otherwise the non-secure application cannot access it. In order to allocate that much space we need to increase the maximum heap size in the FreeRTOS configuration as well as the heap configuration in the MCU settings or drastically increase the maximum stack size of the task ,because every task is created with a maximum task size, as well as the stack configuration in MCU settings.
Once the check request message was published the device will stay in non-secure world until the client receives a reply message. Once the devices receives this reply message or any other message under the specified topic, the MQTT client will execute the defined callback function. This function will take the received message and call a secure world function that checks the received message and act appropriately. The reply message to a check request tells the client if a fresh attestation result is required or not. If this is the case, the message also contains a timestamp. If not, the timestamp will be empty. Once the device receives an attestation request, the attestation function takes the timestamp and calculates a fresh attestation evidence, creates the batch and passes the data back to the non-secure application, where the MQTT client publishes the attestation evidence. Nearly the same applies for trust queries. If the device needs to check the trust status of an other device within the network, it has to query the blockchain in order to receive the stored trust status.

5.4. Implementation of Transaction Families

In order to implement our functionality we had to implement two transaction families. An administrator transaction family and an attestation transaction family. Both transaction families are implemented in Python. We used the Sawtooth Python SDK in our implementation as it provides some very useful components that simplified developing application for Sawtooth. Additionally, we used google protocol buffer as Sawtooth relies on this to serialize transactions and batches. Google protocol buffer is a fast an simple way to serialize structured data. We had to define a proto file for every message type used in our transactions. These proto files were then used to generate a special source code using protocol buffer compiler for Python. These files are then used in our implementation to serialize and deserialize payload data.

5.4.1. Implementation of Administrator Transaction Family

Once the blockchain is running, multiple databases storing necessary data for the whole system must be initialized. These databases are important as they contain relevant in-

Device Identity	Time Function	Measurement	xmin	xmax	Reliability Score
073B	-0.003333333*x + 1.2	7A09AB47D4	60	120	0.7
998D	-0.001666667*x + 2	C956F3FE76	600	1200	0.9
0CD1	-0.0006666667*x + 1.2	69F3BCD492	300	600	0.8

Table 5.1.: Device Database

formation like deviceIDs of legitimate users and expected attestation evidences for the devices. For our approach two different databases are required. The first one is the device database displayed in 5.1. This database contains the unique device identifier of every legitimate user of the network. Additionally, it includes a time function, the expected integrity measurement a lower and upper threshold xmin and xmax and the reliability score. The value xmin is the threshold before an evidence starts to decay. Once a submitted evidence is older then xmin the trust store will reduce over time following the defined function. Whenever the trust score is below the minimum reliability or older then xmax, the attestation is considered invalid. The exact procedure is explained in the next section. The other one defines the system parameters used in the blockchain. All these information are require to validate a submitted evidence and answer trust queries. The administrator transaction processor is used to upload these administrative databases to the global state of the blockchain. This step must take place before starting any other transaction, as the data is relevant for other transaction families especially the attestation transaction family in order to operate properly. In order to upload these database the administrator client is used. This client uploads the data by sending transactions containing the corresponding data. Once the administrative databases are initialized and the data was successfully loaded to global state, the attestation transaction processor can be used.

5.4.2. Implementation of Attestation Transaction Family

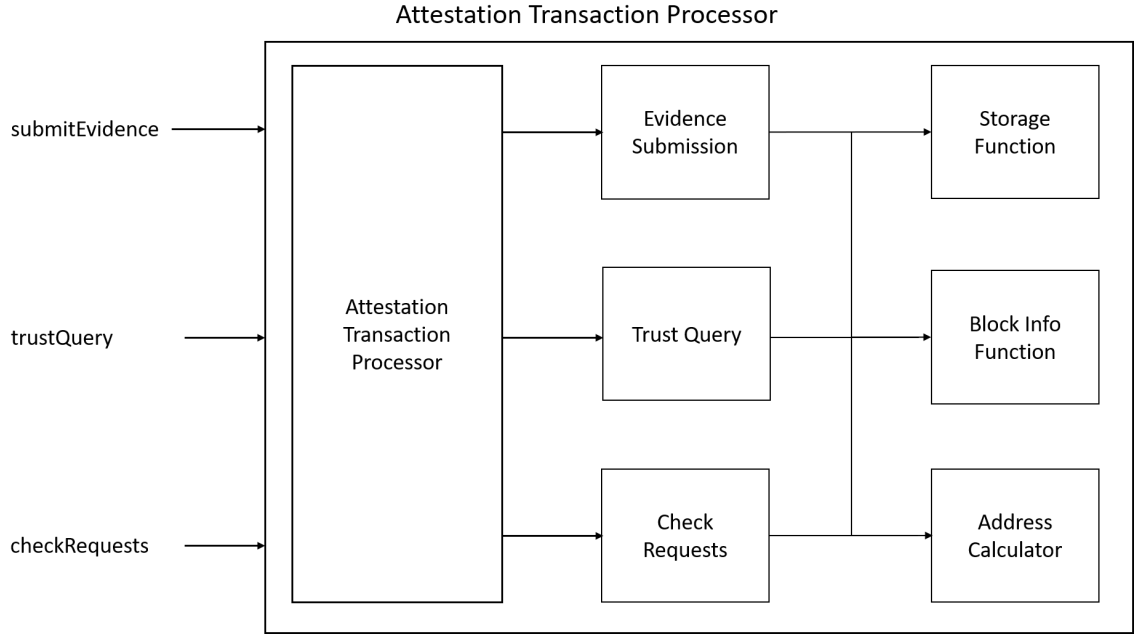

Figure 5.6.: Modules involved in the Attestation Processor

The attestation transaction family is the core component of our system. The transaction processor takes all incoming transactions. The payload of every transaction is CBOR

encoded and must include two pieces of information, the action and the actual payload. Because of this, the payload of every received transaction is decoded and split into action and actual inner payload. Depending on the action, the processor will call the appropriate action module and pass the current blockchain state, the transaction payload and the public key of the signer to the function. As displayed in figure 5.6, we have defined three different action modules in our system and three additional helper modules. Before going to the main modules, the action modules we briefly describe the helper modules:

Storage Functions This function queries the global state of the blockchain and is used during validation to get a list of legitimate device IDs.

Block Info Functions Used to read block related information like the ID of a block or the timestamp of a block with given ID.

Address Calculator The purpose of this module is to calculate the addresses in order to store data or read already stored data.

The first action is to submit attestation evidence, the second one to query the trust status of a prover device and the last one to check pending requests. These actions are required in order to realize the functionality of the proposed approach and are implemented as own python scripts. In the following, all three functions are explained starting with the trust query.
The purpose of the trust query is to query the trust status of the requested prover device. In order to do so, the payload must contain the deviceID of the device sending the trust query as well as the ID of the prover device and a minimum reliability value. This is checked in the first part of the function. In addition to that we require both IDs to be assigned to legitimate peers. This means that the used device IDs must be stored in the device ID database uploaded from admin client in beforehand. Therefore, we validate the received IDs using the storage functions module. This module queries the global state of the blockchain and checks if the deviceIDs are part of the deviceDB. Once this is done, we pass the proverID to the address calculation module where the address of the stored attestation result as well as the address of the associated timestamp is calculated. The next step is to read the stored values of these addresses. The last step before assembling the reply message is to check the timestamp and calculate the trust status of the prover device. The pseudocode in 19 displayes the calculation of the trust status for a given proverID. We use the block info module in this step in order to get the timestamp of the last block submitted. At first we check if there is any evidence and timestamp for this prover device in the system. In the case that no attestation result or timestamp is stored, a check request for the proverID is set and attestation pending is return, telling the requesting device that no attestation evidence of the requested prover device is currently available in the system and it has to wait until an evidence is submitted. If both values are available, the execution continues and checks if there is a strike registered for this prover device. The device will get a strike whenever a wrong evidence is submitted what means that the device is malicious. If the device has at least one strike, the function will return untrusted and stores a new attestation request for the proverID. When there is evidence available and no strike registered, the trust status is calculated using the timestamp. We registered a minimum and maximum value as well as a decay function required for the computation of the trust score. At first the age of the submission is calculated. This age is stored in the variable x and is the difference between the timestamp of the latest block submitted and the evidence timestamp. If x is below xmin, this means the evidence is fresher then the defined threshold and we consider the device trusted. In the case that the evidence is older then xmin but fresher then xmax the defined decay function determines the trust score. A decay function is defined for every proverID and defines the remaining trust score. We consider the device as trusted as long as the remaining trust score is above

Algorithm 1 Calculation of Trust Status

```
Require: proverID
    timestamp ← getTimestamp(proverID)
    trustStatus ← getTrustStatus(proverID)
    currentTime ← getLastBlockTime()
    if (timestamp ≠ ∅ ∨ trustStatus ≠ ∅) then
      setRequest(proverID)
      return attestationPending (2)
    else if proverID.Strikes ≥ 1 then
      setRequest(proverID)
      return untrusted (3)
    else
      x = currentTime - timestamp
      if x ≤ proverID.xMin then
        return trusted (1)
      else if x ≥ proverID.xMin ∧ x ≤ proverID.xMax then
        decayedTrustStatus ← calculateDecay(proverID)
        return decayedTrustStatus [0:1]
      else
        setRequest(proverID)
        deleteEvidence(proverID)
        return attestationPending (2)
      end if
    end if
```

the minimum reliability defined in trust query payload. If the evidence is older then xmax, we consider it outdated, delete the stored evidence as it is not longer valid and store an attestation request for the proverID. The function will return a three if the prover device is untrusted, two if attestation is pending, one if the device is trusted and a value between one and zero if the evidence is still valid but decaying.
The next functionality explained is check request. This one is rather simple as the only required functionality is to check the pending request for a prover device submitting this request. The payload message only has to contain the device ID of the device checking its pending request. The ID is then validated in the same way as described in the trust query, before pending requests are read and returned. We additionally check if the device has strikes. If it is the case, we set pending request true even if no actual requests are stored to get a fresh attestation evidence of the device. This has the benefit, that the system automatically checks if the malicious device is still malicious or if it was healed. If there are pending requests for prover device, the return message will additionally contain the block ID of the latest block submitted to the blockchain. This timestamp is then later included in the evidence submission message.
The final functionality is evidence submission. The evidence submission message contains following information, the device ID of the prover, the measurement itself and the timestamp. Before calculating the trust status, the legitimacy of the prover device is checked. Additionally, the timestamp of the evidence message is checked. This is done to detect messages that did not arrive in time and are therefore already invalid. With this check, we prevent an unnecessary calculation of a trust status that will be deleted by the trust query anyways. If the message passed both checks, the transaction processor queries the blockchain for the expected attestation measurement and compares it with the received one. If both values equal, the trust status of the device is set to one, meaning the device is considered trusted otherwise the device will get a strike, meaning that it is considered

malicious. In the case that the received evidence did not pass the timestamp check, we will set the trust status to two, meaning attestation is pending and additionally set the check request bit to inform the prover that a fresh attestation evidence is required.

5.4.3. Other used Transaction Families

In addition to the two above described transaction families, we use two transaction families that are provided by Sawtooth. The first one it the identity transaction familiy. This transaction family deal with the permissioning of the participants. The purpose of the identity transaction family is to define roles and policies for specific user groups. In order to make changes in the family, a validator need to set keys that are allowed to do so. After that, roles are created and policies are assigned to those roles. An example role might be the "transactor" and refers to a policy that controls which entity is allowed to submit batches and transaction to the network.
The other used transaction family is the settings family. This one is also provided by Sawtooth and is used to administrate the systems settings for the entire blockchain network. In order to change global settings, a voting-based concept is introduced. This concept enables permitted participants to decide on changes in the global policy. The changes are executed once a certain threshold of votes is reached, resulting in a change of the global settings.

6. Evaluation

6.1. Performance Evaluation

Action	1 Transaction	10 Transactions	100 Transactions
Check Requests	9	55	483
Submit Evidence	41	252	2496
Trust Query	26	173	1559

Table 6.1.: Execution Time in Milliseconds

The performance of the whole system is mainly dependent on two factors. On the one hand, the performance of the attestation scheme and on the other hand, the rate with which Sawtooth Hyperledger can handle incoming transactions. The performance of both parts is highly dependent on the capabilities of the underlying hardware components. The performance of Sawtooth is additionally dependant on multiple other factors like the network delay, the performance of the validator nodes and the amount of validator nodes in our system. In order to eliminate the varying network delay we decided not to evaluate the system as a whole but every component separately to get a more precisive result of the capability of our approach. To evaluate the performance of Sawtooth, we measured the execution time of all three implemented actions for a single transaction, ten and hundred transactions. The transactions where directly submitted to the transaction processor to minimize network delay. The results of our measurements are displayed in table 6.1. It is clear to see that the evidence submission requires the longest while check requests is the fastest. This is obviously resulting from the computational effort required for this action. We see that even handling 100 transactions at a time is very fast. This shows the feasibility of this approach for larger networks.

6.2. Security Evaluation

In this security evaluation we are going to show that our approach is able to detect a malicious prover device and additionally provides sufficient counter measures against common attacks targeting remote attestation schemes in IoT. In the definition of the attacker model we considered two different possible attackers, attacker targeting the device itself excluding hardware attacks and malicious entities within our network. For our security evaluation we divided it in two sections with the first one focusing on an attacker that

infiltrated the prover device. This means that the attacker has access to every unprotected memory region and can execute every function that is executable from non-secure memory. While the second section considers an attacker within the network that might have also access to non-secure memory of prover device but first of all the attacker was able to infect some device within the network.

6.2.1. Attacker located on Prover Device

An attacker that managed to infect a prover device is able to fully control the non-secure application. This includes writing to and reading from every accessible memory space, using every provided function especially the MQTT functions and the veneer table functions as well as gathering every information that is available to in the non-secure environment. In our setting, a malicious prover device can gather full control over the MQTT client. The attacker can exploit this for example and use the MQTT client to send messages to other network participants or publish transactions to the blockchain. In order to generate a valid transaction message a prover device must sign the transaction using his private key. This key is only available in secure environment which is protected by ARM TrustZone and therefore not accessible. So even if a malicious prover device tries to send a spoofed message, the recipient just needs to check the signature and will detect that this message is not created by a trusted communication party.
There is a possibility for the attacker to get a valid transaction by using the veneer functions. These functions will return the non-secure environment either a trust query or an attestation evidence message. These messages are valid because they are generated and signed by the secure application and can be generated and published by any malicious device. The attacker does not have any advantage by generating either one of these messages as he can only query data stored in the blockchain or submit his own attestation evidence. Calling the attestation function and publishing its result would additionally result in the attacker revealing himself, as the attestation code calculated a hash over the whole non-secure memory and publishes this as part of the attestation evidence batch. Once the blockchain receives this package and the smart contract compares the provided attestation result with the expected value, it will detect the infected device and store the result accessible for every other network participant making them aware that the prover device is not trusted anymore. Usually after every boot the device will check with the blockchain if any pending attestation requests were received. If a request was received, the prover will calculate the attestation evidence and publish it causing a malicious prover to be revealed. As the non-secure application is responsible for publishing the request in first hand and the attestation evidence later on, a malicious application can prevent the request or the attestation evidence from being sent. With this strategy a malicious device can try to hide its malware infection by just not calculating and publishing any attestation evidence ever again. This can be a good strategy especially if the last evidence submitted did not contain any malware and thereby the prover is considered trusted. In order to mitigate this, every attestation evidence has a time stamp. This time stamp is the time stamp of the latest block submitted at the time and is sent to the prover device as an answer to check attestation request. This times stamp is stored for every attestation evidence submitted and checked everytime the attestation result is queried. If the time stamp of latest attestation evidence submitted is older then a certain threshold, the prover device that submitted this evidence is considered not trusted until a fresh attestation evidence is submitted. This has the benefit that once a prover device turns malicious and stops submitting attestation evidences the system automatically marks the device as attestation pending and make every other device aware that this device did not submit any evidence over a longer period of time. Obviously this security measure is highly dependent on the actual threshold as a more generous threshold will result in a longer period of time until

a malicious device is marked as attestation pending and thereby can be detected by other devices. So for security critical applications and devices it is recommended to use a low threshold to detect malicious devices faster with the downside that devices must submit attestation evidences more often what results in an increased security overhead.
With the possibility of generating valid transaction messages, the malicious prover device can try to impersonate another device and publish spoofed transactions. Sawtooth stores the data of every received and validated transaction to a specified address. This address is calculated using an ID that is transmitted as part of every transaction. Thereby it can be interesting for a malicious device to use other IDs within their transactions in order to hide his own identity or impersonate another device. Such an attack is not possible in our system as the proverID is stored in secure memory and included in every transaction. This is done by the secure application and cannot be manipulated by a malicious non-secure application. Even trying to change the proverID once the finished batch is handed over to non-secure application is not possible. Because of the signature, every change to the transaction message will make the signature and thereby also the transaction invalid.
Another attack vector that must be considered summarizes attacks that cause denial of service. There are already some attacks proposed using remote attestation schemes to cause denial of service on a prover device by excessive execution of the attestation code. Causing denial of service from non-secure environment, by executing the attestation code is also possible in our scenario. A malicious non-secure environment can use the veneer table functions to force the secure application to execute the attestation code over and over again and prevent the board from doing anything else. However, even if this attack is possible for an attacker that controls the non-secure environment we do not consider it as reasonable. That is because first of all, an attacker running this attack would run a denial of service attack against himself. The second point why this attack is not reasonable is because the non-secure environment could run a denial of service attack much easier then with excessive execution of attestation code by just turning of the board or the WiFi. This means that even if denial of service attacks are still possible against a device that is infected with malware it is not really profitable for the attacker.
The last attacks that must be evaluated are attacks targeting the WiFi and the WiFi board. As already mentioned in the implementation section, we run and initialize the WiFi board as a part of the non-secure application. This allows the attacker to power up and down the WiFI as well as connecting to a WiFi. The WiFi credentials are required to connect to the WiFi network and are thereby stored as part of the non-secure application leaving the possibility for an attacker to leak the WiFi credentials. For most industrial IoT scenarios this is not a huge concern as the WiFi network has additional protection. If this is not the case, an attacker infecting a device can leak the WiFi credential. This enables the possibility for other malicious devices located within the reach of the WiFi to join the network. This is only possible if the attacker is able to locate the additional devices within reach of the WiFi. And even if the attacker has access to the location and can sneak malicious devices into the network this devices can not act as trusted communication parties because the blockchain does not have their public key stored. This will cause the validator to dismiss every transaction that is published by such a device because the signature check fails. Additionally, every normal device that requests a trust status of such a device will receive a not trusted as their is no value available what causes the smart contract to consider the device as not trustworthy.

6.2.2. Attacker located outside of Prover Device

In this section we are going to evaluate an attacker model where the attacker lies within the network, but did not infect the prover device itself but some other network device.
This attacker is able to eavesdrop and intercept communication between devices or prover

and blockchain, record an replay intercepted messages, submit attestation requests and can even be part of the validation process.

Evesdropping. An attacker eavesdropping the network communication is able to listen to transaction messages published by devices. These messages are not encrypted and the attacker can read all the included information. Transaction messages are publicly verifiable and are thereby readable by every network participant and do not contain any sensitive data. This has the result that an attacker eavesdropping does not gather any new and private informations that are not available to every network participant anyway.

Replay Attacks. A more advanced attacker can try to intercept communication, record and replay intercepted messages. As already explained above, an attacker can eavesdrop communication easily and thereby also intercept communication and record the messages. This will allow the attacker to delay messages sent from possible trusted parties. The attacker could for example intercept an attestation evidence AE published by an genuine prover device P before infiltrating the device P and turning it malicious. This message AE can then be stored by the attacker and published at a later stage. With this attack, the attacker is able to get a valid attestation evidence representing the device trusted while being malicious. A countermeasure to this attack is the time stamp of the attestation evidence. Once the transaction message comes trying to convince the system that P is trusted the smart contract will check the timestamp of the evidence and the transaction is declared invalid once the time stamp is too old. This will result in the transaction being dismissed and the smart contract setting the check request bit as well as setting the trust status to attestation pending so every device querying this value knows that fresh evidence is missing.

Denial of Service. Another possible attack vector against honest devices is again denial of service. A malicious device can publish massive requests for attestation evidence and make the device calculate attestation over and over. In our system, these messages are not sent directly from device to device, but one device needs to submit a request to the blockchain and the prover device will query this value after some time. Thereby, the attacker has to naturally wait until the prover device queries the submitted request and cannot send a massive amount of requests directly to the device like in many other remote attestation schemes in IoT.

As DoS attacks against the device are not possible, an attacker could try to run an DoS attack against the blockchain and its validator nodes. This is a concern in Sawtooth as unlike most other blockchain platforms, Sawtooth does not provide its own cryptocurrency and thereby also no transaction fees. In order to alleviate this issue, the smart contract provides a penalty function. This function can provide a penalty and even timeout devices that are sending too many transactions in order to prevent such attacks.

Predicting Timestamp. The last attacker we need to evaluate, is an attacker that tries to predict the timestamp of upcoming blocks in the system. We use the block ID as a nonce of freshness for the attestation evidence. An attacker that can guess the upcoming blocks several minutes or hours in advance can request attestation evidence from a prover device and provide a spoofed timestamp that lies in the future. This will give the attacker the opportunity to get an unaltered attestation evidence in advance before infecting the memory of the prover device. Usually this evidence will be timed out quickly by the system but as the timestamp lies in the future, the attacker will have some additional time. In large networks it is nearly impossible to reliably influence the block generation and the order in which transactions are submitted. In order to manipulate this process, an attacker is required to control half of the devices within the network or at least large parts of it and is thereby not considered in this thesis.

7. Conclusion

> The internet is no longer a web that we connect to. Instead, it's a computerized, networked, and interconnected world that we live in. This is the future, and what we're calling the Internet of Things.

This quote of Bruce Schneider emphasizes the future importance of IoT. Billions of IoT devices are already deployed with numbers rapidly increasing. With a further expansion of IoT networks, security within the networks becomes more and more important. The restricted capabilities of the devices as well as the heterogeneity and large number of devices within an IoT network causes most traditional security features as well as trust establishment methods to suffer from some major downfalls when being applied to IoT devices. In this thesis a new approach to establish and maintain trust between different IoT devices is presented. The proposed system design uses distributed ledger to store attestation requests, answer trust queries and validate attestation evidences. In the presented approach, a distributed ledger platform and a smart contract is used to replace the central verification authority used in most remote attestation schemes. With this step, several flaws usually resulting from the reliance of systems on a central authority are solved. In addition to that, the overall trust is increased as every piece of information is public. The possibility of evaluating attestation evidences and storing their outcome without relying on central servers is a important step to establish and maintain trust in a scalable manner in IoT.
We evaluated that the proposed architecture is able to solve every challenge formulated at the beginning of this thesis. Additionally, it was presented that our approach reliably protects against the formulated attacker model and prevents common attacks like denial of service. Additionally, a prove of concept implementation of the overall system is provided. This implementation contains the implementation of Sawtooth transaction processors as well as the attestation code and a MQTT client on the LPC55S69-EVK. We conclude that the presented concept solves several issues regarding remote attestation in IoT and enables trust establishment to large and heterogeneous networks.
We are glad that we were able to fully implement the whole approach on Sawtooth as well as providing a implementation for a prover device. Nonetheless, there are some interesting challenges for future work in order to further advance the system. The first step could be to deploy a real world setup with multiple different devices. Implementing other attestation schemes for other devices with different hardware and adding these devices to the system is also a future step to improve the system and bring the concept closer to a real world deployment.
Another interesting aspect for future research could be the consideration whether and how

it is possible to port this concept to a public blockchain platform like Ethereum.

List of Figures

List of Tables

Listings

Bibliography

[1] Knud Lasse Lueth, "State of the iot 2018: Number of iot devices now at 7b – market accelerating," 2018.

[2] I. Krontiris, T. Giannetsos, and T. Dimitriou, "Launching a sinkhole attack in wireless sensor networks; the intruder side," in *2008 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, IEEE, 10/12/2008 - 10/14/2008.

[3] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Communications Surveys & Tutorials*, 2016.

[4] Sam Kottler, "February 28th ddos incident report," 2018.

[5] Trusted Computing Group, "Tpm main specification version 1.2," 2012.

[6] F. Brasser, K. B. Rasmussen, A.-R. Sadeghi, and G. Tsudik, "Remote attestation for low-end embedded devices," in *Proceedings of the 53rd Annual Design Automation Conference*, (New York, NY), ACM, 2016.

[7] W. Meng, T. Jiang, and J. Ge, "Dynamic swarm attestation with malicious devices identification," *IEEE Access*, vol. 6, pp. 50003–50013, 2018.

[8] M. Ammar and B. Crispo, "Wise: A light weight intelligent swarm attestation scheme for the internet of things," *ACM Trans. Internet Things*, vol. 1, June 2020.

[9] Satoshi Nakamoto, "bitcoin,"

[10] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab," in *Financial Cryptography and Data Security* (J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, eds.), (Berlin, Heidelberg), pp. 79–94, Springer Berlin Heidelberg, 2016.

[11] A. Bahga and V. K. Madisetti, "Blockchain platform for industrial internet of things," *Journal of Software Engineering and Applications*, vol. 09, no. 10, pp. 533–546, 2016.

[12] M. Pilkington, *Blockchain technology: principles and applications*. Cheltenham, UK: Edward Elgar Publishing, 2016.

[13] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," CCS '16, (New York, NY, USA), p. 3–16, Association for Computing Machinery, 2016.

[14] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen, and E. Dutkiewicz, "Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities," *IEEE Access*, vol. 7, pp. 85727–85745, 2019.

[15] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.

[16] M. Giancaspro, "Is a 'smart contract' really a smart idea? insights from a legal perspective," *Computer Law & Security Review*, vol. 33, no. 6, pp. 825–835, 2017.

[17] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab," in *International conference on financial cryptography and data security*, pp. 79–94, Springer, 2016.

[18] W. Zou, D. Lo, P. S. Kochhar, X. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu, "Smart contract development: Challenges and opportunities," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[19] K. Olson, M. Bowman, J. Mitchell, S. Amundson, D. Middleton, and C. Montgomery, "Sawtooth: An introduction," *The Linux Foundation*, 2018.

[20] B. L. Miguel Castro, "Practical byzantine fault tolerance," 1999.

[21] F. McKeen, "Innovative instructions and software model for isolated execution,"

[22] G. F. Jerome Azema, "M-shield whitepaper," 2008.

[23] ARM Limited, "Arm security technology building a secure system using trustzone technology,"

[24] "ftpm: A firmware-based tpm 2.0 implementation."

[25] Diya Souba, "Trustzone for armv8-m."

[26] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "Swatt: software-based attestation for embedded devices," in *2004 Symposium on Security and Privacy*, (Los Alamitos, Calif), IEEE Computer Society, 2004.

[27] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, "Smart: secure and minimal architecture for (establishing dynamic) root of trust.," in *Ndss*, vol. 12, pp. 1–15, 2012.

[28] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "Trustlite," in *Proceedings of the 9th ACM European Conference on Computer Systems ; April 13 - 16, 2014, Amsterdam, Netherlands* (D. Bultermann, H. Bos, A. Rowstron, and P. Druschel, eds.), (New York, NY), pp. 1–14, ACM, 2014.

[29] X. Carpent, N. Rattanavipanon, and G. Tsudik, "Remote attestation of iot devices via smarm: Shuffled measurements against roving malware," in *Proceedings of the 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, (Piscataway, NJ), pp. 9–16, IEEE, 2018.

[30] A. Ibrahim, A.-R. Sadeghi, and G. Tsudik, "Healed: Healing & attestation for low-end embedded devices," in *International Conference on Financial Cryptography and Data Security*, pp. 627–645, Springer, 2019.

[31] B. et al., "Tytan," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, (Piscataway, NJ), IEEE, 2015.

[32] K. Eldefrawy, N. Rattanavipanon, and G. Tsudik, "Hydra," in *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (G. Noubir, ed.), (New York), pp. 99–110, Association for Computing Machinery, July 2017.

[33] M. N. Aman, M. H. Basheer, S. Dash, J. W. Wong, J. Xu, H. W. Lim, and B. Sikdar, "Hatt: Hybrid remote attestation for the internet of things with high availability," *IEEE Internet of Things Journal*, p. 1, 2020.

[34] M. N. Aman and B. Sikdar, "Att-auth: A hybrid protocol for industrial iot attestation with authentication," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 5119–5131, 2018.

[35] A.-R. Sadeghi, *Towards Hardware-Intrinsic Security: Foundations and Practice*. Information Security and Cryptography, Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2010.

[36] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, "VRASED: A verified hardware/software co-design for remote attestation," in *28th USENIX Security Symposium (USENIX Security 19)*, USENIX Association, Aug. 2019.

[37] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, and G. Tsudik, "A verified architecture for proofs of execution on remote devices under full software compromise."

[38] Ivan De Oliveira Nunes, Karim Eldefrawy, Norrathep Rattanavipanon, and and Gene Tsudik, "Pure: Using verified remote attestation to obtain proofs of update, reset and erasure in low-end embedded systems,"

[39] F. Brasser, K. B. Rasmussen, A. Sadeghi, and G. Tsudik, "Remote attestation for low-end embedded devices: The prover's perspective," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2016.

[40] M. Bampatsikos, C. Ntantogian, C. Xenakis, and S. C. A. Thomopoulos, "Barrett blockchain regulated remote attestation," in *Proceedings, 2019 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WI 2019 companion)* (P. Barnaghi, ed.), (New York, New York), pp. 256–262, The Association for Computing Machinery, 2019.

[41] J. Park and K. Kim, "Tm-coin: Trustworthy management of tcb measurements in iot," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, (Piscataway, NJ), pp. 654–659, IEEE, 2017.

[42] J. N. et al, "Legiot: Ledgered trust management platform for iot,"

Appendix

A. First Appendix Section

ein Bild

Figure A.1.: A figure

...