An Open Source Attestation Mechanism for Trusted Execution Environments on TrustZone devices

Frank Nijeboer (s2011972)
f.j.nijeboer@student.utwente.nl

September 5, 2023

Abstract As technology advances, secure computing environments become increasingly important. Arm TrustZone, a hardware-based security extension for the Cortex family of computer chips, offers a secure environment for running sensitive applications. However, attesting the security and functionality of TrustZone-based applications remains an ongoing challenge, unlike Intel SGX, which has reliable open-source implementations for attestation. In this proposed research, we aim to investigate the state of attestation mechanisms for Arm TrustZone and propose a new protocol that satisfies security and functional requirements. The research involves a thorough comparison of existing attestation mechanisms and open-source implementations, followed by the design of a new protocol, and the evaluation of its performance and formal security. The outcome of this study aims to provide a practical and efficient solution for attesting the security and functionality of applications running inside TrustZone on Arm SoCs with Cortex-A chips.

Keywords attestation · TrustZone · Arm · security · cryptographic protocols

1 Introduction

In today's digital era, the rise of connected devices and the internet of things (IoT) has led to an explosion of data being generated at the edge of the network. This has resulted in a significant shift towards distributed architectures that leverage the power of Edge Computing. The benefits of Edge Computing are numerous, including faster data processing, reduced latency, and improved data privacy [1]. As a result, the field of Edge Computing, which aims to bring computing power to the "edge" of the network through small devices, is expected to grow rapidly in the near future [2]. However, as the number of edge devices continues to grow, so does the risk of cyber threats, making security a critical concern for developers and manufacturers. Moreover, the Dutch General Intelligence and Security Service (AIVD), has raised the alarm for cyber threats [3] in their yearly report about 2022.

Hardware-based security mechanisms like Trusted Execution Environments (TEEs), such as Arm's TrustZone technology, have emerged as a promising solution to address security concerns in Edge Computing environments [4]. TEEs provide an isolated execution environment, separate from the operating system, which allows sensitive applications to run securely. However, while the attestation concept, which allows the integrity of the code running in a TEE to be verified, is well- established in e.g. Intel SGX, there has yet to emerge an open source (remote) attestation implementation for the TrustZone, even though it has been around since the early 2000s [4]. This research proposal aims to investigate this issue and propose a new attestation protocol that is specifically designed for distributed environments with Arm TrustZone SoCs.

The proposal aims to examine existing remote and local attestation mechanisms and come up with a new mechanism or a combination of existing technologies that can be used as an open source implementation for attestation in TrustZone. The proposed mechanism will be evaluated for its security and performance. The research will contribute to filling the gap in attestation mechanisms for TEEs in Edge Computing environments, and may form the basis for an open source mechanism that can be used in future secure applications. The proposal will begin by providing background information on the issue in Section 2, followed

by a review of related work in Section 3. It will then formulate research questions in Section 5, outline the methodology in Section 6.

2 Background

This section is a brief introduction to some topics which are relevant to this research. "Trusted Execution Environment" (Section 2.1) will discuss the concept of a TEE and its uses. Then "Attestation" (Section2.2), will introduce the attestation concept and the use cases associated with it. "Arm TrustZone" (Section 2.3), provides a high level overview of the TrustZone technology as well as some information about the chips which are equipped with TrustZone. Finally, "Attestation Features" (Section 2.4) discusses the set of features that distinguishes attestation mechanisms.

2.1 Trusted Execution Environment

GlobalPlatform, a standardization organization, was the first to come up with the term Trusted Execution Environment (TEE) [5] [6]. They also create standards which developers and manufacturers of devices with a TEE can use by providing Application Programming Interfaces (APIs) and device specifications [4]. Having an open API is desirable to allow other manufacturers and developers to easily interface with TEEs or to create new devices which can leverage existing code bases that are built upon the open specification. This is as opposed to proprietary systems like Samsung KNOX [7], for which it is difficult to verify the security claims or to create applications that work inside such a TEE. Currently, the Confidential Computing Consortium (CCC), a community at the Linux Foundation, wants to bring together vendors and developers to create open standards with the aim to accelerate the adoption of TEEs [8].

GlobalPlatform's definition of a TEE is "a secure area of the main processor of a connected device that ensures sensitive data is stored, processed and protected in an isolated and trusted environment" [9]. In another paper by Sabt, Achemlal and Bouabdallah [5] about the definition of a TEE, the authors denote a Trusted Execution Environment as a tamper-resistant processing environment with guarantees about:

- Authenticity of the executed code
- Integrity of the runtime states
- Confidentiality of data and state of the system

Furthermore, they state that the content in a TEE should not be static. Instead, it should allow for secure updating of the contents in the TEE [5]. Pinto and Santos in their TrustZone survey dedicate a large portion to TEEs and their ability to address the issues that result from the use of large (bloated) Trusted Computing Bases (TCBs) due to their complexity. They say that a TEE can host critical applications in an environment that is smaller than a Rich OS as a TCB. As their definition of a TEE they state that "(..) a TEE consists of an isolated environment in which *Trusted Application (TA)s* can execute without the interference of the local (untrusted) OS." [4]. A *TA*, is software which runs inside a TEE. These TAs are isolated from eachother and isolated from the Rich Execution Environment (REE) which only runs regular applications.

As seen above, the definitions of TEEs can vary in the details. For example, from GlobalPlatform's definition, it would seem as only the processor is placed in a secure state when entering the TEE. But, Pinto and Santos define a TEE as an *environment* and, as such, the peripherals and memory are also in a secure state inside the TEE. In this research, we will follow the latter definition, due to our focus on Arm devices which support secure states for peripherals and memory, as we will see in Section 2.3.

Examples of popular TEEs are: OP-TEE (TrustZone based) [10], Intel SGX [11], and Multizone for RISC-V [12].

2.2 Attestation

Trusted Execution Environments make sure that the code and data inside the TEE cannot be reached from the regular OS. However, this does not provide safeguards against all attacks; attackers with physical access to the device can tamper with memory on the device

and change data or firmware on the device. Furthermore, remote attackers can abuse flaws in device software, such as stack-buffer overflows [13] and command injection [14], to overwrite the code on the device to run malicious applications. Since the contents of a TEE cannot be observed from the outside, changes in the TEE cannot be detected.

Attestation mechanisms aim to prevent such attacks from succeeding. With an attestation architecture and attestation mechanisms, one party can prove to another party that it is running trusted (attested) software and that it has not been subject to any other form of tampering. Attestation mechanisms can aid in putting more trust in the Trusted Execution Environments and likewise, TEEs can strengthen attestation mechanisms [15].

2.2.1 Terminology

Existing works on attestation make use of different terminology for the same actors and parts in the attestation architecture. This work will adopt the terminology as described in RFC 9334 by the IETF [16], which Ménétrey et al. also explain in their work [15]. Therefore we have the following definitions for attestation:

Improve description of TA

Attester The actor that proves some properties about itself to another party.

Claim A *Claim* is a piece of asserted information, such as a cryptographic hash of the code on the device.

Evidence Information which the Attester uses to prove their identity to the Relying Party. It consists of a set of Claims which the Attester has gathered on the system, which are then signed such that the Relying Party can check its validity .

Relying Party The actor that decides whether the *Attester* can be trusted. In RFC 9334, there is also the Verifier role, but in this work we assume the Verifier and Relying Party to be the same entity.

Trusted Application An Attester implementation in the form of an application.

2.2.2 Core Concepts for Attestation

As described by Coker et al. [17], attestation mechanisms should consist of building blocks on top of which the attestation mechanism can be built. Coker et al. describe these as constraints, but we will refer to them as the the core concepts blocks for attestation mechanisms.

Types of Evidence The number of evidence types which the *Attester* can use to prove their identity to the *Relying Party (RP)* has no limit. But it is important that the mechanism utilizes useful information during the attestation process. Most existing attestation mechanisms use the compiled application software as one of the *Claims* in some form, usually a hash.

Another example of a trivial type of *Evidence* for a running TEE, would be to create a hash over the current memory contents. However, such information would not be useful to an attestation protocol, because this memory may change rapidly and unpredictably depending on the application. For these situations, Coker et al. suggest to use only certain parts of the memory which do not change often [17].

In addition to that, freshness of the *Evidence* is also a goal according to Coker et al. That means that in addition to performing the measurement before executing a program the *Attester* must deliver *Evidence* as often as possible. With some attestation mechanisms this means that the *Relying Parties* can trigger a new measurement to get fresh measurement information. Ménétrey et al. also use freshness as one of the fields in their comparison between Attestation Mechanisms for TEEs [15].

Separate Domains Attestation measurement tooling must be able to provide accurate results about the state of the target, even when the target has been compromised. The measurement tool must have access to the target to be able to determine whether it's state is still uncompromised. But the other way around should be impossible. This is to make sure that a compromised target cannot influence the results of the measurement tool.

Coker et al. state that using VMs can be a good way to achieve this separation [17].

They describe a measurement tool which resides in a VM and the target which resides in another VM. The Virtual Machine Monitor should then configure cross-VM visibility such that the measurement VM can inspect the target, and protect the measurement VM from the target. Creating separate TEE enclaves would be an alternative method for separating domains. The measurement tool in this case can be rooted in secure hardware, such as the CPU in the case of Intel SGX [11].

Trust Base Having domain separation and reasonable types of *Evidence* will not be useful without a base of trust in the system. Without this trust base, the fundamental parts of the attestation architecture cannot be verified. Physical compromise of the hardware may enable attackers to overwrite firmware on the device in order to trick the parts of the attestation architecture into attesting parts of the system which are in fact compromised.

Such a trust base should start in the boot process and it should be hardware enforced. Secure Boot or Authenticated Boot are good options to use as a trust base, and many other works also utilize these technologies [5] [18] [19]. Secure Boot or Authenticated Boot can be offered by hardware vendors for their devices with an Arm chip [18] [20]. This secure boot process makes sure that only verified software/firmware can run on the SoC. Usually, the device verifies the firmware that it wants to boot with a public-key that is specific for a software vendor. The software vendor generates a signature of the firmware binary with their secret key and pushes both the firmware and the signature to the device. The device holds the public key, which should be stored in such a way that it can be read but not replaced by a public key from the attacker. UEFI systems also have a similar secure boot process, which can be enabled [21].

2.2.3 Local Attestation

Local Attestation enables an *Attester* TEE to authenticate its identity to a *RP* TEE residing on the same device. The process can then, for example, be based on a key that is bound to the hardware on which the software runs [22].

Intel SGX is a good example of a local attestation mechanism since local attestation serves as one of the key pillars for SGX's security enclaves [11] , and as such, has a well-researched implementation. In SGX, local attestation relies on the fact that both TEEs share the same CPU. This CPU is provisioned with a secret that serves as a message authentication code (MAC) for the attestation mechanism. The *Attester* TEE generates a MAC, which is then checked by the *RP* TEE to validate the identity of the *Attester*. In SGX, Local Attestation relies on the fact that two enclaves on the same device share the same root key from the CPU, from which a report key can be derived.

2.2.4 Remote Attestation

Remote attestation is a natural extension of the concept of local attestation, since the latter is typically used to ensure that a device's hardware and software components have not been tampered with. However, local attestation is limited in its scope, as it only provides assurance about the trustworthiness of the local platform.

Remote attestation, on the other hand, enables a remote entity to verify the trustworthiness of a device. Remote attestation can provide a higher level of assurance about the trustworthiness of a platform compared to local attestation, as it enables verification from a trusted third party. It can, for example, block execution of code if the remote device is not in a known state, possibly preventing malicious software from running. As an example of this concept, the CoCo project uses remote attestation to get decryption keys for software binaries, preventing them from running without attesting first [23].

Remote- and local attestation are not mutually exclusive. On the contrary: remote- and local attestation can complement each other to create a "chain of trust" that utilizes both mechanisms to achieve the desired level of security in a product. Some remote attestation mechanisms, such as Intel SGX [11], require local attestation as part of the remote attestation procedure.

2.2.5 Governing Incentives

The realm of Attestation and Trusted Execution Environments (TEEs) has been subject to a number of incentives that aim to standardize the various aspects that are relevant to this area. Two notable examples of such incentives are the aforementioned Confidential Computing Consortium (CCC) [8] and the Internet Engineering Task Force (IETF) with their Remote Attestation Procedures (RATS) RFC 9334 [16]. These initiatives aim to provide a common framework and guidelines that facilitate the development, deployment, and interoperability of TEE-based systems.

Despite the efforts of these entities, however, there remain many open questions that must be addressed before suitable standards can be established without the risk of unintended consequences. For example, there is the need to ensure that the standards are flexible enough to accommodate a variety of hardware platforms and use cases, while still providing robust security guarantees. Another area of concern is the question of interoperability between different attestation solutions, particularly as the industry continues to evolve and new technologies emerge. And as such, these standards have not yet been established. This still leaves room for industry and researchers as well as open source initiatives to experiment with attestation in to find out what works well.

2.3 Arm TrustZone

The Arm TrustZone is a System-on-Chip (SoC) security solution, which is available on most devices with an Arm chip nowadays. It has already been available since 2004, but only in recent years, with the industry becoming more security focused has it seen more widespread adoption [4].

The TrustZone technology centers around the idea that there are two domains in the computer that must be isolated from each other: the *secure world* and the *normal world*. TrustZone not only controls the CPU state, but the concept of being *secure* or *normal* also extends to other parts of the SoC, such as peripherals and buses (the communication lanes between the parts of the SoC). Arm TrustZone provides strong hardware isolation between the Secure and Normal worlds. The processor's current state is determined by the Non-Secure (`NS`) bit that Arm has added to the processors. The *secure world* consists of everything that runs when the processor is in the secure (`S`) state and the *normal world* of everything when the state is `NS`. The hardware is created in such a way that the *normal world* is incapable of accessing the regions of the SoC that are considered *secure* [18] and the processor can only work in one of the states at a time. Switching between *secure* and *normal* depends on the type of chip that is used. This can be either a Cortex-A or Cortex-M chip [24] [25]. Since this research will mainly focus on the Cortex-A, the Cortex-M TrustZone will not be explained in-depth.

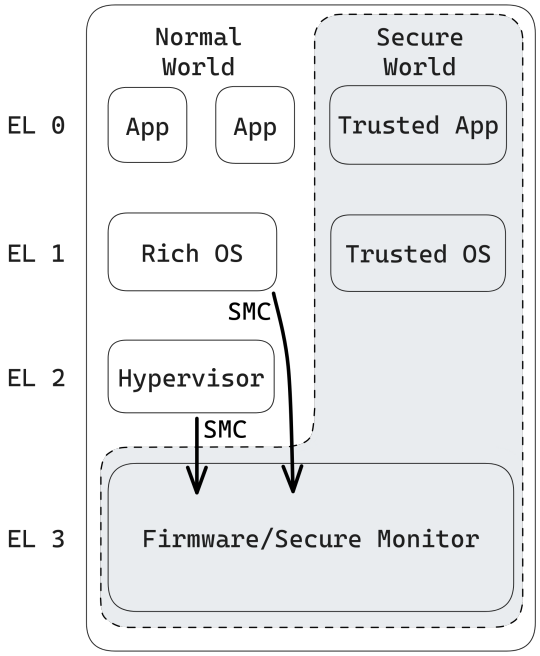

Figure 1: TrustZone in Cortex-A SoCs, adapted from [18]

It is important to note that the Arm TrustZone itself is not a Trusted Execution Environment. However, the TrustZone enables the creation of a TEE by utilizing the isolation, authenticity, and integrity features that it provides.Lastly, it is important to note for this research that the definition of the Arm TrustZone lacks any form of attestation.

2.3.1 TrustZone for Cortex-A

Cortex-A processors are made for devices that feature a full Operating System [24]. In this SoC, the *secure monitor* software provides the context switching functionality between worlds in the TrustZone. Code from both the Secure and the Non-Secure worlds can switch between the processor state by using the `smc` exception. Thus changing the state of the `NS` bit. This is shown in a schematic way in Figure 1.

Within these systems, the secure world extends beyond the processor: also the memory can be in either a *secure* or *non-secure* state. The TrustZone Address Space Controller (TZASC) handles this for DRAM and the TrustZone Memory Adapter (TZMA) does this for the SRAM and ROM [18].

2.4 Attestation Features

This section discusses the distinguishing features between attestation mechanisms. Creators of these mechanisms must make decisions about which features will be supported by the mechanism, and some features may or may not make it into the final version of a mechanism. Therefore, implementations of attestation mechanisms will each have their own set of supported features. A comparison between attestation mechanisms can be made by looking at the differences in feature sets.

2.4.1 Functional Features

The term "Functional Features" refers to the specific functionality of an attestation mechanism that makes it appealing for software manufacturers to incorporate into their systems. These features are closely tied to the overall design of a particular attestation architecture and as such the implementation of Functional Features can vary significantly across different attestation architectures, and the effectiveness of these features is highly dependent on the underlying design and implementation of the attestation mechanism.

Local Attestation (LA) Mechanisms that carry out local attestation have the capability of verifying software locally without the need to connect to a remote entity. In terms of trusted execution environments, local attestation enables a TEE (the *Attester*) to prove its identity to another TEE (the *RP*) that lives on the same hardware. The *RP* can then verify the proof based on information that is available locally on the machine.

Remote Attestation (RA) These mechanisms allow an *Attester* to prove its identity to a remote *RP*. Here, the *Attester* sends *Evidence* over the network to the *RP*, which lives on another device. Usually, a *RP* will carry out remote attestation with the end goal to safely send confidential data to the *Attester* or to verify the integrity of the attesting device.

Mutual Attestation (MA) Mechanisms which feature mutual attestation have the capability of doing attestation on both sides of the (remote) connection. Some *Trusted Application (TA)s* need this as a stronger assurance of trust.

Secure Channels (SC) Some attestation mechanisms can create secure channels during the attestation process to provide confidentiality of the data sent during the attestation process. Sometimes, these secure channels can live longer than solely during attestation and allow for secure messages between the *Attester* and the Relying Party after the former has proven its identity.

Session Resumption (SR) Attestation mechanism that provide session resumption functionality do not need to perform the full attestation mechanism each time. Such protocols would typically feature some sort of "session cookie" which the parties can use to pick up their connection without doing all the steps that would normally be necessary in the attestation process. Most often, these "cookies" expire after a set amount of time after which a full attestation must happen again.

TEE Agnostic (TA) Between different hardware implementations on which Trusted Execution Environments run, the TCB could differ drastically. The TEE agnostic feature shows the extent to which the attestation mechanism relies on specific hardware fea-

tures that prevent it from running on a different hardware architecture. Usually, hardware-specific implementations allow for a greater amount of trust in the attestation mechanism, but at the cost of software portability between devices.

Open Source (OS) An attestation mechanism that fulfills this feature has an open-source implementation. Although this is not a fundamental difference between mechanisms, an open-source attestation mechanism may see greater adoption.

Attester Anonymity (AA) Attestation can strike a balance between privacy and the level of trust required between the *Attester* and the *RP*. If an attestation mechanism uses fewer uniquely identifying features in the *Evidence*, it will be harder to trace the attesting device (and possibly the device's owner). However, this may also result in less convincing *Evidence* for the *RP*.

2.4.2 Security Features

In the context of attestation mechanisms, security features refer to the features that impact the security of the scheme being evaluated or compared. These features are essential for ensuring the integrity and confidentiality of the attestation process, and help to ensure that attestation mechanisms are resistant to a range of potential attacks, including those that attempt to intercept, tamper with, or compromise the attestation process.

Message Verification (MV) From a security perspective, message verification can help ensure the authenticity and integrity of messages exchanged during the attestation process, which prevents man-in-the-middle attacks.

(Mutual) Key Establishment (KE) This feature determines how the protocol establishes new keys during the attestation mechanism and what kind of key agreement protocol it uses for this.

Forward Secrecy (FS) This refers to the property of an attestation scheme that states that sessions in the past are not compromised when an attacker gets access to the session key in another communication session. Attestation mechanisms typically achieve this by using ephemeral session keys. When taken into consideration that devices which use attestation mechanisms could have lifespans of 10 years or more, forward secrecy becomes an important security feature to take into consideration.

Non-Repudiation (NR) Non-Repudiation is the ability of an attestation mechanism to prevent the sender of a message from denying that they ever sent the message. This also means that these messages should arrive unaltered at their destination, or that receivers can detect modifications to the original messages. Attestation mechanisms can achieve non-repudiation by using digital signatures or other cryptographic techniques.

Runtime Verification (RV) Runtime Verification enables an attestation mechanism to monitor and verify the integrity and security of software and firmware components during runtime. This feature is necessary for detecting any tampering that may occur after the initial attestation process has completed. By continuously verifying the runtime environment, attestation mechanisms can ensure that any deviations from the expected behavior are detected. Additionally, runtime verification can also play a role in enabling the attestation mechanism to detect and respond to potential zero-day vulnerabilities. The downside is that runtime verification can be unique per piece of software, and a general solution might not work in each situation.

Hardware Verification (HV) This feature allows an attestation mechanism to detect tampering or unauthorized modifications to the hardware components, as opposed to only detecting software modifications.

Root of Trust (RT) This is not so much a feature as it is a question as to where the root of trust lies in the attestation mechanism. Some attestation mechanisms require a root of trust on the hardware level, whereas others might

put their trusted components in kernel modules on the device.

Freshness (FR) Freshness

3 Related Work

In the following section we will discuss works that have already explored attestation in the context of the Arm TrustZone. This section should provide the reader a grasp on the existing research and their gaps.

3.1 Shepherd et al.

In "Establishing Mutually Trusted Channels for Remote Sensing Devices with Trusted Execution Environments" [26], Shepherd et al. address the challenge of trusting data from unattended sensing devices. In their work they present a new trusted channel protocol for performing attestation and setting up a secure channel between two remote TEEs on devices for use in trusted sensing devices (e.g. IoT health devices). Their end result is a bi-directional attestation mechanism, with the capability to also do only uni-directional attestation for situations where bi-directional attestation is not possible.

3.1.1 System Design

From an operational perspective the attestation architecture runs inside a Trusted Execution Environment (as opposed to the TPM based attestation mechanisms which they state are not sufficient for their use case). They use the 'quote' abstraction in their architecture to send the current state of the TEE. In this work, we refer to a 'quote' as *Evidence*. The mechanism is TEE-agnostic so it does not require TEE features specific to any manufacturer. What they do suggest, is using a Trusted Measurer (TM) to sign the *Evidence* with a key that resides on the device. The Trusted Measurer should be verified in the authenticated boot process. Lastly, they mention that the use of a TPM could increase hardware level tamper resistance, but that would go at the cost of additional required hardware.

We will now discuss the bi-directional protocol. We also have a more formal notation in Protocol 1. Here, there are 2 *Trusted Applications* on both sides of a connection; we call these A and B. One of these initiates the protocol by sending the ID of both TAs (ID_A and ID_B respectively), a nonce n_A, a Diffie-Hellman exponentiation G_A, an attestation request for the other application AR_B and a session cookie S_{cookie}, which is a hash of all the attributes that A sent to B except the attestation request. The parties can use this cookie to pick up an already established attestation session, without performing the full attestation again. This increases efficiency, because it means that the devices do not have to calculate new DH exponentiations. It does, however, mean that the loss of a session key can have a larger impact compared to systems which do not feature session resumption.

After A has sent the first message, B will respond with a message that consists of both IDs, a nonce which they generated (n_B) another Diffie-Hellman exponentiation G_B. Then B also sends a part which is encrypted with symmetric key K_E and has a MAC with key K_{MAC}, denoted by: $[\sigma_{TA_B}(X_{TA_B})||\sigma_{TA_B}(V_{TA_B})]^{K_E}_{K_{MAC}}$. Where X_{TA_B} is a hash of the IDs, nonces and DH exponentiations and V_{TA_B} is *Evidence* from B's TEE, concatenated with both nonces. $\sigma_x(m)$ denotes that message m is signed with x's public/private key pair.

The final step involves A responding with another encrypted message $[\sigma_{TA_A}(X_{TA_A})||\sigma_{TA_A}(V_{TA_A})]^{K_E}_{K_{MAC}}$ and S_{cookie}, where X_{TA_A} should be the same as X_{TA_B} and V_{TA_A} is now *Evidence* generated by the TEE of A along with both nonces. Note that the protocol supports both bi-directional as uni-directional attestation, and in the latter case, the *Evidence* by A is not required.

3.1.2 Limitations

In their article, Shepherd et al. provide a clear explanation of their protocol, which has the ability to function in both bi-directional and uni- directional modes, a promising feature. However, the authors acknowledge that their protocol yields a 4x overhead compared to conventional TLS with Diffie- Hellman and RSA. This could have an impact on tightly constrained devices. Furthermore, due to its

Protocol 1: Bi-Directional Trust Protocol (BTP) by Shepherd et al. [26]

(1) $TA_A \rightarrow TA_B : ID_A \mid\mid ID_B \mid\mid n_A \mid\mid G_A \mid\mid AR_B \mid\mid S_{cookie}$
$S_{cookie} = H(G_A \mid\mid n_A \mid\mid ID_A \mid\mid ID_B)$
(2) $TA_B \rightarrow TA_A : ID_B \mid\mid ID_A \mid\mid n_B \mid\mid G_B \mid\mid [\sigma_{TA_B}(X_{TA_B}) \mid\mid \sigma_{TA_B}(V_{TA_B})]^{K_E}_{K_{MAC}} \mid\mid AR_A$
$X_{TA_B} = H(ID_A \mid\mid ID_B \mid\mid G_A \mid\mid G_B \mid\mid n_A \mid\mid n_B)$
$V_{TA_B} = Q_{TA_B} \mid\mid n_B \mid\mid n_A$
(3) $TA_A \rightarrow TA_B : [\sigma_{TA_A}(X_{TA_A}) \mid\mid \sigma_{TA_A}(V_{TA_A})]^{K_E}_{K_{MAC}} \mid\mid S_{cookie}$
$X_{TA_A} = H(ID_A \mid\mid ID_B \mid\mid G_A \mid\mid G_B \mid\mid n_A \mid\mid n_B)$
$V_{TA_A} = Q_{TA_A} \mid\mid n_B \mid\mid n_A$

multi-architecture compatibility, the protocol is unable to leverage advanced hardware features that offer stronger security assurances for uncompromised devices. Additionally, the protocol relies on the Diffie- Hellman key exchange, which may not be sufficient for devices requiring long- term functionality, particularly as quantum computing advances and potentially breaks the key-exchange algorithm (for example by using Shor's algorithm [27]). Lastly, the protocol requires that both devices always have an active network connection to satisfy the protocol. This makes sense for the use case of trusted sensing devices, but this makes the protocol not suitable for certain other deployment scenarios.

3.2 Ménétrey et al.

Ménétrey et al. implemented a secure WebAssembly runtime environment for TrustZone in their WaTZ paper [28]. WebAssembly (Wasm) [29] provides a runtime environment which was originally designed for web browsers, but can also run in a standalone runtime environment nowadays. As part of WaTZ, the authors also needed to develop an attestation mechanism, since TrustZone lacks a native attestation mechanism. We will now focus on that attestation mechanism here.

3.2.1 System Design

In WaTZ, a kernel module generates *Evidence* which includes:

- An anchor value that binds the parameters to a session
- The WaTZ version number
- The hash of the Wasm bytecode
- The public key of the attestation service
- The signature of the evidence

We rewrote their (SGX inspired) protocol in the same style as the previous section, which is shown in Protocol 2.

The protocol starts with the *TA* sending the public part of a session key-pair that the *TA* generated (G_{TA}) to the *RP*. Then, the *RP* responds by generating their own session key pair (with public part G_{RP}) Both the session key pairs are combined to create a *Key Derivation Key* which is subsequently derived in 2 shared secret keys: K_m and K_e for generating MACs and encrypting messages with a symmetric key, respectively. *RP* sends a message to *TA* with the G_{RP}, its public key Pk_{RP} and a signature of the session keys. After receiving and verifying the message, *TA* gathers evidence locally and responds with a message that contains G_{TA}, the evidence (with a so-called *anchor* which is the hash of the public session keys) along with the *TA*'s public key, a signature of the evidence and a MAC of the entire message. *RP* receives and verifies this message again. Then it checks the evidence that it received and it makes the decision whether the device is in a known state. If that is the case, *RP* can send a new message with arbitrary confidential data that is encrypted with AES-GCM.

3.2.2 Limitations

The authors have verified the remote attestation mechanism with Scyther to ensure its formal correctness and have provided a useful example of how a remote attestation mechanism could work in TrustZone. However, there are several limitations to this mechanism that should be considered.

Protocol 2: Remote attestation protocol by Ménétrey et al. in WaTZ [28]

(1) $TA \rightarrow RP : G_{TA}$
(2) $RP \rightarrow TA : X_{RP} \,||\, MAC_{K_m}(X_{RP})$
$X_{RP} = G_{RP} \,||\, Pk_{RP} \,||\, \sigma_{RP}(G_{RP} \,||\, G_{TA})$
(3) $TA \rightarrow RP : Y_{TA} \,||\, MAC_{K_m}(Y_{TA})$
$Y_{TA} = G_{TA} \,||\, evidence \,||\, \sigma_{TA}(evidence)$
$evidence = (H(G_{TA}||G_{RP})||Pk_{TA}||...)$
(4) $RP \rightarrow TA : iv \,||\, AES - GCM_{K_e}(data)$

First, the protocol is specific to WaTZ and as such can only support Wasm applications. This may not be easily adaptable for other applications, since these may have specific requirements and build environments. Second, the protocol sends sends the messages in plaintext over the network. More privacy oriented applications might require higher levels of confidentiality. The authors also mention this in their paper and state that these messages could also be encrypted with AES-GCM. Finally, the Wasm application does not perform local attestation and relies on Secure Boot to ensure the device is uncompromised. If the software does not have network access, it cannot determine whether it is running on an uncompromised device.

4 Use Case & Motivation

Scalys [30], the company behind TrustBox [31], has developed a networking device for use in zero-trust environments. The TrustBox is powered by an NXP chip that provides access the Arm TrustZone [32], and it can create TEEs by with e.g. OP-TEE. However, TrustZone does not provide a full attestation mechanism, as outlined in Section 2.3. Because the device can operate in a variety of crucial scenarios (e.g. in an offshore wind farm which are hard to reach by engineers, but where consequences of attacks are large), it is essential to maintain trust in the software running on the device, and attestation provides an added layer of security.

Over time, the TrustBox could be made accessible to other software vendors for loading their software. In this case, it would be necessary for software vendors to verify that their software still operates on an authentic TrustBox. Attestation mechanisms would be invaluable in this scenario, as Scalys would need to ensure the device's safety while pushing new firmware updates, and software vendors should receive guarantees that their software still runs in a secure environment.

5 Research Questions

The Related Work and Background demonstrate that there is a lack of a well-documented, standardized attestation mechanism for applications running on TrustZone-enabled devices that fits the context described in Section 4, which enables both local and remote attestation with an open-source implementation.

To address this challenge, this research aims to design a new attestation protocol by comparing existing attestation mechanisms for the TrustZone. The proposed protocol will be designed to fit the context described in Section 4 and will take into account the specific requirements of TrustZone-enabled devices. We have formulated a set of research questions that will guide our investigation and design process.

5.1 Comparing Mechanism Designs

RQ1 How do existing works in research and open source projects compare to each other in the context of the feature sets that the specific mechanisms support?

5.2 Protocol Design & Implementation

RQ2 What would a design for a TrustZone-based attestation mechanism that supports both local and remote attestation look like?

RQ2.1 How can we make use of existing open source projects to imple-

ment the aforementioned attestation mechanism protocol?

5.3 Evaluating the protocol

RQ3 What is the performance of the proposed protocol in terms of speed, memory footprint and security?

RQ3.1 How can we formally verify the correctness of the protocol?

RQ3.2 How does the performance of a device running our protocol compare to the scenario where the device does not use it?

6 Methodology

6.1 Comparing Designs

We want to get a good grasp of the attestation protocols out there and see what features form their strong and weak points. The answers to these questions should help in developing our own protocol as an answer to RQ2. In our research we will compare the following CCC attestation mechanisms and systems that already have an open source implementation:

- Open Enclave SDK [33]
- Veraison [34]
- Apache Teaclave [35]

The works to be considered in this research include those described in Section 3 [28] [26], Sanctum by Costan et al. [36], SecTEE by Zhao et al. [37], the TrustZone-based attestation work by Quaresma et al. [38], "Practical Runtime Attestation for Tiny IoT Devices" by Hristosov et al. [39], SWATT by Seshradi et al. [40] VRASED by Nunes et al. [41], C-FLAT by Abera et al. [42], "An Open-Source Framework for Developing Heterogeneous Distributed Enclave Applications" by Scopelliti et al. [43], Komodo by Ferraiuolo et al. [44], "Remote Attestation for Embedded Systems" by Kylänpää and Rantala [45], ERAMO by Ostergaard et al. [46], AdAttester by Li et al. [47] and SofTEE by Lee and Park [48].

In addition to that, we will try to include some information for RQ2.1 already by taking open source libraries in the comparison into account. These may not necessarily have been created for attestation, but they could be used as a building block for a new attestation mechanism. Our main requirement for these is that their implementation works on Arm devices with a Cortex-A chip. An example of such a library would be the Mbed-TLS Project [49].

The works will be compared based on the feature sets we described in Sections 2.4.1 and 2.4.2.

6.2 Protocol Design & Implementation

Answering RQ2 requires us to come up with a protocol which can be implemented as an open source library for use on TrustZone devices (most notably Cortex-A devices). In order to answer this question in a well founded manner, the answer largely depends on the results from RQ1. We will use the result from that comparison in the design of our own attestation mechanism.

The aim of RQ2.1 is to explore the feasibility of leveraging existing open source libraries that were not specifically designed for attestation, but could potentially serve as foundational components in our mechanism. Essentially, we seek to investigate whether we can use existing software building blocks to reduce implementation time and enhance trustworthiness in our attestation solution.

6.3 Examining the Protocol

First to answer RQ3.1, we will formally verify our protocol by implementing it in Scyther [50]. Other works, such as the ones by Shepherd et al. [26] and Ménétrey et al. [28] also took this step, which can aid in our endeavour to create the protocol in Scyther.

To address RQ3 and RQ3.2, a Proof-of-Concept (PoC) implementation will be developed. Depending on the outcomes of RQ2 this PoC will likely be integrated as a *TA* in OP-TEE or an extension of the existing OP-TEE system. Given the popularity of OP-TEE as a TEE framework for TrustZone and its relevance to the Use-Case (Section 4), it is the most logical choice for this purpose. Furthermore, the PoC will most likely be created in C,

since that remains the dominant language in the OP-TEE ecosystem and embedded sytems in general. However, Rust [51] could be an alternative choice due to the potential security benefits it offers, and it is also supported by OP-TEE for *Trusted Applications* [52]. Again, this will depend on the actual protocol design from RQ2.

After finalizing the protocol design and its implementation we can measure the protocol through measurements of our PoC implementation. Since it is unclear at the time of writing this Methodology what the implementation will look like, it becomes challenging to determine the exact parameters that will be evaluated in this phase. Nonetheless, the measurements will likely focus on the following aspects:

- The time it takes to start applications and boot the machine. We will also compare this to the case where attestation is omitted.
- The total time it takes to perform the attestation.
  - Depending on the results of RQ2 we might also measure the time it takes for a *RP* to verify an attestation.
- The maximum amount of memory used in running the application.

7 Results

7.1 Comparison

This section describes the results for the comparison phase as result to RQ1

7.1.1 Functional Features

Table 1 shows the results of our comparison as answer to RQ1 with regards to the Functional Features from Section 2.4.1.

First, we can observe that most of the mechanisms discussed in the research support Remote Attestation (except for SofTEE). On the other hand, only half of the examined mechanisms from the papers support Local Attestation. However, all mechanisms that do not support local attestation do support remote attestation. Among the mechanisms supporting remote attestation, only three provide mutual attestation, while six out of fourteen support Secure Channels with the remote server. There is only one mechanism that supports session resumption. Additionally, there are only five TEE-agnostic mechanisms; the others are TrustZone-only. Furthermore, six of the papers also offer an Open-Source implementation for their mechanism or enclave implementation. Lastly, measuring attester anonymity is challenging, resulting in three unknowns in this regard. Five of the papers lack this anonymity, potentially allowing user data to be sent during attestation. In contrast, the remaining papers can usually uniquely identify a device (this seems to be inherent to many attestation mechanisms) without obtaining user data.

7.1.2 Security Features

Clean up the KE table

The security features are challenging to combine into one table since the protocols differ significantly in their core. Consequently, the *KE* and *RT* columns contain text instead of symbols. Moreover, two papers (ERAMO [46] and SANCTUARY [19]) leave the choice of the key algorithm to the system implementer. Hence, certain fields in the table are marked with "✍" to indicate that the answer for that particular field depends on the specific implementation.

There are also some abbreviations in the table that we will list here as reference:

- *KDF*: Key Derivation Function
- *DH*: Diffie-Hellman
- *SecMon*: Security Monitor (part of the SofTEE system [48])
- *ECDHE*: Elliptic Curve Diffie-Hellman Ephemeral
- *TZ*: TrustZone
- *HUK*: Hardware Unique Key
- *SB*: Secure Boot

reference or explanation

Now, moving on to the comparison. Here, we observe that all the mechanisms perform message verification, but only one of them lacks non-repudiation. In SWATT, the challenge key is transmitted in plain text over the network, and there is no additional key, allowing others

Table 1: Functional Features Table

Paper Name	*LA*	*RA*	*MA*	*SC*	*SR*	*TA*	*OS*	*AA*	*Ref*
AdAttester	✗	✓	✗	✗	✗	✗	✗	✗✓	[47]
C-FLAT	✓	✓	✗	✗	✗	?	✓	✗	[42]
End-to-End Security for Distributed...	✓	✓	✓	✓	✗	✓	✓	?	[53]
ERAMO	✗	✓	✗	✗	✗	✗	✗	✗	[46]
Establishing Mutually Trusted Channels	✗	✓	✓	✓	✓	✗	✗	?	[26]
Komodo	✓	✓	✗	(✓)	✗	✗	✓	?	[44]
Practical Runtime Attestation for ...	✗	✓	✗	✗	✗	✗	✗	✗	[39]
Remote Attestation for Embedded ...	✗	✓	✗	✗	✗	✗	✗	✓	[45]
SANCTUARY	✓	✓	✗	✗	✗	✗	✗	✗	[19]
SecTEE	✗	✓	✗	✓	✗	✗	?	✓	[37]
SEDA									[54]
SofTEE	✓	✗	✗	✗	✗	✓	✗	✓	[48]
SWATT	✗	✓	✗	✗	✗	✓	✗	✗	[40]
TrustZone based Attestation in ...	✓	✓	✓	✓	✗	✗	✓	✓	[38]
VRASED	✗	✓	✗	✗	✗	✓	✓	✗	[41]
WaTZ	✗	✓	✗	✓	✗	✗	✓	✓	[28]

✓ – Supported. (✓) – Supported by design, but no implementation. ✗ – Unsupported. ? – Unclear whether supported.

Table 2: Security Features Table

Paper Name	*MV*	*KE*	*FS*	*NR*	*RV*	*HV*	*RT*	*FR*	*Ref*
AdAttester	✓	Device Keys	✗	✓	✓	✓	Secure Boot	✓	[47]
C-FLAT	✓	?	✗	✓	✓	✗	TrustZone	✓	[42]
End-to-End Sec...	✓	KDF which?	✓	✓	✗	✓*	HUK	✓	[53]
ERAMO	✍	✍	✍	✍	✓	✓*	TrustZone	✓	[46]
Establishing M...	✓	DH	✓	✓	✗	✍	TM in TrustZone	✓	[26]
Komodo	✓	MAC with secret	✓	✓	✗	✓*	Hardware chain	✓	[44]
Practical Runt...	✓	KDF for asym	✗	✓	✗	✗	TrustZone & HUK	✓	[39]
Remote Attesta...	✓	Key from TPM	✗	✓	✗	✗	TZ, kernel & SB	✓	[45]
SANCTUARY	✍	✍	✍	✍	✗	✓	TrustZone & HUK	✓	[19]
SecTEE	✓	DH	✓	✓	✗	✗	TrustZone & HUK	✓	[37]
SEDA									[54]
SofTEE	✓	Key in SecMon	(✗)	✓	✗	✗	TPM	✓	[48]
SWATT	✓	Random generated MAC	✓	✗	✓	✗	Random MAC	✓	[40]
TrustZone base...	✓	Same root key	✓	✓	✓	✗	TrustZone & HUK	✓	[38]
VRASED	✓	✗	✓	✓	✓	✓	HUK and form. ver.	✓	[41]
WaTZ	✓	ECDHE	✓	✓	✗	✓*	HUK & SB	✓	[28]

✓ – Supported. ✓* Supported with a caveat. (✓) – Supported by design, but no implementation. ✗ – Unsupported. (✗) – Unsupported due to nature. ? – Unclear whether supported. ✍ – Depends on chosen algorithms/implementation

to intercept the key and generate their own responses. Additionally, four mechanisms lack forward secrecy in the messages transmitted over the network (SofTEE does not support remote attestation, hence it is marked with an (✗)). Then, only 6 mechanisms support runtime verification. Finally, Only six mechanisms support runtime verification. Lastly, we notice that only three mechanisms actually support hardware verification, while others rely on secure boot to assume this capability.

7.2 Protocol Design

Sho the Design of the protocol.

7.2.1 Goals

The goals of our newly created attestation mechanism are as follows:

- The mechanism should also support local attestation
- Third parties can perform attestation for their own apps
- Device manufacturers should perform attestation on the device and device firmware
- The device manufacturer and app developer should not be required to share sensitive information about the attestation
- Keys should be upgradeable
- After successful attestation, there should be an option to retain a secure connection between TA and RP

7.2.2 Merkle Trees

A Merkle Tree is a form of Tree-Based Data Structure where each leaf is the hash of its actual value, and each node is the hash of its children, concatenated [55]. They are sometimes also referred to as Hash Trees.

Despite its simplicity, Merkle Trees are useful in many different scenarios; ranging from Blockchain [56], and Digital Signatures [55] to Certificate Transparency Logs [57].

In Figure 2, the leafs of the tree are the hashes of A, B, C and D ($H_{\{A,B,C,D\}}$)). Then, those hashes are concatenated in their parent node and the concatenation is hashed as well

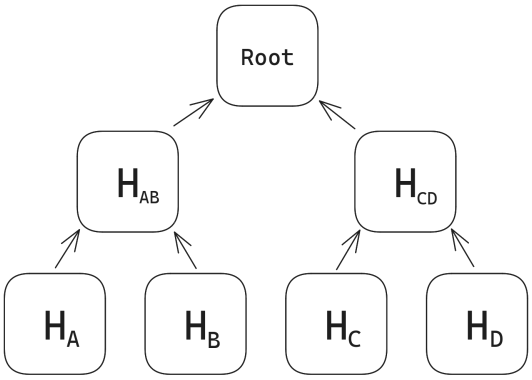

Figure 2: A simple Merkle Tree

(e.g. $H_{AB} = H(H_A|H_B)$). Finally, these parents are also concatenated and hashed in the root of the tree. The idea is that when a single leaf in the tree changes, the root of the tree will also change. Therefore, inconsistensies in the leaf data will result in a different root.

Merkle Proof A Merkle Proof can be utilized to establish the inclusion of a specific value within a Merkle Tree root. In the example proof in Figure 3, the objective is to demonstrate that A is part of the tree, and the prover has already transmitted the Merkle Tree root to the verifier. The prover must subsequently transmit H_B and H_{CD} to the verifier. Now they can independently compute H_A and H_{AB} since H_B has been provided. Subsequently, by employing H_{AB} and the transmitted H_{CD}, the verifier can compute the root of the tree and check if it matches to the previously received root.

Make professional images

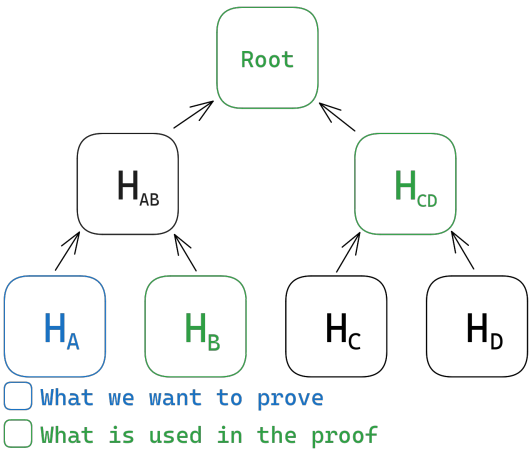

Figure 3: A simplified Merkle Tree Proof

Notice that the prover is not necessarily required to sent the plaintext value of A to the verifier. If the verifier knows the expected value A, they can calculate the hash of A themselves.

7.2.3 Roles & Assumptions

Provide some framework to place our protocol in.

Roles Our proposed protocol contains a couple of roles which we will define here. The roles follow the conventions as described in RATS by the IETF [16].

- Attester: The entity, usually a device, that performs the attestation and must deliver evidence. (See also *Attester*)
- Relying Party: The party which, in the end, makes the end decision on the *Evidence* that is delivered by the *Attester*. (See also *RP*)
- Verifier: One or multiple parties that appraises the *Evidence* which the *Attester* delivered.

Assumptions Place assumptions here.

- The *RP* has a secure connection to the verifiers. Man-in-the-middle attacks and other interference should not be possible here. An option could be to set up an IPSEC connection with pre-shared symmetric keys over a secure channel, but setting up this connection is out of scope of this paper.

7.2.4 Proposed Protocol High Level

Figure 4 shows our implementation of a Merkle Tree to use as *Evidence* in our proposed protocol. In this specific implementation of the Merkle Tree, we have a subtree for each role in the attestation process. Figure 4, e.g. has a subtree with information that the *RP* knows, and 2 Verifier subtrees: V_x and V_y. Together, they form the root of the tree: E. The root of the tree, E, will be sent by the *Attester* application/device to the *RP*. Structuring the data which is sent to the *RP* in this way makes sure

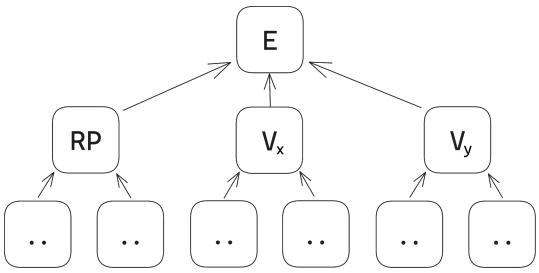

Figure 4: Merkle tree with evidence for the protocol

that Verifiers and the *RP* do not learn private information about each other, since that information is stored in the Merkle Tree and thus only available in a Merkle Proof.

Figure 5 shows a high level overview of the attestation flow in the proposed protocol, after the *RP* has initiated the application attestation. We will now provide a description to the numbers in the image.

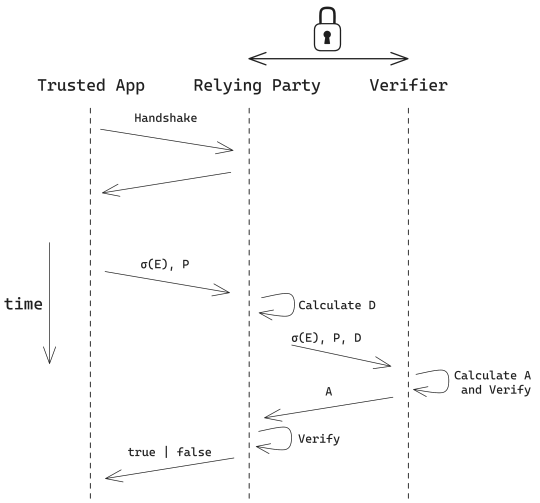

Figure 5: The Attestation flow in the proposed protocol

1. The *TA* gathers *Evidence* from the Trusted OS and the device firmware.
2. The signed *Evidence* is sent to the *RP*.
3. The *RP* checks the application related *Evidence* forwards the attestation to the Device Verifier.
4. Device Verifier checks the device and firmware related *Evidence* and returns the result to the *RP*.

Algorithm 3: The Hash Key Derivation Function

Function `HKDF(`*old_chaining_key, input_data*`)`**:**
  temp_key = `HMAC(`*old_chaining_key, input_data*`)`
  new_chaining_key = `HMAC(`*temp_key,* 0x01`)`
  new_symmetric_key = `HMAC(`*temp_key, output_1* || 0x02`)`
  return *new_chaining_key*, *new_symmetric_key*

5. The *RP* verifies if the results match and and returns the final result to the device.

7.2.5 Proposed Protocol Handshake

Before sending the Merkle Tree from Section 7.2.2 to the *RP* a secure connection should be set up between the *TA* and the *RP*. The comparison study outlined in Section 7.1 reveals that numerous existing protocols employ either a symmetric encryption algorithm or a variant of Diffie-Hellman key exchange. The symmetric key approach poses difficulties with regards to scalibility. Moreover, it is desirable for the protocol to be capable of generating a new key for each attestation-performing application. Finally, a secure channel should be established after the handshake to enable fast and secure message exchange. A promising framework for such a handshake protocol is the Noise Protocol Framework developed by Trevor Perrin [58].

The Noise Protocol Framework is a framework for constructing secure cryptographic protocols that support authentication, forward secrecy, and identity hiding. Additionally, protocols created using the Noise Framework are simple and require only a hash function, a symmetric cipher function (in AEAD mode), and a DH function. The simplicity of this framework makes it a logical choice for adopting it in our attestation mechanism, particularly considering the constraints that are associated with developing embedded systems (where ARM devices are often used). The framework has also been investigated in research [59] and been used to create the message protocols for WhatsApp [60], Wireguard [61] and the Lightning Network [62]. Lastly, due to its relatively simple nature, noise protocols are also relatively easy to prove in a formal manner, which is useful in answering RQ3.1. We will now continue by explaining the handshake mechanism.

In this handshake protocol, we will use the `XK` variant of the Noise Protocol Framework. This variant allows the *TA* to send their static public key to the *RP* during the handshake (`X`), while already having knowledge of the *RP*'s static public key (`K`). The handshake is shown in Protocol 4. The next paragraph aims to explain in more detail what has been described in the protocol notation.

During the setup of the *TA*, it should have received the *RP*'s static public key (0). Manufacturers of devices should be free to decide in what manner this key ends up on the device with the *TA*. As long as the key is protected from tampering.

Then, the handshake starts with the *TA* generating an ephemeral Diffie-Hellman key and sending it to the *RP* in (1). Along with that, the *TA* can also encrypt some additional payload data with a temporary key that has been generated from Function 3 by using the previous chaining key *ck* and the Diffie-Hellman calculation between the private part of the *TA*'s Diffie-Hellman key and the (already known) public part of the *RP*'s static *DH* key. However, the payload in this first message should not be treated as trusted input by the server, because the plaintext is encrypted with an ephemeral key. Therefore, the server cannot yet know if it is interacting with a genuine *TA*. Since the encryption is a form of AEAD, we also encrypt with associated data and a nonce that increments after encryption and resets after every *HKDF* function. Each time encryption is mentioned from now on, we refer to AEAD encryption with h as associated data and this nonce. In this protocol, that is h; a hash chain of data which both the *TA* and *RP* learn during the handshake. In the case of this first message, the previous h value is hashed together with the public part of the Diffie-Hellman key to act as associated data. After the encryption of the payload, h is up-

explain this somewhere in an earlier section

Protocol 4: Our Proposed Protocol Handshake

(0) $RP \rightarrow TA : S_{RP}$

(1) $TA \rightarrow RP : G_{TA} \mid\mid ciphertext$
$h = \texttt{HASH}(h||G_{TA})$
$ck, k1 = \texttt{HKDF}(ck, \texttt{DH}(g_{TA}, S_{RP}))$
$ciphertext = [payload]^{h}_{k1}$
$h = \texttt{HASH}(h \mid\mid ciphertext)$

(2) $RP \rightarrow TA : G_{RP} \mid\mid ciphertext$
$h = \texttt{HASH}(h||G_{RP})$
$ck, k2 = \texttt{HKDF}(ck, \texttt{DH}(g_{RP}, G_{TA}))$
$ciphertext = [payload]^{h}_{k2}$
$h = \texttt{HASH}(h \mid\mid ciphertext)$

(3) $TA \rightarrow RP : encrypted\_key \mid\mid ciphertext$
$encrypted\_key = [S_{TA}]^{h}_{k2}$
$h = \texttt{HASH}(h \mid\mid encrypted\_key)$
$ck, k3 = \texttt{HKDF}(ck, \texttt{DH}(s_{TA}, G_{RP}))$
$ciphertext = [payload]^{h}_{k3}$
$h = \texttt{HASH}(h \mid\mid ciphertext)$

(4) $TA \rightarrow RP : [payload]_{c1}, RP \rightarrow TA : [payload]_{c2}$
$c1, c2 = \texttt{HKDF}(ck, \_)$

dated by hashing its previous value with the ciphertext. Note that in the case that there is no payload, we will still be performing AEAD, just with 0 bytes as the input. Once the *RP* receives the message from the *TA*, they will first calculate the value of h with the *TA*'s public key, then decrypt the payload and verify the AD and lastly calculate the current value of h by hashing with the ciphertext.

For step (2), the *RP* also generates an ephemeral DH key. This key will be the first part of the message which it will send back to the *TA*. The *RP* further chains the h value with the generated ephemeral public key and calculates a new key to encrypt the payload for this message by 'mixing in' their ephemeral key with the *TA*'s ephemeral public key. Then a new value for h is calculated by adding the new ciphertext to the hash chain. The *TA* receives the ephemeral key along with the ciphertext and mirrors these steps to decrypt the payload.

Step 3 is a bit different, since the *TA* will send their static public key in encrypted form to the *RP*. To encrypt the static public key, they use the same key as used in the payload encryption from step 2 and the updated h. Then, the encrypted key is hash-chained to h and a new DH calculation occurs between the *TA*'s static key and the *RP*'s ephemeral key. Again, an optional payload is encrypted with the new key and afterwards h gets a new value.

Step 4 finishes the handshake by creating 2 "CipherStates". $c1$ for traffic from *TA* to *RP* and $c2$ for the other way around. From this moment on, all messages are encrypted symmetrically with these 2 keys and associated data set to zero length and an incrementing nonce per cipher state. So secret data can now be exchanged safely between *TA* and *RP*. This includes the Merkle Tree as well as any other secret data that may be exchanged between these parties.

8 Discussion

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur ac lectus metus. Pellentesque vulputate sem lobortis, fringilla dolor vitae, malesuada purus. Ut feugiat pellentesque porta. Phasellus fermentum scelerisque felis, dignissim vulputate lorem dignissim eget. Nulla non bibendum erat. Nulla efficitur augue ac justo cursus rutrum at non magna. Aliquam erat volutpat. Proin ut lacus nisl. Donec venenatis dignissim ante id molestie. Proin et nibh in nunc faucibus ullamcorper vitae quis augue. Vivamus est elit, ultrices vitae vulpu-

tate quis, volutpat sit amet risus. Sed aliquet lorem turpis, quis dapibus urna pulvinar eget. Morbi viverra lectus consectetur, suscipit mauris id, eleifend magna. Cras nec vehicula nisl, eget gravida massa.

Morbi tempus faucibus diam non semper. Aliquam vitae magna vitae magna dapibus volutpat eu ut justo. Curabitur quis mi sapien. Donec ac arcu erat. Vestibulum non consectetur nulla, a egestas lectus. Donec vel nisl porttitor, faucibus sem quis, fringilla lacus. Duis lacinia id eros et pellentesque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec vel leo fringilla, mollis metus id, vestibulum ipsum. Cras tincidunt fermentum sem eu ultrices. Aliquam erat volutpat. Sed id mi placerat, ultricies arcu vitae, ultrices turpis. Vestibulum quis tortor vel sem iaculis tincidunt. Fusce condimentum sit amet quam et posuere. Aenean et ex arcu. Nullam ultricies diam dui, non bibendum lorem bibendum quis. Maecenas facilisis dignissim convallis. Duis augue dolor, efficitur ut tincidunt ut, hendrerit quis velit. Donec a odio feugiat, consequat dolor in, hendrerit turpis. Donec sagittis libero urna. Praesent eu urna sapien. Phasellus tortor nulla, tincidunt ac nisi in, pellentesque feugiat arcu. Praesent euismod magna id dui lacinia consectetur. Curabitur vel turpis ut enim feugiat porttitor. Vestibulum elementum eros ac lectus porta scelerisque. Pellentesque ultricies, massa non aliquet ultricies, elit tortor pulvinar orci, quis venenatis dui velit non ipsum. Nullam consectetur suscipit mi, sed dictum sapien lobortis eu. Vestibulum posuere nulla ac dignissim ornare. Praesent et vestibulum elit. Nullam non velit nec nibh rutrum ultrices id eget erat. Aliquam vehicula semper elit. Phasellus non elit vel nisi tincidunt placerat vitae at justo. Nullam tristique tincidunt maximus. Sed facilisis nisi vel condimentum ultrices.

9 Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur ac lectus metus. Pellentesque vulputate sem lobortis, fringilla dolor vitae, malesuada purus. Ut feugiat pellentesque porta. Phasellus fermentum scelerisque felis, dignissim vulputate lorem dignissim eget. Nulla non bibendum erat. Nulla efficitur augue ac justo cursus rutrum at non magna. Aliquam erat volutpat. Proin ut lacus nisl. Donec venenatis dignissim ante id molestie. Proin et nibh in nunc faucibus ullamcorper vitae quis augue. Vivamus est elit, ultrices vitae vulputate quis, volutpat sit amet risus. Sed aliquet lorem turpis, quis dapibus urna pulvinar eget. Morbi viverra lectus consectetur, suscipit mauris id, eleifend magna. Cras nec vehicula nisl, eget gravida massa.

Morbi tempus faucibus diam non semper. Aliquam vitae magna vitae magna dapibus volutpat eu ut justo. Curabitur quis mi sapien. Donec ac arcu erat. Vestibulum non consectetur nulla, a egestas lectus. Donec vel nisl porttitor, faucibus sem quis, fringilla lacus. Duis lacinia id eros et pellentesque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec vel leo fringilla, mollis metus id, vestibulum ipsum. Cras tincidunt fermentum sem eu ultrices. Aliquam erat volutpat. Sed id mi placerat, ultricies arcu vitae, ultrices turpis. Vestibulum quis tortor vel sem iaculis tincidunt. Fusce condimentum sit amet quam et posuere. Aenean et ex arcu. Nullam ultricies diam dui, non bibendum lorem bibendum quis. Maecenas facilisis dignissim convallis. Duis augue dolor, efficitur ut tincidunt ut, hendrerit quis velit. Donec a odio feugiat, consequat dolor in, hendrerit turpis. Donec sagittis libero urna. Praesent eu urna sapien. Phasellus tortor nulla, tincidunt ac nisi in, pellentesque feugiat arcu. Praesent euismod magna id dui lacinia consectetur. Curabitur vel turpis ut enim feugiat porttitor. Vestibulum elementum eros ac lectus porta scelerisque. Pellentesque ultricies, massa non aliquet ultricies, elit tortor pulvinar orci, quis venenatis dui velit non ipsum. Nullam consectetur suscipit mi, sed dictum sapien lobortis eu. Vestibulum posuere nulla ac dignissim ornare. Praesent et vestibulum elit. Nullam non velit nec nibh rutrum ultrices id eget erat. Aliquam vehicula semper elit. Phasellus non elit vel nisi tincidunt placerat vitae at justo. Nullam tristique tincidunt maximus. Sed facilisis nisi vel condimentum ultrices.

References

[1] F. Almeida, J. Duarte Santos, and J. Augusto Monteiro, "The Challenges and Opportunities in the Digitalization of Companies in a Post-COVID-19 World," *IEEE Engineering Management Review*, vol. 48, no. 3, pp. 97–103, 2020, ISSN: 1937-4178. DOI: 10.1109/EMR.2020.3013206.

[2] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017, ISSN: 0018-9162. DOI: 10.1109/MC.2017.9. [Online]. Available: http://ieeexplore.ieee.org/document/7807196/ (visited on 02/13/2023).

[3] M. v. B. Z. en Koninkrijksrelaties. "Internationale dreigingen - Jaarverslagen - AIVD." (), [Online]. Available: https://www.aivd.nl/onderwerpen/jaarverslagen/jaarverslag-2022/internationale-dreigingen (visited on 04/19/2023).

[4] S. Pinto and N. Santos, "Demystifying Arm TrustZone: A Comprehensive Survey," *ACM Computing Surveys*, vol. 51, no. 6, 130:1–130:36, Jan. 28, 2019, ISSN: 0360-0300. DOI: 10.1145/3291047. [Online]. Available: https://doi.org/10.1145/3291047 (visited on 02/07/2023).

[5] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted Execution Environment: What It is, and What It is Not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, Helsinki, Finland: IEEE, Aug. 2015, pp. 57–64, ISBN: 978-1-4673-7952-6. DOI: 10.1109/Trustcom.2015.357. [Online]. Available: http://ieeexplore.ieee.org/document/7345265/ (visited on 02/13/2023).

[6] "Trusted Execution Environment (TEE) Committee," GlobalPlatform. (), [Online]. Available: https://globalplatform.org/technical-committees/trusted-execution-environment-tee-committee/ (visited on 02/15/2023).

[7] "Samsung Knox." (), [Online]. Available: https://docs.samsungknox.com/admin/whitepaper/kpe/samsung-knox.htm (visited on 02/20/2023).

[8] "About – Confidential Computing Consortium." (), [Online]. Available: https://confidentialcomputing.io/about/ (visited on 03/28/2023).

[9] "Introduction to Trusted Execution Environments," GlobalPlatform. (), [Online]. Available: https://globalplatform.org/resource-publication/introduction-to-trusted-execution-environments/ (visited on 02/15/2023).

[10] "Open Portable Trusted Execution Environment," Linaro. (), [Online]. Available: https://www.op-tee.org/ (visited on 02/20/2023).

[11] V. Costan and S. Devadas. "Intel SGX Explained." (2016), [Online]. Available: https://eprint.iacr.org/2016/086 (visited on 02/20/2023), preprint.

[12] "MultiZone Security TEE for RISC-V," Hex Five Security. (Jun. 30, 2019), [Online]. Available: https://hex-five.com/multizone-security-tee-riscv/ (visited on 03/28/2023).

[13] "CWE - CWE-121: Stack-based Buffer Overflow (4.10)." (), [Online]. Available: https://cwe.mitre.org/data/definitions/121.html (visited on 02/20/2023).

[14] "CWE - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (4.10)." (), [Online]. Available: https://cwe.mitre.org/data/definitions/78.html (visited on 02/20/2023).

[15] J. Ménétrey, C. Göttel, A. Khurshid, *et al.*, "Attestation Mechanisms for Trusted Execution Environments Demystified," in *Distributed Applications and Interoperable Systems*, D. Eyers and S. Voulgaris, Eds., vol. 13272, Cham: Springer International Publishing, 2022, pp. 95–113, ISBN: 978-3-031-16092-9. DOI: 10.1007/978-3-031-16092-9_7. [Online]. Available: https://link.springer.

com/10.1007/978-3-031-16092-9_7 (visited on 02/10/2023).

[16] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan, "Remote ATtestation procedureS (RATS) Architecture," RFC Editor, RFC9334, Jan. 2023, RFC9334. DOI: 10.17487/RFC9334. [Online]. Available: https://www.rfc-editor.org/info/rfc9334 (visited on 02/20/2023).

[17] G. Coker, J. Guttman, P. Loscocco, *et al.*, "Principles of remote attestation," *International Journal of Information Security*, vol. 10, no. 2, pp. 63–81, Jun. 2011, ISSN: 1615-5262, 1615-5270. DOI: 10.1007/s10207-011-0124-7. [Online]. Available: http://link.springer.com/10.1007/s10207-011-0124-7 (visited on 02/13/2023).

[18] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, "TrustZone Explained: Architectural Features and Use Cases," in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, Pittsburgh, PA, USA: IEEE, Nov. 2016, pp. 445–451, ISBN: 978-1-5090-4607-2. DOI: 10.1109/CIC.2016.065. [Online]. Available: http://ieeexplore.ieee.org/document/7809736/ (visited on 02/13/2023).

[19] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf, "SANCTUARY: ARMing TrustZone with User-space Enclaves," in *Proceedings 2019 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2019, ISBN: 978-1-891562-55-6. DOI: 10.14722/ndss.2019.23448. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_01A-1_Brasser_paper.pdf (visited on 03/02/2023).

[20] "ARM Security Technology Building a Secure System using TrustZone Technology." (), [Online]. Available: https://developer.arm.com/documentation/PRD29-GENC-009492/c/TrustZone-Software-Architecture/Booting-a-secure-system/Secure-boot (visited on 03/01/2023).

[21] R. Wilkins and B. Richardson, "UEFI secure boot in modern computer security solutions," in *UEFI Forum*, 2013.

[22] J. Ménétrey, C. Göttel, M. Pasin, P. Felber, and V. Schiavoni. "An Exploratory Study of Attestation Mechanisms for Trusted Execution Environments." arXiv: 2204.06790 [cs]. (Apr. 15, 2022), [Online]. Available: http://arxiv.org/abs/2204.06790 (visited on 02/13/2023), preprint.

[23] "Understanding the Confidential Containers Attestation Flow." (), [Online]. Available: https://www.redhat.com/en/blog/understanding-confidential-containers-attestation-flow (visited on 04/17/2023).

[24] A. Ltd. "TrustZone for Cortex-A – Arm®," Arm | The Architecture for the Digital World. (), [Online]. Available: https://www.arm.com/technologies/trustzone-for-cortex-a (visited on 02/13/2023).

[25] A. Ltd. "TrustZone for Cortex-M – Arm®," Arm | The Architecture for the Digital World. (), [Online]. Available: https://www.arm.com/technologies/trustzone-for-cortex-m (visited on 02/13/2023).

[26] C. Shepherd, R. N. Akram, and K. Markantonakis, "Establishing Mutually Trusted Channels for Remote Sensing Devices with Trusted Execution Environments," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, Reggio Calabria Italy: ACM, Aug. 29, 2017, pp. 1–10, ISBN: 978-1-4503-5257-4. DOI: 10.1145/3098954.3098971. [Online]. Available: https://dl.acm.org/doi/10.1145/3098954.3098971 (visited on 03/02/2023).

[27] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Journal on Computing*,

vol. 26, no. 5, pp. 1484–1509, Oct. 1997, ISSN: 0097-5397, 1095-7111. DOI: 10.1137/S0097539795293172. [Online]. Available: http://epubs.siam.org/doi/10.1137/S0097539795293172 (visited on 03/28/2023).

[28] J. Menetrey, M. Pasin, P. Felber, and V. Schiavoni, "WaTZ: A Trusted WebAssembly Runtime Environment with Remote Attestation for TrustZone," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, Bologna, Italy: IEEE, Jul. 2022, pp. 1177–1189, ISBN: 978-1-66547-177-0. DOI: 10.1109/ICDCS54860.2022.00116. [Online]. Available: https://ieeexplore.ieee.org/document/9912246/ (visited on 04/06/2023).

[29] "WebAssembly." (), [Online]. Available: https://webassembly.org/ (visited on 04/06/2023).

[30] "Security at the Edge ⋆ Scalys." (), [Online]. Available: https://scalys.com/ (visited on 04/05/2023).

[31] "Networking ecosystem ⋆ Scalys - NXP Layerscape TrustBox Edge," Scalys. (), [Online]. Available: https://scalys.com/solutions/networking-ecosystem/ (visited on 02/13/2023).

[32] "Trustbox Family ⋆ Scalys - Cyber secure IoT Edge devices," Scalys. (), [Online]. Available: https://scalys.com/solutions/networking-ecosystem/trustbox-edge/ (visited on 04/05/2023).

[33] *Open Enclave SDK*, Open Enclave, May 11, 2023. [Online]. Available: https://github.com/openenclave/openenclave (visited on 05/12/2023).

[34] "Veraison," GitHub. (), [Online]. Available: https://github.com/veraison (visited on 05/12/2023).

[35] "Rust Teaclave TrustZone Sdk," GitHub. (), [Online]. Available: https://github.com/apache/incubator-teaclave-trustzone-sdk (visited on 04/14/2023).

[36] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal Hardware Extensions for Strong Software Isolation,"

[37] S. Zhao, Q. Zhang, Y. Qin, W. Feng, and D. Feng, "SecTEE: A Software-based Approach to Secure Enclave Architecture Using TEE," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, London United Kingdom: ACM, Nov. 6, 2019, pp. 1723–1740, ISBN: 978-1-4503-6747-9. DOI: 10.1145/3319535.3363205. [Online]. Available: https://dl.acm.org/doi/10.1145/3319535.3363205 (visited on 03/29/2023).

[38] M. M. Quaresma, "TrustZone based Attestation in Secure Runtime Verification for Embedded Systems," 2020. [Online]. Available: https://mquaresma.github.io/assets/dissertation.pdf.

[39] S. Hristozov, J. Heyszl, S. Wagner, and G. Sigl, "Practical Runtime Attestation for Tiny IoT Devices," in *Proceedings 2018 Workshop on Decentralized IoT Security and Standards*, San Diego, CA: Internet Society, 2018, ISBN: 978-1-891562-51-8. DOI: 10.14722/diss.2018.23011. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/07/diss2018_11_Hristozov_paper.pdf (visited on 03/02/2023).

[40] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "SWATT: Software-based attestation for embedded devices," in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004.*, Berkeley, CA, USA: IEEE, 2004, pp. 272–282, ISBN: 978-0-7695-2136-7. DOI: 10.1109/SECPRI.2004.1301329. [Online]. Available: http://ieeexplore.ieee.org/document/1301329/ (visited on 04/11/2023).

[41] I. D. O. Nunes, K. Eldefrawy, and N. Rattanavipanon, "VRASED: A Verified Hardware/Software Co-Design for Remote Attestation," in *Proceedings of the 28th USENIX Conference on Security Symposium*, ser. SEC'19, Santa Clara, CA, USA: USENIX Association, 2019, pp. 1429–1446, ISBN: 978-1-939133-06-9.

[42] T. Abera, N. Asokan, L. Davi, *et al.* "C-FLAT: Control-FLow ATtestation for Embedded Systems Software." arXiv:

1605.07763 [cs]. (Aug. 17, 2016), [Online]. Available: http://arxiv.org/abs/1605.07763 (visited on 04/25/2023), preprint.

[43] G. Scopelliti, S. Pouyanrad, J. Noorman, F. Alder, F. Piessens, and J. T. Mühlberg, "POSTER: An Open-Source Framework for Developing Heterogeneous Distributed Enclave Applications," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, Virtual Event Republic of Korea: ACM, Nov. 12, 2021, pp. 2393–2395, ISBN: 978-1-4503-8454-4. DOI: 10.1145/3460120.3485341. [Online]. Available: https://dl.acm.org/doi/10.1145/3460120.3485341 (visited on 04/25/2023).

[44] A. Ferraiuolo, A. Baumann, C. Hawblitzel, and B. Parno, "Komodo: Using verification to disentangle secure-enclave hardware from software," in *Proceedings of the 26th Symposium on Operating Systems Principles*, Shanghai China: ACM, Oct. 14, 2017, pp. 287–305, ISBN: 978-1-4503-5085-3. DOI: 10.1145/3132747.3132782. [Online]. Available: https://dl.acm.org/doi/10.1145/3132747.3132782 (visited on 03/29/2023).

[45] M. Kylänpää and A. Rantala, "Remote Attestation for Embedded Systems," in *Security of Industrial Control Systems and Cyber Physical Systems*, A. Bécue, N. Cuppens-Boulahia, F. Cuppens, S. Katsikas, and C. Lambrinoudakis, Eds., vol. 9588, Cham: Springer International Publishing, 2016, pp. 79–92, ISBN: 978-3-319-40384-7 978-3-319-40385-4. DOI: 10.1007/978-3-319-40385-4_6. [Online]. Available: http://link.springer.com/10.1007/978-3-319-40385-4_6 (visited on 04/25/2023).

[46] J. H. Ostergaard, E. Dushku, and N. Dragoni, "ERAMO: Effective Remote Attestation through Memory Offloading," in *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, Rhodes, Greece: IEEE, Jul. 26, 2021, pp. 73–80, ISBN: 978-1-66540-285-9. DOI: 10.1109/CSR51186.2021.9527978. [Online]. Available: https://ieeexplore.ieee.org/document/9527978/ (visited on 04/25/2023).

[47] W. Li, H. Li, H. Chen, and Y. Xia, "AdAttester: Secure Online Mobile Advertisement Attestation Using TrustZone," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, Florence Italy: ACM, May 18, 2015, pp. 75–88, ISBN: 978-1-4503-3494-5. DOI: 10.1145/2742647.2742676. [Online]. Available: https://dl.acm.org/doi/10.1145/2742647.2742676 (visited on 02/10/2023).

[48] U. Lee and C. Park, "SofTEE: Software-Based Trusted Execution Environment for User Applications," *IEEE Access*, vol. 8, pp. 121 874–121 888, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3006703. [Online]. Available: https://ieeexplore.ieee.org/document/9131703/ (visited on 02/13/2023).

[49] *Mbed TLS project*, Mbed TLS, Apr. 12, 2023. [Online]. Available: https://github.com/Mbed-TLS/mbedtls (visited on 04/13/2023).

[50] C. J. F. Cremers, "The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols," in *Computer Aided Verification*, A. Gupta and S. Malik, Eds., vol. 5123, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 414–418, ISBN: 978-3-540-70543-7 978-3-540-70545-1. DOI: 10.1007/978-3-540-70545-1_38. [Online]. Available: http://link.springer.com/10.1007/978-3-540-70545-1_38 (visited on 04/11/2023).

[51] "Rust Programming Language." (), [Online]. Available: https://www.rust-lang.org/ (visited on 04/13/2023).

[52] "OP-TEE with Rust — OP-TEE documentation documentation." (), [Online]. Available: https://optee.readthedocs.io/en/latest/building/optee_with_rust.html (visited on 04/13/2023).

[53] G. Scopelliti, S. Pouyanrad, J. Noorman, *et al.*, "End-to-End Security for Distributed Event-Driven Enclave Applications on Heterogeneous TEEs," *ACM Transactions on Privacy and Security*, Apr. 13, 2023, ISSN: 2471-2566. DOI: 10.1145/3592607. [Online]. Available: https://dl.acm.org/doi/10.1145/3592607 (visited on 05/02/2023).

[54] N. Asokan, F. Brasser, A. Ibrahim, *et al.*, "SEDA: Scalable Embedded Device Attestation," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Denver Colorado USA: ACM, Oct. 12, 2015, pp. 964–975, ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813670. [Online]. Available: https://dl.acm.org/doi/10.1145/2810103.2813670 (visited on 05/04/2023).

[55] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," in *Advances in Cryptology — CRYPTO '87*, C. Pomerance, Ed., red. by G. Goos, J. Hartmanis, D. Barstow, *et al.*, vol. 293, Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 369–378, ISBN: 978-3-540-18796-7 978-3-540-48184-3. DOI: 10.1007/3-540-48184-2_32. [Online]. Available: http://link.springer.com/10.1007/3-540-48184-2_32 (visited on 06/08/2023).

[56] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System,"

[57] R. Dahlberg, T. Pulls, and R. Peeters, "Efficient Sparse Merkle Trees," in *Secure IT Systems*, B. B. Brumley and J. Röning, Eds., vol. 10014, Cham: Springer International Publishing, 2016, pp. 199–215, ISBN: 978-3-319-47559-2 978-3-319-47560-8. DOI: 10.1007/978-3-319-47560-8_13. [Online]. Available: http://link.springer.com/10.1007/978-3-319-47560-8_13 (visited on 06/08/2023).

[58] "The Noise Protocol Framework." (), [Online]. Available: http://www.noiseprotocol.org/noise.html#dh-functions-cipher-functions-and-hash-functions (visited on 06/27/2023).

[59] B. Dowling, P. Rösler, and J. Schwenk, "Flexible Authenticated and Confidential Channel Establishment (fACCE): Analyzing the Noise Protocol Framework," in *Public-Key Cryptography – PKC 2020*, A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, Eds., vol. 12110, Cham: Springer International Publishing, 2020, pp. 341–373, ISBN: 978-3-030-45373-2 978-3-030-45374-9. DOI: 10.1007/978-3-030-45374-9_12. [Online]. Available: https://link.springer.com/10.1007/978-3-030-45374-9_12 (visited on 06/27/2023).

[60] Meta, *WhatsApp Encryption Overview: Technical White Paper*. [Online]. Available: https://scontent-ams2-1.xx.fbcdn.net/v/t39.8562-6/328495424_498532869106467_756303412205949548_n.pdf?_nc_cat=104&ccb=1-7&_nc_sid=ad8a9d&_nc_ohc=qCPjUO66XgcAX-_LgsV&_nc_ht=scontent-ams2-1.xx&oh=00_AfAZihFjwQFNipTCWjODlA8i0I6Yoi0E5zDl5cXzQcbBtQ&oe=64A01EFC (visited on 06/27/2023).

[61] J. A. Donenfeld. "WireGuard: Fast, modern, secure VPN tunnel." (), [Online]. Available: https://www.wireguard.com/ (visited on 06/27/2023).

[62] *Lightning Network In-Progress Specifications*, Lightning Network, Jun. 25, 2023. [Online]. Available: https://github.com/lightning/bolts (visited on 06/27/2023).