Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Mariam Moustafa

Remote Attestation for Constrained Relying Parties

Master's Thesis
Espoo, July 1, 2023

Supervisors:	Nicola Dragoni, Denmark Technical University
	Lachlan Gunn, Aalto University
Advisor:	Arto Niemi, Huawei Technologies Oy

Aalto University
School of Science

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Mariam Moustafa		
Title: Remote Attestation for Constrained Relying Parties			
Date:	July 1, 2023	**Pages:**	120
Major:	Computer Science	**Code:**	SCI3113
Supervisors:	Nicola Dragoni Lachlan Gunn		
Advisor:	Arto Niemi		

In today's interconnected world, which contains a massive and rapidly growing number of devices, it is important to have security measures that detect unexpected or unwanted behavior of those devices. Remote attestation – a procedure for evaluating the software and hardware properties of a remote entity – is one of those measures. Remote attestation has been used for a long time in Mobile Device Management solutions to assess the security of computers and smartphones. The rise of the Internet of Things (IoT) introduced a new research direction for attestation, which involves IoT devices.

The current trend in the academic research of attestation involves a powerful entity, called "verifier", attesting and appraising a less powerful entity, called "attester". However, academic works have not considered the opposite scenario, where a resource constrained device needs to evaluate the security of more powerful devices. In addition, these works do not have the notion of a "relying party" – the entity that receives the attestation results computed by the verifier to determine the trustworthiness of the attester.

There are many scenarios where a resource constrained device might want to evaluate the trustworthiness of a more powerful device. For example, a sensor or wearable may need to assess the state of a smartphone before sending data to it, or a network router may allow only trusted devices to connect to the network.

The aim of this thesis is to design an attestation procedure suitable for constrained relying parties. Developing the attestation procedure is done through analyzing possible attestation result formats found in the industry, benchmarking the suitable formats, proposing and formally analyzing an attestation protocol for constrained relying parties, and implementing a prototype of a constrained relying party.

Keywords:	remote attestation, formal verification, attestation results, encoding, serialization, attestation protocol, relying party, constrained devices, benchmark
Language:	English

Acknowledgements

I would like to take this opportunity to express my sincere gratitude and appreciation to all those who have supported me throughout this thesis. Their guidance, assistance, and encouragement have been invaluable, and I am truly grateful for their presence during my thesis.

First of all, I would like to thank my advisor Arto Niemi (Huawei Technologies) for his constant help, insightful suggestions, and constructive criticism that have significantly contributed to the overall strength of my thesis. I would also like to thank my supervisors, Professor Nicola Dragoni, for guiding me to create a more scientific and well-motivated thesis, and Lachlan Gunn whose work in platform security helped me understand the necessary background related to attestation. Furthermore, I would like to thank Professor Tuomas Aura for helping me in the formal verification of the proposed protocol and Senior Researcher Philip Ginzboorg (Huawei Technologies) for his invaluable assistance in improving the quality of the research.

I am sincerely grateful to my family for their constant support and motivation. Finally, I would like to conclude with a verse of the Quran that is very dear to my heart:

"O believers! Patiently endure, persevere, strengthen each other, and be mindful of Allah so that you may find success" 3:200

Espoo, July 1, 2023

Mariam Moustafa

Abbreviations and Acronyms

APCR-LPM	Attestation Protocol for Constrained Relying party Live Passport Model
BLE	Bluetooth Low Energy
CBOR	Concise Binary Object Representation
CDDL	Concise Data Definition Language
DoS	Denial of Service
DICE	Device Identifier Composition Engine
EAR	EAT Attestation Result
EAT	Entity Attestation Token
IoT	Internet of Things
JCS	JSON Canonicalization Scheme
MS-DHA	Microsoft Device Health Attestation
RA	Remote Attestation
RATS	Remote ATtestation procedureS
REE	Rich Execution Environment
RTOS	Real-Time Operating System
TEE	Trusted Execution Environment
TPM	Trusted Platform Module

Contents

Abbreviations and Acronyms **4**

1 Introduction **8**

1.1 Thesis Context 9
1.2 Problem Statement 10
1.3 Project Scope 11
1.4 Contributions 11
1.5 Thesis Structure 12

2 Background **15**

2.1 Attestation in Practice 16
2.2 Attestation 17
2.2.1 Attestation Participants and Artifacts 18
2.2.2 Data Flow 21
2.2.2.1 Background Check Model 21
2.2.2.2 Passport Model 22
2.3 Attestation Results 22
2.3.1 Non-repudiable Identity 24
2.3.2 Trustworthiness Claims 24
2.3.3 Claim Freshness 25
2.4 Attestation Software - Veraison 25
2.5 EAR Message Format 26
2.6 Serialization 30
2.6.1 Binary vs. Text-based Serialization 30
2.6.2 Schema-less vs Schema-driven Serialization 31
2.7 Studied Encoding Formats 31

3 Survey of Potential Attestation Result Formats **36**

3.1 Selection Criteria 36
3.1.1 Canonicality 37
3.1.2 Programming Language Independence 39

3.1.3 Standardization 39
3.1.4 Format Versatility 39
3.2 Comparison Results 40
3.3 Related Work 42

4 Implementation and Benchmarking 47
4.1 Comparison Metrics 47
4.2 Benchmarking Inputs 48
4.3 Implementation of Applications 49
4.3.1 Environment 50
4.3.2 Application Structure 50
4.3.3 ASN.1 DER 51
4.3.4 CBOR 54
4.3.5 JSON 55
4.4 Benchmarking 55
4.4.1 Object Size 56
4.4.2 Processing Speed 57
4.4.3 Memory Consumption 58
4.5 Analysis 60

5 Attestation Protocol for Constrained RP 63
5.1 System Model 63
5.2 Adversary Model 64
5.3 Security Requirements 65
5.4 Protocol Description 65
5.4.1 Initial State 66
5.4.2 Description of Initial Protocol 67
5.5 Formal Verification of APCR-LPM 70
5.5.1 Modelling with ProVerif 70
5.5.2 Modeling Participants' Behavior 70
5.5.3 Defining Security Queries 73
5.5.4 Query Results 75
5.5.5 Introducing another Attester 75
5.5.6 Attack Description 77
5.6 Improving the Protocol 78
5.7 Analysis 80
5.8 Related Work 82
5.8.1 Methodology 82
5.8.2 Related Protocols 82

6 Prototype Implementation **86**
6.1 Board Specifications 86
6.2 Connection Setup 87
6.3 RP Implementation 87
6.3.1 Key Management 87
6.3.2 Sending an Attestation Request 88
6.3.3 Receiving an Attestation Result 89
6.4 Results and Analysis 89

7 Discussion **91**

8 Conclusion **96**

A Benchmarking Inputs **107**
A.1 Input 2 (Two Attesters) 107
A.2 Input 3 (Raw Evidence) 108

B Benchmarking Code **110**

C ProVerif Models **111**
C.1 Model of APCR-LPM 111
C.2 Model of Attack on APCR-LPM 113
C.3 Model of Improved Protocol 116

D Prototype Implementation Code **120**

Chapter 1

Introduction

The Internet has become a tool for exchanging information between numerous and different types of devices. Studying how to protect this information exchange and to ensure that the information originated from a trusted source is of paramount importance. Information protection is not the only factor to be considered. The source generating the information must also be trusted. Platform security is concerned with ensuring that the computing platform (both hardware and software) is secure and behaving as expected [30].

One method to evaluate the security of a platform is remote attestation. Remote attestation is a procedure that allows the evaluation of some properties or behavior of a device. Those properties or behavior can be key possession, loaded code in memory, running processes or any other application dependent metric. Attestation has been used in various applications in both industry and academia. Large device manufacturers like Microsoft, Google, Apple, and Huawei use attestation to evaluate the security of devices. Their solutions enable evaluating the state of the devices that request online services, so that outdated or compromised devices cannot misuse resources.

Meanwhile, remote attestation has gained popularity in academia due to the rise of the Internet of Things (IoT). IoT devices tend to have limited capabilities and are deployed on a large scale. Academic research has so far focused on how remote attestation can be used to efficiently evaluate the security of those small IoT devices.

Remote attestation architectures generally comprise of three main active entities. The attester, the relying party, and the verifier. The attester, sometimes referred to as the prover, is the entity that needs to be evaluated. It can be a small IoT device, a smartphone or a laptop. The relying party is an entity that needs to know the state of the attester either to grant the attester access to some resource or to ensure that the environment has not been compromised. The third active entity is the verifier which performs the actual evaluation of the attester and relies the information to the relying party. The current literature focuses on the interaction between the attester and verifier and completely abstracts the notion of the relying party. Furthermore, the research focuses on the scenarios where the verifier is a powerful device and the attester is a small device.

The aim of this thesis is to bridge the research gap in attestation for constrained relying parties. The entity that performs the evaluation is not necessary the entity that needs the evaluation. The distinction between these roles has not been thoroughly studied in the literature. Furthermore, no attestation schemes have been proposed for scenarios where the relying party is a constrained device. Such scenarios are frequently occurring especially in devices like sensors or wearables that might need to authenticate a device before sending information to it.

The constrained relying parties this thesis studies conform to the definition of constrained nodes provided by the Internet Engineering Task Force (IETF). The IETF classifies constrained nodes into three classes [16]:

- Class 0: very constrained devices that cannot communication directly with the Internet and has less than 10 KiB of RAM and less than 100 KiB of flash (code).
- Class 1: constrained devices that can only use protocol stacks specifically designed for constrained nodes. Those nodes have approximately 10 KiB of RAM and 100 KiB of flash.
- Class 2: less constrained devices that are able to support the same protocol stacks used in more powerful devices. They tend to have 50 KiB of RAM and 250 KiB of flash.

Creating attestation applications for these device classes especially class 0 and class 1 is a challenge. The relying party applications need to at least verify that the results it received came from a trusted verifier, decode the attestation results, and process the content of the results. The purpose of this thesis is to study and present remote attestation schemes that will allow even class 0 devices acting as relying parties to benefit from remote attestation.

1.1 Thesis Context

This thesis is done as part of Erasmus Mundus Joint Master's program SECCLO (Security and Cloud Computing) in collaboration between Aalto University in Finland and Denmark Technical University (DTU) in Denmark.

The thesis is also done during an internship in Huawei's Helsinki System Security Lab (HSSL) which is part of Huawei's Finland Research Center. The center has been working on the security of consumer devices for more than a decade and has contributed to developing hardware-assisted trusted environments.

This thesis is done as part of the HSSL's research of remote attestation for consumer devices. The target of this project is to analyze and develop attestation applications that can be deployed on Huawei's open-source operating system OpenHarmony. The operating system uses component-based design that enables flexible integration of different components depending on a device's hardware capabilities [40]. The thesis focuses on attestation applications that work on the Mini device class. Devices that belong to this class have micro controller units and memory in the order of kilobytes.

1.2 Problem Statement

The study of attestation results produced by the verifier is an underdeveloped area.

1. An impartial evaluation of the strengths and weaknesses of the formats that can encode attestation results is missing. The attestation result format is an important area to investigate as the attestation results message size has a direct impact on the device's buffer capacity and transmission bandwidth. These resources cannot be taken for granted in devices like that of class 0.

2. The attestation results need to be protected when communicated between the verifier and relying party. There is a gap in the research that relates to attestation schemes for constrained relying parties. Most proposed attestation protocols utilize public-key cryptography for integrity and authentication. However, a typical public-key cryptography implementation consumes more resources compared to symmetric cryptography implementation: not only tens of kilobytes of code in the memory, but also more processing power, and more energy [39].

Therefore, there is a need to present an attestation solution that protects attestation results in an efficient manner using symmetric cryptography.

The problem this thesis targets can be expressed as follows: **Can small devices benefit from remote attestation to assess the security of more powerful devices.** Attestation is a wide topic with multiple principles working together. In order to determine whether a small device can make use of attestation those different principles must be studied in the context of constrained relying parties. The problem statement can be further divided into research questions that tackle the suitable representation and encoding of attestation results as well as how the attestation results can be protected in a constrained environment. The research questions are as follows:

- **RQ1:** What are the important features that should be present when encoding attestation results and what are the existing encoding formats that have those features?
- **RQ2:** What is the most suitable attestation results format in a constrained environment?
- **RQ3:** Are there existing attestation protocols designed for constrained relying parties?
- **RQ4:** How can a remote attestation protocol be resource efficient while meeting the security requirements needed in an attestation solution?
- **RQ5:** Are the proposed solutions ensured to work in a constrained environment?

1.3 Project Scope

This thesis is concerned with the analysis of attestation result formats and the communication of those attestation results to a constrained relying party. The attester's metric collection mechanisms as well as the verification mechanisms are out of the scope of this thesis.

1.4 Contributions

The contributions of this thesis can be summarized as follows:

- **C1:** Provide an impartial evaluation of the strength and weaknesses of available formats for encoding attestation results.
- **C2:** Implement and benchmark the encoding formats to determine the most suitable one for a constrained relying party.
- **C3:** Survey remote attestation protocols designed for constrained relying parties.

- **C4:** Propose and formally verify a new attestation protocol for constrained relying parties based on symmetric cryptography.
- **C5:** Implement a remote attestation solution that combines the attestation result format encoding with the symmetric cryptography-based attestation protocol.

Some of the results of this thesis work have been published:

- The paper "Platform Attestation in Consumer Devices" [62] submitted to the 2023 33rd Conference of Open Innovations Association (FRUCT) includes attestation result discussions from this work.
- A patent application "Apparatus and Method for Remote Attestation Using Symmetric Keys PCT/EP2023/066367" where the author is a co-inventor has been filed. The patent is a variant of the protocol described in this thesis.

Furthermore, a paper presenting the novel symmetric encryption-based attestation protocol for constrained relying parties will be submitted to the NordSec 2023 conference.

1.5 Thesis Structure

- **Chapter 2**

  This chapter explains the concepts and terms required to understand the thesis. It gives an overview of attestation from how it is currently used in the industry, the actors in an attestation procedure, then it narrows down the discussion to attestation results. It also discusses the encoding formats available in the industry.

- **Chapter 3**

  This chapter provides an impartial analysis of the strength and weaknesses of possible encoding formats for attestation results. It answers RQ1 and provide all the content relevant to achieving C1.

- **Chapter 4**

  This chapter describes an implementation for a subset of the formats surveyed in chapter 3. The subset consists of those formats that were deemed suitable for constrained environments. The implementations are also benchmarked in terms of message size, encoding speed, decoding speed and memory footprint to determine which is the most suitable format for constrained devices. The work in this chapter is relevant to C2 and answers RQ2.

- **Chapter 5**

  This chapter proposes a symmetric encryption-based attestation protocol specifically designed for constrained relying parties. The protocol is initially proposed by HSSL and was modelled and improved as part of this thesis work. In addition, the chapter analyzes the importance and novelty of the protocol by surveying similar attestation protocols in the literature. The work in this chapter is relevant to contributions C3 and C4 and answers RQ3 and RQ4.

- **Chapter 6**

  This chapter provides an implementation that combines the work done in chapter 4 and 5. The purpose to highlight the feasibility of having an attestation application running on a constrained relying party. The work in this chapter is related to C6 and answers RQ6.

- **Chapter 7**

  This chapter analyzes the results and the work of the entire thesis and gives an explicit answer to the main problem this thesis is trying to solve.

Figure 1.1 summarizes the relation between the contributions and chapters and shows where each research question is answered.

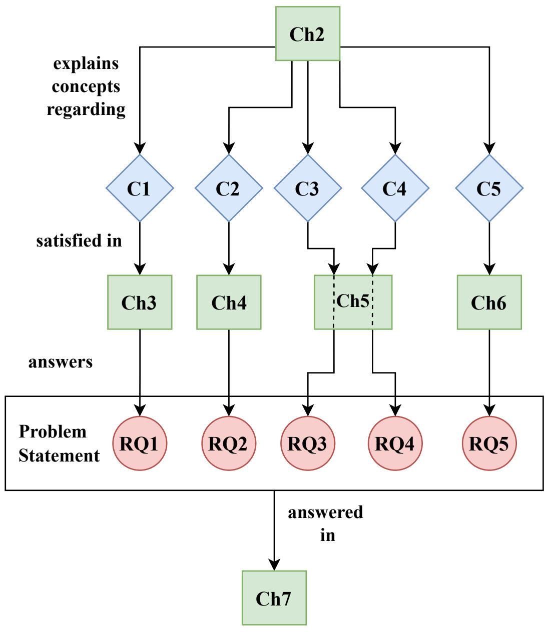


Figure 1.1: Thesis structure and relation between research questions, contributions, and chapters.

Chapter 2

Background

Attestation has been studied and investigated since the early 2000s [80]. The first attestation methods were based on the Trusted Platform Module[1] [77], specified by the Trusted Computing Group [35] to attest boot-time identities of system software. It has gained popularity in various other applications like Mobile Device Management (MDM) and Internet of Things (IoT). MDM solutions use attestation to validate the state of a mobile devices based on information like the current version of the device's operating system or whether a device has been rooted.

Many organizations are using attestation in MDM to implement the zero-trust security model [72]. Zero-trust in the context of MDM means that devices should not be trusted simply because they belong to a particular vendor or are part of the organization's network. Instead, the devices are constantly evaluated and granted access to resources on a least-privilege per-request basis. The device's state is evaluated based on policies that are constantly developed, maintained and enforced [72]. Attestation is then found beneficial in MDM as it provides a way to evaluate a device's trustworthiness based on the specified policies.

There is a large number of vendors that use remote attestation in their MDM solutions [62]. Microsoft provides an attestation framework called Device Health Attestation (MS-DHA) [58] that is used to assess the boot state of Windows devices. MS-DHA can be used in any device that supports the TPM and HTTPS connections. In addition, it enables devices to report their security configuration in a tamper-resistant way. The state is later evaluated by the MDM server to determine whether the device is compliant or not.

[1]The TPM is a hardware chip with a cryptographic processor, volatile and non-volatile memory. It allows the device it is embedded in to provide data sealing, integrity checks [82], measurement reports, and attestation evidence.

Apple also provides its own framework called Managed Device Attestation [7] for evaluating the security properties of Apple devices. Before enrolling to an MDM, an organization's certificate authority can request an attestation of the enrolling device's properties to ensure the device's integrity. In addition, the MDA framework is also used by the MDM to acquire fresh attestation results after the initial certificate enrollment phase. Google also provides an attestation service called Play Integrity [33] to test the authenticity of an application, the phone using the application, or both. Play Integrity provides information on the application's certificate, whether the device is able to enforce application integrity and whether the user has a licensed and authentic version of the application. As another example, Huawei provides an attestation service called SysIntegrity [41] that evaluates the integrity of the devices in the Trusted Execution Environment (TEE).

These attestation frameworks are just a few examples of attestation being used in practice. Remote attestation of Internet of Things (IoT) devices has also been studied in the academic literature. The incorporation of small devices in IoT environments led to an increase in the attack surface as any vulnerable device with access to a system can be the infiltration point for attackers [4]. Therefore, RA has gained more interest as a way to help detect compromised devices in IoT networks. As a result of the increasing popularity of attestation, standardization efforts for attestation terminology, requirements, and architecture are being pursued to accommodate the diverse range of attestation applications found in both industry and research. A set of IETF RFCs for Remote ATtestation procedureS (RATS) [12] provides the conceptual framework and terminology that will be used in this thesis.

The remainder of this chapter describes all the concepts used throughout the thesis. It explains the goals and mechanisms of attestation, gives an example of an attestation results object, and discusses encoding formats that can be used to represent the attestation result object.

2.1 Attestation in Practice

Aside from MDM and mobile security solutions, there are numerous other applications [12] where remote attestation would facilitate the verification process and protect systems from attacks. For example, network operators might only want to grant full network access to devices that meet some definition of "hygiene". Therefore, they would need to get information about the state of the devices and verify the validity of this information. Hence, RA can be used in order to prevent vulnerable or compromised devices from entering the network. In this scenario, the attester is any kind of device that wants to connect to the network and the relying party might be, for example, a router or access point that grants access to the network. Trusted Network Connect [85] is an example of such a network framework. It allows the administrators of a network to set policies. Based on those policies access to the network can be granted or denied.

Attestation can also be used in Hardware Watchdog implementations. There is a class of malware that holds devices hostage and disables them from updating. One solution to this malware is to implement a watchdog in a protected environment like a Trusted Platform Module [34]. If this watchdog timer does not receive regular and fresh attestation results that contain information about the state of the device, then it can force a reboot. The attester in this scenario is any device that needs to be prevented from being a victim to such a malware. The relying party in this scenario is a watchdog that has the ability to reboot devices into a known operational state. The authors in [45] proposed a network node that provides network resilience, a feature useful in industrial IoT. This node utilizes security measures like remote attestation and hardware watchdogs to monitor the state of the network while reducing the downtime of production systems. The node contains an authenticated watchdog that is able to reset the system in case of an attack. Remote attestation is then used to report the firmware state to the Security Operations Center (SOC). If the authenticated watchdog does not receive a signed digest from the SOC in a certain period of time, the watchdog forces a network reboot.

A small IoT device like a sensor or a wearable also needs to verify the trustworthiness of the device it is transmitting the data to. One example of such small devices is a medical implant like a pacemaker or insulin pump that requires secure communication to a patient or doctor's device. Remote attestation can be used in these scenarios to validate the integrity of the device the medical implants are sending information to. An example of an attester in this case is a mobile phone and the sensor/implant is the relying party. This use case is more related to the scope of this thesis where the relying party is a resource-constrained device.

2.2 Attestation

Attestation is the process of verifying that an entity (called *attester*) is in an intended operating state before authorizing it to perform an action or to access a resource. To this end, the attester provides evidence that proves its trustworthiness to another entity called the relying party [63]. This evidence consists of a cryptographically protected set of claims about the attester and its environment. The relying party uses those claims to judge whether the attester is secure or not. The collection process of the claims sent by the target attester is done in a manner that is reliable such that the attester is not able to lie about its trustworthiness properties. Section 2.2.1 explains the different entities involved in the attestation procedure.

The end goal of attestation is to allow the relying party to decide whether to trust the attestation target or not. Trust is the choice the system makes about the attestation target. On the other hand, trustworthiness is the quality that the system measures to determine whether to trust the target [12]. Section 2.3 discusses in detail the parameters and notions that are used to represent trustworthiness of a device.

Based on this notion of trust and trustworthiness, it can be considered safe to trust a device/entity if the following conditions are met [56]:

1. The entity can be unambiguously identified. Which means that based on the provided identification, the attestation system can know exactly which entity it is communicating with.

2. The entity operates unhindered. Which means that there is nothing that stops or influences the expected behavior of the entity.

3. The system has first-hand information that the entity is displaying consistent and good behavior or it trusts someone that affirms the entity's good behavior.

All of these conditions are necessary in the attestation procedure in order to establish trust.

2.2.1 Attestation Participants and Artifacts

The RATS architecture [12] provides a standard terminology that can be applied to existing and emerging remote attestation procedures. These standard terms help unify the current and future research by defining standard roles and actions. The actors in a RATS architecture are said to "consume" and "produce" artifacts. Those actors can be divided into administrators who set policies and devices/entities who use these policies in the attestation protocol. The administrators set the rules based on which the other entities are able to prove and verify an attester's trustworthiness. The overall interaction between the different roles can be found in figure 2.1.

The **attester** is usually part of the target device that requires access to some resource or action. It produces **evidence** about its operating state to be appraised. The evidence may include configuration data, measurements, or inferences. The attester needs to have a trustworthy mechanism [62] that collects claims for the verifier to validate. Some literature use the terms attester and attestee to distinguish between the entity that is measuring and generating the evidence (attester) and the entity that is measured (attestee). Generally, the attester is implemented in a way that can be trusted where the implementation relies on some hardware component or isolation mechanism. There are generally three different categories of attesters that are able to create evidence in a trusted manner. Those types use different mechanisms to fulfil requirements 1-2 found in section 2.2. The three categories are:

1. Secure co-processors: these are hardware components separated from the device's CPU. There are many examples of such processors including Hardware Security Modules (HSM) which are traditionally used for key management and Trusted Platform Modules (TPMs) which are used to assess the integrity of devices during boot [38]. These modules are generally very restricted and cannot run arbitrary code. There are other secure processors like Memory Encryption Engine (MEE) [37] and Apple's Secure Enclave Processor (SEP) [5] that can run custom operating systems but typically only execute manufacturer-provided code. Co-processors offer the most secure form for implementing attestation mechanisms as it requires physical access to device to break such systems.

2. Processor secure environments: these utilize the secure mode of a devices CPU or use some isolation techniques to protect the attester from the rest of the system. This category can be further divided into Process-based, VM-based and Split-world architectures. In the process-based architecture, the attester is an individual process whose runtime memory is encrypted such that no other process can read or decrypt it. One example of such a category is SGX [2]. While in VM-based architectures, the attester is running on a guest virtual machine isolated from the rest of the system. One example of such an Attester type is Intel's TDX [3]. Finally, in Split-world architecture, a devices hardware and software resources can be split between secure and non-secure world. An example of such an architecture is ARM's TrustZone.

3. Software-based: these are software applications that perform attesta-

[2]Intel's Software Guard Extensions (SGX) is a set of processor extensions that enables application isolation [76].

[3]Intel's Trust Domains Extensions is a technology that helps deploy hardware-isolated virtual machines. It is usually used by cloud tenants to secure their data from cloud providers [44].

tion by relying on some features in how the device and operating system is loaded. For example, temporally isolated software attesters in the bootloader can perform attestation before the rest of the system is launched. Another example is software attesters that are isolated by the devices privilege rings where the kernel running in ring 0 can attest user processes running in ring 3.

These categories of attesters help guarantee that the evidence and claims extracted can be trusted since the attester collecting and measuring evidence is running in an isolated and tamper-resistant space. In addition, the keys used for encryption and hashing help prove the attester's identity.

The **verifier** is an entity that appraises the evidence produced by the attester. It assesses the trustworthiness of the attester based on the rules provided by the administrators. The rules include the **appraisal policy for evidence** which is provided by the **verifier owner** administrator and is used by the verifier to determine the validity of the evidence provided by the attester. The verifier may also consult a secure statement called **endorsement** that vouches for the capabilities of the attester. This Endorsement is provided by an **endorser**. In addition, the verifier also consults the **reference values** produced by the **reference value provider**. Both the endorser and the reference value provider are typically vendors.

One major difference between endorsements and reference values is that an endorsement is a statement provided by a vendor that vouches for the attester's integrity. While the reference values are a set of expected measurements that are used by the verifier to compare evidence. One example of an endorsement is a manufacturer certificate that signs a public key whose private key is only known to a device's hardware. Therefore, the manufacturer is guaranteeing that any message signed by that device's private key is done by the device's signing capabilities in the hardware [84]. On the other hand, reference values are the expected measurements or results that are provided by e.g. the vendor. For example, they might include version numbers of operating system and bootloader, list of trusted certificates, trusted temperature ranges, and trusted power supply characteristics. The verifier can then cross-reference the claims collected in the evidence with these known acceptable values. Based on the evidence, appraisal policy for evidence, endorsement, and reference values artifacts, the verifier creates the attestation results.

The **attestation results** contain information about the attestee and whether the verifier vouches for the trustworthiness of the attestee or not. The **relying party** consults the **appraisal policy for attestation results** which is a set of rules that guide the relying party on how to validate the attestation results produced by the verifier. Based on this appraisal policy and the values in the attestation results, the relying party determines whether it can reliably grant application-specific resources to the attestee. It is assumed that attestation architectures or protocols provide means of authentication for the relying party, verifier and the different vendors and manufacturers.

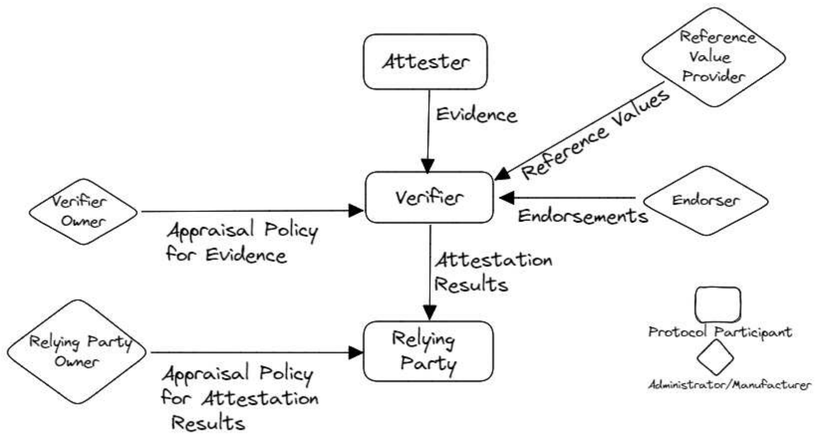


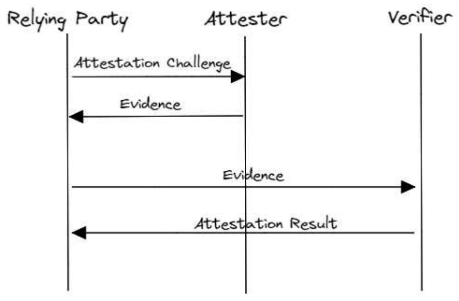
Figure 2.1: Remote Attestation architecture according to the RATS RFC

2.2.2 Data Flow

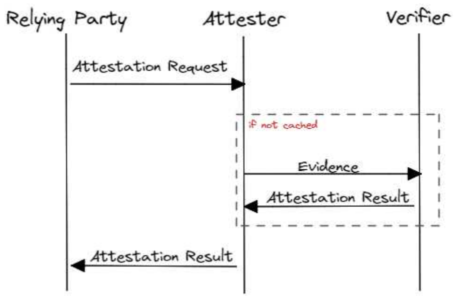
The RATS architecture specifies two generic models that describe the data flows during attestation. These data flow models include the Background Check and Passport models. The choice of model to use is dependent on the kind of system or environment RA is deployed in. In addition, these models are just examples on possible ways the relying party, attester, and verifier can communicate. Other interaction patterns also exist. For example, MS-DHA protocol is a combination of the two models presented here.

2.2.2.1 Background Check Model

In the background check reference model, shown in figure 2.2a, the attester sends the evidence directly to the relying party. In turn, the relying party forwards it to the verifier. It is analogous to an employer running a background check on its new employee before allowing the employee to start working in the company. The verifier attests the claims found in the evidence and compares the values to its appraisal policy. Based on the appraisal policy and claims, the verifier creates its verdict and adds it to the attestation results along with any other information required by the relying party. Finally, the



(a) Background-check Model



(b) Passport Model

Figure 2.2: Data flow models

verifier sends the attestation result to the relying party. This model might not be suitable for constrained relying party as it involves conducting two separate sessions with the attester and verifier.

2.2.2.2 Passport Model

Similar to the physical passport that is carried by citizens in order to be granted access to countries, the attester carries the attestation results produced by the verifier. The attester might not be able to understand the attestation results; it stores and forwards it to the relying party whenever the results are requested. The attester is also able to forward the cached attestation results to several relying parties. This model is suitable for resource-constrained relying parties as the relying party only communicates with the attester. Figure 2.2b illustrates the communication taking place between the attestation entities in the passport model. In real-world applications, the results include a timestamp and/or nonce that makes the results valid only for a certain period of time. If the results are older than this duration, the attester should produce a fresh set of claims and request new attestation results from the verifier.

2.3 Attestation Results

As discussed in 2.2.1, attestation results are consumed by the relying party and used to decide whether the attester can be authorized to conduct the requested action or not. The verifier produces the attestation results after appraising the evidence collected by the attester. The attestation results should be in a format that the relying party can understand.

According to RATS and Trusted Computing Group (TCG), attestation results cn be defined defined as:

- *"the output generated by a Verifier, typically including information about an Attester, where the Verifier vouches for the validity of the results".* (RATS RFC [12])
- *"messages containing the results of attestation Evidence appraisals. Attestation Results may contain Claims or other application specific Assertions meaningful to the Relying Party. Attestation Results are authenticated and integrity and confidentiality protected by the Verifier. Attestation Results from a Verifier are presumed to comply with Verifier Owner policies. Consequently, Attestation Results are actionable values in the context of the Relying Party."* (TCG [36])

These definitions essentially view attestation results as the verifier's assessment of the claims sent by the attesters.

By default, the relying party assumes that the attester, more specifically the attestee, is untrustworthy. The relying party needs to receive positive attestation results from the verifier and compare the attestation results to its own appraisal policy in order to decide the attester's trustworthiness. In contrast to attestation evidence, the information in the attestation results is more abstract, and does not depend so much on device or vendor-specific details. Therefore, it is easier to standardize. The relying party can have a simpler and interoperable appraisal policy based on a standardized attestation result format. One other difference between evidence and attestation results is that the evidence is signed by the attester (device). Whereas the attestation results are signed by the verifier. Hence, the relying party needs to establish a trust relationship only with a single verifier rather than a large set of attester entities.

The standardization efforts for attestation results the relying party to interact with many types of attesters without having to adapt to vendor-specific syntax [84]. The information found in the attestation results are independent of the constraints imposed by the different relying parties as the structure and type of information is already known.

The minimum amount of information the relying party needs from the attestation results is non-repudiable identity evidence, trustworthiness claims and claim freshness [84]. TThe attestation results should contain the identity of the attester or the identity of the verifier that vouches for the attester's identity. These identities should be non-repudiable meaning that the attester and verifier cannot deny having this identity. Non-repudiable identity is generally achieved by public-key cryptography where the private key acts as the device's identity. Attestations results should also contain the claims the verifier has appraised. Those include what the verifier's verdict is about the attester evidence. Finally, the attestation results should also include a measure of freshness where the relying party can know how old the attestation results are and whether it should request newer results. The following section provides more details about each of those three concepts.

2.3.1 Non-repudiable Identity

As a minimum requirement to satisfy non-repudiable identity, the relying party must be able to identify the verifier. The attester identity may then be verified through signed and encrypted communication with the verifier or using pre-provisioned attester public keys. In most cases, the relying party would need to verify the identity of both the verifier and attester to meet the non-repudiable identity requirement. Since the verifier is assumed to be trusted, its identity must have been provisioned by a trusted operator. As for the attester, the evidence is tied to the identity of the attester which is evaluated by the verifier. Therefore, the verifier includes sufficient identity evidence of the attester it has appraised before signing the attestation results. These two pieces of identity evidence are the bare minimum to meet the non-repudiable identity requirement. Other use cases might require evidence about how the initial trust with the verifier was established [84].

2.3.2 Trustworthiness Claims

The second piece of information that should be in the attestation result is the trustworthiness claims. These are conclusions the verifier reaches based on the evidence the attester produces. In other words, trustworthiness claims are the verifier's verdict of the attester evidence. The terminology might differ from one application to another. For example, in Google's Play Integrity attestation service, the verifier sends an "integrity verdict" about the application. The verdict acknowledges whether an application is recognized by Play Store and whether the device meets the integrity checks defined by Google, but it does not give much details about the state of the application or device [33]. There are other similar application-specific implementations for attestation frameworks. They all provide the relying party an assessment of the attester being attested. In some situations, a single value (trusted/not trusted) might be the only requirement needed from the trustworthiness claims. The more information the verifier provides about its decision the better, since it allows the relying party to know in which way an attester has been compromised.

2.3.3 Claim Freshness

In addition to a non-repudiation identity proof and verifier verdict, the relying party should know when the attestation results were issued by the verifier. There are two notions of attestation result freshness that need to be considered. The first relates to when the attestation results were created and the second relates to whether the relying party has seen the exact attestation result from a previous session. Timestamps are usually added in the attestation result to satisfy the first notion. The second notion targets situations where an attacker might replay an unexpired attestation result to the relying party. Nonces are generally used to mitigate these kinds of attacks as they are used only once so the relying party knows that the results are not replayed.

Some attestation implementations require the use of timestamps and have the nonces as an optional field. One example is Apple's MDA [7] solution which allows setting a nonce in the attestation request only once every seven days to limit the amount of resources required to generate and verify attestation claims and results. The EAR RFC [26] also has the nonce as an optional field while the issued at parameter as a mandatory one. The notion of freshness varies from one application to the other but it is generally recommended to use nonces for security purposes.

2.4 Attestation Software - Veraison

The verifier performs the complex task of comparing evidence against reference values and checking cryptographic signatures. The complexity arises, in part, because the evidence is highly dependent on the attester which can be any entity in the deployed system. In addition, in order to obtain the reference values, business relationships with authoritative sources need to be established [9]. The different deployments and authorities complicate the process of developing a consistent and standard attestation system. The open-source project Veraison [9] aims to solve that problem by creating components that abstract the verification and provisioning pipelines.

Veraison builds the software components that are used in Attestation Verification Services. The software implements the core structure for verification and provisioning. Specific attestation technologies are then incorporated with the use of plugins.

The Verification architecture in Veraison is presented in figure 2.3. In Veraison terminology, The verifier receives a token (data from attester) which is sent to the Veraison Trusted Services (VTS). The VTS then parses the token in order to create the Evidence. Then, the Verifier checks the Policies that have been registered for this particular attestation scheme and appraises the attestation results. Veraison also has a library that implements the EAR [26] RFC draft for attestation results. This library will be used as a guide when implementing and comparing the different attestation results encoding formats.

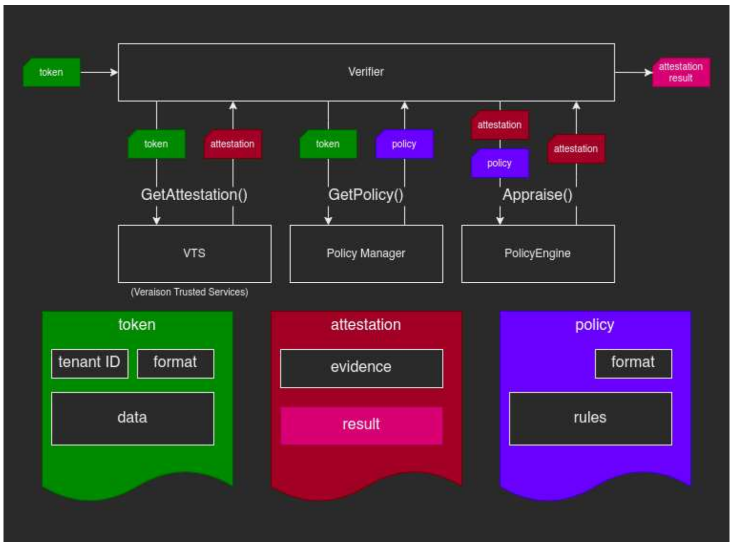


Figure 2.3: Veraison verification architecture. Image is taken from Veraison documentation [9].

2.5 EAR Message Format

The attestation results that are being studied and encoded in this thesis are based on the Entity attestation token Attestation Result (EAR) RFC draft [26]. These results are more generic and the claims are designed to allow a broad range of attesters, verifiers and relying parties to understand each other. EAR includes a trust tier assigned by the verifier. The trust tier is the status of an attester claim that a verifier deduces. The possible values for the trust tier are summarized in table 2.1.The trust tier is encoded in 8-bit integers in order to simplify the processing done by the relying party. The range of integers for each tier is given in table 2.1.

In addition to the overall status of the attestation, the EAR object also includes trustworthiness claims. These claims correspond to the trustworthiness claims concept discussed in section 2.3.2. They are designed to en-

Trust Tier	Definition	Values
None	The verifier makes no assertions regarding this aspect of trustworthiness	0: insufficient evidence 1: Unexpected values in evidence -1: verifier malfunction
Affirming	The verifier affirms that the attester supports this aspect of trustworthiness	2 to 31: standard reason for affirming -2 to -32: nonstandard reason
Warning	The verifier warns about this aspect of trustworthiness in the attester	32 to 95: standard reason for warning -33 to -96: nonstandard reason
Contrain-dicated	The verifier explicitly asserts that the attester is untrustwo-rthy regarding this aspect	96 to 127: standard reason for contraindication -97 to -128: nonstandard reason

Table 2.1: Possible trust tiers in attestation results

compass various systems in the attester from the attester's identity to the attester's file system. Each attestation claim is represented as an integer (the trust tier) and this integer value represents the verifier's verdict of the claim. For example, the number assigned to the claim can mean verifier malfunction, successful cryptographic validation or failed cryptographic validation. More details can be found in the EAR RFC [84]. The different claims the verifier may appraises include:

1. Configuration
2. Executables (runtime files, scripts, or other objects that are loaded into the Target environment's memory)
3. File-system
4. Hardware
5. Instance-identity (the attester's unique identity based on public key cryptography)
6. Runtime-opaque (the visibility of attester environments from the perspectives of outside party. This claim is to ensure that no unauthorized

party is able to read or manipulate the attester's trusted execution environment(s).)

7. Sourced-data (the integrity of objects in external systems that are used by the attester, these include any metrics the attester has collected about the target environment or received from other attesters)

8. Storage-opaque (the attester's ability to encrypt persistent storage)

The minimum information required to adhere to the EAR format include the EAT profile (version of claims set and encoding used), issued at time, verifier identification, and at least one attestation verdict found in the 'submods' object presented below. The following JSON object shows an example of an EAR claims-set. It corresponds to an attester "PSA" with the "contraindicated" trustworthiness tier.

```
{
  "eat_profile": "tag:github.com,2023:veraison/ear",
  "iat": 1666529184,
  "ear.verifier-id": {
    "developer": "https://veraison-project.org",
    "build": "vts 0.0.1"
  },
  "ear.raw-evidence": "NzQ3MjY5NzM2NTYzNzQK",
  "submods": {
    "PSA": {
      "ear.status": "contraindicated",
      "ear.trustworthiness-vector": {
        "instance-identity": 2,
        "executables": 96,
        "hardware": 2
      },
      "ear.appraisal-policy-id":
        "https://veraison.example/policy/1/60a0068d"
    }
  }
}
```

The values in the trustworthiness vector help explain why the attester is contraindicated. The instance identity and hardware both have a value 2 which means they are uncompromised and genuine. The configuration, file system, runtime opaque, storage opaque, and sourced data fields are not present which signifies that no claim has been made on those parameters. The executables parameter has value 96 making the attester contraindicated. The reason such a value would get assigned is that the attester has unauthorized code or configuration loaded in its runtime memory. An explanation for all possible values in a trustworthiness vector can be found in [84]. If there was another attester, then an additional object with its own status and trustworthiness vector would be added to the submods object. There are various real-world scenarios where multiple attesters would be present on the same device. For example, a server with CPU and GPU environments. EAR's attestation result schema also includes optional fields like raw evidence and nonces to allow for miscellaneous relying party requirements. This thesis uses EAR's attestation result format for comparing and benchmarking the different encoding schemes. It was chosen because it provides a comprehensive and open-source attestation result format that can be used by any project using Veraison or Veraison-compatible plugins.

Minimizing the parsing code is important for code footprint and attack surface reduction. This is even more necessary with constrained relying parties. Therefore, verifiers should send attestation results to relying parties in a format it already knows and can easily parse. In addition, decoupling the information sent in the attestation results from the encodings can allow a verifier to serialize the same attestation result using the format suitable for each of the relying parties it is communicating with. This decoupling will allow the standardization of the content of the attestation results, while providing the relying party with the format it is able to parse and use. Figure 2.4 shows how the verifier can accept evidence in different formats and create the corresponding attestation results in the format the relying party expects. An analysis and implementation of some of the most well-known formats is found in chapters 3 and 4 respectively.

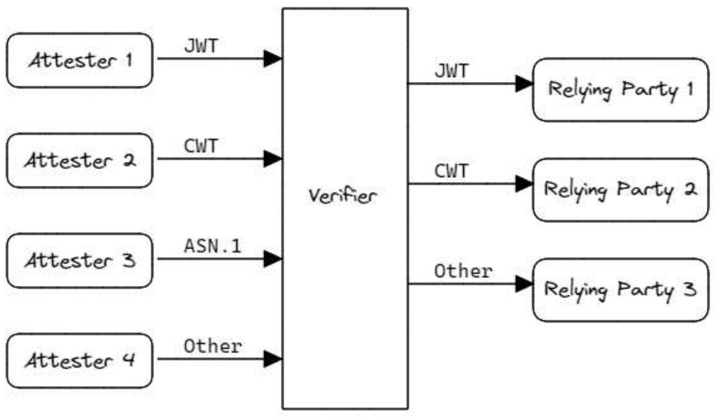


Figure 2.4: The Verifier consuming and producing different data formats

2.6 Serialization

The encoding formats for attestation results is an important factor to consider especially in constrained relying parties because factors like the size of the attestation result, the decoding code size and decoding speed are limited by the device specifications. These factors require certain storage capacity, transmission buffer capacity and RAM. The study of the effects of the attestation result format is still missing in literature. The EAR RFC draft provides conventions for CBOR and JSON encoding formats [26]. However, it does not give a reason why those were the chosen formats for representing attestation results.

Serialization, or *marshalling*, is the process of converting a data object or data structure into a sequence of bits for storage and transmission [83]. The inverse operation of converting the sequence of bits back to the original data structure is called deserialization, or unmarshalling. The term encoding refers to how data is going to be represented, for example, UTF-8 [86] is a type of text encoding. While serialization is converting the encoding into bit-strings based on the format's encoding rules.

2.6.1 Binary vs. Text-based Serialization

Serialization is generally categorized by how the information is represented. It can be either text-based or binary. The text-based serialization produces the bits corresponding to a sequence of characters in a text encoding such as ASCII or UTF-8. Text-based serializations are more user-friendly and ubiquitous because they can be read by most computer applications. On the other hand, binary serializations are more application-specific and usually require special parsers to understand the format. Those kinds of serializations are usually accompanied with documentation and special tools in order to deserialize them. Generally, binary formats are easier and faster for computers to serialize and deserialize [61].

One dummy example to illustrate the difference between binary and text-based serialization is serializing the number 127:

```
Text-based (UTF-8): 00110001 00110010 00110111
Binary: 1111111
```

The text-based serialization encodes each digit separately, while the binary can encode the number as a whole. Binary serializations are generally more compact, as this small example illustrates. Typically, they have some additional information related to the size and type of the serialized data. For example, in the binary format CBOR, there are 5 bits of additional information that state whether the unsigned integer will be represented in 8, 16, 32, or 64 bits [17]. Nevertheless, those 5 bits of additional information are constant regardless the size of the actual number being serialized.

2.6.2 Schema-less vs Schema-driven Serialization

Serialization can also be categorized into schema-less and schema-driven based on whether the serialization and deserialization processes require a formal description [83]. A schema is a definition or description of the data object being serialized. A serialization is schema-driven if a bit-string cannot be deserialized without knowledge of its structure. On the other hand, with schema-less serialization, a bit-string can be deserialized without any prior knowledge to the structure of the data. The following is an example of a high-level schema of a simple fictitious TODO application where each TODO item is split into tasks. The schema is based on the ASN.1 encoding format. Table 2.2 summarizes the differences between schema-driven and schema-less serializations.

```
TODO DEFINITIONS ::= BEGIN

    TODOItem ::= SEQUENCE {
        trackingNumber INTEGER,
        body           UTF8String
    }

    TODOTask ::= SEQUENCE {
        itemNumber             INTEGER,
        taskNumber             INTEGER,
        taskDescription        UTF8String,
        isDone                 BOOLEAN
    }

END
```

2.7 Studied Encoding Formats

The formats being analyzed and compared in this thesis are popular in the open-source community. In addition, they can easily represent attestation

Schema-driven	**Schema-less**
Custom schema for each serialization specification that cannot be used by other schema-driven formats	Easier to use because any bit-string can be deserialized without prior knowledge of the data
Avoids encoding structural information into the bit-string because it is already specified by the schema which results in space efficient bit-strings	Embed information about structure of data so that each serialization is standalone
Requires writing and maintaining schema definitions	No overhead with handling changes of schemas or considerations to backward and forward compatibility
Mostly used in network-efficient systems or systems with low bandwidth	Mostly used in loosely coupled systems where components don't have prior knowledge of structure of the data
Tend to be binary	Tend to be text-based

Table 2.2: Comparison between schema-driven and schema-less serializations

result objects similar to the one illustrated in section 2.5. The considered formats are Protocol Buffers, Flat Buffers, ASN.1 DER, X.509, CBOR, JSON, XML, Java Object Serialization and Pickle. While this is not a complete list of formats, it contains representatives of all types of serializations discussed in section 2.6. For example:

- XML is a schema-driven text-based format.
- JSON is a schema-less (generally) text-based format.
- CBOR, JOS, and Pickle are schema-less binary serialization.
- ASN.1 DER, Protocol Buffers and FlatBuffers are schema-driven binary serialization.

Abstract Syntax Notation One (ASN.1) is a language-independent notation for specifying data structures [52]. It is mainly used in computer networking. ASN.1 allows defining the structure of the data and the kind of information and data types included in this structure as shown in the ASN.1

schema in section 2.6.2. There are multiple encoding rules that can be used to represent the data defined by the schema as a sequence of bytes. Some of those rules use the Tag, Length, Value (TLV) approach. This approach specifies the tag defining the data type of the value being encoded, the number of bytes needed to encode the value, and the actual bytes encoding of the value. Even though most of the encoding rules use the TLV approach, they have different methods for encoding the Value part. The Basic Encoding Rules (BER) provides rules for encoding TLVs of indefinite length. While in Distinguished Encoding Rules (DER), the structures are always encoded to definite lengths with the exact bytes [52].

Concise Binary Object Representation (CBOR) is another binary format designed for small code and message size. It is also designed for extensibility such that any updates or changes to the specification would not render previous CBOR implementations unusable [17]. Unlike ASN.1, CBOR does not require a schema or any previous data structure definition. The first byte of every encoded value contains information about the value that is stored. More specifically the first byte specifies the data type of encoded value and how that value should be loaded (in case of unsigned integer data type).

JSON (JavaScript Object Notation) is a text-based format based on the data objects in the JavaScript programming language. Despite the name, JSON is a widely popular format that can be parsed by all programming languages used to create internet-enabled applications. The main reason why JSON is parse-able by all these programming languages is that it has become the main format for information exchange over the internet [68].

The Java programming language provides its own binary serialization mechanism called **Java Object Serialization (JOS)**. JOS is able to serialize any object defined in JAVA as long as the object implements or inherits the Serializable interface. The serialization is usually independent of the Java Virtual Machine being used [69], so, serialization and deserialization can happen in different platforms. Unlike JSON, this form of serialization can only be used if the intended application is written in Java.

Similar to JOS, Python has its own module called **Pickle** for serializing and deserializing Python objects. This form of binary serialization is guaranteed to be backwards compatible with all Python versions. In addition, Pickle is designed to be able to automatically represent a large number of Python's built-in types [27]. Both the encoding and decoding code needs to be written in Python for this serialization scheme to work.

Protocol Buffers (ProtoBuf) is a binary serialization format created by Google to handle structured data. ProtoBuf is designed to be faster and smaller than JSON and can be used for network traffic and persistent storage [32]. Like ASN.1, ProtoBuf requires the structure of the data to be defined beforehand and is suitable for data objects that are up to a few megabytes in size. In addition, ProtoBuf is primarily intended to provide a serialization mechanism that allows for seamless integration and changes to the defined schemas without breaking or changing existing implementations.

FlatBuffers is also another serialization scheme created by Google for encoding gaming data. It provides efficient cross-platform binary serialization intended for performance-critical applications as it flattens (hence the name) out large data structures. In addition, Flatbuffers serialization is able to access part of the object without unpacking the entire object [31]. However, FlatBuffers should only be used when extracting a few entities from a data object, as is typical in gaming. Its code for parsing the entire object is larger than that of other formats and tends to be slower [49].

Extensible Markup Language (XML) is another text-based format designed to be both human and machine readable. In XML, the information is enclosed in tags in order to provide structure to the encoded data. In addition, XML allows the users to define their own tags and add attributes to the tags that can provide additional information about the data. But this flexibility of creating tags can make parsing XML a complex and slow process [71].

The **X.509** standard defines the ASN.1 types for public key certificates. It is usually used for digital signatures and end-point authentication [75]. X.509 has been a standard since the 1980s and have various parsers that are already developed and tested extensively. Even though X.509 is technically not an encoding format, it is still used by almost all devices that connect to the internet since it defines how certificates are encoded. X.509 is considered and analyzed in this thesis because it provides extensions that could be used to represent attestation results.

Table 2.3 shows the formats used in some attestation solutions. EAR is a generic specification, while the rest are attestation results used in MDM solutions provided by major companies. While these companies utilize different formats, they don't give reason for their choices. Cloud-based and modern solutions mostly use the JSON format since JSON has become the most widely used in internet-enabled applications.

Attestation Framework	Format			
	Text-based		Binary	
	XML	JSON	ASN.1	CBOR
EAR [26]		✓		✓
Google Play Integrity [33]		✓		
MS Windows 10 DHA [58]	✓			
MS Windows 11 DHA [58]		✓		
Samsung Knox [74]		✓		
Huawei SysIntegrity [41]		✓		
Apple DeviceCheck [6]			✓	

Table 2.3: Attestation result formats in attestation solutions and frameworks

Chapter 3

Survey of Potential Attestation Result Formats

This chapter analyzes which of the following nine data formats (introduced in section 2.7) are suitable for the serialization of attestation results, i.e., encoding them in a serial form suitable for transmission and storage: JSON, FlatBuffer, JOS, Pickle, ProtoBuf, XML, X.509, ASN.1, CBOR. The primary analysis is qualitative as it is based on the criteria of (i) canonicality, (ii) programming language independence, (iii) versatility, and (iv) standardization.

Peeking forward, the most suitable serialization methods are JSON, ASN.1, and CBOR. In the next chapter a quantitative analysis is provided for these methods. An encoding and decoding application for attestation result is implemented for each of the serialization methods and the performance of those applications is measured and benchmarked.

3.1 Selection Criteria

The goal of the presented selection criteria is to highlight some of the features necessary for serializing attestation results. Overall, the chosen formats should be secure, usable, maintainable and efficient.

Efficiency: The attestation result format should be suitable for resource-constrained devices. Performance criteria such as encoding and decoding speed, code footprint, and output size are considered. However, there is no academic literature or online analysis that compares between all the formats discussed in section 2.7 in terms of the speed, code footprint and message size. There are some papers and blogs that compare between JSON and XML [64], JSON and CBOR [55], or JSON and Protocol Buffers [81]. Those measurements do not provide a foundation where we can directly compare between CBOR and Protocol Buffers, for example. Those kinds of metrics that requires comparisons to be done in a similar environment and with the same input are going to be evaluated in chapter 4. It is not feasible to provide an implementation for each of the considered formats especially those that are not suitable for encoding attestation results. The selection criteria presented in this chapter are used as a kind of filtration that allows the benchmarking of only the formats suitable for remote attestation in general and specifically with constrained relying parties.

Security: Security of the encoding of attestation results is important because the entire goal of attestation is to validate the trustworthiness of a device. Some formats, like ProtoBuf and FlatBuffer, while implemented to grant flexibility can also result in attacks on integrity and confidentiality. **Canonicality** or determinism of the encoding is an important criterion to consider when selecting the formats. Canonicality is not the only property that influences the security of the format, but it was chosen due to some attacks, discussed in section 3.1.1, that exploited its absence.

Usability: Remote attestation typically involves different platforms. It is important that the encoding and decoding of the format can be easily implemented across multiple platforms. The usability feature of the formats is divided into two criteria: **programming language independence** and **versatility**.

Maintainability: Maintainability of a serialization format means the ability of the format to be updated and supported over time. The format used for attestation result should not be deprecated or not widely used since this affects the likelihood of it being supported on different platforms. **Standardization** is a good measure for such a feature since standardized formats are guaranteed to have a dedicated community that maintain and provide timely updates that reflect the changes and improvements of the format.

3.1.1 Canonicality

Canonicalization or normalization is the process of converting some data into a standard or canonical form. OWASP provides an example of the kind of attacks that can happen because of the non-canonicality of a URL encoding. For example, the opening and closing tags $<, >$ can be represented as %3c and %3e respectively. Hence, an attacker can exploit this non-canonical representation and embed some script in the URL without using the explicit opening and closing tags [59].

Canonical serialization produces a unique bit-string of a data structure. Some sources use the term “deterministic serialization” synonymously with canonical serialization [17, 20]. Numerous serialization formats are non-canonical, such that a single data structure can be serialized to different bit-string values. This section describes real-world attacks that exploited this feature.

Non-canonicality is sometimes an intended feature in serialization formats especially with those designed for complex and large amounts of data. It is able to improve performance by reducing the time and resources required for the serialization and deserialization code to check and guarantee canonicality [14].

A non-canonical serialization scheme can lead to difficulty in validating the authenticity of the data. For example, an attacker may perform an injection attack by adding some malicious script or code as part of the serialized bit-string. The user, not knowing the expected size of the data, would deserialize the entire object and unintentionally parse the script. Non-canonical serializations maybe susceptible to Denial of Service (DoS) and data integrity attacks. Two real-life examples of attacks on non-canonical formats are:

1. ProtoBuf allows decoding bit-strings with unknown fields. So, two serialized bit-strings, one with unknown fields and one without, can map to the same data object. The ProtoBuf Java implementation suffered a DoS attack where input streams that contain unknown fields causes frequent pauses during parsing. Therefore, a malicious input containing unknown fields can occupy a parser for minutes at a time [22] .

2. The RSA-sign JavaScript Library encountered a vulnerability that accepted wrong ECDSA digital signatures that were semantically the same as valid ones. It accepted different representations of the same number. The reason was that there was no strict ASN.1 DER checking. For example, a signature that contained an ASN.1 INTEGER with prepended zeroes such as $0x00000123$ was accepted if the valid signature contained the integer $0x0123$ [21].

These kinds attacks can be mitigated by implementing appropriate safeguards and using secure channels for transmission. This protection is not due to the nature of the serialization, but to the safeguards implemented which can differ from one programming language or library to another. Therefore, formats that are non-canonical are excluded from the present study as they can increase the risk for exploitation when transmitting attestation results. ASN.1 and CBOR have a canonical and non-canonical representation [17, 52], but only the canonical subsets will be considered.

3.1.2 Programming Language Independence

Programming language independence of the format provides flexibility in encoding and decoding attestation results. The encoding and decoding can be written in different programming languages that are the most suitable for the platforms running the relying party and verifier. For example, in an attestation application that uses Veraison for verification, the result encoding is likely implemented using the Go programming language[1]. However, in case of a constrained relying party with a simple microcontroller and a C compiler, a decoder written in C would be a better choice. Therefore, it is essential that the format is language-agnostic.

Serialization formats like Java Object Serialization (JOS) and Pickle, while being strong and detailed enough to serialize complex objects like maps, are dependent on their corresponding languages Java and Python. For example, a Pickle object can only be loaded by Python [78]. These formats are excluded from the present study because they would only limit the implementations to Java and Python-based applications. They might be of interest when interoperability between other languages and devices is not a requirement.

3.1.3 Standardization

There is currently no standard format that is developed specifically for attestation artifacts like evidence or attestation results. Standardization guarantees that the format will be maintained and updated to match the changes of its environment or a new standard would be created to replace the obsolete ones. In either case, there would be a guarantee that the format is continuously improved and maintained. In addition, with standardized formats, serialization implementations becomes simple, reliable, and vendor-independent as the rules and description is already specified for the concerned parties to follow.

3.1.4 Format Versatility

Format versatility refers to how easy it is to use the format in other use cases than the one it was originally intended for. This property is considered because some formats are designed to work in a specific context and might not be easily adapted in other domains. The RATS RFC [12] suggests the possibility of retrofitting deployed protocols with remote attestation by adding

[1]The Veraison project (see section 2.4) uses Go because it is an efficient and memory managed language [9].

attestation messages as extensions. According to RATS, the motivation behind this suggestion is to minimize the amount of parsing coded needed, especially in a constrained relying party, by using an already supported format. However, such a feature needs to be carefully studied and investigated. Even if a format is already supported by a relying party, it does not mean that the format is the most secure or efficient to use for attestation.

The RATS RFC gives an example of incorporating attestation capabilities in a TLS handshake by embedding attestation evidence and results in an ad hoc X.509 certificate extension. However, this study is concerned with devices that cannot perform public key cryptography and consequently do not support certificates. The X.509 extension could still be used to encode attestation result, but in this case the certificate might not adhere to the standard specifications which could lead to vulnerabilities [65]. Therefore, X.509 might not be suitable for encoding attestation results for constrained relying parties where the object is just added in the extensions, leaving almost all other parts of the certificate fields empty.

FlatBuffers is another format that might not be suitable for attestation results. FlatBuffers is designed for performance critical applications. It is intended for accessing only one or two entities in the serialized object as opposed to reading entire objects. The code required for deserializing the entire object is much larger than the one for JSON [49]. Hence, such a serialization is not suitable for attestation results where the entire object needs to be read. The chosen format should be able to work without any assumptions about the environment, so formats like X.509 and FlatBuffers that are designed for specific use cases are not considered.

3.2 Comparison Results

Table 3.1 compares the formats under investigation and determine which formats satisfy the canonicality, language independence, standardization and versatility properties. The formats that are marked in yellow satisfy all those properties and are chosen for the quantitative analysis done in chapter 4.

Formats like Java Object Serialization and Pickle, while powerful and able to serialize complex data structures, are language dependent. Consequently, they are not suitable for encoding attestation results where the encoding can be done in a language and the decoding in another.

The ProtoBuf and FlatBuffer are language-independent formats that are designed to handle large amounts of data for efficient storage and transmission. However, FlatBuffers is designed for scenarios where a small part of the data needs to be read and not the entire data object. In addition, both Pro-

Format	**Serialization**	**Canonical**	**Language independent**	**Standardized**	**Versatility**
ASN.1 DER	Binary	✓	✓	✓	✓
CBOR	Binary	Deterministic CBOR	✓	✓	✓
FlatBuffer	Binary	✗	✓	✗	✗
JOS	Binary	✗	✗	N/A	✗
JSON	Text-based	JCS	✓	✓	✓
Pickle	Binary	✗	✗	N/A	✗
ProtoBuf	Binary	✗	✓	✓	✓
XML	Text-based	Canonical XML	✓	✓	✓
X.509	N/A	N/A	✓	✓	✗

Table 3.1: Properties of serialization formats. Notation: ✓– the format has the property; ✗– the format does not have the property; N/A – not applicable.

toBuf and FlatBuffer, and while non-canonicality can provide flexibility and ease of use, it can lead to various attacks on the availability and integrity of the data. Therefore, only canonical encoding is considered for the attestation result encoding format.

The X.509 standard is a well-known and widely implemented scheme that is used for endpoint authentication. Using X.509 to communicate the attestation result object means that TLS should be used otherwise many fields in the certificate will be empty creating a parsing overhead. A constrained relying party in an attestation procedure might not be able to use public key cryptography and in turn TLS. Hence, the considered formats should not depend on the environment and limit any assumptions made about the communication.

The two text-based formats considered in this study are XML and JSON. One goal is to evaluate how binary serialization performs in comparison to text-based formats when encoding attestation results. JSON and XML are non-canonical by nature but there are schemes and rules to make them canonical. For example, the JSON Canonicalization Scheme (JCS) [73] defines rules to create a canonical representation of JSON. The XML format was excluded from further consideration because the average time for transmitting and decoding the objects can be much slower than that of JSON [64]. Therefore, the JSON format will be the basis for comparing the different chosen binary formats.

CBOR is interoperable between different devices and does not depend on any specific programming language. Its extensibility allows older decoders to decode messages even if there are new extensions or features. Due to these properties, CBOR is suitable for network communication. In general CBOR can allow non-canonical serializations, but the specification sets some rules and restrictions for creating Deterministically Encoded CBOR. The Deterministically Encoded CBOR is the only CBOR format that will be considered and any further mention of CBOR refers to the serialization that follows the deterministic set of rules unless explicitly stated otherwise.

The ASN.1 format is language, vendor, and platform independent. Every device that uses HTTPS is able to decode ASN.1-encoded X.509 certificates [75]. ASN.1 has several encoding rules, from which here only the Distinguished Encoding Rules (DER) are considered, because they provide canonical serializations. Another advantage of ASN.1 DER is that there are various libraries that have undergone extensive security checks. This is due to the fact that ASN.1 has been around for decades and it is used in security critical applications like encoding certificates. One final point to highlight is that those three chosen formats (ASN.1 DER, CBOR, JSON) are all standardized and maintained by an organization or community.

3.3 Related Work

Some of the work done in the literature was investigated to determine whether any research has compared between ASN.1, CBOR and JSON and whether those comparisons are in terms of encoding size, encoding speed, decoding speed, and memory usage. Google Scholar was used to search for relevant work. The search phrases "survey of encoding formats", "survey of serialization formats", and "formats for IoT devices".

The papers surveyed provide examples on how the benchmarks were implemented and help motivate the implementations and benchmarks done in chapter 4. Table 3.2 contains the relevant formats each paper has surveyed, the comparison metrics used in the survey, and the key discoveries or results found by each survey. It is important to note that the comparisons and measurements are based on library implementations of the formats, but the chosen libraries are designed to be efficient.

Table 3.2: Summary table of literature that compare serialization formats

Title	**Relevant Formats**	**Used Metrics or Use cases**	**Key Results and Observations**
Towards Lightweight and Interoperable Trust Models: The Entity Attestation Token [65]	CBOR, JSON, X.509	Size of encoded data, ease of implementation, size of implementation	- A CBOR based implementation requires substantially fewer lines of code than X.509 - CBOR based encoding size is smaller than JSON based - X.509 has a poor record for exploitable software vulnerabilities
A Survey of JSON-compatible Binary Serialization Specifications [83]	ASN.1, CBOR, FlatBuffers, ProtoBuf	Space-efficiency, runtime-efficient deserialization, partial reads, streaming deserialization, streaming serialization, in-place updates, constrained devices	- ASN.1 Packed Encoding Rules (PER^2) is suitable for space-efficiency as serializations tend to have minimal metadata - CBOR is suitable for constrained devices due to small code size
A Comparison of Data Serialization Formats for Optimal Efficiency on a Mobile Platform [81]	XML, JSON, ProtoBuf	Size of encoded data, serialization speed, ease of use.	- JSON is a superior alternative to XML - JSON should be used when dealing with existing web services - Binary serialization is more suitable for storage purposes

Impacts of Data Interchange Formats on Energy Consumption and Performance in Smartphones [29]	XML, JSON, ProtoBuf	Size of encoded data, average time for querying and compressing data, average time of parsing, energy expended in CPU, WiFi, 3G, Average time for data synchronization between mobile and server	- JSON with compression had the best time synchronization (time compressing in mobile, sending to server and decompressing in server) and better battery management. - ProtoBuf are not as inefficient as JSON when there is data compression, but performs the best when no compression is carried out
Web Service Composition on Smartphones: The Challenges and a Survey of Solutions [57]	XML, JSON, ProtoBuf	Complexity, performance, flexibility, footprint, reusability, usability, scalability	- Binary serializations are more complex as they need tools to be parsed - ProtoBuf has the smallest footprint and is faster than the other surveyed format - Text-based formats are more flexible and usable than binary

Performance Comparison of Messaging Protocols and Serialization Formats for Digital Twins in IoV [70]	JSON, ProtoBuf, FlatBuffers	Size of encoded data, serialization speed, deserialization speed, memory usage	- ProtoBuf had the smallest message size and serialization speed - FlatBuffers had the fastest deserialization speed and lowest memory consumption
Smart Grid Serialization Comparison [67]	XML, JSON, ProtoBuf, CBOR	Serialization time, deserialization time, compression time, decompression time, memory use for serialization, memory use for compression, serialized message size, compressed message size	- Compressing text-based formats defeats purpose of having them since they become unreadable - Serialization libraries have a large impact on the performance of the format - JSON is a better alternative to XML - Binary serializations like ProtoBuf are faster and more compact

The general observation is that binary serializations tend to be more efficient (less memory consumption, faster, smaller message size) but text-based serializations are more usable and flexible. None of the found studies address all the desired formats. Some research was more focused on quantitative measures like serialization speed and output size while others focused more on qualitative measures like ease of implementation, usability and scalability. Furthermore, most studies did not specifically consider constrained devices or have all the desired comparison metrics like message size, encoding speed, decoding speed, memory consumption. The authors in [83] include constrained device as a use case where they define a suitable format as that having a simple specification, simple binary layout and small generated code. One observation the authors in [83] made is that the implementation rather than the serialization specification tends to be the contributing factor when determining whether a binary serialization is suitable for constrained devices.

Table 3.3 shows the difference between the relevant studies found in the literature and the work done in this thesis. None of the found studies compare between ASN.1, CBOR, and JSON in terms of message size (MS), encoding speed (ES), decoding speed (DS), encoding memory consumption (EMC) and decoding memory consumption (DMC). Almost all studies use JSON in their comparison. This can be attributed to the fact that it has become the most widely used format for exchanging data over the Internet. However, JSON is neither run-time efficient nor space efficient [83].

There were no comprehensive quantitative comparison of different formats in the context of attestation. The comparison in [65] only used the encoding size and the lines of code (LOC) for comparing the encodings of attestation tokens (evidence and results). Chapter 4 fills the gap in the literature by providing a quantitative comparison between ASN.1, CBOR, and JSON in the attestation result domain.

Work	**Atte-station**	**Relevant Format**			**Metrics**				
		ASN.1	CBOR	JSON	MS	ES	DS	EMC	DMC
[65]	✓	X.509	✓	✓	✓				
[83]		✓	✓		✓				✓
[81]				✓	✓	✓			
[29]				✓	✓	✓	✓		
[57]				✓					
[70]				✓	✓	✓	✓	✓	✓
[67]			✓	✓	✓	✓	✓	✓	
this work	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 3.3: Comparison between the literature and this work.

[2]ASN.1 PER is a canonical non-TLV-based set of rules that are designed to produce smaller encoding size [52].

Chapter 4

Implementation and Benchmarking

The constrained environment assumed in the relying party influences the kind of applications it would be able to run for attestation. The relying party needs to parse the attestation results it receives in order to issue a verdict of whether to trust the attester. The attestation results it receives needs to be minimal in size as the relying party might have small packet buffers or restricted transmission bandwidth. Hence, an evaluation of the encodings chosen in section 3.2 in terms of size and performance is needed to determine which encoding is suitable for constrained relying parties.

This chapter will provide details on the implementation of the encoding and parsing of attestation results in ASN.1 DER, CBOR and JSON. It will also provide a comparison between the performance of the different formats in encoding and decoding attestation results in EAR format (see section 2.5). Finally, the chapter will provide an analysis of the performance results and the choice of encoding format.

4.1 Comparison Metrics

The metrics chosen for comparing the encoding formats relate to the performance of the respective applications. The comparisons determine how efficient the encoding is in terms of message size, encoding speed, decoding speed, encoding memory consumption and decoding memory consumption. In the relying party application, encoding speed is not a relevant metric as the relying party does not perform the encoding. This metric was included to analyze whether the format with the highest decoding speed will also have the highest encoding speed or not. The definition of the metrics are as follows:

- Encoding size: the size of the attestation result object when encoded to a specific format and written to file.
- Encoding speed: the time it takes to convert an attestation result object that has been created by the verifier into a serialized object.
- Decoding speed: the time it takes to parse a serialized object into an attestation result object that can be processed and used in the application.
- Encoding memory consumption: the size of the code loaded into memory when encoding the attestation result object and writing it to file.
- Decoding memory consumption: The size of the code loaded into memory when reading an attestation result object from file and parsing it.

4.2 Benchmarking Inputs

The same attestation results objects are used to benchmark all implementations in order to accurately measure the performance of the three formats. The baseline attestation results object (Input 1) shown below in JSON format includes all details about the attestation procedure based on the EAR format [8]. The object contains the profile (version) of the attestation result used, the verifier identity, issued at time, and raw evidence which is what the attester sends to the verifier for appraisal. In addition, the result includes the verifier's verdict for the attester 'PSA', the attributes it has appraised, as well as the policy it used for appraisal. More importantly, this object contains all the data types that would need encoding like string, integer, long, and maps. Section 2.5 contains more details about the information and values present in EAR objects.

```
{
    "eat_profile": "tag:github.com,2023:veraison/ear",
    "iat": 1666529300,
    "ear.verifier-id": {
        "developer": "https://veraison-project.org",
        "build": "vts 0.0.1"
    },
    "ear.raw-evidence": "3q2-7w",
    "submods": {
        "CCA Platform": {
            "ear.status": "affirming",
            "ear.trustworthiness-vector": {
                "instance-identity": 2,
                "configuration": 2,
                "executables": 3,
                "file-system": 2,
                "hardware": 2,
                "runtime-opaque": 2,
                "storage-opaque": 2,
                "sourced-data": 2
            },
            "ear.appraisal-policy-id":
                "https://veraison.example/policy/1/60a0068d"
        }
    }
}
```

The applications are also measured on two other inputs:

- Input 2: Consists of all information found in Input 1 with an additional attester object, i.e. adding another map with another status, trustworthiness vector, and policy id.
- Input 3: Consists of all information found in Input 1 with a large raw evidence value that is attached to the baseline input. The raw evidence is a large base64 encoded string that represents binary data.

The JSON form of Input 2 and Input 3 can be found in appendix A. Those inputs were chosen in order to study how the behavior of each format changes depending on the data types it is encoding.

4.3 Implementation of Applications

In order to accommodate for a constrained relying party, the parsing code was done in the C programming language. During the time of this project, there were no available applications that encode and decode the attestation result object for the three formats. The Veraison project (section 2.4) provides a minimalist C implementation of the decoding in JSON [25]. In the Veraison implementation, some of the attributes were not decoded and the functionality was specific to the Veraison verifier. Therefore, applications that parse the attestation results objects in the three formats where developed in order to provide accurate comparisons between the formats. The code can be found in appendix B. The remainder of this section discusses the structure of the applications, the environment these applications were written in, and the differences encountered when implementing the encoding and decoding in each format.

4.3.1 Environment

The tests are run in an environment with the following properties:

- Device: Lenovo ThinkPad T14s Gen 2i laptop
- RAM: 16 GB
- Processor: 11th Gen Intel(R) Core™ I7-1185G7 @ 3.00GHz
- Architecture: x86_64
- Operating system: Ubuntu 20.04.6 LTS (Focal Fossa)

The code was written in C in order to create applications that are similar to the ones found on small IoT devices. Most IoT operating systems like Zephyr, Embedded Linux, Contiki and Raspberry Pi allow running C applications. In addition, the C programming language is low level and small in size [48] which allows writing programs with small memory footprint and fast run-time. Therefore, it is the most suitable for small or resource constrained IoT devices.

4.3.2 Application Structure

The three applications for ASN.1 DER, CBOR, and JSON all implement the same APIs. There are two applications for each format, one for encoding and the other for decoding. The main reason for this split is that the encoding and decoding operations do not happen on the same device. All the libraries chosen in the implementation are designed to have low memory footprint and are developed to work in constrained embedded devices.

The goal of the decoding applications is to parse the encoded attestation results object into a C-struct (see appendix B). The struct can then be used to perform any operation on the attestation results. The C-struct has a similar structure in all implementations with some minor modifications. The status in the appraisal is represented as an integer in ASN.1 DER and CBOR implementations and as a string in JSON. This was done in conformance to the EAR RFC draft's trust tier definitions for CBOR and JSON [26]. The ASN.1 implementation also had a corresponding integer attribute for specifying the length of each string found in the object. Those attributes were needed to parse the string from the encoded TLV.

Since subsequent operations are not in the scope of the performance measurements, the decoding applications simply print the values in the struct. All the decoding applications are structured as follows:

- Input: All applications have a `read_file` API that reads the contents of a file into a byte array. The file has extension .der, .cbor, or .json according to the application. The byte array is simply the contents of the file.

- Parsing: The API `parse_ear` is used to decode the byte array that was read from file. This method differs completely from one application to the other and thus has the most impact on performance measurements. The method results in the same parsed attestation result struct in all implementations.

- Output: In an actual attestation application the attestation result struct would be used to perform checks and appraisals. However, the implemented applications are only concerned with the performance of the different encodings. Therefore, the `print_ear` API, as the name suggests, simply prints the contents of the attestation results struct. The printed format and information are exactly the same in all applications.

The encoding applications perform the reverse functionality where the attestation result C-struct is encoded and then written to a file with the respective extension .der, .cbor, or .json. The encoding performance is not relevant for a relying party but it is also measured for the sake of providing a comprehensive performance report

4.3.3 ASN.1 DER

As mentioned in section 2.6.2, ASN.1 is a schema driven binary serialization format. Both a schema and a set of encoding rules are required in order to develop an ASN.1 representation of data. The schema defines the structure of the data, and the encoding rules translate the schema into bytes. Each element defined in the schema is represented as a tag-length-value (TLV). Where for each defined attribute in the schema there is the tag representing the attribute's data type, the length specifying the number of bytes used to represent the value, and the value of the information in bytes. Generally, the value can in itself be another TLV. The different binary TLV-based encoding rules provided by ASN.1 determine how those TLVs are created. Due to the reasons provided in section 3.1.1 on the importance of canonicality, only the Distinguished Encoding Rules (DER) will be considered. DER eliminates any variations in encoding by specifying a strict order for elements, using the same rules to encode the tag and length for all data types, and eliminating any variations in encoding that do not affect the value of the data. For example, DER enforces a shortest possible form representation of integers where any leading zeroes are removed.

The encoding and decoding of the attestation result object in ASN.1 was done using the ASN.1 module offered by Huawei's embedded TLS library HTLS. HTLS provides a small, stand-alone, and efficient implementation of TLS 1.3 and related cryptography and remote attestation. The library performs no memory allocation and only requires an entropy source to work. Therefore, it can run in constrained devices such as the relying party considered in this thesis. In addition, only the ASN.1 parser is used and there is no additional overhead from the other functionalities provided by the library.

The ASN.1 schema for an attestation result object is listed below. It is specified according to the Concise Data Definition Language (CDDL) [1] provided in [8].

```
World-Schema DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
EAR ::= SEQUENCE {
    eat-profile UTF8String ("tag:github.com,2023:veraison/ear"),
    iat INTEGER,
    ear-verifierID  AR4si-verifierID,
    ear-rawEvidence UTF8String (PATTERN "[A-Za-z0-9_=-]+") OPTIONAL,
    eat-submods SEQUENCE SIZE(1..100) OF SEQUENCE {
        attesterID UTF8String,
        ear-appraisal  EAR-APPRAISAL
    },
    eat-nonce UTF8String (SIZE (10..74)) OPTIONAL
}

AR4si-verifierID ::= SEQUENCE {
  build UTF8String,
  developer UTF8String

}

EAR-APPRAISAL ::= SEQUENCE {
ear-status ENUMERATED {none(0),
                       affirming(2),
                       warning(32),
                       contraindicated(96)},
ear-trustworthinessVector AR4si-trustworthinessVector OPTIONAL,
ear-appraisalPolicyID UTF8String OPTIONAL

}

AR4si-trustworthinessVector ::= SEQUENCE  {
   instance-identity [0] IMPLICIT AR4si-trustworthinessClaim OPTIONAL,
   configuration [1] IMPLICIT AR4si-trustworthinessClaim OPTIONAL,
   executables [2] IMPLICIT AR4si-trustworthinessClaim OPTIONAL,
   file-system [3] IMPLICIT AR4si-trustworthinessClaim OPTIONAL,
   hardware [4] IMPLICIT AR4si-trustworthinessClaim OPTIONAL,
   runtime-opaque [5] IMPLICIT AR4si-trustworthinessClaim OPTIONAL,
   storage-opaque [6] IMPLICIT AR4si-trustworthinessClaim OPTIONAL,
```

[1]CDDL is a notation convention used to express data structures encoded in CBOR and/or JSON [13].

```
    sourced-data [7] IMPLICIT AR4si-trustworthinessClaim OPTIONAL
}

AR4si-trustworthinessClaim ::= INTEGER (-128..127)
END
```

The ASN.1 schema definition enables the specification of the various rules and patterns in an in-depth manner. For example, it specifies that `eat-profile` can only be the value `tag:github.com,2023:veraison/ear`. In addition, it can set rules on string patterns, size of attributes, and whether the attributes are optional or not. Since ASN.1 is based on tag-length-value encoding, implicit tags are specified for the `AR4si-trustworthinessVector`. The reason behind this is that if it were not for those implicit tags, the decoder might not be able to know which claim it is parsing since they are all integers, i.e. have the same tag. Therefore, implicit tagging is used to indicate the claim. The schema also defines `ear-status` as an enum that can only be one of the specified options. These specifications allow the developer of the encoder to know exactly how to encode the object into a TLV and allow the decoder to know exactly the structure and possible values to expect.

The schema does not conform entirely to the CDDL definition of attestation results defined in the EAR RFC draft [8]. The specification defines the `eat-submods` as an object `eat.submods => { + text => EAR-appraisal }`, however such a definition of a sequence with an arbitrary key name is not possible in ASN.1. Therefore, the submods definition was adjusted for ASN.1 by defining submods as a sequence of a pair of attributes `attesterID` and `ear-appraisal`. The `+text` part of the CDDL definition is substituted by the attribute `attesterID`.

HTLS provides the functions for encoding and decoding the basic data types specified in ASN.1 DER like integer, boolean, string, and sequence. The code for encoding and decoding the attestation results utilized both those basic functions and the schema listed above. There were a few extra operations that needed to be done when parsing integers and strings. A helper method was created to parse the integers from a byte array to an int value. As for parsing strings, the TLV object returned by HTLS performs no memory allocation for the sake of efficiency and returns a pointer to the desired attribute. Therefore, in order to know where the string attributes end, an additional attribute is added in the C-struct for the length of the string. Then during printing the string, the length attribute is used to determine where the string ends.

4.3.4 CBOR

Since CBOR is schema-less, the information about the encoded value including the type and length needs to be added in the serialization. As mentioned in section 2.7, the initial byte of each data item includes information about the data type and any additional information required for encoding. This additional information can either be the integer value (in case of small numbers), the length of the data item, or an indication that the data item is of indefinite length. When encoding indefinite-length items, like arrays, maps, or strings, a special 'break' character is used to signal when the decoder should stop. By default, CBOR is not canonical, but the standard specifies rules for creating canonical CBOR encodings. These rules include representing signed and unsigned integers and floats in the smallest possible number of bytes, sorting keys in the map from lowest to highest values based on the byte representation of the key, and converting indefinite-length items into definite-length items.

Encoding and decoding attestation results in CBOR was done by Intel's industrial strength implementation called TinyCBOR. According to the documentation [43], the library is optimized for very fast operation and small code footprint. In addition, the main encoder and decoder does not use memory allocations. TinyCBOR has been compiled and used in many IoT frameworks like Arduino and RIOT OS.

CBOR is known for the efficient and small encoding of integers. Therefore, map keys are generally transformed into integers as opposed to strings like in JSON. For example, in the JSON object provided in section 4.2, the key `eat_profile` is mapped to the integer `265` in the equivalent CBOR encoding of the same object. Similarly, all other known attributes also have an integer value. One point to note is that there is development overhead when an application supports both JSON and CBOR as any changes to the attestation results object requires a modification to the CBOR serialization mapping. In addition, applications must adhere to the same integer mapping. The keys mapping implemented in the CBOR attestation results encoding and decoding follows the EAR RFC draft [8].

The most notable function used when decoding the CBOR attestation results byte array is `cbor_value_advance` which runs in $O(n)$ time when it is parsing a container like a map or array as it will recurse into itself. In addition, the function will use $O(n)$ memory for the number of nested containers. Here `n` represents the number of elements it needs to parse. Therefore, this function has a high impact on the performance of the CBOR encoding. One other note worth mentioning in the implementation is that the function used to parse the string values performs memory allocation in order to copy the value to the corresponding attribute in the EAR C-struct. In addition, like ASN.1, the status of each submodule is encoded as an integer instead of a string.

4.3.5 JSON

The Jansson library [53] was used for encoding and decoding the JSON attestation results object. The library is not dependent on other libraries and is suitable for use on any system including small embedded devices. The library only supports UTF-8 encoded texts. Furthermore, it contains two definitions of numbers "real" and "integer". In C, all numbers in a JSON object with type "real " are parsed into a "double" and "integer" numbers are parsed into "json_int_t" which is equivalent to "long" or "long long" depending on the C compiler. Hence, there is no efficient encoding for unsigned types and smaller numbers like "int" or "short". Generally, JSON is more concerned with text encoding and has no optimization when encoding other types.

The implementation of the attestation results JSON parser utilizes the method `json_object_get` which given the JSON object and key of type string, returns a pointer to an object of type `json_t` corresponding to the specified key. The type of this object is checked and the value of the attestation results attribute is parsed. In order to parse string attributes, the C function `strdup` is used to copy the value found in the JSON object to the corresponding attribute in the attestation results C-struct. In addition, unlike ASN.1 and CBOR the status in the JSON object is encoded as a string rather than an int. All other values are parsed as expected. String attributes like profile and appraisal policy are parsed into 'char' arrays and integer values like issued at time and the attributes in the trustworthiness vector are parsed into 'int'.

4.4 Benchmarking

The three implementations were tested on the same inputs discussed in section 4.2. The comparison is split into three categories relating to output size, speed (encoding and decoding), and memory consumption (code, heap, and stack). The profiling tool Valgrind is used to analyze the implementations and determine which part of the implementation used the most memory or was slowest in terms of decoding.

4.4.1 Object Size

The implemented encoders were used to encode the inputs (input 1, input 2, input 3) discussed in section A in the corresponding formats. It is important to measure the object size because this is the message the relying party receives from the verifier. Hence, it affects the bandwidth, the receiving buffer in the device, and maybe even the file system if the attestation results are to be stored. Table 4.1 compares the attestation result encoding size for each of the three formats. The JSON object has the biggest size in comparison to the other formats as it is text-based. One interesting observation is that, unlike the other formats, it performs better when encoding a large raw evidence value than when encoding an additional attester. The reason behind this is that it encodes each character in the JSON object hence adding an additional map in input 2 increased the size of the stored object. ASN.1 and CBOR have better and more efficient encodings for input 2 as it does not have to store the additional map as is and can optimize the integer encoding for the trustworthiness vector. The size difference between ASN.1 and CBOR is not substantial as they differ in a few number of bytes. CBOR performs better when encoding an additional trustworthiness vector because it requires 2 bytes (key, value) for each claim while ASN.1 requires 3 bytes (tag, length, value). It is important to note that CBOR has additional bytes used to include the key and other attributes like data length as it is an unstructured binary serialization.

Input	**Encoding size [bytes]**		
	ASN.1 DER	**CBOR**	**JSON**
Input 1: baseline	203	209	753
Input 2: two attesters	309	291	1,132
Input 3: raw evidence	443	448	991

Table 4.1: Encoding size for attestation results.

Binary serialization techniques do not perform any modifications to the encoding of strings. Therefore, adding strings or larger strings leads to increasing the size of the objects by the corresponding string size regardless of the format in use. In fact, binary serialization might actually need more bytes to encode the length of the string. Therefore, the difference between input 1 and input 3 encoding for the JSON object is just the size of the raw evidence. Whereas in CBOR, there is an additional byte for representing the length of the raw evidence. Furthermore, the bigger the string, the more bytes ASN.1 and CBOR would need to encode the string length.

An additional point to note about Input 3 is that it contains a large base64-encoded string. Binary data is usually represented in base64 in JSON, as well as other text-based formats. Base64 encoded data tend to increase the output size by at least 33% of the size of the original data [54]. This raw evidence string can, in fact, be represented in ASN.1 DER and CBOR as an array of bytes, which would lead to smaller encoding size and possibly faster execution time. The raw evidence field is represented as string in all formats to measure how the behavior differs when encoding strings in binary and text-based serializations. Measuring raw evidence as a string led to the observation that an additional benchmarking input, with a large binary array, can be insightful when comparing how different serializations handle different data types.

4.4.2 Processing Speed

In order to guarantee the accuracy of the application measurements, all the applications are compiled with the same flags designed for optimizing code speed. The optimization flag "-O3" generally optimizes for code speed at the expense of code size by, for example, unrolling loop instructions. In addition, as mentioned in section 4.3.1, all applications are run in the same environment on the same device. Table 4.2 displays the results of the decoding speed of the three inputs under investigation. The decoding speed is important because it is part of the code that will be running in relying parties. The encoding speed results are shown in table 4.3 for the sake of providing a complete comparison. The tables state the time for 100,000 decoding and encoding operations respectively. The JSON implementation for decoding is the fastest

Input	**Decoding time [ms]**		
	ASN.1 DER	**CBOR**	**JSON**
Input 1: baseline	50	203	44
Input 2: two attesters	82	305	67
Input 3: raw evidence	50	213	48

Table 4.2: Decoding time for 100,000 iterations in milliseconds

and ASN.1 is closely behind. ASN.1 decoding time did not get affected by the large raw evidence input. Valgrind [60] was used to study the large time difference between CBOR and the other formats. Due to the nested container structure of attestation results, the method `cbor_value_advance` has to recurse and thus it utilizes half of the time of the decoding. The other method used in the implementation is `cbor_value_dup_string` which copies the string to the desired attribute using memory allocation. While memory allocation does take 17% of the entire time, the remaining processing time is spent to advance the CBOR object pointer. Hence, the bottleneck of the CBOR implementation comes from the recursive function used to iterate over the CBOR object and the time it takes to pre-parse the next value. The function `json_object_get` utilized the most time in the JSON application. In addition, the function `htls_buf_read_asn1_der_tlv_check_tag` which parses the TLV encodings of ASN.1 values utilized the most time in the ASN.1 DER implementation. This behavior is expected of the implementations.

Input	**Encoding time [ms]**		
	ASN.1 DER	**CBOR**	**JSON**
Input 1: baseline	65	67	116
Input 2: two attesters	102	106	195
Input 3: raw evidence	100	70	143

Table 4.3: Encoding time for 100,000 iterations in milliseconds

The encoding time for JSON is the longest of the three formats. ASN.1 performed the best in the baseline input and the one with two attesters. It was slower in encoding input 3 that contains a large string. Overall, CBOR has the best performance when encoding attestation result objects as its performance is almost constant regardless of the size of the strings. In addition, it performs almost as fast as ASN.1 in input 2 with two attesters. However, as mentioned earlier, constrained relying parties only need to decode the attestation results.

4.4.3 Memory Consumption

There are several factors to consider when measuring the memory consumption of an application. Each application has its own memory layout that contains segments like text, heap, and stack [19]. The functionality of each segment is as follows:

1. Text: stores the application code that is being executed.
2. Heap: The dynamic memory that the application allocates during execution.
3. Stack: The local variables being used in the program.

In order to measure each parameter, the "size" command is used to determine the text code size of the applications. The optimization flag "-Os" is used during compiling the applications in order to optimize the applications' code size. The heap and stack are measured using the Massif tool in Valgrind. Massif is a heap profiler that measures how much heap space a program utilizes. In addition, it can also measure the size of a program's stack. Since this tool requires the program to execute, the encoding and decoding applications use the baseline input (input 1) for measurements. Furthermore, the encoding and decoding functionality for each format is separated to accurately simulate the application size that would be in the relying party. One important note to mention is that, aside from the encoding and decoding code, all other methods are implemented in the same manner and perform the exact functionality. Therefore, while the memory consumption of the entire applications is measured, the changing factor belongs to the encoding and decoding of the formats. Table 4.4 shows the code, heap, and stack size of each of the encoding applications. While table 4.5 shows the same metrics for the decoding application which is more relevant to the constrained relying party use case.

Metrics	**Encoding memory consumption [bytes]**		
	ASN.1 DER	**CBOR**	**JSON**
Text	10008	4397	4319
Heap	5624	5624	8392
Stack	10032	8440	10264

Table 4.4: Memory allocation of encoding application.

Metrics	**Decoding memory consumption [bytes]**		
	ASN.1 DER	**CBOR**	**JSON**
Text	12003	7612	7762
Heap	5840	4824	5368
Stack	9808	7832	8904

Table 4.5: Memory allocation of decoding application.

The recorded heap and stack values are the maximum value each parameter reached during program execution. CBOR has the lowest memory footprint when it comes to encoding and decoding, while ASN.1 has the highest footprint. Overall there is no major difference in memory allocation between the different implementations as they differ only in a few kilobytes. In addition, the core functionality of encoding and decoding is implemented in all the formats. After decoding, the actual application in the relying party would perform additional operations that do not depend on the encoding format, like verifying the status or trustworthiness claims.

4.5 Analysis

The implementations for the three formats were compared in terms of encoding size, encoding speed, decoding speed, and various memory allocation parameters. Table 4.6 shows the formats' performance in each of the metrics. The encoding size and speed measurements are the average over the three inputs. While the memory consumption measurements are the sum of the text and heap memory. The reason text and heap are chosen (and not stack) is that the text is the actual assembly code of the program loaded in memory and the heap is the memory that was dynamically allocated during program execution. On the other hand, the stack is usually associated with static memory allocation because the stack size limit is assigned by the operating system and if this limit is reached the program terminates. Therefore, code size and heap are a more accurate representation of the memory needed to run the encoders and decoders.

It is important to note that the table compares the average performances and that the behavior of a format differed depending on the type of input being encoded and decoded. Some formats performed better with large strings (Input 3) while others performed better with additional maps (Input 2). Therefore, the choice of the format should take into consideration the type of data required for encoding and more specifically decoding.

Metric	**ASN.1 DER**	**CBOR**	**JSON**
Encoding size [bytes]	318	**316**	959
Encoding speed [ms]	89	**81**	151
Decoding speed [ms]	61	240	**53**
Encoding memory consumption [bytes]	15632	**10021**	12711
Decoding memory consumption [bytes]	17843	**12436**	13130

Table 4.6: Summary of formats' performance.

According to the results presented in table 4.6, on average, both ASN.1 DER and CBOR have a much smaller encoding size than JSON. Therefore, if the application is constrained in terms of buffer size or storage size, then

JSON might not be a suitable option. On the other hand, compared to ASN.1 and CBOR, JSON has the fastest average decoding speed. Hence, an application that wants fast decoding should most probably utilize JSON. These calculations were done over 100,000 decoding iterations. The time it would take to decode a single object is much smaller. Therefore, the time difference might not be an important factor if just a single object is being decoded. One other point to note is that even though JSON has the fastest decoding time, more time will be spent in transmission and reception due to its relatively large size.

The format with the fastest decoding speed (JSON) is different than the format with the fastest encoding speed (CBOR). Therefore, if the performance of the encoding speed is important, then CBOR or ASN.1 are a more suitable choice than JSON. Furthermore, CBOR applications have the lowest memory consumption on average for both the encoding and decoding applications. If RAM size is the bottleneck in a device, then CBOR is the most suitable format. Overall, CBOR is the most suitable format in an attestation environment where the relying party is a constrained device that needs to parse the attestation result object. While its decoding speed is slower than that of ASN.1 and JSON, it has the lowest code footprint and the smallest message size.

Aside from performance, there are several other points that are worth analyzing when it comes to choosing a format. These points were faced during the implementation and affect the application usage and maintainability. As mentioned in section 4.3.3, the ASN.1 schema for attestation results needed to be implemented before any encoding or decoding could be done. This schema needs to be maintained and any changes to the attestation result objects will render the encoding and/or decoding code useless as the code expects a specific structure. Therefore, modifications to the object needs changing to the schema, encoding, and decoding rules. One other difficulty faced when implementing the ASN.1 decoding is checking for optional fields because there are no specified tags for each encoded item. In short, ASN.1 DER tends to have additional implementation overhead than other formats and might not be backwards compatible with older versions of the schema.

The CBOR encoding size significantly improves when the keys in the key-value pairs are integers, a feature not possible in JSON. The downside of this improvement is that there should be a mapping between the integer key value and the actual information it represents. This mapping adds some overhead when communicating the encoded messages since even though CBOR is an unstructured binary serialization, the mapping needs to be communicated in all applications performing encoding and decoding. Unlike ASN.1, JSON and CBOR are usually backwards compatible where older decoders and encoders can still work even when new values are added to the object.

An important consideration is the availability of the library in the target platform. It is also important to choose a library that has been extensively tested so that no library-related problems are faced during production.

Finally, based on the benchmarks and kinds of inputs, it is shown that the text-based serialization is actually comparable to the binary serializations when it comes to encoding speed, decoding speed, and code footprint. Most of the work surveyed in section 3.3 showed that binary serialization is faster than text-based. But the results derived here show that JSON is the fastest, at least in decoding speed. There are several factors that could contribute to such an outcome like the libraries used and the kind of data being encoded. Some online benchmarks [47, 66] have also reported that JSON was faster, at least in decoding, than several binary serializations. Therefore, the generalized assumption that binary serializations are faster is not accurate at least in the case of attestation results. On the other hand, the encoding size of binary serialization does tend to be smaller.

The number of strings used in the benchmark inputs could be one factor contributing to the comparable performances of JSON to the binary formats. The more strings used, the more similar the results of the different serializations are, since strings are encoded in the same manner and binary serializations use the same encoding (1 byte per character). It is important to note that while the raw evidence was represented as a base64 string, it could have been represented as an array of bytes in CBOR and ASN.1 DER which would lead to different results. It could also be observed that the binary serializations were more efficient when encoding integers compared to the text-based serializations because binary serializations have different rules for integer encoding. This is evident in section 4.4.1 where adding an additional attester object with more integer measurements led to significantly smaller sizes in ASN.1 and CBOR than in JSON. In other words, the type of data being encoded is an important factor to consider when choosing a format.

Chapter 5

Attestation Protocol for Constrained RP

To the best of our knowledge, there are no protocols in the literature that consider the case where the relying party is constrained. In the era of IoT, we are surrounded by numerous resource-constrained devices. Even those small devices need to verify the integrity of the device it is communicating with in order to reduce the possibility of attacks in IoT applications. Therefore, there is a need for a lightweight and efficient protocol that allows a constrained relying party to validate and authenticate a device before interacting with it. This chapter introduces and formally verifies the Attestation Protocol for Constrained Relying party–Live Passport Model (APCR-LPM) which is a lightweight and resource efficient protocol for communicating attestation evidence and results. One key design choice in this protocol is the use of only symmetric key-cryptography in the relying party. This ensures that any kind of device acting as a relying party can deploy the protocol without requiring any special hardware capabilities. This chapter introduces the initial version of the APCR-LPM protocol, discusses the attack found using formal verification, and presents the improved version of APCR-LPM.

5.1 System Model

We consider an environment where the entity that wants a proof of authenticity, the relying party, is a constrained device. Therefore, the protocol describes how other entities in the attestation procedure interact and relay information to the relying party. The protocol consists of the following entities:

- Relying Party RP: The relying party is a constrained device that wants to evaluate the trustworthiness of a device it is communicating with. This device has a small memory, may be too constrained to establish more than one connection at the same time, and might not have the buffer or processing capacity to receive and verify evidence. In addition, the relying party cannot perform public key cryptography. On the other hand, it can do symmetric key cryptography and generate 128-bit pseudorandom numbers.

- Attester A: The attester is a non-constrained device, e.g. a smartphone, that wants to prove its trustworthiness to the relying party. It must at least contain a Trusted Execution Environment (TEE) (see chapter 2) and be able to perform public key cryptography. We assume that the attester generates evidence or metrics that the verifier can understand. In addition, the TEE provides key attestation of a key that is stored in the TEE and cannot be extracted.

- Verifier V: The verifier is also a non-constrained (**trustworthy**) device. It can be a cloud server, for example. The purpose of the verifier is to validate or appraise the evidence sent by the attester. The verifier has a policy according to which it appraises the received evidence. The policy is agreed between RP and A. After appraising the evidence, the verifier sends the attestation results to the relying party.

5.2 Adversary Model

We assume an active attacker model with Dolev-Yao [23] capabilities. The adversary can read, intercept, insert, and modify the messages exchanged by the parties in the protocol. However, the attacker is not able to break or guess the key unless explicitly stated. In other words, cryptography is assumed to be perfect. In addition, the attacker might perform physical attacks on some of the participants. More specifically, the attacker can compromise the REE [1] part of the attester. The attacker can also play a role of another relying party towards the attester. Possible goals of the attacker include:

- Falsifying the attestation evidence and making the verifier accept the falsified metrics.

- Using some other device to generate evidence that the relying party would accept as genuine.

[1]The Rich Execution Environment (REE) refers to the normal mode of operation, i.e, the normal state of the CPU and the software that runs in that security state. It can be considered the non-TEE part of the device.

Assumptions

- The attacker cannot compromise Verifier V.
- The attacker cannot compromise the TEE of the concerned attester.
- The attacker can use genuine parties RP or A as oracles.
- The attacker can pretend to be any of the participating entities.

5.3 Security Requirements

- Freshness: Participants should be able to detect whether the messages being sent belong to an old or different run of the protocol. Freshness mechanisms can prevent replay attacks where an attacker repeats an old message that will lead the verifier or relying party to accept. In other words, freshness is necessary because it reflects the state of the attester such that the verifier does not accept old metrics that might not be representative of the current attester state.
- Data Origin Authentication: Participants should be able to verify that a message has been generated by a particular entity.
- Message Integrity: Participants should be able to detect if the attacker has corrupted or modified any of the messages sent in the protocol.

5.4 Protocol Description

APCR-LPM is a low-resource and lightweight protocol proposed by Huawei's Helsinki System Security Lab that enables small devices to evaluate the trustworthiness of the device it is communicating with. It uses the passport model discussed in section 2.2.2. In addition, all the cryptographic primitives and remote attestation terms are highlighted in section 2.2. APCR-LPM assumes that a key distribution protocol has taken place beforehand. This key distribution protocol can be conducted only once and is independent of APCR-LPM. Figure 5.1 illustrates the protocol and the interactions between its participants. In addition, table 5.1 explains the notation used in the protocol.

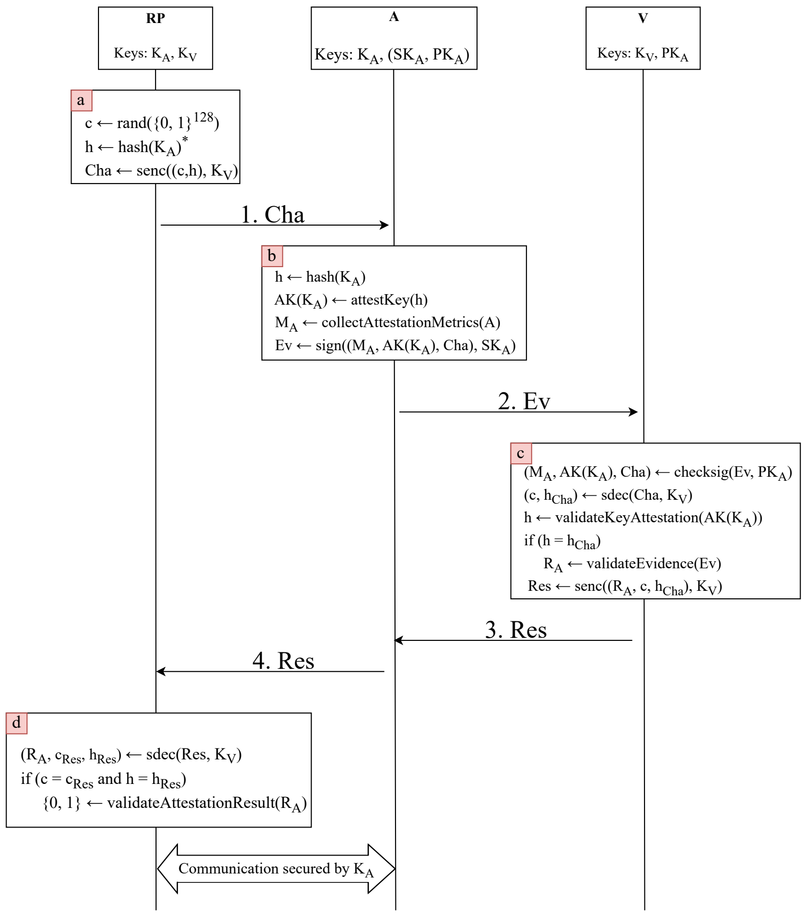

RP
Keys: K_A, K_V
A
Keys: K_A, (SK_A, PK_A)
V
Keys: K_V, PK_A
a
c ← rand({0, 1}^128)
h ← hash(K_A)*
Cha ← senc((c,h), K_V)
1. Cha
b
h ← hash(K_A)
AK(K_A) ← attestKey(h)
M_A ← collectAttestationMetrics(A)
Ev ← sign((M_A, AK(K_A), Cha), SK_A)
2. Ev
c
(M_A, AK(K_A), Cha) ← checksig(Ev, PK_A)
(c, h_Cha) ← sdec(Cha, K_V)
h ← validateKeyAttestation(AK(K_A))
if (h = h_Cha)
R_A ← validateEvidence(Ev)
Res ← senc((R_A, c, h_Cha), K_V)
3. Res
4. Res
d
(R_A, c_Res, h_Res) ← sdec(Res, K_V)
if (c = c_Res and h = h_Res)
{0, 1} ← validateAttestationResult(R_A)
Communication secured by K_A

Figure 5.1: The initial protocol.

(*) The hash should be computed once when the RP receives key K_A

5.4.1 Initial State

At the start of the protocol, the relying party has symmetric keys K_A, K_V shared with the attester and verifier, respectively. The RP also has the hash of the key K_A. In addition, the attester has a private/public key pair which is used to authenticate the attester to the verifier, i.e., the verifier knows the attester's public key.

5.4.2 Description of Initial Protocol

In order to understand the attack found using formal verification, the initial version of APCR-LPM must first be explained and the formal modelling discussed. The explanation and purpose of each of the steps illustrated in figure 5.1 are as follows:

Step a: the relying party prepares the challenge it will send to the attester by generating the nonce c. The RP then encrypts using K_V the nonce and the hash of the key K_A it shares with the attester. The nonce c acts as a session identifier as well as a freshness value. The value h is the identity of a specific attester that the RP wants to verify.

Message 1: The RP sends the challenge to the attester. The attester cannot parse or determine the contents of the message as it is encrypted with a key that it does not know.

Step b: The attester collects the attestation metrics of the device and performs key attestation on the stored key K_A. The key attestation is used as an indicator of the identity of A. By including the key attestation, the TEE vouches that K_A cannot be extracted from A. The collected attestation metrics are in an agreed upon format that the verifier can parse and understand.

Message 2: The attester signs the metrics, the key attestation, and the challenge it has received from the RP using its private key to produce attestation evidence. The signing allows the verifier to know and authenticate the identity of the attester it has received the evidence from.

Step 3: The verifier first verifies the signature of the evidence coming from A and decrypts the challenge to extract c and h. Then, it verifies the key attestation and ensures the values of h sent by the RP and A are equal. If any of those steps fail, the verifier does not complete the protocol or respond to A. Based on the metrics, the verifier generates a verdict or attestation result for the relying party. The RP can then use the verdict to decide whether the attester is trustworthy or not.

Message 3: The verifier sends the attester the attestation results together with the nonce c and h_{Cha}. The message is encrypted with K_V, hence the attester is not able to read the contents of the message. The hash h_{Cha} is included in the message as an indicator to the relying party that the verifier has verified the evidence of the attester with identity h_{Cha}. It also includes the decrypted challenge c, h_{Cha} to maintain freshness of message and to indicate to the relying party that the intended verifier has received the challenge and decrypted it.

Message 4: The attester forwards Message 3 to the RP unchanged. The verifier does not send it directly to RP in order to limit the number of endpoints that the relying party needs to know.

Step d: The relying party decrypts the received message and checks that the values of c_{Res} and h_{Res} are equal to the ones it sent in the challenge. The relying party can then process the attestation result and determine whether to trust the attester or not.

Subsequent communication between the relying party and attester can be secured using K_A. This communication depends on the application where the relying party and attester are deployed. APCR-LPM allows the relying party to authenticate the attester and verifier using only symmetric cryptography. In addition, the protocol only requires the relying party to establish a single connection making it suitable for small and limited IoT devices. Furthermore, using this protocol, the relying party is able to verify the authenticity or trustworthiness of a device by sending a single message that is minimal in the size of bits.

Term	**Description**
RP	the relying party, a small device that wants to evaluate A
A	the attester, an unconstrained device that wants a resource or data from RP
V	the verifier, an unconstrained device that evaluates A for RP
K_A	shared symmetric key between A and RP
K_V	shared symmetric key between V and RP
SK_A	secret key of the attester
PK_A	public key of the attester
h	the hash of K_A
c	a 128-bit pseudorandom value
M_A	the attestation metrics/claims produced by the attester
R_A	the attestation result produced by the verifier
Procedure	**Description**
$r \leftarrow rand(\{0,1\}^{128})$	generating pseudorandom bitstrings of 128 bits
$c \leftarrow senc(m, K)$	symmetric encryption of message m with key K
$m \leftarrow sdec(c, K)$	symmetric decryption of ciphertext c with key K
$sig \leftarrow sign(m, SK)$	signing the message m with secret key SK
$m \leftarrow checksig(sig, PK)$	verifying the signature with the public key PK
$h \leftarrow hash(m)$	computing the hash of value m
$AK(K) \leftarrow attestKey(h)$	key attestation of key K that is protected by device TEE
$h \leftarrow validateKeyAttestation(AK(K))$	validates that key K is coming from device TEE
$M \leftarrow collectAttestationMetric(E)$	compute the attestation metric of entity E
$R \leftarrow validateEvidence(Ev)$	compute attestation results R based on evidence Ev
$\{0,1\} \leftarrow validateAttestationResult(R)$	determine trustworthiness based on the attestation results R

Table 5.1: Notation used in APCR-LPM

5.5 Formal Verification of APCR-LPM

Formal verification is the process of modelling a system or component–in our case a communication protocol–using mathematical structures to prove the correctness of the system. Formal verification of protocols can help uncover attacks that might go unnoticed as protocol verification tools can simulate unbounded number of protocol sessions and different kinds of attacks on the protocol. Formal verification has been used to evaluate and discover some vulnerabilities in well-known protocols such as TLS 1.3 [11] and Signal [50].

5.5.1 Modelling with ProVerif

ProVerif [15] was used for modelling APCR-LPM as it provides flexibility in defining functions and actions. ProVerif is a tool that is used for analyzing the security of cryptographic protocols. It can test the protocol for different security properties like confidentiality, authenticity, integrity, privacy, and traceability. In order to test the desired security properties, a security query describing the property is specified in the ProVerif model. Then, ProVerif explores all possible execution paths of the protocol trying to find a path that will make the query fail. In addition, ProVerif assumes the Dolev-Yao attacker model and can perform replay, man-in-the-middle and spoofing (impersonation) attacks [15]. Hence, it aligns with our assumed adversary model discussion in 5.2.

Formulating the model of APCR-LPM in ProVerif was divided into two steps; the first step was modeling the behavior of the protocol and the second was defining the security queries. In the first step the cryptographic primitives and the actions that each of the participants performs throughout a run of the protocol were specified. The second step, which is a far more difficult task, was to correctly define the security requirements of the protocol for ProVerif to test.

5.5.2 Modeling Participants' Behavior

Modeling the protocol participants and their behavior can further be divided into 3 steps: defining the data types, defining the functions/constructors, and finally specifying the interactions.

Types: ProVerif's input language is strongly typed with a finite set of pre-defined types. If a user wants to use a type that is not part of this pre-defined set, like key or nonce, then they must declare them. In addition, all functions are defined by their input and output types. Types help enforce stricter rules on functions as they constrain the kind of input functions can take. The APCR-LPM ProVerif model has 4 defined types which correspond to the shared symmetric keys, SK_A, PK_A, and c described in table 5.1.

$$\begin{aligned}&type\ nonce.\\&type\ key.\\&type\ skey.\\&type\ pkey.\end{aligned}\tag{5.1}$$

In addition, The APCR-LPM ProVerif model also uses the built-in types *bitstring* and *channel*. The *bitstring* type is used for all inputs that are generic, like messages. The *channel* type is used to indicate a communication channel. Our model specifies only a single public channel 'network' $free\ network : channel.$ which is used for communication. This channel is globally known and part of the attacker's knowledge. The attacker can listen to and intercept the messages sent through the channel.

Functions: Constructors, or function symbols, are used to build cryptographic primitives and relationships between (different) subjects in the protocol. The constructor is a function that defines a behavior based on input and output types. For example, to model symmetric encryption, the constructor *senc*, specified in definition 5.2, takes as input a bitstring (message) and key and returns a bitstring (ciphertext). Destructors, on the other hand, take as input constructors as well as other types, and return a constructor input. The *sdec* function, specified in definition 5.2, is an example of a destructor. *sdec* specifies that for any message m inserted as input in *senc*, if the same key is used in both *senc* and *sdec*, the same message is returned in *sdec*. Destructors are named that way because they break down the input parameters of constructors to output one of those parameters.

$$\begin{aligned}&fun\ senc(bitstring, key) : bitstring.\\&reduc\ forall\ m : bitstring, k : key; sdec(senc(m, k), k) = m.\end{aligned}\tag{5.2}$$

Equivalently, a similar definition of asymmetric encryption can be formulated. The function *pk* is used to model the relationship between the private and public keys where the public key is defined as a function of the private key. Formulating the cryptographic primitives this way allows to abstract the inner workings of cryptography and to treat the algorithm as a black box. APCR-LPM requires symmetric encryption in the *RP* and signing with public key cryptography in the attester. Therefore, definitions 5.2 and 5.3 are used in the APCR-LPM ProVerif Model.

$$\begin{gathered}fun\ pk(skey) : pkey.\\ fun\ sign(bitstring, skey) : bitstring.\\ reduc\ forall\ m : bitstring, k : skey; checksign(sign(m, k), pk(k)) = m.\end{gathered} \tag{5.3}$$

Interactions: The final step is to specify the participants' interaction in the protocol. ProVerif has specific functions *in* and *out* used to receive input from and send output to the network respectively. In turn, the attacker knows when a party is expecting an input and when it is sending an output. Essentially, receiving a message is just waiting on the $'in'$ interface where an attacker maybe the one sending the message. Furthermore, when sending a message, the party does not specify to which entity it is directed at; the attacker tries all possible combinations and some of those combinations are the intended protocol flow. In essence, the attacker is the one transmitting the messages, as per the Dolev-Yao model.

Modelling the different protocol participants can now be easily done by using the concepts of types, constructors, destructors and interactions. For example, the following piece of code in definition 5.4 describes **step a** and **message 1** in figure 5.1. The relying party generates a new pseudorandom number (specified with keyword new) c of type nonce. It then computes h and generates Cha which is the encryption of c and h using key K_V. Then, the relying party sends Cha over the network.

$$\begin{aligned}&new\ c : nonce;\\ &let\ h = hash(K_A)\ in\\ &let\ Cha = senc((c, h), K_V)\ in\\ &out(network, Cha);\end{aligned} \tag{5.4}$$

The full ProVerif code containing all steps and messages of the different parties can be found in appendix C.1. ProVerif allows running multiple instances of participants in parallel. Therefore, there may in principle be an unbounded number of sessions running in parallel between the relying party, attester and verifier. However, there are only multiple sessions, not multiple relying parties, attesters and verifiers.

Modeling Challenges: There were some challenges in modeling the remote attestation related assumptions in ProVerif. Primarily, the attester's TEE is an environment that is trusted but an attacker can still compromise the REE. This involves modeling part of the attester as trustworthy and part as untrustworthy. In addition, it is difficult to formulate the key attestation

$AK(K_A)$ which is a guarantee from the attester's TEE that K_A cannot be extracted.

One solution to this problem is to split the attester into two entities and create a private channel between them. Another solution would be to accept some input from the network, essentially from the attacker, and let M_A be a function of that input. The key attestation would then be produced by the trustworthy entity. We simplified the modeling by assuming that the attester is a trustworthy participant in the protocol, essentially modeling only the functionality of the TEE. The motivation behind this is that the protocol's security should be independent from the content of the metrics M_A. Therefore, validating the metrics is left to the procedures *validateEvidence* and *validateAttestationResults*.

One other challenge is the way to model the attestation metrics M_A. M_A must adhere to the format agreed upon by the attester and verifier. Therefore, the structure of the data found in the metrics is not changing or random as it must adhere to a certain format. In addition, if the state of the attester remains the same, multiple runs of the protocol can result in the exact same value of M_A. Modeling such a behavior, where the data is not always new and not always constant, in ProVerif is difficult. In this formulation, M_A is modeled as a changing value which is more similar to attestation metrics in practice than modeling it as a constant value.

5.5.3 Defining Security Queries

Now that the behavior of the participants has been modeled, the remaining step is defining the security queries that ProVerif uses to test the protocol. ProVerif uses symbolic model checking[2] to try to reach a state where a security query is not satisfied. It is important to correctly represent the protocol's security requirements in terms of security queries for ProVerif to prove. The queries used in APCR-LPM use ProVerif events and the Injective Correspondence property described below:

1. Events: The processes in ProVerif (in this case the relying party, attester, and verifier) are annotated with events that mark important stages reached by the protocol. These events do not affect the behavior of the protocol, they are just indicators that the protocol has reached a particular step. There are four events defined in the APCR-LPM

[2]Model checking is a technique used to check whether all possible states of a system satisfy certain conditions. Symbolic model checking is a type of model checking used to handle state-space explosions by representing the system as sets of states and sets of transitions instead of listing all possible states and transitions [10].

model.

$$\begin{aligned}&event\ relyingPartyBegins(K_V, K_A, c);\\&event\ attesterBegins(PK_A, K_A, M_A);\\&event\ verifierAccepts(PK_A, K_V, M_A, c);\\&event\ relyingPartyAccepts(K_V, K_A, c, R_A);\end{aligned} \tag{5.5}$$

The first event signals that the relying party has started a run of the protocol with the parameters K_V, K_A, and c. The second event indicates that the attester has received a message (Cha) and collected its attestation metrics. The third event signifies that the verifier has received the message (Ev) and verified the key attestation. Finally, the last event represents a state where the relying party has received a message (Res) and verified that the values of c and h it has received is equal to the ones it has sent.

2. Injective Correspondence: Correspondence in ProVerif is a way to determine the relationship between the events. ProVerif also introduces injective correspondence for specifying that a one-to-one relation between the number of protocol runs performed by each process is desired. In the APCR-LPM model, injective correspondence is used to specify that the relying party will not accept attestation results R_A unless there are distinct earlier events where the relying party has issued a challenge, the attester collected its metrics, and the verifier validated said metrics.

There are two security queries specified for APCR-LPM model. The first is used to test the secrecy of the attestation result R_A and the second is used to test the authentication and integrity of the protocol. The first query 5.6 states that the events of the attacker acquiring R_A and the relying party accepting R_A cannot happen in the same protocol run. Note that the secrecy of R_A is not a security requirement. This query was added to demonstrate the effects of the relationship between the metrics M_A and attestation results R_A.

$$\begin{aligned}&query\ K_A : key, K_V : key, R_A : bitstring, c : nonce;\\&attacker(R_A)\ \&\&\ event(relyingPartyAccepts(K_V, K_A, c, R_A)) \implies false.\end{aligned} \tag{5.6}$$

The second query 5.7 specifies the injective correspondence between the event of the relying party accepting R_A and all other events in the protocol. The relying party will only accept the protocol run if there has been a previous run where it initiated the protocol by sending Cha, the attester has accepted this Cha and collected attestation metrics M_A, and the verifier has received

M_A and decrypted Cha. There is an added constraint as well on the relation between the metrics M_A and the attestation results R_A.

$$\begin{aligned}
&query\ PK_A : pkey, K_A : key, K_V : key, R_A : bitstring, c : nonce,\\
&h : bitstring, M_A : bitstring, Cha : bitstring;\\
&inj - event(relyingPartyAccepts(K_V, K_A, c, R_A))\\
&\quad \Longrightarrow inj - event(relyingPartyBegins(K_V, K_A, c))\ \&\&\\
&\qquad inj - event(attesterBegins(PK_A, K_A, M_A, Cha))\ \&\&\\
&\qquad inj - event(verifierAccepts(PK_A, K_V, M_A, c, h))\ \&\&\\
&\qquad h = hash(K_A)\ \&\&\\
&\qquad R_A = validateEvidence(M_A)\ \&\&\\
&\qquad Cha = senc((c, h), K_V).
\end{aligned} \tag{5.7}$$

This query models the security requirements (section 5.3) by including c in the events for freshness, the keys for data origin authentication and M_A and R_A for integrity. The metrics M_A is matched to both the attester and verifier event and the relying party will not accept the final message unless R_A is a function of M_A and it has received back the encrypted c and h it used in the first message.

5.5.4 Query Results

For the query defined in 5.6 the attacker was able to determine the contents of R_A. The reason is that the attestation evidence is signed but not encrypted. Therefore, the attacker can easily derive the attestation results R_A by applying the function *validateEvidence*. The secrecy of the attestation result is not a security requirement, but the query highlights how the attestation parameters have an influence on the security of the protocol and whether the metrics should be confidential or not.

Since APCR-LPM is a simple protocol, ProVerif is able to check all possible protocol states and terminate. ProVerif did not find an attack against the query defined in 5.7. Therefore, at least when there is only one relying party, one attester, and one verifier, the protocol satisfies the security requirements. Logically, the next step is to introduce more than one attester and evaluate how the protocol behaves in this situation.

5.5.5 Introducing another Attester

To introduce another attester to the ProVerif model, a new symmetric key K_{A_2}, a new asymmetric key pair SK_{A_2}, PK_{A_2}, and a new attester process with those keys were added to the model. There were no attacks on the protocol when allowing the relying party and verifier to have sessions with both attesters. However, a question arises of what will happen if one attester is compromised, i.e. has its TEE broken and its private key leaked. Naturally, if the relying party wants to verify this corrupted attester, an attacker can generate its own metrics M_{A_2} and use the leaked private key to sign the evidence. It is important to verify that this compromised attester will not affect the security of the protocol between the relying party and the uncompromised attester.

The model needs to be refined to represent the corrupted attester and the security query 5.7 should be updated to test the security of the uncorrupted attester. In other words, ProVerif should verify that the security query for the uncorrupted attester still holds. The following changes in the model are required in order to introduce a compromised attester and refine the query to verify the security of the protocol when there is a corrupted attacker present.

1. The main process now initializes a new key K_{A_2} and a public/private key pair PK_{A_2}, SK_{A_2} that represents the new attester A_2.
2. The main process leaks A_2's private key SK_{A_2} to the network. This action simulates a corrupted attester in the network as its private key is now known by the attacker.
3. An event '$attesterCorrupted(K_A)$' is defined in order to identify the corrupted attester by binding the event to the attester's identity K_A.
4. Query 5.7 is updated to enable ProVerif to test for the security of the protocol for an uncompromised attester. Query 5.8 now states that the relying party will only accept *Res* either from a compromised attester, annotated by the event *attesterCorrupted* or by the conjunction of events discussed in section 5.5.3. This formulation allows to test the security of only the authentic attester since in the case of the compromised attester $event(attesterCorrupted(K_A))$ will always be true. If this event was not present, the query will always be false because of the trivial reason that the attacker can fake signatures when the relying party wants to verify the compromised attester. Adding this event allows testing the non-trivial case where the relying party wants to verify an uncompromised attester. The ProVerif model for adding a

new attester and refining the query can be found in the appendix C.2.

$$\begin{aligned}
&query\ PK_A : pkey, K_A : key, K_V : key, R_A : bitstring, c : nonce,\\
&h : bitstring, M_A : bitstring, Cha : bitstring;\\
&inj-event(relyingPartyAccepts(K_V, K_A, c, R_A))\\
&\quad \Longrightarrow event(attesterCorrupted(K_A)) \mid\mid\\
&\qquad (inj-event(relyingPartyBegins(K_V, K_A, c))\ \&\&\\
&\qquad\ inj-event(attesterBegins(PK_A, K_A, M_A, Cha))\ \&\&\\
&\qquad\ inj-event(verifierAccepts(PK_A, K_V, M_A, c, h))\ \&\&\\
&\qquad\ h = hash(K_A)\ \&\&\\
&\qquad\ R_A = validateEvidence(M_A)\ \&\&\\
&\qquad\ Cha = senc((c, h), K_V)).
\end{aligned} \tag{5.8}$$

5.5.6 Attack Description

ProVerif discovered an attack on the uncompromised attester A. When the relying party sends a challenge to A, the attacker can generate its own attestation metrics and sign the challenge with the compromised private key of the attester A_2. The verifier receives the signed signature and verifies the signature as it knows the 'trusted' public key of the attester. Then it verifies the rest of the message as done in **step c** in section 5.4.2. The relying party will then accept the attestation result sent by the verifier assuming the verifier has validated the correct metrics of attester A, while in fact it was the attacker that generated the metrics. A visual representation of the attack is found in figure 5.2. As shown in the figure, the attacker intercepts **message 2** intended for the verifier to acquire $AK(K_A)$ and Cha. Then it forges the attestation metrics, signs evidence Ev' with the compromised key and sends it to the verifier. The verifier has no binding between the identity h and the public key. Therefore, it verifies the signature using the public key that belongs to a different attester; then it compares the key attestation with the value sent in Cha. All the checks in the verifier pass, and it generates attestation results based on those forged metrics. It is important to note that only the steps that demonstrate a different behavior from the original protocol were specified in figure 5.2.

The reason that this attack is possible is that there is no connection between the how the relying party identifies the attester and how the verifier identifies the attester. The relying party uses h for identification while the verifier uses the public key PK_A where there is no tie between both values.

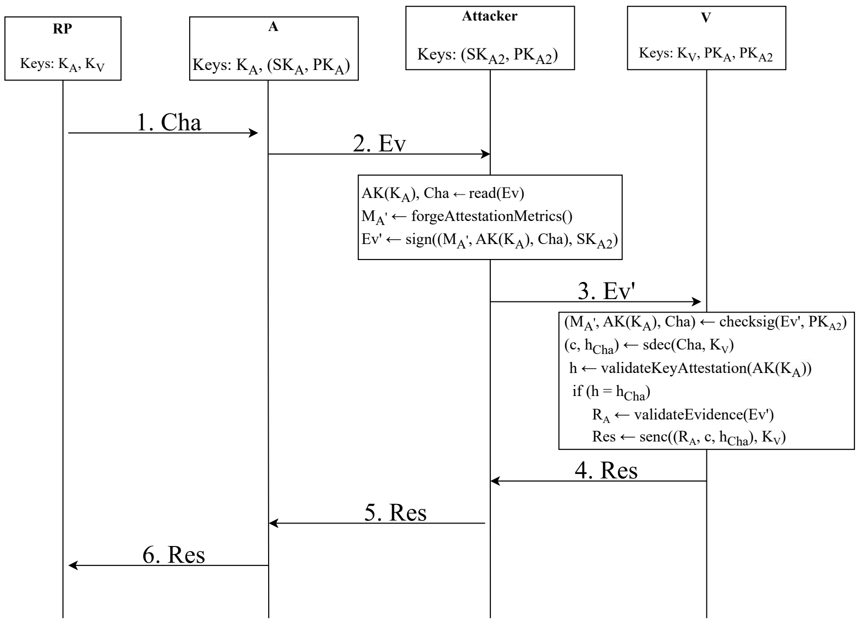

RP
Keys: K_A, K_V
A
Keys: K_A, (SK_A, PK_A)
Attacker
Keys: (SK_A2, PK_A2)
V
Keys: K_V, PK_A, PK_A2
1. Cha
2. Ev
AK(K_A), Cha ← read(Ev)
M_A' ← forgeAttestationMetrics()
Ev' ← sign((M_A', AK(K_A), Cha), SK_A2)
3. Ev'
(M_A', AK(K_A), Cha) ← checksig(Ev', PK_A2)
(c, h_Cha) ← sdec(Cha, K_V)
h ← validateKeyAttestation(AK(K_A))
if (h = h_Cha)
R_A ← validateEvidence(Ev')
Res ← senc((R_A, c, h_Cha), K_V)
4. Res
5. Res
6. Res

Figure 5.2: The attack by a compromised attester on APCR-LPM.

In addition, the messages don't include the parties intended and rely on the keys for authentication. There are several ways to fix the protocol including encryption, self-certifying public keys and certificates.

5.6 Improving the Protocol

A correlation between how RP and V identify A is needed in order to prevent the attack from being carried out. One common solution to this problem is introducing certificates where the identity h is bound to the public key PK_A. However, this adds another dimension to this protocol by introducing certificate management. Another solution, illustrated in figure 5.3, is that the identifier used by the relying party to specify the attester can be a function of A's public key. The benefits of this solution include:

1. Minimal changes to the original protocol as instead of having h a function of K_A, RP now has an identifier of A that is a function of both K_A and PK_A.

2. No additional assumptions, like a certificate authority, is needed to implement the solution.

3. The simplistic nature of the protocol is unchanged as there is no additional cryptography used. In addition, the messages size and content remain the same.

4. The relying party does not need to know how the identifier *id* is generated, it can just be a value used to identify A.

5. No additional functionality is required from the attester under evaluation.

6. The identification of attester A from the relying party's perspective can now be validated by the verifier.

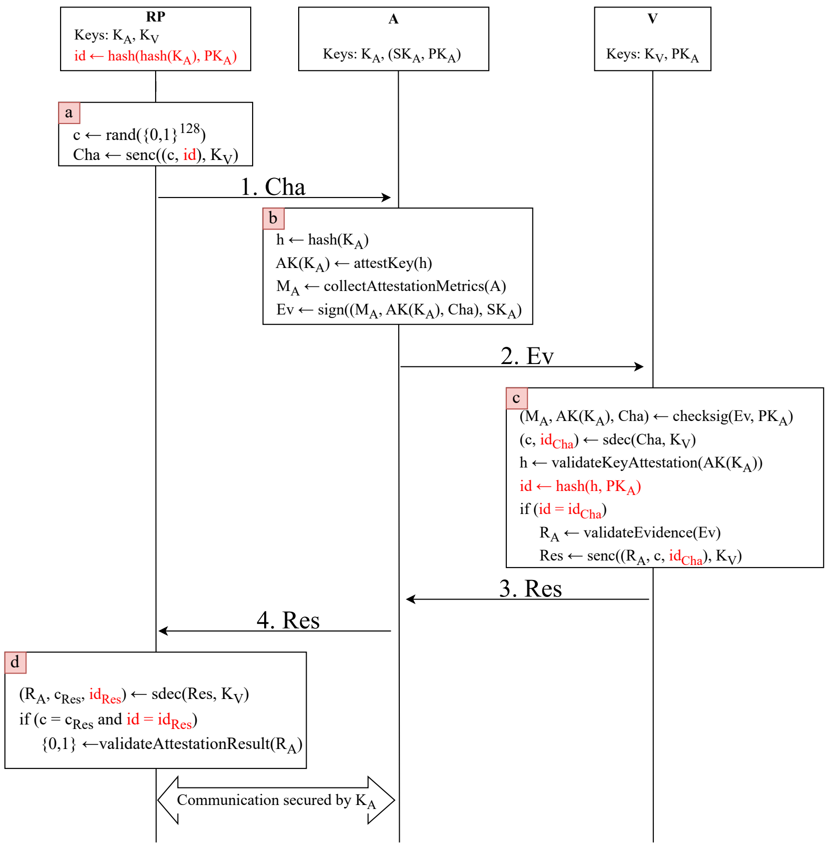

RP
Keys: K_A, K_V
id ← hash(hash(K_A), PK_A)
A
Keys: K_A, (SK_A, PK_A)
V
Keys: K_V, PK_A
a
c ← rand({0,1}^128)
Cha ← senc((c, id), K_V)
1. Cha
b
h ← hash(K_A)
AK(K_A) ← attestKey(h)
M_A ← collectAttestationMetrics(A)
Ev ← sign((M_A, AK(K_A), Cha), SK_A)
2. Ev
c
(M_A, AK(K_A), Cha) ← checksig(Ev, PK_A)
(c, id_Cha) ← sdec(Cha, K_V)
h ← validateKeyAttestation(AK(K_A))
id ← hash(h, PK_A)
if (id = id_Cha)
R_A ← validateEvidence(Ev)
Res ← senc((R_A, c, id_Cha), K_V)
3. Res
4. Res
d
(R_A, c_Res, id_Res) ← sdec(Res, K_V)
if (c = c_Res and id = id_Res)
{0,1} ←validateAttestationResult(R_A)
Communication secured by K_A

Figure 5.3: The improved protocol.

The changes to the original protocol (see figure 5.1) are highlighted in red in figure 5.3. The verifier now needs to compute the *id* value based on the value of h it has received from the attester and the public key it has used to verify the signature. Then, the verifier compares the values of the id it has computed to the value of the id it has received from RP in Cha. As shown in the figure, the behavior of the protocol is essentially the same with just an additional computation of id done in the verifier. The ProVerif model of the improved protocol can be found in appendix C.3.

5.7 Analysis

APCR-LPM is a small and lightweight protocol created with the goal of allowing even simple devices to attest the integrity of other devices. It involves no asymmetric cryptography from the relying party's side making it suitable for all classes of devices especially those with limited resources. The novelty of the protocol stems from the fact that no literature has proposed an attestation scheme where the relying party (and not the attester) is a constrained device. In addition, according to the formal verification done by ProVerif, the protocol is safe under an active attacker given the assumptions listed in section 5.2. The improved protocol also takes into consideration compromised devices as a physical attack on one TEE does not affect the rest of the network.

The utilization of ProVerif enabled exploring all possible execution paths of the protocol to test that it achieved the security properties specified in the query. Even though the improved protocol passes the security query 5.7, there are a few technical and conceptual aspects that need to be highlighted. The first of which is that the protocol assumes that the attester can communicate with only one verifier as it has no way of knowing to which verifier **message 1** in the protocol is intended for. Therefore, messages in the APCR-LPM might need modification if the application involves many verifiers.

The protocol also assumes a key distribution mechanism has taken place before the protocol run. One thing to consider is whether this key distribution, at least from the relying party's perspective, happens during manufacturing or bootstrapping process between the relying party and the attester. A supply chain viewpoint needs to be taken if it is during manufacturing since the different devices for the different parties might belong to the same or different vendors. In addition, the key distribution mechanism might also need to employ some revocation rules in order to handle cases where the private key is compromised and exploited as explained in section 5.5.6. One other problem is that the participants, especially the attester, might be susceptible to Denial of Service (DoS) attacks as it does not perform any checks on the message it receives. However, the attester can assume that the surrounding environment performs DoS protection. Generally, all these problems have been previously explored and solved in the literature, thus they are out of this scope. The focus of the protocol is only on the ability to securely communicate attestation metrics and results to the intended parties in the specific domain of a constrained relying party.

As the protocol is concerned with the attestation of devices, the relation between attestation metrics and results influence the security of the protocol. The metrics are not signed and hence can be read by any passive or active attacker. Furthermore, because the attestation result is always a function of the metrics, then the results are also public even if they are encrypted before being sent. In addition, even encrypting the evidence before sending them to the verifier might not completely hide the values of M_A since in reality M_A can be a predictable value. The main purpose of the protocol is to ensure the integrity of the results received by the relying party which is achieved. However, the privacy of the attester is not regarded as the network can know the collected metrics and hence the state of the attester. This raises the question of whether M_A should be public or not.

According to the assumptions set for the protocol, the attester and verifier run on devices that can use public key cryptography and TLS. Hence, the communication between RP and V can be secured by a TLS channel. This will provide confidentiality of the metrics M_A and make the communication between attester and verifier more secure.

The improved APCR-LPM can still be considered a standalone protocol that can work with any communication channel between the participants. The protocol also provides flexiblity as the communication channel between the RP and A can be different than that of A and V. For example, RP and A can communicate over Bluetooth and A and V can communicate over the Internet.

One further aspect to consider is how the differences between the protocol model and the actual protocol impact the formal verification results. The protocol abstracted the relation between the REE and TEE in the attester to simulate only the behavior of the TEE. In addition, the model treated M_A as a new and never before seen value with every run of the protocol. But the actual metrics might be similar or even the same in multiple protocol runs. This behavior is difficult to model. The metrics M_A was additionally simulated as a constant non-changing value and tested to ensure that adding a new value with every run of the protocol does not give the modeled protocol strength that is not a reflection of its real-world application. This simulation did not result in any attacks and the security query 5.7 is still achieved.

5.8 Related Work

There exists a gap in the literature for constrained devices that want to perform attestation. Most of the proposed literature handle situations where a 'powerful' device attests low-resource devices. The term powerful in this case refers to devices that have memory at least in the order of megabytes and can perform public key cryptography. Conversely, the devices that APCR-LPM is concerned with have memory in the order of kilobytes, cannot perform public key cryptography and cannot open multiple connections. It is important to consider these kinds of devices as sensors and IoT devices are employed in different environments and might collect sensitive information that should not be disclosed to compromised devices.

5.8.1 Methodology

Remote attestation is a well-known topic that has been researched for decades. Search phrases including 'lightweight attestation protocols for resource constrained devices using symmetric keys', 'attestation protocols using symmetric keys', 'attestation protocols for low-resource verifiers', 'attestation protocols for low-resource relying parties' were used in Google Scholar to narrow down the attestation literature to investigate. Only literature directly related to attestation and where constrained devices use symmetric encryption were studied. The terminology in some of the literature differ slightly from those in the attestation standard discussion in section 2.2.1. The attester in some cases is referred to as the 'prover' which is the entity trying to 'prove' its integrity.

5.8.2 Related Protocols

SlimIoT [3] is a lightweight attestation protocol that uses symmetric encryption and can be implemented on constrained devices. It is concerned with scalability and performs swarm attestation against physical and remote attacks. Swarm attestation is the process of attesting several devices in an IoT network at the same time. The protocol operates by performing a delayed disclosure of keys that a prover needs to check before generating its report. After receiving and verifying the keys, the prover aggregates all received reports from other provers and sends it to the verifier. In SlimIoT, the entities being attested are the constrained devices which is different from the APCR-LPM use case. In addition, the protocol operates under the assumption that all devices are loosely time-synchronized and can only detect a physical attack if the device stops responding. The protocol also involves having the constrained devices always listening to messages and receiving messages from other nodes in the network.

SCAPI [51] is another swarm-based attestation protocol that can detect software and physical attacks. It is similar to APCR-LPM as they both assume that the attesters contain a TEE to execute tamper-resistant tasks. SCAPI allows the verifier to communicate with only one device that collects reports from the remaining devices in the network and sends it to the verifier. SCAPI incurs code footprint and power consumption overhead because each device in the network performs neighbor discovery and exchanges channel keys with each discovered device. Similar to SlimIoT, both protocols assume some kind of clock synchronization between devices and that the constrained devices are those being attested.

One other lightweight attestation protocol is suggested by Jäger et al. [46]. Similar to APCR-LPM, this protocol is proposed for resource constrained devices and utilizes only symmetric cryptography. One major difference is that the protocol in Jäger et al. [46] is based on DICE (Device Identifier Composition Engine). DICE specifies an engine that allows devices with no special hardware, like TPMs, to perform attestation. One other difference is that the protocol assumes that the attester is the constrained device. Theoretically, the roles of attester and server can be switched with some minimal modification to the protocol so that the resource constrained device becomes the relying party. But the protocol is designed only for key attestation. Hence, it does not consider attestation metrics where additional claims about the device might need to be included.

AAoT [24] is an attestation protocol based on physical unclonable functions (PUFs). The most interesting feature of this protocol is that it provides mutual authentication for both verifier and prover. The first stage of the protocol allows the prover, which is a low-end device, to challenge the verifier to ensure the authenticity of the verifier. Therefore, while the roles here are reversed, the resource constrained device is still able to attest a device's integrity. However, the protocol assumes that the small device contains a PUF used to generate the symmetric keys between the prover and verifier. Therefore, the protocol is not adaptable to any small device as it requires specific hardware.

SIMPLE [1] is another protocol proposed to enable attestation for resource constrained devices. It relies only on nonces, counters and shared symmetric keys. In addition, neither the verifier or the prover needs special hardware or perform resource-intensive tasks. Therefore, the verifier can be a low-end device. The protocol assumes that validating attestation claims and generating attestation results is a task that can be done by the constrained devices, which might not be the case as more detailed appraisal policies could be used. Furthermore, the protocol can only perform software-based attestation as it relies on Security MicroVisor which is a software-based memory isolation technique. Generally, software-based attestation is the least secure form of attestation and is not viable for attestation performed over the network [28].

There are numerous other protocols like WISE [2], ERASMUS [18] and DARPA [42] that present attestation schemes for low-resource devices. Even though they consider constrained devices and only use symmetric keys, they are solving different problems than that APCR-LPM tries to solve. These protocols either deal with swarm attestation or present cases where the attester is the constrained device. In addition, the protocols cannot be extended to allow for a constrained verifier as they were too dependent on the assumption that the verifier is powerful and any change to this assumption will not allow the protocol to work. Table 5.2 summarizes the differences between the analyzed protocols and APCR-LPM.

The protocols found either abstract the concept of a relying party and assume that the verifier performs all the functionality, or require certain hardware in the constrained devices for the protocol to work. In real life, a relying party might be a network gateway or a sensor that would not be able to perform all the functionality that is requested of a verifier. In addition, it is more scalable to have a protocol that does not require special hardware. Therefore, APCR-LPM provides the first step to allowing even the smallest and resource constrained devices to perform attestation. The table compares the protocols against features like whether the protocol considers constrained relying parties, whether it requires special hardware like TPM or clock in the IoT device, whether it is hardware or software-based, and whether it allows for any kind of attestation or specific type of attestation, e.g. key attestation.

Protocol	**Constrained RP**	**Any IoT device**	**HW-based RA**	**Any attestation evidence**
SlimIoT	✗	✗	✓	✓
SCAPI	✗	✗	✓	✓
Rolling DICE	*	✓	**	✗
AAoT	✗	✗	✓	✗
SIMPLE	✗	✓	✗	✗
APCR-LPM	✓	✓	✓	✓

Table 5.2: Differences between APCR-LPM and other symmetric encryption-based RA protocols.

(*) No RP but theoretically the roles of attester and verifier can be reversed so that verifier is a constrained device.
(**) DICE is a software solution based on hardware properties present in IoT devices like immutable memory.

Chapter 6

Prototype Implementation

In order to prove the feasibility of the attestation result decoder and the APCR-LPM protocol, a prototype of a constrained relying party is implemented in this chapter. The prototype serves as a proof of concept that the work done in chapters 4 and 5 can be implemented on a constrained device. The constrained device used is Nordic Semiconductor's nRF5340 development kit [79] which has been set up to communicate with a laptop over Bluetooth. Since this thesis is mainly concerned with the behavior of the constrained relying party, only the communication between relying party and attester (messages 1 and 4) and the actions in the relying party (steps a and d) found in figure 5.3 are implemented.

6.1 Board Specifications

The nRF5340 board has the following specifications:

- Processor: 64 MHz Arm Cortex-M33 CPU
- Memory: 256 KB Flash + 64 KB RAM
- Interfaces: Bluetooth Low Energy, Bluetooth mesh, NFC, Matter, Thread and Zigbee

The nRF5340 supports Zephyr-RTOS (Real-Time Operating System) which is an operating system based on a small-footprint kernel designed for resource constrained devices. This OS supports a wide range of devices from small embedded environment sensors to IoT gateways. Zephyr can run on devices with 32KBs of RAM and has built-in support for multiple cryptographic libraries and a light-weight CBOR encoder and decoder called zcbor. These features make it a suitable choice for implementing the APCR-LPM protocol and decoding and validating attestation results.

6.2 Connection Setup

The connection between the relying party and attester in this prototype is IPv6 over BLE (Bluetooth Low Energy). This setup was used because it provides a simple way to communicate data to a device using common networking tools, such as telnet and curl, even though the underlying communication is Bluetooth. Zephyr provides a sample application [87] called IPSP (Internet Protocol Support Profile) which utilizes 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks). IPSP allows each node to have its own IPv6 address and communicate IPv6 packets over Bluetooth.

The Zephyr IPSP sample application is written in C and sets a static random Bluetooth MAC address for the nRF5340 board. After a Bluetooth connection is established, the application listens on port `4242` for any incoming UDP and TCP packets. The sample application also contains a simple echo server that is modified to communicate Cha and Res over the interface as well as decode the received attestation results R_A. It is worth noting that the adaptable and channel-independent nature of the APCR-LPM protocol allows it to be implemented on any communication channel including Bluetooth.

6.3 RP Implementation

The prototype setup is shown in figure 6.1. After the Bluetooth connection is established, the relying party generates a nonce and encrypts both the nonce c and id using K_V. The output of this ecryption, Cha, is sent over the channel. The application on the other end decrypts Cha to extract the values c and id. Then it encodes the attestation result object and sends the encrypted c, id, and R_A over the Bluetooth channel. The current prototype does not implement all the steps in the APCR-LPM protocol, it is mainly concerned with implementing the relying party application and incorporating the attestation result decoder. The technical aspects of the relying party implementation are described in the sections below.

6.3.1 Key Management

In the initial state, the relying party has two shared symmetric keys K_A and K_V. Currently, the keys are hardcoded static arrays of 16 bytes of random values. This was done to facilitate the usage of the keys. For more security, the small device should acquire the key from an external channel during key distribution and store it in a secure key storage component.

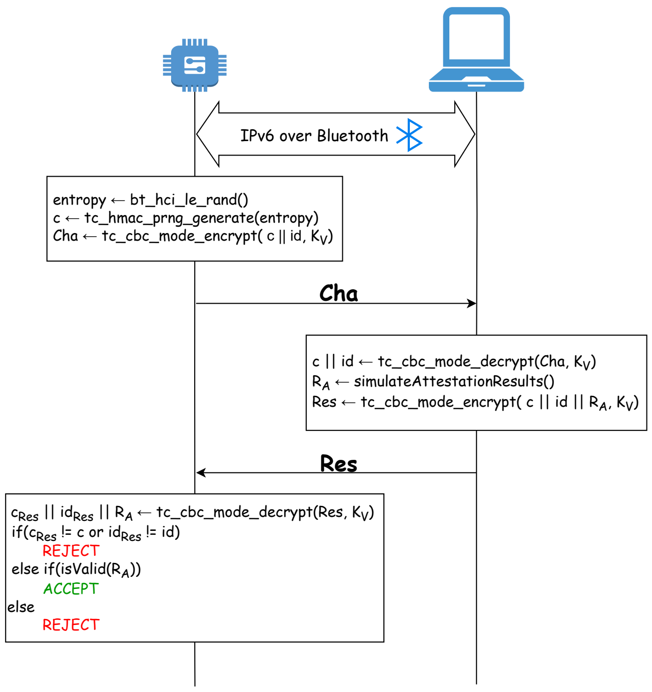

IPv6 over Bluetooth
entropy ← bt_hci_le_rand()
c ← tc_hmac_prng_generate(entropy)
Cha ← tc_cbc_mode_encrypt(c || id, K_V)
Cha
c || id ← tc_cbc_mode_decrypt(Cha, K_V)
R_A ← simulateAttestationResults()
Res ← tc_cbc_mode_encrypt(c || id || R_A, K_V)
Res
c_Res || id_Res || R_A ← tc_cbc_mode_decrypt(Res, K_V)
if(c_Res != c or id_Res != id)
REJECT
else if(isValid(R_A))
ACCEPT
else
REJECT

Figure 6.1: Prototype of relying party application.

6.3.2 Sending an Attestation Request

Zephyr comes with built-in support of several resource-efficient cryptographic libraries including TinyCrypt. The TinyCrypt library provides a set of cryptographic primitives and has a small code size and minimal dependency among primitives. This library was used for generating the nonce and performing basic encryption.

Generating a random value is the first step performed when creating Cha. TinyCrypt provides an implementation for HMAC-PRNG which is a pseudorandom number generator based on HMAC. The implementation requires both a personalization string and a source of entropy for the seed. The personalization string can be any value computed at initialization like the name of the board. The source of entropy used in this implementation comes from the BLE controller where the Zephyr Bluetooth module provides a function called `bt_hci_le_rand()` that returns a random number from the Bluetooth controller. There is no special hardware that is required and the source of entropy stems from the communication channel itself. The generated output c is concatenated with id to generate the data in Cha. Both c and id are 16 bytes in length, however that number can be easily adjusted depending on the security requirements of the message.

The encryption of c and id is done using the AES-128 CBC mode with a 128-bit key length provided by TinyCrypt. CBC mode requires a 128-bit initialization vector which can easily be computed using the HMAC-PRNG module. The generated output, Cha, is sent through the Bluetooth channel established with the attester.

6.3.3 Receiving an Attestation Result

The received result Res is decrypted with the same key used for Cha and the values c and id are extracted and compared to the sent ones. If either value does not match then the relying party rejects the message. Otherwise, the remaining bytes from the received packet is considered the attestation result object.

As discussed in section 4.5, CBOR could be a good fit for encoding attestation results in constrained environments. It had the smallest encoding size and least memory consumption when encoding attestation result objects. Zephyr provides support for Nordic Semiconductor's lightweight CBOR library called zcbor which is tailored for microcontrollers. This library was chosen instead of TinyCBOR because it is already provided as part of Zephyr and would not need any extra implementation or additional code to be added.

A simple EAR attestation result encoder and decoder is implemented using zcbor. The implementation is similar to the one discussed in section 4.3.4, however, only the required fields in the EAR object are considered. The decoded attestation results are then checked to ensure that the issued at time is not less than a particular value, that the verifier id is as expected and that the attester status is affirming. The implementation code for this prototype, which is based on Zephyr's IPSP sample, is found in appendix D. Currently, the application is designed to run the protocol only once. This behavior can easily be adjusted depending on the application requirements.

6.4 Results and Analysis

The relying party application was compared to the original IPSP application to determine the effects of the prototype on the size of the RAM and Flash. The relying party application imports the additional libraries TinyCrypt and zcbor, so it will definitely consume more resources. Table 6.1 shows a comparison between the original sample and the relying party application. The prototype uses approximately 6KB of extra Flash and few hundred bytes of additional RAM. Furthermore, to exclude Zephyr from the measurements, the applications' object files (file containing compiled assembly code) were analyzed and the size of the program was computed. The additional relying party code in the implementation (PRNG, encryption, decryption, zcbor decoding) required less than 2KB of extra code size.

Measurement [bytes]	**IPSP Sample**	**RP Application**
Total Flash	241320	247312 (+5992)
Total RAM	60280	60840 (+560)
Application Code Size	1792	3396 (+1604)

Table 6.1: Relying party application size measurements

The demonstrated prototype shows the feasibility of an attestation procedure on a resource constrained relying party. Using only the embedded device's available resources, a communication channel was established with the device acting as an attester, encrypted data was sent through the channel, and an EAR attestation result object was received, decoded, and evaluated. Furthermore, since the channel between relying party and attester is already established, any subsequent communication can be done over the provided channel with minimal additions to the current implementation.

It is important to note that this prototype is not a complete solution. There are several other factors to consider like how keys are stored on the device and how a solution can work for even more resource constrained devices. Additionally, the functionality of the attester and verifier was not implemented as the focus is on developing an attestation procedure suitable for constrained relying parties. Possible next steps include providing a key management solution and testing the application on various attestation result values.

Chapter 7

Discussion

The work in this thesis encompassed several fields related to attestation for constrained relying parties. To rephrase the problem statement 1.2, the main goal of this thesis is to understand and analyze whether constrained relying parties can benefit from attestation and whether there are attestation solutions that can be directly employed in constrained relying parties. The problem statement was broken down into research questions where each question targets a specific attestation area for constrained relying parties. The remainder of the chapter highlights the main insights and takeaways learned while answering the questions and provides a discussion of the answer to the problem statement.

RQ1: What are the important features that should be present when encoding attestation results and what are the existing formats that have those features?
Attestation results is an underdeveloped area with a large room for research. It is important to motivate why certain formats would work for attestation results, especially in a constrained environment, while other formats would not be suitable. To answer this question, different types of formats were surveyed and compared against a set of features. Those features were chosen because they are representatives of the security, usability and maintainability of a format. The features are:

- canonicality as non-canonical representation can lead to attacks
- maintainability as standardized formats have a community of people who work on updating and improving the format
- programming language independence as it allows formats to be used on different platforms and devices

- versatility as it ensures that the format can easily represent attestation results

The surveyed formats are popular in the industry and are specifically chosen to cover all types of serialization: binary vs text-based and schema-driven vs schema-less. The comparison found that ASN.1 DER, CBOR, and JSON are suitable formats for attestation results as certain specifications of those formats satisfy the desired features. However, the selected features are not enough when deciding on suitable formats for encoding. In a constrained environment with limited RAM, storage and buffer capacity, quantitative evaluation of the formats is also necessary. Such evaluations include message size, encoding speed, decoding speed, and memory consumption. Surveyed literature shows that these evaluations have not been performed between ASN.1 DER, JSON and CBOR. This lack of evaluation in the literature motivated the implementation and benchmarking done in chapter 4 which is needed to determine the suitable formats for encoding attestation results.

RQ2: What is the most suitable format in a constrained environment?
There are several factors that were considered when answering this question. These include encoding size, encoding speed, decoding speed, encoding memory consumption and decoding memory consumption. As the thesis focuses on constrained relying party receiving and decoding attestation results, the encoding size, decoding speed and decoding memory consumption are the important measurements to consider. The results of the measurements on ASN.1 DER, CBOR, and JSON led to several interesting findings, like that JSON provided the fastest decoding speed and that CBOR's decoding speed is an order of magnitude slower than ASN.1 DER and JSON. These findings refute the widely held assumption that binary serialization tends to be faster than text-based serialization. The implementation of the encoding and decoding play a big role in the performance of the formats as the libraries might have different optimization techniques. Furthermore, the type of data being encoded and decoded can impact the measurements. For example, JSON is more optimized for strings and CBOR is more optimized for integers, therefore, the kind and amount of data being encoded should be considered when choosing a format.

For decoding an attestation result object, while JSON was the fastest, it had the largest size which might not be suitable for a device with a limited transmission buffer or bandwidth. ASN.1 had additional overhead in creating and adapting the EAR schema and it was difficult to parse and check the optional fields in the decoding application. Overall, CBOR can be a good compromise between the three formats as it produces the smallest encoded messages and has less maintenance and development cost than ASN.1.

RQ3: Are there existing attestation protocols designed for constrained relying parties?
According to our knowledge and based on the research papers found in section 5.8.2, most protocols presented in the literature solve the problem of a powerful verifier and a constrained attester. In addition, no protocol was discovered that can be adapted to a constrained relying party that does not perform public-key cryptography. Some of the protocols use symmetric cryptography in the attestation protocol, but the roles between verifier and attester cannot be easily reversed as there are certain hardware and software requirements, like having a TPM or clock, imposed on the verifier and attester. Furthermore, most literature do not have the notion of a relying party and assume that the entity that performs verification is the same as the entity that needs the verification. This assumption is not always true as there are some real-world applications where a constrained device like a sensor, wearable or router might need the attestation result to allow access to a resource or to ensure that the attester is in an intended state. In this case, a separation of roles between relying party and verifier would be necessary.

RQ4: How can remote attestation protocol be resource-efficient while meeting the security requirements needed in an attestation solution?
Based on the result of RQ3, which found that there is no protocol in the literature that addresses the use case of a constrained relying party, the logical next step was to provide such a protocol. The APCR-LPM protocol is designed with the intention of allowing even the most resource constrained relying parties that cannot perform public-key cryptography to benefit from attestation. The protocol is lightweight and resource-efficient as it does not impose any specific hardware properties on the relying party device. The goals of the protocol include message integrity, freshness, and data origin authentication. These goals ensure that the relying party can detect whether the received attestation result has been modified, whether the attestation result is the response to the request it sent, and whether the result came from the intended verifier.

Formal analysis of the protocol using ProVerif has been conducted to ensure the security of the protocol in an environment where there is an active attacker. The analysis concludes that the desired security properties are achieved and that the attestation results are securely transmitted to the relying party. However, there are some aspects of the protocol that require further consideration. The protocol does not encrypt the attestation metrics collected by the attester. This leads to possible privacy concerns from the attester. Furthermore, since symmetric keys are used between the relying party and verifier and between the relying party and attester, non-repudiation cannot be achieved. The relying party and verifier are assumed to be trusted participants and the non-trusted participant (the attester) does use public-key cryptography which provides non-repudiation.

One final point to mention is that the attestation values, like the metrics and attestation results, influence the security of the protocol. The results are a function of the metrics, so even if the results are encrypted, they can still be derived because the metrics are public.

RQ5: Are the proposed solutions ensured to work in a constrained environment?

In order to prove that the solutions can in fact work in a constrained environment, a prototype of the protocol and the CBOR decoder are implemented on an embedded board. A Bluetooth connection is established with the attester and encrypted data is transmitted through the channel. There were no additional libraries required than what is provided by the embedded board's platform. Which demonstrates that the attestation solutions provided in this thesis can work for constrained devices. The board used in the prototype has around 60KB of RAM and 250KB of flash which makes it a slightly more powerful device than the intended constrained relying party. Nevertheless, the measurements show that the relying party application itself (generating nonce, encrypting challenge, sending challenge, receiving results, decrypting results, and decoding CBOR attestation result object) requires only about 3KB of RAM.

Problem statement: can small devices benefit from remote attestation to assess the security of more powerful devices?

The work done in this thesis showed that, while attestation for constrained devices is still a relatively unexplored field, there can be attestation solutions that allow constrained relying parties to securely request and receive attestation results, decode them, and finally evaluate the contents of the results. Based on the answers to the research questions, it was shown that resource-constrained devices can benefit from remote attestation.

Future Work

While the thesis provides an attestation solution for constrained relying parties, there is still room for further research. ASN.1 DER was chosen because it is a canonical representation of data that is widely deployed. There are other

ASN.1 canonical encoding rules like PER. While PER is not used frequently in the industry, a comparison between the performance of the two encoding rules could show whether PER can perform better than the other formats. Furthermore, as CBOR is substantially slower than the other formats, an implementation with another CBOR library can help in understanding whether this behavior is a result of the library used or it is format-specific. There are other metrics, like scalability and power consumption, that can also be studied when making the decision for an encoding format.

One other possible direction for the future work includes studying key distribution mechanisms that effectively allow the symmetric keys used in APCR-LPM to be shared. The protocol model can also be modified by having different representations of the metrics and analyzing whether the different representations affect the security of the protocol. Additionally, the prototype can also be improved by providing a complete implementation of the attester and verifier and running several scenarios where the attestation results have different values.

Chapter 8

Conclusion

Remote attestation is the process of evaluating the hardware and software properties of a remote entity. An attestation procedure has three active parties: 1) the attester: the entity that collects cryptographically secure evidence about itself to prove its trustworthiness 2) the verifier: the entity that receives and attests the evidence to produce attestation results 3) the relying party: the entity that receives the attestation results and determines the attester's trustworthiness.

Not only was there a lack of an unbiased assessment that examines the strengths and weaknesses of formats capable of encoding attestation results, but also there was a gap in the research for attestation protocols for constrained relying parties. Furthermore, there were no practical attestation solutions for constrained relying parties.

This thesis is a starting point for filling this gap in the literature and providing an attestation solution for constrained relying parties. First, it analyzed formats for encoding attestation results. Then, it proposed an attestation protocol for constrained relying parties. Finally, the thesis presented a proof-of-concept implementation of the protocol and the attestation result decoder on an embedded device. The summarized results of the work are as follows:

- The analysis of different types of encoding formats concluded that ASN.1 DER, CBOR, and JSON are the most suitable for encoding attestation results. Those formats were then compared in a quantitative manner by implementing attestation result encoders and decoders. The comparison showed that JSON had the fastest decoding time but the largest message size and CBOR had the smallest message size but the slowest decoding time. ASN.1 DER comes a close second to CBOR in encoding size and to JSON in decoding speed, but it has an additional implementation cost of creating and maintaining the ASN.1

schema. Based on these results, CBOR was deemed a good compromise between the three.

- The proposed attestation protocol for constrained relying parties was based on symmetric cryptography and did not impose any hardware requirements on the relying party device. The protocol was designed to allow the relying party to make an attestation request and receive an attestation result in a secure manner. The protocol was formally analyzed with ProVerif to ensure that the protocol's security requirements are fulfilled. The formal analysis led to the discovery of an attack that happens when an attester in the environment is compromised. Based on this discovery, the protocol was improved and no other attack was found by ProVerif.

- To provide a proof of concept of the solutions presented in this thesis, a prototype of a relying party application was implemented. A Bluetooth connection was created between a Linux laptop and Nordic Semiconductor's nRF5340 board. All the relying party steps found in the protocol were implemented with Zephyr RTOS. The application took around 250KB of Flash and 60KB of RAM with further room for improvement.

The results validate that a constrained device, acting as a relying party, can attest a more powerful device, e.g., a smart phone. Due to the popularity of IoT, the importance of this setting is likely to increase.

Based on the work conducted in this thesis and the obtained results, it is evident that there is room for further research in this area. There are other security requirements, such as confidentiality of the attestation results, that are not targeted by the protocol but could be important in other use cases. The key distribution mechanism for the shared symmetric keys the relying party uses is another area to explore. As for the prototype, it can be further tested by deploying it on IoT devices in a real-world scenario and analyzing how the protocol influences the behavior of those devices.

Bibliography

[1] Ammar, M., Crispo, B., and Tsudik, G. SIMPLE: A Remote Attestation Approach for Resource-constrained IoT devices. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)* (2020), pp. 247–258.

[2] Ammar, M., Washha, M., and Crispo, B. WISE: Lightweight Intelligent Swarm Attestation Scheme for IoT (The Verifier's Perspective). In *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)* (2018), pp. 1–8.

[3] Ammar, M., Washha, M., Ramabhadran, G. S., and Crispo, B. SlimIoT: Scalable Lightweight Attestation Protocol for the Internet of Things. In *2018 IEEE Conference on Dependable and Secure Computing (DSC)* (2018), pp. 1–8.

[4] Ankergård, S. F. J. J., Dushku, E., and Dragoni, N. State-of-the-art software-based remote attestation: Opportunities and open issues for Internet of Things. *Sensors 21*, 5 (2021), 1598.

[5] Apple. Secure Enclave Processor. `https://support.apple.com/en-gb/guide/security/sec59b0b31ff/web`, 2021. Last Accessed: 26-04-2023.

[6] Apple. Apple DeviceCheck: Assessing Fraud Risk. `https://developer.apple.com/documentation/devicecheck/assessing_fraud_risk`, 2022. Last Accessed: 12-06-2023.

[7] Apple. Managed Device Attestation for Apple devices. `https://support.apple.com/en-gb/guide/deployment/dep28afbde6a/web`, 2023. Last Accessed: 24-04-2023.

[8] ARM. EAR (EAT Attesation Result). `https://github.com/veraison/ear`. Last Accessed: 08-02-2023.

[9] ARM. VERificAtIon of atteStatiON , Veraison. `https://github.com/veraison`. Last Accessed: 08-02-2023.

[10] Baier, C., and Katoen, J.-P. *Principles of Model Checking*, vol. 26202649. MIT press, 01 2008.

[11] Bhargavan, K., Blanchet, B., and Kobeissi, N. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In *2017 IEEE Symposium on Security and Privacy (SP)* (2017), pp. 483–502.

[12] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and Pan, W. RFC 9334 Remote ATtestation procedureS (RATS) Architecture, 2023.

[13] Birkholz, H., Vigano, C., and Bormann, C. Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures. RFC 8610, RFC Editor, June 2019.

[14] Bittner, S., and Hinze, A. On the benefits of non-canonical filtering in publish/subscribe systems. In *25th IEEE International Conference on Distributed Computing Systems Workshops* (2005), pp. 451–457.

[15] Blanchet, B., Smyth, B., Cheval, V., and Sylvestre, M. Proverif 2.00: automatic cryptographic protocol verifier, user manual and tutorial. *Version from* (2018), 05–16.

[16] Bormann, C., Ersue, M., and Keranen, A. Terminology for constrained-node networks. RFC 7228, RFC Editor, May 2014. `http://www.rfc-editor.org/rfc/rfc7228.txt`.

[17] Bormann, C., and Hoffman, P. Concise binary object representation (cbor). RFC 7049, RFC Editor, October 2013.

[18] Carpent, X., Tsudik, G., and Rattanavipanon, N. ERASMUS: Efficient remote attestation via self-measurement for unattended settings. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2018), pp. 1191–1194.

[19] Chen, J., and Guo, R. Stack and Heap Memory. `https://courses.engr.illinois.edu/cs225/fa2022/resources/stack-heap/`, 2022. Last Accessed: 25-05-2023.

[20] Christidis, K. Notes on protocol buffers and deterministic serialization (or lack thereof). `https://gist.github.com/kchristidis/39c8b310fd9da43d515c4394c3cd9510`, 2017. Last Accessed: 03-05-2023.

[21] Database, N. V. ECDSA signature validation vulnerability by accepting wrong ASN.1 encoding in jsrsasign. `https://github.com/advisories/GHSA-p8c3-7rj8-q963`, 2020. Last Accessed: 03-05-2023.

[22] Database, S. V. Denial of Service (DoS) Affecting com.google.protobuf:protobuf-java package. `https://security.snyk.io/vuln/SNYK-JAVA-COMGOOGLEPROTOBUF-2331703`, 2022. Last Accessed: 28-06-2023.

[23] Dolev, D., and Yao, A. On the security of public key protocols. *IEEE Transactions on Information Theory 29*, 2 (1983), 198–208.

[24] Feng, W., Qin, Y., Zhao, S., and Feng, D. AAoT: Lightweight attestation and authentication of low-resource things in IoT and CPS. *Computer Networks 134* (2018), 167–182.

[25] Fossati, T., and Howard, P. C-EAR: A C library that implements a minimalist EAR (EAT Attestation Result) decoder and verifier. `https://github.com/veraison/c-ear`, 2023. Last Accessed: 25-05-2023.

[26] Fossati, T., Voit, E., and Trofimov, S. EAT Attestation Results. `https://thomas-fossati.github.io/draft-ear/draft-fv-rats-ear.html`. Last Accessed: 09-02-2023.

[27] Foundation, P. S. pickle — Python object serialization. `https://docs.python.org/3/library/pickle.html`, 2023. Last Accessed: 27-04-2023.

[28] Francillon, A., Nguyen, Q., Rasmussen, K. B., and Tsudik, G. A minimalist approach to remote attestation. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2014), pp. 1–6.

[29] Gil, B., and Trezentos, P. Impacts of data interchange formats on energy consumption and performance in smartphones. In *Proceedings of the 2011 Workshop on Open Source and Design of Communication* (New York, NY, USA, 2011), OSDOC '11, Association for Computing Machinery, p. 1–6.

[30] Gong, L., Ellison, G., and Dageforde, M. *Inside Java 2 Platform Security: Architecture, API Design, and Implementation.* Addison-Wesley Professional, 2003.

[31] Google. FlatBuffers Documentation. `https://flatbuffers.dev/index.html#flatbuffers_overview`. Last Accessed: 27-04-2023.

[32] Google. Protocol Buffers Documentation. `https://protobuf.dev/overview/`. Last Accessed: 27-04-2023.

[33] Google. Overview of the Play Integrity API. `https://developer.android.com/google/play/integrity/overview`, 2023. Last Accessed: 24-04-2023.

[34] Group, T. C. Trusted Platform Module Library Part 1: Architecture. `https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf`, 2019. Last Accessed: 05-06-2023.

[35] Group, T. C. Trusted Platform Module (TPM). `https://trustedcomputinggroup.org/work-groups/trusted-platform-module/`, 2019. Last Accessed: 05-06-2023.

[36] Group, T. C. DICE Attestation Architecture. `https://trustedcomputinggroup.org/wp-content/uploads/TCG_DICE_Attestation_Architecture_r22_02dec2020.pdf`, 2020. Last Accessed: 25-04-2023.

[37] Gueron, S. A memory encryption engine suitable for general purpose processors. Cryptology ePrint Archive, Paper 2016/204, 2016. `https://eprint.iacr.org/2016/204`.

[38] Gunn, L. J., Asokan, N., Ekberg, J.-E., Liljestrand, H., Nayani, V., Nyman, T., et al. Hardware platform security for mobile devices. *Foundations and Trends® in Privacy and Security 3*, 3-4 (2022), 214–394.

[39] Halak, B., Yilmaz, Y., and Shiu, D. Comparative analysis of energy costs of asymmetric vs symmetric encryption-based security applications. *IEEE Access 10* (2022), 76707–76719.

[40] Huawei. OpenHarmony System Types. `https://github.com/openharmony/docs/blob/master/en/device-dev/device-dev-guide.md`, 2022. Last Accessed: 31-05-2023.

[41] Huawei. SysIntegrity API. `https://developer.huawei.com/consumer/en/doc/development/Security-Guides/dysintegritydevelopment-0000001050156331`, 2022. Last Accessed: 24-04-2023.

[42] Ibrahim, A., Sadeghi, A.-R., Tsudik, G., and Zeitouni, S. Darpa: Device attestation resilient to physical attacks. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks* (2016), pp. 171–182.

[43] Intel. TinyCBOR 0.5.2 API. `https://intel.github.io/tinycbor/current/`, 2015. Last Accessed: 25-05-2023.

[44] Intel Trust Domain Extensions. `https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-final9-17.pdf`. Last Accessed: 08-02-2023.

[45] Jäger, L., Lorych, D., and Eckel, M. A Resilient Network Node for the Industrial Internet of Things. In *Proceedings of the 17th International Conference on Availability, Reliability and Security* (2022), pp. 1–10.

[46] Jäger, L., Petri, R., and Fuchs, A. Rolling dice: Lightweight remote attestation for cots iot hardware. In *Proceedings of the 12th International Conference on Availability, Reliability and Security* (2017), pp. 1–8.

[47] Speed test loading the different formats CBOR, MessagePack, BSON, JSON. `https://github.com/nlohmann/json/discussions/2581`, 2021. Last Accessed: 27-06-2023.

[48] Kernighan, B. W., and Ritchie, D. M. *The C Programming Language*, 2nd ed. Prentice Hall Professional Technical Reference, 1988.

[49] Khare, K. JSON vs Protocol Buffers vs FlatBuffers. `https://codeburst.io/json-vs-protocol-buffers-vs-flatbuffers-a4247f8bda6f`, 2018. Last Accessed: 04-05-2023.

[50] Kobeissi, N., Bhargavan, K., and Blanchet, B. Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)* (2017), pp. 435–450.

[51] Kohnhäuser, F., Büscher, N., Gabmeyer, S., and Katzenbeisser, S. Scapi: a scalable attestation protocol to detect software

and physical attacks. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (2017), pp. 75–86.

[52] Larmouth, J. *ASN. 1 complete.* Morgan Kaufmann, 2000.

[53] Lehtinen, P. Jansson Documentation. `https://jansson.readthedocs.io/en/latest/`, 2023. Last Accessed: 25-05-2023.

[54] Lemire, D. What is the space overhead of Base64 encoding? `https://lemire.me/blog/2019/01/30/what-is-the-space-overhead-of-base64-encoding/`, 2019. Last Accessed: 29-06-2023.

[55] Lohmann, N. Speed test loading the different formats cbor, msgpack, bson, json. `https://github.com/nlohmann/json/discussions/2581#discussioncomment-282855`, 2021. Last Accessed: 03-05-2023.

[56] Martin, A. C. The ten-page introduction to trusted computing. `https://ora.ox.ac.uk/objects/uuid:a4a7ae67-7b2a-4516-801d-9379d613bab4`, 2008. Last Accessed: 13-02-2023.

[57] Mesfin, G., Ghinea, G., Grønli, T.-M., and Younas, M. Web Service Composition on Smartphones: The Challenges and a Survey of Solutions. In *Mobile Web and Intelligent Information Systems* (Cham, 2018), M. Younas, I. Awan, G. Ghinea, and M. Catalan Cid, Eds., Springer International Publishing, pp. 126–141.

[58] [MS-DHA]: Device Health Attestation Protocol. `https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-dha/18db0839-9a75-434e-9c7c-efb4b767007b`. Last Accessed: 08-02-2023.

[59] Morana, M., and Nusbaum, S. Input Validation Vulnerabilities, Encoded Attack Vectors and Mitigations. `https://owasp.org/www-pdf-archive/Encoded_Attacks_Threats_Countermeasures_9_30_08.pdf`, 2008. Last Accessed: 03-05-2023.

[60] Nethercote, N., and Seward, J. Valgrind: A framework for heavyweight dynamic binary instrumentation. In *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA, 2007), PLDI '07, Association for Computing Machinery, p. 89–100.

[61] Niazi, M. A. Machine-readable vs. Human-readable Data. `https://control.com/technical-articles/machine-readable-vs-human-readable-data/`, 2021. Last Accessed: 26-04-2023.

[62] Niemi, A., Nayani, V., Moustafa, M., and Ekberg, J.-E. Platform attestation in consumer devices. In *2023 33rd Conference of Open Innovations Association (FRUCT)* (2023), pp. 198–209.

[63] Niemi, A., Sovio, S., and Ekberg, J.-E. Towards Interoperable Enclave Attestation: Learnings from Decades of Academic Work. In *2022 31st Conference of Open Innovations Association (FRUCT)* (2022), IEEE, pp. 189–200.

[64] Nurseitov, N., Paulson, M., Reynolds, R., and Izurieta, C. Comparison of JSON and XML Data Interchange Formats: A Case Study. In *ISCA International Conference on Computer Applications in Industry and Engineering* (2009).

[65] O'Donoghue, J. Towards lightweight and interoperable trust models: The entity attestation token. In *Living in the Internet of Things (IoT 2019)* (2019), pp. 1–11.

[66] Patterson, S. Node-TinyCbor: A Node.js Addon for the tinyCBOR Library. `http://controlenvy-foss.gitlab.io/node-tinycbor/`, 2017. Last Accessed: 27-06-2023.

[67] Petersen, B., Bindner, H., You, S., and Poulsen, B. Smart grid serialization comparison: Comparision of serialization for distributed control in the context of the internet of things. In *2017 Computing Conference* (2017), pp. 1339–1346.

[68] Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., and Vrgoč, D. Foundations of JSON Schema. In *Proceedings of the 25th International Conference on World Wide Web* (Republic and Canton of Geneva, CHE, 2016), WWW '16, International World Wide Web Conferences Steering Committee, p. 263–273.

[69] Point, T. Java Serialization. `https://www.tutorialspoint.com/java/java_serialization.htm`. Last Accessed: 27-04-2023.

[70] Proos, D. P., and Carlsson, N. Performance Comparison of Messaging Protocols and Serialization Formats for Digital Twins in IoV. In *2020 IFIP Networking Conference (Networking)* (2020), pp. 10–18.

[71] Ray, E. *Learning XML*. Learning Series. O'Reilly Media, 2003.

[72] Rose, S., Borchert, O., Mitchell, S., and Connelly, S. Zero trust architecture. Tech. rep., National Institute of Standards and Technology, 2020.

[73] Rundgren, A., Jordan, B., and Erdtman, S. Json canonicalization scheme (jcs). RFC 8785, RFC Editor, June 2020.

[74] Samsung. Knox Attestation API reference (V3.0). `https://docs.samsungknox.com/devref/knox-attestation/index.htm`, 2022. Last Accessed: 12-06-2023.

[75] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and Adams, C. X.509 internet public key infrastructure online certificate status protocol - ocsp. RFC 6960, RFC Editor, June 2013. `http://www.rfc-editor.org/rfc/rfc6960.txt`.

[76] Scarlata, V., Johnson, S., Beaney, J., and Zmijewski, P. Supporting third party attestation for Intel® SGX with Intel® data center attestation primitives. *White paper* (2018).

[77] Segall, A. *Trusted Platform Modules: Why, when and how to use them.* Institution of Engineering and Technology, London, United Kingdom, 2017.

[78] Selvaraj, N. Python Pickle Tutorial: Object Serialization. `https://www.datacamp.com/tutorial/pickle-python-tutorial`, 2023. DataCamp Tutorial Last Accessed: 16-06-2023.

[79] Semiconductor, N. nRF5340 DK. `https://www.nordicsemi.com/Products/Development-hardware/nRF5340-DK`. Last Accessed: 25-06-2023.

[80] Stumpf, F., Tafreschi, O., Röder, P., Eckert, C., et al. A robust integrity reporting protocol for remote attestation. In *Proceedings of the Workshop on Advances in Trusted Computing (WATC)* (2006), p. 65.

[81] Sumaray, A., and Makki, S. K. A Comparison of Data Serialization Formats for Optimal Efficiency on a Mobile Platform. *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication* (2012).

[82] Trust@HsH. Trusted Computing and Trusted Network Connect in a Nutshell. Hochschule Hannover, `http://trust.f4.hs-hannover.de/2008/11/21/trusted-computing-and-trusted-network-connect-in-a-nutshell.html`. Last Accessed: 08-02-2023.

[83] Viotti, J. C., and Kinderkhedia, M. A Survey of JSON-compatible Binary Serialization Specifications. *CoRR abs/2201.02089* (2022).

[84] Voit, E., Birkholz, H., Hardjono, T., Fossati, T., and Scarlata, V. Attestation results for secure interactions. Tech. rep., Internet-Draft draft-voit-rats-attestation-results-02. Internet Engineering . . . , 2021.

[85] von Helden, J., Bente, I., Vieweg, J., and Hellmann, B. Trusted network connect (TNC). *European Trusted Infrastructure Summer School,.(Referenced on page.)* (2009).

[86] Yergeau, F. UTF-8, a transformation format of ISO 10646. STD 63, RFC Editor, November 2003. `http://www.rfc-editor.org/rfc/rfc3629.txt`.

[87] Zephyr. Bluetooth: IPSP Sample. `https://docs.zephyrproject.org/latest/samples/bluetooth/ipsp/README.html`, 2023. Last Accessed: 25-06-2023.

Appendix A

Benchmarking Inputs

A.1 Input 2 (Two Attesters)

```
{
   "eat_profile": "tag:github.com,2023:veraison/ear",
   "iat": 1666529300,
   "ear.verifier-id": {
     "developer": "https://veraison-project.org",
     "build": "vts 0.0.1"
   },
   "ear.raw-evidence": "3q2-7w",
   "submods": {
     "CCA Platform": {
       "ear.status": "affirming",
       "ear.trustworthiness-vector": {
         "instance-identity": 2,
         "configuration": 2,
         "executables": 3,
         "file-system": 2,
         "hardware": 2,
         "runtime-opaque": 2,
         "storage-opaque": 2,
         "sourced-data": 2
       },
       "ear.appraisal-policy-id":
         "https://veraison.example/policy/1/60a0068d"
     },
     "CCA Realm": {
       "ear.status": "affirming",
       "ear.trustworthiness-vector": {
          "instance-identity": 2,
          "configuration": 2,
          "executables": 3,
          "file-system": 2,
          "hardware": 2,
          "runtime-opaque": 3,
          "storage-opaque": 2,
          "sourced-data": 3
        },
        "ear.appraisal-policy-id":
          "https://veraison.example/policy/1/60b0068d"
      }
    }
}
```

A.2 Input 3 (Raw Evidence)

```
{
    "eat_profile": "tag:github.com,2023:veraison/ear",
    "iat": 1666529300,
    "ear.verifier-id": {
        "developer": "https://veraison-project.org",
        "build": "vts 0.0.1"
    },
    "ear.raw-evidence":
            "aGV5dGhlcmUsdGhpc2lzbXl0ZXN0Zm9ydGhlZWZmZWN0b2ZyYXdl
            dmlkZW5jZW9udGhlZGF0YWFuZHNpemVvZmVuY29kaW5nLHdob2V2Z
            XJ0b29rdGhldGltZXRvZGVjb2RldGhpcyx0aGFua3lvdSx5b3VoYX
            ZlbXlhZG1pcmF0aW9uYW5kcGxlYXNlY29udGFjdG1lZm9yYWNoYXR
            pZnlvdWxpa2Usa3R4YnlleG94b3hveG8=",
    "submods": {
        "CCA Platform": {
            "ear.status": "affirming",
            "ear.trustworthiness-vector": {
                "instance-identity": 2,
                "configuration": 2,
                "executables": 3,
                "file-system": 2,
                "hardware": 2,
                "runtime-opaque": 2,
                "storage-opaque": 2,
                "sourced-data": 2
            },
            "ear.appraisal-policy-id":
            "https://veraison.example/policy/1/60a0068d"
        }
    }
}
```

The line breaks used in the `ear.raw-evidence` attribute are used for readability. The actual string is represented as a single long line since JSON does not support multiline strings without the new line character `\n`.

Appendix B

Benchmarking Code

The code used for benchmarking can be found in GitHub. Please note that the ASN.1 encoder and decoder were implemented with HTLS, a closed-source library. Therefore, the code cannot be published.

Appendix C

ProVerif Models

All the ProVerif codes can also be found in GitHub.

C.1 Model of APCR-LPM

```
type nonce.

fun validateEvidence(bitstring): bitstring.

(*Symmetric Encryption*)
type key.
fun senc(bitstring, key): bitstring.
reduc forall m: bitstring, k: key; sdec(senc(m,k),k) = m.

(* Digital Signatures *)
type skey.
type pkey.
fun pk(skey): pkey.
fun sign(bitstring, skey) : bitstring.
reduc forall m: bitstring, k: skey; checksign(sign(m, k), pk(k)) = m.

(* Hashing *)
fun hash(key) : bitstring.

free network: channel.

(* events mark important stages reached by the protocol but
do not otherwise affect behavior *)

event relyingPartyAccepts(key, key, nonce, bitstring).
event relyingPartyBegins(key, key, nonce).
event attesterBegins(pkey, key, bitstring, bitstring).
event verifierAccepts(pkey, key, bitstring, nonce, bitstring).


(* Injective correspondence assertions capture the one-to-one relationship
and are denoted: query x1 : t1, . . . , xn : tn ; inj-event
(e(M1, . . . , Mj )) ==> inj-event (e'(N1, . . . , Nk)) .
Informally, this correspondence asserts that, for each occurrence of
the event e(M1, . . . , Mj ), there is a distinct earlier occurrence
of the event e'(N1, . . . , Nk). *)

query  PK_a: pkey, K_a: key, K_v: key, R_a: bitstring, M_a: bitstring,
c:nonce, h:bitstring, Cha:bitstring;
inj-event(relyingPartyAccepts(K_v, K_a, c, R_a))
==> inj-event(relyingPartyBegins(K_v, K_a, c)) &&
    inj-event(attesterBegins(PK_a, K_a, M_a, Cha)) &&
    inj-event(verifierAccepts(PK_a, K_v, M_a, c, h)) &&
    h = hash(K_a) &&
    R_a = validateEvidence(M_a) &&
    Cha = senc((c, h), K_v)
.

query  K_v: key, K_a: key, R_a: bitstring, c:nonce;
attacker(R_a) && event(relyingPartyAccepts(K_v, K_a, c, R_a))
==> false
.


let relyingPartyRP(K_a: key, K_v: key) =
  new c: nonce;
  let h = hash(K_a) in
  let Cha = senc((c,h), K_v) in
  event relyingPartyBegins(K_v, K_a, c);
  out(network, Cha);
  in(network, Res: bitstring);
  let (R_a: bitstring, =c , =h) = sdec(Res, K_v) in
  event relyingPartyAccepts(K_v, K_a, c, R_a);
  0.


let attesterA(K_a: key, SK_a: skey, PK_a: pkey) =
  in(network, Cha: bitstring);
  new M_a: bitstring;
  let h = hash(K_a) in
  let Ev = sign((M_a, h, Cha) , SK_a) in
  event attesterBegins(PK_a, K_a, M_a, Cha);
  out(network, Ev);
  (* in(network, Res: bitstring);
  out(network, Res); *)
  0.


let verifierV(K_v: key, PK_a: pkey) =
  in(network, Ev: bitstring);
  let (M_a: bitstring, h_att: bitstring, Cha: bitstring) = checksign(Ev, PK_a) in
  let (c: nonce, h_rp: bitstring) = sdec(Cha, K_v) in
  if(h_att = h_rp) then
  let R_a = validateEvidence(M_a) in
  event verifierAccepts(PK_a, K_v, M_a, c, h_rp);
  let Res = senc((R_a, c, h_rp), K_v) in
  out(network, Res);
  0.


process (* main method *)
  new K_a: key;
  new K_v: key;
  new SK_a: skey;
  let PK_a = pk(SK_a) in out(network, PK_a); (* Attacker knows PK_a *)
  (
      (!relyingPartyRP(K_a, K_v)) |
      (!attesterA(K_a, SK_a, PK_a)) |
      (!verifierV(K_v, PK_a))
  )
```

C.2 Model of Attack on APCR-LPM

```
type nonce.

fun validateEvidence(bitstring): bitstring.


(*Symmetric Encryption*)
type key.
```

```
fun senc(bitstring, key): bitstring.
reduc forall m: bitstring, k: key; sdec(senc(m,k),k) = m.

(* Digital Signatures *)
type skey.
type pkey.
fun pk(skey): pkey.
fun sign(bitstring, skey) : bitstring.
reduc forall m: bitstring, k: skey; getmess(sign(m, k)) = m.
reduc forall m: bitstring, k: skey; checksign(sign(m, k), pk(k)) = m.

(* Hashing *)
fun hash(key) : bitstring.


free network: channel.

(* Authentication queries *)

event relyingPartyAccepts(key, key, nonce, bitstring).
event relyingPartyBegins(key, key, nonce).
event attesterBegins(pkey, key, bitstring, bitstring).
event verifierAccepts(pkey, key, bitstring, nonce, bitstring).
event attesterCorrupted(key).



query  PK_a: pkey, K_a: key, K_v: key, R_a: bitstring, M_a: bitstring,
c:nonce, h:bitstring, Cha:bitstring;
inj-event(relyingPartyAccepts(K_v, K_a, c, R_a))
==>
    event(attesterCorrupted(K_a)) ||
    (inj-event(relyingPartyBegins(K_v, K_a, c)) &&
    inj-event(attesterBegins(PK_a, K_a, M_a, Cha)) &&
    inj-event(verifierAccepts(PK_a, K_v, M_a, c, h)) &&
    h = hash(K_a) &&
    R_a = validateEvidence(M_a) &&
    Cha = senc((c, h), K_v)
    )
.

query  K_a: key, K_v: key, R_a: bitstring, c:nonce;
attacker(R_a) && event(relyingPartyAccepts(K_v, K_a, c, R_a))
```

```
==> false
.


let relyingPartyRP(K_a: key, K_v: key) =
  new c: nonce;
  let h = hash(K_a) in
  let Cha = senc((c,h), K_v) in
  event relyingPartyBegins(K_v, K_a, c);
  out(network, Cha);
  in(network, Res: bitstring);
  let (R_a: bitstring, =c , =h) = sdec(Res, K_v) in
  event relyingPartyAccepts(K_v, K_a, c, R_a);
  (* out (network, senc(secretR_a, bitstring_to_key(R_a))). *)
  0.


let attesterA(K_a: key, SK_a: skey, PK_a: pkey) =
  in(network, Cha: bitstring);
  new M_a: bitstring;
  let h = hash(K_a) in (* TEE is abstracted  *)
  let Ev = sign((M_a, h, Cha) , SK_a) in
  event attesterBegins(PK_a, K_a, M_a, Cha);
  out(network, Ev);
  (* in(network, Res: bitstring);
  out(network, Res); *)
  0.

let verifierV(K_v: key, PK_a: pkey) =
  in(network, Ev: bitstring);
  let (M_a: bitstring, h_att: bitstring, Cha: bitstring) = checksign(Ev, PK_a) in
  let (c: nonce, h_rp: bitstring) = sdec(Cha, K_v) in
  if (h_att = h_rp) then
  let R_a = validateEvidence(M_a) in
  event verifierAccepts(PK_a, K_v, M_a, c, h_rp);
  let Res = senc((R_a, c, h_rp), K_v) in
  out(network, Res);
  0.


process (* main method *)
  new K_a: key;
  (* let h = hash(K_a) in out(network, h); *)
  new K_v: key;
  new SK_a: skey;
  let PK_a = pk(SK_a) in out(network, PK_a); (* Attacker knows PK_a *)
  new K_a2: key;
  event attesterCorrupted(K_a2);
  new SK_a2: skey;
  out(network, SK_a2);
  let PK_a2 = pk(SK_a2) in out(network, PK_a2);
  (* 1 RP, 2 A and 1 V *)
  (
      (!relyingPartyRP(K_a, K_v)) |
      (!attesterA(K_a, SK_a, PK_a)) |
      (!verifierV(K_v, PK_a)) |


      (!relyingPartyRP(K_a2, K_v)) |
      (!attesterA(K_a2, SK_a2, PK_a2)) |
      (!verifierV(K_v, PK_a2))
  )
```

C.3 Model of Improved Protocol

```
type nonce.

fun validateEvidence(bitstring): bitstring.


(*Symmetric Encryption*)
type key.
fun senc(bitstring, key): bitstring.
reduc forall m: bitstring, k: key; sdec(senc(m,k),k) = m.

(* Digital Signatures *)
type skey.
type pkey.
fun pk(skey): pkey.
fun sign(bitstring, skey) : bitstring.
reduc forall m: bitstring, k: skey; getmess(sign(m, k)) = m.
reduc forall m: bitstring, k: skey; checksign(sign(m, k), pk(k)) = m.

(* Hashing *)
fun hash(bitstring) : bitstring.
fun key_to_bitstring(key): bitstring [data,typeConverter].


free network: channel.

(* Security queries *)

event relyingPartyAccepts(key, bitstring, nonce, bitstring).
event relyingPartyBegins(key, nonce, bitstring).
event attesterBegins(pkey, bitstring, bitstring, bitstring).
event verifierAccepts(pkey, key, bitstring, bitstring, nonce).
event attesterCorrupted(key).

query  PK_a: pkey, K_a, K_v: key, R_a: bitstring, c:nonce,
id: bitstring, h: bitstring, M: bitstring, Cha: bitstring;
inj-event(relyingPartyAccepts(K_v, R_a, c, id))
==>
    event(attesterCorrupted(K_a)) ||
    (inj-event(relyingPartyBegins(K_v, c, id)) &&
    inj-event(attesterBegins(PK_a, h, M, Cha)) &&
    inj-event(verifierAccepts(PK_a, K_v, M, id, c)) &&
    R_a = validateEvidence(M) &&
    id = hash((h, PK_a)) &&
    Cha = senc((c,id), K_v))
.

(* query  K_v: key, R_a: bitstring, c:nonce, h: bitstring;
event(relyingPartyAccepts(K_v, R_a, c, h))
(* ==> false *)
.  *)

(* Secrecy queries *)

query  PK_a: pkey, K_v: key, R_a: bitstring, c:nonce,
h: bitstring;
attacker(R_a) && event(relyingPartyAccepts(K_v, R_a, c, h))
==> false
.

let relyingPartyRP(K_a: key, K_v: key, id: bitstring) =
  new c: nonce;
  (* let h = hash(K_a) in *)
  let Cha = senc((c,id), K_v) in
  event relyingPartyBegins(K_v, c, id);
  out(network, Cha);
  in(network, Res: bitstring);
  let (R_a: bitstring, =c , =id) = sdec(Res, K_v) in
  event relyingPartyAccepts(K_v, R_a, c, id);
  (* out (network, senc(secretR_a, bitstring_to_key(R_a))). *)
  0.

let attesterA(K_a: key, SK_a: skey, PK_a: pkey) =
  in(network, Cha: bitstring);
  new M: bitstring;
  let h = hash(key_to_bitstring(K_a)) in (* TEE key attestation is abstracted *)
  let Ev = sign((M, h, Cha) , SK_a) in
  event attesterBegins(PK_a, h, M, Cha);
  out(network, Ev);
  (* in(network, Res: bitstring);
  out(network, Res); *)
  0.

let verifierV(K_v: key, PK_a: pkey) =
  in(network, Ev: bitstring);
  let (M: bitstring, h: bitstring, Cha: bitstring) = checksign(Ev, PK_a) in
  let (c: nonce, id_cha: bitstring) = sdec(Cha, K_v) in
  let id = hash((h, PK_a)) in
  if (id = id_cha) then
  let R_a = validateEvidence(M) in
  event verifierAccepts(PK_a, K_v, M, id, c);
  let Res = senc((R_a, c, id), K_v) in
  out(network, Res);
  0.

process (* main method *)
  new K_a: key;
  new K_v: key;
  new SK_a: skey;
  let PK_a = pk(SK_a) in out(network, PK_a); (* Attacker knows PK_a *)
  let id = hash((hash(key_to_bitstring(K_a)), PK_a)) in out(network, id);
  new K_a2: key;
  event attesterCorrupted(K_a2);
  new SK_a2: skey;
  out(network, SK_a2);
  let PK_a2 = pk(SK_a2) in out(network, PK_a2);
  let id2 = hash((hash(key_to_bitstring(K_a2)), PK_a2)) in out(network, id2);
```

```
(* 1 RP, 2 A and 1 V *)
(
    (!relyingPartyRP(K_a, K_v, id)) |
    (!attesterA(K_a, SK_a, PK_a)) |
    (!verifierV(K_v, PK_a)) |

    (!relyingPartyRP(K_a2, K_v, id2)) |
    (!attesterA(K_a2, SK_a2, PK_a2)) |
    (!verifierV(K_v, PK_a2))
)
```

Appendix D

Prototype Implementation Code

The code presented in this section is an extension of Zephyr's IPSP bluetooth sample [87]. The code can be found in GitHub.