UNIVERSIDADE D
COIMBRA

Antonio Alexandre Marques de Sá

IOT DEVICE FUNCTIONALITY ATTESTATION MECHANISMS

Dissertation in the context of the Master in Informatics Security, advised by Professor Bruno Sousa and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September of 2022

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

Antonio Alexandre Marques de Sá

IoT Device Functionality Attestation Mechanisms

Dissertation in the context of the Master in Informatics Security, advised by Prof. Bruno Sousa and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September of 2022

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Antonio Alexandre Marques de Sá

Mecanismos de atestação de funcionalidades de dispositivos IoT

Dissertação no âmbito do Mestrado em Segurança Informática, orientada pelo Professor Doutor Bruno Sousa e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2022

Acknowledgements

I would like to thank professor Bruno Sousa, my advisor, who welcomed me, even late, and trust that we can work together.

In memory of my mother and brother, victims of complications from our war against SARS-Cov19.

Abstract

Attestation is a mechanism that is employed to verify the authenticity and integrity of the other(s) part(s), i.e., in hardware and/or software of a device. The remote attestation is the activity of verifying the authenticity and integrity of a target that provides evidence to a verifier over a network that should be accepted or denied as a result of this process. Classic authorization relies in the information provided by a device and gives permission for a specific operation. The attestation adds a new a layer of information, not only we need to know who the device is, but we also need to know if it is in good standing (i.e. performing according to its design) before authorization. This document proposes the use of the Passport model, using the Challenge/Response development based on the architecture described by the IETF working group RATS - Remote Attestation Procedures Architecture [1]. The elaborated Proof-of-Concept is designed and evaluated using docker containers and TPM software simulation.

Keywords

Remote attestation, IoT, challenge-response, Passport Model, framework, RATS, IETF,

Resumo

Atestação é o mecanismo pelo qual um parceiro usa para verificar a autenticidade e integridade da (s) outra (s) parte (s), ou seja, no hardware e / ou software de um dispositivo. Já a atestação remota é a atividade de verificação da autenticidade e integridade de um alvo que fornece evidências a um verificador em uma rede que devem ser aceites ou negadas como resultado desse processo. A autorização clássica está relacionada com o acreditar no que um dispositivo afirma quem é e, no dar permissões para uma operação específica. O atestado adiciona um novo mecanismo; não só precisamos saber quem é o dispositivo, como também precisamos de saber se está em boas condições (i.e. a funcionar conforme foi desenhado) antes de conceder autorização. Este documento propõe a utilização do modelo Passport, utilizando o desenvolvimento Challenge/Response baseado na arquitetura descrita pelo grupo de trabalho da IETF RATS - Remote Attestation Procedures Architecture [1]. A prova de conceito foi elaborada e avaliada usandos *containers* docker e simulação de software TPM.

Palavras-Chave

Atestação remota, IoT, desafio-resposta, Modelo Passaporte, estruturas, RATS, IETF

Contents

Acronyms

ASCII American Standard Code for Information Interchange.
CA Cetrification Authority.
CBOR Concise Binary Object Representation.
CHARRA-PM CHAllenge-Response based Remote Attestation - Passport Model.
CoAP Constrained Application Protocol.
DTLS Datagram Transport Layer.
EK Endorsement Key.
IETF Internet Engineering Task Force.
IoT Internet of Things.
JSON JavaScript Object Notation.
M2M Machine to Machine.
MITM Man in the Midlle.
MQTT Message Queuing Telemetry Transport.
PCR Platform Configuration Registers.
PoC Proof of Concept.
PSK Pre-Shared Key.
RA Remote Attestation.
RATS Remote Atestation Procedures.
RPK Raw Public Keys.
RTS Root of Trust Storage.
SARA Secure Asunchronous Remote Attestation for IoT Systems.
SHA Secure Hash Algorithm.
SHeLa Scalable Heterogeneous Layered Attestation.
SSL Secure Sockets Layer.
TLS Transport Layer Security.
TPM Trusted Platform Module.
UDP User Datagram Protocol.
URI Uniform Resource Identifier.
ZigBee Zonal Intercommunication Global-standard.

List of Figures

List of Tables

Listings

Chapter 1

Introduction

The wide range of low-cost embedded devices with the ability to command other devices on a network has grown enormously. The network to which these devices belong is called the Internet of Things (IoT), which at the year 2021 registered a growth of 9% compared to the previous year (12.3 billion dollars) according to IoT Analytic [2].

Such devices have low cost, low consumption and easy integration. Their application is possible in several layers of the industry, such as the medical, automotive, smart cities, smart-home sectors.

IoT devices generally do not have built-in security features or the ability to install or update software. That was a limitation without major problems when installed in isolated networks and not connected to the outside world. However, as technology advances, the IoT interconnectivity with the internet's enterprise network grows [3]. Another point of observation is the manufacture of these devices on a large scale, where manufacturers want to produce quickly and at a lower cost without safety, security concerns. On the other hand, we have the installation of devices without proper precautions. Due to this, IoT systems are becoming a favourite target for cyber attacks [4]. That said, we can use Remote Attestation (RA), which allows us to find which device is not following the network's policies, or not functioning as intended and was possibly the target of an attack.

In 2019, the Internet Engineering Task Force (IETF) started a working group [5] to discuss the creation of standards for remote attestation models and procedures (models for Remote Atestation Procedures (RATS)). IETF is one of the organizations responsible for internet technology standards [6].

Not all IoT devices have computation power to perform complex calculations or have security features implemented hardware (e.g. TPM explained in section 2.1). That opens a gap in this field of research. In addition, RA protocols are randomly executed at unpredictable times, causing the tester's regular operation to be interrupted for the attestation process. Such protocol features are not tolerated on devices that perform critical functions or operate under intermittent connectivity. Disruption of service is a challenge to be improved in the remote attestation.

In this work, We will study aspects of recent proposals for attestation of IoT de-

vices based on Asynchronous attestation - Secure Asunchronous Remote Attestation for IoT Systems (SARA) [7] and based on scalable systems, such as Scalable Heterogeneous Layered Attestation (SHeLa) [8]. Both try to solve synchronisation problems, heterogeneity and device overload, considering static and dynamic network problems.

1.1 Objectives

In this work, the objective is a remote attestation model for IoT environments. To achieve such goal, I've looked for a realistic implementation of a remote attestation model in which it was possible to add security functionality to increase the reliability and integrity of information in IoT environments. Make use of the result produced by the Challenge/Response model, allowing this work to contribute to future investigations where tests and analyses are necessary for validating Remote Attestation models.

1.2 Contributions/Achievements

The main contribution of this work, includes a new proof of concept based on the RATS Passport Model [1] - the CHAllenge-Response based Remote Attestation - Passport Model (CHARRA-PM). This contribution is an approach from the attestation Challenge/Response Model with the addition of a Relying Party responsible to grant authorization for IoT devices. This contribution is available as open source code on GitHub [9]

Within the context of a European project - ARCADIAN-IoT - where the University of Coimbra and the IPN are involved, the IPN engaged us during the study and development phase to assist them in reviewing the ideas put forward in this dissertation. Several meetings were held where the application of CHARRA and the implementation of the Passport Model were presented. In addition, some chat and video sessions were made with a project member to explain interactions using RATS concepts in more detail.

1.3 Structure

This document is organized as follows:

- Chapter 2 presents the main technologies to support the implementation.
- Chapter 3 documents the state of the art and standardization on remote attestation.
- Chapter 4 the research methodology.
- Chapter 5 presents the implementation of the CHARRA-PM.
- Chapter 6 The Proof of concept is explained.
- Chapter 7 concludes the document.

Chapter 2

Background

This chapter will present the technologies used to implementation of CHARRA and CHARRA-PM. The Trusted Platform Module (TPM) 2.0 simulator and other protocols that used also to guarantee security and integrity in many implementations, either in IoT devices with low capacity and in Windows 11 devices with TPM for secure boot.

2.1 Trusted Platform Model 2.0 - TPM 2.0

The Trust Platform Module (TPM) is an international standard (ISO/IEC 11889-1:2015) [10] that defines the architectural elements of a low-cost, secure crypto-processor, enabling trust in general computing platforms.

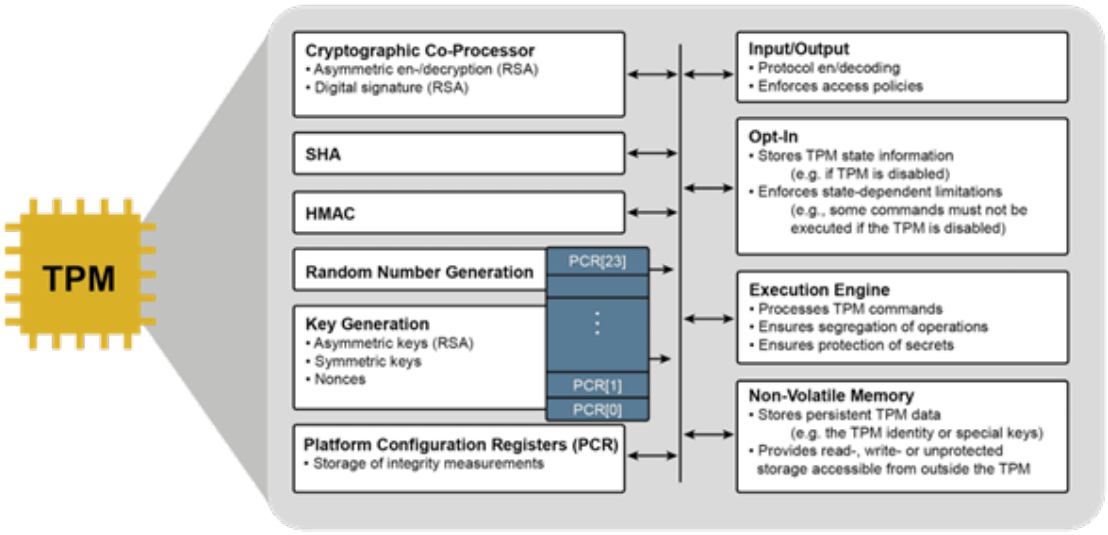

Figure 2.1: Trusted Platform Module (TPM)

The implementation consists of a physical device connected to the motherboard that communicates with the main processor (CPU) over a slow, low-bandwidth channel, providing secure, hardware-based, tamper-proof key generation and en-

crypted key storage. Malicious software cannot tamper with security functions as it implements physical security features to prevent tampering. The use of TPM technology has several significant benefits, including [11; 12]:

- Create, maintain and restrict the use of cryptographic keys.
- Device authentication using the TPM's unique RSA key, which was written to the device by the manufacturer.
- Allows guaranteeing the platform's integrity by storing unique values in its registers.

The most adopted use by the industry is the measurement of system integrity and the creation and use of keys. Let us imagine that during a system's boot process, the boot code (the firmware and operating system components) can be measured and recorded in the TPM. These measurements can be used as evidence of system boot health. The TPM first version (1.2), published around 2005, used Secure Hash Algorithm (SHA) version 1.0 as the hashing algorithm. However, after the attacks against SHA-1, the working group started the development of version 2.0 of the TPM, which was completely redesigned, allowing the use of more secure hashing algorithms (SHA-256, SHA-384 and SHA- 512) [11; 13]. Figure 2.1 presents the general components of TPM [14]

The TPM cannot interrupt normal CPU execution because it is a passive device. To function needs to be fed with data from the host (such as information measured during a boot process) on which it is installed and accessed by programs specifically built to interact with the TPM. This information is recorded in the Platform Configuration Registers (PCR) Registers, explained in the PCR section 2.1.2. TPM instruction execution is single-thread; in other words, it executes one instruction at a time. Each statement must wait for the executing statement to complete.

2.1.1 Endorsement Certificate

Each TPM chip has a unique and secret Endorsement Key (EK), usually certified by a trusted Cetrification Authority (CA). Figure 2.2 shows the flow of creating and writing the EK in the TPM. The flow starts during the fabrication of the TPM chip. First, a self-signed root certificate is created by an internal CA (it may already exist). Subsequently, the processes of creating the public and private keys, where the private key is stored on the TPM chip, and the creation of the endorsement certificate, signed by a proprietary key known only by the manufacturer and not stored on the TPM chip. Finally, the endorsement certificate is engraved on the TPM chip.

2.1.2 Platform Configuration Registers - PCR

Platform Configuration Registers (PCR), are a set of fixed-size registers in the TPM. Unlike regular registers, PCR values cannot be set to arbitrary values. The only operation they support (other than Read) is Extend, which is explained in

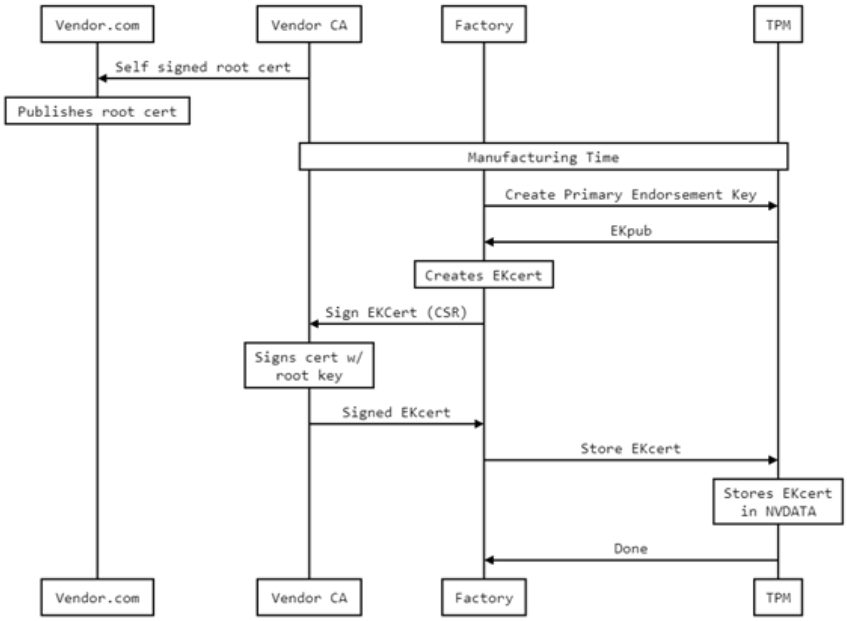

Figure 2.2: Endorsement Key)

the following paragraphs. Platform Configuration Registers (PCRs) are particular TPM objects that can only be modified or written by the hash extension mechanism. The new input value is concatenated with the existing value in the PCR and hashed. The new hash value now replaces the old value, which means that even though it is a single location, the final value reflects a history of all hash extensions. PCRs are organized into banks (up to 32 PCRs), one for each specific hash algorithm family of SHA1 and SHA2. The Trusted Computing Group has a specification indicating which part of the system software should be extended for PCR indices. [15]. Robustness mechanisms prevent physical tampering with these making the TPM a Root of Trust Storage (RTS).

```
// Get initial PCR value.
01:  var pcr = ByteArrayToForgeBuffer(sim.GetPcr(1));
// Extend PCR1 with the SHA256 digest of 'Hello'.
02:  var rc = app.ExtendPcr(/*pcr=*/1, 'Hello');
03:  assert(rc == TPM2_RC_SUCCESS, 'ExtendPcr failed');
// Verify PCR extend semantics.
04:  var measurement = new forge.sha256.create()
05:  measurement.update('Hello');
// PCR := Hash(PCR || M)
06:  var extend = new forge.sha256.create()
07:  extend.update(pcr.data);
08:  extend.update(measurement.digest().data);
09:  pcr = extend.digest();
10:  var actual = ByteArrayToForgeBuffer(sim.GetPcr(1));
11:  print('Expected: ', forge.util.bytesToHex(pcr.data));
12:  print('Actual   : ', forge.util.bytesToHex(actual.data));
13:  assert(_.isEqual(pcr.data, actual.data) == true, 'PCR value
   does not match');
14:  print('OK');
```

Listing 2.1: Extending a PCR

The code listing 2.1 is an example in C demonstrating the PCR1 "extend" process, as shown in [16]. Explaining in more detail the PCR extension process, we have:

- In line 01, the value stored in PCR1 that is retrieved and placed in a PCR variable. On lines 02 and 03, the PCR extension is performed by the 'ExtendPcr' function, which does all the work, applying hash 256 to the "Hello" value, and then concatenating it with the previous hash value stored in PCR1. Finally calculates the hash of the concatenated value and stores it in PCR1.
- In the block of lines 04 to 13, the step-by-step is presented in case we do not use the 'ExtendPcr' function.
- In line 04, hash 256 of the 'Hello' value is created and stored in the variable 'measurement' in line 05.
- On line 06, a variable called 'extend' is created that will receive the values of the concatenated hashes. In line 07, the value retrieved in line 01 (which is already a hash) is inserted. On line 08, the word 'Hello' hash is created and inserted into the variable 'extend'. Finally, the final hash (over the previously concatenated hash data) is created and stored in PCR1.
- From line 10 to the end, a check is made between the value calculated (PCR) and the value retrieved (current) from the PCR. If the comparison is correct, the word "Ok" is displayed.

The size stored in each PCR is defined by the associated hash algorithm, which can be updated per the policy defined for the PCR. Version 1.2 only supports SHA-1, and version 2.0 supports SHA-2 algorithms in addition to this.

In a typical trust boot, operations measure the first stage firmware (reading and hashing the binary code), using the result to extend the first PCR. Each subsequent stage of the process measures the next stage before executing it. Eventually, when the kernel is measured and initialized, the TPM will have a set of PCRs describing the software it has run. If these PCRs do not contain the expected values, someone has tampered and the system is unreliable.

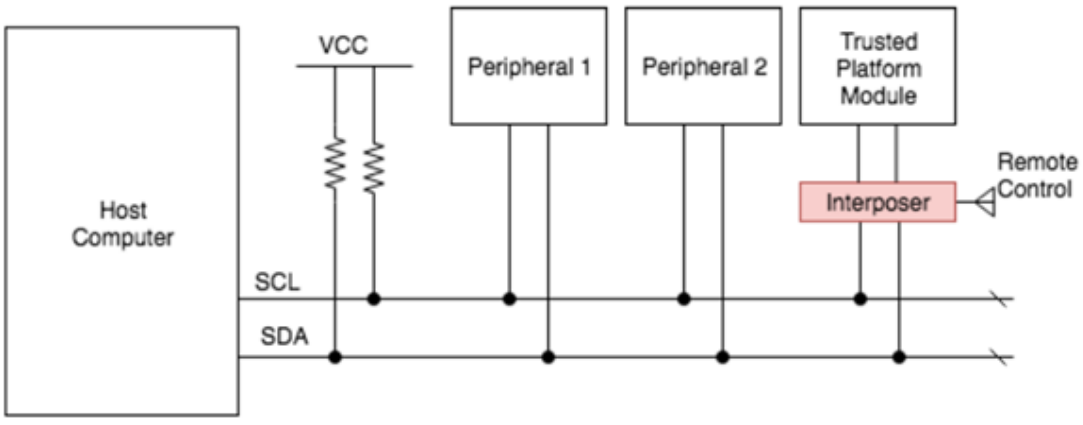

Figure 2.3: High Level block diagram of interposer on an IC2 bus

The TPM is a device accessed through a device driver, so any process that wants to query the TPM must do so through the kernel. There is a potential problem: if the kernel has been corrupted, it can lie about the values stored in the PCRs, thus nullifying the entire measurement process. The article [17] presents possible attacks using the IC2 Serial Bus responsible for the communication interface of the TPM with the device hardware:

"The tool, which we call TPM Genie, has the ability to intercept and modify all traffic that is sent over the bus and is intended to assist in vulnerabilities research of discrete TPMs and the host-side drivers that interact with them. Leveraging TPM Genie, we show how an interposer device can easily spoof measurements stored in Platform Configuration Register (PCR) banks. This act of PCR spoofing serves to carry out a variety of attacks against the main functions of a TPM that depend on the integrity of the PCR, namely: metered boot, remote attestation, and sealed storage" [17].

It should be noted that such attacks come from techniques at the circuit level (hardware), as shown in figure 2.3. Software attacks, which manipulate the data extracted from the TPM, can be avoided with time, nonces and hash control techniques, such as replay attacks.

The operation called "quoting" ensures that this information is reliable. This operation provides current PCR values signed by a private factory key on the TPM device. When these values are requested, a random nonce is inserted in the request payload (digest) of the signed response, preventing the "quotes" from being reused by malicious software.

Thus, a system that wants to verify a "quote" uses the TPM public key and ensures that the "nonce" matches what was provided. If all goes well, it can be assumed that the PCR values provided by the TPM are healthy.

It is possible to read the PCR values directly, but this process does not guarantee that the values returned are intact; that is, there is the possibility of an attacker carrying out a Man in the Midlle (MITM) attack, tampering with the values of one or several PCRs. Reading by "quote" is the safest, and a random nonce must be generated for each new request.

Note: For the scope addressed by this dissertation, we will use version 2.0 of TPM, with values starting at zero, as it is not a boot process but a proof of concept that uses a TMP emulator by Software [18; 19]

2.2 Constrained Application Protocol - CoAP

The Constrained Application Protocol (CoAP) [20] is a specialized web transfer protocol for use in constrained nodes (e.g. low-power, 8-bit microcontrollers with small amounts of ROM and RAM). The protocol is designed for Machine to Machine (M2M) applications such as smart energy, building automation, and IoT in general. It is a protocol implemented at the Application layer, which interacts with the lower layers.

CoAP provides a request/response interaction model between application end-

points, supports integrated service and resource discovery, and includes key web concepts such as URIs and Internet Media Types. CoAP is designed to easily interface with HTTP for web integration, meeting specialized requirements such as multicast support, very low overhead, and simplicity for constrained environments. The purpose of CoAP is not to blindly compress HTTP [21], but to perform a common subset with HTTP in an optimized fashion for M2M applications. CoAP has the following main characteristics [20]:

- Web protocol that meets M2M requirements in constrained environments;
- UDP binding [22] with optional reliability that supports unicast and multicast requests;
- Asynchronous message exchanges;
- Low header overhead and parsing complexity;
- URI and Content-Type support.
- Simple proxy and caching features;
- Stateless HTTP mapping, allowing building proxies providing uniform access to CoAP resources via HTTP;
- Security link to Datagram Transport Layer Security (DTLS) [23].

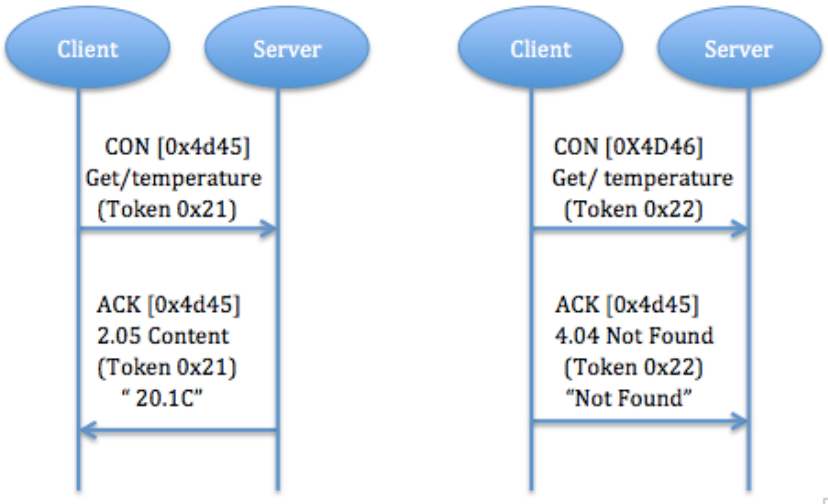

Figure 2.4: CoAP Message Diagram

The CoAP interaction model is similar to the HTTP client/server model. However, when there are M2M interactions, the CoAP implementation will act as a client and server. Equivalent to HTTP, a CoAP request can be as follows: A request is sent by a client, requesting a particular action (using a Method Code) on a resource (identified by a Uniform Resource Identifier (URI)) on a server. Unlike HTTP, CoAP handles these exchanges asynchronously in a datagram-oriented transport such as UDP, using a messaging layer that supports optional reliability. CoAP defines four types of messages [20]: Confirmable, Non-committable, Acknowledgment, Reset. The Method Codes and Response Codes included in

some of these messages cause them to carry requests or responses.

A piggybacked Response is included right in a CoAP Acknowledgment (ACK) message that is sent to acknowledge receipt of the Request for this Response. The Piggybacked Response is illustrated in figure 2.4.

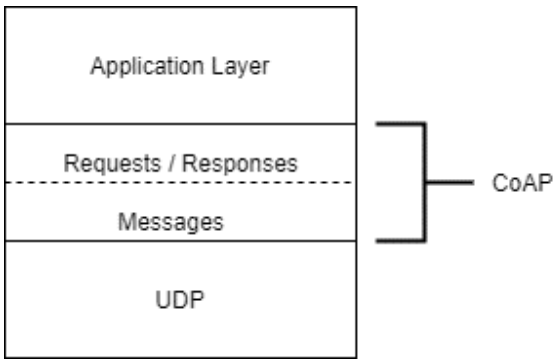

Figure 2.5: Abstract Layering of CoAP

We can logically think of CoAP as using a two-tier approach; a CoAP messaging layer used to handle UDP and request/response interactions using Method and Response Codes, as represented in figure 2.5.

Ver	T	TKL	code	Message ID
Token (if any, TKL bytes) ...				
Options (if any) ...				
Payload (if any) ...				

Table 2.1: CoAP Message Format

Based on the exchange of compact messages that, by default are transmitted over User Datagram Protocol (UDP), each CoAP message occupies the data section of a UDP datagram. [20]. The header has a fixed size of 4 bytes, a variable-length token value, the CoAP options, and the Payload as described in Table 2.1.

The fields that make up the CoAP message header are composed of, as per RFC 7252 [20]:

Version (ver) : CoAP version number.

Type (T) : Indicates the message type: Confirmable (0), Non-confirmable (1), Acknowledgment (2), or Reset (3).

Token Length (TKL): Indicates the length of the Token field (variable). Lengths between 9 and 15 are reserved and must not be sent. It should be processed as a message format error.

Code: It is a field divided into a 3-bit class (most significant bits) and a 5-bit detail (least significant bits), documented as "c.dd", where "c" is a digit from 0 to 7 to represent the class, and "dd" are two digits from 00 to 31 to represent the detail.

The class can indicate a request (0), a success response (2), a client error response (4), or a server error response (5). All other class values are reserved.

The code (0.00) indicates an Empty message. In case of request, the Code field indicates the Request Method; in case of response, a Response Code. [20] sections 3, 5 and 12.1.

Message ID: Used to detect message duplication and to match Commitable/Unconfirmable type messages.

CoAP proposes using Datagram Transport Layer (DTLS) as the security protocol for data encryption, integrity protection, and authentication that will be explained in section 2.3.

2.2.1 CoAP FETCH Method

Like HTTP, CoAP uses the REST model. In other words, REST can be defined as servers making resources available in a URL, and clients access those resources using methods such as GET, PUT, POST and DELETE. However, three new methods were introduced to CoAP [24]: FETCH, PATCH and IPATH.

The **FETCH** method used to perform the equivalent of a GET with a request body;

The twin methods **PATCH** and **iPATCH**, to perform partial modifications of an existing CoAP resource (similar to HTTP PATCH).

The FETCH method is used in our development because it provides a solution that spans the usage range of GET and POST. Like the POST method, the input on the FETCH method is passed within the request payload and not as part of the request URI.

2.3 Datagram Transport Layer Security - DTLS

The DTLS protocol [23] was created to be responsible for providing secure communication over a datagram network (UDP). The DTLS design is to be similar to Transport Layer Security (TLS) as possible. This fact allowed the reuse of existing code and infrastructure.

The primary purpose of DTLS is to build "TLS over datagram transport". As is well known, datagram transport does not require or provide reliable or orderly delivery of data. Due to the delay-sensitive nature of the transported data (media streaming, Internet telephony, online gaming, and even IoT applications), the behaviour of applications that use UDP will remain the same when the DTLS protocol is used for security. DTLS will not compensate for lost or reordered data traffic.

The following are some reasons why TLS cannot be used directly over UDP: [23]

- TLS relies on an implicit sequence number in the records.

  DTLS solves this problem by adding sequence numbers to records.

- The TLS handshake is a block-step cryptographic protocol. Messages must be transmitted and received in a defined order, and any other order is considered an error.

  The DTLS handshake includes message sequence numbers, allowing the reassembly of fragmented messages and in-order delivery in the event of reordering or loss of datagrams.

- Handshake messages are larger than a single datagram.

  DTLS adds fields to handshake messages to support fragmentation and reassembly.

DTLS has mechanisms to handle issues, such as:

- **Packet Loss**: DTLS uses a simple retransmission timer to handle packet loss.
- **Reordering**: each handshake message receives a specific sequence number. If the message is not in the expected sequence, it is queued and processed when all previous messages have been received.
- **Fragmentation**: DTLS handshake messages can be fragmented into multiple DTLS records. Each record is intended to fit into a single UDP datagram containing the offset and fragment length allowing the recipient to reassemble the fragmented handshake message once it has received all the bytes of the message.
- **Repetition Detection**: DTLS maintains a bitmap window of received records. It is optional. Without detailing the process, it should be noted that DTLS supports Raw Public Keys (RPK) [25] and Pre-Shared Key (PSK) [26].
- **Pre-Shared keys**: In this case, a secret is shared (or already known) that needs to be available to the device as well as to the other party in the communication before the process starts.
- **Raw Public Keys**: The parties involved have a public/private key stored, and the public key must be known to authenticate the other party.

The C library, Mbed-TLS [27], implements cryptographic primitives, handling X.509 certificates, Secure Sockets Layer (SSL)/TLS and DTLS protocols. Its code is extremely small, which makes it suitable for embedded systems. Implementation are detailed in section 5.

2.4 Concise Binary Object Representation - CBOR

The Concise Binary Object Representation (CBOR) [28] is a data format whose design goals include the possibility of very small code and message size. It is a format similar to JavaScript Object Notation (JSON) [29] with binary encoding, whose goals are:

1. Unambiguous encoding of the most common data formats of Internet standards; i.e. arrays, maps, numbers and trees

2. Compact code for encoder or decoder (to work on restricted devices);
3. No schema description is needed;
4. Fairly compact serialization;
5. Applicability to restricted and high-volume applications;
6. Good conversion to JSON;
7. Extensibility and Backward Compatibility, the format can be extended while maintaining backward compatibility with older decoders.

An example of very simple use of encoding and decoding can be :

```
data = "coimbra"
encoded = CBOR.encode(data) // 67436F696D627261
data = CBOR.decode(encoded) // coimbra
```

Listing 2.2: Coding CBOR

The encoded text "coimbra" is represented as an American Standard Code for Information Interchange (ASCII) value 67436F696D627261, where:

- 67 represents the octal of the number seven (see 2.6 - the total size of the "Coimbra" string is seven characters;
- 436F696D627261 - The table 2.2 help us to understand the conversion. Also, using the figure 2.6, we can convert the hexadecimal to the individual character of the word "Coimbra".

Position	1	2	3	4	5	6	7
Hexa	43	6F	69	6D	62	72	61
Text	C	o	i	m	b	r	a

Table 2.2: COBR Text encoding

We will not go into the details of CBOR representation. This format is very similar to the JSON data model, using binary data serialization (according to the data type), which significantly reduces the size, and time for encoding and sending the data over the network. The detail of data representation (major types) can be found in the appendices of RFC8949 [28].

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Figure 2.6: ASCII Table found in [30]

Chapter 3

State of the Art and Standardization

This section is intended to describe recent works in remote attestation. All proposals came from a simple challenge-response protocol, as shown in figure 3.1, which is basically the attestation of a single Verifier and a single Prover. [31]

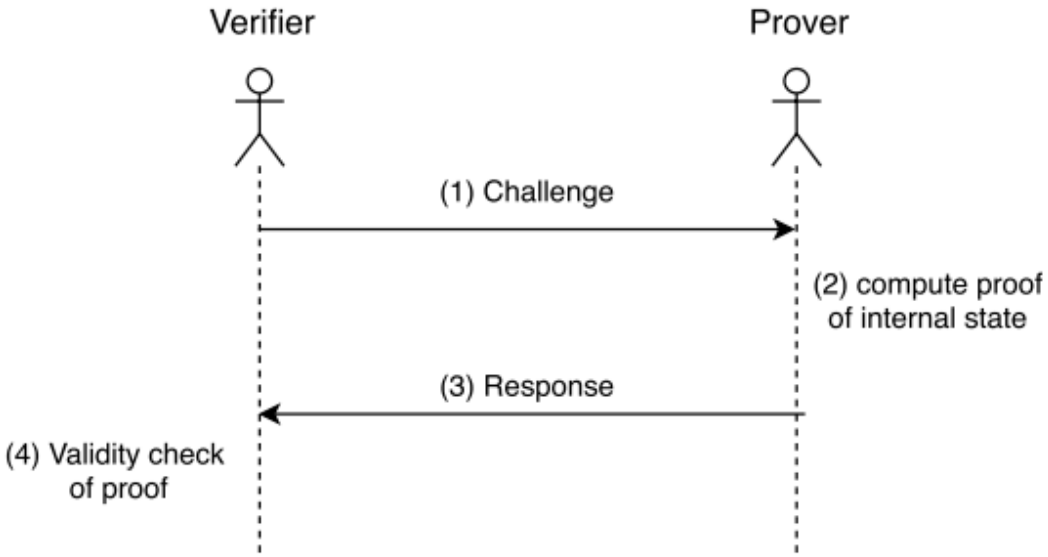

Figure 3.1: Challenge-response protocol [31]

1. Initially, the Verifier generates a challenge.
2. The Verifier sends the challenge to the Prover.
3. The Prover calculates a proof of state and forwards it to the Verifier.
4. Ultimately, the Verifier validates the response to the challenge and can, for example, agree or not with the state of that device.

There is no industry standard for remote attestation. There are IETF studies that intend to recommend the use of a framework for remote attestation. Some researchers have proposed different types of remote attestation that have been improved over the years published on websites and conferences, but they are not

used as standards. Two of the most current and the IETF proposal are documented in section 3.2.1.

3.1 Standardized Attestation Mechanisms

There is an agreement on the four general approaches that remote attestation should be categorized: implementation-based: hardware-based, software-based, and hybrid remote attestation. In addition, we must rely on security architectures as they provide the foundation for many remote attestation protocols:

- Security Architectures - It should provide reliable computing for low-cost IoT devices, for example, key protection.
- Hardware-based Attestation - Chips or physical hardware modules that guarantee the reliability of the remote attestation.
- Software-based Attestation - The program runs in memory to validate the state of the system software. Communication with other devices is generally not allowed while the attestation protocol runs, and communication is limited to one-hop distance.
- The Hybrid Remote Attestation - The goal is to combine the security of hardware attestation with the lowest cost of software attestation and thereby solve software-based remote attestation problems.
- Swarm attestation - Instead of describing the attestation of a single device, it describes the attestation of many devices.

3.2 Attestation Mechanisms in the literature

An untrusted remote prover's status can be determined by a remote verifier using the security service known as Remote Attestation. However, remote attestation includes extensions like code updates, device resets, and runtime state attestation. Additionally, the demand for device swarm attestation grows with the popularity of networked IoT devices. In this section, the RATS Model will be presented in more detail since it is the focus of this work. The section 3.3 analyze the Proof of Concept (PoC) of RATS Challenge/Response model.

The main disadvantage of RATS, is that it is still under specification and is not yet a formal RFC document. Thus, other approaches may provide more adherence given their stability. The Scalable embedded device attestation (SEDA) stood out among the studies, which defines the term swarm as a set of 8 devices, where there is the possibility of different hardware and software [32]. However, SEDA is the basis of improvement for other more recent research, such as Scalable Heterogeneous Layered Attestation (SHeLA) [8], section 3.2.2 and Secure Asynchronous Remote Attestation for IoT Systems (SARA) [7] in section 3.2.3.

3.2.1 RATS Model

The objective of the RATS working group is to propose a framework (architecture and standardized data formats) that can guide the secure and reliable connection between IoT devices. The structure currently in place consists of a device - "attester" - that produces reliable information - "evidence" - about itself, transmitting it to another device responsible for validating the information received - "verifier". A third party is responsible for receiving the verification result - "Relying Party" - to make decisions about communication, access, and permissions, which the attester may or may not perform on the network.

The Verifier, when evaluating the Evidence, or the Relying Party, when evaluating the Attestation Results, verify that the Claims values correspond to their evaluation policy. Such checks include, for example:

- The equality comparison with a reference value, or
- Be in a range delimited by reference values, or
- Belonging to a set of reference values or
- Appear as a value in other claims.

A concrete example would be the need for a device - Attester - to prove that it can convey with another device. For that, it has to provide Evidence from its system that must be validated by a trusted entity - Verifier. The verifier will appraise the Evidence and return an attestation result to prove the attester's state. This example can be enough for Challenge/Response Model but not for other models defined in RATS documents [33].

Upon completion of all checks regarding the assessment policy, values are accepted as input to determine Attestation Results when evaluating Evidence or as input to a Relying Party when evaluating Attestation Results. A Relying Party can be any equipment with software capable of performing authorizations on the network, such as a router, switch, or access point responsible for admitting certified devices to the network.

Within the scope of this dissertation, it is essential to give relevance to the following documents from the RATS Study Group:

- Remote Attestation Procedures Architecture that deals with the definition of the RATS working architecture, in version 15 [1];
- Reference Interaction Models for Remote Attestation Procedures, in version 05 [34], deals with implementing remote attestation based on the architecture.

In the conceptual flow of messages proposed by RATS in Figure 3.2, we have three main actors: Attester, Verifier and Relying Party. In addition to the entities that support the actors with pre-defined policies, reference values and endorsements. These support entities are not part of the scope of implementation proposed by the document [34] and for which they will not be mandatory in the development demonstrated by this dissertation. We will stick with the three main actors.

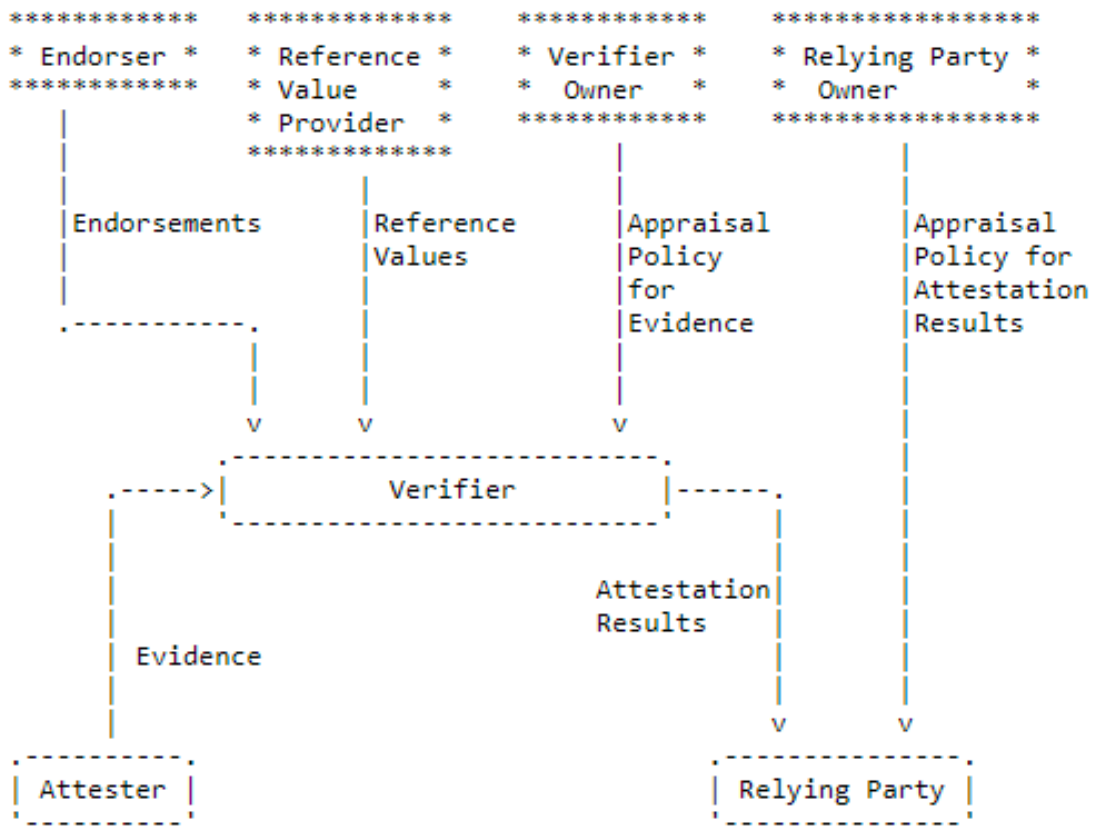

Figure 3.2: RATS Dataflow

In the RATS conceptual flow, we have [1]:

- An “Attester” sends evidence to the “Verifier”, - which the verifier may have requested in advance.
- The “Verifier” assesses the evidence using or not “Endorsements”, “Reference Values”, or “Policies” and thereby attests to the reliability of the “Attester”. The “Verifier” then generates "Attestation Results" that may or may not be used by the "Relying Party", as we will see later.
- The “Relying Party” uses the attestation result (which can be positive, negative or have other values) and can apply its evaluation policy to make application-specific decisions, such as authorization decisions, or accept the received attestation result as valid. A negative result can mean a limitation of permissions, or even the attesting party is not confident or has had part of its system corrupted and needs corrective actions.
- "Evidence" is a set of information requested about the target environment that reveals operational status, health, and configuration with security relevance.
- "Attestation Results" are the input the Relying Party uses to decide to what extent it will trust a specific Attester and allow it to access data or perform some operation.

- An "Endorsement" is a secure statement that some entity (e.g. a manufacturer) attests to the device's integrity.
- When evaluating Evidence or when evaluating Attestation Results, the Verifier, the Relying Party or the Relying Party, use the corresponding Claims values against the restrictions specified in its evaluation policy, which can be, for example, the comparison between values (true/false, valid/not valid)
- The other entities serve to support the decisions of the "Verifier" and the "Relying Party" with values and depend a lot on the type of implementation and expected results.

The table 3.1 shows the main roles defined by RATS and helps us understand each actor shown in figure 3.2.

Role	Function performed by an entity:	Produces	Consume
attester	produces evidence about itself, which must be evaluated for the attester to be considered reliable.	evidence	N/A
verifier	assesses the validity of an attester's evidence and produces an attestation resulting from that assessment	attestation result	- Evidence, - Reference Values, - Endorsements, - Appraisal Policy for Evidence
Relying Party	depends on the validity of information about an Attester, for the purpose of applying trusted actions	Access validation	- Attestation Results, - Appraisal Policy for Attestation Results
Relying Party Owner	is authorized to configure an Assessment Policy for attestation results in a relying party (admin)	Appraisal Policy for Attestation Results	N/A
Verifier Owner	is authorized to configure the Assessment Policy for evidence in a verifier (admin)	Appraisal Policy for Evidence	N/A
endorser	whose Endorsements can help Verifiers assess the authenticity of Evidence and infer other capabilities of the Attester.	endorsements	N/A
Reference Value Provider	whose Benchmark Values help Verifiers assess Evidence to determine whether Known Acceptable Claims have been recorded by the Attester (Manufacturers)	Reference Values	N/A

Table 3.1: Roles

Terms and Concepts

In the RATS specification proposal document [34], the concepts are related to information produced, requested and exchanged between entities. A complete list can be found in the document, as mentioned earlier.

For a better understanding, a list of those that will be used in the context of the dissertation, is provided below:

- **Claim** - Make up the structure of evidence and other artefacts in the information chain in RATS. They can be a statement or a value pair, for example.
- **Evidence** - A set of Claims generated by the Attester to be evaluated by a Verifier. Evidence can include configuration data, measurements, telemetry, or inferences.

- **Attestation Result** - This is the output generated by a verifier containing information about an Attester, where the Verifier guarantees the validity of the results.

Reference Models

According to the RATS architecture document, two reference models are proposed to increase the fundamental mode (Challenge/Response)l [1; 34]: Passport Model and Background-check model.

The **Passport-Model** is an analogy to the model of issuing passports by a nation to an individual and its use at an immigration counter. So, in this immigration counter analogy, the citizen is the Attester, the passport issuing agency is a Verifier, the passport application and identification information (e.g. birth certificate) is the Proof, the passport is a Result Certificate, and the immigration desk is a Relying Party.

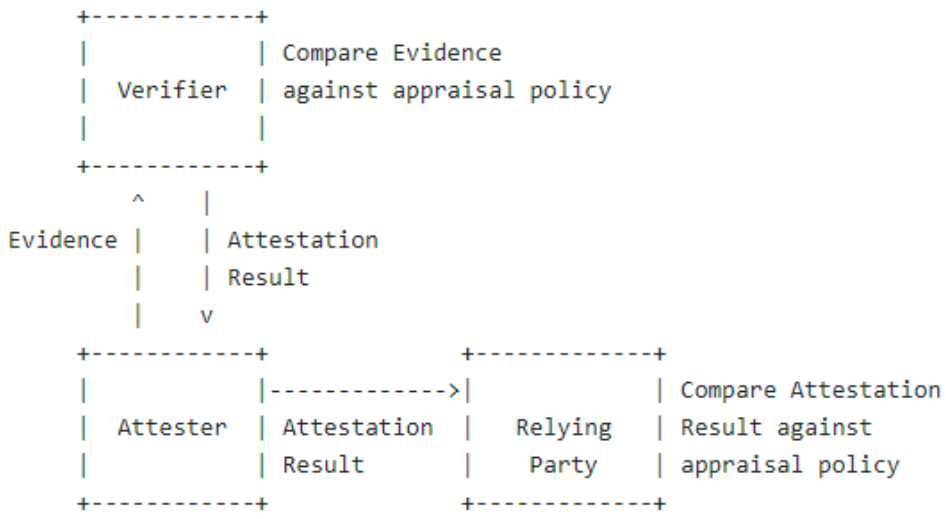

Figure 3.3: Passport Model

In this model. as illustrated in figure 3.3, an Attester transmits the Evidence to a Verifier, who compares it against its assessment policy or verifies the requested data. The Verifier then returns an Attestation Result to the Attester, which does not consume the Attestation Result but can cache it. The Attester can then present the Attestation Result (and possibly additional Claims) to a Relying Party, who then compares this information against its own assessment policy.

The **Background-check model** has this name because of the similarity of how employers and organizations perform background checks. When a prospective employee submits information about previous education or experience, the employer will contact the respective institutions or employers to validate such information provided. So, in this analogy, a potential volunteer is an Attester, the organization is the Relying Party, and the reporting organization is a Verifier.

In this model - figure 3.4, an Attester transmits Evidence to a Relying Party, who

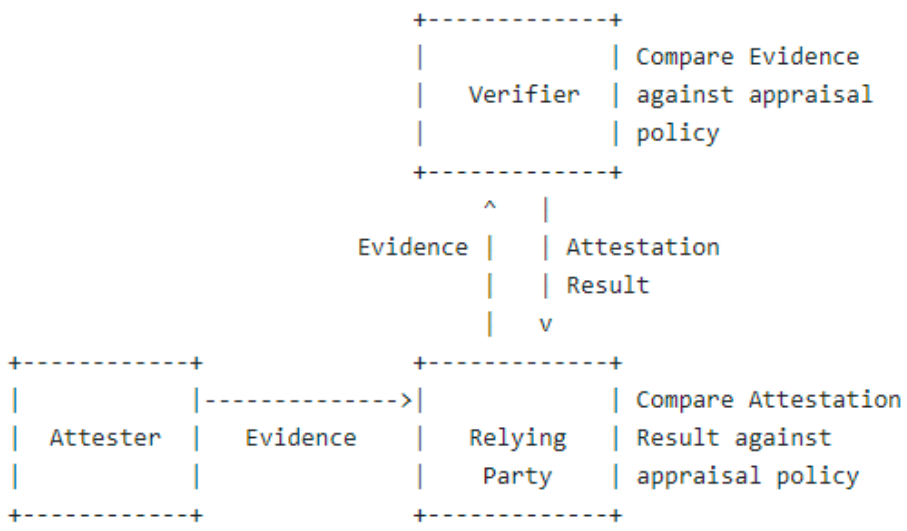

Figure 3.4: Background-check Model

forwards it to a Verifier. The Verifier compares the Evidence against its assessment policy and returns an Attestation Result to the Relying Party. The Relying Party then compares the Attestation Result with its own evaluation policy.

The Model, namely "challenge/response", only allows the validation that an attester meets the conditions to convey with another entity. This would be enough in some cases to implement a model with these two entities. However, using a third entity whose function would be to validate that the information generated by the Verifier is trustworthy and, from there, grant access permission to resources or other entities provides us with an increase in security.

Considerations

The Relying Party believes in the Verifier. This trust can be expressed "by storing one or more trust anchors in a secure location known as a trust anchor store" [1], causing the Relying Party to store the Verifier's public key or certificate in its trust anchor store, which can be on local storage or an external entity defined in RFC6024 [35].

For a more robust level of security, the Relying Party may require the Verifier to provide information about itself that the Relying Party can use in assessing the Verifier's reliability before accepting the Attestation Results. In order to implement a trust model that fully utilizes the "trust anchor" concept, it is necessary to have implemented the "Relying Party Owner". In none of the models mentioned above, the use of the Relying Party Owner is essential.

In the scope of this dissertation, we will consider that the Passport Model has prior knowledge about the Verifier's public key, thus making it unnecessary to implement a "Relying Party Owner".

The disadvantage of the Models are as follows [1]:

1. The Verifier may not issue a positive Attestation Result due to the Evidence

not passing the Evidence Assessment Policy;

2. The Attestation Result is examined by the Relying Party and based on the Attestation Results Evaluation Policy, and the result does not pass the policy.

3. The Verifier is inaccessible or unavailable.

For 1 and 2, actions can be configured according to the fault level. These actions depend directly on where the protocol is implemented. For example, in a request for access permission, the Attestation Result can take a non-positive value that indicates the level of access granted. Another example is receiving a negative attestation where the protocol implementation does not support this value but has means of denying access to the network or even requesting a reboot and even a firmware reapplication. Failure 3 to be relevant depends on the implementation of the network, as the Attester and the Relying Party can have a list of verifiers and choose the one that is available, closest, or has another availability policy.

Besides that, Passport Model has less network traffic than the Background-check Model and the possibility to cache the attestation result (for some time) the Attestation Result in case of no communication with a Relying Party.

3.2.2 SHeLA Model

It is a model of at least three layers, as showed in the figure 3.5, where the Root Vefifier is at the top layer, which is the "owner" of the attestation network and has unlawful computing power.

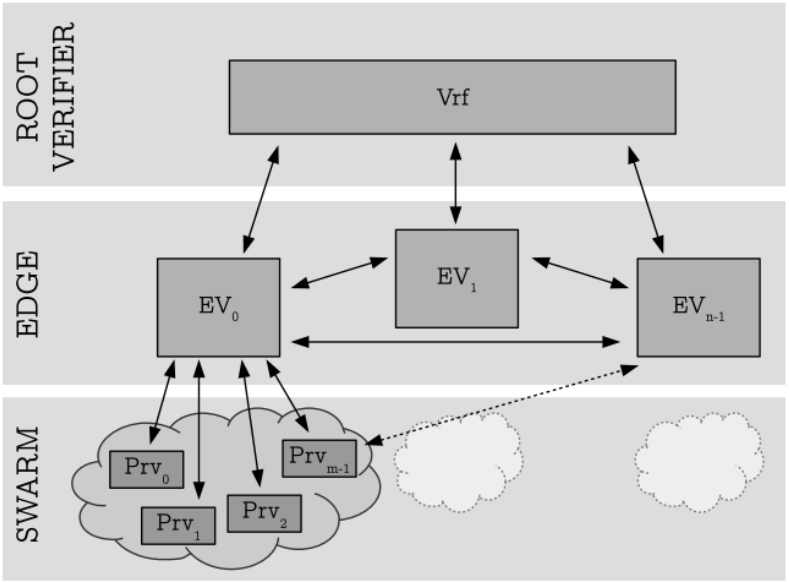

Figure 3.5: SHeLA Topology

Communicate with the edge verifiers through the network (wired or wifi). The Edge Verifiers are devices with significant computing power and storage, so they are more potent than high-end devices, called Swarm Nodes. Edges verifiers have a permanent connection to root verifiers. The SHeLA model assumes that

Edge devices are trusted entities supporting secure hardware that the root verifier can attest. Swam nodes (the provers) are low-end devices that communicate using particular wifi protocols such as Zigbee, Wifi or Bluetooth. These can be static or mobile devices that allow one-to-one attestation [8]

The possible limitation of applying the SHeLA model to our framework would be the high cost of Root and Edge Verifier equipment.

3.2.3 SARA Model

SARA proposes using asynchronous protocols for group attestation, resulting in the non-interruption of all devices simultaneously. In addition, it keeps the history of previous interactions of the tasters [7]

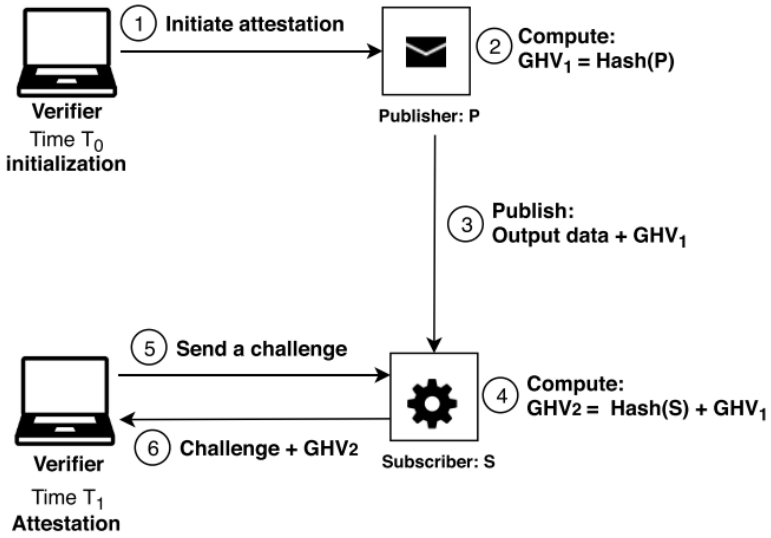

Figure 3.6: SARA system Model

Sara proposes the publisher/subscriber model as the Message Queue Telemetry Transport (MQTT) [36] or Advanced Message Queuing Protocol (AMQP) [37], as depicted in figure 3.6. However, MQTT is already used in services provided by Cloud providers (e.g., Google IoT and AWS IoT) and is present in the simplest IoT devices such as a smart socket plug (e.g., Shelly Plug S).

The use of queue managers is an advantage. However, it can be a point of failure, which we should observe. Another positive point is the historical attestation about previous attestations that can serve us as a reputation record.

3.3 CHARRA Project

CHARRA - CHAllenge-Response based Remote Attestation with TPM 2.0, as stated on its website: "is a proof-of-concept implementation of the Remote Challenge/Response Attestation interaction model of the IETF RATS Reference Interaction Models for Attestation Procedures Remote using TPM 2.0." CHARRA

conforms to the RATS architecture description [38]. It is an implementation developed by the Fraunhofer Institute for Secure Information Technology, whose responsible developer, Michael Eckel [39], has completed the proof of concept, as shown in figure 3.7.

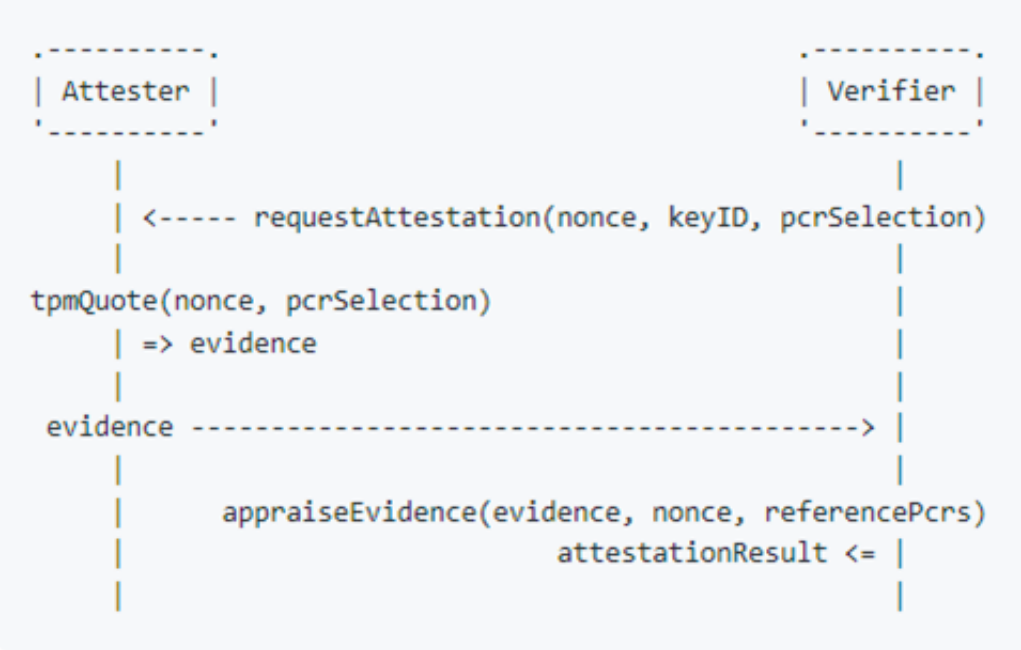

Figure 3.7: CHARRA protocol flow

The documents generated by the RATS working group standardize formats, procedures and protocols to describe the exchange of information between the involved trusted parties. The working group does not determine or indicate one specific technology implementation to be used in its application.

3.3.1 Advantages

CHARRA was written in C language, implementing libraries for accessing data on TPM chips and transporting this data using protocols suitable for use in more restricted environments such as the IoT.

The implementation was entirely developed using software, not being linked to a specific hardware pltform for code execution. Docker containers are using the official repository of the TPM2 Software community [40]. Each container uses a TPM 2.0 chip simulator, developed by IBM [41] and a C library for interaction with the TPM [42].

The network communication uses the CoAP (Constrained Application Protocol) protocol (recall 2.2), developed to use UDP instead of TCP, optimizing for use on devices and networks with fewer resources. CoAP uses UDP to transport information, and considering the security required when dealing with IoT devices, CoAP can use DTLS (Datagram Transport Secure Layer) over UDP 2.3. Three modes, depending on the configuration indicated at the time of code execution, are supported by CHARRA:

- UDP without using DTLS;

- UDP with PSK (Pre-Shared Key) using DTLS (Datagram Transport Secure Layer);
- RPK (Raw Public Keys) using DTLS (Datagram Transport Secure Layer).

The collected data (before being sent) is encoded in the CBOR (Concise Binary Object Representation) format. When the data is received, it is decoded to the original format, enabling content to be evaluated. In the case of the Attester, the content received is the necessary claims, basically, TPM data, for Verifier to carry out the attestation.

CHARRA is under the BSD 3 - Clause License - "A permissive license similar to the BSD 2-Clause License, but with a 3rd clause that prohibits others from using the name of the project or its contributors to promote derivative products without written consent" [43].

3.3.2 Limitations

CHARRA implements the secure challenge/response flow to validate evidence collected by an attester but does not return the attestation result back to the attester. It does not fully apply to the CDDL Specification for a Simple CoAP Challenge/Response Interaction - Appendix A [34].

This Challenge/Response model can be used where, for example, we want to ensure that a system has started correctly and is compliant by running only authentic software. An excellent way to better understand and extend remote attestation to user-level software (Java, Python) can be seen in the article [44].

Limitations of implementing the Challenge/Response Model in CHARRA, are as follows:

- What should be the purpose of the certificate received?
- When establishing a new communication with a Relying Party, how can the Attester verify whether or not he is trustworthy?
- Challenge/Response is not an answer to Remote Attestation if you have no Response

The answer lies in the proposal of this dissertation that, from the open-source code of CHARRA, the development of the CHAllenge-Response based Remote Attestation - Passport Model (CHARRA-PM) [9] according to the specifications of IETG RATS working group.

3.4 Summary

Recent surveys [45; 46] identify relevant remote attestation mechanisms for IoT scenarios, given the context of 5G networks and its support for Machine to Machine (M2M) communications.

Besides RATS, SARA and SHeLA, the aim is to complete the state of the art considering these recent surveys, in order to identify possible implementations that

can contribute to the goals of this dissertation.

Chapter 4

Research Methodology

This chapter presents the research methodology that was followed to enable the implementation of a remote attestation to enhance the reliability of devices.

4.1 Motivation

Classic Authorization is only concerned with the information that user/device proves to possess, for instance, it knows the credentials that are associated with its identity, and thus can be authenticated. On another side, attestation is concerned with the device functionality and integrity, so that evidence is provided that the device has not been modified, compromised, or tampered with regarding its initial design. Thus, a device providing evidences that it is benign and is not compromised and therefore full access can be provided.

After reading the documentation of IETF RATS, there was the need to choose a model, either background-check or the Passport Model, as explained in section 5. The choice was the Passport Model , given its advantages, and by operating as the citizen's passports:

1. The type of Evidence that a person must present to their local authorities depends on the nation in question., which corresponds to claims in remote attestation.
2. When citizens have to prove their citizenship or identity, they submit the resultant passport document to other organizations, such as an airport immigration counter, while still in control. Analogue to the Attester sending Evidence to a Verifier in the remote attestation.
3. The Passport is regarded as adequate since it attests to the claims of citizenship and identity and because a reliable institution issued it, corresponding to the Attestation Result in the remote attestation.
4. In this comparison, the immigration desk serves as the Relying Party, the Passport is the Attestation Result, and the passport issuing agency is the Verifier.

5. The immigration desk also has the power to approve or deny your Passport and also can give an expiration date (how long you can stay in the country)

4.2 Objectives

The attestation in IoT devices are a challenge, because most devices have limited computing capacities, the majority of the devices does not include trustable and secure elements to act as root of trust, among other aspects. There are proposals that tackle such challenges, proposing solutions for Remote Attestation in heterogeneous environments, as documented in section 4.

In this study, a strategy for remote attestation in IoT devices is specifically designed, tested, and validated.

Focusing on the documents of IETF RATS workgroup[1] and using the open-source code for the Challenge/Response PoC - CHARRA [38]. The aim is to add another entity that will grant or deny access as a principal function based on an appraisal of the Attestation Result.

More specifically, the objectives of this work are:

- Research a Model with a real implementation of a Framework for Remote Attesting functionality for IoT devices.
- Implement Remote Attestation mechanisms for IoT devices.
- Evaluate the Remote Attestation mechanisms for IoT Devices.
- Document the results for dissertation and scientific publication purposes.

4.3 Approach

The approach for the thesis starts with the analysis of current standardization efforts for Remote Attestation (RA). In parallel, relevant works in the state of the art are also analysed to identify relevant aspects that contribute to the major goal of this thesis - A framework for remote attestation in heterogeneous IoT devices.

There are several works in Remote Attestation (RA) [7; 8; 32; 47]. As stated, the study began by looking for the existence of "de facto" standards, which led us to the IETF working group on the Remote Attestation Protocol (RATS) [1].

The RATS proposal intends to cover all situations without indication of protocols, hardware and security we should use. It is like a one size fits all. RATS can be a guideline when we need to check one approach or another in our scope. Its application considers many policies and information from different sources that make it unfeasible to use in low-cost IoT environments. On the other hand, the most current proposals already think about the capacity of low-cost devices, as observed at SARA, which presents us with a proposal for a framework for cheaper devices, using protocols such as Wifi [48], Zonal Intercommunication Global-standard (ZigBee) [49] and Message Queuing Telemetry Transport (MQTT) brokers, for example.

In SHeLA, there is the possibility of implementing a smaller version than the full proposal to meet such devices. However, edge and layer attesters may need powerful hardware.

4.4 Methodology

The research was based on scientific articles (conference papers, Journals) taken from Research Gate [50], IEEXplore [51], and hardware manufacturers' websites, such as IBM. Using these articles and resources from reliable sources, it was possible to design, develop and evaluate the proposed framework for IoT devices - CHARRA-PM.

Among the preliminary studies, we have the proposal of the working group that intends to launch a standardization for remote attestation covering most of the devices that need this layer of reliability. However, there are gaps to be resolved: The clock synchronization issue where the application is geographically distinct; the security of sensitive data that travel through the protocol must be treated separately; the possibility of an attacker having access to attestation policies and carrying out an attack; insufficient packet size in some networks, for example.

There are an implementations (PoC) of the RATS Challenge/Response on GitHub [38]. These development and related documents were the base for put the work into practice.

After identifying an implementation that was available for Remote Attestation, several emails were exchanged with the author, Mickael Eckel [39] to understand how CHARRA works and how it follows the recommendations of the RATS' documentation and his opinion regarding the implementation of the Passport Model.

We delved deeper into the C language to understand the concepts used by CHARRA, its functions and results, using Debug messages, reading functions and examples from libraries and also the technologies related, such as CoAP protocol, DTLS, and CBOR. When the whole operation of CHARRA was understood, We developed functions to modify CHARRA from a Challenge/Response Model to the Passport Model. New functions were created within the CHARRA utilities, CBOR and cryptographic libraries, and finally, the code was written for the Passport Model.

In response to my e-mails with questions about the efficiency of using the Passport Model instead of the Background-check Model, Mickael Eckel replied that the most significant advantage is that the checker is not involved in every attestation, reducing network traffic and additional communication. with the verifier.

" I think the main advantage is that the verifier is not involved on every attestation. The verifier sends the attestation result to the attester, The RP can then directly talk to the attester, and the attester sends the signed attestation result to the RP (so, the RP knows the verifier verified the attester; the RP must trust the verifier by its certificate). So, the main advantage is that network traffic and additional communication with the verifier is reduced." *Michael Eckel answer*

Chapter 5

CHARRA-PM - CHARRA Passport Model

CHARRA is an open-source PoC of the Challenge/Response model defined in RATS [34]. CHARRA-PM is a development of the Passport Model (see in section 6), also defined by RATS workgroup documents using the source code from CHARRA and developing functions and interactions into a new model.

5.1 Passport Model

The motivation for choosing the Passport Model was primarily due to the minimization of traffic processed by the Relying Party, compared to the one observed in the Background-chek model observed in the figures 3.3 and 3.4. On the other hand, we isolated the Relying Party that centralizes claims and responses as described in the Background-Check model, significantly reducing the Relying Party's work, processing and network connections.

The insertion of a trusted third party in the Challenge/Response model aims to validate the attestation result, sent by the Verifier to the Attester that forwards it to the Relying Party. This process, in turn, can decide the type of permission/role the Attester can have on the network, including limiting or blocking communication.

We created a new proof of concept: the CHARRA-PM [9], taking advantage of the development already carried out in CHARRA, including a new device, the Relying Party, and the entire process necessary for the signing, transfer and validation of the Resulting Certificate as described in the Passport Model in the Drafts of RATS.

5.1.1 Considerations

Some considerations are necessary for the global understanding of what was effectively developed. It was also observed what is necessary in case of application in production.

- All traffic is handled via DTLS between devices, although there is an option to use the UDP protocol (without DTLS);
- RP Trust Verifier - We assume that the Relying Party trusts the Verifier by knowing its public key exchanged over secure channels.
- Attestation freshness (not implemented in CHARRA) - in the RATS architecture document, chapter 10 [1], the freshness of attestation information is described. This information is used to minimize replay attacks. However, the considerations for implementing such functionality involve many variables that are costly and beyond the scope. Examples can be found in Appendix A - Time considerations in [1].

5.1.2 The Contribution of CHARRA-PM

We have identified as relevant the third party entity - relying party in the overall challenge/response model. Where could we really apply the attestation result, for policy enforcement ?

The addition of a third party can be thought of in two situations: functioning as a gateway or firewall for granting privileges depending on the attestation result or even an end device that would only establish communication upon a valid attestation result.

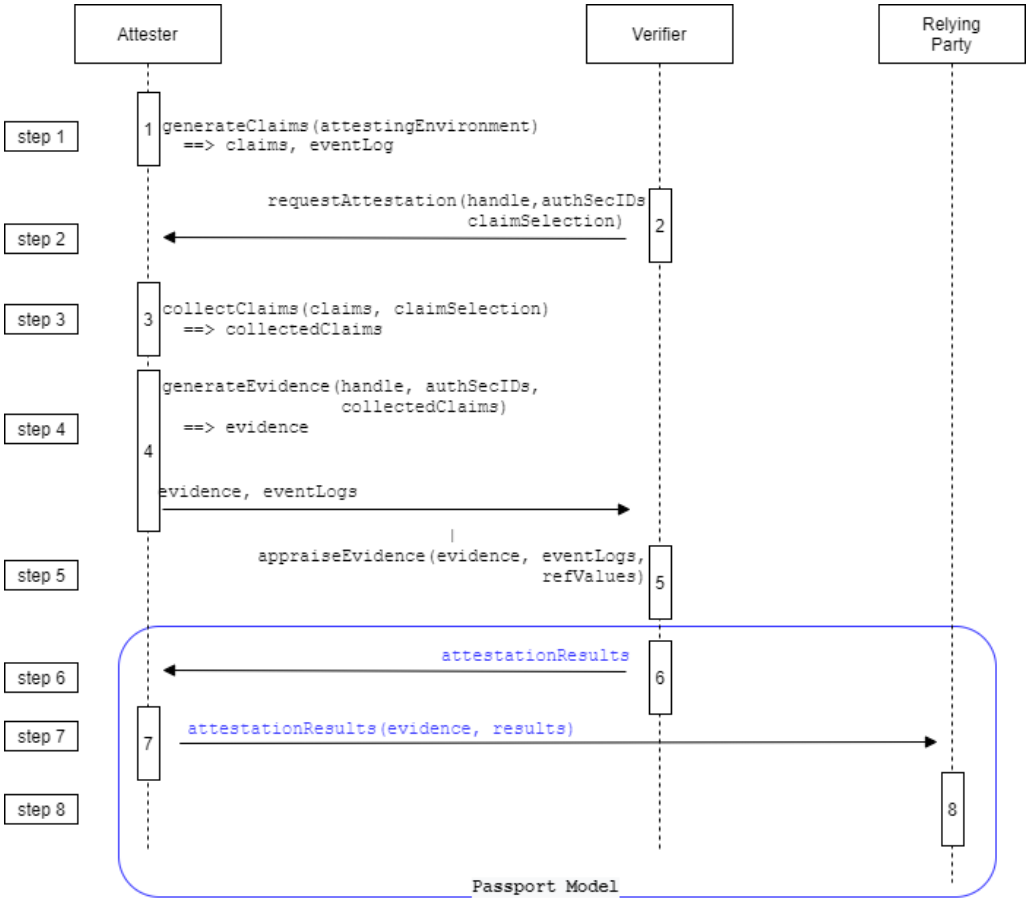

Figure 5.1: CHARRA-PM data Flow

Noting the limitations of the Challenge/Response Model in subsection 3.3.2, we responded for each one:

- The attestation result, validated by the Verifier, is not sent to the Attester.
  - Even being a proof of concept, the model should return the attestation result to the Attester according to the interactions document [34].
- What should be the purpose of the certificate received?
  - The attestation result proves that the Attester has requirements (valid or not) to present to another peer to gain some level of access.
- When establishing a new communication with a Relying Party, how can the Attester verify whether or not he is trustworthy?
  - The Relying Party reviews the submitted certificate by any policy it has or through a relying party owner. This policy can be a true or false check, for example. Most importantly, the Relying Party trusts Verifier and Tester.

The figure 5.1 helps us understanding the generation and data exchange flow between the Passport model parts. Assuming that the communication between the parties is carried out safely and securely, as recommended by the RATS documents, in this case, using the DTLS-RPK or DTLS-PSK, recall section 2.3.The collection of information and communication between the parties involved is summarized in two steps: Challenge/Response and the Passport Model.

5.1.3 Challenge/Response steps

When the Attester starts up, it produces claims about its startup and operational state. Event logs track the statements produced by providing a trail of critical security events in a system, as depicted in steps 1 of Figure 5.1.

The Challenge/Response remote attestation procedure is initiated by the Verifier sending a remote attestation request to the Attester. This request includes an identifier (e.g. in the form of a nonce), authentication secret IDs and claims selection, corresponding to step 2 in Figure 5.1. In the Challenge/Response model, the handle is composed in the form of a virtually unguessable nonce (e.g., a strong random number). The nonce generated by the Verifier ensures that the Evidence is up to date and prevents replay attacks. The authSecID is a key sent by the Verifier to the Attester to sign the Attestation Evidence. Each key is uniquely associated with an Attestation Environment in Tester. As a result, a single Authentication Secret ID identifies a single Attestation Environment.

The Attester collects Claims based on the Claim Selection submitted by the Verifier. If Claim Selection is omitted, all Claims available in the Attester must be used to create the corresponding evidence by default. For example, in a boot health assessment, the Verifier may only ask for a subset, such as Evidence on BIOS/UEFI and firmware, and does not include other information about the currently running environment. This corresponds to step 3 in Figure 5.1.

With Collected Claims, Handle, Authentication Secret IDs, Attester produces the evidence and signs it. It digitally signs the Handle and Claims collected with a cryptographic secret identified by the Authentication Secret ID. It is done once per Attestation Environment, which is identified by the specific Authentication Secret ID. The Attester communicates the signed evidence and all accompanying Event Records back to the Verifier, as depicted in step 4 of Figure 5.1.

Note: The implementation of CHARRA behaves differently, using the TPM private key of the Attester and sending the public key along with the message to the Verifier, as documented in section 2.1.

When the Verifier receives the Evidence and Event Logs, it starts the Evidence assessment process, validating the signature, received nonce against sent nonce, and received claims. The assessment procedures are application-specific and can be conducted by comparing values, using reference measures, and producing the Attestation Results. This is done in step 5 of Figure 5.1.

5.1.4 Passport Model Steps

This subsection discusses the steps involving the passport model, one of the key objectives of this work.

After producing the attestation, the Verifier signs it with its private key and sends it, through a secure channel, to the Attester, as illustrated in step 6 of Figure 5.1. The Attester must immediately send the received certificate to be evaluated by the Relying Party, as per step 7 in Figure 5.1. The Attester cannot process, verify or change the certificate received. At most, it can keep until it expires.

According to RATS documents [1], Attestation results can contain a Boolean value indicating compliance or non-compliance with a verifier's assessment policy, dispensing with the use of a Relying Party Owner to provide an additional evaluation policy. The Relying Party confers greater confidence in the model, as it is responsible for verifying whether the attestation result is reliable and valid, deciding the level of access it will give (or not) to the system.

When the relying party receives the certificate, it starts evaluating the information, verifying the signature and the value(s) received. At this time, and depending on the policy, it will or will not give privileges to the Tester, as per step 8 in Figure 5.1.

In the stages where information is exchanged between parties, it is carried out through a secure communication channel, as mentioned in section 2.3. Evidence from TPM chips created exclusively to store and generate reliable information, such as binary serialization of data, encryption of keys and hash using algorithms from the SHA-2 family, used by version 2.0 of TPM as explained in section 2.

5.2 Implementation of CHARRA-PM

In development, we reused CHARRA code, such as calling connection and cryptographic functions. New functions were also developed from scratch to handle

the Passport Model part. It should be noted that it was not a development from scratch but an improvement to the existing code with the endorsement of developer Michael Eckel [39]. The CHARRA-PM was coded as PoC to the Passport Model as described in RATS documentation [1] and the opensource code is available on GitHub [9].

5.2.1 Establishing a CoAP session with DTLS

This section will present and explain the piece of code functions that handle CoAP sessions using DTLS PSK and RPK.

```
coap_session = charra_coap_new_client_session_psk (ctx,
LISTEN_RP, port_rp, COAP_PROTO_DTLS, tls_psk_identity,
( uint8_t *)dtls_psk_key, strlen(dtls_psk_key)
```

Listing 5.1: Function to set up PSK CoAP connection

The function `charra_coap_new_client_session()` listed in List 5.1, is responsible for establishing a channel through the CoAP protocol using DTLS-PSK (Preshared key). The shared key is known to all parties involved (`dtls_psk_key`). The other fields identify the Relying Party, the DTL protocol (COAP_PROTO_DTLS), the shared key, and its size.

```
charra_coap_setup_dtls_pki_for_rpk(&dtls_pki,
dtls_rpk_private_key_path, dtls_rpk_public_key_path,
dtls_rpk_peer_public_key_path, dtls_rpk_verify_peer_public_key) ;
```

Listing 5.2: Function to set up RPK CoAP connection

The function `charra_coap_setup_dtls_pki_for_rpk()` listed in List 5.2, is responsible for establishing a channel through the CoAP protocol using DTLS-RPK (Raw Public Keys). Here the public key of the target device must be known. That is, the Attester knows the public key of the Verifier and the Relying Party; Verifier knows the Tester's public key; the Relying Party knows the public key of the Tester and the Verifier.

5.2.2 Certificate Signing Process

This section shows CHARRA-PM calling the responsible function to sign the attestation result.

```
01: /* signing attestationResult */
charra_sign_att_result(dtls_rpk_private_key_path,
( unsigned char *) attestationResult,
signature,&signature_len);
02: /* compiling data into struct */
att_result = {
.attestation_result_data_len = sizeof (attestationResult),
.attestation_result_data = ( unsigned char *) attestationResult,
.attestation_signature = signature,
.attestation_signature_len = signature_len,
};
03: /* unmarshal data */
charra_marshal_attestation_result(&att_result, &ares_buf_len,
```

```
&ares_buf);
```

Listing 5.3: Sign and encoding the attestation Results

The listing 5.3 shows the piece of code where the Verifier call functions responsibles to sign the `attestetionResult` and encoding using CBOR.

This process occurs in the Verifier, within the following steps:

- Step 01: The `attestationResult` is signed using your private key.

  charra_sign_att_result() : is the function responsible for signing the attestation Result. The signature and its length are returned in the `signature` and `signature_len` fields.

- Step 02: Is where the values are associated with the message structure

  (`att_result`);

- Step 03: The entire message structure is formatted for CBOR in the

  `charra_marshal_attestation_result()` function.

5.2.3 Receipt of the certificate by the Relying Party

This is where the Relying Party call functions responsible to appraise the Attestation Result the `attestetionResult`.

```
01: /* Reading CoAP data*/
coap_get_data_large(in, &data_len, &data,
&data_offset, &data_total_len);

02: /* convert from CBOR to data */
charra_unmarshal_attestation_passport(data_len, data, &att_result);

03: /* Validating signature */
charra_verify_att_result(verifier_public_key_path,
att_result.attestation_result_data,
att_result.attestation_signature,
att_result.attestation_signature_len);

04: /* mbedtls function related to signature verify */

mbedtls_pk_parse_public_keyfile(&peer_public_key,
   peer_public_key_path);
charra_crypto_hash(MBEDTLS_MD_SHA256, att_result, att_result_len,
   hash);

mbedtls_pk_verify( &peer_public_key, MBEDTLS_MD_SHA256, hash, 0,
   signature, sig_size );
```

Listing 5.4: Receive and apreaise Attestation Results

The listing 5.4 shows the piece of code, with the followins steps:

- Step 01: Reading the data received via CoAP;

- Step 02: Reverting the CBOR encoding in data that the code can process;
- Step 03: Verifies the validity of the received signature using the Verifier's public key. This process will generate a new hash (SHA256) that will be used with the public key to verify the validity of the signature.
- Step 04: Functions called by step 03, coming from the C mbedtls library used to read the public key, generate the hash of the `attestationResult` and validate the signature.

5.2.4 CoAP Endpoints

The development was based on using the CoAP protocol, documented in section 2.2, with endpoints for communication using the FETCH method, as documented in section 2.2.1. Each endpoint in the code is associated with a handler function that handles the information received. Figure 5.2 illustrates the CoAP endpoints exported by each device and the flow and sequence of data.

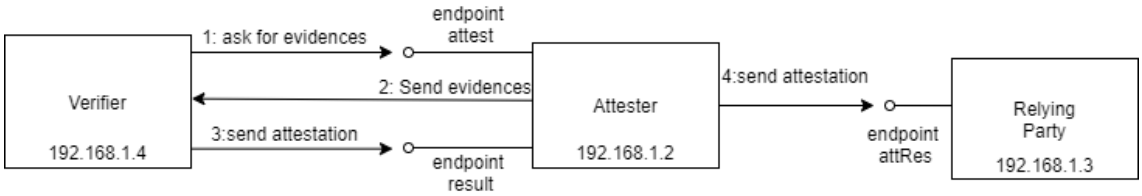

Figure 5.2: Application's endpoints

5.2.5 Docker Environment

The running environment chosen for this PoC was Docker containers. Three docker containers Attester, Verifier and Relying Party, run the same docker image and map a volume in a local folder on the host with the binaries.

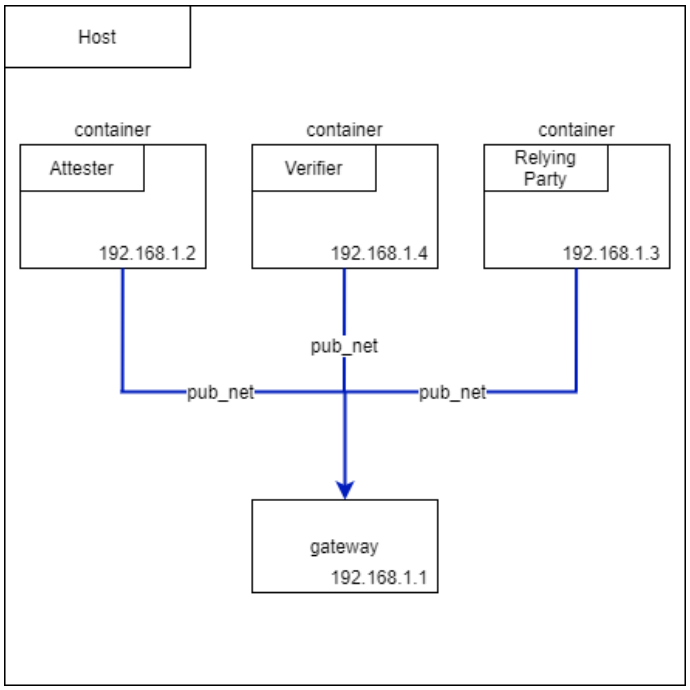

Figure 5.3: Docker Containers

Figure 5.3 help to understand the container distribution and the shared network configuration.

Example of docker command to creating a network (pub_net) for use in a Docker environment.

```
docker network create -d macvlan --subnet=192.168.1.0/24 \
--gateway=192.168.1.1 -o parent=eth0 pub_net
```

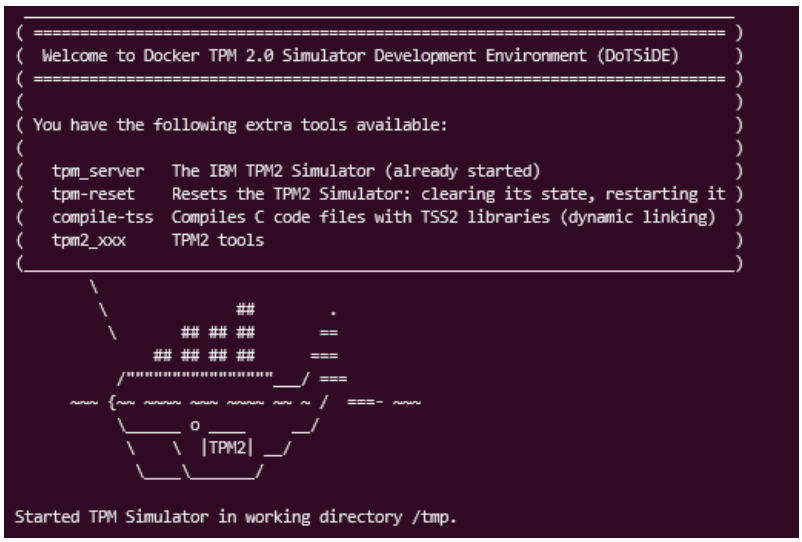

Figure 5.4: Docker with TPM 2.0 configured

The docker container image was based on the official *tmp2software* repository [40][41], which simulates a TPM environment showed in figure 5.4. All detailed steps to create the container are in the README.md file at GitHub [9].

5.2.6 Considerations

The Attester must know the IP of the Relying Party. The Verifier needs to know the Attester's IP, and only the Relying Party does not need to point to an IP, as it will receive the attestation result through the Attester. There is no mechanism yet published to perform the discovery of the Relying Pary aand Attester endpoints.

All binaries can be configured from options in the command line. For example, "-r" indicates that the DTLS protocol will use the RPK; "-p" will use DTLS with PSK; "–ip" respectively indicates the IP of a pair; "–ip-rp" indicates the IP of the relying part and can only be configured in the Attester. See more detailed options in appendix B.

The Relying Party can be configured in a future version with a list of IPs corresponding to authorized testers. So only the attesters on this list would have their claim/evidence confirmed.

Chapter 6

Validation, Evaluation and Results

This chapter introduces the validation, the evaluation methodology and the results achieved in the proof of concept CHARRA-PM.

6.1 Proof of Concept

A Proof of Concept to validate CHARRA-PM was developed implementing in C, the steps 6, 7 and 8 of figure 5.1. This section aims to show and explain the PoC running process using screenshots.

The PoC consists of 3 (three) different devices, each running its isolated environment in a Docker container on a internal shared network, as illustrated in Figure 5.3.

Figure 6.1: Attester ready

When starting the Attester, we can see in the figure 6.1 information about creating two CoAP endpoints using the FETCH method: **attest** and **result**.

The endpoint **attest** is where the Verifier requests evidence and the endpoint **result** is where the Verifier will send the attestation result back to the Attester.

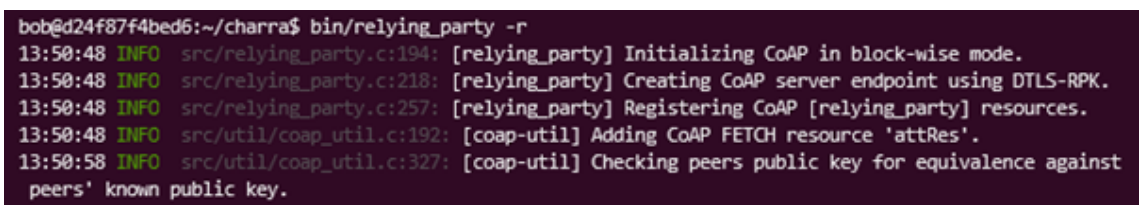

Figure 6.2: Relying Party ready

The figure 6.2 shows the Relying Part endpoint **attRes** to where the Attestation Result from the Attester will be received.

```
bob@f9f983574125:~/charra$ bin/verifier -r --ip=192.168.1.2
13:50:57 INFO  src/verifier.c:249: [verifier] Initializing CoAP in block-wise mode.
13:50:57 INFO  src/verifier.c:257: [verifier] Registering CoAP response handler.
13:50:57 INFO  src/verifier.c:274: [verifier] Creating CoAP client session using DTLS-RPK.
13:50:57 INFO  src/verifier.c:328: [verifier] Creating attestation request.
13:50:57 INFO  src/verifier.c:473: [verifier] Generated nonce of length 20:
                                              0x352c58f53677efec334120296aa1715a469d782a
13:50:57 INFO  src/verifier.c:338: [verifier] Marshaling attestation request data to CBOR.
13:50:57 INFO  src/verifier.c:348: [verifier] Adding CoAP option URI_PATH.
13:50:57 INFO  src/verifier.c:356: [verifier] Adding CoAP option CONTENT_TYPE.
13:50:57 INFO  src/verifier.c:366: [verifier] Creating request PDU.
13:50:57 INFO  src/verifier.c:380: [verifier] Sending CoAP message.
13:50:57 INFO  src/verifier.c:388: [verifier] Processing and waiting for response ...
```

Figure 6.3: Verifier ready

When the Verifier starts, it creates a CoAP connection with the Attester, in this case (IP 192.168.1.2). Then it creates an attestation request indicating what it needs (claims) as a response. The listing 6.1 is the request structure format. Then creates the CoAP message (encoded in CBOR) and sends it. Then wait for the Attester's response, as depicted in Figure 6.3.

```
13:50:57 INFO  src/attester.c:327: [attester] Resource 'attest': Received message.
13:50:57 INFO  src/attester.c:341: [attester] Received data of length 52.
13:50:57 INFO  src/attester.c:343: [attester] Received data of total length 52.
13:50:57 INFO  src/attester.c:348: [attester] Parsing received CBOR data.
13:50:57 INFO  src/attester.c:358: [attester] Preparing TPM quote data.
13:50:57 INFO  src/attester.c:369: Received nonce of length 20:
                                   0x352c58f53677efec334120296aa1715a469d782a
13:50:57 INFO  src/attester.c:397: [attester] Loading TPM key.
13:50:57 INFO  src/core/charra_key_mgr.c:36: Loading key "PK.RSA.default".
13:50:57 INFO  src/util/tpm2_util.c:117: Primary Key created successfully.
13:50:57 INFO  src/attester.c:406: [attester] Do TPM Quote.
13:50:57 INFO  src/attester.c:414: [attester] TPM Quote successful.
13:50:57 INFO  src/attester.c:442: [attester] Preparing response.
13:50:57 INFO  src/attester.c:466: [attester] Marshaling response to CBOR.
13:50:57 INFO  src/attester.c:474: [attester] Size of marshaled response is 1277 bytes. XTCG
13:50:57 INFO  src/attester.c:481: [attester] Adding marshaled data to CoAP response PDU and send it.
```

Figure 6.4: Attester Processing Evidence

```
typedef struct {
    bool     hello;
    size_t    sig_key_id_len;
    uint8_t   sig_key_id[ SIG_KEY_ID_MAXLEN ];
    size_t    nonce_len;
    uint8_t   nonce[ sizeof (TPMU_HA)];
    uint32_t  pcr_selections_len;
    pcr_selection_dto pcr_selections[TPM2_NUM_PCR_BANKS];
    uint32_t  event_log_path_len;
    uint8_t * event_log_path;
} msg_attestation_request_dto;
```

Listing 6.1: Structure of request claims

```
typedef struct {
    uint32_t attestation_data_len;
    uint8_t  attestation_data[ sizeof (TPM2B_ATTEST)];
    uint32_t tpm2_signature_len;
    uint8_t  tpm2_signature[ sizeof (TPMT_SIGNATURE)];
    uint32_t tpm2_public_key_len;
    uint8_t  tpm2_public_key[ sizeof (TPM2B_PUBLIC)];
    uint32_t event_log_len;
    uint8_t * event_log;
} msg_attestation_response_dto;
```

Listing 6.2: Structure of attestation response

```
typedef struct {
    uint32_t attestation_result_data_len;
    unsigned char * attestation_result_data;
    uint8_t * attestation_signature;
    size_t   attestation_signature_len;
} msg_attestation_appraise_result_dto;
```

Listing 6.3: Structure of Attestation Result

Once Attester receives the message formatted, as listing 6.1 in the attest endpoint, it extracts the information from the CBOR-formatted message, turning it into readable data. It then collects the requested information, transforms it into CBOR format, and sends it to Verifier, as depicted in figure 6.4, using the response format shown by listing 6.2.

```
13:50:57 INFO  src/verifier.c:521: [verifier] Resource 'attest': Received message.
13:50:57 INFO  src/verifier.c:538: [verifier] Received data of length 1277.
13:50:57 INFO  src/verifier.c:540: [verifier] Received data of total length 1277.
13:50:57 INFO  src/verifier.c:545: [verifier] Parsing received CBOR data.
13:50:57 INFO  src/verifier.c:572: [verifier] Starting verification.
13:50:57 INFO  src/verifier.c:592: [verifier] Loading TPM key.
13:50:57 INFO  src/verifier.c:598: [verifier] External public key loaded.
13:50:57 INFO  src/verifier.c:602: [verifier] Preparing TPM Quote verification.
13:50:57 INFO  src/verifier.c:613: [verifier] Verifying TPM Quote signature with TPM ...
13:50:58 INFO  src/verifier.c:619: [verifier]     => TPM Quote signature is valid!
13:50:58 INFO  src/verifier.c:629: [verifier] Converting TPM public key to mbedTLS public key ...
13:50:58 INFO  src/verifier.c:640: [verifier] Verifying TPM Quote signature with mbedTLS ...
13:50:58 INFO  src/verifier.c:646: [verifier]     => TPM Quote signature is valid!
13:50:58 INFO  src/verifier.c:667: [verifier] Verifying nonce ...
13:50:58 INFO  src/verifier.c:672: [verifier]     => Nonce in TPM Quote is valid! (matches the one
13:50:58 INFO  src/verifier.c:685: [verifier] Verifying PCRs ...
13:50:58 INFO  src/verifier.c:687: [verifier] Actual PCR composite digest from TPM Quote is:
                                              0x2d5565fb483d8ea4525a7a9229677d1038ad34b6e22c8d5152e

13:50:58 INFO  src/core/charra_rim_mgr.c:82: Found matching PCR composite digest at index 0 of the
13:50:58 INFO  src/verifier.c:698: [verifier]     => PCR composite digest is valid!
13:50:58 INFO  src/verifier.c:752: [verifier] +----------------------------+
13:50:58 INFO  src/verifier.c:755: [verifier] |   ATTESTATION SUCCESSFUL   |
13:50:58 INFO  src/verifier.c:760: [verifier] +----------------------------+
```

Figure 6.5: Evaluating Evidences and issuing the Attestation Result

The Verifier receives the message from the Attester, transforms the CBOR into readable information, and then verifies the received data, as listed in listing 6.2, and issues the Attestation Result, as illustrated in Figure 6.5.

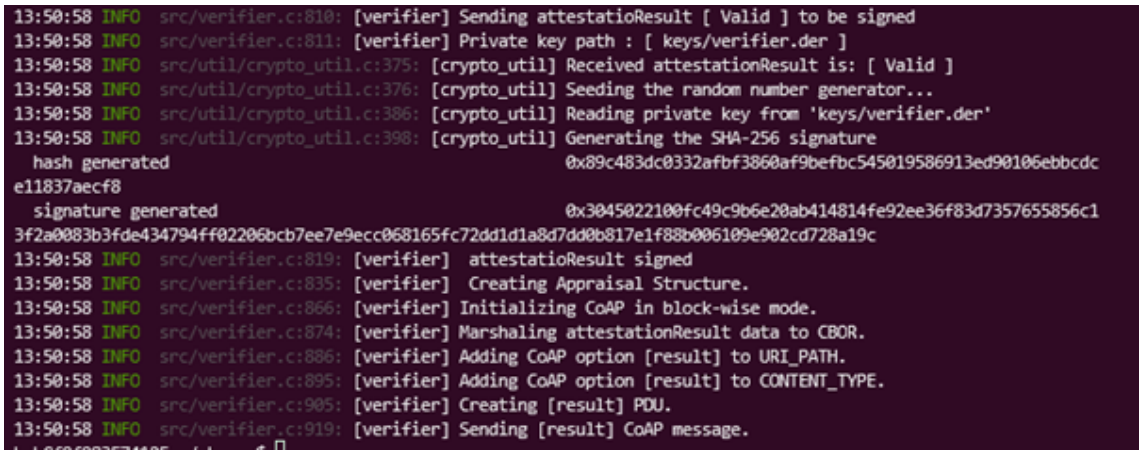

Figure 6.6: Evaluating Evidences and issuing the Attestation Result

After issuing the result, Verifier signs the content of the attestation and fills in the message information - listing 6.3, converts the message into CBOR format and transmits it to the attester. That is Verifier's final participation in the model, as per figure 6.6.

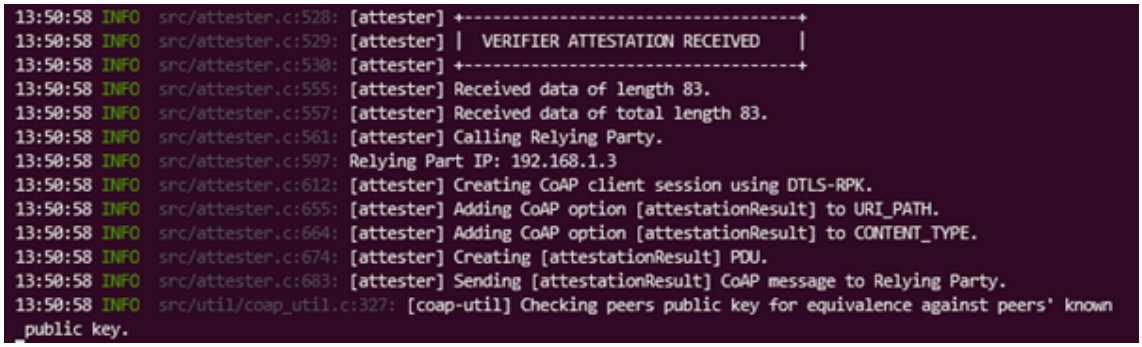

Figure 6.7: Attestation Result Receives from Verifier

The Attester receives the certificate, creates the connection via CoAP with the Relying Party and forwards the received message, not keeping a copy of it, as per figure 6.7.

Figure 6.8: Relying Party Receives and appraise the Attestation Result

The signed attestation is received at the Relying Party, and verification begins. Once the signature is valid, the attestation value can be used to grant the appropriate level of permission to the Attester or even block communication. For example, if the Relying Party is a firewall, it can allow or block access, as per figure 6.8.

Note: In this proof of concept, we only send a text value due to the attestation - see listing 6.3. However, it is possible to add other relevant information depending on the application built to improve the granularity of the evaluation in Relying Party. In figures 6.1, 6.2 and , 6.3, it is also possible to verify that the Passport Model runs on a CoAP connection using DTLS with RPK.

In Appendix A, we exemplify, with screenshots, the CoAP, DTLS and CBOR traffic between the devices.

6.2 Evaluation Methodology

To assess the performance of the PoC, we consider the time required to execute each step in the attestation process. This measurement also allowed us to quantify the impact of the additional steps that were introduced with CHARRA-PM

Looking the figure 6.9 helps us understand which process steps were measured. Each step was measured by capturing the time registered by the CPU clock before and immediately after execution. For the time measurement, the *clock()* function of the *time.h* C library language was our choice. This function returns the number of elapsed clock pulses since it was called. To get the number of seconds used by the CPU, one needs to divide by the number of CPU ticks per second (*CLOCKS_PER_SEC*).

CLOCKS_PER_SEC is the macro responsible for storing the number of pulses per second of the machine's processor on which the program is being executed [52].

For different executions the times go up or down regardless of the use of encryption using the DTLS protocol (PSK or RPK).

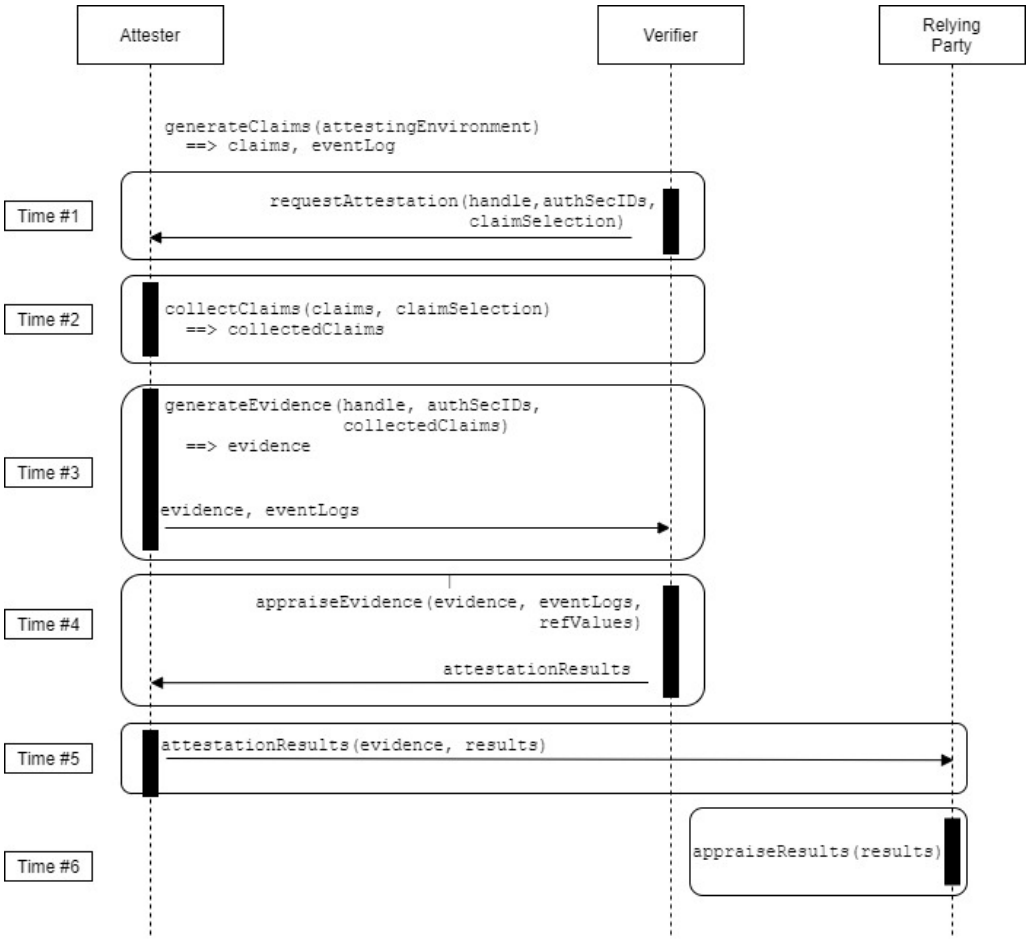

Figure 6.9: Times measurement interactions

Times were computed for the *remote functions* (libraries call) and the *total execution time of the local functions* (which are part of the execution code).

```
01: t = 0 ;
02: t = clock();
03: charra_r = charra_unmarshal_attestation_request(data_len, data,
    &req);
04: t = clock() - t;
04: time_taken = (( double )t)/CLOCKS_PER_SEC;
06: total_func = total_func + ( double )t;
07: charra_log_info("[ TIME ] charra_unmarshal_attestation_request
   () tooks %f secs" , time_taken);
```

Listing 6.4: CBOR decoding time measurement

```
01: t = 0 ;
02: t = clock();
03: if ((charra_sign_att_result(dtls_rpk_private_key_path,
        (unsigned char *) attestationResult,
        signature, &signature_len) != 0 ))
        { charra_log_error(
           "[" LOG_NAME "] error signing attestation result." );
          result = CHARRA_RC_CRYPTO_ERROR;
          goto cleanup;
        }
04: t = clock() - t;
05: time_taken = (( double )t)/CLOCKS_PER_SEC;
06: total_func = total_func + ( double )t;
07: charra_log_info(
    "[ TIME ] AttestatioResult signed tooks %f secs" , time_taken);
```

Listing 6.5: Attestation Signature time measurement

The listings 6.4 and 6.5 presents two code fragments to exemplify how the time were collected.

Times were computed for the remote functions (which are called by external libraries) and the total execution time of the local functions (which are part of the programs). Below is explained what each line of code represents:

- Line 01 - resets the variable;
- Line 02 - start counting time;
- Line 03 - call the remote function;
- Line 04 - ends the time discounting the elapsed time;
- Line 05 - calculate the value in seconds
- Line 06 - accumulates the time spent in the function it is running on. This will be the total value of the running function;
- Line 07 - displays the result in seconds in the log;

6.3 Results

The measured results take into account the steps shown in figure 6.9. They are related to the execution of the CoAP protocol using DTLS PSK versus DTLS RPK and the overhead in the CHARRA-PM implementation.

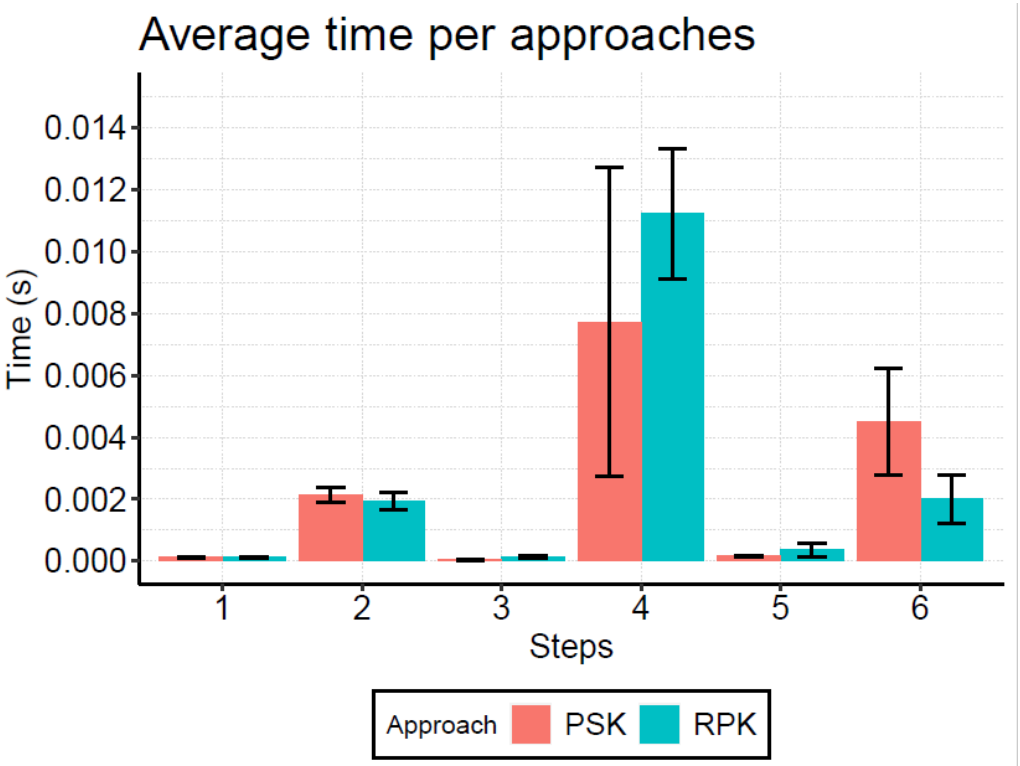

Figure 6.10: Avgerage execution time per aproaches

The total processing time of CHARRA-PM implementation increases the execution time, as expected.

The following considerations are formulated considering the results in figure 6.10 for the diverse steps in the attestation process.

1. The development of CHARRA does not consider the signature of the attestation, the data packaging time and its transmission to the Attester [step 4]. This step, besides having the higher average times also have an higher variation, mainly due to the signature, packaging and transmission processes;
2. Although shown in the appraising results - figure 5.1 [step 6], it was implemented only in the development of the CHARRA-PM;
3. The signature process can be time-consuming and will always run;
4. PSK is faster than RPK when signing but longer when checking the signature [step 6];
5. All other processes and communications are below the order of milliseconds.

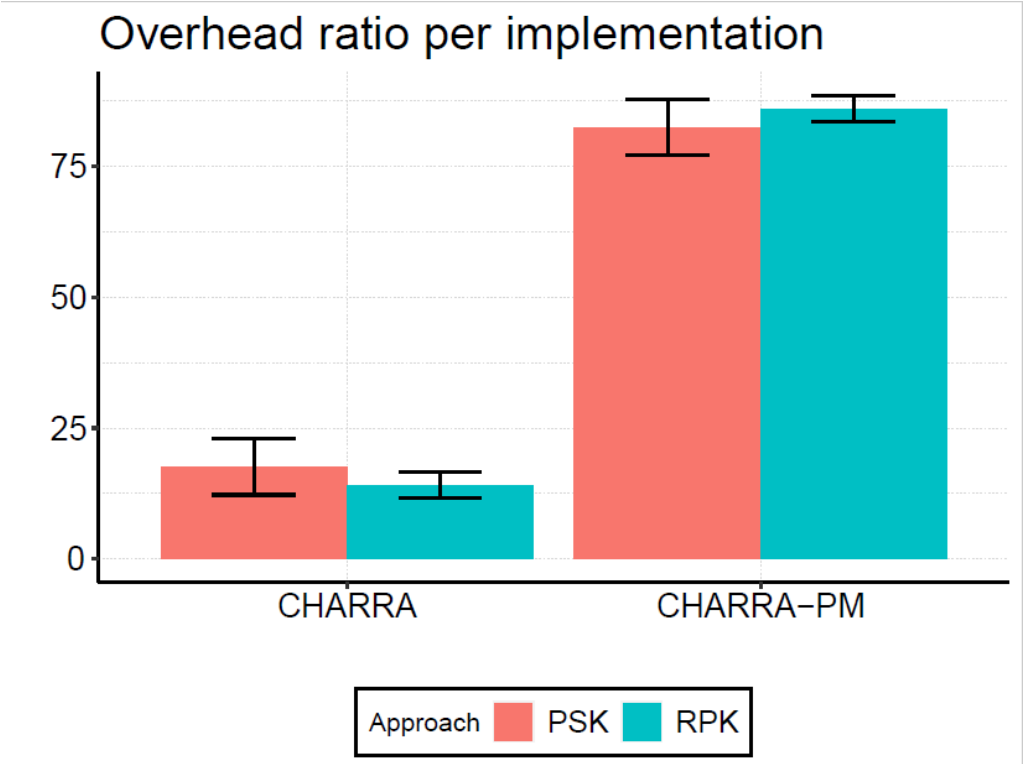

Figure 6.11: Overhead per Implementations

Another observable point is about the total overhead between CHARRA and CHARRA-PM as pictured in figure 6.11. The results put in evidence that CHARRA-PM introduces more overhead, as referred and illustrated. It should be noticed that this overhead is justified by the fact of involving a third entity - relying party to allow the application of policies.

The difference between the use of PSK and RPK is almost negligible, since there is not a pattern. In some steps the RPK takes longer (step 4, 5) in figure 6.10, while PSK takes longer in other step 6. RPK has the advantage of not requiring a full PKI infrastructure [53], and thus might be suitable to scenarios with low complexity, which also is inline with the PSK model.

The measurement table used for constructing the graphics is summarized in appendix C for verification.

Chapter 7

Conclusion

This work presented the implementation of the passport model described in the RATS workgroup documents, which proposed using a trusting third party to grant access to devices that have a certificate validated by a verifying party.

We have seen that the use of TPM chips is an enabler. It has cryptographic and security functions, in addition to allow the storage and retrieval of data in its records securely on the chip. CoAP is a versatile and easy-to-implement protocol that allows to use or not use of a secure transport layer, DTLS. Both were built for use and application in restricted environments, with little processing and network traffic consumption. In addition, we saw the CBOR format, which has the same structures as JSON, only converting the structure to a binary representation, which helps speed encoding/decoding processing.

The technologies used can be replaced by others with the same function, as long as due care is observed. Security, implementation language, and network usage are some examples.

The choice of the Passport Model over the Background-check Model was due to the interpretation of the number of requests and traffic that the Relying Party would have to support in this model, where the Relying Party is the centre of requests, possibly making it a more robust device.

CHARRA-PM can be used as a source of study and knowledge for implementations of domestic IoT networks, where the Relying Party can be software on the router. Tests show that it is necessary to investigate the delay in create the attestation results. We believe that the implementation should be improved, but as a PoC, the times are acceptable since it is one of the essential processes.

As a next step, a compelling extension to this PoC would be the insertion of a policy provider for Attestation Results, the Relying Party Owner, adding more reliability and integrity without adding structural complications to the model.

References

[1] Henk Birkholz, Dave Thaler, Michael Richardson, Ned Smith, and Wei Pan. Remote attestation procedures architecture. Internet-Draft draft-ietf-rats-architecture-15, Internet Engineering Task Force, 05 2022. Work in Progress.

[2] State of iot 2021: Number of connected iot devices growing 9%. https://iot-analytics.com/number-connected-iot-devices/.

[3] Aamir Lakhani. Examining Top IoT Security Threats and Attack Vectors | Fortinet. https://www.fortinet.com/blog/industry-trends/examining-top-iot-security-threats-and-attack-vectors, 2021.

[4] Erik David Martin, Joakim Kargaard, and Iain Sutherland. Raspberry Pi Malware: An Analysis of Cyberattacks towards IoT Devices. *Conference Proceedings of 2019 10th International Conference on Dependable Systems, Services and Technologies, DESSERT 2019*, pages 161–166, 2019.

[5] Remote attestation procedures workgroup. https://datatracker.ietf.org/wg/rats/about/.

[6] Internet engineering task force. https://www.ietf.org/about/who/.

[7] Edlira Dushku, Md Masoom Rabbani, Mauro Conti, Luigi V. Mancini, and Silvio Ranise. SARA: Secure Asynchronous Remote Attestation for IoT Systems. *IEEE Transactions on Information Forensics and Security*, 15:3123–3136, 2020.

[8] Md Masoom Rabbani, Jo Vliegen, Jori Winderickx, Mauro Conti, and Nele Mentens. SHeLA: Scalable Heterogeneous Layered Attestation. *IEEE Internet of Things Journal*, 6(6):10240–10250, dec 2019.

[9] Antonio Marques. Charra passport mode (charra-pm) - source code. https://github.com/aamarques/CHARRA-PM.

[10] ISO-IEC 11889-1:2015. Information technology — Trusted platform module library — Part 1: Architecture. https://www.iso.org/standard/66510.html.

[11] Trusted Computing Group. Trusted Computing Group - Trusted Platform Module - TPM - Latest Version. https://trustedcomputinggroup.org/work-groups/trusted-platform-module.

[12] Microsoft. Trusted Platform Module Technology Overview. https://docs.microsoft.com/en-us/windows/security/information-protection/tpm/trusted-platform-module-overvie.

[13] Will Arthur Challener and David. *A Pratical Guide to TPM 2.0*. Apress Open, 2015.

[14] Professor Norbert Pohlmann. Trusted Platform Module (TPM). https://norbert-pohlmann.com/glossar-cyber-sicherheit/trusted-platform-module-tpm.

[15] Trusted Computing Group. TCG PC Client PlatformFirmware Profile Specification. https://trustedcomputinggroup.org/wp-content/uploads/TCG_PCClient_PFP_r1p05_v23_pub.pdf.

[16] TPM-JS - Learn Trusted Platform Module (TPM) in your browse. https://google.github.io/tpm-js/#pg_pcrs.

[17] J. Boone. Tpm genie: Interposer attacks against the trusted platform module serial bus. *NCC GROUP*, 112:1–23, 2018.

[18] IBM's Software TPM 2.0 - This project is an implementation of the Trusted Computing Group's (TCG) TPM 2.0 specification. It is based on the TPM specification Parts 3 and 4 source code donated by Microsoft, with additional files to complete the implementation. https://sourceforge.net/projects/ibmswtpm2.

[19] Tpm2-tools - the source repository for the trusted platform module (tpm2.0). https://github.com/tpm2-software/tpm2-tools.

[20] Rfc 7252 - the constrained application protocol (coap). https://www.rfc-editor.org/rfc/rfc7252.html.

[21] Rfc 2616 - the hypertext transfer protocol (http). https://www.rfc-editor.org/rfc/rfc2616.html.

[22] Rfc 0768 - user datagram protocol - udp. https://www.rfc-editor.org/rfc/rfc768.htm.

[23] Rfc 9147 - datagram transport layer security (dtls) version 1.3. https://www.rfc-editor.org/rfc/rfc9147.

[24] Rfc 8132 - patch and fetch methods for the constrained application protocol (coap). https://www.rfc-editor.org/rfc/rfc8132.

[25] Rfc 7250 - using raw public keys in transport layer security (tls) and datagram transport layer security (dtls), ietf. https://www.rfc-editor.org/rfc/rfc7250.html.

[26] Rfc 4279 - pre-shared key ciphersuites for transport layer security (tls). https://www.rfc-editor.org/rfc/rfc4279.html.

[27] Mbed tls is a c library that implements cryptographic primitives, x.509 certificate manipulation and the ssl/tls and dtls protocols. its small code footprint makes it suitable for embedded systems. https://github.com/Mbed-TLS/mbedtls.

[28] Rfc 8949 - concise binary object representation (cbor). https://www.rfc-editor.org/rfc/rfc8949.html.

[29] Rfc 7159 - the javascript object notation (json) data interchange format. https://www.rfc-editor.org/info/rfc7159.

[30] RISHABH DEV Founder and Editor at Durofy. Ascii values and table generator in c. https://durofy.com/ascii-values-table-generator-in-c.

[31] Alexander Sprogø Banks, Marek Kisiel, and Philip Korsholm. Remote Attestation: A Literature Review. pages 1–34, 2021.

[32] N. Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. SEDA: Scalable embedded device attestation. *Proceedings of the ACM Conference on Computer and Communications Security*, 2015-Octob:964–975, 2015.

[33] Remote attestation procedures workgroup documents. https://datatracker.ietf.org/wg/rats/documents/.

[34] Henk Birkholz, Michael Eckel, Wei Pan, and Eric Voit. Reference Interaction Models for Remote Attestation Procedures. Internet-Draft draft-ietf-rats-reference-interaction-models-05, Internet Engineering Task Force, January 2022. Work in Progress.

[35] R. Reddy and C. Wallace. Trust anchor management requirements. https://www.rfc-editor.org/info/rfc6024.

[36] MQTT - The Standard for IoT Messaging. https://mqtt.org/.

[37] Home | AMQP. https://www.amqp.org/.

[38] Fraunhofer Institute for Secure Information Technology. Fraunhofer-SIT/charra: Proof-of-concept implementation of the "Challenge/Response Remote Attestation" interaction model of the IETF RATS Reference Interaction Models for Remote Attestation Procedures using TPM 2.0. https://github.com/Fraunhofer-SIT/charra.

[39] Michael Eckel. Michael eckel website. https://eckelmeckel.me/.

[40] Developer community for those implementing APIs and infrastructure from the TCG TSS2 specifications. Repository of metadata and scripts used to generate the container images used by the various tpm2-software projects. https://github.com/tpm2-software/tpm2-software-container.

[41] IBM's Software. Tpm 2.0 - this project is an implementation of the trusted computing group's (tcg) tpm 2.0 specification. it is based on the tpm specification parts 3 and 4 source code donated by microsoft, with additional files to complete the implementation. https://sourceforge.net/projects/ibmswtpm2.

[42] Developer community for those implementing APIs and infrastructure from the TCG TSS2 specifications. The source repository for the trusted platform module (tpm2.0) tools. https://github.com/tpm2-software/tpm2-tools.

[43] Charra opensource bsd-3 license infomration. https://github.com/Fraunhofer-SIT/charra/blob/master/LICENSE.md.

[44] Michael Eckel and Tim Riemann. Userspace software integrity measurement. ARES 21, New York, NY, USA, 2021. Association for Computing Machinery.

[45] Boyu Kuang, Anmin Fu, Willy Susilo, Shui Yu, and Yansong Gao. A survey of remote attestation in internet of things: Attacks, countermeasures, and prospects. *Computers and Security*, 112:102498, 2022.

[46] Sigurd Frej Joel Jørgensen Ankergård, Edlira Dushku, and Nicola Dragoni. State-of-the-art software-based remote attestation: Opportunities and open issues for internet of things. *Sensors*, 21(5), 2021.

[47] Raja Naeem Akram, Konstantinos Markantonakis, and Keith Mayes. Remote attestation mechanism for embedded devices based on physical unclonable functions. *Cryptology and Information Security Series*, 11:107–121, 2013.

[48] , Institute of Electrical and Electronics Engineers. Ieee 802.11, The Working Group Setting the Standards forWireless LANs. [Online; accessed 2022-09-03].

[49] Zigbee specification. https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf.

[50] Research Network Website. Research gate. https://www.researchgate.net/.

[51] Institute of Electrical and Electronics Engineers. Ieeexplore. https://ieeexplore.ieee.org/.

[52] Programa de educação tutorial (pet) da universidade de são carlos, brasil. https://petbcc.ufscar.br/time.

[53] A. Gonzalez Robles. M2m and mobile communications: an implementation in the solar energy industry (dissertation). Technical report, 2015.

Appendices

Appendix A

Protocol Message Exchanges

This appendix will show some connection messages between devices after enabling DEBUG mode. We can see the CoAP, DTLS and CBOR traffic in the figures below. All these figures were screenshots of CHARRA-PM execution.

The figure A.1 and A.2 shows some messages exchanged between Attester and Verifier using CoAP with DTLS-RPK. The figure A.1 also shows the creation of two endpoints: attest and result.

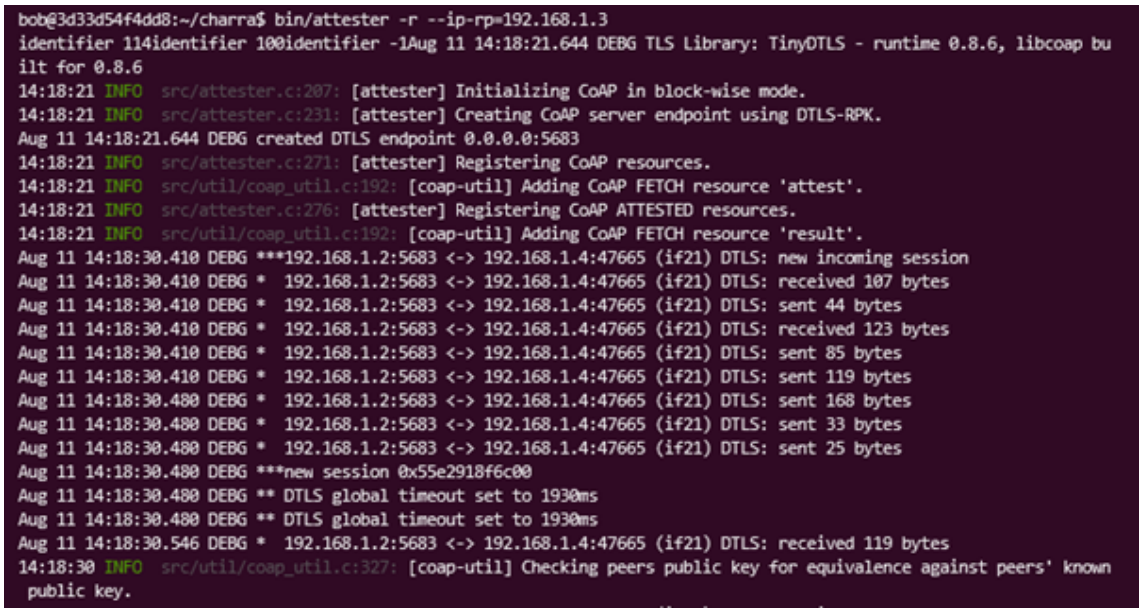

Figure A.1: CoAP with DTLS-RPK

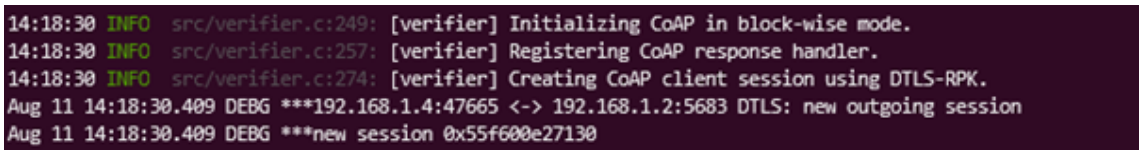

Figure A.2: CoAP with DTLS-RPK

The figure A.3, we can verify the use of the FETCH method receiving a message in "attest" endpoint usig the CBOR format.

In the figure A.4 is a complete CBOR Message with the evidence **sent** by the Attester to the Verifier. Moreover, The figure A.5shows the same message **received** by Verifier.

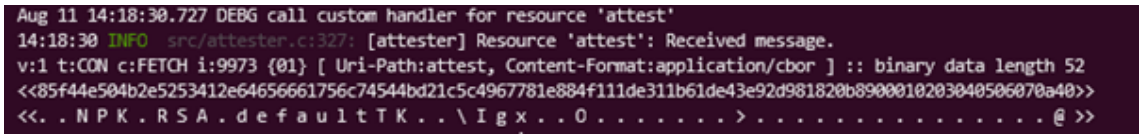

Figure A.3: CoAP with FETCH method

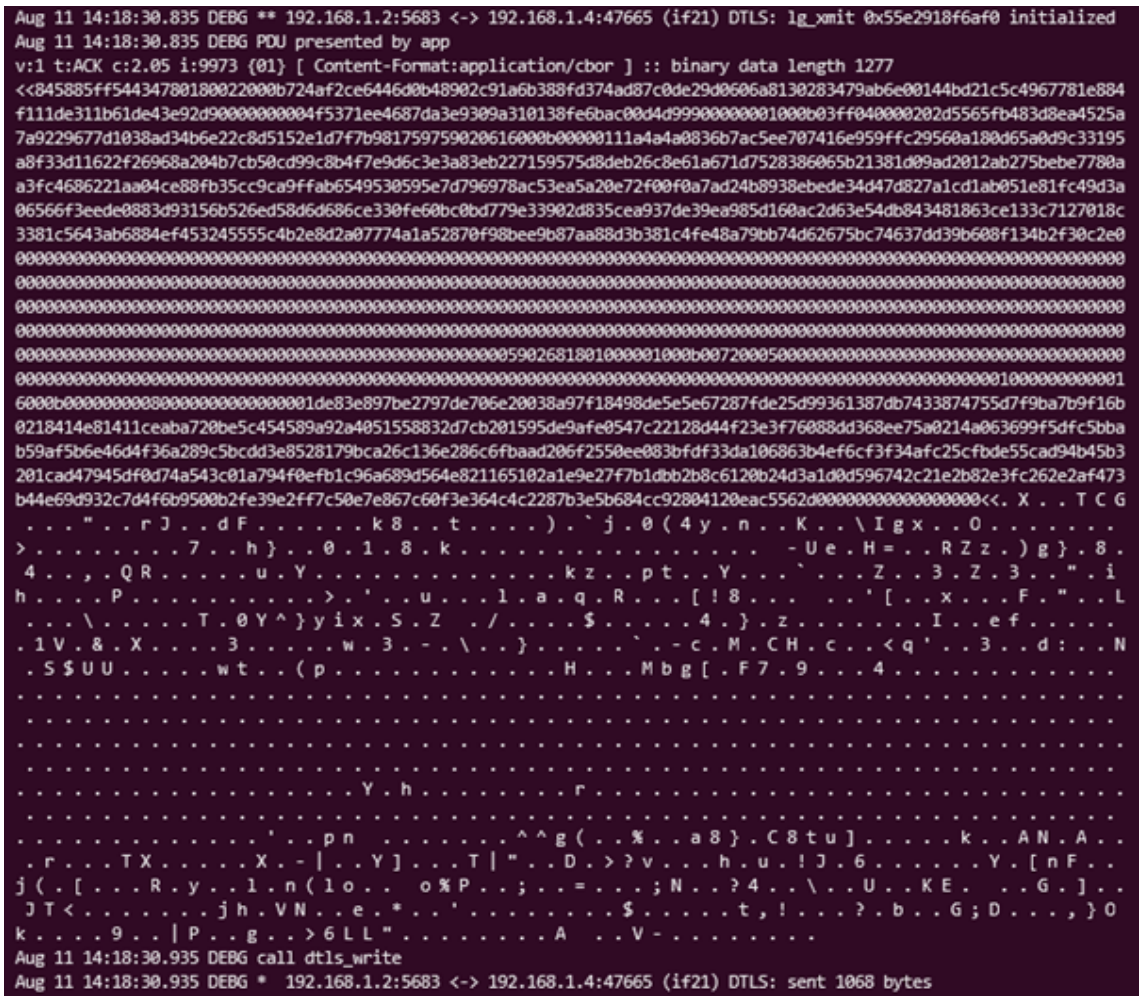

Figure A.4: CBOR complete message sent

The figure A.6 shows the Attestation Result received, encoded in CBOR, by the Relying Party.

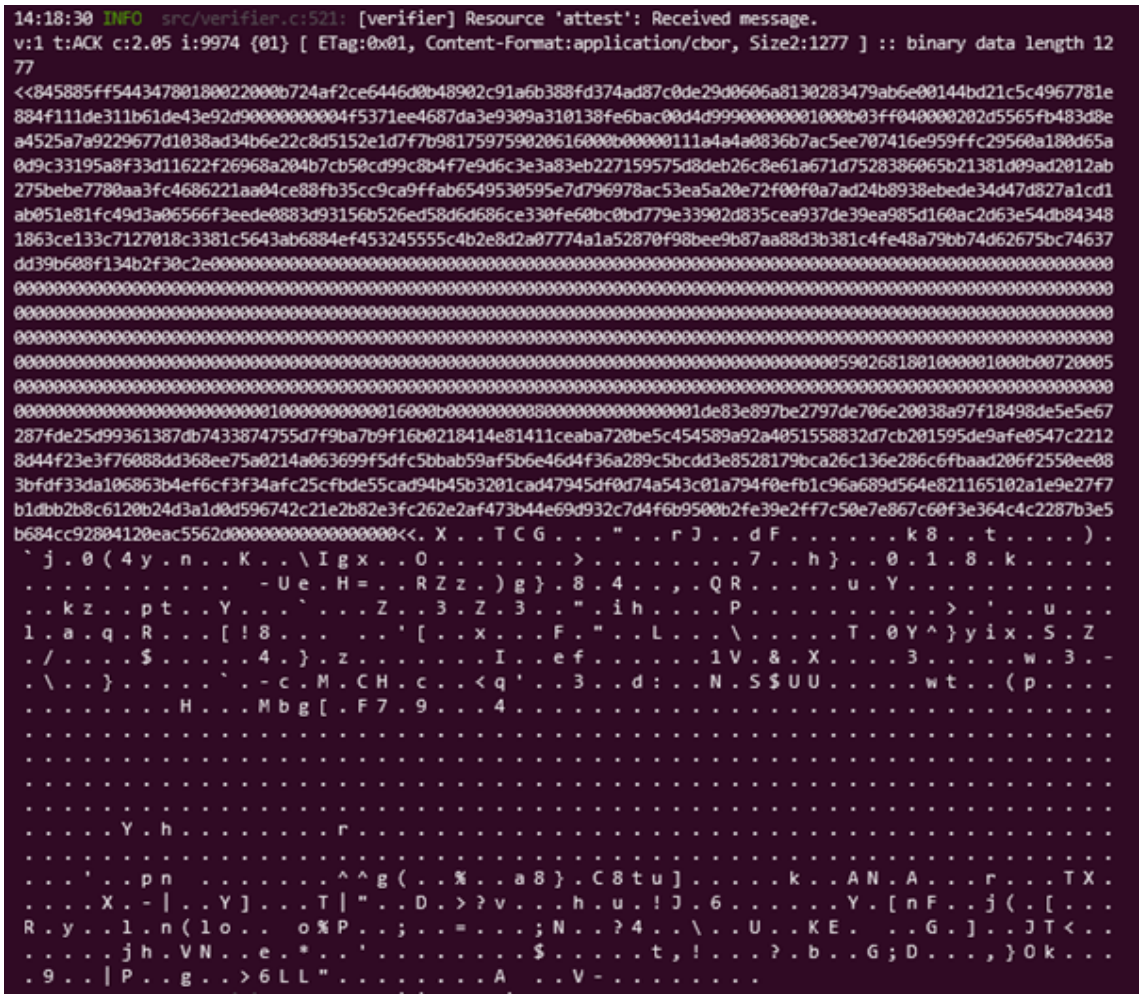

Figure A.5: CBOR complete message received

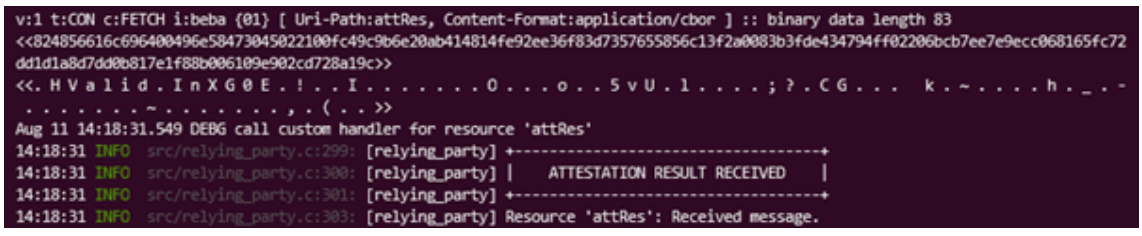

Figure A.6: Attestation Result Received

Appendix B

Command Line Parameter for CHARRA-PM

This appendix shows the option to pass to any program used in CHARRA-PM and/or CHARRA also.

OPTIONS	DESCRIPTION
–help:	Print this help message.
-v, –verbose:	Set CHARRA and CoAP log-level to DEBUG.
-l, –log-level=LEVEL	Set CHARRA log-level to LEVEL. Available are: TRACE, DEBUG, INFO, WARN, ERROR, FATAL. Default is INFO.
-c, –coap-log-level=LEVEL	Set CoAP log-level to LEVEL. Available are: DEBUG, INFO, NOTICE, WARNING, ERR, CRIT, ALERT, EMERG, CIPHERS. Default is INFO.
–ip=IP	local IP address of service.
–port=PORT	Open PORT instead of port 5683.
DTLS-PSK Options:	
-p, –psk	Enable DTLS protocol with PSK. By default the key 'Charra DTLS Key' and hint 'Charra Attester' are used.
-k, –key=KEY	Use KEY as pre-shared key for DTLS. Implicitly enables DTLS-PSK.
-h, –hint=HINT	Use HINT as hint for DTLS. Implicitly enables DTLS-PSK.
DTLS-RPK Options:	
	Charra includes default 'keys' in the keys folder, but these are only intended for testing. They MUST be changed in actual production environments!
-r, –rpk	Enable DTLS-RPK (raw public keys) protocol . The protocol is intended for scenarios in which public keys of either attester or verifier or both of them are pre-shared.
–private-key=PATH	Specify the path of the private key used for RPK. Currently only supports DER (ASN.1) format.
–public-key=PATH	Specify the path of the public key used for RPK. Currently only supports DER (ASN.1) format.
–peer-public-key=PATH:	Specify the path of the reference public key of the peer, used for RPK. Currently only supports DER (ASN.1) format.
–verify-peer=[0,1]:	Specify whether the peers public key shall be checked against the reference public key. 0 means no check, 1 means check. By default the check is performed.
	WARNING: Disabling the verification means that connections from any peer will be accepted. This is primarily intended for the verifier, which may not have
	the public keys of all attesters and does an identity check with the attestation response. Implicitly enables DTLS-RPK.

Table B.1: Parameters of CHARRA-PM

The usage is: binary [OPTIONS], where binary should be attester, verifier or relying_party.

For exemple:

```
attester --ip-rp 192.168.1.5 -r
relying_party -r
```

The table B.1 shows each [OPTIONS] and respective description.

Appendix C

Measurements Data Source

This appendix shows the collected data between five runs of CHARRA-PM for each approach - RPK and PSK. These data were used to plot the graphics in subsection 6.3

Device	Step no.	Results in sec	Run	Approach
Verifier	1	0.000111	1	RPK
Attester	2	0.001770	1	RPK
Attester	3	0.000174	1	RPK
Verifier	4	0.009740	1	RPK
Attester	5	0.000693	1	RPK
Relying P.	6	0.000693	1	RPK
Verifier	1	0.000113	2	RPK
Attester	2	0.002157	2	RPK
Attester	3	0.000115	2	RPK
Verifier	4	0.009363	2	RPK
Attester	5	0.000502	2	RPK
Relying P.	6	0.002596	2	RPK
Verifier	1	0.000120	3	RPK
Attester	2	0.001644	3	RPK
Attester	3	0.000084	3	RPK
Verifier	4	0.014291	3	RPK
Attester	5	0.000258	3	RPK
Relying P.	6	0.002596	3	RPK
Verifier	1	0.000130	4	RPK
Attester	2	0.002284	4	RPK
Attester	3	0.000111	4	RPK
Verifier	4	0.012431	4	RPK
Attester	5	0.000207	4	RPK
Relying P.	6	0.002100	4	RPK
Verifier	1	0.000121	5	RPK
Attester	2	0.001870	5	RPK
Attester	3	0.000152	5	RPK
Verifier	4	0.010247	5	RPK

Device	Step no.	Results in sec	Run	Approach
Attester	5	0.000192	5	RPK
Relying P.	6	0.002017	5	RPK
Verifier	1	0.000148	1	PSK
Attester	2	0.002416	1	PSK
Attester	3	0.000086	1	PSK
Verifier	4	0.012709	1	PSK
Attester	5	0.000184	1	PSK
Relying P.	6	0.004307	1	PSK
Verifier	1	0.000127	2	PSK
Attester	2	0.002327	2	PSK
Attester	3	0.000056	2	PSK
Verifier	4	0.013625	2	PSK
Attester	5	0.000207	2	PSK
Relying P.	6	0.007467	2	PSK
Verifier	1	0.000101	3	PSK
Attester	2	0.001933	3	PSK
Attester	3	0.000027	3	PSK
Verifier	4	0.003875	3	PSK
Attester	5	0.000118	3	PSK
Relying P.	6	0.003970	3	PSK
Verifier	1	0.000111	4	PSK
Attester	2	0.001857	4	PSK
Attester	3	0.000026	4	PSK
Verifier	4	0.003717	4	PSK
Attester	5	0.000132	4	PSK
Relying P.	6	0.003796	4	PSK
Verifier	1	0.000126	5	PSK
Attester	2	0.002192	5	PSK
Attester	3	0.000031	5	PSK
Verifier	4	0.004692	5	PSK
Attester	5	0.000143	5	PSK
Relying P.	6	0.003029	5	PSK

Table C.1: Measurements Table