DPU-LARK: DPU-LEVERAGED ATTESTATION OF REMOTE KERNELS FOR SECURING OT NETWORKS

BY

WILLIAM KOZLOWSKI

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2024

Urbana, Illinois

Adviser:

Professor Deming Chen

Abstract

It is widely recognized that critical infrastructures of nations present a significant and lucrative target for cyber-attacks and threat actors. With numerous real-world examples, such as the attack on the Ukranian power grid in 2015 which caused 225,000 customers to go without power, the need to secure the networks of such systems is more important than ever. This study aims to examine the efficacy of using a smart-NIC, such as the Bluefield-2 DPU, as a solution to enable greater cyber-security in the Smart Grid and other operational technology networks. While studies have been conducted to introduce remote attestation to OT networks, they fail to consider high-speed performance. Presented here is DPU-LARK, the first remote attestation scheme which considers message latency for common OT network messaging protocols. It tests the performance of this scheme using a Bluefield-2 DPU as a verifier, and compares them to solutions using a CPU alone. It also evaluates the performance of the scheme on low-power prover devices (micro-controllers) to determine the effect of the construction on latency, availability, memory, and network traffic. It finds that the DPU is able to outperform expensive, server-grade CPU's with relevant hardware acceleration for high prover counts by up to 1.7x, and match performance at lower counts with much lighter cost. For CPU's without acceleration, the DPU surpasses their performance by 11x-22x. Furthermore, under typical circumstances, DPU-LARK offers 99.9% availability for a 33s attestation period. Furthermore, message validation latency is negligible whenever a device is available.

I would like to dedicate this work to my Mother and Father for teaching me the skills to succeed, to my brother Joey and my good friend Rhylan for pushing me to continue my education, and to my three sisters Elly, Mary, and Katy for their endless support in my endeavors.

Acknowledgments

This work could not have been accomplished without the helpful advice of my advisor, Professor Deming Chen. Without his patience, I would not have been able to complete my thesis. I would also like to acknowledge Professor David Nicol, who provided much needed insight and area expertise, which helped form the direction and details of the research. Finally, I would like to thank CITES, ITI, and the AMD Center of Excellence at UIUC for generously funding this research.

Table of contents

List of Abbreviations

DPU	Nvidia Bluefield-2 Data Processing Unit.
DPDK	Data Plane Development Kit.
OT	Operational Technology.
IED	Intelligent Electronic Device.
MCU	MicroController.
SuV	Security MicroVisor.
IoT	Internet of Things.
RA	Remote Attestation.
NIC	Network Interface Card.

Chapter 1

Introduction

1.1 Motivation

Operational technology (OT) networks are prevalent in all major societies today. From running commercial manufacturing plants to controlling critical infrastructure (water supply systems, power grids, transportation networks, etc.), the performance of these networks (as well as their security) play a crucial role for production companies, economies, and governments alike. In the realm of cyber-security, these networks are extremely high-value targets worthy of any amount of resources to attack. Thus, they are typically considered to have a number of advanced persistent threats (APT's) as adversaries.

There are a few well-known and concerning examples of such adversaries accomplishing their goals, which illustrate just how pivotal the cyber-security of these systems (or lack thereof) can be. One example of such an intrusion is the power station attack in Ukraine. While the precise attack vector is unknown, the culmination of the security breach used a malware called KillDisk to wipe critical files and corrupt master boot records from disks, causing affected systems to become un-bootable. The end-result was power outages in surrounding areas, impacting around 225,000 residences. Perhaps the most notorious among these examples is the Stuxnet worm. Stuxnet was installable on windows operating systems, and could spread to target machines on the network. After spreading it was able to configure programmable logic controllers (PLC's) and control them. This was demonstrated on multiple Iranian sites, one of which was a uranium enrichment plant, where it is reported that Stuxnet caused the PLC's to destroy some centrifuges [1].

The aforementioned examples highlight two useful insights, one of which is a concrete visualization of the devastating impact cyber-attacks on OT networks can have. For instance, consider the former example. While the number of people affected by the power-grid attack is alarming, there are numerous examples of cascading power failures, where relatively small

outages ripple outwards, leading to massive blackouts impacting millions of people. A few such examples occurred in the northeastern United States, impacting about 45 million customers [2], and in northern India affecting almost ten times as many [3]. If an adversary were to execute a well-timed, well-placed attack (say in the middle of winter in New York City), the effects could go from alarming to devastating.

The latter example, on the other hand, helps to show another issue which is becoming ever-more prevalent in modern OT networks: integration with information technology (IT) networks and remote access. OT networks are not a new industry; they have been evolving along with automation in controls systems for quite a while, with the term being coined sometime around 2006. Traditionally, the security of OT networks was based upon a so-called "air gap" or isolation. With this model, when OT networks were not as complex as they are now, they were kept secure by allowing little to no connection of the OT network to other systems or networks. To attack them, an adversary would ostensibly have to physically access the site to interact with it. With the desire to optimize the control systems, however, came the need to monitor more data, store it, and control these systems remotely. The push for greater connectivity, more exhaustive data collection, and ease of access through VPN's significantly increased the attack area for these systems [4]. The Stuxnet worm captures this idea well, and showcases that a threat can now come from many different paths with the result being an intrusion of the OT network.

On top of this, many OT systems are seeing a convergence with the Internet of Things (IoT) and now also incorporate large numbers of Intelligent Electronic Devices (IED's) which function as smart circuit breakers and sensors. Due to cost and power concerns, these devices typically consist of little more than a low-power micro-controller with a few sensors, and a poor capacity to compute staple cryptographic algorithms in a timely manner. Even without this capability the devices are connected to the OT network. Compounded with the fact that certain levels of this network require fast communication, messaging in this environment becomes tricky to secure, leading to even more targets for threat actors [5].

Given the severity of an intrusion, along with the expanding area of attack, it is evident that steps should be taken towards securing these OT networks. In such a large field, there are a myriad of strategies taken to accomplish this task. At higher levels of the network, devices are usually more powerful and messaging latency is not as pertinent. Thus, well-developed cryptographic standards and methods (i.e. symmetric/asymmetric encryption, message authentication) can be applied. When it comes to lower levels of the network, however, choices must be made between performance and cost. Assume we have an IED in the form of a large circuit breaker, for example, which controls a critical portion of the power grid. If this breaker gets a valid message telling it to trip, then it should trip as fast as possible

to avoid damage to critical equipment and propagation of the power surge. On the other hand, if this breaker receives a spoofed message from an attacker to trip, it should be able to ignore the message, as a false trip can be just as damaging. This serves counter-purpose to the previous case, as authenticating each message on this IED would increase the message latency dramatically for many of these devices.

This problem has a few partial remedies, one being the improvement of signature schemes on the devices themselves to meet the stringent latency demands [6], [7]. Such schemes typically struggle to meet timing demands as device computational power is reduced. Other schemes attempt to offload this authentication to bump in the wire (BiTW) devices [8]. With benefits such as modularity from the existing system and separation of secrets, this is a promising line of research.

Orthogonal to this approach, Remote Attestation (RA) is a strategy more commonly found in IoT networks. While signature schemes focus on message authentication, RA focuses on monitoring the network as a whole through validation of the software state of devices [9]. RA constructions consist of trade-offs between hardware costs, device availability, and performance. While this is a commonly studied topic in IoT networks in general, it has not been well explored with respect to OT networks. There have been a number of studies discussing smart grids, however they fail to consider the low-latency message demands present at certain levels of these networks [10], [11], [12], [13].

The research presented here falls under the second umbrella, and will examine the efficacy of implementing an RA scheme which can meet the high performance requirements found at low levels of the OT network. It will leverage inherent network properties, such as a static network configuration, as well as a powerful verifier to reduce the computation needed performed by the IEDs. Lastly, it will make use of a software-based security middleware to provide security for the system, while helping to reduce hardware assumptions.

1.2 Contributions

This study accomplishes a few goals.

- It presents a novel remote attestation scheme, DPU-LARK, to improve cyber-security in OT networks in Section 3.2. This is the first remote attestation scheme to consider stringent message latency requirements in OT networks as a design goal. As such, it is the first implemented RA construction suitable for networks requiring sub-2ms message latency, an important metric for critical infrastructure, demonstrating that attestation at this network level is a promising line of research.

- It evaluates the performance of an advanced Data Processing Unit (DPU) from the verifier side against CPUs differing in cost and attributes in Section 4.1. This is the first RA scheme in which the verifier was implemented entirely on a DPU, completely removing the host CPU's involvement.
- The evaluation in Section 4 and the discussion in Section 5 find that DPU-LARK, and therefore remote attestation in general, is suitable for deployment in low-level OT networks. They also conclude that the DPU can be a suitable replacement for a CPU as a standalone, trusted verifier in remote attestation schemes.

Chapter 2

Background

This chapter will provide background and context necessary to understand the motivation and impact of DPU-LARK.

2.1 OT Networks

As mentioned previously, OT networks are very important to the operation and functioning of a variety of industries, and have become quite complex. A typical industrial site needs to "speak many languages" to effectively communicate with all devices, machines, controllers, and operators. Consider a pump supplying water to a tank somewhere on a plant. A PLC needs to run this pump at specific speeds using a simple current loop, and then communicate back to the whole system (often supervisory data acquisition and control, or SCADA) the speed at which the pump is running. All devices in this system should be able to see this change if needed, and human operators need to see this information in a way they can understand. For efficiency, this pump speed needs to be accessible from workers in an office or from home in emergencies. Aside from this pump, there are many other devices and machines which speak their own languages, and many areas of the network to both attack and secure. The necessity to communicate with machines using different protocols differentiates OT networks from typical IT networks. Thus, basic understanding of the topology of OT networks is important to understand exactly where DPU-LARK fits in. Figure 2.1 depicts the topology of a typical OT network. All connecting lines between machines for our purposes are considered as physical connections. While only a subset of those used in industry, the protocols referenced in the legend demonstrate that lowering in network layers often results in lowering of message abstraction in relation to the Open Systems Interconnection (OSI) network model and messages require smaller latency. Specifically, the top layer (purple) uses typical L4 protocols to interface with the internet. Messages here are not generally considered

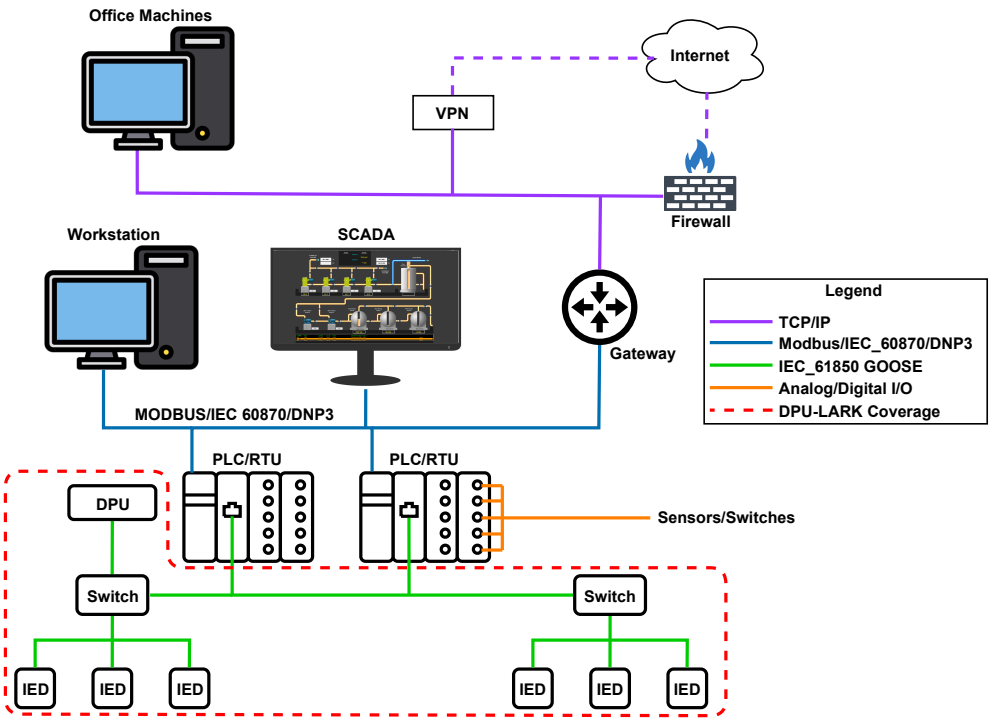


Figure 2.1: A scaled-down depiction of a typical OT network setup. Modified from [15]

to have latency demands other than for convenience of users. The next layer down (blue) uses a mix of protocols ranging from L2-L4, and most latency requirements here lie in the 300-500 ms range if considering IEC 61850 protocols [14]. The lowest layers (green, orange) are either L2 protocols or pure data, and can have much stricter message latency requirements ($< 2\,ms$) [6], [14].

While there is much research in edge security and preventing attackers from ever getting a foothold in the network, precautions still need to be taken in case the first line of defense is passed. Considering the lack of remote attestation research in this area, DPU-LARK will focus on securing this last layer (the area encompassed in red in Figure 2.1). In particular, it will consider the IEC 61850 GOOSE protocol. Found frequently in power grid substations, GOOSE has application data embedded directly into the payload of ethernet frames allowing for low latency message delivery. Given the stringent message latency requirement found at this level of the network, it will be vital for the scheme to have as little impact as possible in this regard.

2.2 Remote Attestation

At its root, remote attestation (RA) is a method of attaining cryptographic evidence about the properties of some party in a network [9]. Given that DPU-LARK is an RA scheme, a few core concepts of remote attestation will be introduced.

2.2.1 Single Prover

All attestation schemes rely on at least two parties, which will be referred to as the verifier and the prover. The verifier begins the protocol with some previously attained knowledge about properties of a prover from a trusted third-party (the state of instruction memory, the boot sequence, or the vm-image used being some commonly used examples). At some point, the verifier will request the prover to "prove", cryptographically, that it exhibits the aforementioned properties. Typically, from the side of the prover, this task requires an environment which cannot be influenced by the prover itself, where the evaluation of the properties can occur. Once the prover has provided its evidence, it is the job of the verifier to determine if this evidence is sufficient to trust the prover. If a prover is determined to be untrusted, a number of actions can be taken ranging from a simple alert to a redeployment of the prover's code [9].

This is the essence of remote attestation between a single prover and a single verifier. There are a couple of characteristics to consider, but RA schemes are commonly categorized into three groups: hardware-based, software-based, or hybrid, depending on how the evaluation environment of the prover is provided [16].

Hardware Based Attestation

Hardware-based attestation is predicated upon the fact that the prover is equipped with special hardware. This can include Trusted Execution Environments (TEE), Trusted Platform Modules (TPM), Physically Uncloneable Functions (PUF), and other hardware. This hardware can be used for things like secure storage of keys, secure execution of the attestation code, and identification of proving devices. Hardware-based schemes usually benefit over software-based schemes from the ability to make more realistic assumptions about adversaries. Hardware-based constructions often use crytptographic hashing mechanisms, such as HMACs, to provide evidence [17], [18].

Software Based Attestation

Software-based RA schemes attempt to make no assumptions about the hardware present on prover devices, changing the approach taken. These schemes ordinarily rely upon probabilistic techniques such as pseudorandom memory traversal. As an example, the verifier will send a nonce along with the attestation request. The prover will then use the nonce to hash corresponding locations of its memory, meaning a malicious prover cannot craft a false response before the request is sent. After the request is sent, a combination of a minimum hashing function and well-known response times are used to ensure that evidence cannot be forged by either the prover itself or another adversary acting as a prover. The minimum hashing function serves to prevent forged evidence being presented in time, as the algorithm is provably the shortest it can be. With less hardware assumptions, software-based schemes make more assumptions about the attacker [19], [20].

DPU-LARK is a software-based attestation scheme, as it will not make any hardware assumptions about the prover aside from the possession of a Microcontroller Unit (MCU).

Hybrid Attestation

Hybrid RA schemes are hardware schemes at heart, but make only the bare minimum assumptions about hardware present on the prover devices. Common requirements are memory protection units and read-only memory, which can be cheaply purchased and attached to most Micro-Controllers (MCU's). With these minor assumptions about hardware, hybrid schemes are able to provide similar levels of security to hardware-based schemes [21], [22].

2.2.2 Swarm Attestation

Swarm attestation schemes are built on top of single-prover attestation, and encompass the attestation of networks as a whole. Swarm schemes, by necessity, take network topology into account. Some schemes consider the network as static, where provers will not move in relation to each other during the attestation period. Such schemes often use a spanning tree method, where provers will form a tree depending on their location, and provers closer to the verifier (root) will aggregate results from those on the leaves and report them up the tree [23], [21].

Other schemes consider the network to be dynamic. Contrary to static network schemes, dynamic network schemes experience a greater degree of freedom when it comes to the network structure. They are also relevant in wireless schemes, as wired networks do not have movement of proving devices very often. As such, a "broadcast"-based scheme is prevalent

in this area, where provers aggregate attestation results based on messages they receive, and propagate results by broadcasting messages. This continues until the verifier is able to communicate with any device to receive the results of the attestation. While flexible in regards to network topology, these schemes exhibit relatively large traffic overhead [24], [25], [16].

Although DPU-LARK considers wired networks, it should be considered a broadcast scheme, as the verifier in DPU-LARK does not need specific knowledge of network topology (i.e. device placements in relation to each other).

2.3 Security MicroVisor

The Security MicroVisor (SuV) is a middleware, deployable onto MCU's, which aims to provide a greater level of security for IoT devices using MCU's with no additional hardware requirements. This is a particularly useful assumption for OT networks where, due to unwillingness to upgrade equipment, outdated and legacy devices are common.

The SuV is implemented as a memory virtualization middleware, and is formally verified to be secure against memory attacks. It operates by dividing the on board memory into a trusted section (MicroVisor memory) and an untrusted section (application memory). Further more, the application memory is divided into instructions and data memory, all of which is enforced by the SuV. MicroVisor memory is not subject to restrictions, and is allowed to modify any other section of memory without bias; however, to maintain formal security, the code lying in the trusted section must be crash-free and memory safe (not subject to attacks related to buffer overflows). This section of memory is reserved for supervisory code and data (e.g. keys, RA code).

The application memory will hold instructions and data for all untrusted software. Here, the security of the scheme does not require such a high standard of code (memory safety and crash-free). Instead, untrusted code is statically analyzed before loading by the SuV to ensure the following:

- Application instructions can only jump to application instruction memory or predetermined entry points to MicroVisor memory.
- Read/write operations can only address untrusted data memory.

Code being uploaded to a platform using the SuV will first go through a post-compilation procedure where unsafe instructions are replaced with safe virtual versions. For example, a jump call to an unknown address may run code placed by the attacker in untrusted data

memory. This untrusted jump is instead replaced by an instruction which first loads the address and checks if it is in a safe location to jump to, and then performs the jump. Following this process, the program to be uploaded to the MCU is analyzed and, if it does not meet the criteria listed above, is rejected for loading [26]. It should be noted that, by necessity, this secure function (instruction memory rewriting) is callable from the untrusted application. Thus, an untrusted application is able to change its instruction memory, but not without consulting the MicroVisor. Figure 2.2 shows the memory architecture for this design.

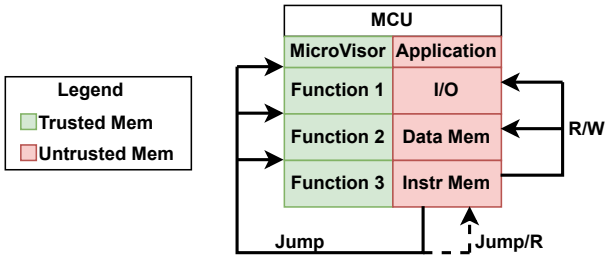


Figure 2.2: Memory Architecture of the Security MicroVisor, modified from [26]

If all conditions are met, the SuV formally guarantees MicroVisor memory to be incorruptible by application memory barring physical access (over-writing the SuV). Seemingly, this provides an interesting use case for remote attestation, as it allows assumptions to be made about security of keys, data, and integrity of code execution similar to hardware-based schemes while remaining purely software-based.

2.4 SmartNICs and the Bluefield-2 Data Processing Unit

Smart Network Interface Cards (SmartNICs) play a vital role in modern computing. Typical NICs do little more than serving packets from the line to the CPU for processing. SmartNICs, on the other hand, allow for the offloading of many common networking related tasks to the interface card itself. Among these tasks are filtering unwanted packets, encrypting packets, performing checksums, inserting or stripping Virtual LAN (VLAN) headers, and many more. The more the SmartNIC is able to do the less the CPU is required to do, freeing it up for other tasks. On top of this, SmartNICs can help prevent denial of service attacks through intelligent packet dropping, providing an extra layer of security.

Data Processing Units (DPUs) build upon this idea by increasing the level of work performed on the NIC. Such platforms retain the fast packet processing time of SmartNICs, but add extra features and hardware accelerators, as well as levels of independence and

modularity, to the design considerations. Unlike many smartNICs which use languages such as P4 for programmability, DPUs often have CPU cores directly on the chip where an operating system can be installed. This allows for more flexible programming of the NICs with languages like C/C++, and pre-made API's to access the on-board accelerators. DPUs also have more compute memory, allowing for these devices to run full-fledged networking applications completely isolated from the server [27] [28].

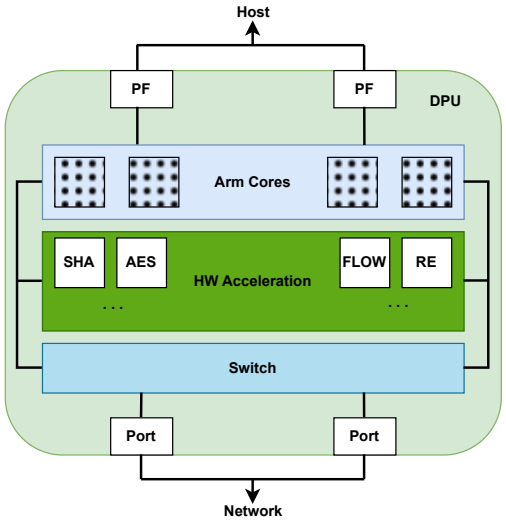


Figure 2.3: A diagram depicting the hardware architecture of the Bluefield-2 DPU. Modified from [28]

To better illustrate this, Figure 2.3 shows the overall hardware architecture of the Bluefield-2 DPU. Packets can come from the two network ports attached to the network, or from the physical functions exposed over PCIe to the host. The ARM cores can run arbitrary programs, as there is an linux-based OS installed. These cores have access to all packets as well as availability to offload tasks to the on-board accelerators using a given C-based API called DOCA. Due to its proximity to the network, along with the possession of relevant Secure Hash Algorithm (SHA) hardware accelerators, the DPU is a good candidate for performing the verifier's role in an RA scheme. On top of this, the host CPU can only access the program space on the DPU via SSH. Thus, without knowledge of a password, a compromised host should still be unable to compromise the verifier.

Chapter 3

Security Preliminaries

To argue for the security of DPU-LARK, the security assumptions and threat model must be established. Here, these foundations will be defined, and an analysis for the security of the scheme will be provided.

3.1 Threat Model

Figure 3.1 shows an overview of the network with the threat model. This scheme defends against adversaries which have remote access to the network, or have established a foothold in an adjacent machine as in [29]. The adversary is given the ability to perform actions such as writing to or reading from any memory locations which allow it. Thus, the adversary can fully control servers on the network, including the servers which have a DPU attached, as well as any application memory of IEDs. The scheme does not consider Distributed Denial Of Service (DDOS) or physical attacks, as is the case with many previous RA schemes. While this scheme will add a level of authentication, it cannot alone provide the necessary primitives to assure message authenticity; however, it should be noted that true message authentication (e.g. based on HMAC schemes) can be added in conjunction with the work provided here. Thus, assumptions for the security of the scheme will be stated below.

3.1.1 Security Assumptions

DPU-LARK makes assumptions similar to many other hardware-based RA schemes, with a few additional assumptions necessary for the validation portion of the scheme. These assumptions are as follows:

1. The proving devices have a capacity to securely hold data such as keys.

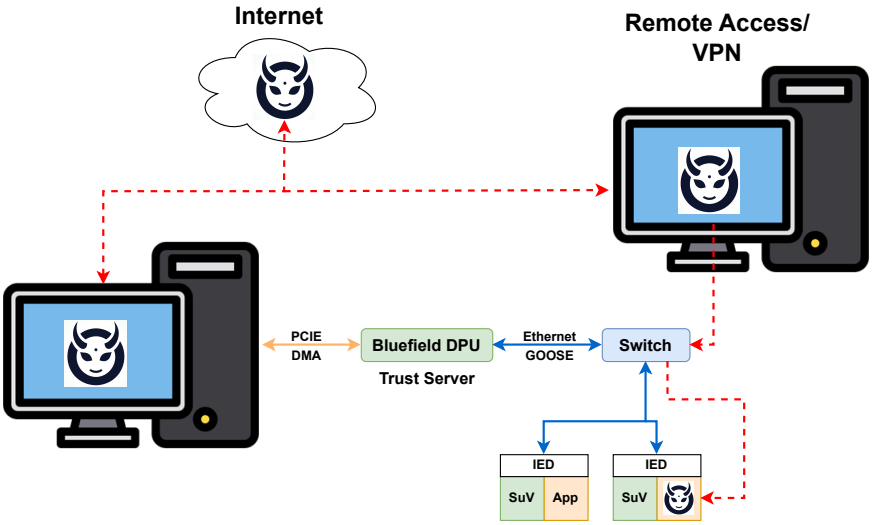


Figure 3.1: A depiction of the threat model and the areas of the network it may effect.

2. The proving devices can guarantee correct execution of remote attestation code.
3. The proving devices have a method of controlling read/write permissions of memory.
4. The adversary does not have physical access to the network or any devices installed there.
5. The ethernet switches must have configurable flow tables.
6. The ethernet switches are considered to be secure.

The first four of these assumptions are typical for hardware-based attestation constructions, which necessitate that remote devices are guaranteed to execute the prover end of the attestation protocol correctly. For the first three assumptions, if hardware is not present on the device to ensure these properties (e.g. a Trusted Execution Environment or Read-Only Memory and a Memory Protection Unit), then they can be added with the Security MicroVisor[1]. As far as the fourth assumption, physical access to such devices opens up devices to side-channel and physical attacks, which are typically considered a separate issue and thus disregarded in RA schemes [26], [21].

The final two assumptions are necessary for the prover-prover authentication portion of the scheme, and are not necessary for the actual remote attestation. These two assumptions could be disregarded if implemented alongside other authentication methods mentioned in Section 1.1. In fact, if dedicated hardware were to be added to IEDs for authentication

[1]If the SuV is deployed, the scheme inherits two more assumptions. First, the device must have a single-threaded processor. Second, the device must allow the disabling of global interrupts.

as in [8], this would prevent the prover devices from having to perform their own message authentication during attestation, improving performance. The ability to configure flow-tables on switches is common, and most decent switches support this feature. Such a feature follows the "deny by default" principle suggested by NIST for use in OT networks [4]. Therefore, this work considers assumption 5 and 6 to be justifiable.

3.2 Canonical DPU-LARK Protocol

The name DPU-LARK was inspired by bird behavior (namely the Eastern Meadowlark common here in Illinois), where a bird calls and others of the same species respond with a similar call. Even species which intentionally mimic other species' bird calls (like the Northern Mockingbird), are not able to fool the mimicked species. Much like the behaviors of birds, RA schemes start with a "call," known as an attestation request, from a verifier and prover devices must respond to prove they are indeed valid. Any attempted "mimicry" of these responses by malicious provers or other devices listening to the attestation request should expose their invalidity to the verifier.

Here, the details of the DPU-LARK protocol will be laid out. As an overview, both the verifier and the provers will be initialized and deployed by a trusted third party. After deployment, there will be an attestation period where provers will provide evidence of their validity. The third state is the active state, where provers run their applications, and can "authenticate" messages from other provers based on the results of the attestation. This authentication is not the same as typical authentication, which cryptographically ensures that a message came from a specific source. Rather, this authentication will mean that the remote prover is running valid software. This evidence, in combination with proper configuration of switch flow tables, gives evidence not only that the message came from the appropriate source, but that the software of the source was not corrupted. This is stronger evidence than typical authentication that the message can be trusted.

After deployment, the prover devices will oscillate between the attestation state and the application state with some predetermined frequency known as the attestation period. For this scheme, it is assumed that a prover corresponds directly (one-to-one) with a device, and that each device is running only one application per attestation. The following symbols are defined for readability:

- $\mathbf{V}$ - The verifier. In DPU-LARK, this role will be played by the bluefield-2 DPU.
- ρ_n - The prover device with id n. These are the IEDs whose memory is being attested by $\mathbf{V}$.

- **A** - The adversary.
- TTP - The Trusted Third Party, whose security is assured.
- α_n - The untrusted application (instruction memory) to be deployed on prover device with id n.
- **K** - Secret symmetric key, used for HMAC.
- MAC_n - Mac address for prover with id n.
- $X_n \rightarrow Y_n$ - Mapping from property X to Y based on id n.
- $meta_n$ - Metadata for a prover, consisting of the validity of the prover, and whether the information is final for this attestation period.
- $\ll$ - The bitwise leftshift operation.

3.2.1 Initialization

The initialization phase occurs prior to deployment of devices. In this phase, it is necessary for the TTP to have explicit knowledge of the applications to be deployed on all provers. Each prover ρ_n will be assigned a unique id n, and this id will be mapped to a static mac address as well as the application α_n for that prover[2]. The id $n = 0$ will be assigned to the verifier. Also, a symmetric key **K** will be generated. In the secure memory of each ρ_n the following data will be installed by the TTP

- Symmetric key **K**
- Id n
- The prover side of the RA protocol
- A counter initialized to 0
- The mapping $(MAC_m \rightarrow m, meta_m)$ for each $m \neq n$ on the network[3]

The application α_n will be installed into non-secure memory on each ρ_n.

For the verifier **V**, the TTP will install the following:

[2]It is possible, and typical, for α_n to be identical for different ρ_n

[3]For memory consideration, it is not necessary for each ρ_n to have this mapping for every other ρ_m. It is only necessary for each ρ_n to have this mapping present for the verifier and each other ρ_m that it can receive communications from. For many devices, specifically non-powerful IEDs, this number should be quite small.

- **K**
- The mapping $(MAC_n \rightarrow n, \alpha_n, meta_n)$
- A counter initialized to 0

Apart from these the flow table of each switch in the network needs to be configured to allow only traffic from expected src MAC addresses from each port. Aside from following recommended whitelist security practices, this allows for an authentication scheme to be added based on the attestation of the network.

3.2.2 Attestation

After initialization, the network will periodically experience attestation which begins with an attestation request. This request will be prepared and sent by the verifier according to Function 1.

Function 1: Verifier:Attestation Request

Data: *ctr*

for $i \leftarrow 1$ **to** n_{max} **do**
    $meta_i \leftarrow 0$; `/* Sets validity of provers to False */`
$nonce \leftarrow rand()$;
$keyword \leftarrow req\_const$;
$att\_req \leftarrow ctr|nonce|keyword$;
$broadcast(att\_req)$;

The *ctr* variable, initialized to 0, is incremented by one after each attestation period. This, in combination with the *nonce* (a value only used once during the duration of the scheme), serve to prevent replay attacks of HMAC authenticated messages. The keyword is a predetermined constant, to allow provers to determine what type of message they are receiving from the verifier. The message is constructed as a concatenation of these values, and broadcasted over the network. There is no need to sign this message since, according to security assumptions 5 and 6, the switches will drop spoofed messages with the same source MAC address as the verifier[4]. Upon receiving this attestation request, each prover device will perform Function 2 within secure memory, in order to provide the attestation response.

Function 2 runs on each ρ_n upon receipt of a message from **V** with the keyword *att_req*. First, as in Function 1, the prover must reset the metadata associated with each other relevant

[4]A statically configured flow table uses rules to associate a specific source address with a specific port on the switch, and packets must match a rule to be forwarded. Thus, if a packet with the same source address as the verifier arrives from the incorrect port, it will be dropped.

Function 2: Prover:Attestation Response

Data: $att\_req, \mathbf{K}, n_{self}, (MAC_n \rightarrow n, meta_n)$

1 **for** $i \leftarrow 1$ **to** n_{max} **do**

2 $\quad meta_i \leftarrow 0$; `/* Sets validity of relevant provers to False */`

3 $ctr \leftarrow att\_req.ctr$;

4 $nonce \leftarrow att\_req.nonce$;

5 $keyword \leftarrow resp\_const$; `/* Predetermined constant to signal an attestation response message */`

6 $memory\_state \leftarrow hmac_{SHA256}(\mathbf{K}, instruction\_memory)$;

7 $att\_resp \leftarrow n_{self}|ctr|nonce|keyword|memory\_state$;

8 $sig \leftarrow hmac_{SHA256}(\mathbf{K}, att\_resp)$;

9 $send(verifier, att\_resp|sig)$;

remote prover. This ensures that ρ_{self} will not authenticate a message from a ρ_{remote} without evidence from **V**. The prover then updates *ctr* and *nonce* based on the information from *att_req*. The most important piece is the actual attestation in Function 2.6, performed as a SHA-256 HMAC of the untrusted application's instruction memory. Finally, the prover uses the same symmetric key **K** to sign the attestation response, and send it back to **V**

The final portion of the attestation period is the validation performed by the verifier, shown in Function 3.

Function 3 is not complex; it simply ensures that the attestation response received was from a prover with valid instruction memory, and was able to correctly sign the attestation response. In this case, contrary to the attestation request, message authentication is necessary due to replay attacks. Since **K** does not change between iterations of the protocol, if the attestation responses were not authenticated, any device on the network could spoof a valid response without consulting the secure memory simply by reusing the *memory_state* located in the last attestation response. The *nonce* and *ctr*, along with HMAC-based message authentication, work to ensure that the secure memory (MicroVisor) in each prover device is actually performing attestation.

3.2.3 Active Period

This is the state of normal operation for the network. Prover devices will be considered in this state whenever they are not computing an attestation response. During the active period, there will be typical network traffic between devices (only L2 traffic will be considered for this scheme). During this period, each prover device needs to accomplish two tasks. 1) Establish trust of other devices it can receive messages from, based on the trust of only **V**. 2) Demonstrate this trust to the untrusted application deployed on the device, allowing the

Function 3: Verifier:Prover Validation

Data: $ctr, nonce, att\_resp, \mathbf{K}, (MAC_n \rightarrow n, meta_n, \alpha_n)$

```
if att_resp.ctr ≠ ctr then
    meta_n.valid ← false
else
    if att_resp.nonce ≠ nonce then
        meta_n.valid ← false
    else
        if att_resp.memory_state ≠ hmac_SHA256(K, α_n) then
            meta_n.valid ← false
        else
            AR ← att_resp;
            sig ← hmac_SHA256(K, AR.n|AR.ctr|AR.nonce|AR.keyword|AR.memory_state);
            if att_resp.sig ≠ sig then
                meta_n.valid ← false
            else
                meta_n.valid ← true
```

application to receive messages from trusted devices.

Establishing Trust

The primary method for a prover to establish this trust is via attestation update messages from **V**, which can come in two forms. The first is SystemStatusValid. This message is broadcast from **V** upon validating all provers on the network, and indicates that all provers are in a valid state. Upon receiving this message, each prover will run Function 4. This

Function 4: Prover:All Provers Valid

Data: $MAC_n \rightarrow n, meta_n$

```
for i ← 1 to n_max do
    meta_i.valid ← 1;
```

function simply iterates through the stored mapping of each prover id and updates the metadata for each one, indicating that messages from these MAC addresses can be trusted.

The second method will be from periodic SystemStatusUpdate messages broadcasted from **V**. The purpose of these messages are twofold: 1) To ameliorate the delay from the time an attestation request is sent to the time a device can accept a message from another device.

2) To reduce the network traffic[5]. The SystemStatusUpdate messages consist of a list of bits, with each bit indicating the validity of the device with correlated id. For example, assume the network has 128 prover devices. The valid list will be 128 bits long. A 1 in the least and most significant bits indicates that ρ_1 and ρ_{128} are valid. Upon receipt of SystemStatusUpdate messages, provers will perform Function 5. This function iterates through the MAC addresses

Function 5: Prover:System Status Update

```
   Data: status_update, (MAC_n → n, meta_n)
1  for i ← 1 to n_max do
2      if status_update.valid_list & (1 ≪ i − 1) then
3          meta_i.valid ← 1;
```

on a prover, and updates their validity if found in the SystemStatusUpdate message.

After a predetermined timeout, **V** will send a SystemStatusFinal message. As a side note, only one of SystemStatusFinal or SystemStatusValid will be sent within one attestation period. The SystemStatusFinal message is structured in the same manner as a SystemStatusUpdate message, but it indicates that any provers which have not passed the attestation can no longer be trusted, and their messages should be discarded. Provers will run Function 6 after receiving this message. This is similar to Function 5, but it updates an extra piece

Function 6: Prover:Final System Status Update

```
   Data: status_update, (MAC_n → n, meta_n)
1  for i ← 1 to n_max do
2      if status_update.valid_list & (1 ≪ i − 1) then
3          meta_i.valid ← 1;
4      else
5          meta_i.final ← 1 ;  /* Metadata indicating a message source will not
            become valid.  */
```

of metadata ($meta_n.final$) for non-validated devices indicating that this prover device no longer has a chance to pass validation. Thus, messages from the device with the associated MAC address should not be trusted.

Authenticating Messages

To this point, all considerations have been towards the secure portion of memory stored in the MicroVisor. The untrusted applications, however, make up the functional portion of

[5]If a device receives a message from another device and the valid bit is not set, it must send an Update Request to **V**. More frequent SystemStatusUpdate messages will reduce the amount of these messages.

the network. Therefore, the final portion of the scheme will be introduced. In this domain, the consideration is from the view of the application running in untrusted memory. When this application receives a message from another device, it will run Function 7 lying in the trusted portion of its memory. This function will return valid only if the remote device has

Function 7: Prover:Remote Device Authentication

```
Data: MAC, (MAC_n → n, meta_n)
1  meta ← (MAC_n → n, meta_n){MAC} ;     /* Inverse the mapping to retrieve
     metadata of device with associated MAC address */
2  if meta.valid then
3  |  return valid;
4  else
5  |  if meta.final then
6  |  |  return invalid;
7  |  else
8  |  |  if ¬meta.requested then
9  |  |  |  send(V, update_req);
10 |  |  |  meta.requested ← 1;
11 |  |  return invalid;
```

been validated by **V** in the current attestation period. In this case, the application has evidence that **V** trusts the memory state of the remote device. In combination with flow table configuration to prevent message spoofing, the receiving device can trust this message. If the sending device is not valid, there are two cases, the first of which means the device has failed to attest itself within the timeout, and can no longer be trusted. Messages will never be trusted from this device. In the last case shown in Function 7, the sending device may still be valid, but the receiving device may not have received a SystemStatusUpdate in time. In this case, the device will send an update request message to **V**. When the verifier receives such a request, it will broadcast another SystemStatusUpdate message. The prover will send this update request only one time per period to reduce potential network traffic.

Thus, the DPU-LARK protocol is defined. In the initialization and attestation periods, the verifier and provers use remote attestation techniques to ensure that the untrusted memory on each prover device is in a valid state. Then, during the active phase, applications are able to query their associated trusted memory to determine the safety of incoming messages with low latency.

3.2.4 Improvements

An improvement to this scheme can be made quite simply and regards the fact that, in order for instruction memory to change, a function call must be made to the MicroVisor to write data to the untrusted instruction memory. This means one more variable, *mem_changed*, can be added to the scheme, which will track whether or not this function has been called. When the memory writing function is called, it will set *mem_changed*, as well as broadcast an *att_update* message. When other provers receive this message, they will set the associated *meta.valid* to 0, and the verifier will initiate the attestation protocol with only the device which sent the *att_update*. If *mem_changed* is not set when the prover receives an *att_req*, the prover can re-use the attestation calculation from the previous iteration of the attestation protocol, meaning it needs to perform HMAC only on the attestation response for signing rather than the entirety of untrusted memory, saving much computation and improving total network attestation time as well as availability. This will be the case for the vast majority of the time, as it is highly unusual for applications to overwrite their own instruction memory. This is an important novelty of the scheme, as it removes the vast majority of SHA computation, a very expensive algorithm on resource-constrained devices. Function 8 shows the improved version of Function 2.

Function 8: Prover:Improved Attestation Response

Data: $att\_req, \mathbf{K}, n_{self}, (MAC_n \rightarrow n, meta_n), mem\_changed, prev\_mem\_state$

for $i \leftarrow 1$ **to** n_{max} **do**
    $meta_i \leftarrow 0$; `/* Sets validity of relevant provers to False */`
$ctr \leftarrow att\_req.ctr$;
$nonce \leftarrow att\_req.nonce$;
$keyword \leftarrow resp\_const$; `/* Predetermined constant to signal an attestation response message */`
if *mem_changed* **then**
    $memory\_state \leftarrow hmac_{SHA256}(\mathbf{K}, instruction\_memory)$;
else
    $memory\_state \leftarrow prev\_mem\_state$;
$att\_resp \leftarrow n_{self}|ctr|nonce|keyword|memory\_state$;
$sig \leftarrow hmac_{SHA256}(\mathbf{K}, att\_resp)$;
$send(verifier, att\_resp|sig)$;

3.2.5 Security Analysis

The security of this scheme will be shown here. To begin, the game will be between an adversary **A** and a prover ρ. **A** will be considered victorious in this game if ρ accepts a

message from a compromised prover ρ'. The capabilities of $\mathbf{A}$ are to spoof messages from any source MAC address, and to fully compromise any prover. The first of these attacks, message spoofing, is thwarted by security assumption 5 and 6 in Section 3.1.1 in the following manner. Static flow tables map a switch port to a source MAC address. Thus, if a message is received on a port with a source address which does not match the flow table configuration, that message will be dropped by the switch. This means a device cannot spoof messages from a source MAC address which is not its own without those messages being dropped by the switch.

For the second attack, the malicious message from ρ' will only be accepted by ρ if ρ has received a message from $\mathbf{V}$. Therefore, there are two attack vectors to be taken by ρ'. The first method is to alter the untrusted memory without knowledge of the verifier and other provers. Based only on the security of SuV, which is formally verified, the attacker must make a call to its secure memory to rewrite its own instruction memory. In DPU-LARK, the instruction is modified to broadcast a message to all devices, indicating it can no longer be trusted. Thus, any message sent by this newly modified ρ' will be received only after ρ' has already been invalidated by the provers, meaning they will reject all messages from ρ'. Now, in the eyes of ρ, ρ' can only become valid again by receiving a message from $\mathbf{V}$, necessitating that ρ' must fool $\mathbf{V}$. This is discussed next.

Th other attack vector, then, is for ρ' to fool $\mathbf{V}$ that its attestation is correct during an attestation period. The attestation response, however, is authenticated via HMAC-256 with a symmetric key stored in secure memory on prover devices. Since each attestation period has a new counter, as well as a unique nonce, any party on the network will only be able to authenticate their messages if they possess the secret key based on the security of SHA256. Thus, the security of this scheme can be again leveled to the security of the SuV in hiding the symmetric key.

Chapter 4

Evaluation

In this chapter, the DPU-LARK protocol will be evaluated. First, the verifier portion of the protocol will be evaluated, followed by the prover portion of the protocol. The important considerations for the verifier will be, mainly, time to attest the whole network. For the prover side of the protocol, the important consideration will be availability and maximum/average message latency overhead. All results and their implications will be further elaborated on and discussed in Section 5.

4.1 Verifier

The verifier will be tested in three environments: 1) Running on the Bluefield-2 DPU which supports SHA hardware acceleration. 2) Running on a powerful CPU, specifically an AMD EPYC 7V13 64-Core Processor, which also supports SHA accleration. 3) The verifier running on a less powerful CPU, an Intel Xeon CPU E5-2630 v4, which does not support SHA hardware acceleration.

4.1.1 System Setup

Figure 4.1 shows the experiment setup for scenarios 1) and 2). In order to experiment on this system, a basic simulator application was developed. A simulator was chosen over a true system setup as it was infeasible to attach hundreds of IEDs to the available network. Shown on the right side of Figure 4.1, the simulator ran on a DPU and was able to act like any number of prover devices. It accomplished this by receiving an attestation request, and responding with the predetermined number of attestation responses. It never performed true memory attestation; rather, it was given a precalculated HMAC of the application to be running on the provers, computed the HMAC-based signature of each attestation response,

and sent all attestation responses. The aforementioned simulator ran on a Bluefield-2 DPU. Here, attestation responses were all constructed, signed, and stored in a large buffer before being sent out at varying line rates. Apart from the prover simulator, the experiments were carried out in a real network testbed in which there were two host machines, each with a DPU attached. The verifier was implemented on one of the DPUs, and communicated with the other DPU (running the prover simulator) over a fiber-optic link.

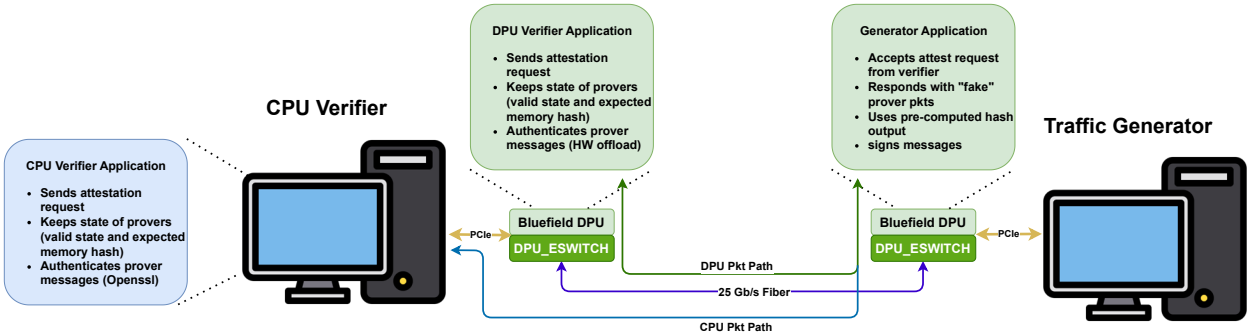


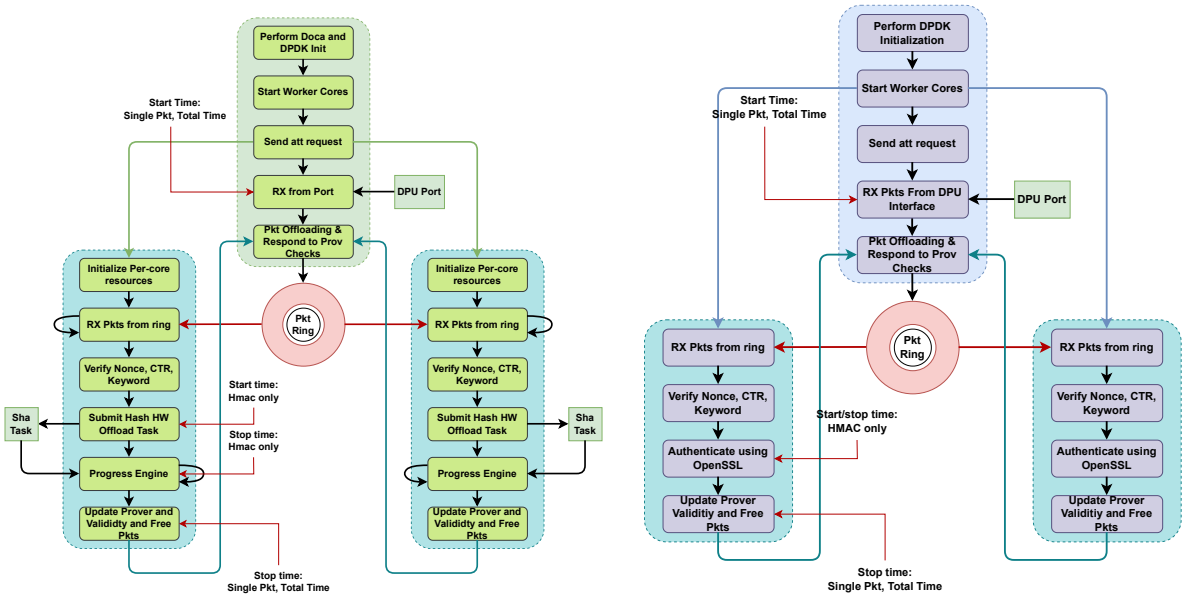
Figure 4.1: A Depiction of the System Setup for Verifier Evaluation: Scenarios 1 & 2

The left side of Figure 4.1 depicts the two verifier application evaluations. In setup 1), the verifier application is running on the ARM cores located on the Bluefield-2 DPU. During this scenario, the packet path is shown in green and source or destination of packets is always the DPU ARM cores. From here, packets are either sent or received over fiber, and offloaded to SHA256 hardware accelerators when applicable (during message authentication). For setup 2), packets follow the same path but the DPU is configured to "NIC mode", where packets are directly passed through the DPU's on-board switch to the physical functions (these expose the DPU ports to the host CPU), and bypass the DPU's ARM cores entirely. Now, the CPU verifier application can handle them in a similar manner to the DPU. Thus, the same network link was used for these scenarios. For setup 3), the same experiment was run as setup 2), but used a more comparable CPU (price-wise to the DPU) which did not have SHA256 hardware acceleration. This helped to determine if DPUs, such as the Bluefield-2, are a good "drop-in" solution over a server.

4.1.2 Experiments

In order to evaluate the verifier portion of the protocol and understand the results, four experiments were performed. Each experiment was performed at least 5 separate times, and results are averages. Figure 4.2 shows a general software flow of the verifier application running on the DPU and the CPU, as well as where timers are started and stopped. 1) For the first experiment, the time to authenticate each message and check that its memory state

is valid is measured. A separate timer is started when each attestation response is received and stopped when the prover who sent the response has been validated. Results are shown in Figure 4.3.



(a) DPU (b) CPU

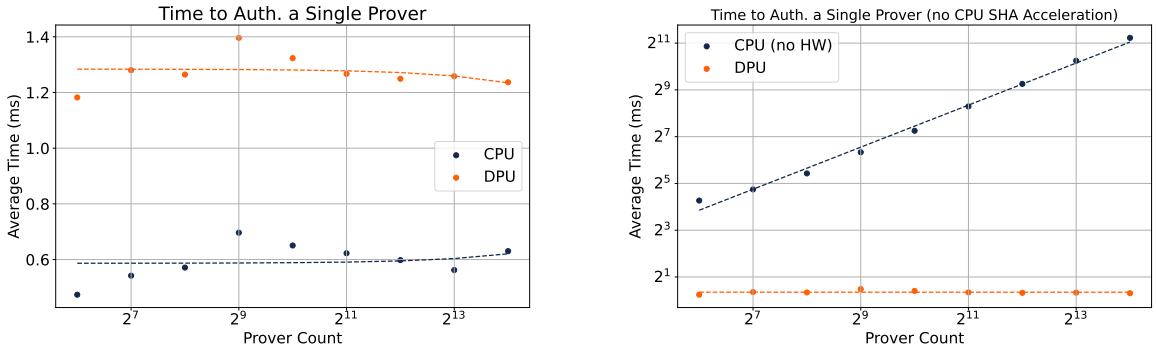
Figure 4.2: An Overview of the Software Flow of the Verifier Application

2) The second experiment determines the total time to validate all prover devices, assuming each prover sends its messages within a small enough time frame. Here, the timer is started when the first attestation response is received, and stopped when the final prover has been validated. Results are shown in Figure 4.4.

3) The third experiment is similar to the first one, but now only the time to authenticate each response message, being the most computationally complex portion of the protocol, is measured. This will help to determine the reason for various other results. 4) The fourth and final experiment simply measures the time to perform a single SHA256 hash operation for varying input data sizes. This will further clarify other results and the reason behind them. For each experiment, there will be a comparison between the DPU setup and both of the CPU setups. Results for these experiments are presented in Figures 4.5 and 4.6 respectively.

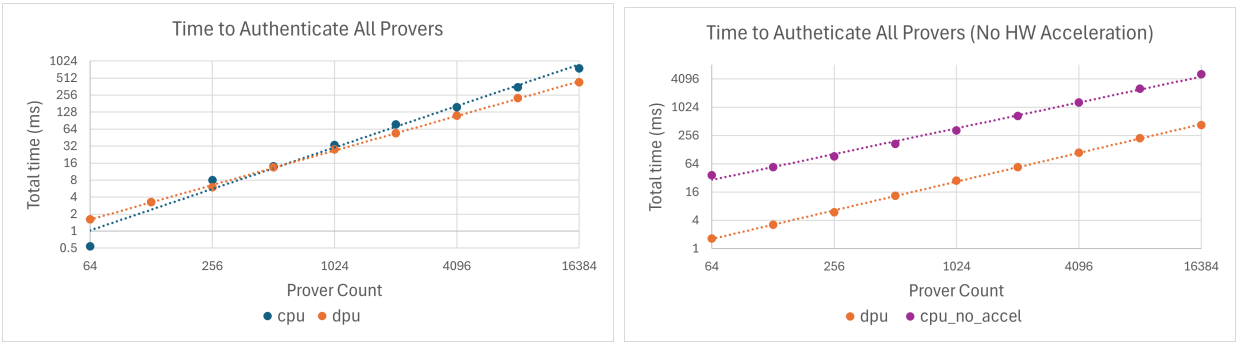
4.1.3 Results

In Figure 4.3, it can be seen that the time to validate a single prover device is about twice as slow for the DPU than for the powerful, accelerator enabled CPU. As for the CPU without SHA acceleration, the DPU has a significantly lower time to validate a prover than the CPU.



(a) CPU HW SHA Acceleration (b) No CPU HW SHA Acceleration

Figure 4.3: The results of experiment 1, comparing a DPU verifier (orange) to a CPU verifier with SHA acceleration (left blue) and w/o SHA acceleration (right blue)



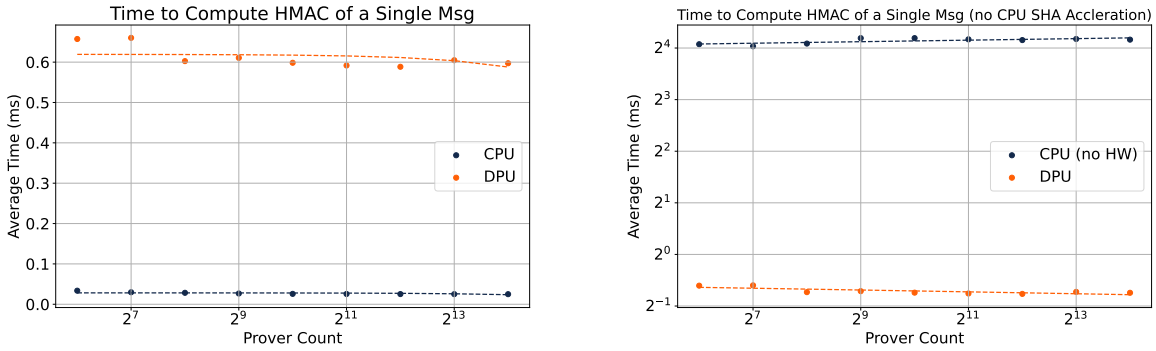
(a) CPU HW SHA Acceleration (b) No CPU HW SHA Acceleration

Figure 4.4: The results of experiment 2, comparing a DPU verifier (orange) to a CPU verifier with SHA acceleration (left blue) and w/o SHA acceleration (right magenta)

Figure 4.4 shows the comparison of the time to validate all provers on the network. These results show that, in order to validate all provers, the DPU performs much better than the CPU without SHA acceleration, and also outperforms the CPU with SHA acceleration by an amount proportionally increasing with the number of prover devices. While these results seem counter-intuitive to the findings of experiment 1, they can be explained by memory bound differences of the SHA hardware accelerators on the DPU and the CPU.

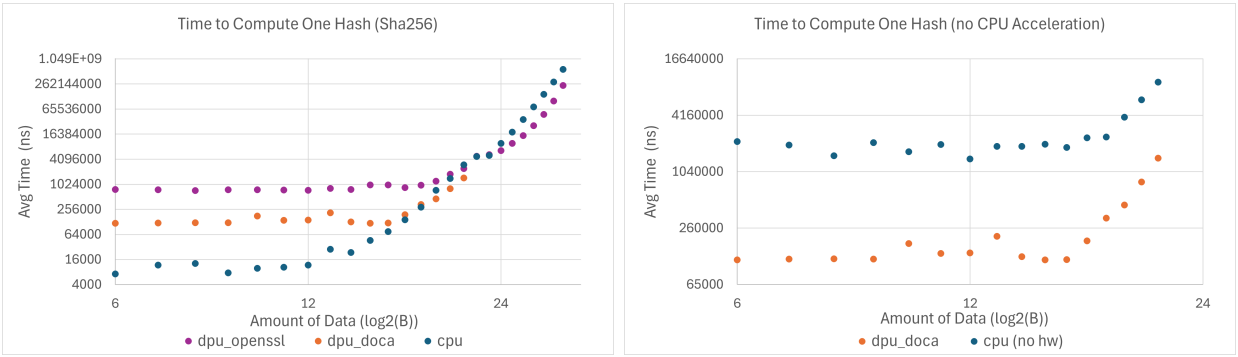
Presented in Figure 4.5 are the results of experiment 3, which compare the performance of only the message authentication portion of the verifier protocol on the various setups. The results of this experiment help to describe the difference in the results of experiment 1, and it is made obvious that most of the difference in the time to validate a single message lies in the time taken to perform the HMAC for a single message.

Lastly, Figure 4.6 shows the results of a simple benchmark, which tested the time to perform one SHA256 hash with input data of varying sizes. In Figure 4.6.a, notice there are



(a) CPU HW SHA Acceleration (b) No CPU HW SHA Acceleration

Figure 4.5: The results of experiment 3, comparing a DPU verifier (orange) to a CPU verifier with SHA acceleration (left blue) and w/o SHA acceleration (right blue)



(a) CPU HW SHA Acceleration (b) No CPU HW SHA Acceleration

Figure 4.6: The results of experiment 4, comparing a DPU (orange) to a CPU with SHA acceleration (left blue) and w/o/ SHA acceleration (right blue)

two sets of data associated with the DPU. This is due to the fact that the bluefield-2 DPU offers two methods of offloading SHA hardware acceleration. One is through an OpenSSL offload engine (magenta) and the other is through the bluefield-2's proprietary API called DOCA (orange). Due to the superior performance of the DOCA offloading, this is the method used for all other experiments. As can be seen in Figure 4.6.a, DPU SHA acceleration does not begin to outperform the CPU SHA acceleration until the input data becomes relatively large (about 260 kB). As for the right, the DPU outperforms the CPU with no HW acceleration significantly. Given that the size of attestation response messages are 108 bytes, this reult helps to explain the slower single message authentication time of the DPU compared to the more powerful CPU, as the DPU hardware accelerator seems to have been designed with a higher memory bound.

Table 4.1 shows the speedup of the DPU running the verifier algorithm over the both CPU's. For these calculations, the speedup was calculated as $CPU_{t\_attest}/DPU_{t\_attest}$, where

Table 4.1: A comparison of the Total Time to Validate All Provers in the Network for Different Verifiers

Time to Validate All Provers					
# of Provers	**CPU (no HW) (ms)**	**CPU (ms)**	**DPU (ms)**	**Speedup (no HW)**	**Speedup**
64	36.551	0.540	1.629	22.440	0.331
128	53.929	3.248	3.247	16.607	1.000
256	92.743	8.007	5.993	15.474	1.336
512	170.615	14.038	13.444	12.691	1.044
1024	327.806	33.580	28.034	11.693	1.198
2048	665.635	77.929	53.990	12.329	1.443
4096	1285.049	156.400	110.075	11.674	1.421
8192	2555.688	351.095	225.203	11.348	1.559
16384	5143.783	758.835	429.503	11.976	1.767

t_attest is the total time to validate all provers on the network, taken from the results of experiment 2. The DPU beats the CPU without SHA acceleration at any prover count by 11x-22x, and matches or beats the accelerator-enabled CPU at all prover counts above 128, up to a max of about 1.8x[1].

4.2 Prover

Next, the prover side of the protocol will be evaluated for both latency and availability. This evaluation is performed on an Arduino Uno R3 which utilizes the ATmega328p for a processor, a representative choice for low power IEDs. Results will be discussed and analyzed in Section 5.

4.2.1 System Setup and Experiments

For the prover evaluation, SuV was installed on the arduino, with all prover Functions (2, 4, 5, 6, 7, 8) stored in the the secure[2] area of memory. In the untrusted memory, an application was developed which constructed realistic packets, such as *att_req* and SystemStatusUpdate messages. Once the message was constructed, the timer was started and the message was fed to the RA application. The timer was stopped once the received message was fully processed, including packet construction for functions which require it. These timings were performed for each and every function necessary to perform the DPU-LARK scheme. The final performed experiment examines the effect of extra memory on attestation time. This result was collected by calling Function 2, with the size of *instructionmemory* being varied.

[1]For the DPU to maintain this speedup, it assumes that an average of at least 8 provers send their attestation responses every 0.5 ms. This is the line rate necessary to overburden the CPU's SHA accelerator.

[2]Figure 2.2 may help visualize this.

4.2.2 Results

Table 4.2 shows the results of the prover experiments. The first column (Prover Count) indicates the number of potential message sources for the IED in consideration. For columns 2-4, the processing time displayed is necessary when a SystemStatus message is received from the verifer. Overheads in columns 5-6 occur upon receipt of an *att_req* message from the verifier. Column 5 shows the time spent by the prover to craft an attestation response under full memory attestation, where the HMAC of all memory on the device is computed. Column 6 shows the time to craft an attestation response under the improved version of the scheme, where the HMAC of the memory can be known ahead of time. The final two columns in Table 4.2 show the time spent by the prover when receiving a message from a remote source (very low). Considering that these are the only overheads of DPU-LARK during normal runtime, it becomes a very promising construction.

Table 4.2: Time to perform each function related to DPU-LARK on the prover.

Prover Count	Time Spent for DPU-LARK Prover Functions (ms)						
	SystemStatusUpdate	SystemStatusValid	SystemStatusFinal	Att_Resp	Att_Resp_Imp	Accept	Reject
32	3.703	0.143	3.713	3342.103	31.955	0.024	0.033
64	24.723	0.167	24.93	3342.127	31.979	0.024	0.032
96	69.956	0.188	70.493	3342.15	32	0.049	0.057
128	181.453	0.215	182.625	3342.178	32.028	0.024	0.033
160	239.796	0.239	241.736	3342.202	32.051	0.048	0.056

Since some fields in Table 4.2 scale with prover count, a few graphs are presented to give more insight. Figure 4.7 shows the different processing times necessary for the prover upon receipt of status updates from the verifier. Figure 4.7 shows that, for larger numbers of potential message sources, such status updates can incur a relatively large overhead, up to almost 240 ms. The valid status update, on the other hand, is much more efficient and costs about a tenth of the overhead compared to other status update messages. Figure 4.7 results correlate to Table 4.2, columns 2-4.

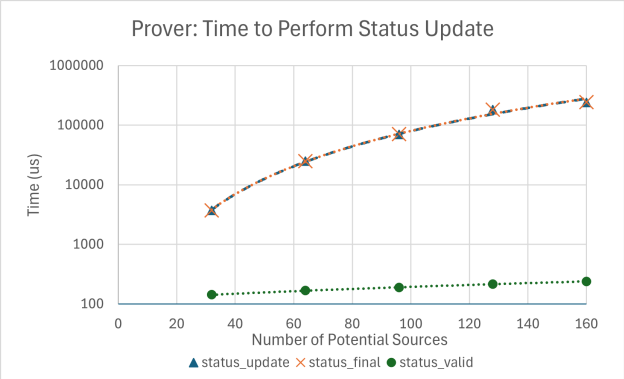


Figure 4.7: A graph showing the status update overhead for prover devices.

Figure 4.8 shows the results of the final prover experiment. The time required to perform attestation scales linearly with the amount of memory being attested, and adds 3.342s of computational overhead when attesting the full memory available on this system setup. With larger memory sizes, this could be substantial if attestation requests are sent too often. While this overhead is significant, it is only required under the conditions of full memory attestation, and must only be performed once per attestation period, which is on the order of minutes. If the improved version of the scheme is used, this overhead is reduced by about 100x as shown in Table 4.2.

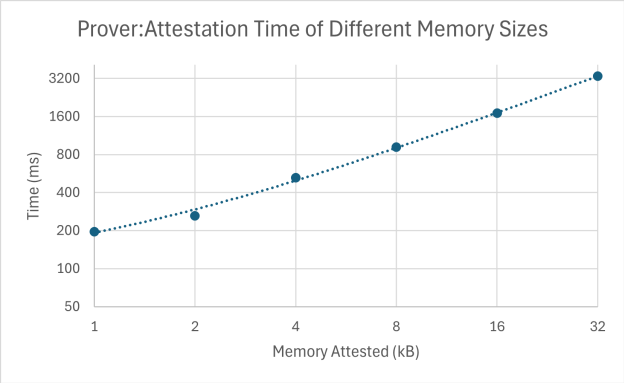


Figure 4.8: Total time to attest different memory sizes.

Chapter 5

Discussion

In this chapter, a discussion of the results demonstrated in Section 4 will be presented. This will include reasons for the results, as well as what they entail for the scheme and its suitability to attest OT networks.

The main considerations for this discussion will be upon the following:

1. Latency - This is a critical consideration for OT networks, as some devices are required to meet very stringent and important latency demands. For critical messages using the GOOSE protocol, for example, devices must have a final delivery latency of no longer than 2 ms [6].

2. Availability - This is another crucial component for devices in OT networks. While it is important that devices remain secure, and can have knowledge of the security of other devices on the network, it is also important that devices are available for high percentage of time so they can perform their usual duties. Given that the SuV requires the device to become dysfunctional during attestation, this becomes an important metric.

3. Traffic Overhead - It is important for application messages to get through to their final destinations. If the traffic overhead of the scheme becomes too high, it will flood the network leading to increased packet drop which is undesired.

4. Memory - While memory is very cheap today, it is still possible to outgrow the memory available on IEDs. If the memory cost of the remote attestation code stored in the secure memory of prover devices becomes too large, it could lead to the inability to deploy the secure code alongside the application code.

5.1 Verifier Results

Here, the results presented in Section 4 will be examined. As a preliminary to this section, experiments 1-3 were performed using varying numbers of cores. For the more powerful CPU, this core count varied from 2-64. For the less powerful CPU, this count varied from 2-16. Finally for the DPU, the core count varied from 2-8. In all cases, one core (the main core) was needed to offload packets to a thread-safe ring, which worker cores consumed packets from. For the results shown in Figure 4.3, each prover count result was taken with total core counts of $\{2^1, 2^2, ..., c\_max\}$, where c_max is the total number of cores available on the tested platform. For both DPU and accelerator-enabled CPU, varying the core count did not have any meaningful impact on any results collected. These results make sense as, for both DPU and CPU SHA hardware accelerated cases, a single worker core was able to keep the queue for the accelerator full. For the CPU with no SHA accelerator, increasing the worker core count saw a base-2 logarithmic improvement with the number of cores.

As for the results of experiment 1, these would initially indicate that the CPU with hardware acceleration outperforms the DPU by around 2x for single message latency. Experiment 3, displayed in Figure 4.5, shows that this discrepancy is almost entirely due to the SHA accelerator performance and experiment 4 (Figure 4.6) helps to show the reason for this result. Comparing the outcome of experiment 4 with the size of an attestation response (108 bytes), it is clear that the CPU accelerator will have a much better performance when only considering the latency of this single message. Eventually, however, the DPU hardware acceleration outpaces the CPU as the input hash size increases. To add to this, the DPU verifier is operating much nearer to the network, and has quicker responses when it comes to receiving and freeing packets from the queue. Therefore, when considering the total amount of data input to these accelerators along with proximity to the network, the higher throughput of the DPU accelerator allows the DPU to match the CPU at any prover count above 128. As a result, the DPU can outpace the high-end CPU for higher prover counts when considering the total time to validate all provers, a more important metric than single prover validation[1]. As a side note, this improvement necessitates that, during the attestation, an average of at least 8 devices per 0.5 ms send their attestation responses after the attestation request. This number was determined through design space exploration, where the line-rate was increased until one of the verifiers (either DPU or SHA accleerated CPU) began to drop packets due to an overfull queue. This should be considered reasonable since, for the large majority of the time, devices will not need to actually perform attestation of their memory making the response time much closer to deterministic. When considering the purpose of DPU's, to

[1]The reason for this will be discussed in Section 5.4

offload tasks and free CPU resources for other computation, this indeed aligns well with the overall goal.

As far as the results of the experiments when comparing to a CPU without SHA acceleration, the DPU far outpaces this CPU in all experiments: a promising result when looking at the scheme as a whole.

5.2 Prover Results

As a preliminary, the Prover Count column of Table 4.2 does not mean the same thing as it does for the verifier results. Here, prover count references the number of sources from which each IED can receive messages rather than provers on the entire network.

The results for the DPU-LARK run on low power IEDs are promising. Most importantly, recall from Section 3 that the protocol has an attestation phase and an active phase. Here, it should be mentioned that the active phase will occupy the large majority of time of the devices. In this phase, devices can validate or reject received messages in around .05 ms, shown in Table 4.2 columns Accept and Reject (which relate to Function 7). This low latency is enabled by the Security MicroVisor, as it can securely store information regarding the state of other devices. Thus, an application only needs to make a very simple function call to its secure memory to inquire the validity of other devices.

All other prover results shown in Table 4.2 are processing overheads that will be acquired during the attestation period. Of these, the status update[2] overheads are the only three to grow in cost with the number of potential sources, as the prover must iterate through each of them when receiving a status update. The overheads for SystemStatusValid messages are much lower than those for SystemStatusUpdate and SystemStatusFinal because of the extra comparisons and data movement required for Functions 5 and 6.

The fundamental purpose of RA is to receive a statement that the software running on the remote device is what it is meant to be. The results in columns Att_Resp and Att_Resp_Imp, which correspond to Functions 2 and 8, of Table 4.2 show the overhead of creating the attestation response when requested by the verifier. As a sidenote, all results in Att_Resp attested the all memory available on the MCU platform. Under full memory attestation, this process takes a consistent 3.35s (portrayed in Table 4.2 column Att_Resp). In systems where critical messages must be delivered in 2ms, such an overhead may be too costly to consider. Fortunately, the improved version eliminates the need to perform memory attestation as long as the instruction memory was not changed since the previous attestation request. Under

[2]Columns labeled SystemStatusUpdate, SystemStatusValid, and SystemStatusFinal relating to Functions 5, 4, and 6 respectively.

these circumstances, the overhead of creating a valid attestation response is reduced by about 100x to 32ms (shown in Table 4.2 column Att_Resp_Imp), and consists mainly of authenticating said response. Considering this is a one-time cost per attestation period, the much smaller overhead becomes viable. Implications of these results on availability and latency are discussed further in Sections 5.4 and 5.5 respectively.

5.3 Traffic Overhead

Traffic overhead for this scheme should be relatively low, and a quick analysis will be performed to determine traffic overhead of the scheme in the worst-case scenario, as well as a more realistic one. Figure 5.1 will aid in the visualization of the analysis. In this figure, the messages encompassed in blue represent one time costs. Each prover needs to provide an attestation response when requested, thus a traffic overhead of n, where n is the number of provers, is immediately accrued. These messages only occur once per attestation period. After these messages, there may be a varying number of **update_req** and **update** messages, encompassed in red in Figure 5.1. Recall from Section 3.2 that after the **final/valid** message, of which there is only one, no messages related to the RA scheme will be sent for the remainder of the attestation period. Thus, the traffic overhead consists of n **att_resp** messages plus some number of **update/update_req** messages, determined next.

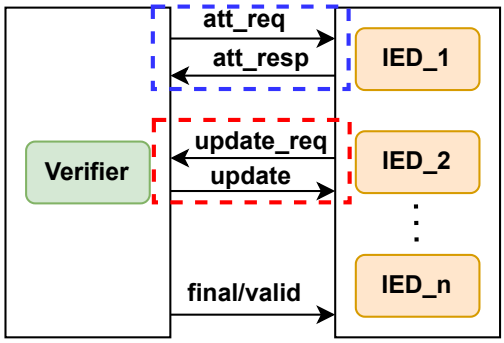


Figure 5.1: A depiction of the traffic overhead in DPU-LARK

In the worst case scenario, each prover relies on application messages from each other prover. On top of this, each prover will send an application message that every remote prover is concerned with. Finally, all of these application messages must be sent before the verifier sends even one update. Recall again from Section 3.2 that each prover will only send one update request per remote prover of interest. In this case, each prover would send an **update_req** message for each other prover, contributing n^2 messages to the traffic. Totaled with the baseline n overhead of the attestation messages, this contributes $\frac{n^2+n}{t\_att}$, where t_att is

the attestation period. Considering t_att is on the order of minutes, this overhead should be trivial on average. When considered as a burst, however, there is a possibility of flooding the network. Given a message size of 64 bytes (as an **update_req** is well below the minimum L2 message size), if we take a line rate of 1 Gb/s[3], the function to determine if the traffic will cause packet dropping is given by Equation 5.3

$$\frac{msg\_count \times bits/msg}{s} \geq line\_rate \tag{5.1}$$

$$\rightarrow \frac{n^2 \times 512}{t} Gb/s \geq 1Gb/s \tag{5.2}$$

$$\rightarrow n \geq \sqrt{\frac{1e9 \times t}{512}} \tag{5.3}$$

where n is the number of provers and t is the burst period in seconds. For context, Equation 5.3 shows that, in order to flood the network to the point of dropping packets, 140 provers would need to send messages within a period of 1 ms.

In a more realistic case, provers will not be concerned with every other prover on the network. Assuming the prover is a critical device such as a large circuit breaker, it will likely be able to receive commands to switch from a few PLCs along with a handful of other IEDs. Conservatively, we can say that critical devices listen to 20 other devices on the network. Also, the verifier will send much more frequent updates, allaying the traffic caused by provers which need to send **update_req** messages. Finally, even in a state of distress, a critical device will only send approximately one message per ms [7] whereas other less critical devices or devices in a typical state send messages about once per second. Taking into account these more realistic assumptions, the traffic overhead becomes extremely small. Yet another conservative assumption is that one in three devices are critical, and are sending messages as if they were experiencing an emergency. Notwithstanding the provers' interest in each others' messages, the function to determine packet drop is now given by Equation 5.6.

$$(\frac{1}{3} \times n \times 1\, msg/ms \times 512\, bits/msg) + (2/3 \times n \times 1\, msg/s \times 512\, bits/msg) \geq 1\, Gb/s \tag{5.4}$$

$$\rightarrow 171,008 \times n\, bits/s \geq 1e9\, bits/s \tag{5.5}$$

$$\rightarrow n \geq 5848 \tag{5.6}$$

With extremely conservative realistic assumptions, Equation 5.6 shows that possibility for flooding due to messages from the attestation scheme requires that the number of prover devices be greater than or equal to 5848. Overall, the network traffic overhead analysis finds

[3]The minimum deployed in most modern networks

the excess traffic introduced by DPU-LARK to be a potential issue at worst, and otherwise a non-concern.

5.4 Latency Overhead

The latency overhead of this scheme is complex, but for typical operation of the network (active phase) latency overhead of a message is negligible as it costs around .05ms on the tested platform.

Another consideration is the worst-case scenario, which occurs during attestation phase. For the analysis, two devices dev_{src} and dev_{dest} will be considered. In the worst case, dev_{src} sends an application message at the same time dev_{dst} receives an attestation request (due to proximity to the verifier). If the prover performs full attestation according to Function 2, then the final receive time is almost entirely governed by this protocol (3.3s). Including a SystemStatusFinal message to update the validity list of dev_{dst} (0.26s), the message from dev_{src} will not be received for about 4s.

Finally, the overhead of DPU-LARK on latency in a more general scenario is considered. Here, the more performant Function 8 is used. Also, devices have not changed their memory and all devices are in a valid state. The case where dev_{src} sends a message to dev_{dst} an instant before dev_{dst} receives an attestation request will still be considered. In such a circumstance, the limiting factor becomes a combination of the provers to respond and the verifier to validate all provers. If the network has 512 total provers, assuming all devices receive the attestation request within 2ms and take similar amounts of time to provide the attestation response, then the latency will be Att_Resp_Imp (32ms) combined with the time to validate all provers (13.5ms from Table 4.1 column DPU). With these reasonable assumptions, dev_{dst} will not receive the message for about 50ms. While still significant, when considering this overhead only occurs once per attestation period, it becomes a much more reasonable sacrifice to make for improved security in the network. As a note, when inspecting the prover time spent upon receipt of status update messages sent by the verifier (shown in Table 4.2, Column SystemStatusUpdate), it appears that status updates do not provide much value. This is due to the time taken for the prover to parse the valid list being longer than expected, and is discussed further in Section 5.7.

5.5 Availability

Availability is crucial for IEDs serving important functions, such as circuit breakers. Due to the design of SuV, the untrusted application cannot run any code and all interrupts are

disabled while performing trusted functions. Thus, in DPU-LARK, device availability is entirely dependent upon which functions IEDs are required to run within an attestation period, and is quite a simple calculation. If performing full memory attestation and all devices on the network are not valid, then the IED will need to perform Function 2 along with Function 6. According to Table 4.2, this will require the device to be unavailable for about 3.5s during each attestation period. To achieve 99% availability, attestation could only be performed every approximately every 6 min (358s to be precise). If the memory has not changed, Function 8 is used, and all devices are valid, then the IED will be unavailable for about 33ms per attestation period. This should be the case for a large majority of the running time, and attestation could be performed every 3.3s to maintain 99% availability. While performing attestation this often would be overkill, the availability in this scenario is a very good result for such a low-level network.

5.6 Memory Overhead

Lastly, a quick discussion of the memory overhead will be performed. All prover code for the DPU-LARK scheme, including required cryptographic libraries and SuV code but excluding the maps for prover ids and remote prover metadata, consumed about 3.2kB of memory on the tested platform. The only portions of memory subject to grow according to potential message sources are the arrays of potential message source id's (each entry consumes 2 bytes) and the metadata array (1 byte per entry). While at first thought it would seem necessary to store the MAC addresses (6 bytes each) of these potential sources as well, this can be avoided through a perfect hash function. Given the relatively low number of mappings, (at most on the order of a few hundred), a perfect hash function can be easily crafted to perfectly map MAC src's to their ID's. This was used in testing to improve performance and save memory. Thus, the total memory consumed by the protocol on a given IED *dev* is $3200 + 3n$, where n is the number of devices that *dev* can receive messages from. Considering the low cost of memory, this is more than reasonable and on tested hardware, consumed about 11% of available memory.

5.7 Future Work

There are a number of improvements to be made for scheme, as well as more experimental results which would be helpful to perform. First among these would be to actually attach the prover device implementation to the network with the DPU. Currently, our latency results are only theoretical, and, while most likely accurate, they do not actually capture traffic

between a verifier and a true MCU functioning as a prover (IED). Instead, the verifier was evaluated independently with a simulation of provers. Similarly, an MCU functioning as an IED was evaluated independently by feeding it messages that would typically be sent by the verifier or other IEDs. Such an experiment would give real results for latencies, and potentially reveal unknown issues with the scheme.

Another useful experiment would be to run a true network simulation, such as OT-sim [30] with accurate IEC 61850 traffic. This would aid in ascertaining the scalability of the scheme, and provide a deeper insight into the functionality as a whole.

The main piece of the scheme which could be improved is the robustness against physical intrusions. Considering that all prover devices use the same key for signing, and this key is not updated, it would mean that a single compromised prover affects the integrity of every other device. Perhaps using a hash-chaining method, like that used in [31], could help to alleviate this concern.

Another useful experiment would be to have a OT IED applications installed in the untrusted memory of multiple MCUs and run DPU-LARK on top of this. Such a setup would provide the most accurate results in terms of latency overhead and the effect of the attestation scheme on the untrusted application.

In order to maintain security claims, the code lying in trusted memory using the Security MicroVisor must be memory-safe and crash-free [32]. To make the security claims of the scheme stronger, validation of these claims should be made by verifying the code using a tool like VeriFast [33].

An interesting finding of this analysis is that it can cause a much greater overhead for the verifier to send status updates in the manner designed, than to simply finish validating all messages and update the network at the end. If a prover needs to verify a message before the verifier has finished network attestation, it would likely be more suitable for the verifier to use one of its extra cores to respond directly to the prover. Since these cores do not seem to improve network attestation time, the DPU can make use of the extra computation to perform the specific validity lookup for the prover using a hash. This would avoid the prover having to iterate through the entire device list, which may be much larger than the number of devices a given prover may actually have interest in.

Chapter 6

Related Works

6.1 Authentication Improvement

The most explored method for securing messages between devices in OT network lies in the improvement of message authentication, based on network and protocol specific properties. Interesting research in this avenue is discussed in the following sections.

6.1.1 F-Pro

The F-Pro scheme emerges as a pivotal advancement in the authentication of data within smart grids. By embedding data provenance (the origin of data) into the scheme, F-Pro ensures that every message can be tracked across network channels, enhancing anomaly detection and thwarting malicious activities. This provenance-aware mechanism not only fortifies the data's integrity, but also ensures its authenticity from origin to endpoint.

F-Pro's flexible architecture adapts to the security requirements and operational conditions distinctive to OT network environments. Through hash-chaining and offloading computation to bump in the wire devices, F-Pro achieves rapid authentication responses, essential for the real-time operational demands critical for maintaining both stability and efficiency. It also introduces the concept of message provenance to OT networks, providing a greater level of trust than authentication alone. The adaptability and expedited processing capabilities of F-Pro attempt to redefine existing security benchmarks within operational technology infrastructures, marking a substantial progression in developing robust, resilient energy systems [8].

While this scheme shows that authentication within the environment of OT networks is possible, it requires a peripheral device to be installed at each device on the network. This can be a limiting factor for many existing networks due primarily to cost concerns.

6.1.2 LOMOS

This research introduces the LoMoS scheme, which seeks to make public key cryptography authentication viable for OT networks by shifting the computationally intensive tasks of digital signing to an offline phase, thereby minimizing the processing time required during the online phase when systems are actively exchanging messages.

LoMoS acts as an improvement upon similar schemes in terms of flexibility and online signing/authenticating latency. To accomplish this, a tree similar to those used in Merkle Signature Schemes is constructed offline by each source. This allows the sources to provide provide proofs they sent messages by providing different nodes of the tree to the destination. Then, destinations can authenticate the messages while only possessing the root of the tree. Eventually, the trees used for online proofs of authentication do expire, and new trees must be constructed by each source offline. By handling most of the cryptographic computations offline, LoMoS significantly reduces the latency typically associated with digital signatures during live communication scenarios. This feature is particularly critical in environments where every millisecond of processing time can impact system performance and safety.

Additionally, the LoMoS explores the security implications of the scheme, ensuring that despite the minimized online computation, the integrity and non-repudiation properties of the digital signatures are upheld. This approach not only accelerates the authentication process, but also maintains a robust security posture, making it an attractive solution for industries that operate under stringent time constraints [7].

LoMoS, though it remains a separate implementation than F-Pro, remains a scheme which seeks to improve the message authentication latency for low power devices located on OT networks. Consequently, while maintaining many unique properties from F-Pro, it holds the same philosophy.

6.1.3 Cache-Based Authentication

Multicast messaging is essential for efficient communication in large-scale control systems, but introduces significant security challenges in authentication. As stated above, traditional methods can introduce delays that are unacceptable in time-critical operations. This paper proposes a caching-based solution to address the problem by reducing the overhead associated with the authentication process. Through the leveraging of a hash-caching mechanism, the system can pre-validate parts of messages and frequently used data based on commonly sent/received messages, decreasing the time required to authenticate messages with no compromise to security [6].

This study's significance lies in its realization that a majority of messages sent by devices

are repeated and predictable, providing potential to enhance the responsiveness of secure industrial communication networks and ensuring that security measures do not impede the operational efficiency of critical infrastructure. This approach could be particularly valuable in settings where safety and uptime are paramount, such as layers of the network which implement IEC 61850 GOOSE, offering a balance between security and performance.

6.2 Remote Attestation

Contrary to the improvement of message authentication discussed previously, another strategy proposed to improve the cyber security of OT networks lies in remote attestation. Due to convergence of OT networks with IT and IoT networks, such an approach is becoming more relevant by the day. Contrarily, RA constructions are not typically focused on prover-prover validation or low-latency, as is required in the lower levels of OT networks. These schemes provide a different type of security to networks than the authentication schemes previously discussed. Rather than ensuring a message came from a certain source, remote attestation schemes seek to ensure that each device on the network is in a valid software state (running the correct code). Thus, a few related methodologies will be examined in the following sections.

6.2.1 SlimIOT

The slimIoT protocol addresses the challenge of securing OT networks by introducing a lightweight, scalable solution for ensuring the integrity and authenticity of IEDs. Its main contribution lies in the fact that it is robust to physical attacks by design. Essentially, during attestation, a virtual spanning tree is formed in the network via prover devices re-broadcasting the attestation request message. Then, provers mark their presence in the scheme by setting the bit correlated to their id. Assuming the physical attack takes the device offline for a time longer than the period between two attestation requests, then a compromised prover device cannot mark its presence in the protocol. This method does require loose time-synchronization, however, due to the method in which messages are encrypted and authenticated which necessitates a real-time clock for all prover devices[31]. As for the case of OT networks, slimIoT does not make considerations for latency-critical prover-prover validation.

6.2.2 Simple+

SIMPLE is designed to be lightweight and efficient, minimizing the resource usage on the IED while still providing a robust verification mechanism. By focusing on the unique challenges posed by resource-constrained environments, SIMPLE offers a practical solution that can be widely deployed in various IoT applications, from smart homes to industrial IoT systems. Crucially, it is the first RA scheme which utilizes the Security MicroVisor, allowing it to make assumptions similar to hardware-based schemes while remaining a software-based RA scheme. A swarm attestation scheme, SIMPLE+, is also introduced. This scheme uses similar methodology to other swarm schemes by forming a virtual spanning tree to allow prover devices to aggregate results of other devices.[32]. While able to make robust security claims for a software-based scheme, through the use of SuV, this scheme also does not consider stringent latency requirements present at lower levels of OT networks.

6.2.3 SARA

SARA's core innovation lies in its asynchronous nature, which allows IoT devices to perform attestation checks without the need for a continuous connection to a central verifier. Contrary to other swarm schemes, SARA does not require each prover to stop execution of its usual tasks until the entire network is validated. This is particularly useful in scenarios where IoT devices operate in remote or network-constrained environments. The protocol ensures that these devices can still maintain high levels of security, verifying their software integrity independently of their network state. SARA also introduces a unique consideration, which is the attestation of services (consisting of multiple devices and their communications) as a whole. This means that, if one device in a service does not pass the attestation, it will consider other devices affected by the malicious device and consider that entire service as affected.

Furthermore, SARA is designed to be lightweight, minimizing the computational and power demands on the IoT devices, which often have limited resources. This makes it an ideal solution for sprawling IoT networks that need to manage thousands of devices efficiently. While improving availability due to the asynchronous nature, SARA does not consider low-latency prover-prover validation.

Contrary to both slimIoT and Simple+ as well as other RA schemes, DPU-LARK is the first one to focus on low-level portions of OT networks. On top of this, DPU-LARK distinguishes itself by utilizing the Security MicroVisor not only to make security claims similar to hardware based schemes, but to concretely improve the performance of the scheme.

Chapter 7

Conclusion

With the convergence of OT and IT networks, cyber attacks on critical infrastructure become increasingly threatening. While edge security of these networks is a popular area of research, considerations must be made in case an attack breaches these barriers and spreads to lower levels. One helpful step to take in this direction is to improve the message authentication procedure enough to implement authentication on low-latency communication protocols. Another step is to ensure that each device on the network is running the appropriate software and not compromised by an adversary. To this end, this research presents a new RA scheme, DPU-LARK, which seeks to attest lower levels of OT networks with strict message latency demands. Furthermore, the feasibility of offloading the significant task of acting as a verifier to a DPU is evaluated.

The findings are promising. Using the DPU as a verifier is able to match or improve upon the performance of a powerful, much more expensive CPU with relevant hardware acceleration, as long as there are 128 devices on the network. When compared to a less powerful CPU lacking the acceleration, the DPU performs better by a range of 11-22x. On top of this, for the vast majority of cases, availability of IED's on the network is very high. During normal operation, DPU-LARK does not introduce meaningful loss in availability and, even during attestation, DPU-LARK can attain 99.9% availability while performing attestation every 33s. Furthermore, the additional time to validate a message from remote device is negligible for a similar ratio.

References

[1] D. Kushner, *The real story of stuxnet*, Jan. 2024. [Online]. Available: https://spectrum.ieee.org/the-real-story-of-stuxnet.

[2] U.S.D.O.E., *August 2003 blackout*. [Online]. Available: https://www.energy.gov/oe/august-2003-blackout.

[3] S. R. Das, *Northern india recovering from huge blackout*, Jun. 2021. [Online]. Available: https://spectrum.ieee.org/northern-india-recovering-from-huge-blackout.

[4] K. A. Stouffer, M. Pease, C. Tang, *et al.*, *Guide to operational technology (ot) security*, en, Sep. 2023. DOI: https://doi.org/10.6028/NIST.SP.800-82r3. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=956505.

[5] M. S. Team, *New research shows iot and ot innovation is critical to business but comes with significant risks*, May 2023. [Online]. Available: https://www.microsoft.com/en-us/security/blog/2021/12/08/new-research-shows-iot-and-ot-innovation-is-critical-to-business-but-comes-with-significant-risks/.

[6] U. Tefek, E. Esiner, D. Mashima, B. Chen, and Y.-C. Hu, "Caching-based multicast message authentication in time-critical industrial control systems," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 1039–1048. DOI: 10.1109/INFOCOM48880.2022.9796767.

[7] E. Esiner, U. Tefek, H. Erol, *et al.*, "Lomos: Less-online/more-offline signatures for extremely time-critical systems," English (US), *IEEE Transactions on Smart Grid*, vol. 13, no. 4, pp. 3214–3226, Jul. 2022, Publisher Copyright: © 2022 IEEE., ISSN: 1949-3053. DOI: 10.1109/TSG.2022.3156897.

[8] E. Esiner, D. Mashima, B. Chen, Z. Kalbarczyk, and D. Nicol, "F-pro: A fast and flexible provenance-aware message authentication scheme for smart grid," in *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2019, pp. 1–7. DOI: 10.1109/SmartGridComm.2019.8909712.

[9] G. Coker, J. Guttman, P. Loscocco, *et al.*, "Principles of remote attestation," *International Journal of Information Security*, vol. 10, pp. 63–81, 2011.

[10] Y. Wang, J. Li, X. Chen, H. Lin, F. Yu, and J. Luo, "Remote attestation for intelligent electronic devices in smart grid based on trusted level measurement," *Chinese Journal of Electronics*, vol. 29, no. 3, pp. 437–446, 2020. DOI: https://doi.org/10.1049/cje.2020.02.019. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/cje.2020.02.019. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cje.2020.02.019.

[11] X. Yang, X. He, W. Yu, *et al.*, "Towards a low-cost remote memory attestation for the smart grid," *Sensors*, vol. 15, no. 8, pp. 20 799–20 824, 2015, ISSN: 1424-8220. DOI: 10.3390/s150820799. [Online]. Available: https://www.mdpi.com/1424-8220/15/8/20799.

[12] G. Karopoulos, C. Xenakis, S. Tennina, and S. Evangelopoulos, "Towards trusted metering in the smart grid," in *2017 IEEE 22nd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2017, pp. 1–5. DOI: 10.1109/CAMAD.2017.8031643.

[13] B. Fraune, T. Woltjen, B. Siemers, and R. Sethmann, "Integration of remote attestation into iec 61850," in *2023 IEEE Belgrade PowerTech*, IEEE, 2023, pp. 1–7.

[14] S. Kumar, A. Abu-Siada, N. Das, and S. Islam, "Review of the legacy and future of iec 61850 protocols encompassing substation automation system," *Electronics*, vol. 12, no. 15, 2023, ISSN: 2079-9292. DOI: 10.3390/electronics12153345. [Online]. Available: https://www.mdpi.com/2079-9292/12/15/3345.

[15] [Online]. Available: https://projectbinder.eu/the-importance-of-monitoring-your-ot-network/.

[16] B. Kuang, A. Fu, W. Susilo, S. Yu, and Y. Gao, "A survey of remote attestation in internet of things: Attacks, countermeasures, and prospects," *Computers & Security*, vol. 112, p. 102 498, 2022, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2021.102498. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821003229.

[17] W. Feng, Y. Qin, S. Zhao, and D. Feng, "Aaot: Lightweight attestation and authentication of low-resource things in iot and cps," *Computer Networks*, vol. 134, pp. 167–182, 2018, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2018.01.039. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128618300471.

[18] M. Kucab, P. Boryło, and P. Chołda, "Remote attestation and integrity measurements with intel sgx for virtual machines," *Computers & Security*, vol. 106, p. 102 300, 2021, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2021.102300. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821001243.

[19] R. Vieira Steiner and E. Lupu, "Towards more practical software-based attestation," *Computer Networks*, vol. 149, pp. 43–55, 2019, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2018.11.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128618307631.

[20] Y.-G. Choi, J. Kang, and D. Nyang, "Proactive code verification protocol in wireless sensor network," in *Computational Science and Its Applications – ICCSA 2007*, O. Gervasi and M. L. Gavrilova, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 1085–1096, ISBN: 978-3-540-74477-1.

[21] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, "VRASED: A verified Hardware/Software Co-Design for remote attestation," in *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1429–1446, ISBN: 978-1-939133-06-9. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/de-oliveira-nunes.

[22] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, and G. Tsudik, "APEX: A verified architecture for proofs of execution on remote devices under full software compromise," in *29th USENIX Security Symposium (USENIX Security 20)*, USENIX Association, Aug. 2020, pp. 771–788, ISBN: 978-1-939133-17-5. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/nunes.

[23] S. Kumar, P. Eugster, and S. Santini, "Software-based remote network attestation," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 2920–2933, 2022. DOI: 10.1109/TDSC.2021.3077993.

[24] F. Kohnhäuser, N. Büscher, and S. Katzenbeisser, "Salad: Secure and lightweight attestation of highly dynamic and disruptive networks," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ser. ASIACCS '18, Incheon, Republic of Korea: Association for Computing Machinery, 2018, pp. 329–342, ISBN: 9781450355766. DOI: 10.1145/3196494.3196544. [Online]. Available: https://doi.org/10.1145/3196494.3196544.

[25] M. Ambrosin, M. Conti, R. Lazzeretti, M. M. Rabbani, and S. Ranise, "PADS: practical attestation for highly dynamic swarm topologies," *CoRR*, vol. abs/1806.05766, 2018. arXiv: 1806.05766. [Online]. Available: http://arxiv.org/abs/1806.05766.

[26] M. Ammar, B. Crispo, B. Jacobs, D. Hughes, and W. Daniels, "Sμv—the security microvisor: A formally-verified software-based security architecture for the internet of things," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 5, pp. 885–901, 2019. DOI: 10.1109/TDSC.2019.2928541.

[27] Aug. 2022. [Online]. Available: https://nextgeninfra.io/smartnics-infrastructure-acceleration/.

[28] Apr. 2024. [Online]. Available: https://docs.nvidia.com/doca/sdk/index.html.

[29] Cybersecurity and Infrastructure Security Agency, *Ir alert h-16-056-01*, Accessed: 2024-04-17, 2023. [Online]. Available: https://www.cisa.gov/news-events/ics-alerts/ir-alert-h-16-056-01.

[30] P. S. LLC, *Ot-sim: A trace-driven co-simulation framework for heterogeneous system-on-chip*, Online; accessed 29-April-2024, 2022. [Online]. Available: https://github.com/patsec/ot-sim.

[31] M. Ammar, M. Washha, G. S. Ramabhadran, and B. Crispo, "Slimiot: Scalable lightweight attestation protocol for the internet of things," Cited by: 12; All Open Access, Green Open Access, 2019. DOI: 10.1109/DESEC.2018.8625142. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85060786859&doi=10.1109%2fDESEC.2018.8625142&partnerID=40&md5=7b8e692fb1af60ba8adf298910483c2c.

[32] M. Ammar, B. Crispo, and G. Tsudik, "Simple: A remote attestation approach for resource-constrained iot devices," in *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, 2020, pp. 247–258. DOI: 10.1109/ICCPS48487.2020.00036.

[33] B. Jacobs, J. Smans, P. Philippaerts, F. Vogels, W. Penninckx, and F. Piessens, "Verifast: A powerful, sound, predictable, fast verifier for c and java," in *NASA formal methods symposium*, Springer, 2011, pp. 41–55.