ERAMO: Effective Remote Attestation through Memory Offloading

Jeppe Hagelskjær Østergaard
DTU Compute
Technical University of Denmark
2800 Kgs. Lyngby, Denmark
Email: s190312@student.dtu.dk

Edlira Dushku
DTU Compute
Technical University of Denmark
2800 Kgs. Lyngby, Denmark
Email: edldu@dtu.dk

Nicola Dragoni
DTU Compute
Technical University of Denmark
2800 Kgs. Lyngby, Denmark
Email: ndra@dtu.dk

2021 IEEE International Conference on Cyber Security and Resilience (CSR) | 978-1-6654-0285-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/CSR51186.2021.9527978

***Abstract*—Remote Attestation (RA) has gained a broad attention over recent years as an essential security mechanism that enables integrity verification of remote IoT devices. Typically, existing RA protocols aim at detecting malware presence in *program memory*. Recent RA schemes work towards attesting also *data memory* and focus mainly on detecting runtime attacks that manipulate stack pointers to hijack the execution flow of a running program. Despite different RA approaches, some data memory attacks still remain undetected. This paper proposes ERAMO, a novel RA protocol that investigates memory offloading technique in attesting broad memory regions of IoT devices. Instead of running a complex RA protocol on a resource-constrained IoT device, ERAMO leverages the emerging paradigm of Fog Computing to securely offload memory contents of IoT devices to nearby powerful devices. This approach aims at increasing the effectiveness of RA protocols by attesting larger data memory regions and allowing powerful devices to perform complex analysis of IoT devices' state. We validate and evaluate ERAMO with a hardware proof-of-concept implementation using an ARM Cortex-M33 based microcontroller that provides ARM TrustZone to support secure isolation of the RA procedure. The conducted experiments confirm the feasibility of ERAMO and demonstrate that offloading technique increases the RA effectiveness in attesting dynamic memory regions.**

I. Introduction

Internet of Things (IoT) devices are more and more getting integrated into every aspect of our everyday life. The number of active IoT devices is currently estimated around 11.7 billion and is expected to reach 30 billion by 2025 [1]. Unfortunately, IoT security is often lacking, with devices having weak passwords or vulnerabilities to various attacks. For many organizations, IoT security is not considered until late in the design and prototyping phase, and security updates are not issued after deployment [2].

Aimed at securing IoT devices, *remote attestation* (RA) has been proposed as a security protocol for malware detection on an untrusted platform. A typical RA protocol consists of two parties, a trusted Verifier and an untrusted Prover. The Verifier aims at assuring the integrity of the Prover (i.e., untrusted IoT device). During the attestation, the Prover sends proofs about its current state of the memory (typically a hash of the memory) to the Verifier, whereas the Verifier matches the received evidence with the expected legitimate state (known in advance) of the Prover, and according to that it validates whether the Prover is trustworthy or not.

Based on the memory regions considered for attestation, RA schemes can be classified into either static or dynamic. Static schemes verify the integrity of the Prover's memory, which does not change at runtime (e.g., the program binary). In contrast, dynamic schemes verify the Prover's integrity based on the memory regions that change at runtime, such as the RAM. These schemes monitor the execution flow of running devices and mainly focus on detecting control-flow attacks. However, current dynamic schemes are vulnerable to other runtime attacks that manipulate data pointers through Data-Oriented Programming (DOP) technique [3] without deviating the control-flow execution of the running software. Additionally, many memory regions of IoT devices remain unattested by the existing RA schemes. Consequently, IoT devices might be vulnerable to mobile attacks [4], [5] that can relocate themselves during attestation. Designing lightweight RA protocols that run inside resource-constrained IoT devices to detect data attacks and mobile attacks is not a trivial task.

Contribution of the Paper. This paper proposes a new RA approach that aims at addressing the aforementioned issues. The main idea is simple: instead of performing RA directly on resource-constrained IoT devices, our approach is based *on offloading Prover's memory* to a powerful platform with more resources and computational capabilities, which will then perform the attestation. The approach leverages the opportunities offered by the emerging Fog computing paradigm [6], where a layer of distributed powerful computing entities (i.e., Fog nodes) can enable the deployment of RA services. This allows the Verifier, deployed and running on a Fog node, to perform a much more accurate remote attestation of the IoT devices (Provers) the Verifier is responsible for.

In particular, the paper brings the following main contributions to the research field of remote attestation:

- The paper proposes a novel RA protocol (ERAMO - Effective Remote Attestation through Memory Offloading) that takes into account both static and dynamic memory regions of an IoT device and checks the integrity of all memory-mapped peripheral.
- To the best of our knowledge, this paper is the first work that successfully implements and evaluates secure

memory offloading as a means for enhancing remote attestation. ERAMO has been implemented and evaluated on an ARM Cortex-M33 based microcontroller, leveraging the security features provided by ARM TrustZone. Experiments confirm the feasibility and effectiveness of the protocol in attesting dynamic memory regions.

Outline of the Paper. The remainder of this paper is organized as follows. We explain the problem statement in Section II. Section III presents different RA approaches and compares ERAMO with the existing RA schemes. The paper describes the system model in Section IV and explains the adversary model in Section V. Next, the paper presents the protocol details in Section VI and the performance evaluation in Section VIII. Finally, we present a protocol discussion in Section IX and concluding remarks in Section X.

II. Problem statement

Consider an attacker that discovers and exploits a program vulnerability such as a buffer overflow. By leveraging Return-Oriented Programming (ROP) technique [7], the attacker alters at runtime the execution flow of legitimate code already loaded on the device's memory to produce a malicious operation. Additionally, the attacker can use Data-Oriented Programming (DOP) technique [3] to compromise variables' values and manipulate data pointers. Such attacks are common in IoT [8] as resource-constrained IoT devices are exposed to many well-known vulnerabilities e.g., *format string* and *integer overflow*.

The dynamic RA protocols in the literature (e.g., [9], [10], [11]) which aim to detect control-flow attacks rely on tracing the software execution inside an IoT device and representing each execution flow as a single hash value. Since these approaches detect the control-flow subversion, they do not detect data attacks which do not maliciously deviate from the legitimate control-flow executions. The RA schemes presented in [4], [5] aim to detect mobile adversaries, which during the attestation, relocate to different memory blocks of a memory region (i.e., memory blocks that comprise the program memory). However, the existing RA schemes do not attest all the memory regions of an IoT device. Thus, at the attestation time, a mobile adversary could also move to the unattested memory regions and relocate again on the original memory once the RA procedure has finished.

In the context of the attacks described above, we propose a new protocol that uses memory offloading to shift the attestation from low-end devices to nearby devices with more powerful computational capabilities. This approach is aligned with and leverages the emerging Fog computing paradigm, which extends the Cloud by bringing computational resources next to IoT devices [6].

III. Related Works

This section summarizes the single-device and dynamic state-of-the-art RA protocols in the IoT domain.

RA overview. RA approaches are generally classified into three main categories: software-based, hardware-based and hybrid approaches. Software-based schemes (e.g., SWATT [12], Pioneer [13]) do not make any hardware assumption and purely rely on the strict execution time of the RA protocol. Despite their advantages, software-based RA schemes do not provide strong security guarantees [22], [23]. Hardware-based schemes (e.g., [24], [25]) use a tamper-resistance hardware module as a Trusted Execution Environment (TEE). While hardware-based designs provide strong security guarantees, they are not suitable for low-cost resource-constrained IoT devices. To provide lightweight secure RA protocols, hybrid designs (e.g., SMART [14], TrustLite [15], TyTan [16]) rely on minimal hardware changes to ensure that the RA protocol and associated authentication keys cannot be tampered with. All these schemes perform attestation on a single device. *Collective attestation* schemes (e.g., SEDA [18], SANA [19], SARA [20]) aim to provide scalable RA solutions that attest efficiently large-scale IoT networks.

SMARM [4] aims to detect mobile adversaries that, during attestation, relocate to different memory blocks of the program memory. SMARM uses a probabilistic approach to compute memory measurements in a random order, which cannot be predicted by malware. However, the probabilistic random (shuffled) measurements increase the attestation time. ERASMUS [5] is a non-interactive RA protocol that allows the Prover to self-initiate the attestation procedure at pre-defined times. The attestation results are stored locally, and the Verifier retrieves a set of attestation results. The sequence of the attestation results allows the Verifier to detect mobile adversaries that may leave or get relocated during attestation.

Dynamic RA. While the aforementioned RA schemes perform only static attestation, dynamic RA schemes aim to attest dynamic data memory. C-FLAT [9] is the first dynamic RA protocol for resource-constrained devices, and it focuses on detecting control-flow attacks. C-FLAT relies on software instrumentation to trace the execution of a running software and generates an accumulative single hash value for each execution flow. At the verification phase, the Verifier compares the generated hash value with a set of expected legitimate values to determine whether the device is trustworthy or not. C-FLAT is implemented in a TEE such as TrustZone. However, C-FLAT introduces a high overhead because at runtime each instrumented code instruction is intercepted and redirected to the TrustZone secure world. LO-FAT [10] enhances C-FLAT by replacing software instrumentation with a hardware module, implemented on an external FPGA, which intercepts the executed instructions at runtime. Likewise, ATRIUM [11] extends C-FLAT and LO-FAT by attesting both executed instructions and the control-flow. However, these schemes detect control-flow deviations and do not consider data attacks which leverage DOP technique [26] to corrupt data variables without altering control-flow information. LiteHAX [17] aims to detect both control-flow and data-attacks. However, LiteHAX detects only the memory operations *load* and *store*, thus, it works only on RISC-based architectures.

Memory offloading. Beside RA protocols, some works within the field of offloading are of interest. In particular, CloneCloud [21] allows a resource-constrained mobile device

TABLE I: Related work summary

Scheme	Static memory	RAM	Peripheral	Verification	Type	Attestation
SWATT [12], Pioneer [13]	●	○	○	Program checksum	One-to-one	On-demand
SMART [14], TrustLite [15], TyTan [16]	●	○	○	Program checksum	One-to-one	On-demand
C-FLAT [9], LO-FAT [10]	○	◐	○	Control flow integrity (CFI)	One-to-one	On-demand
ATRIUM [11], LiteHAX [17]	●	◐	○	Program checksum & CFI	One-to-one	On-demand
SMARM [4]	●	○	○	Program checksum & Shuffled Measurements	One-to-one	On-demand
ERASMUS [5]	●	○	○	Program checksum	One-to-one	Self-initiated
SEDA [18], SANA [19], SARA [20]	●	○	○	Program checksum	One-to-many	On-demand
CloneCloud [21]	○	●	○	○	One-to-one	○
ERAMO	●	●	◐	**Memory offloading**	**One-to-one**	**On-demand**

to offload its execution threads to a clone of itself operating in a virtual machine with more computational capabilities.

In the context of remote attestation it will, however, be more useful to gather an accurate clone of the device memory, on which the memory forensics can be performed, rather than replicating the functionality of the device. Additionally, certain security guarantees not considered by offloading techniques must be provided by RA designs, as they are intended to be used on potentially malware-infected platforms. Due to these differences in security requirements and their purpose, the works within the field of offloading are not directly applicable.

The possibility of offloading in the RA context is first mentioned in RAaS [27], where RA is proposed as a cloud service. While this proposal is mostly focused on increasing the efficiency of the protocol and reducing its associated downtime, the proposal has not been implemented and evaluated. Furthermore, we want to investigate whether offloading the memory is feasible and how this technique can be used to improve the effectiveness of remote attestation.

Discussion. Table I summarizes the works discussed so far. In short, the static RA approach does not consider runtime attacks, while recent dynamic approaches are limited to detecting control-flow attacks. Thus, even if the approaches are combined, data attacks will still remain undetectable.

This paper proposes a new protocol (ERAMO) aimed at addressing the limitations of current attestation designs. Instead of relying on the control-flow attestation or checksum/hash comparisons, we propose to transmit the entire memory to the Verifier. This allows the Verifier to employ sophisticated methods of attesting the dynamic memory (e.g., the open-source Volatility memory forensics framework [28]), while expanding the protocol to also cover the Prover's peripherals (e.g. ADC and I2C configurations). ERAMO design is based on ARM TrustZone [29], which is a hardware-enforced isolation method. However, as opposed to other hardware-based methods, TrustZone is built into the CPU, providing a TEE without the need for external specialised hardware.

IV. System model

We consider an IoT system which adopts Fog computing paradigm [6]. In this system, an untrusted resource-constrained IoT device interacts with a nearby powerful device named Fog node. To design a RA protocol in this setting, we consider the presence of the following two entities as shown in Figure 1:

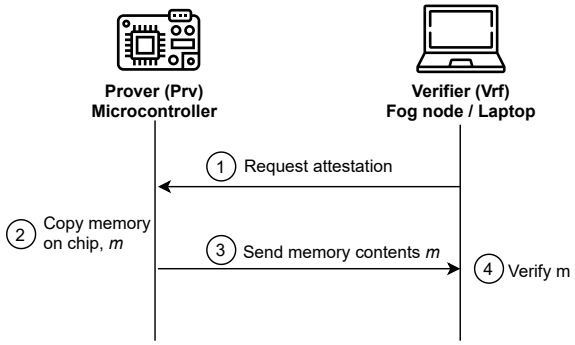

Fig. 1: System model of the memory offloading protocol

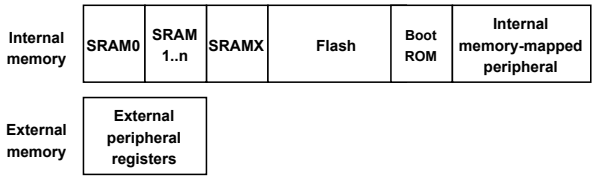

Fig. 2: Memory on chip of a Prover device

- The *Prover* (Prv) is an untrusted IoT device. This device can be infected by malware or can be misconfigured as a result of previous attacks. Prover's memory consists of a set of *memory regions* as shown in Figure 2. Each memory region can be seen as a set of smaller units called *memory blocks*. The Prover might be a non-monolithic system, e.g., a multi-sensor IoT device that provides a set of sensing capabilities using external peripheral.
- The *Verifier* (Vrf) is a powerful device that has resources and computational capabilities to perform complex operations. Besides, the Verifier has the required resources to adopt advanced security and trust techniques (e.g., it is equipped with a TPM [30]); thus, it is assumed to be trusted. Additionally, aligned with other RA schemes in the literature, we assume that the Verifier knows in advance the legitimate program binaries of the Prover. The Verifier randomly initiates the attestation on the Prover, after which it can perform memory forensics

techniques to determine the Prover's integrity.

The Verifier initiates the attestation by sending a request to the Prover (Step (1) in Figure 2). After obtaining the attestation request, the Prover copies the content of its entire internal memory associated with the device application (Step (2)) and offloads it to the Verifier (Step (3)). Upon receiving the Prover's memory, the Verifier will perform the verification (Step (4)) to check the Prover's trustworthiness. The verification process includes two main parts: a comparison of the static memory (e.g., flash memory) with the legitimate program binaries known in advance and a detailed investigation of the transferred data memories (e.g., SRAM).

V. Adversary model and Security requirements

In the following, we define the adversarial capabilities and corresponding security requirements w.r.t. the system model described in Section IV.

A. Adversary model

In line with the adversary model described in [31], [32], and [27], we consider an adversary with the following capabilities.

- **Software attack:** A software adversary compromises the Prover's program memory by injecting and executing malicious code. Additionally, this adversary can exploit a software vulnerability to compromise data memory, for instance, by modifying variable's value, corrupting control-flow pointers, data pointers. This can also be exploited to misconfigure internal or external peripheral to cause unintended device behaviour.
- **Communication attack:** The communication adversary can fully control communications between the Prover and the Verifier by forging, dropping, delaying, eavesdropping the exchanged messages.
- **Mobile attack:** A mobile adversary is a smart adversary that tries to avoid detection by deleting itself during the attestation time or relocating itself to different memory blocks or memory regions which have already been transmitted to the Verifier.
- **Replay attack:** An adversary precomputes a valid attestation response and sends this old legitimate response to hide an ongoing attack.

Assumptions. We assume that a software adversary does not compromise the hardware-protected memory. Following the assumptions of other RA schemes [18], [9], [14], we rule out physical adversaries, Denial of Service (DoS), and Time-Of-Check Time-Of-Use (TOCTOU) attacks. While we do not consider TOCTOU attacks, we limit these attacks by transmitting complete device memory to a powerful Verifier that performs advanced analysis or historical comparison over the memory contents.

Device requirements. In line with common assumptions of the state-of-the-art RA schemes, we assume the presence of two trusted components inside a Prover.

- **Read-Only Memory (ROM).** A ROM memory region contains the code of ERAMO protocol. The protocol code resided in this memory region cannot be tampered with by a software adversary.
- **Secure key storage.** A secure memory region stores the Prover's keys. Only ERAMO protocol has read permissions in this memory region.

B. Security requirements

Based on the adversarial actions described in Section V, in the following we define the required security properties.

- **Integrity.** The protocol should provide reliable evidence guaranteeing that the transmitted memory contents correspond to the Prover's memory at the time of the attestation request.
- **Authenticity.** The protocol should provide verifiable evidence for the origin of the memory contents transmitted.
- **Integrity of communication data.** The protocol should ensure that any memory contents transmitted cannot be altered without it being detectable.
- **Freshness.** The protocol should ensure that any given response to an attestation request can be reliably linked to that request.

VI. ERAMO: Protocol proposal

ERAMO protocol consists of three main phases: (1) Setup phase, (2) Attestation phase, and (3) Verification phase. In the following, we describe each phase in detail.

Setup phase. A network operator guarantees the secure bootstrap of the software deployed on each Prover. Considering the limited capabilities of Provers, the Verifier and the Prover establish a shared symmetric attestation Message Authentication Code (MAC) key k. To prevent untrusted parties from using Prover's key, the shared attestation key k is stored in a hardware-protected memory. Alternatively, a Prover can establish a secure communication channel with the Verifier by possessing an asymmetric key-pair (pk, sk) and knowing the Verifier's public key. Note that the key management details are out of scope of this paper. The protocol description is independent of the key management, thus, the symmetric key usage can be easily replaced by an asymmetric key-pair. For simplicity, preserving our work's generality, we assume that the Prover and the Verifier share a symmetric key k.

Attestation phase. Figure 3 illustrates the protocol. To initiate the attestation, the Verifier generates a nonce N and sends it to the Prover (Step (1) in Figure 3). The Prover then relinquishes control to the RA protocol residing in the hardware-protected component. The Prover's RA protocol reads the device memory contents m (Step (2)) and computes a hash $h = hash(m)$. Next, the Prover concatenates the computed hash h with the received nonce N and authenticates it by computing a keyed Hash Message Authentication Code (HMAC) over the obtained result $s = HMAC(k, (h||N))$. Finally, the memory m and HMAC s are transmitted to the Verifier (Step (3)), which checks whether it corresponds to the transmitted data. The transmission may be split into smaller chunks, e.g., by authenticating individually memory blocks or regions. In that case, integrity, authenticity and temporal

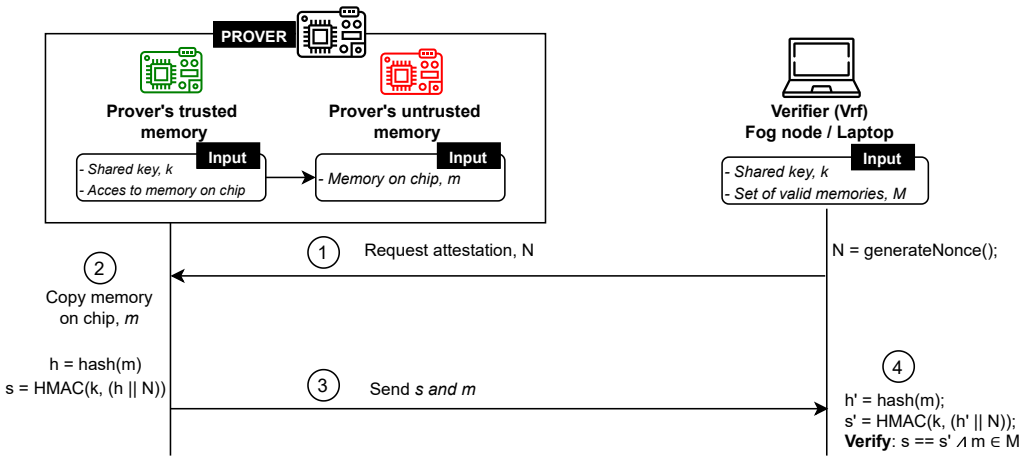

Fig. 3: ERAMO protocol

freshness must be ensured for each transmitted memory chunk, e.g., by adding a unique extra byte for each chunk or securely generating a pseudo-random number inside the Prover.

Verification phase. The verification phase starts when the Verifier receives an attestation response from the Prover. By using the shared attestation key k, the Verifier checks the authenticity and integrity of the attestation result (Step (4)). Assuming that the Verifier knows all valid combinations of memory M, the Verifier has the ability to determine whether a given memory m is in the set M. A powerful Verifier that is able to perform advanced memory forensics analysis (e.g., by using the open-source Volatility Framework [28]) can use the offloaded dynamic memory contents to provide a detailed attestation and precisely determine the Prover's integrity.

Attested memory and device integrity. Figure 4 shows the attested memory regions verified by ERAMO protocol for a device with a flash memory and a memory-mapped peripheral region. A certain portion of the flash region allocates data memory, whereas the memory-mapped peripheral region contains both readable and write-only registers. All readable memory can be attested apart from the secure memory allocated to the trusted component performing attestation.

The inclusion of the aforementioned memory regions in the attestation result is crucial to ensure Prover's integrity. In particular, the attestation of the memory-mapped on-chip peripheral address space guarantees that any on-chip peripheral in use works as intended, and an adversary has not altered the device's peripheral configurations. These configurations may range from the ADC channel chosen, the I2C communication speed, or the internal timer setup. However, due to its dynamic status and configuration registers, this region cannot be attested by the comparison of hashes. Additionally, registers may have unused or reserved bits with undefined read-values, which further complicates the hash verification. Therefore, this memory region should instead offloaded to the Verifier.

Furthermore, if a region of the flash/EEPROM is used for data, such as calibration values or network information, this region may also be verified through offloading. This data may change during runtime and may depend on the electrical characteristics of the specific device, and thus may not be verifiable through hashing. Assuming that the Verifier has some notion of what differentiates legitimate values of this region, the integrity verification of this region is possible through offloading.

Attestation of non-monolithic systems. When the Prover is a non-monolithic platform, its integrity also depends on the integrity of any attached external peripheral devices. Peripheral devices may range from temperature sensors to external digital-to-analog-converters. In this setting, if an adversary manages to change the configuration of a peripheral temperature sensor to provide an inaccurate representation of the temperature, any internal process dependent on this data or any other system to which this inaccurate data is propagated may behave in an unintended manner. Therefore, to accurately verify the Prover, it is necessary to verify the state of any attached peripheral devices.

The peripheral devices are typically not programmable but rely on limited interfaces, such as SPI, I2C, and UART, to read or write to their register contents. These registers, in the same manner as the on-chip peripherals, determine the peripheral configurations and contain their data. Consequently, the contents of these external peripheral registers should be verified to guarantee Prover's integrity.

Before authenticating and offloading the registers' content, first, the trusted component should read the registers. To accomplish this, an extra step is added to the attestation procedure within the trusted component. The Prover uses the peripheral interface (such as SPI or I2C) to read every accessible register on the external peripherals. The contents of these registers now reside within the trusted component and can be offloaded to the Verifier. The Verifier then verifies these external peripherals as it verifies the internal ones. The configuration bits and other data of the peripheral device can be evaluated by considering legal combinations or through more rigorous analysis.

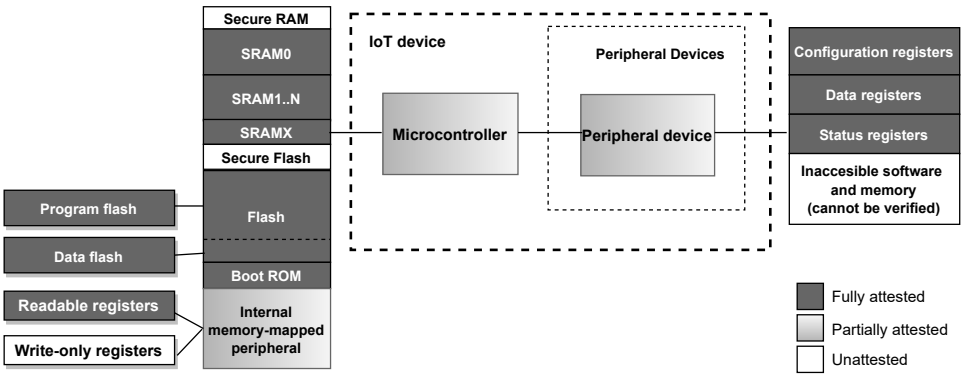

Fig. 4: Device memory regions verified by ERAMO

VII. Implementation details and proof of concept

We implement ERAMO protocol on a TrustZone-capable LPC55S69 running an IoT sensor application on FreeRTOS[1]. This application provides temperature readings at a fixed interval using an external BME280 digital humidity, pressure, and temperature sensor.

To isolate the RA protocol from the non-trusted device application, we use the Armv8-M release of ARM TrustZone. TrustZone is available on various platforms, and unlike other approaches to hardware-based attestation, does not require any external hardware. In general, TrustZone separates the device application into a trusted component allocated to the TrustZone secure world and a non-trusted component allocated in the TrustZone non-secure world. In ERAMO's proof-of-concept implementation, the application is separated as follows:

- **The trusted component** includes the RA procedure and the LPC55S69 hash engine. Additionally, a section of the RAM and flash is allocated to the secure world. The attestation code and key are located in the secure flash, and the key is handled exclusively in secure RAM.
- **The non-trusted component** includes FreeRTOS, the IoT sensor application tasks, and associated interrupts and peripherals. The remaining RAM, flash, and any unused peripheral are also allocated to the non-secure world.

The trusted attestation protocol is initiated by calling it from the non-secure communications thread with the nonce N as an argument (fixed to a length of 8 bytes to prevent inputs of arbitrary length). The protocol performs the authentication using the LPC55S59 on-chip hash-engine because this significantly speeds up the hash computation. The chip supports SHA-1 and SHA-2 with a 256 digest (SHA-256). As SHA-1 has certain vulnerabilities [33] [34], we use SHA-256 for hashing and the HMAC. To prevent key leakage, the hash engine is assigned to the secure world.

Attested memory and device integrity. The developed attestation procedure successfully offloads the RAM, the boot ROM, and the flash memory. Certain sections of the memory-mapped peripheral are write-only, thus, only the readable addresses are transmitted. For this implementation, the I2C peripheral is offloaded, and the Verifier verifies that the I2C interface is configured as intended by verifying the configuration (CFG) register, the interrupt settings (INTENSET), and the settings for the clock and timings (CLKDIV).

The Verifier successfully then verifies the boot ROM and the region of the flash memory containing the program binary by comparing the hashes, before moving on to the dynamic areas of the memory. Currently, no memory forensics is performed on the RAM, however we discuss its feasibility in Section IX.

Attestation of non-monolithic systems. To illustrate the possibilities of verifying non-monolithic devices, the accessible registers of the BME280 external peripheral sensor are offloaded to the Verifier. The trusted attestation procedure on the device performs an I2C burst read on the BME280, resulting in the register contents being transferred to the trusted component's I2C buffer. The burst read is performed using polling to not rely on the interrupts associated with the non-trusted component. The procedure then transmits the memory to Vrf as its internal memory. While offloading the memory allows Vrf to verify with a variety of methods, the current implementation will confirm that certain bits of register 0xF2 (ctrl_hum), 0xF4 (ctrl_meas), and 0xF5 (config) are configured as intended. The dynamic and reserved register are ignored. Additionally, the device-specific calibration values (0xE1-0xF0 and 0x88-0xA1) are read and logged such that they can be compared with internal values used in the device.

Furthermore, analyzing the IoT application used in the implementation, reveals that statically allocated FreeRTOS structures (tasks, queues, timers) are located at a fixed and predictable location in RAM. For instance, the task (thread) responsible for reading the temperature at a fixed interval will be identifiable in the memory dump. This is a very promising property which helps in improving the RA performance, as discussed in Section IX.

VIII. Evaluation

The efficiency of ERAMO highly depends on the choice of hardware. The memory transmissions depend on the choice of communication and its transmissions speed. The time required for authentication depends on Prover's computational capabilities and its available hardware to assist with the process.

[1]https://www.freertos.org

We conducted the experiments, and the runtime measurements of the procedure were measured on the LPC55S69 running at 150MHz. To simplify the connection to the Verifier, a serial connection was established using the on-chip UART configured to a baud rate of 806400. The LPC55S69 hash engine was used to compute the necessary authentication using SHA-256 for hashing and the HMAC. The procedure was tested on different memory sizes, increasing in steps of 1KB. The memory offloaded was the 240KB of non-secure RAM associated with the IoT application.

The time used for the offloading procedure is proportional to the offloaded memory size, as shown in Figure 5. Furthermore, the time used for memory authentication scales the memory size, but it is negligible compared to the time required by data transmission. The duration also scales w.r.t. size but is slightly noisy and requires at least 0.23 ms, as shown in Figure 6.

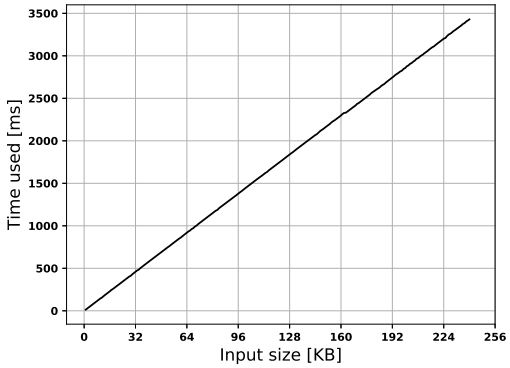

Fig. 5: Time used to transmit memory

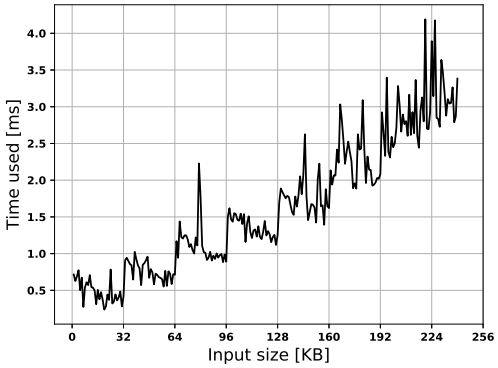

Fig. 6: Time used to authenticate memory

This offloading RA approach can be combined with current methods of static attestation through program checksum comparison. In that case, the code memory (and boot ROM) can be attested through hash comparisons, while offloading the dynamic memory: the RAM, any data region of the flash/EEPROM and internal/external peripheral registers. When combining these methods on the aforementioned IoT sensor application, offloading the RAM, external peripheral registers, and the I2C peripheral, while only transmitting the hash of the flash and boot ROM, the entire process takes 3.94 seconds using the previously specified hardware and communication setup. The experimental results are overall comparable with other RA schemes [32] and confirm the feasibility of ERAMO.

IX. Discussion

ERAMO approach opens the possibilities of various RA schemes, such as allowing ranges of values, complex combinations of settings or even using machine learning to determine the validity of the dynamic memory. Due to the increased computational power of the Verifier, memory forensics tools [28] may be used on the memory dump allowing the Verifier to distinguish a legal state of memory from exploited memory.

Opportunities of multithreaded systems. The recent IoT revolution leads towards multifunction IoT devices that can provide more than one service, e.g., multisensor devices. The result of our analysis that statically allocated FreeRTOS structures (tasks, queues, timers) are located at a fixed and predictable location in RAM is very promising. This insight brings opportunities in improving RA performance in general, and in particular, in memory offloading RA approaches. For instance, by exploiting the knowledge that a given task (thread) allocates a fixed interval, it becomes feasible to perform forensics on the memory and reveal whether it deviates from the legitimate memory. Furthermore, this insight could also be useful in designing a context-aware RA approach, e.g., in a given setting, only certain prioritized tasks will be sent to the Verifier. This could also reduce significantly the RA overhead.

Mobile adversary and memory locking mechanism. While ERAMO aims to improve mobile adversary detection by attesting many memory regions, still during the attestation, a mobile adversary may evade detection by relocating itself in different memory blocks within one memory region. If the trusted component on the Prover deploys a *memory locking* technique, it would be possible to guarantee the result's integrity while allowing the execution of regular operations to run simultaneously with the attestation. Memory locking is already used in Linux to lock memory pages in RAM. Recently, it has been proposed for embedded systems [35].

Using the memory-locking technique, it is possible to lock a memory block, preventing it from being changed until it is again unlocked. This can be used to lock the memory before being attested and gradually unlock it as soon as it is offloaded or attested. Consequently, any malware or effects caused by malware and memory exploits will be locked in memory until it is offloaded, causing it to be detected by the Verifier. Furthermore, TrustZone will ensure that any malware present on the non-trusted component cannot interfere with the trusted offloading procedure, ensuring that the offloading procedure can run securely simultaneous with Prover's regular operations.

X. Conclusions and future works

In this paper, we have provided a first investigation and experimental results on using memory offloading as part of remote attestation. To this end, a new RA protocol (ERAMO) based on memory offloading to verify the Prover's integrity effectively has been designed and presented. The proof-of-concept implementation and the conducted experiments show that ERAMO allows the verification of more dynamic memory areas (such as the internal and external peripheral) which is not covered by existing RA schemes. We believe that these initial results help upcoming research works that explore the memory offloading approach in the RA context.

While advanced forensics tools are required to perform a detailed RAM verification, this paper shows that even a simple implementation of this approach to attestation is able to ensure the integrity of internal and external peripheral, along with any data stored in flash/EEPROM, given that the Verifier possesses knowledge regarding the correct configurations.

As future work, we plan to implement the proposed memory locking scheme and evaluate it w.r.t. its effects on the efficiency of the attestation procedure. Furthermore, it is of interest to investigate how this approach can be used in large scale IoT networks.

References

[1] (2020) State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time. [Accessed 30-April-2021]. [Online]. Available: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/

[2] N. Dragoni, A. Giaretta, and M. Mazzara, "The Internet of hackable things," in *Proceedings of 5th International Conference in Software Engineering for Defence Applications*, P. Ciancarini, S. Litvinov, A. Messina, A. Sillitti, and G. Succi, Eds. Springer, 2017, pp. 129–140.

[3] H. Hu, S. Shinde, S. Adrian, Z. L. Chua, P. Saxena, and Z. Liang, "Data-Oriented Programming: On the Expressiveness of Non-control Data Attacks," in *IEEE Symposium on Security and Privacy*, 2016.

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

[5] X. Carpent, G. Tsudik, and N. Rattanavipanon, "ERASMUS: Efficient remote attestation via self-measurement for unattended settings," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018.

[6] M. De Donno, K. Tange, and N. Dragoni, "Foundations and evolution of modern computing paradigms: Cloud, iot, edge, and fog," *IEEE Access*, vol. 7, pp. 150 936–150 948, 2019.

[7] H. Shacham, "The Geometry of Innocent Flesh on the Bone: Return-into-Libc without Function Calls (on the X86)," in *ACM CCS'07*, 2007.

[8] (2017) Devil's Ivy. [Accessed 30-April-2021]. [Online]. Available: https://blog.senr.io/devilsivy.html

[9] T. Abera, N. Asokan, L. Davi, J. Ekberg, T. Nyman, A. Paverd, A. Sadeghi, and G. Tsudik, "C-FLAT: control-flow attestation for embedded systems software," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016.

[10] G. Dessouky, S. Zeitouni, T. Nyman, A. Paverd, L. Davi, P. Koeberl, N. Asokan, and A. Sadeghi, "LO-FAT: Low-overhead control flow attestation in hardware," in *Proceedings of the 54th Annual Design Automation Conference (DAC)*, 2017.

[11] S. Zeitouni, G. Dessouky, O. Arias, D. Sullivan, A. Ibrahim, Y. Jin, and A. R. Sadeghi, "Atrium: Runtime attestation resilient under memory attacks," in *2017 IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, 2017, pp. 384–391.

[12] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "SWATT: software-based attestation for embedded devices," in *IEEE Symposium on Security and Privacy*, 2004.

[13] A. Seshadri, A. Perrig, M. Luk, L. Van Doom, E. Shi, and P. Khosla, "Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems," *Operating Systems Review (ACM)*, vol. 39, no. 5, pp. 1–16, 2005.

[14] K. Eldefrawy, D. Perito, and G. Tsudik, "SMART: Secure and minimal architecture for (establishing a dynamic) root of trust," in *The Network and Distributed System Security Symposium (NDSS)*, 2012.

[15] P. Koeberl, S. Schulz, V. Varadharajan, and A. Sadeghi, "TrustLite: A security architecture for tiny embedded devices," in *Proceedings of the Ninth European Conference on Computer Systems (EuroSys)*, 2014.

[16] F. Brasser, B. El Mahjoub, A. Sadeghi, C. Wachsmann, and P. Koeberl, "TyTAN: Tiny trust anchor for tiny devices," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[17] G. Dessouky, T. Abera, A. Ibrahim, and A.-R. Sadeghi, "LiteHAX: Lightweight Hardware-Assisted Attestation of Program Execution," in *Proceedings of the Int. Conf. on Computer-Aided Design*, 2018.

[18] N. Asokan, F. Brasser, A. Ibrahim, A. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann, "SEDA: Scalable Embedded Device Attestation," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2015.

[19] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A.-R. Sadeghi, and M. Schunter, "SANA: Secure and Scalable Aggregate Network Attestation," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security CCS '16*, 2016.

[20] E. Dushku, M. M. Rabbani, M. Conti, L. V. Mancini, and S. Ranise, "SARA: Secure Asynchronous Remote Attestation for IoT Systems," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3123–3136, 2020.

[21] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *European Conference on Computer Systems, Proceedings of the Sixth European conference on Computer systems (EuroSys '11)*, 2011.

[22] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente, "On the Difficulty of Software-Based Attestation of Embedded Devices," in *Proceedings of ACM CCS '09*, 2009.

[23] S. F. J. J. Ankergård, E. Dushku, and N. Dragoni, "State-of-the-Art Software-Based Remote Attestation: Opportunities and Open Issues for Internet of Things," *Sensors*, vol. 21, no. 5, 2021.

[24] C. Kil, E. C. Sezer, A. M. Azab, P. Ning, and X. Zhang, "Remote attestation to dynamic system properties: Towards providing complete system integrity evidence," in *2009 IEEE/IFIP Int. Conf. on Dependable Systems Networks*, 2009, pp. 115–124.

[25] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a tcg-based integrity measurement architecture," in *13th USENIX Security Symposium*, 2004.

[26] H. Hu, S. Shinde, S. Adrian, Z. L. Chua, P. Saxena, and Z. Liang, "Data-oriented programming: On the expressiveness of non-control data attacks," in *IEEE Symposium on Security and Privacy (SP)*, 2016.

[27] M. Conti, E. Dushku, L. V. Mancini, M. Rabbani, and S. Ranise, "Remote Attestation as a Service for IoT," in *2019 Sixth Int. Conf. on Internet of Things: Systems, Management and Security*, 2019.

[28] (2021) The Volatility Foundation. [Accessed 30-April-2021]. [Online]. Available: https://www.volatilityfoundation.org/

[29] S. Pinto and N. Santos, "Demystifying Arm TrustZone: A Comprehensive Survey," *ACM Comput. Surv.*, vol. 51, no. 6, 2019.

[30] W. Arthur and D. Challener, "A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security," 2015.

[31] T. Abera, N. Asokan, L. Davi, F. Koushanfar, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "Invited – Things, Trouble, Trust: on Building Trust in IoT Systems," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 121.

[32] M. Ambrosin, M. Conti, R. Lazzeretti, M. Rabbani, and S. Ranise, "Collective Remote Attestation at the Internet of Things Scale: State-of-the-art and Future Challenges," *IEEE Communications Surveys & Tutorials*, no. c, pp. 1–1, 2020.

[33] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The First Collision for Full SHA-1," in *Advances in Cryptology*, 2017.

[34] G. Leurent and T. Peyrin, "Sha-1 is a shambles: First chosen-prefix collision on sha-1 and application to the PGP web of trust," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[35] K. Eldefrawy, X. Carpent, G. Tsudik, and N. Rattanavipanon, "Temporal consistency of integrity-ensuring computations and applications to embedded systems security," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security (ASIACCS)*, 2018.