SPEED: Secure Provable Erasure for Class-1 IoT Devices

Mahmoud Ammar
imec-DistriNet, KU Leuven
mahmoud.ammar@cs.kuleuven.be

Wilfried Daniels
imec-DistriNet, KU Leuven
wilfried.daniels@cs.kuleuven.be

Bruno Crispo
imec-DistriNet, KU Leuven
University of Trento, Trento, Italy
bruno.crispo@cs.kuleuven.be

Danny Hughes
imec-DistriNet, KU Leuven
danny.hughes@cs.kuleuven.be

ABSTRACT

The Internet of Things (IoT) consists of embedded devices that sense and manage our environment in a growing range of applications. Large-scale IoT systems such as smart cities require significant investment in both equipment and personnel. To maximize return on investment, IoT platforms should support multiple third-party applications and adaptation of infrastructure over time. Realizing the vision of shared IoT platforms demands strong security guarantees. That is particularly challenging considering the limited capability and resource constraints of many IoT devices.

In this paper, we present SPEED, an approach to secure erasure with verifiability in IoT. *Secure erasure* is a fundamental property when it comes to share an IoT platform with other users which guarantees the cleanness of a device's memory at the beginning of the application deployment as well as at the time of releasing the underlying IoT device. SPEED relies on two security primitives: memory isolation and distance bounding protocol. We evaluate the performance of SPEED by implementing it on a simple bare-metal IoT device belongs to Class-1. Our evaluation results show a limited overhead in terms of memory footprint, time, and energy consumption.

ACM Reference Format:
Mahmoud Ammar, Wilfried Daniels, Bruno Crispo, and Danny Hughes. 2018. SPEED: Secure Provable Erasure for Class-1 IoT Devices. In *CODASPY'18: Eighth ACM Conference on Data and Application Security and Privacy, March 19–21, 2018, Tempe, AZ, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3176258.3176337

1 INTRODUCTION

The IoT envisions a future where billions of Internet-connected devices are deployed in our environment to support novel Cyber-physical applications. Contemporary IoT networks are large and growing in scale from smart buildings to smart cities. Research deployments such as City of Things [20] already incorporate tens of thousands of IoT devices. The majority of such devices are very tiny and belong to Class-1 [7]. The Internet Engineering Task Force (IETF) identifies Class-1 IoT devices with 10KB RAM and 100KB ROM as having the minimal resources necessary to communicate securely with the Internet [7]. The ideal multi-app IoT platform would execute efficiently on typical Class-1 embedded devices and enable the secure execution of coexisting third-party applications.

However, commercial deployments of similar scale have been slow to appear. One reason for this slow adoption is the unclear Return-on-Investment (RoI) for large-scale IoT networks that demand significant upfront investment in the infrastructure as well as technical staff to deploy, manage and maintain the system. Supporting multiple applications enables IoT infrastructure providers to increase their RoI. Multi-app nodes allow an IoT deployment to satisfy multiple stakeholders and therefore to cover hardware and associated staff costs arising from the deployment, management and maintenance of the infrastructure. IoT infrastructure providers could, for example, lease out resources on underutilized devices to third parties to increase revenue or specialize in deploying IoT infrastructure as a service. Realizing the idea of shared IoT platforms requires security mechanisms that ensure among other things that: (i) the memory of a public shared IoT device does not store any unwanted software or malware, (ii) the user can delete the entire memory footprint when he releases the IoT device, and (iii) any user can use the public IoT platform, so no complex key distribution and management is required.

This paper addresses the problem of *secure remote erasure of IoT with verifiability* without depending on pre-shared secret keys. A proof of secure erasure (PoSE) is the ability of the end user to verify the outcome of the erasure operation of a black-box system such as a remote IoT device. In spite of its importance, the topic of the provable secure erasure in IoT has been neglected and started to attract attention recently by some proposals [15]. This is an important omission that we aim to address in this research work. Our approach, SPEED, targets Class-1 IoT devices [7]. We build on two security primitives: (i) *memory isolation* and (ii) *distance bounding protocol* (DB). An isolated and secure portion of memory is needed to store the security-relevant functions for deletion and communication with the outside world. DB is implemented in this trusted part of the memory, and needed for proximity-based authentication and preventing man-in-the-middle (MITM) attack, as explained in Section 5. Henceforth, the term *verifier* refers to the user who wants to erase the memory of a target IoT device and verify the outcome of this remote erasure operation; and the term *prover* refers to the target IoT device that has to give a proof of erasure. In particular, the verifier should be able to securely erase the prover's memory and get a proof of erasure if he is in vicinity. The prover has to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CODASPY '18, March 19–21, 2018, Tempe, AZ, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5632-9/18/03…$15.00
https://doi.org/10.1145/3176258.3176337

verify the distance of the verifier and give a proof of secure erasure if the verifier's distance is less than the predefined maximum threshold, where the threshold is an application-dependent value. Please, notice that there is an interesting swap of roles between the verifier and the prover, where the verifier has to verify the secure erasure and the prover has to verify the distance bounding.

To sum up, the chief contributions of this paper are:

- Overcoming the security limitation of the majority of Class-1 IoT devices, identified by the lack of the memory protection unit (MPU), by designing and developing an efficient memory isolation technique for low-end embedded devices that acts as a software-based MPU.
- Designing a flexible and secure memory erasure primitive, that can guarantee the deletion of any memory contents without any predefined information about the device (e.g. type, size of memory, shared keys, etc.). The only requirement is that the device should run our software-based memory isolation mechanism or some related functions only if it has a hardware-based MPU.
- Advancing the secure erasure in the IoT by brining it closer to reality through a proper implementation of SPEED on one of the Class-1 IoT devices and showing the efficiency and practicality of using DB to prevent MITM attack instead of selective jamming, as assumed by other research papers.

Paper outline. The remainder of this paper is organized as follows. Section 2 reviews the related work. Design principles of memory isolation are presented in Section 3. Section 4 describes the chosen distance bounding protocol. Section 5 proposes SPEED, our approach to secure erasure. Implementation details and evaluations are reported in Section 6 and 7 respectively. Section 8 concludes and gives directions for future work.

2 RELATED WORK

In 2010, Perito and Tsudik proposed an approach called Proofs of Secure Erasure (PoSE) [15]. PoSE is a protocol to perform secure erasure and secure code update. Both mechanisms are interleaved since the secure erasure can be considered as a prelude to secure code update. Remote attestation can be applied by erasing the memory each time and updating the code again. PoSE takes advantage of the flash memory which is common in all embedded devices by designating a small portion of it to be read-only. This small ROM on the prover side hosts the main functions needed for interacting with the verifier and erasing the contents of memory. The verifier starts the protocol by sending true randomness in order to fill and overwrite the prover's memory. The last k bits of these randomness are used as a session key by the prover to compute the message authentication code (MAC) of the memory and send it back to the verifier. On the other side, the verifier computes his own MAC and compares it with the received one. If they match, then proof of secure erasure holds. The security parameter k is known in advance to both parties. Also, the size of prover's memory has to be known in advance to the verifier. To prevent man-in-the-middle attack, PoSE relies on the assumption of jamming all other nearby devices during the run of the protocol. Moreover, the protocol incurs high overhead in terms of communication as the verifier has to send random bytes equal to the size of the entire writable memory of the target IoT device.

Dziembowski et al. [10] proposed a cryptographic scheme that minimizes the communication complexity of PoSE [15]. The verifier sends a few number of bits (seed) to the prover. Using this seed, the prover performs a set of deterministic computations and expansion functions (e.g. calculation of a hash function recursively) that require the usage of the whole memory and thus overwriting its content. The secure erasure is proved if the computed hash value is correct as it can only be generated once. The idea behind this is that the prover stores a secret key in its memory. The size of this key should be at least half of the size of the available memory. Assuming bounded-retrieval model and restricted write/read operations, the adversary can not leak the key or make a copy of it internally due to its size. This key will be (at least partially) destroyed after recursively executing a set of hash functions. Therefore, the adversary can not preform the computations again. The main drawback of this solution is the computational complexity. It is quadratic on the size of the prover's memory.

Karame et al. [12] introduced a lightweight version of the previously mentioned PoSE [15] approach which reduces the overhead of computations. This scheme does not rely on the computation of MAC but on the correct structure of data in the prover's memory. The verifier selects a random secret K and a seed s of size m bits each. Then, he generates n random data blocks of length m bits each, where the result of multiplying m and n equals the size of the writable memory at the prover side. The verifier computes K' after performing a procedural set of cyclic shifts and XOR functions. The prover has to compute and send K back after receiving the n random data blocks in addition to the s and K' from the verifier. If both values of K match, the proof of secure erasure holds due to the idea of designing the ShiftXOR function, where K can not be computed correctly on the fly without storing the random data blocks in their exact locations. Similarly to PoSE, this approach still depends on the assumption of selective active jamming of nearby devices and does not solve the high overhead of the number of the transmitted packets.

In nutshell, we show that SPEED enhances over all of the proposed approaches in the following:

- Excluding the assumption of selective jamming to prevent MITM attack. Selective jamming of all other nodes around the prover is difficult to implement and suffers from some limitations (e.g. illegal at the standard wireless bands, not fully secure and can be bypassed [16], cost ineffective which requires an extra hardware, etc.).
- The communication overhead required is very minimal which limited to the exchange of few bytes regardless the size of the remote IoT device's memory. The computational overhead also is limited to the computation of a simple MAC function. Moreover, no prior keys have to be stored and known in advance.
- Suitable for a wide range of legacy IoT devices, where the implementation of SPEED requires a minimal memory footprint and takes into account the lack of hardware security features.

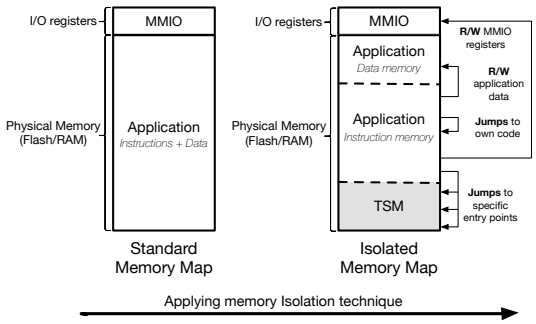

Figure 1: Overview of the unrestricted standard memory map and the restricted memory map after applying isolation technique.

3 MEMORY ISOLATION

Memory isolation is a security feature that provides a way to control memory access rights and prevent damage or leakage of private data through the unauthorized access that could happen by a software bug or a malware infection. The class of IoT devices (e.g. IETF Class-1), we are targeting, depends on the commercial off-the-shelf (COTS) microcontrollers that are optimized for low cost and low power consumption. So, they are deployed without any security properties. Furthermore, the existing hardware protection mechanisms can not be applied on such devices due to the restrictions of their own hardware architectures.

The basic property required to implement any form of secure erasure is memory isolation. It allows to control the memory addresses and reserve a part of it to store the private data and the security-related functions. Therefore, as a first step to build the secure erasure approach, we had to overcome the problem of lacking MPUs in Class-1 IoT devices by designing and implementing a pure software-based memory isolation technique. This security feature is required to isolate and guarantee the integrity of the Trusted Software Module (TSM) that running within a single address space from other untrusted software modules. The TSM refers to the cryptographic primitives, secrets keys, private data, and all other helping functions that we trust to execute. Our technique can be easily applied to any MCU with the following characteristics: (i) has no memory protection unit, (ii) supports disabling of global interrupts to ensure atomic execution of SPEED, (iii) does not support multi-threading, and (iv) still has sufficient flash memory to store the TSM code as a requirement of SPEED.

Design of Software-based Memory Isolation. Our design of the memory isolation shares similarities with the Software-based Fault Isolation (SFI) approach proposed by Whabe et al. [19]. As SFI approach has been designed for computer systems, we are taking advantages of it by designing and implementing a pure software-based memory isolation approach for embedded systems in an optimized way.

We use selective software virtualization and assembly-level code verification to provide sandboxing between software modules. At initialization time, the TSM should be installed using a physical programming device (e.g. JTAG) through an edited toolchain. Upon successful installation, the occupied portion of the memory by TSM acts like an isolated and *virtual* ROM. The TSM memory can not be written or even read after the deployment of the microcontroller without a physical access. Furthermore, the execution of this area is only allowed from specific entry points as we see later. Therefore, the access to this part of the memory is protected by the employed virtualization mechanism. The TSM has no restrictions at all and has full access over other parts of the memory. As shown in Figure 1, the remaining part of the memory, denoted as *Application Memory*, should host other (untrusted) software modules. It consists of two subareas: the *Instruction Memory*, that holds the application code, and the *Data Memory* for data. In contrast to the TSM memory, the *Application Memory* is subject to the following restrictions:

- *Control Transfer:* branch and jump operations can only target either the instruction memory or specific entry points in the TSM memory.
- *Read and Write:* read and write operations address only the data memory and Memory Mapped IO (MMIO) registers.
- *Deployment:* updating the application memory can only occur by the TSM. Restrictions on the application code are enforced at the instructions level by *verifying* the adherence of the instructions to the listed rules and replacing unsafe and essential instructions by *safe virtualized* ones. Applications that violate the rules are rejected instantly by the TSM.

During the deployment process of an application, the TSM takes care of checking each instruction at the assembly level. Two types of illegal instructions can be identified:

- *static jump operations:* instructions that have a target address encoded in it statically, and this target address refers to a restricted point of memory.
- *dynamic jump operations:* instructions that access any memory dynamically as the target address is encoded in a pointer register.

Any application with at least one unsafe instruction will be rejected by the TSM. The vast majority of the control transfer instructions (e.g. program counter relative branches) have a direct target address and thus they can be checked statically at deployment time. Dynamic jump operations that use indirect addressing mode will be replaced by virtual safe instructions. The actual verification of their safety takes place at runtime rather than the deployment time. If at least one of these operations is unsafe (e.g. access restricted memory), the MCU will perform a soft reset preventing illegal operation. From a practical point of view, replacing the unsafe indirect call with a safe virtual indirect call takes place between the assembly and linking stages, as we add a post-processor to do this substitution during the compilation of the application code.

4 DISTANCE BOUNDING PROTOCOL

Distance Bounding (DB) protocols allow to establish an upper-bound on the physical distance between two parties which are typically denoted as *verifier* and *prover*. The first DB protocol was introduced by Brands and Chaum [8] in order to prevent mafia fraud (relay) attacks on bank ATMs [9], a special version of MITM attack. Brands and Chaum's DB is a challenge-response protocol which estimates the distance between two entities by measuring the round-trip time (RTT) of challenges and responses that travel at

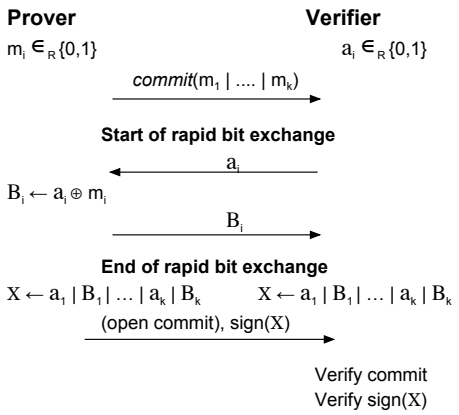

Figure 2: Distance bounding protocol of Brands and Chaum

the speed of light. Relying on this fact, we guarantee to obtain the upper bound of the distance between two parties, as the dishonest party can not claim to be closer than he really is because nothing propagates faster than the light. Thus preventing distance fraud attack too. The relay attack is prevented by forcing the adversary to come closer to the victim, which increases the possibility of being detected even by offline methods (i.e. visual detection).

An important feature of implementing distance bounding protocol is to rule out the assumption of selective jamming. To the best of our knowledge, all related approaches depend on this assumption. Jamming techniques work in theory as they depend on the noise level or channel properties, and such methods are not easy to implement [11]. Also, jamming techniques have a negative impact on the network performance and can be bypassed as demonstrated in [16]. On the other hand, Time Of Flight (ToF) based distance bounding protocols (e.g. the one we use) are more reliable, and often used to authenticate and evaluate the distance of the node in many applications such as keyless entry systems in vehicles, RFID door access systems, payment systems, and real time location systems (e.g. RADAR) [6].

Figure 2 shows the basic principles of the distance bounding protocol proposed by Brands and Chaum. The protocol encompasses three general phases. In the first phase, both the verifier and the prover generate a series of random bits. The number of bits depends on a chosen security parameter, k, that expresses the degree of confidence desired. The prover has to commit to the chosen values using a secure commitment scheme in order to avoid cheating. The backbone of the protocol is the second phase, which is the rapid exchange of bits to measure the distance, where both parties exchange challenges and responses represented as single bits. The basic idea is to precisely measure the round-trip time between two unpredictable messages (a challenge and a response). The process is repeated k times and each time the verifier computes the elapsed time. The verifier's challenges are unpredictable and each response has to be computed as a function of the corresponding challenge to ensure sending it after receiving the correct challenge. This helps in estimating the upper bound of the real distance of the prover node. Since we are interested in the propagation time, the processing delay should be very small and negligible compared to the time of flight, taking into account that 1 nanosecond processing time yields 30 cm accuracy of RTT (15 cm of propagation time). This prevents a computationally powerful malicious node claiming false position. The proposed operation in the protocol is a simple XOR between two bits. In the last phase, the prover opens the commit. The verifier verifies it and computes the upper bound of the distance according to the following equation:

$$(\frac{max(RTT_i) - \alpha}{2}) \times C$$

where α is the processing time and C is the speed of the light.

If node authentication is required, public key cryptography can be used in the last phase to sign and verify the exchanged nonces. Public key identification schemes such as Fiat-Shamir can also be used for authentication, as described in the original paper [8].

5 SPEED

SPEED is a challenge-response protocol dedicated for secure erasure by allowing a node (e.g. a user of mobile phone) to securely and remotely delete the memory of another node (e.g. IoT device) and to get a proof of secure erasure without any pre-knowledge in advance. PoSE is given by the construction of SPEED itself without the requirement to run other protocols. Node authentication occurs by verifying the proximity of the two corresponding nodes through the implementation of a secure version of the DB protocol.

We start by outlining the attacker model. We then introduce and analyze SPEED, and explain the working mechanism behind the integration of the aforementioned building blocks in Section 3 and Section 4.

5.1 Adversarial Model

As a consequence of the advances in the IoT domain, attention started to shift from designing architectures relying on dedicated infrastructure to new trends of design using shared and distributed infrastructure [3, 14]. Hence, our case reference is that a single infrastructure provider deploy a set of smart devices for the public use as a shared hardware where any user can deploy his software and data over a custom IoT device for a period of time. Upon accomplishing the work, the user releases the underlying IoT device after verifying the cleanness of its memory and that nothing is left in it. Also, the user can perform the secure erasure of the memory before the software deployment in order to ensure that it doesn't store any unwanted software or malware.

We consider two types of adversaries based on the recently proposed taxonomy [1]:

- Remote adversary: exploits software bugs remotely at the prover side to infect it with malware.
- Local adversary: a standalone device, located in vicinity of the prover to eavesdrop on and interfere with the prover's communication.

As most other related work, we rule out all types of physical attacks. Also, denial-of-service (DoS) attacks are beyond the scope of this paper. Furthermore, we assume that TSM code is bug and exploit free and is deployed on the IoT device by a trusted party.

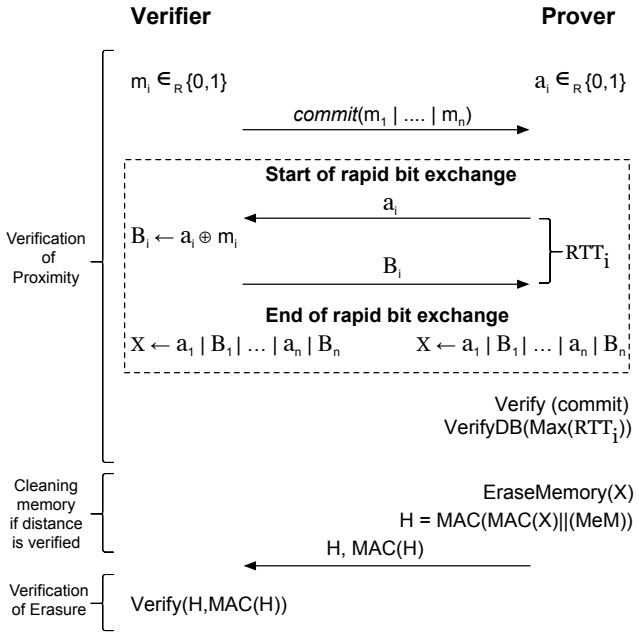

Figure 3: Overview of SPEED

5.2 Design Rationale

SPEED relies on the integration of two security primitives; memory isolation and distance bounding protocol. The latter is needed for the prover to verify the distance of the verifier, where the verifier is the node that wants to clean the memory of another node (prover) and get a proof of erasure. If the verifier is within range, the target device (prover) executes the request of secure erasure, otherwise, no change occurs. All necessary functions needed to run SPEED are located in the TSM. Memory isolation is guaranteed by the sandboxing technique we designed and implemented in Section 3.

Measuring the round-trip-time (RTT) of a given message provides a bound on the distance and this helps in preventing MITM attack and all of its related forms (e.g. mafia-fraud attack) since the simplest technique of relaying messages consumes an extra time and consequently results in a longer distance. The security of the DB protocol, we implement, is verified in [17] and thus it prevents against MITM attack as measuring RTT according to the speed of light can't be spoofed even if the adversary has a powerful hardware. This advantage of the DB replaces the assumption of using selective jamming, where DB is simpler to implement on the resource-constraint devices and easier to prove its security.

Figure 3 illustrates SPEED. Denoting the verifier as v and the prover (target device) as ρ, the protocol starts by generating random nonces of length n bits at both sides. v sends to ρ the hash value (the commit) of the chosen nonce to avoid cheating of sending responses in the future. Upon receiving the commit, ρ starts the rapid bit exchange phase by sending a challenge composed of one bit and wait for a response from v. The rapid bit exchange process is repeated n times. Assuming that we are in the ith iteration, the response is the result of XORing the two corresponding bits (the ith challenge bit and the ith bit in v's nonce). By the end of this phase, ρ knows the maximum round-trip time (RTT) and is able to compute the hash value of the nonce generated by v. Hence, ρ can verify the received commit (*e.g. Verify(commit)*) and establish an upper bound of v's distance (*e.g. VerifyDB(Max(RTT_i))*). If both verification steps are passed successfully, the secure erasure of the memory takes place by resetting the value of each byte in the memory to be equal to either a default value or to a value of one of the random bytes in the nonces (*e.g. EraseMemory(X)*). In the last step, ρ generates a session key by computing the MAC value of the alternating bits-concatenated nonces and then computes the message authentication code (MAC) of the entire memory using this key (*e.g. H = MAC(MAC(X)||(MeM))*). ρ sends the result back to v along with its MAC value to ensure integrity. Finally, v verifies the outcome by following the same deterministic steps of computing the MAC. If both values are equal, secure erasure has occurred. We notice that our protocol has minor modifications with regards to the original DB. v does not have to open the commit after the rapid bit exchange phase, as we use the same public hash function used for computing the MAC of the memory to compute the hash value of the chosen randomness by v in the initial phase. ρ can verify it easily after the second phase. Moreover, we do not consider the authentication using public key cryptography since our scenario simulates the case where there is no prior knowledge between v and ρ. Any v close to any ρ can execute the secure erasure of the memory. The execution of SPEED occurs only from a valid entry point and is not interruptible as all global interrupts are disabled prior to the start and activated again after clearing the RAM from all temporary variables used during the run of the protocol.

5.3 Security Analysis

To analyse the security of SPEED, we first present the existing and related attacks on both Distance bounding [6] and similar challenge-response (e.g. Remote Attestation) protocols [18] and then defend against them considering the aforementioned adversarial model (Section 5.1) and under the following assumptions:

- *AS1*: We assume a source of true and unpredictable randomness on both sides, and all responses depends on the corresponding challenges (e.g. *r = f(c)*).
- *AS2*: The cryptographic hash function used for commitment and computing MAC is secure.
- *AS3*: The TSM guarantees that the prover device can only communicate with the verifier during the run of SPEED (e.g. single uninterruptible thread of execution).

5.3.1 Security of TSM. The software-based memory protection unit (TSM), that we implement, is the core of providing the security of SPEED. SPEED, similarly to all other challenge-response protocols, can be vulnerable to one or more of the following attacks [18]:

- Precomputation attack: The ability of the adversary to predict the challenges sent by the IoT device (e.g. the node (ρ) that wants to verify the distance of another node (v)) and precompute valid responses in advance.
- Replay attack: The ability of the adversary to eavesdrop to the correct outcome of the erasure routine from a non-compromised node (ρ), store it, and then reply it when needed.

- Forgery attack: The adversary may alter the erasure routine to forge a real but not valid outcome from (ρ) in order to give a fake proof of erasure.
- Impersonation attack: The adversary might impersonate a genuine node (ρ) and send valid but fake proofs of erasure to v.
- Proxy attack: A dishonest node (v) might relay the challenges to a more powerful node that is able to compute a valid response in its behalf.
- Collusion attack: The ability of a compromised node (ρ) to collude with other nodes to provide a valid proof of erasure without erasing the actual content of memory.
- Memory copy attack: If there is an enough free space in the prover's memory, the adversary can keep a copy of the original memory contents. Then, he can modify the erasure routine so it computes a response over the memory locations where his copy is not maintained.
- Compression attack: The adversary might save a compressed version of the prover's memory in a random location and decompress it after the execution of the erasure routine. It is a special case of the memory copy attack.
- Return-oriented programming (ROP) attack: the adversary may modify the control-flow of the prover's code to execute arbitrary operations by linking together small sequences of instructions, called gadgets, without making any changes to the program memory.

Precomputation and replay attacks are no longer feasible as *AS1* guarantees that the challenges are unpredictable and the hash value of the memory is variable since it depends on the nonce generated by these challenges.

The forgery attack aims to overcome the secure erasure routine itself and tamper with the TSM code. This is prevented by the access rules enforced by the TSM at deployment time, where read and write operations to this part of the memory are not allowed, whereas execution can only occur from specific entry points after disabling all global interrupts to ensure atomicity. Moreover, *AS1* and *AS2* guarantee that MAC values can't be forged.

Impersonation attack is prevented and can easily be detected using either offline methods (e.g. visual detection) as we are considering small distances or by the responses which would reflect an invalid value of MAC.

Proxy and collusion attacks are not valid under the adapted threat model and *AS3*. Moreover, Proxy attack is already prevented by the countermeasure of the terrorist fraud attack as we see later.

Memory-copy, compression, and ROP attacks have the same goal as the forgery attack, aiming to modify the TSM code and alter with the secure erasure routine. This is totally prevented by the employed sandboxing mechanism under the aforementioned threat model (e.g. No physical access). Nevertheless, any bug of the deployed user application (e.g. stack overflow) increases the possibility of ROP attack to occur. However, this attack is implementation-specific and requires high efforts to cause damage. The security of the TSM still can not be broken by this attack. However, if succeeded, it executes arbitrary lines of the TSM code without modifying or altering it. This means that TSM still guarantees to provide a valid proof of erasure by the end of a correct execution of the secure erasure routine.

5.3.2 Security of Distance Bounding. There are four major attacks that threaten the security of the DB protocol:

- Impersonation: An impersonation fraud is an attack where an adversary acting alone purports to be a legitimate prover.
- Distance fraud: The claim of a dishonest party (v) to be closer than (s)he really is.
- Mafia fraud: A special type of man-in-the-middle attack, in which, the third party is passive and simply relays messages regardless of their content.
- Terrorist fraud: A variant of mafia fraud attack, in which, the prover (e.g. in our case: v) colludes with the adversary to deceive the verifier (e.g. in our case: ρ) that (s)he in vicinity without disclosing the secret key to the adversary, where the adversary performs the first two phases of the protocol (e.g. see Figure 2) and the dishonest prover (v) performs only the signing phase.

Countermeasures to such attacks are already proposed and discussed in literature [6, 17] and hence we just implement them.

In our case, the impersonation attack is useless as all devices can act as legitimate provers (v) since authentication using secret/private key is not required.

The RTT distance-estimation technique, that we use, depends on precise timing, as a deviation of a few nanoseconds affects negatively on the estimated distance (e.g. 6 ns add an extra 1 meter). Depending on this strict timing property and taking into account *AS1*, we prevent mafia fraud attack as even a simple relay of messages unavoidably adds an extra time and thus results in longer distance. Accordingly, distance fraud attack is prevented too.

In SPEED, we do not depend on secret or private keys to authenticate the prover as we demonstrate our scenario in the public space and anyone in vicinity of the IoT device can delete it's memory. Therefore, in this case, terrorist fraud attack is the same as mafia fraud attack and thus it is implicitly prevented.

6 IMPLEMENTATION

We implemented a prototype of SPEED on an 8-bit AVR ATmega 1284p microcontroller (MCU) [4] mounted on the MicroPnP IoT platform [22]. This MCU belongs to Class-1 devices which runs at 10 MHz, with 16 KB of SRAM, 4 KB of EEPROM, and 128 KB of flash memory. In addition to the MCU, MicroPnP offers an IEEE 802.15.4e [21] radio for wireless communication. In contrast to *von Neumann* architecture, the AVR family of microcontrollers uses the modified Harvard architecture, where MMIO, instructions and data memory are physically separated and not mapped on to a single address space. The sandboxing techniques we implement, in SPEED, can be applied to both architectures without any limitations.

The secure erasure of a device's memory may be considered as a prelude or a consequence of software deployment. We consider the secure deployment of software out of scope in this paper. However, in the implementation of the memory isolation, we take care that the current untrusted application in the *Instruction memory* should not violate the restricted rules explained in Section 3. This means that the software installation should pass through some functions

in the TSM in order to ensure the adherence to the rules and the success of secure erasure later on.

From a programming point of view, we have two entry points in the TSM (e.g. two public functions), from which we can execute the code. The first entry point is called the *Loader*. The *Loader* function is responsible to load the produced binary image of the application in a temporary area in the application memory and verify its instructions to see if they adhere to the rules or not. Verification is done line by line at the assembly level. If even only one statement violates the rules (e.g. jump to a restricted point in the TSM), the whole application is rejected. As argued before, in a standard toolchain, producing the binary image of the user application goes through the compiler, the assembler, and then the linker. We edited the toolchain by adding a post-processor between the assembler and the linker in order to replace all unsafe dynamic instructions with safe virtual ones. These instructions cannot be checked statically by the *Loader* function at the deployment time and therefore they are checked at runtime. Any violation caused by one of these instructions makes the device restarting itself in order to avoid the unauthorized execution of the malicious code.

The second entry point is the *SecureErasure* function, using which, we execute the erasure routine described in Figure 3. All related functions are located in the TSM memory and can be executed through this entry point. The protocol starts by disabling all global interrupts. Therefore, the correct and atomic execution of this function is guaranteed. The verification of secure erasure requires the computation of the message authentication code (MAC) of the entire memory. We developed two versions of our system. In the first one, we use an optimized implementation of the sponge Keccak-256 (SHA-3 standard) function in the assembly level to compute the MAC. In the second, HMAC-SHA1 is used. The same hash function is used for computing the commit in DB and verifying it. The output of Keccak is 256 bits long, whereas HMAC-SHA1 output is 160 bits long. We considered a length of 128 bytes for generating nonces which can be adjusted according to the specific domain of application. At the end of the rapid exchange bytes process, both parties have a nonce of length $2n$ (256 bytes).

7 EVALUATION

We evaluate SPEED according to many factors: (i) performance, (ii) memory footprint, (iii) power consumption, and (iv) the accuracy of estimating the distance between two parties.

7.1 Performance

The main parameters that affect the time overhead of SPEED are:

- **Number of exchanged bits**. Launching SPEED requires exchanging a limited number of bits (seed) between the verifier and the prover. The length of this seed does not depend at all on the size of the memory and can be adjusted by the user, where the longer the sequence of bits exchanged, the more accurate the measurement is. In our experiment, we considered a length of 128 bytes as a seed. The objectives of this seed are manifold: (i) establishing the upper-bound of the distance between the verifier and the prover, (ii) using it as a nonce to satisfy the freshness property, and (iii) generating a session key to compute the corresponding MAC. In contrast to other existing approaches [12, 15], SPEED has the advantage that it does not require exchanging a number of bits equals to the size of prover's memory to overwrite it. Thus, it incurs very small communication overhead.
- **Memory access time**. SPEED requires accessing each byte address in the memory twice. First, to erase it's content. Second, to compute the MAC. The speed of accessing the memory mainly relies on the clock rate of the MCU. In our experiment, the microcontroller operates at 10 MHz. This means that the total time of accessing the memory accounts just for a small fraction of the total run time of SPEED.
- **Computation of MAC**. Table 1 shows the time required to compute the MAC of the application memory using either HMAC-SHA1 or Keccak-256. It is clear that total time consumed by running SPEED mainly depends on the time of computing the MAC. This metric is approach-independent which relies on the device capabilities (e.g. the speed of the clock), the type of MAC selected, and the performance of the MAC implementation used.

Table 1: Evaluation of MAC constructions

MAC	Memory footprint		Performance
	ROM (bytes)	RAM (bytes)	Time (sec)
HMAC-SHA1	1296	86	6.8
Keccak-256	1512	174	12.9

7.2 Memory overhead

- **Flash Memory** The TSM code has to be placed in the bootloader section which is part of the flash memory. SPEED requires no more than 4 KB of the flash memory. The exact code size of SPEED using HMAC-SHA1 is 2866 bytes, whereas it is 3102 bytes when using Keccak-256.
- **RAM** The overhead of using RAM is limited only to the use of the stack for holding temporary buffers and variables. In AVR, execution of instructions is only allowed from the Flash memory. Our evaluation shows that SPEED requires either 494 bytes or 582 bytes of RAM when using HMAC-SHA1 or Keccak-256 respectively. Table 1 presents the memory overhead of the MAC constructs without other helping functions in the TSM.

7.3 Power Consumption

The MicroPnP IoT platform, where we perform our experiment, consumes 3.54 mA when operating on 10 MHz in the active mode, and 54.5 μA in the idle mode. Every MicroPnP platform is powered by a standard 3000 mAh battery pack. The baseline battery lifetime, if the MCU is in the sleeping mode constantly, is 6.5 years. Considering these values, Figure 4 shows the estimated lifetime of the battery when running SPEED using either HMAC-SHA1 or Keccak-256 under different rates of time.

7.4 Accuracy of measuring the distance

The DB primitive in SPEED aims to establish the upper bound on the distance of the other party and does not target the exact location

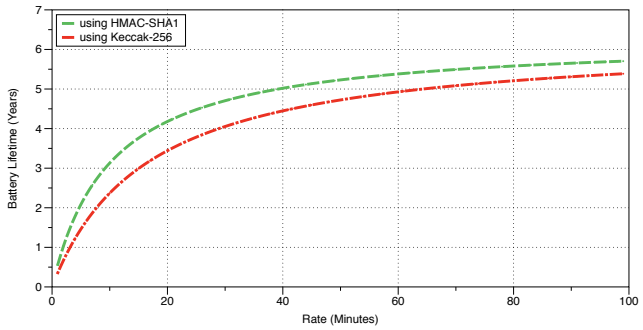

Figure 4: The impact on battery lifetime when using either HMAC-SHA1 or Keccak-256 primitives in SPEED.

of this party. We measure according to the speed of the light. So, a delay of 1 *ns* affects the distance of 15 cm. The microcontroller between hands (e.g. ATmega 1284p) operates on 10 MHz frequency and this means having a resolution of 100 *ns* in the ideal condition (e.g. The processing time is identified accurately on both sides, etc.), where a device with a distance of 1 *m* is detected as 15 *m* farther. Longer the range the more accurate is the measurement. For example, with this resolution, all devices within a range of 1 *m* to 14 *m* will be labeled with a distance of around 15 *m*. Since we are conducting our experiment within close range, this resolution does not help us. Therefore, we depend on the distance measurement functionality in the transceiver module itself. The IEEE 802.15.4 transceiver [5] between hands has a *time-of-flight* facility built into the hardware that improves the accuracy of measuring distance. Experimentally, we got an accuracy of 3 meters, which means a resolution of 20 *ns*. However, there are some distance measurement technologies [13] that integrate the aforementioned transceiver with a custom firmware to acquire special features like a RADAR system, thus giving a high accuracy where the resolution is near 1 *ns*.

Nevertheless, not all microcontrollers are integrated with such type of transceivers. In this case, the microcontroller should rely on the internal capabilities to calculate RTT accurately. Though recent work has yielded some proposals for establishing the upper bound on the distance between wireless sensor nodes with standard hardware [2], we still believe that this research problem is hardware-dependent and remains an open issue.

8 CONCLUSION & FUTURE WORK

Secure remote decommissioning (e.g. erasure) is as important as secure remote provisioning (e.g. deployment), and should be a key requirement for IoT devices. This paper proposed SPEED, an approach to secure provable erasure for embedded devices. It can be applied to all Class-1 IoT devices without any limitations. Our approach depends on isolating part of the flash memory using selective software virtualization and assembly level verification to store the trusted software module. We then build the secure erasure mechanism using DB protocol to prevent man-in-the-middle attack. The evaluation results show that SPEED incurs an acceptable overhead in terms of memory footprint, power consumption and performance. A fundamental limitation of SPEED is that it is limited to small (visual) distances.

In future work, we plan to investigate SPEED with a stronger attacker model where physical attack (e.g. the invasive one) should be taken into account by implementing some techniques in the TSM to detect it (e.g. detecting loss of power). Finally, a formal verification of the TSM code and a demonstration of a real and complete scenario including secure software deployment as well is another future goal.

REFERENCES

[1] Tigist Abera, N Asokan, Lucas Davi, Farinaz Koushanfar, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. 2016. Things, trouble, trust: on building trust in IoT systems. In *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 121.

[2] Stephan Adler, Stefan Pfeiffer, Heiko Will, Thomas Hillebrandt, and Jochen Schiller. 2012. Measuring the distance between wireless sensor nodes with standard hardware. In *Positioning Navigation and Communication (WPNC), 2012 9th Workshop on*. IEEE, 114–119.

[3] Muneeb Ahmad, Jalal S Alowibdi, and Muhammad U Ilyas. 2017. vIoT: A first step towards a shared, multi-tenant IoT Infrastructure architecture. In *Communications Workshops (ICC Workshops), 2017 IEEE International Conference on*. IEEE, 308–313.

[4] Atmel. 2009. AVR Atmega 1284p 8-bit microcontroller. http://www.atmel.com/images/doc8059.pdf. (2009). [Online; accessed 13-April-2017].

[5] Atmel. 2014. AT86RF233 . http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8351-MCU_Wireless-AT86RF233_Datasheet.pdf. (2014). [Online; accessed 13-July-2017].

[6] G Avoine, MA Bingol, Ioana Boureanu, S Capkun, G Hancke, S Kardas, CH Kim, C Lauradoux, B Martin, J Munilla, et al. 2017. Security of Distance-Bounding: A Survey. *Comput. Surveys* (2017).

[7] Carsten Bormann, Mehmet Ersue, and A Keranen. 2014. *Terminology for constrained-node networks*. Technical Report.

[8] Stefan Brands and David Chaum. 1993. Distance-bounding protocols. In *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 344–359.

[9] Yvo Desmedt. 1988. Major security problems with the àĂŸunforgeableâĂŹ(Feige)-Fiat-Shamir proofs of identity and how to overcome them. In *Proceedings of SECURICOM*, Vol. 88. 15–17.

[10] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. 2011. One-time computable self-erasing functions. In *Theory of Cryptography Conference*. Springer, 125–143.

[11] Kanika Grover, Alvin Lim, and Qing Yang. 2014. Jamming and anti–jamming techniques in wireless networks: a survey. *International Journal of Ad Hoc and Ubiquitous Computing* 17, 4 (2014), 197–215.

[12] Ghassan O Karame and Wenting Li. 2015. Secure erasure and code update in legacy sensors. In *International Conference on Trust and Trustworthy Computing*. Springer, 283–299.

[13] metirionic. 2015. ATSAMR21-XPRO. http://www.metirionic.com/en/technology.html. (2015). [Online; accessed 13-July-2017].

[14] Job Noorman. 2017. Sancus: A Low-Cost Security Architecture for Distributed IoT Applications on a Shared Infrastructure. (2017).

[15] Daniele Perito and Gene Tsudik. 2010. Secure code update for embedded devices via proofs of secure erasure. In *European Symposium on Research in Computer Security*. Springer, 643–662.

[16] Alejandro Proano and Loukas Lazos. 2012. Packet-hiding methods for preventing selective jamming attacks. *IEEE Transactions on dependable and secure computing* 9, 1 (2012), 101–114.

[17] Dave Singelee and Bart Preneel. 2005. Location verification using secure distance bounding protocols. In *Mobile Adhoc and Sensor Systems Conference, 2005. IEEE International Conference on*. IEEE, 7–pp.

[18] Rodrigo Vieira Steiner and Emil Lupu. 2016. Attestation in Wireless Sensor Networks: A Survey. *ACM Computing Surveys (CSUR)* 49, 3 (2016), 51.

[19] Robert Wahbe, Steven Lucco, Thomas E Anderson, and Susan L Graham. 1993. Efficient software-based fault isolation. *ACM SIGOPS Operating Systems Review* 27, 5 (dec 1993), 203–216. https://doi.org/10.1145/173668.168635

[20] Nils Walravens. 2016. Operationalising the Concept of the Smart City as a Local Innovation Platform: The City of Things Lab in Antwerp, Belgium. In *International Conference on Smart Cities*. Springer, 128–136.

[21] Thomas Watteyne, M Palattella, and L Grieco. 2015. *Using ieee 802.15. 4e time-slotted channel hopping (TSCH) in the Internet of Things (IoT): Problem statement*. Technical Report.

[22] Fan Yang, Nelson Matthys, Rafael Bachiller, Sam Michiels, Wouter Joosen, and Danny Hughes. 2015. μ PnP: plug and play peripherals for the internet of things. In *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 25.