Attestation in Wireless Sensor Networks: A Survey

RODRIGO VIEIRA STEINER and EMIL LUPU, Imperial College London

Attestation is a mechanism used by a trusted entity to validate the software integrity of an untrusted platform. Over the past few years, several attestation techniques have been proposed. While they all use variants of a challenge-response protocol, they make different assumptions about what an attacker can and cannot do. Thus, they propose intrinsically divergent validation approaches. We survey in this article the different approaches to attestation, focusing in particular on those aimed at Wireless Sensor Networks. We discuss the motivations, challenges, assumptions, and attacks of each approach. We then organise them into a taxonomy and discuss the state of the art, carefully analysing the advantages and disadvantages of each proposal. We also point towards the open research problems and give directions on how to address them.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; D.4.6 [**Operating Systems**]: Security and Protection—*Invasive software*

General Terms: Security

Additional Key Words and Phrases: Attestation, wireless sensor networks

ACM Reference Format:
Rodrigo Vieira Steiner and Emil Lupu. 2016. Attestation in wireless sensor networks: A survey. ACM Comput. Surv. 49, 3, Article 51 (September 2016), 31 pages.
DOI: http://dx.doi.org/10.1145/2988546

1. INTRODUCTION

Wireless Sensor Networks (WSNs) have a broad range of applications that can be employed to a wide variety of scenarios, from the most simple to extremely complex ones [Borges et al. 2014]. In critical applications such as medical emergency detection [Ko et al. 2010], volcano monitoring [Werner-Allen et al. 2006], and forest fire detection [Hartung et al. 2006], to cite a few, it is easy to argue that security is of utmost importance. A compromised node can disrupt the network operation or make it report false information with potentially disastrous consequences. Therefore, one must be able to decide whether or not to trust the data reported by such sensors. Nevertheless, even in less critical applications, such as sensor-based information appliances used in smart homes [Petriu et al. 2000], where the data monitored by sensors are not of much significance, the security, and, in particular, the integrity of sensor nodes is still important. An attacker can potentially use a compromised node not only to compromise other sensors but also as an entry point to undermine the system to which the sensor network reports its data.

Most of the related work on the security of WSNs focuses on securing its underlying protocols [Zhou et al. 2008] or on the trust and reputation management [Lopez

This work was supported by the Brazilian National Council for Scientific and Technological Development (CNPq) under Grant No. 206707/2014-2 awarded to Rodrigo Vieira Steiner.
Authors' addresses: R. V. Steiner and E. Lupu, Department of Computing, Imperial College London, South Kensington Campus, 180, Queen's Gate, London, SW7 2AZ, U.K; emails: {r.v.steiner, e.c.lupu}@imperial.ac.uk.

DOI: http://dx.doi.org/10.1145/2988546

et al. 2010] in the network. However, by exploiting existing software vulnerabilities, an adversary can easily compromise high-reputation nodes without breaking their protocols. As observed by Gu and Noorani [2008], vulnerabilities arising from low-level memory faults, such as the infamous stack overflow [One 1996], and more recent, sophisticated, exploit techniques, such as return-oriented programming (ROP) [Shacham 2007; Buchanan et al. 2008], pose a real threat to WSNs. In fact, these vulnerabilities have already been exploited in the form of code injection attacks [Francillon and Castelluccia 2008] and self-propagating mal-packets [Gu and Noorani 2008]. Thus, a mechanism to attest the integrity of sensor nodes is necessary. The integrity of a node is a binary property that indicates whether it has been modified in an unauthorized manner or not [Sailer et al. 2004]. Compromised nodes cannot be trusted as they may malfunction or present malicious behavior.

WSNs can comprise hundreds or thousands of sensor nodes that are distributed in the environment. In many cases, nodes are deployed in an unsystematic fashion, making it difficult to know their location. While it is not practical for most application scenarios, for some of them it is not even possible to physically reach and verify the integrity of each node composing the network. Thus, there is a need for a mechanism that not only verifies the integrity of the nodes but also that can do this verification remotely, without physical access to the device being verified.

Many challenges arise under these circumstances. How to scale the verification for the high number of devices. The heterogeneous hardware and software architecture of such devices. The limited amount of energy they have available. The intrinsically unreliable wireless medium in which they communicate. Nevertheless, several attestation mechanisms have been proposed over the past few years [Spinellis 2000; Seshadri et al. 2004; Kil et al. 2009]. However, existing approaches make different assumptions over the system and adversary models. Consequently, it is difficult to compare them and analyse the security properties provided by these schemes. Moreover, this lack of coherence has already resulted in new proposals [Jin et al. 2010; Vetter and Westhoff 2012; Kiyomoto and Miyake 2014; Tan et al. 2015; Yang et al. 2015; Asokan et al. 2015; Ibrahim et al. 2016] being vulnerable to formerly known attacks [Castelluccia et al. 2009].

The variety of assumptions made, the plethora of techniques proposed, and a number of controversies encountered in the literature are the motivation for this survey. In this article, we extensively analyse the state of the art of attestation mechanisms in the context of WSNs. We start in Section 2 by giving an overview of attestation, followed by a comprehensive analysis of the system and adversary models. We then cover common attacks and assumptions made and their relevance in the context of WSNs. In Section 3, we introduce a taxonomy that captures the fundamental differences between existing solutions. Each distinguishing characteristic used in our taxonomy is illustrated with a representative example taken from the state of the art, which allows us to evaluate the advantages and disadvantages of existing approaches. Following this analysis, we discuss open research problems and give directions on how to tackle them in Section 4. Finally, we conclude the article in Section 5.

2. ATTESTATION

A typical attestation mechanism follows a challenge-response protocol, as shown in Figure 1. A trusted device verifies the integrity of an untrusted device that has to prove its innocence. The devices are commonly named after their roles: *Verifier* and *Prover*, respectively. The goal of the attestation procedure is to allow an honest, non-compromised *Prover* to generate a response that assures the *Verifier* that the prover is in a legitimate state [Francillon et al. 2014]. A compromised *Prover* will either generate

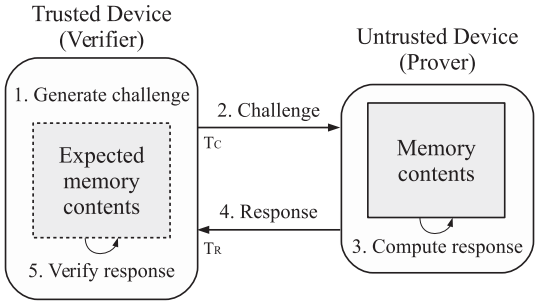

Fig. 1. Attestation overview.

invalid responses or will not be able to generate a valid response within an expected time limit.

It is generally assumed that the *Verifier* knows, in advance, the correct internal state—the memory contents—of the *Prover*. Therefore, the *Verifier* challenges the *Prover* to demonstrate that it is in a valid, expected, state. The *Prover* then executes an attestation routine, which will compute and send back a response based on the challenge received from the verifier and its internal state. The *Verifier* compares the answer received from the prover with the expected one, and if there is a match, then it can assert that the device has not been compromised. However, the *Verifier* only awaits the prover's response for a limited amount of time T_A, which must be at least as long as the time taken by the *Prover* to execute the genuine attestation routine. If the time difference between receiving the response and issuing the challenge, $T_R - T_C$, exceeds T_A, then the *Verifier* knows something may be wrong with the *Prover*. Note here that when performing attestation over a network, the time delay to send and receive messages must also be taken into account. As we will discuss further in Section 3, there are two different approaches regarding timing control: *strict* and *loose*.

We are not the first to analyse the attestation process. Nonetheless, existing analyses are built under different assumptions. For example, Datta et al. [2009], Coker et al. [2011], and Francillon et al. [2014] assume that the verifier and the prover need to share some secret information, for example, a cryptographic key. They presume that this secret is defined prior to network deployment and that there exists some hardware support preventing the adversary from accessing it. Their models, however, also do not consider timing requirements. On the other hand, Armknecht et al. [2013] and Li et al. [2014] focus on software-based attestation techniques that assume no hardware support and are only concerned with attesting the program's memory. The analysis in Li et al. [2014] also covers approaches that completely erase the prover's data memory. However, neither of them cover proposals that attempt to verify the data memory instead of simply wiping out its contents.

In this work, we follow in the footsteps of existing analyses; however, we aim to provide a more comprehensive model capable of encompassing all approaches relevant for WSNs, which allows us to evaluate the tradeoffs among different techniques. In this context, the intrinsic characteristics of WSNs play an important role. The simplified hardware and software architecture of sensor nodes has both a positive and negative side. As a consequence of their reduced storage space, there is less memory available for an adversary to explore. Furthermore, differently from general-purpose computers, each node has a well-defined application to execute. Thus, it is easier to know what the expected memory contents of these devices are. Moreover, sensor nodes are usually equipped with single-core processors and have a single flow of execution, which reduces the possibilities of an adversary to perform parallel operations during attestation. On

Table I. Notation Summary

Term	Description
A	Adversary
c	Challenge
M	Memory
M_e	External memory
M_p	Program memory
M_d	Data memory
M_{mmio}	Memory Mapped Input/Output
M_r	Registers
P	Prover
r	Response
S	Internal state
T_A	Attestation time limit
T_C	Challenge sending time
T_R	Response reception time
V	Verifier

the downside, the limited amount of energy powering sensor nodes is certainly one of the biggest restrictions imposed. The fact that the sensors may be placed in hostile environments and communicate over a wireless channel that may suffer from external interference is another complicator.

2.1. System Model

There are three entities that need to be considered when modeling an attestation mechanism: the verifier V, the prover P, and the adversary A. Although attestation mechanisms can be applied to other scenarios, we focus here only on WSNs. Therefore, both the verifier and the prover are wireless sensor nodes. The verifier can possess more computational power than common network nodes, but this is not mandatory. Meanwhile, the adversary is either launching attacks remotely or using an already-compromised network node. Table I summarizes the notation adopted in this article.

A prover P has an internal state $State(P) = S$ that reflects the contents of its memory M. Ideally, all memory contents should be attested including the program memory M_p, data memory M_d, registers M_r, MMIO M_{mmio}, and even external memories M_e. However, each of the different attestation mechanisms covers different sections of the memory, and, in practice, some parts of M are left unverified. For instance, Spinellis [2000] only checks M_p, while Zhang and Liu [2010] partially validates M_d and nothing else. For simplicity, we consider (as reflected in the adopted notation) that the state S of a prover P corresponds only to the portions of memory being attested.

One might think that the attestation is safer when a larger amount of memory is being covered. However, this is not necessarily so. Usually, M_p is far greater than M_d but some attacks that only need to alter M_d to succeed [Shacham 2007; Buchanan et al. 2008]. Therefore, a safer approach would be to attest all different types of memory [Castelluccia et al. 2009]. However, even when all memories are covered, an adversary can still perform several types of attacks. In practice, attestation only provides a probabilistic guarantee of the integrity of a prover. In this work, we aim to identify the factors that influence this probability.

An attestation process has three main constituents: CHALLENGE, ATTEST, and VERIFY, which are present in all the approaches described in the literature although their implementation may vary. Below we discuss each of them individually and highlight the design principles that must be followed to provide secure attestation. As stated before, we adopt much of previous analyses [Datta et al. 2009; Coker et al. 2011; Armknecht

et al. 2013; Li et al. 2014; Francillon et al. 2014] while centering the discussion around WSNs.

CHALLENGE() This procedure is executed by the verifier. It outputs a *random* challenge c that is transmitted to the prover and that may contain a nonce, a timestamp, memory addresses, or even the attestation routine to be executed by P. CHALLENGE must follow three design principles: *Authenticity*, *Freshness*, and *Unpredictability*. The first two principles are essential to prevent an adversary from performing Denial-of-Service (DoS) attacks by unrestrictively forcing P to perform ATTEST. Nevertheless, DoS attacks are not a major concern in attestation of sensor nodes as there are easier methods to achieve the same effect such as channel jamming. The last principle prevents the adversary from calculating the result of the attestation routine in advance since ATTEST either takes as input or is itself, the unpredictable challenge generated. Formally, this could be described as follows: there exists no efficient algorithm *Alg* such that, for non-negligible ϵ,

$$Pr[Alg()_i = Challenge()_{i+1} : i \in \mathbb{N}^+] > \epsilon.$$

ATTEST(S, c) The attestation routine is executed by the prover. It takes as input the state S of P and the challenge c sent by V. When c is the attestation routine itself, it is downloaded and installed in a pre-defined memory space in M_p. The goal of ATTEST is to compute a small attestation response r, which must directly be based on both S and c. This drastically reduces the amount of data transmitted from P to V, as otherwise the prover would have to transmit the entirety of its memory contents to the verifier to prove its authenticity. ATTEST must adhere to five design principles. *Authenticity* allows the verifier to confirm the source of r. *Atomicity* guarantees that ATTEST is not interrupted during execution preventing the adversary from modifying S, moving the malware around to avoid detection or parallelizing the computation. *Unforgeability* prevents an adversary from producing the same response r, at least not faster than ATTEST. *Dynamicity* in the sense that r should reflect the actual running system and not just some static part of the memory. Finally, *Determinism* enables the verifier to reach the same result r on its own. Formally:

$$Pr[\exists\ S \neq \widetilde{S} : Attest(S, c) = Attest(\widetilde{S}, c)] \leq \epsilon.$$

VERIFY(S, c, r, T_A, T_C, T_R) This function, executed by the verifier, takes as inputs the expected state S of the prover, the challenge c that the verifier has generated, the response r from P, the attestation time limit T_A, the time when the challenge has been sent T_C, and the response received T_R. VERIFY must respect one design principle: *Determinism* so it *accepts* iff r reflects both S and c and $T_R - T_C \leq T_A$. Furthermore, VERIFY must always *accept* responses from uncompromised provers. Formally:

$$\begin{aligned} Pr[Verify(S, c, r, T_A, T_C, T_R) = accept | State(P) = S \\ \wedge\ r = Attest(S, c) \\ \wedge\ T_R - T_C \leq T_A] = 1. \end{aligned}$$

Having identified the main components, we can now formalize the interactions between the verifier and the prover. Figure 2 shows all steps described so far, which are present in all existing approaches. In some approaches, however, these interactions may slightly differ. For example, they can be preceded by an additional step where the prover itself requests to be verified. Park and Shin [2005] demands that each sensor proves its integrity to a verification server before accessing the network, so new sensors must ask to be attested before using network resources. AbuHmed et al. [2009] provides another example of variation of the interactions: To verify the freshness and authenticity of a challenge request, a prover sends the challenge, encrypted together

$$
\begin{aligned}
& V : c \xleftarrow{R} Challenge() \\
V \rightarrow P : & \; c \\
& V : T_C \leftarrow current\ time \\
& P : r \leftarrow Attest(S, c) \\
P \rightarrow V : & \; r \\
& V : T_R \leftarrow current\ time \\
& V : Verify(S, c, r, T_A, T_C, T_R)
\end{aligned}
$$

Fig. 2. Generic attestation procedure.

with a random number, back to the verifier. The verifier then decrypts the message and sends back to the prover the encrypted random number.

2.2. Adversary Model

The objective of an adversary is to compromise a sensor node without being detected by the attestation procedure. We do not address techniques used to compromise nodes—the interested reader may refer to Francillon and Castelluccia [2008], Gu and Noorani [2008], and Giannetsos and Dimitriou [2013] for additional information on this topic—but rather cover the attacks an adversary may perform to overcome attestation. As observed by Armknecht et al. [2013], the adversary has two different phases to perform an attack: an initial phase that occurs before the attestation begins and a second one that starts when the prover is challenged. Before attestation, the adversary can use unlimited resources and may modify the state S of a prover at will, resulting in a new state $\widetilde{S}$. However, after being challenged the adversary has a limited amount of time to send a response $\widetilde{r}$.

2.2.1. Attacks. A series of attacks are typically portrayed in the literature on how an adversary may subdue the attestation process after having compromised a device. We describe each of them individually below.

Precomputation. An adversary can precompute all operations of the attestation routine that are not influenced by the challenge, thus gaining time that may be used to execute other operations during the attestation. Furthermore, if the adversary can predict the challenges generated by the verifier, then it can then precompute valid responses.

Replay. An adversary can eavesdrop a valid attestation response from a non-compromised node, store it, and retransmit the message when challenged by the verifier. Since the response reflects both the prover's internal state and the challenge, this approach can only work if both nodes execute the same program and receive the same challenge.

Forgery. The adversary can attempt to generate a valid response despite modifications in the prover's internal state. This can be achieved by executing a modified version of the attestation routine or by altering memory contents in such a way that modifications neutralize one another during response computation [Castelluccia et al. 2009].

Memory copy. If there is enough free space in memory, then an adversary can store its malicious data and still keep a copy of the original memory contents [Wurster et al.

2005]. Then, the adversary can modify the attestation routine so it computes a response over the memory locations where the original contents copy is maintained.

Data substitution. This attack is a special case of the memory copy attack and occurs when there is not enough space to keep a full copy of the original memory contents. The adversary can then modify part of the original contents and keep a copy only of the overwritten data [Seshadri et al. 2005]. In this case, the attestation routine must be adjusted to redirect memory reads from altered memory addresses to the original contents' copy location.

Compression. In another special case of the memory copy attack an adversary can compress the original contents to obtain enough space to store its malicious data [Castelluccia et al. 2009]. During attestation, the adversary can then decompress the original contents on the fly to compute the expected response.

Collusion. Compromised nodes can act together to compute valid responses. For example, multiple nodes that execute the same application can install the malware in different memory locations [Yang et al. 2007]. Then, during attestation, the nodes can exchange messages to recover the original memory contents. Another possibility is to divide the attestation routine operations across multiple devices to speed up the computation of the response.

Impersonation. An adversary can take multiple identities and impersonate other nodes (also known as a Sybil attack [Newsome et al. 2004]). In doing so, it can masquerade as a genuine node during attestation and send an invalid response, thus making the verifier believe the original node has been compromised. Besides, a compromised node may also impersonate the base station and forward the challenges it receives to a genuine node and then forward the correct response back to the base station.

Proxy. This attack is a special instance of collusion and impersonation attacks because it requires a device with better computing capabilities than the prover. Whenever a compromised node is challenged, it forwards the challenge to this proxy device. The proxy keeps a copy of the node's original memory contents and is able to impersonate it to compute a valid response. Since the proxy device has greater computing resources, it can also compute the attestation routine faster than common nodes.

Return-oriented programming. Return-oriented programming (ROP) attacks [Shacham 2007; Buchanan et al. 2008] use existing code, without altering it, to execute malicious operations. An attacker can perform arbitrarily complex malicious operations by linking together small sequences of instructions, called *gadgets*, present in existing programs. Originally, each gadget would end with a *ret* instruction, and the attacker could chain different gadgets together by modifying the stack, making them execute one after another. It was shown subsequently that these attacks can be performed without using *return* operations but also through other instructions that alter the program control flow, such as *branch*, *call*, and *jump* [Checkoway et al. 2010]. Because gadgets are built from original program instructions, ROP attacks can circumvent defenses that assume the adversary must modify or insert new code [Buchanan et al. 2008].

It is interesting to see that the majority of attacks do not require the malicious node to interact with other devices. Only the *Replay*, *Collusion*, *Impersonation*, and *Proxy* attacks require such interaction.

2.3. Assumptions

We present here the most common assumptions encountered in the literature and examine existing exceptions and conflicts among them.

The verifier cannot be compromised by the attacker. In many cases, the verifier is the network's base station. Since the base station acts as a gateway connecting the sensor network to the outside world, it is common to assume that it cannot be compromised [Seshadri et al. 2006]. Nonetheless, not all approaches make this assumption. For instance, Yang et al. [2007] proposes a distributed attestation scheme where all nodes in the network can play the role of the verifier. However, since all nodes are vulnerable to compromises, the attestation process no longer relies on a single verifier. Instead, multiple neighbors of a node collaborate to attest it. In this case, the attestation result also depends on how many devices an attacker has compromised.

The verifier knows the expected state of the prover. To attest that a device has not been compromised, a verifier must know what to expect from such device. In most approaches, the verifier has complete knowledge of the software that should be running on the prover device [Kennell and Jamieson 2003]. With this knowledge, the verifier can know the set of valid states a prover can be. A different strategy is to issue each node with a certificate of its valid configuration [Asokan et al. 2015]. This allows the verifier to attest a device without knowing its settings in detail.

The verifier knows the hardware architecture of the prover. The prover's hardware plays a major role in the construction of the attestation routine ATTEST. It defines which operations must be performed by the prover to demonstrate its integrity. For instance, ATTEST can take advantage of any tamper-resistant hardware available to perform its operations and protect secret information. On the other hand, when no such hardware is available, the attestation routine has to be carefully designed. In such scenarios, ATTEST is usually assumed to be optimal so an adversary cannot optimize it and execute further operations to hide its modifications and still reply in a valid time. Therefore, the verifier must know the microcontroller, clock speed, Instruction Set Architecture, and memory architecture of the prover [Seshadri et al. 2004].

The adversary can reverse engineer the prover's software and hardware. Commodity sensor nodes [Hill et al. 2004; Hempstead et al. 2008; Healy et al. 2008] do not provide any tamper-resistant hardware since such nodes are supposed to be cheap and small in size. So the node's software is usually stored in unprotected memory, which an attacker can read, reverse engineer, and modify. Therefore, the attestation procedure cannot rely on secret information such as cryptographic keys that would be stored in unprotected memory. Needless to say, this assumption does not hold for mechanisms developed targeting devices with tamper-resistant hardware.

The adversary has full control of the prover's memory. In the absence of hardware controls to protect the prover's program memory, an adversary can modify the underlying software at will. Consequently, the attacker can control the prover and all its communications and can perform both passive and active attacks, such as eavesdropping, packet injection, replay, selective forwarding, and many others. Some mechanisms explore the use of tamper-resistant hardware, Read Only Memory (ROM), or even a Memory Protection Unit (MPU) to limit the adversary control.

The adversary cannot modify the prover's hardware. It is generally assumed that an attacker cannot perform any hardware modification to the sensor node, such as attaching more memory, altering memory access timing, or even changing the processor clock speed [Seshadri et al. 2004]. This assumption is usually justified in terms of the cost and practicality of the attack. Not only would an attacker require physical access to the sensor to modify its hardware but also the means and time to carry out the modification on a significant subset of the sensors in the WSN. To remain consistent with this assumption, one must also consider that in such cases the attestation mechanism must similarly not rely on physical access to validate the integrity of the node and must

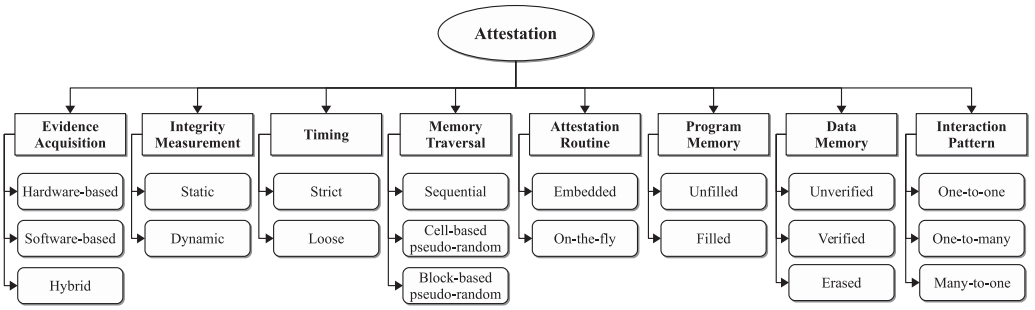

Fig. 3. A taxonomy of attestation mechanisms.

be done remotely. Note that attestation works as a first line of defense, compelling the adversary to either modify individual sensors or deploy new, already modified, sensors in the network. As observed by Park and Shin [2005], other techniques like intrusion detection systems [da Silva et al. 2005; Roman et al. 2006; Sun et al. 2007] can be deployed in conjunction with attestation to defend against such attacks.

3. TAXONOMY

Over the past few years, several attestation mechanisms have been proposed. Nevertheless, as seen in Section 2, they make different, and sometimes conflicting, assumptions. Therefore, they use intrinsically different ways of achieving their goals. The current literature most commonly separates attestation into two categories: *hardware based* and *software based*. However, we do not believe this classification is always helpful as it hinders the comparison of existing approaches in terms of the security properties achieved. There is more to the process of attestation than the use or not of tamper-resistant hardware. In this section, we review the main characteristics of existing approaches and propose a new taxonomy to classify the different techniques proposed. This allows us to compare their advantages, disadvantages, and vulnerabilities. Our proposed taxonomy, shown in Figure 3, identifies eight major characteristics of attestation mechanisms, each of which can be realized in different ways. We discuss them in more detail below.

3.1. Evidence Acquisition

Arguably, the biggest issue in attestation is the manner in which the verifier draws evidence of the prover's integrity and the extent to which this evidence can be trusted. Three main different approaches can be identified in the literature: *hardware-based*, *software-based*, and *hybrid* techniques.

Hardware-based techniques rely on tamper-resistant hardware such as the Trusted Platform Module (TPM) [TCG 2011] or Physical Unclonable Functions (PUFs) [Pappu et al. 2002; Gassend et al. 2002]. For example, Tan et al. [2015] describes a TPM-enabled Remote Attestation Protocol (TRAP) for WSNs in which all sensor nodes are equipped with TPMs responsible for securing preloaded secrets. Prior to the network deployment, each node is preloaded with cryptographic keys to safely communicate with neighboring nodes and the base station. When a node is powered on, it transfers control to its bootloader, which, differently from the application, cannot have its code updated after node deployment. Therefore, the bootloader code is used as a first line of defense. During the initialization phase, each node computes a hash of the bootloader code, stores it into a TPM Platform Configuration Register and uses it to seal the cryptographic keys into the TPM. When the bootloader is running, it computes and stores a hash of its own code. Consequently, if the bootloader code has been altered,

then the TPM unseal command will fail, and the node will be unable to generate a valid attestation response. The bootloader also computes and stores a hash of the application code into the TPM. During attestation, the verifier, which can be any neighbor of the prover, uses the TPM to generate a random number and encrypts it using the key shared with the prover. It then sends the challenge to the prover and asks the base station for the value of the prover's hash code and its public key. On receiving the challenge, the prover has to decrypt it, pass it as a nonce to the TPM, and construct a response based on the TPM output. This response can only be correct if the application code has not been altered. However, this approach only verifies the prover's M_p. Therefore, it is vulnerable to ROP attacks, which only need to alter the call stack to succeed.

Software-based techniques do not rely on secure hardware. Instead, the prover executes an attestation routine that produces an allegedly unforgeable result. Spinellis [2000] is one of the earliest works on *software-based* attestation. The author proposes the use of reflection to perform software integrity verification, and although the proposed approach is not specifically designed for WSNs, its computational performance remains within the practical limits of WSN nodes. In this approach, the verifier randomly chooses two overlapping memory regions of the prover's M_p such that one region covers the initial memory addresses and the other covers the last addresses, and they overlap somewhere in-between. The prover then computes a cryptographic hash for each region. The verifier similarly calculates the corresponding hash values, from its own copy of the memory being attested, and compares them with the values received. Because the hash values cover the entire M_p any modification to it will be detected. This, however, is based on the assumption that the attacker cannot interrupt the verification process and move the malware around, always relocating it to somewhere out of the current hash computation range. Moreover, as the hash computations are independent of one another, colliding nodes could compute them separately and in parallel to avoid detection by timing differences. Spinellis also describes an extension of this procedure where the prover also sends data regarding its processor state, such as the CPU cache or performance counter. However, as this information is not used in the hash computation, once an adversary eavesdrops a valid response, it could simply extract this part and replay it.

We classify as *hybrid* techniques approaches that do not depend on a tamper-resistant hardware but do require specific hardware, such as ROM, to achieve attestation. For example, Perito and Tsudik [2010] presents a secure code update mechanism for embedded devices based on *Proofs of Secure Erasure (PoSE)*. This procedure requires a small amount of ROM to store the attestation routine and thus prevents an adversary from modifying it. The verifier sends incompressible random noise large enough to completely fill the prover's writable memories. The prover uses the code stored in ROM to compute a Message Authentication Code (MAC) over all data received, using the last bits as the key, and sends it back to the verifier, which checks it. This procedure is also particularly targeted at secure code updates as the "incompressible random noise" can be an encrypted form of the new code for the node. Once the verifier has verified the MAC, it can then send to the prover the key used for encryption, which the prover uses to decrypt the code and perform the code update. The main disadvantage of this approach is the high communication overhead it introduces. The need to transmit enough data to completely fill the prover's writable memories consumes significant amounts of time and energy. Furthermore, there is an implicit assumption that nodes under attestation cannot collude.

3.1.1. Discussion. Tamper-resistant hardware works as a root of trust, and all the information and services it provides are considered to be reliable, thus facilitating the attestation procedure. However, a major drawback of *hardware-based* techniques is

that they cannot be used on legacy devices that do not have such hardware. Furthermore, using tamper-resistant hardware increases both sensor cost and energy consumption, making it inappropriate in a number of scenarios. The assumption that tamper-resistant hardware is sufficient to engender trust is also increasingly threatened as numerous types of attacks can still be explored by an adversary [Anderson and Kuhn 1996]. Side-channel attacks such as timing [Kocher 1996], power [Kocher et al. 1999], and electromagnetic analysis [Gandolfi et al. 2001] are some examples. More recently, both TPMs [Kursawe et al. 2005; Kauer 2007; Sparks 2007; Parno 2008; Winter and Dietrich 2013] and PUFs [Rührmair et al. 2010; Merli et al. 2011; Helfmeier et al. 2013] have been the target of attacks. In contrast, *software-based* techniques do not rely on tamper-resistant hardware. The benefits of these approaches are that they can be applied to legacy devices and do not increase the node's cost and size. Therefore, it is unsurprising that the majority of attestation mechanisms proposed for WSNs are *software based*. *Hybrid* approaches share advantages and disadvantages of *hardware-* and *software-based* techniques. For example, writing the attestation routine in ROM guarantees that it will not be modified by an attacker. However, it is dependent on having a sufficient amount of space available in the ROM, and this may not be the case with legacy devices. Furthermore, since physical attacks are difficult and costly, the use of tamper-resistant hardware might be considered overkill. For instance, an MPU can be used instead to prevent illegitimate accesses to secrets [Eldefrawy et al. 2012; Koeberl et al. 2014].

There is much discussion on the feasibility of remote attestation using solely *software-based* techniques. Before these schemes started being applied to WSNs—which theoretically facilitates their implementation, as sensor nodes have a much simpler architecture—Kennell and Jamieson [2003] proposed to use CPU execution side effects, such as translation lookaside buffer misses, into a genuine test. The viability and reliability of using such side-effects were then significantly debated [Shankar et al. 2004a; Kennell and Jamieson 2004; Shankar et al. 2004b]. Castelluccia et al. [2009] investigates the shortcomings of existing approaches to embedded devices and presents two generic attacks, which have since been refuted [Perrig and Van Doorn 2010] and then reasserted [Francillon et al. 2010]. They argue that it is very difficult to correctly design attestation schemes with *strict timing* conditions because their implementation must be highly optimized. They also claim that, contrary to some existing schemes, all memories of the prover must be attested and that doing so is quite challenging. Francillon et al. [2014] asserts that *software-based* techniques can be secure only if the prover and verifier communicate directly with no intermediate nodes and thus cannot be used to perform attestation *across a network*. However, Yang et al. [2015] presents a delay-resilient software-based attestation mechanism capable of performing multi-hop attestation named Low-cost Remote Memory Attestation (LRMA). LRMA demands the response packet to go through the same path taken by the challenge. Relay nodes record the time when they receive each packet and report it to the verifier that can then estimate the average single-hop delay and detect compromised nodes by using a Bayesian classifier. Moreover, if the network is using a Time Division Multiple Access– (TDMA) based MAC [van Hoesel et al. 2004], then the network delay is known. Furthermore, it is also possible to attest all network nodes without using multi-hop attestation. For example, Seshadri et al. [2006] proposes an expanding ring method, on which the base station starts by attesting nodes one hop away from it and then asks these nodes to attest their neighbours. The verification then proceeds in a hop-by-hop manner resembling an expanding ring. Another alternative is presented in Yang et al. [2007] where the authors propose a *many-to-one attestation* in which the prover's neighbors execute the verification procedure avoiding the need for multi-hop attestation.

3.2. Integrity Measurement

The internal state of a prover can comprehend its program memory, data memory, registers, MMIO, and even external memories. Memories can be further divided into a *static* part whose contents never change during normal software execution and a *dynamic* part whose contents may be inserted, removed or modified.

Static integrity measurement approaches verify only the *static* part of a prover's memory. For example, in the lightweight attestation scheme for WSNs presented in Kiyomoto and Miyake [2014], all nodes have their memory divided into two parts: a program area M_P for storing program code and data and another area M_A for attestation. Both M_P and M_A are divided in what the authors named *registers* (but without discussing the size of a register—it thus could be a single memory address or a block of addresses). If the program code is not large enough to fill M_P, then further random data is used in order to fill it. During the initialization phase, a sensor node randomly selects a register from one of its neighbors, computes a hash value for it, and stores the result in M_A. The node then repeats this process, randomly selecting different registers from different neighbors until it fills M_A. During attestation, a node randomly chooses a register either from M_P or M_A. If it selects a register from M_P, then the node gets the corresponding M_A register stored in one of its neighbors. Otherwise, if it selects a register from M_A, then it gets the corresponding M_P register stored in one of its neighbors. In both cases, the hash is recalculated from the M_P register, and the result is compared with the M_A register. If the values do not match, then one or both nodes have been compromised. However, there is no way to find out which node has been compromised, and, therefore, both nodes must terminate their operation. A terminated node stops communicating with the network to avoid the propagation of malicious code. It is, however, possible for an adversary that successfully compromises a node to modify the code in such a way that it never terminates, even if it does not pass attestation. Furthermore, the attestation verifies only one register at a time, so it has poor memory coverage and, for example, an adversary that modifies only one register has a good chance of remaining undetected.

Dynamic integrity measurement approaches check runtime properties of the software executing on the prover where such properties represent the behavior that must be satisfied during the normal execution of a program. For instance, the stack frames are arranged as a linked list, where the base pointer of a frame points to the base pointer of the previous frame. Remote Dynamic Attestation System (ReDAS) [Kil et al. 2009] is an example of such an approach. It automatically extracts the properties from each application's source code and binary in an offline phase. Then, during the program execution, any integrity violation evidence is recorded. To prevent an adversary from modifying the recorded evidence, every prover is equipped with a TPM. Therefore, when a violation is detected, it is immediately sealed into the TPM. Then, when challenged by a verifier, all the prover has to do is send the sealed information. However, ReDAS only checks a subset of all possible dynamic system properties and measures the system integrity only at system calls. Therefore, an adversary can still succeed if it only changes properties not covered by ReDAS or if it hides modifications between system calls.

3.2.1. Discussion. To check if a device has been compromised, the verifier must know in advance the set of valid states for the device. Since contents in *static* memory regions do not change during normal software execution, they provide a straightforward way to attest a device. The verifier challenges the prover to calculate a checksum over these memory regions and come up with a valid response, which turns out to be difficult to achieve unless these memory regions have not been modified. It is difficult but not impossible. As we have seen, there are numerous methods an adversary may use

to circumvent attestation, for example, forgery, memory copy, and collusion attacks. If the attestation process is not carefully implemented under realistic assumptions, then the adversary may succeed. Therefore, even if a device comes up with a valid response, it does not mean that it has not been compromised. *Static* integrity measurement approaches are eminently vulnerable to ROP attacks since these attacks use the already-existing code without altering it. Just as important as verifying that the code residing in static memory regions has not been modified is to verify that the code is being executed as it was mentioned to be. This is the aim of *dynamic* integrity measurement approaches. However, due to the diversity and dynamicity of runtime properties, it is not an easy task to identify the known good states of dynamic objects [Kil et al. 2009].

3.3. Timing

In any practical implementation of an attestation mechanism, the verifier will only wait for a limited amount of time for a prover's response after sending a challenge. While some proposals have a *strict* timing condition, others adopt a more *loose* approach.

SoftWare-based ATTestation (SWATT) [Seshadri et al. 2004] is the first attestation mechanism designed specifically for embedded devices. It relies on *strict* timing of challenge/responses to detect compromised provers. The verifier sends the prover a randomly generated nonce that is used as the seed to the prover's Pseudo-Random Number Generator (PRNG). The prover then performs a cell-based pseudo-random memory traversal, iteratively reading memory words and computing a checksum of its program memory contents. Therefore, an adversary cannot predict the order of memory accesses, and if the memory has been altered, then the attacker has to modify the attestation routine and insert statements checking whether the current address was modified. If that is the case, then, to get the right response, the adversary has to redirect the memory access to the memory location where the original value is. The authors' main assumption is that the attestation routine is constructed in a time-optimal way so any modifications to it would result in a detectable increase in computation time that the verifier would detect. So, the verifier detects compromised provers either because the returned checksum is wrong or because the response is delayed. Castelluccia et al. [2009], however, presents an attack that is faster than the one the authors of SWATT considered. Furthermore, another possibility is to overclock the prover's CPU such that even if more instructions have to be executed, the total amount of time taken would still be within the limits. Although, this would be considered a hardware modification attack and, thus, assumed not to occur. Partly to address this, Kong et al. [2014] proposed to incorporate the outputs of a PUF into the checksum computation to overcome overclocking, as well as impersonation, attacks.

Choi et al. [2007] proposes a proactive code verification protocol for WSNs with *loose* timing conditions. In essence, the main idea is to fill the prover's memory so an adversary has no place to hide its malicious code. The base station adopts the role of the verifier and is assumed to share a pairwise key with every node in the network. The prover receives a nonce from the base station and uses it as the seed to a Pseudo-random Function (PRF) whose outputs are used to fill empty memory regions. Afterwards, the prover calculates a hash over its memory and sends the result to the base station for verification. The issue with this approach is that the random contents used to fill the memory are being generated by a PRF executing on the prover. So once a node is compromised, the attacker has access to the PRF and can use it to calculate the hash on the fly without ever storing its outputs in memory. Even if this takes more time than the normal protocol execution, the proposed scheme does not strictly control the execution time of the attestation routine, and the attack would pass undetected. A second issue is, again, the use of cryptographic keys with no tamper-resistant hardware.

3.3.1. Discussion. Theoretically, the larger the time limit for a prover to respond to a challenge, the higher the number of attacks an adversary can explore. Whether an approach relies on accurate measurements of the attestation routine execution time depends on the system model and the assumptions made.

Approaches with *strict* timing conditions do not rely on tamper-resistant hardware and assume a time-optimal implementation of the attestation routine. Otherwise, an adversary could develop a faster routine and use the time saved to forge a valid result. However, the work in Castelluccia et al. [2009] highlights a number of limitations of these approaches. First, it is very difficult to correctly design a time-optimal attestation routine since its implementation must be small and simple. This precludes the use of cryptographic functions, which are complex and time-consuming, and favors the use of simple instructions, such as *add* and *xor*, which may achieve poor security. Second, to achieve maximum speed, these routines are implemented in assembly, which is highly error prone. Third, proving the runtime optimality for both the attestation routine and best possible attack remains an open research problem. Another issue with these approaches is that, in most scenarios, they cannot be used for multi-hop attestation, since the network latency may be unpredictable. On the other hand, approaches that do not depend on accurate measurements of the execution time either rely on tamper-resistant hardware or make assumptions that limit the adversary capabilities. One of the assumptions made is that if all memories of the prover are filled, then an adversary has no space to allocate malware and still manage to compute a valid response. However, these assumptions do not always hold. For example, an adversary might compress the existing code in memory, gaining enough space for the malware, or colluding nodes might install the malware in different memory locations such that when they communicate they can recover the contents that were locally overwritten.

3.4. Memory Traversal

During attestation, a prover uses its memory contents as evidence of its trustworthiness. The prover has essentially two different ways to traverse its memory: *sequentially* or *pseudo-randomly*, where the latter can be further classified into *cell based* or *block based*.

Sequential memory traversal approaches go through a prover's memory in an iterative manner from start to end. For example, Park and Shin [2005] present a soft tamper-proofing scheme for WSNs based on *Program Integrity Verification (PIV)*. The network is divided into clusters, operated by cluster heads, named PIV Servers (PIVSs), which have better computation and storage capabilities than normal sensors. PIVSs work as verifiers, and for each attestation round they generate a different attestation routine, PIV Code (PIVC), which is executed by the prover. By using Randomized Hash Functions (RHFs), PIVSs randomly encode hash computation algorithms for each PIVC created. To protect sensors from adversaries impersonating PIVSs, there are multiple Authentication Servers (AS) deployed over the network. An AS works as a Trusted Third Party (TTP), allowing sensor nodes to confirm the authenticity of PIVSs and, consequently, the PIVC. The authors do not discuss the AS in further details stating they can either share a symmetric key with nodes or use public-key cryptography; though, again, the nodes are not assumed and are not likely to possess tamper-resistant hardware to secure cryptographic keys. Prior to deployment, a nodes' program memory is partitioned into multiple blocks. For each block, a digest is calculated and stored in a database accessible by all PIVSs. The digests are then classified into three categories: common to all nodes, common to a group of nodes, or unique to a specific node. Therefore, the number of digests to be stored can be greatly reduced. Before gaining access to the network, a node must prove its integrity. Thus, during the initialization phase, all nodes ask to be verified. The RHF generates the same result if it takes

either the program block or its corresponding digest as input. Consequently, both the prover and the PIVS are able to obtain the same result. Starting from the first address of the program memory towards the last, a checksum is computed for each memory block, and a final checksum over these checksums represents the entire memory. The authors propose to initialize the data memory with random values that can neither be predicted nor compressed. However, an adversary could still compress the original software, residing at M_p and calculate the digest for the compressed code blocks using on-the-fly decompression techniques.

Secure Code Update By Attestation (SCUBA) [Seshadri et al. 2006] is a mechanism for the detection and recovery of compromised nodes in WSNs. It relies on Indisputable Code Execution (ICE), which guarantees untampered code execution regardless of whether the node has been compromised or not. The approach requires each sensor node to have enough space in ROM to store its own ID and the base station's public key. Therefore, an adversary cannot modify these values, even after it compromises a node. The approach is similar to SWATT; however, it does not verify the entire program memory, and it extends the checksum computation to include dynamic data. Differently from Spinellis, ICE takes the CPU state—Program Counter (PC), Data Pointer, and Status Register—as input to calculate the checksum. To secure its untampered execution, the attestation routine disables all interrupts. Then it computes a checksum over the memory regions containing itself, the SCUBA protocol executable, the node ID, and base station's public key, as well as the CPU state. It sends the result to the base station and starts the SCUBA executable, with interrupts still disabled, guaranteeing it executes untampered. As with SWATT, the attestation routine is constructed in such a way that the checksum will either be incorrect or its computation will be notably longer if the routine is modified. In such cases, the base station can assume the node has been compromised and blacklist it. Otherwise, the SCUBA protocol can further verify the node, inspecting and repairing the rest of its memory. A downside of the proposed scheme is that the PC may not be accessible depending on the platform. Furthermore, Castelluccia et al. [2009] describes an attack that overcomes ICE's checksum computation and is able to execute arbitrary code without being detected. The attack takes advantage of the fact that the additions performed by the ICE checksum function discard the carry. Therefore, changes in the most significant bit (MSB) may not alter the checksum result. This allows an adversary to store a copy of the ICE function in a different memory position, such that its address only differs from the original location in its MSBs.

3.4.1. Discussion. Sequential memory traversal approaches are simple to implement and efficient—they run in linear time according to the memory size. Furthermore, they provide complete coverage over the memory regions being verified, passing through each memory address a single time. Nevertheless, these advantages come at the cost of a disadvantage, which is predictability. The fact that memory addresses are checked only a single time and in a predictable order makes it easier for adversaries to counterfeit attestation. For instance, an adversary can move memory contents around to avoid detection. When verification starts, malware can be moved to the end of the memory, and right after the verification passes through its original position, the malware can be moved back. Another possibility is for two colluding nodes to install the malware in different memory locations, and when one of them is under attestation, it asks the other for the contents it has overwritten. In contrast, pseudo-random approaches are intrinsically unpredictable and not victims of the same attacks. However, they are less efficient and provide only a probabilistic guarantee of memory coverage. To ensure, with high probability, that each memory address is accessed at least once, these schemes rely on the result of the Coupon Collector's Problem [Mitzenmacher and Upfal 2005],

which states that for a memory of size n it is necessary to perform $O(n \ln n)$ memory reads. Consequently, some memory addresses end up being accessed multiple times, introducing unnecessary overhead. To reduce this overhead, Yang et al. [2007] proposed to traverse the memory in a block by block manner. Rather than accessing the memory one address at a time, block-based approaches access blocks of addresses and perform *xor* operations within blocks. For a block size b, $O(\frac{n \ln n}{b})$ iterations are necessary to cover each memory address at least once. It is interesting to see that when $b = 1$ the scheme functions as cell-based approaches, and when $b = n$ it works as a sequential traversal. Choosing the size of b is, therefore, a tradeoff between performance and security. Furthermore, as observed by Armknecht et al. [2013], if the block size is determined prior to attestation, then an adversary can perform collision attacks against block-based schemes. These are forgery attacks that work by altering addresses inside a block in such a way that modifications neutralize one another.

3.5. Attestation Routine

Most existing approaches have their attestation routine *embedded* in the prover's memory prior to the network's deployment. However, a verifier may also generate and send the prover different routines for each attestation round.

For example, Shaneck et al. [2005] proposes a remote software-based attestation framework where the attestation routine is generated *on the fly*. The base station plays the role of the verifier and is assumed to be within communication range of all network nodes. Furthermore, it shares a symmetric key with each node to secure communication. At each attestation round, the base station generates a new attestation routine and sends it to the prover. It waits for a maximum time that comprises the time to send and receive a message, the attestation routine execution time, and the expected network delay. The procedure utilizes techniques such as randomization, encryption, obfuscation, and self-modifying code to prevent an adversary from avoiding detection. They propose to use random keys to encrypt the entire attestation routine and send a corresponding decryption routine together with the code. This routine is also responsible for discovering the key, which is located somewhere in the prover's memory, hidden through opaque predicates [Collberg et al. 1998]. The attestation routine has three main components: seed calculation, memory reads, and hash computation. The first component is responsible for initializing the PRNG, which determines the order of memory accesses. So it is in the interest of an adversary not to modify this part but to infer the seed value. However, the seed computation also uses opaque predicates. The other two components are part of a loop that reads memory addresses and use their contents to compute the hash. As with SWATT, it is necessary to iterate through the loop several times to cover the program memory. The second component is the one an adversary would attempt to modify so it could redirect memory reads. To avoid such modifications, this component has several junk instructions, which appear to be reachable due to opaque predicates, and it self-modifies its code relocating the read instruction at each iteration. After the hash calculation is complete, the result is sent to the base station. The authors have neither implemented nor evaluated their proposal, so it is difficult to discuss its security and even its feasibility. One difficulty to implement this proposal is that several commodity sensor nodes store the executable code in flash memories programmable only by pages. Another issue is the use of cryptographic keys to secure communication, which, once again, is made without considering that commodity nodes do not have tamper-resistant hardware to protect them from attackers.

3.5.1. Discussion. Approaches that embed the attestation routine in the prover's memory provide security by design. The attestation routine is conceived to be secure, such

that a prover will fail, with high probability, to provide a valid response if either it changes the routine or the memory contents under verification. On the other hand, approaches that generate the attestation routine on the fly provide security through obscurity. Since a new routine is generated for each attestation round, attackers cannot predict what instructions should be executed and even less the outcome. The attestation routine may actually have vulnerabilities, but since their implementation is hidden from opponents, these vulnerabilities are unlikely to be explored in a timely manner.

3.6. Program Memory

It is possible that the program of a sensor node does not occupy its entire program memory, thus leaving empty spaces. An adversary can, therefore, take advantage of this space to store data used to overcome attestation. To avoid this, some approaches propose to *fill* the empty space with incompressible random noise.

For example, AbuHmed et al. [2009] introduces two software-based remote code attestation procedures for WSNs. The authors also consider a scenario where the base station acts as verifier and shares cryptographic keys with sensor nodes without any tamper-resistant hardware. In this context they present two techniques, one pre- and one post-deployment, to fill the empty memory space of sensor nodes with incompressible random noise. In the pre-deployment approach, the program memory of each node is filled using a seed and the verifier keeps a register of the seed together with the corresponding node ID. Then, during attestation time, the verifier can generate a copy of the node's memory to compute the checksum and compare the result with the one received. In the post-deployment approach, a node uses some dynamic data gathered from the environment as the seed to generate the noise and fill its memory. After that, the sensor transmits the seed to the verifier and deletes it from its memory. Consequently, an adversary who compromises the node afterwards has no space to store its malicious code, and if it does overwrite the memory, it cannot regenerate the original contents without the seed. The authors also propose two block-based memory traversal algorithms with variable block sizes, in contrast to Yang et al. [2007] where the size is always the same. In the first algorithm, the verifier defines the size of the block together with the challenge. In the second algorithm, a dynamic block size that changes during the checksum computation is used. However, both schemes are still vulnerable to compression attacks, as an adversary can compress the original software and calculate the digest for the compressed code blocks using on-the-fly decompression techniques.

3.6.1. Discussion. Physical memory contents are typically of low entropy and thus compressible [Douglis 1993]. As a result, even if empty spaces are filled with incompressible random noise, it may still be possible for an attacker to compress the original program code and gain enough space for its malicious code [Castelluccia et al. 2009]. To defend against this attack, Vetter and Westhoff [2012] presents a code attestation mechanism with compressed instruction code. Each sensor node is uploaded with a compressed code image, and its remaining program memory is filled with incompressible random noise. The code image is divided into blocks of equal length that are individually compressed. However, after compression these blocks may not have the same size. Therefore, the program memory is divided into two sections, one to store the compressed code blocks and another one, named Line Address Table (LAT), to store the block's offsets. In order to execute the compressed code, they incorporate a hardware extension: A dedicated microcontroller uses the LAT section to decompress code blocks, and a cache, maintained within the node's RAM, is used to store the decompressed code block under execution. They implemented the same attestation routine used by SWATT [Seshadri et al. 2004]; however, they do not have the same strict timing conditions since the

memory is completely full. Nonetheless, the data memory is not verified, and an attacker can perform ROP attacks or even modify the decompressed code held in the cache to compromise a node. Besides, the need for a dedicated microcontroller to decompress code blocks is a strong limiting factor for the applicability of this scheme.

3.7. Data Memory

Depending on the prover's memory architecture, the data and program memory physical addresses can either be shared with (von Neumann architecture) or separated from (Harvard architecture) one another. In the latter case, it is common for the size of the program memory to be much bigger than the data memory. Therefore, an attacker exploring the data memory has only a limited amount of space [Shaneck et al. 2005]. Also, in the Harvard architecture, the contents of the data memory are not executable. For these reasons, some approaches do not verify the data memory. However, attacks exploring the data memory have already been demonstrated. For example, the work in Castelluccia et al. [2009] describes a rootkit-based attack that hides malicious code in the data memory during attestation.

Some *dynamic* attestation mechanisms try to *verify* the data memory. For example, *Dataguard* [Zhang and Liu 2010] is a software attestation scheme for dynamic data integrity based on *data boundary integrity*. Each node has a unique seed, which is erased after initialization, that is used to insert data guards, similarly to canary words [Cowan et al. 1998], around data objects. When challenged by a verifier, the prover sends a response based on the values of all data guards. Therefore, if an adversary overwrites a data guard, by performing a buffer overflow attack, for example, it will not be able to recover the original data guard value since it no longer has the seed. However, this scheme also has its own limitations. First, the method is vulnerable to attacks that do not modify data guards. Second, it only provides a coarse-grained data protection where each memory block is treated as a unique object. Consequently, the scheme does not protect individual array elements.

Other approaches, such as those in Park and Shin [2005] and Perito and Tsudik [2010], overwrite all contents of the data memory, erasing any malicious data in the process.

3.7.1. Discussion. Approaches that do not verify the data memory are inherently vulnerable to ROP attacks. However, existing approaches that verify the data memory are not completely safe either. Due to the difficulty to predict the behaviour of dynamic data, existing approaches only partially cover the data memory. While erasing all the data memory is the safer approach, it is not really attesting the memory contents. Furthermore, together with any malicious data, these approaches also eliminate all data a node has worked to achieve prior to attestation.

3.8. Interaction Pattern

Most existing approaches interact in a *one-to-one* pattern, as depicted in Figure 1, where for each attestation round there is one verifier and one prover. However, this is not the only possible way to perform attestation.

For example, Jin et al. [2010] proposes an *Unpredictable Software-based Attestation Solution (USAS)* to detect compromised nodes in mobile WSNs that uses a *one-to-many* interaction pattern by creating dynamic attestation chains. Each chain comprises a single Initiator (I-node) and several Follower nodes (F-nodes). In each attestation round the base station, acting as the verifier, challenges a randomly selected I-node. The challenge consists of a random number, which is used as input for the attestation routine, and authentication messages for the I-node and the F-nodes. When challenged, the I-node runs the attestation routine and uses its output to challenge the F-nodes,

which then execute the attestation routine and send the result back to the base station. Prior to the network deployment, each node has its program memory filled with pseudo-random noise and the base station keeps a copy of the seeds used for all nodes. The base station then compares the results from the F-nodes with the expected ones to detect compromised nodes. As long as one F-node result is valid the I-node can be considered genuine as well. However, if all F-nodes return invalid results, nothing can be said about all attested nodes. In this case, either all F-nodes have been compromised or the I-node has been. Performing all attestation chain, which is not a cheap operation, and not being able to affirm anything is a significant limitation of this scheme. Another limitation of this proposal is that, even though it creates an attestation chain, it attests only sensors that can directly communicate with the base station to avoid intermediate nodes tampering with the messages.

Another possibility is to orchestrate a distributed attestation, in a *many-to-one* interaction pattern, where the neighbors of a node work together to attest it. For example, Yang et al. [2007] presents two distributed software-based attestation schemes where sensor nodes collaborate with each other to attest the integrity of their neighbors. The main difference from previous approaches is that only regular nodes are involved, and this approach does not require trusted verifiers. The authors propose to fill the empty spaces in each sensor's program memory with pseudo-random numbers, but, differently from Choi et al. [2007], this is done prior to node deployment. For each node, a different seed is used for generating the pseudo-random numbers. After deployment, every sensor discovers and establishes a pairwise key with each of its neighbors—again, cryptographic keys are used with no protection. In the first proposed scheme, a node splits its seed into multiple shares, sends a separate share to each neighbor, and then deletes the seed from memory. An attestation is triggered when more than half of a node's neighbors detect its abnormal behavior. In this case, all neighbors elect a cluster head, which differ for each attestation round. The cluster head challenges the node, with a random number, and collects the seed shares from other neighbors to recover the seed. Then it locally computes the expected result and compares it with the response from the challenged node. The authors propose a new method to randomly traverse the program memory: Instead of reading one memory word at a time, as was done in SWATT, they read a block of memory addresses and perform *xor* operations within blocks at each iteration. By using an appropriate block size, they are able to reduce the total amount of iterations while still covering the whole memory. In the second scheme, every node is also preloaded with tuples of challenge and response to its own M_p. Instead of sending shares of the seed, a node sends a tuple to each of its neighbors and then deletes all the tuples. When attestation is triggered, each neighbor attests the node sequentially using the challenge and response from the received tuple. A node is considered to be compromised if it does not pass the majority of attestations. A limitation of these approaches is that they require a minimum network density to work. If a node does not have enough neighbors, then it cannot be properly attested. Furthermore, both schemes incur considerable communication overhead. The first is more vulnerable to compromised nodes as they can send forged shares of the seed or, even worse, be elected cluster head. In the second scheme, an adversary would have to compromise more than half of neighboring nodes to succeed. However, this latter scheme requires the prover to execute the attestation routine once for each of its neighbors, which is both time- and energy-consuming.

3.8.1. Discussion. Simplicity is the main advantage of the one-to-one interaction pattern. It allows the verifier to target each node of the network individually. However, if several nodes need to be attested, the overall time increases linearly with the number of nodes. Attesting nodes in a one-to-many pattern reduces the overall computation time

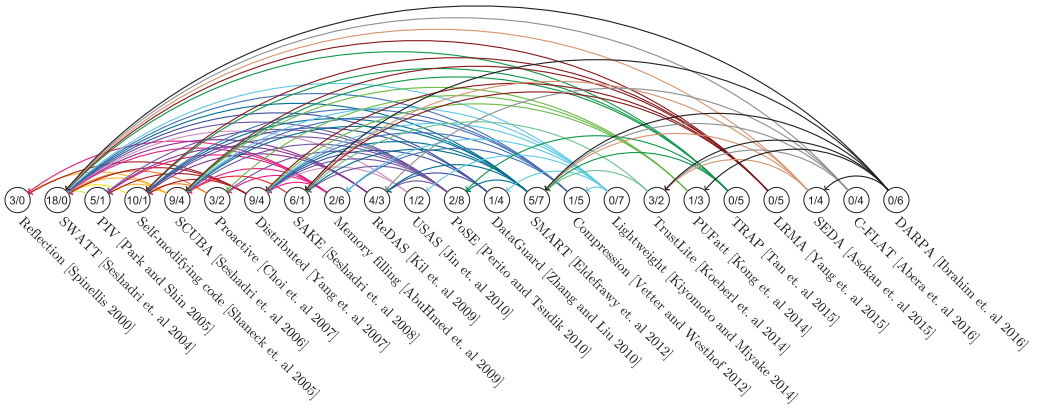

Fig. 4. Attestation mechanisms citation graph.

by allowing multiple nodes to execute the attestation routine in parallel. However, if an attestation chain is used, as in Jin et al. [2010] and Asokan et al. [2015], a compromised node may discredit the entire chain. An important characteristic of both one-to-one and one-to-many approaches is that the verifier must be a trusted entity. This has significant implications because some proposed attestation mechanisms Require the prover to be in the verifier's communication range. Therefore, a mobile verifier or multiple verifiers scattered across the network are necessary. Moreover, a verifier becomes a single point of failure if only a single one exists. The main feature of the many-to-one pattern is that it can be performed without the use of trusted verifiers. This has both advantages and disadvantages. The upside is that it allows several nodes to be attested at the same time in a distributed manner without having a single point of failure. Besides, it closes the gap between detection of misbehaviour and attestation, since neighboring nodes are the direct observers of a compromised node. The downside is that it requires a minimum network density and is more vulnerable to compromised nodes as they can not only attempt to avoid detection but can also mislead the outcome of other nodes' attestation.

3.9. Overview

Having examined all characteristics and their instances, one can realize that there is no definitive attestation mechanism. Each approach has its own advantages and disadvantages. Choosing the best solution for a specific scenario depends on a series of factors, such as the assumptions made about the adversary, the environment in which the nodes are going to be deployed, how the sensors are going to be placed across it, and the underlying nodes hardware. Nevertheless, the discussion held in this section helps to clarify the tradeoffs between different techniques and can serve as a guideline to developers and engineers to find the techniques that best suits their needs.

Table II maps, in chronological order, a representative number of attestation mechanisms to the taxonomy illustrated in Figure 3. To the best of our knowledge, there are no existing mechanisms that would not fit on the proposed taxonomy. Furthermore, we believe that our taxonomy still holds outside the WSNs scenario. One important note is that we classify the mechanisms as they are first described in their original articles, without considering proposed extensions. To better visualise the influence of the mechanisms among themselves, we have plotted a citation graph, depicted in Figure 4. Vertices represent attestation mechanism articles while edges represent citations. An edge from vertex A to vertex B indicates that article A cites article B. Inside each vertex, there are two numbers: the number of citations it has received and the number of

Table II. Mapping of Some Existing Attestation Mechanisms to the Proposed Taxonomy

Approach	Evidence Acquisition	Integrity Measurement	Timing	Memory Traversal	Attestation Routine	Program Memory	Data Memory	Interaction Pattern
Reflection [Spinellis 2000]	Software based	Static	Loose	Sequential	Embedded	Unfilled	Unverified	One-to-one
SWATT [Seshadri et al. 2004]	Software based	Static	Strict	Cell-based pseudo-random	Embedded	Unfilled	Unverified	One-to-one
PIV [Park and Shin 2005]	Software based	Dynamic	Loose	Sequential	On-the-fly	Unfilled	Erased	One-to-one
Self-modifying code [Shaneck et al. 2005]	Software based	Static	Loose	Cell-based pseudo-random	On-the-fly	Unfilled	Unverified	One-to-one
SCUBA [Seshadri et al. 2006]	Hybrid	Dynamic	Strict	Cell-based pseudo-random	Embedded	Unfilled	Unverified	One-to-one
Proactive [Choi et al. 2007]	Software based	Static	Loose	Sequential	Embedded	Filled	Unverified	One-to-one
Distributed [Yang et al. 2007]	Software based	Static	Loose	Block-based pseudo-random	Embedded	Filled	Unverified	Many-to-one
SAKE [Seshadri et al. 2008]	Hybrid	Dynamic	Strict	Cell-based pseudo-random	Embedded	Unfilled	Unverified	One-to-one
Memory filling [AbuHmed et al. 2009]	Software based	Static	Loose	Block-based pseudo-random	Embedded	Filled	Unverified	One-to-one
ReDAS [Kil et al. 2009]	Hardware based	Dynamic	Loose	Sequential	Embedded	Unfilled	Verified	One-to-one
USAS [Jin et al. 2010]	Software based	Static	Loose	Cell-based pseudo-random	Embedded	Filled	Unverified	One-to-many
PoSE [Perito and Tsudik 2010]	Hybrid	Dynamic	Loose	Sequential	Embedded	Filled	Erased	One-to-one
DataGuard [Zhang and Liu 2010]	Software based	Dynamic	Loose	Sequential	Embedded	Unfilled	Verified	One-to-one
SMART [Eldefrawy et al. 2012]	Hybrid	Dynamic	Loose	Sequential	Embedded	Unfilled	Verified	One-to-one
Compression [Vetter and Westhoff 2012]	Hybrid	Static	Loose	Cell-based pseudo-random	Embedded	Filled	Unverified	One-to-one
Lightweight [Kiyomoto and Miyake 2014]	Software based	Static	Loose	Block-based pseudo-random	Embedded	Filled	Unverified	One-to-one
TrustLite [Koeberl et al. 2014]	Hybrid	Dynamic	Loose	Sequential	Embedded	Unfilled	Verified	One-to-one
PUFatt [Kong et al. 2014]	Hardware based	Dynamic	Strict	Cell-based pseudo-random	Embedded	Unfilled	Unverified	One-to-one
TRAP [Tan et al. 2015]	Hardware based	Static	Loose	Sequential	Embedded	Unfilled	Unverified	One-to-one
LRMA [Yang et al. 2015]	Software based	Static	Loose	Cell-based pseudo-random	Embedded	Unfilled	Unverified	One-to-one
SEDA [Asokan et al. 2015]	Hybrid	Static	Loose	Sequential	Embedded	Unfilled	Unverified	One-to-many
C-FLAT [Abera et al. 2016]	Hardware based	Dynamic	Loose	Sequential	Embedded	Unfilled	Verified	One-to-one
DARPA [Ibrahim et al. 2016]	Hybrid	Static	Loose	Sequential	Embedded	Unfilled	Erased	One-to-many

citations it has done. The main observation that we make is that even after compression and ROP attacks have been demonstrated [Castelluccia et al. 2009], new static approaches were proposed [Jin et al. 2010; Vetter and Westhoff 2012; Kiyomoto and Miyake 2014; Tan et al. 2015; Yang et al. 2015; Asokan et al. 2015; Ibrahim et al. 2016] that completely ignored or did not provide sufficient protection against such attacks. We believe that an attestation mechanism should reflect the actual running system and not just some static part of the prover's memory.

4. OPEN RESEARCH PROBLEMS

Existing attestation mechanisms are far from perfect, with much room for improvement. In this section, we examine open research problems and give directions on how to undertake them. We focus on four main topics, which we believe are of great importance and can receive further attention.

4.1. Overoptimistic Assumptions

Problems do not cease to exist just because they were assumed not to occur. In many proposed attestation mechanisms, there is a gap between the assumptions made and the adversary capabilities. Therefore, several existing approaches rely on strong premises that may not hold. For instance, SWATT [Seshadri et al. 2004] and SCUBA [Seshadri et al. 2006] rely on strict timing of the attestation routine execution. Nevertheless, as the authors themselves recognize, proving runtime optimality is still an open problem.

Other approaches assume that filling the empty spaces of a prover's program memory leaves the adversary with no space to store its malicious code without being detected. Yet, only two of these approaches [Perito and Tsudik 2010; Vetter and Westhoff 2012] are capable of defending against code compression attacks, and they have their own limitations. Perito and Tsudik [2010] requires all the prover's writable memories to be overwritten while Vetter and Westhoff [2012] requires a dedicated microcontroller.

Another type of attack that most approaches assume does not happen or impose restrictions to their occurrence are collusion attacks. Although it is possible to imagine solutions to these attacks, such as jamming the prover or having its neighbors monitor its communication during attestation, the feasibility, effectiveness, and impact of these solutions have not been analysed.

Several software-based proposals assume a secure and authentic communication channel between prover and verifier through the use of cryptographic keys defined prior to network deployment. However, the nodes used in these approaches do not have the necessary hardware to protect these keys, and, oddly enough, the absence of such hardware is the very motivation behind their development. Cryptographic keys can, however, be used on commodity sensor nodes when determined at runtime. For example, Software Attestation for Key Establishment (SAKE) [Seshadri et al. 2008] is a protocol for establishing shared keys between two neighbouring nodes without assuming prior authentic or secret information. However, SAKE is based on ICE, the same primitive used by SCUBA, which relies on strict time measurements. One alternative solution to authenticate sensor nodes is the use of Radio Frequency fingerprint techniques which enable the identification of individual devices by the unique characteristics of their radio transmitter [Ureten and Serinken 2007; Rehman et al. 2014; Knox and Kunz 2015].

The vast majority of attestation mechanisms do not pay much attention to the prover's security. They are developed under the assumption that the verifier can always be trusted. However, an adversary can take advantage of this and impersonate the verifier to perform DoS attacks targeting honest devices. Brasser et al. [2016] investigates attacks and countermeasures under this context. They describe three attacks an adversary may explore: replay, reorder, and delay of attestation challenges. The

countermeasures involve the use of a secret key for authenticating the verifier and the use of either a challenge counter or timestamp (which requires a synchronized clock between prover and verifier), all of which need hardware protection.

While it is impractical for an adversary to launch a large-scale attack that requires physical access to compromise a sensor node, it may be enough for an adversary to compromise just a few selected nodes in this manner. Nevertheless, the vast majority of software-based and hybrid approaches do not take into account physical attacks. As an alternative, Ibrahim et al. [2016] proposed Device Attestation Resilient to Physical Attacks (DARPA). Assuming that to perform a physical attack an adversary has to capture and temporarily disable the target sensor for a perceptible amount of time, DARPA requires all nodes to periodically broadcast a message that serves to prove the node is active. All nodes log these messages which can then be collected by the verifier during attestation. While the scheme may suffer from false positives because of device or network failures, it is a first step towards detecting physical attacks.

4.2. Effectiveness

WSNs are usually constituted of resource constrained nodes that have limited energy, low processing power, and little storage space. In many application scenarios, the network needs to operate unattended for long periods of time, so battery depletion is crucial. So while ensuring the integrity of the sensor network is important, it is equally important that security mechanisms should not have a high impact on the system performance. Attestation procedures typically degrade both network throughput and the node's battery lifetime. Furthermore, attestation only allows us to ascertain the state of the prover at a particular point in time. How often should then attestation be performed to provide assurance of the network's integrity while not depleting resources?

Chen et al. [2010] present a model to analyse the frequency with which code attestation should be performed to maximize sensors lifetime while effectively detecting compromised nodes. However, their model considers solely designs in which attestation is invoked probabilistically and ignores aspects such as using network monitoring information to decide when to trigger attestation next. Their model leads then to the rather straightforward conclusion that the frequency at which attestation must be performed depends directly on the rate at which nodes can be compromised (*compromise rate*). In practice, the compromise rate is unknown *a priori* and is highly dependent on many different factors such as the network deployment, types of attack, and value and timeliness of the information sensed by the network. For example, it has been shown that malicious packets can propagate quickly and compromise entire networks in short periods of time [Gu and Noorani 2008; Yang et al. 2008]. However, in their analysis, Chen et al. ignore such aspects and assume a compromise rate in the range of 0.0058 to 0.0072 nodes per hour, corresponding to a node being compromised once every 5 to 8 days.

Further analysis is needed to determine not only when to perform attestation but also which devices to attest and what to do when a compromised node is detected. For example, nodes closer to the base station may have a higher value as targets since they also forward much of the data from peripheral nodes to the base station and vice versa. Such aspects should be taken into account, for example, through different weights, when deciding which nodes to attest. Furthermore, a compromised node can easily broadcast malicious packets to its neighbors. So attesting the neighbours of a compromised sensor would be an effective way of reducing the adversary compromise rate and limit diffusion.

4.3. Time of Check to Time of Use

Perhaps the biggest issue when using attestation in WSN is the gap between the Time of Check to Time of Use (TOCTOU) [Bratus et al. 2008]. Attesting a node occurs at

a particular point in time and does not guarantee that the node has not been temporarily compromised before or that it will not be compromised right after attestation. An adversary can perform a TOCTOU attack as long as it has some unmeasured location to hide its malicious data before attestation starts and is able to reinstall it after attestation ends [Kovah et al. 2012]. In fact, Castelluccia et al. [2009] presents a rootkit-based attack that does exactly this. The difficulty arises from attesting all the memories of a prover. Some dynamic attestation approaches [Kil et al. 2009; Zhang and Liu 2010] try to close the TOCTOU gap by checking runtime properties of the software execution. However, it is not always possible to predict the behaviour of dynamic data, and these approaches end up covering only a subset of the required properties. ICE [Seshadri et al. 2006] attempts to guarantee an untampered execution environment to ensure that a piece of code runs unmodified. The limitation of this approach is that it applies solely to self-contained code that does not invoke other software on the prover and executes with interrupts disabled. Another approach to reduce the TOCTOU gap could be the integration of attestation and Control Flow Integrity (CFI) [Abadi et al. 2005] techniques. Protecting the control flow of a program prevents adversaries from arbitrarily altering its execution. CFI techniques achieve this protection by embedding control code in the original application program. Attesting that both control and application code have not been altered is a further indication that the original program is being executed as it was supposed to. Ferguson and Gu [2011] implements control flow protection in the context of WSNs; however, the described scheme has not been combined with any attestation mechanism. Control-Flow ATtestation (C-FLAT) [Abera et al. 2016] is a similar, but different, approach. Instead of embedding code to protect the authentic application flow, it embeds code to monitor its execution path. This allows the verifier to attest the prover's runtime behaviour, detecting any control flow diversion. Nonetheless, it is important to note that CFI is not infallible [Goktas et al. 2014] and does not ensure data flow security [Chen et al. 2005; Castro et al. 2006; Hu et al. 2016].

Significant effort has been dedicated recently to approaches capable of defending against ROP attacks. These broadly fall into two categories: one is the already-mentioned CFI, while the other is software diversification [Larsen et al. 2014]. However, until now, all proposed mechanisms can be subverted in some way. Therefore, ideas continue to evolve, and new approaches continue to emerge. A promising countermeasure to such attacks is the use of execute-only memory [Backes et al. 2014; Crane et al. 2015], which allows marking pieces of memory as executable but not readable. However, in order to attest a device, the attestation routine needs to read the code it executes, and, thus far, no approach integrates the two techniques.

4.4. Scalability

To attest a device, a verifier must know its expected internal state and hardware architecture. In a simple WSN scenario, all the nodes may be executing the same application on the same hardware platform, making this reasonably easy. However, in many other situations, the network is typically heterogeneous and comprises different types of sensors or different cluster nodes execute different applications. In more extreme cases, the network may contain many sensors from distinct vendors executing several different applications. How can one then store and manage the information necessary to attest these devices in a scalable manner? The approach proposed in Park and Shin [2005] partitions the application code into multiple blocks, calculates a digest for each block, and classifies the block according to whether it belongs to all, a group, or a unique node. Although this method allows us to significantly reduce the amount of information needed, it also has its own security vulnerabilities. For instance, an adversary may compute the prover's digests before modifying its memory and then only keep a copy

of the digests to pass attestation. Another approach is to have similar devices, which execute the same application, attesting each other in a distributed fashion. However, to be viable, this approach requires a minimum density of similar devices scattered across the network.

Software updates further complicate the issue. The slightest change in the software a prover executes can impact the attestation outcome. So additional information needs to be maintained to know which devices are running which versions. Sadeghi and Stüble [2004] propose to attest a device based on the properties that it offers instead of its software and hardware components. However, in most scenarios, property-based attestation requires the use of a trusted third party to map the device's components to properties [Chen et al. 2008]. Moreover, the identification and formal definition of security properties are still open problems [Nagarajan et al. 2009].

A more recent approach, named Scalable Embedded Device Attestation (SEDA) [Asokan et al. 2015], was designed with the main goal of attesting a large amount of devices. The most remarkable aspect of SEDA is that the verifier does not need to know the detailed configuration of all provers. However, to achieve this, the approach relies on cryptographic secrets that must be secured by hardware. With these secrets, the nodes can be issued a certificate of its valid software configuration, during initialization or after an update, and can share the certificate with their neighbors. Attestation is performed in a one-to-many interaction pattern, where each device attests its neighbors and reports back to its parent. As stated in Section 3.8, the main limitation of this interaction pattern is that a compromised node invalidates the results for all the nodes following it on the attestation chain. In the worst-case scenario, the first node in the chain has been compromised, and resources used to perform attestation are discarded. Another limitation of this approach is that it only provides a static integrity measurement.

5. CONCLUSIONS

Attestation is a critical service that enables the detection of compromised devices. However, there is no consensus on how to perform attestation in an effective and secure manner. Consequently, several mechanisms have been proposed over the past few years. They differ not only in design choices and implementation but also in the assumptions they make over the system and adversary models. As we have shown in this article, some assumptions are more realistic than others in the context of WSNs, and not all proposals can be considered secure. Furthermore, we presented a taxonomy that identifies the main characteristics of proposed solutions and surveyed the state of the art mapping existing approaches to our taxonomy. This allowed us to discuss the tradeoffs between design choices made by different proposals. Finally, we have identified open research issues and given directions on how to tackle them.

Although we have centered the discussion around WSNs, we believe our taxonomy is still relevant outside this context. Moreover, all the analysis of advantages and disadvantages of different techniques done in this article can be directly applied to the Internet of Things scenario, since it shares many characteristics in common with sensor networks. We hope this work serves both as a reference to avoid already committed mistakes and as a guide to future attestation mechanisms.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their comments and suggestions, which were valuable contributions to the manuscript. We are also grateful to the members of the RISS group, especially Daniele Sgandurra, for their helpful feedback.

REFERENCES

Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. 2005. Control-flow integrity. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*. ACM, New York, NY, 340–353. DOI: http://dx.doi.org/10.1145/1102120.1102165

Tigist Abera, N. Asokan, Lucas Davi, Jan-Erik Ekberg, Thomas Nyman, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. 2016. C-FLAT: Control-flow attestation for embedded systems software. arXiv preprint arXiv:1605.07763 (2016).

Tamer AbuHmed, Nandinbold Nyamaa, and DaeHun Nyang. 2009. Software-based remote code attestation in wireless sensor network. In *Proceedings of the 28th IEEE Conference on Global Telecommunications (GLOBECOM'09)*. IEEE Press, Piscataway, NJ, 1–8. DOI: http://dx.doi.org/10.1109/GLOCOM.2009.5425280

Ross Anderson and Markus Kuhn. 1996. Tamper resistance: A cautionary note. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce (WOEC'96)*, Vol. 2. USENIX Association, Berkeley, CA, 1–11.

Frederik Armknecht, Ahmad-Reza Sadeghi, Steffen Schulz, and Christian Wachsmann. 2013. A security framework for the analysis and design of software attestation. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS'13)*. ACM, New York, NY, 1–12. DOI: http://dx.doi.org/10.1145/2508859.2516650

N. Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. 2015. SEDA: Scalable embedded device attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*. ACM, New York, NY, 964–975. DOI: http://dx.doi.org/10.1145/2810103.2813670

Michael Backes, Thorsten Holz, Benjamin Kollenda, Philipp Koppe, Stefan Nürnberger, and Jannik Pewny. 2014. You can run but you can't read: Preventing disclosure exploits in executable code. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS'14)*. ACM, New York, NY, 1342–1353. DOI: http://dx.doi.org/10.1145/2660267.2660378

Luís M. Borges, Fernando J. Velez, and António S. Lebres. 2014. Survey on the characterization and classification of wireless sensor network applications. *IEEE Commun. Surv. Tutor.* 16, 4 (2014), 1860–1890. DOI: http://dx.doi.org/10.1109/COMST.2014.2320073

Ferdinand Brasser, Kasper B. Rasmussen, Ahmad-Reza Sadeghi, and Gene Tsudik. 2016. Remote attestation for low-end embedded devices: The prover's perspective. In *Proceedings of the Design Automation Conference (DAC)*.

Sergey Bratus, Nihal D'Cunha, Evan Sparks, and Sean W. Smith. 2008. TOCTOU, traps, and trusted computing. In *Proceedings of the 1st International Conference on Trusted Computing and Trust in Information Technologies: Trusted Computing—Challenges and Applications (Trust'08)*. Springer-Verlag, Berlin, 14–32. DOI: http://dx.doi.org/10.1007/978-3-540-68979-9_2

Erik Buchanan, Ryan Roemer, Hovav Shacham, and Stefan Savage. 2008. When good instructions go bad: Generalizing return-oriented programming to RISC. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*. ACM, New York, NY, 27–38. DOI: http://dx.doi.org/10.1145/1455770.1455776

Claude Castelluccia, Aurélien Francillon, Daniele Perito, and Claudio Soriente. 2009. On the difficulty of software-based attestation of embedded devices. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09)*. ACM, New York, NY, 400–409. DOI: http://dx.doi.org/10.1145/1653662.1653711

Miguel Castro, Manuel Costa, and Tim Harris. 2006. Securing software by enforcing data-flow integrity. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI'06)*. USENIX Association, Berkeley, CA, 147–160.

Stephen Checkoway, Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, Hovav Shacham, and Marcel Winandy. 2010. Return-oriented programming without returns. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS'10)*. ACM, New York, NY, 559–572. DOI: http://dx.doi.org/10.1145/1866307.1866370

Ing-Ray Chen, Yating Wang, and Ding-Chau Wang. 2010. Reliability of wireless sensors with code attestation for intrusion detection. *Inform. Process. Lett.* 110, 17 (2010), 778–786. DOI: http://dx.doi.org/10.1016/j.ipl.2010.06.007

Liqun Chen, Hans Löhr, Mark Manulis, and Ahmad-Reza Sadeghi. 2008. Property-based attestation without a trusted third party. In *Proceedings of the 11th International Conference on Information Security (ISC'08)*. Springer-Verlag, Berlin, 31–46. DOI: http://dx.doi.org/10.1007/978-3-540-85886-7_3

Shuo Chen, Jun Xu, Emre C. Sezer, Prachi Gauriar, and Ravishankar K. Iyer. 2005. Non-control-data attacks are realistic threats. In *Proceedings of the 14th Conference on USENIX Security Symposium (SSYM'05)*. USENIX Association, Berkeley, CA.

Young-Geun Choi, Jeonil Kang, and DaeHun Nyang. 2007. Proactive code verification protocol in wireless sensor network. In *Proceedings of the 2007 International Conference on Computational Science and Its Applications (ICCSA'07)*. Springer-Verlag, Berlin, 1085–1096. DOI:http://dx.doi.org/10.1007/978-3-540-74477-1_97

George Coker, Joshua Guttman, Peter Loscocco, Amy Herzog, Jonathan Millen, Brian O'Hanlon, John Ramsdell, Ariel Segall, Justin Sheehy, and Brian Sniffen. 2011. Principles of remote attestation. *Int. J. Inform. Secur.* 10, 2 (2011), 63–81. DOI:http://dx.doi.org/10.1007/s10207-011-0124-7

Christian Collberg, Clark Thomborson, and Douglas Low. 1998. Manufacturing cheap, resilient, and stealthy opaque constructs. In *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'98)*. ACM, New York, NY, 184–196. DOI:http://dx.doi.org/10.1145/268946.268962

Crispin Cowan, Calton Pu, Dave Maier, Heather Hintony, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, and Qian Zhang. 1998. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th Conference on USENIX Security Symposium (SSYM'98)*. USENIX Association, Berkeley, CA, 63–78.

Stephen Crane, Christopher Liebchen, Andrei Homescu, Lucas Davi, Per Larsen, Ahmad-Reza Sadeghi, Stefan Brunthaler, and Michael Franz. 2015. Return to where? You cant exploit what you cant find. In *Proceedings of Black Hat USA*.

Ana Paula R. da Silva, Marcelo H. T. Martins, Bruno P. S. Rocha, Antonio A. F. Loureiro, Linnyer B. Ruiz, and Hao Chi Wong. 2005. Decentralized intrusion detection in wireless sensor networks. In *Proceedings of the 1st ACM International Workshop on Quality of Service and Security in Wireless and Mobile Networks (Q2SWinet'05)*. ACM, New York, NY, 16–23. DOI:http://dx.doi.org/10.1145/1089761.1089765

Anupam Datta, Jason Franklin, Deepak Garg, and Dilsun Kaynar. 2009. A logic of secure systems and its application to trusted computing. In *Proceedings of the 30th IEEE Symposium on Security and Privacy (SP'09)*. IEEE Computer Society, Washington, DC, 221–236. DOI:http://dx.doi.org/10.1109/SP.2009.16

Fred Douglis. 1993. The compression cache: Using on-line compression to extend physical memory. In *Proceedings of the 1993 USENIX Winter Conference*. USENIX Association, Berkeley, CA, 519–529.

Karim Eldefrawy, Gene Tsudik, Aurélien Francillon, and Daniele Perito. 2012. SMART: Secure and minimal architecture for (establishing dynamic) root of trust. In *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS'12)*. Internet Society, 1–15.

Christopher Ferguson and Qijun Gu. 2011. Self-healing control flow protection in sensor applications. *IEEE Trans. Depend. Secure Comput.* 8, 4 (2011), 602–616. DOI:http://dx.doi.org/10.1109/TDSC.2011.15

Aurélien Francillon and Claude Castelluccia. 2008. Code injection attacks on harvard-architecture devices. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*. ACM, New York, NY, 15–26. DOI:http://dx.doi.org/10.1145/1455770.1455775

Aurélien Francillon, Claude Castelluccia, Daniele Perito, and Claudio Soriente. 2010. Comments on "Refutation of On the Difficulty of Software-Based Attestation of Embedded Devices."

Aurélien Francillon, Quan Nguyen, Kasper B. Rasmussen, and Gene Tsudik. 2014. A minimalist approach to remote attestation. In *Proceedings of the 2014 Conference on Design, Automation and Test in Europe (DATE'14)*. European Design and Automation Association, 3001 Leuven, Belgium, 1–6.

Karine Gandolfi, Christophe Mourtel, and Francis Olivier. 2001. Electromagnetic analysis: Concrete results. In *Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems (CHES'01)*. Springer-Verlag, Berlin, 251–261. DOI:http://dx.doi.org/10.1007/3-540-44709-1_21

Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. 2002. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*. ACM, New York, NY, 148–160. DOI:http://dx.doi.org/10.1145/586110.586132

Thanassis Giannetsos and Tassos Dimitriou. 2013. Spy-sense: Spyware tool for executing stealthy exploits against sensor networks. In *Proceedings of the 2nd ACM Workshop on Hot Topics on Wireless Network Security and Privacy (HotWiSec'13)*. ACM, New York, NY, 7–12. DOI:http://dx.doi.org/10.1145/2463183.2463186

Enes Goktas, Elias Athanasopoulos, Herbert Bos, and Georgios Portokalidis. 2014. Out of control: Overcoming control-flow integrity. In *Proceedings of the 35th IEEE Symposium on Security and Privacy (SP'14)*. IEEE Computer Society, Washington, DC, 575–589. DOI:http://dx.doi.org/10.1109/SP.2014.43

Qijun Gu and Rizwan Noorani. 2008. Towards self-propagate mal-packets in sensor networks. In *Proceedings of the 1st ACM Conference on Wireless Network Security (WiSec'08)*. ACM, New York, NY, 172–182. DOI:http://dx.doi.org/10.1145/1352533.1352563

Carl Hartung, Richard Han, Carl Seielstad, and Saxon Holbrook. 2006. FireWxNet: A multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In *Proceedings of the*

4th International Conference on Mobile Systems, Applications and Services (MobiSys'06). ACM, New York, NY, 28–41. DOI:http://dx.doi.org/10.1145/1134680.1134685

Michael Healy, Thomas Newe, and Elfed Lewis. 2008. Wireless sensor node hardware: A review. In *Proceedings of the 7th IEEE Sensors Conference*. IEEE Computer Society, Washington, DC, 621–624. DOI:http://dx.doi.org/10.1109/ICSENS.2008.4716517

Clemens Helfmeier, Christian Boit, Dmitry Nedospasov, and Jean-Pierre Seifert. 2013. Cloning physically unclonable functions. In *Proceeding of the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST'13)*. 1–6. DOI:http://dx.doi.org/10.1109/HST.2013.6581556

Mark Hempstead, Michael J. Lyons, David Brooks, and Gu-Yeon Wei. 2008. Survey of hardware systems for wireless sensor networks. *J. Low Power Electron.* 4, 1 (2008), 11–20. DOI:http://dx.doi.org/10.1166/jolpe.2008.156

Jason Hill, Mike Horton, Ralph Kling, and Lakshman Krishnamurthy. 2004. The platforms enabling wireless sensor networks. *Commun. ACM* 47, 6 (2004), 41–46. DOI:http://dx.doi.org/10.1145/990680.990705

Hong Hu, Shweta Shinde, Sendroiu Adrian, Zheng Leong Chua, Prateek Saxena, and Zhenkai Liang. 2016. Data-oriented programming: On the expressiveness of non-control data attacks. In *Proceedings of the 37th IEEE Symposium on Security and Privacy (SP'16)*. IEEE Computer Society, Washington, DC, 969–986.

Ahmad Ibrahim, Ahmad-Reza Sadeghi, Gene Tsudik, and Shaza Zeitouni. 2016. DARPA: Device attestation resilient to physical attacks. In *Proceedings of the 9th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'16)*. ACM, New York, NY, 171–182. DOI:http://dx.doi.org/10.1145/2939918.2939938

Xinyu Jin, Pasd Putthapipat, Deng Pan, Niki Pissinou, and S. Kami Makki. 2010. Unpredictable software-based attestation solution for node compromise detection in mobile WSN. In *Proceedings of the 29th IEEE Conference on Global Telecommunications Workshops (GLOBECOM'10)*. IEEE Press, Piscataway, NJ, 2059–2064. DOI:http://dx.doi.org/10.1109/GLOCOMW.2010.5700307

Bernhard Kauer. 2007. OSLO: Improving the security of trusted computing. In *Proceedings of the 16th USENIX Security Symposium (SS'07)*. USENIX Association, Berkeley, CA, 1–9.

Rick Kennell and Leah H. Jamieson. 2003. Establishing the genuinity of remote computer systems. In *Proceedings of the 12th Conference on USENIX Security Symposium (SSYM'03)*. USENIX Association, Berkeley, CA, 295–310.

Rick Kennell and Leah H. Jamieson. 2004. *An Analysis of Proposed Attacks Against Genuinity Tests*. Technical Report. CERIAS, Purdue University, West Lafayette.

Chongkyung Kil, Emre C. Sezer, Ahmed M. Azab, Peng Ning, and Xiaolan Zhang. 2009. Remote attestation to dynamic system properties: Towards providing complete system integrity evidence. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'09)*. IEEE Computer Society, Washington, DC, 115–124. DOI:http://dx.doi.org/10.1109/DSN.2009.5270348

Shinsaku Kiyomoto and Yutaka Miyake. 2014. Lightweight attestation scheme for wireless sensor network. *Int. J. Secur. Appl.* 8, 2 (2014), 25–40.

David A. Knox and Thomas Kunz. 2015. Wireless fingerprints inside a wireless sensor network. *ACM Trans. Sens. Netw.* 11, 2 (2015), 37:1–37:30. DOI:http://dx.doi.org/10.1145/2658999

Jeonggil Ko, Jong Hyun Lim, Yin Chen, Rvăzvan Musvaloiu-E, Andreas Terzis, Gerald M. Masson, Tia Gao, Walt Destler, Leo Selavo, and Richard P. Dutton. 2010. MEDiSN: Medical emergency detection in sensor networks. *ACM Trans. Embed. Comput. Syst.* 10, 1 (2010), 11:1–11:29. DOI:http://dx.doi.org/10.1145/1814539.1814550

Paul C. Kocher. 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'96)*. Springer-Verlag, Berlin, 104–113. DOI:http://dx.doi.org/10.1007/3-540-68697-5_9

Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'99)*. Springer-Verlag, Berlin, 388–397. DOI:http://dx.doi.org/10.1007/3-540-48405-1_25

Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. 2014. TrustLite: A security architecture for tiny embedded devices. In *Proceedings of the 9th European Conference on Computer Systems (EuroSys'14)*. ACM, New York, NY, 1–14. DOI:http://dx.doi.org/10.1145/2592798.2592824

Joonho Kong, Farinaz Koushanfar, Praveen K. Pendyala, Ahmad-Reza Sadeghi, and Christian Wachsmann. 2014. PUFatt: Embedded platform attestation based on novel processor-based PUFs. In *Proceedings of the 51st Annual Design Automation Conference (DAC'14)*. ACM, New York, NY, 1–6. DOI:http://dx.doi.org/10.1145/2593069.2593192

Xeno Kovah, Corey Kallenberg, Chris Weathers, Alexander Herzog, Matthew Albin, and John Butterworth. 2012. New results for timing-based attestation. In *Proceedings of the 33rd IEEE Symposium on*

Security and Privacy (SP'12). IEEE Computer Society, Washington, DC, 239–253. DOI:http://dx.doi.org/10.1109/SP.2012.45

Klaus Kursawe, Dries Schellekens, and Bart Preneel. 2005. Analyzing trusted platform communication. In *Proceedings of the ECRYPT Workshop on Cryptographic Advances in Secure Hardware (CRASH'05)*.

Per Larsen, Andrei Homescu, Stefan Brunthaler, and Michael Franz. 2014. SoK: Automated software diversity. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy (SP'14)*. IEEE Computer Society, Washington, DC, 276–291. DOI:http://dx.doi.org/10.1109/SP.2014.25

Li Li, Hong Hu, Jun Sun, Yang Liu, and Jin Song Dong. 2014. Practical analysis framework for software-based attestation scheme. In *Proceedings of the 16th International Conference on Formal Engineering Methods (ICFEM'14)*. Springer International Publishing, Switzerland, 284–299. DOI:http://dx.doi.org/10.1007/978-3-319-11737-9_19

Javier Lopez, Rodrigo Roman, Isaac Agudo, and Carmen Fernandez-Gago. 2010. Trust management systems for wireless sensor networks: Best practices. *Comput. Commun.* 33, 9 (2010), 1086–1093. DOI:http://dx.doi.org/10.1016/j.comcom.2010.02.006

Dominik Merli, Dieter Schuster, Frederic Stumpf, and Georg Sigl. 2011. Side-channel analysis of PUFs and fuzzy extractors. In *Proceedings of the 4th International Conference on Trust and Trustworthy Computing (TRUST'11)*. Springer-Verlag, Berlin, 33–47. DOI:http://dx.doi.org/10.1007/978-3-642-21599-5_3

Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

Aarthi Nagarajan, Vijay Varadharajan, Michael Hitchens, and Eimear Gallery. 2009. Property based attestation and trusted computing: Analysis and challenges. In *Proceedings of the 3rd International Conference on Network and System Security (NSS'09)*. IEEE Computer Society, Washington, DC, 278–285. DOI:http://dx.doi.org/10.1109/NSS.2009.83

James Newsome, Elaine Shi, Dawn Song, and Adrian Perrig. 2004. The Sybil attack in sensor networks: Analysis & defenses. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN'04)*. ACM, New York, NY, 259–268. DOI:http://dx.doi.org/10.1145/984622.984660

Aleph One. 1996. Smashing the stack for fun and profit. *Phrack Mag.* 7, 49 (1996).

Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. 2002. Physical one-way functions. *Science* 297, 5589 (2002), 2026–2030. DOI:http://dx.doi.org/10.1126/science.1074376

Taejoon Park and Kang G. Shin. 2005. Soft tamper-proofing via program integrity verification in wireless sensor networks. *IEEE Trans. Mobile Comput.* 4, 3 (2005), 297–309. DOI:http://dx.doi.org/10.1109/TMC.2005.44

Bryan Parno. 2008. Bootstrapping trust in a "trusted" platform. In *Proceedings of the 3rd Conference on Hot Topics in Security (HOTSEC'08)*. USENIX Association, Berkeley, CA, 1–6.

Daniele Perito and Gene Tsudik. 2010. Secure code update for embedded devices via proofs of secure erasure. In *Proceedings of the 15th European Conference on Research in Computer Security (ESORICS'10)*. Springer-Verlag, Berlin, 643–662. DOI:http://dx.doi.org/10.1007/978-3-642-15497-3_39

Adrian Perrig and Leendert Van Doorn. 2010. Refutation of "On the Difficulty of Software-Based Attestation of Embedded Devices." (2010).

Emil M. Petriu, Nicolas D. Georganas, Dorina C. Petriu, Dimitrios Makrakis, and Voicu Z. Groza. 2000. Sensor-based information appliances. *IEEE Instrum. Meas. Mag.* 3, 4 (2000), 31–35. DOI:http://dx.doi.org/10.1109/5289.887458

Saeed Ur Rehman, Kevin W. Sowerby, and Colin Coghill. 2014. Analysis of impersonation attacks on systems using RF fingerprinting and low-end receivers. *J. Comput. Syst. Sci.* 80, 3 (2014), 591–601. DOI:http://dx.doi.org/10.1016/j.jcss.2013.06.013

Rodrigo Roman, Jianying Zhou, and Javier Lopez. 2006. Applying intrusion detection systems to wireless sensor networks. In *Proceedings of the 3rd IEEE Consumer Communications and Networking Conference (CCNC'06)*. IEEE Press, Piscataway, NJ, 640–644. DOI:http://dx.doi.org/10.1109/CCNC.2006.1593102

Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. 2010. Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS'10)*. ACM, New York, NY, 237–249. DOI:http://dx.doi.org/10.1145/1866307.1866335

Ahmad-Reza Sadeghi and Christian Stüble. 2004. Property-based attestation for computing platforms: Caring about properties, not mechanisms. In *Proceedings of the 2004 New Security Paradigms Workshop (NSPW'04)*. ACM, New York, NY, 67–77. DOI:http://dx.doi.org/10.1145/1065907.1066038

Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. 2004. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th Conference on USENIX Security Symposium (SSYM'04)*. USENIX Association, Berkeley, CA, 223–238.

Arvind Seshadri, Mark Luk, and Adrian Perrig. 2008. SAKE: Software attestation for key establishment in sensor networks. In *Proceedings of the 4th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS'08)*. Springer-Verlag, Berlin, 372–385. DOI:http://dx.doi.org/10.1007/978-3-540-69170-9_25

Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. 2006. SCUBA: Secure code update by attestation in sensor networks. In *Proceedings of the 5th ACM Workshop on Wireless Security (WiSe'06)*. ACM, New York, NY, 85–94. DOI:http://dx.doi.org/10.1145/1161289.1161306

Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. 2005. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. *ACM SIGOPS Operat. Syst. Rev.* 39, 5 (2005), 1–16. DOI:http://dx.doi.org/10.1145/1095809.1095812

Arvind Seshadri, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. 2004. SWATT: Software-based attestation for embedded devices. In *Proceedings of the 25th IEEE Symposium on Security and Privacy (SP'04)*. IEEE Computer Society, Washington, DC, 272–282. DOI:http://dx.doi.org/10.1109/SECPRI.2004.1301329

Hovav Shacham. 2007. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07)*. ACM, New York, NY, 552–561. DOI:http://dx.doi.org/10.1145/1315245.1315313

Mark Shaneck, Karthikeyan Mahadevan, Vishal Kher, and Yongdae Kim. 2005. Remote software-based attestation for wireless sensors. In *Proceedings of the 2nd European Conference on Security and Privacy in Ad-Hoc and Sensor Networks (ESAS'05)*. Springer-Verlag, Berlin, 27–41. DOI:http://dx.doi.org/10.1007/11601494_3

Umesh Shankar, Monica Chew, and J. Doug Tygar. 2004a. Side effects are not sufficient to authenticate software. In *Proceedings of the 13th Conference on USENIX Security Symposium (SSYM'04)*. USENIX Association, Berkeley, CA.

Umesh Shankar, Monica Chew, and J. Doug Tygar. 2004b. *Side Effects Are Not Sufficient to Authenticate Software: Addendum*. Technical Report. EECS Department, University of California, Berkeley.

Evan R. Sparks. 2007. *A Security Assessment of Trusted Platform Modules*. Technical Report. TR2007-597, Department of Computer Science, Dartmouth College.

Diomidis Spinellis. 2000. Reflection as a mechanism for software integrity verification. *ACM Trans. Inform. Syst. Secur.* 3, 1 (2000), 51–62. DOI:http://dx.doi.org/10.1145/353323.353383

Bo Sun, Lawrence Osborne, Yang Xiao, and Sghaier Guizani. 2007. Intrusion detection techniques in mobile ad hoc and wireless sensor networks. *IEEE Wireless Commun.* 14, 5 (2007), 56–63. DOI:http://dx.doi.org/10.1109/MWC.2007.4396943

Hailun Tan, Wen Hu, and Sanjay Jha. 2015. A remote attestation protocol with trusted platform modules (TPMs) in wireless sensor networks. *Secur. Commun. Netw.* 8, 13 (2015), 2171–2188. DOI:http://dx.doi.org/10.1002/sec.1162

Trusted Computing Group TCG. 2011. *TPM Main Specification Level 2 Version 1.2, Revision 116*. Retrieved from http://www.trustedcomputinggroup.org/resources/tpm_main_specification, Last accessed: 10 August 2016.

Oktay Ureten and Nur Serinken. 2007. Wireless security through RF fingerprinting. *Can. J. Electr. Comput. Eng.* 32, 1 (2007), 27–33. DOI:http://dx.doi.org/10.1109/CJECE.2007.364330

Lodewijk F. W. van Hoesel, Tim Nieberg, Harm J. Kip, and Paul J. M. Havinga. 2004. Advantages of a TDMA based, energy-efficient, self-organizing MAC protocol for WSNs. In *Proceedings of the 59th IEEE Vehicular Technology Conference (VTC'04)*. 1598–1602. DOI:http://dx.doi.org/10.1109/VETECS.2004.1390523

Benjamin Vetter and Dirk Westhoff. 2012. Simulation study on code attestation with compressed instruction code. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM'12)*. IEEE Computer Society, Washington, DC, 296–301. DOI:http://dx.doi.org/10.1109/PerComW.2012.6197498

Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. 2006. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI'06)*. USENIX Association, Berkeley, CA, 381–396.

Johannes Winter and Kurt Dietrich. 2013. A hijacker's guide to communication interfaces of the trusted platform module. *Comput. Math. Appl.* 65, 5 (2013), 748–761. DOI:http://dx.doi.org/10.1016/j.camwa.2012.06.018

Glenn Wurster, Paul C. Van Oorschot, and Anil Somayaji. 2005. A generic attack on checksumming-based software tamper resistance. In *Proceedings of the 26th IEEE Symposium on Security and Privacy (SP'05)*. IEEE Computer Society, Washington, DC, 127–138. DOI:http://dx.doi.org/10.1109/SP.2005.2

Xinyu Yang, Xiaofei He, Wei Yu, Jie Lin, Rui Li, Qingyu Yang, and Houbing Song. 2015. Towards a low-cost remote memory attestation for the smart grid. *Sensors* 15, 8 (2015), 20799–20824.

Yi Yang, Xinran Wang, Sencun Zhu, and Guohong Cao. 2007. Distributed software-based attestation for node compromise detection in sensor networks. In *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS'07)*. IEEE Computer Society, Washington, DC, 219–230. DOI:http://dx.doi.org/10.1109/SRDS.2007.31

Yi Yang, Sencun Zhu, and Guohong Cao. 2008. Improving sensor network immunity under worm attacks: A software diversity approach. In *Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'08)*. ACM, New York, NY, 149–158. DOI:http://dx.doi.org/10.1145/1374618.1374640

Dazhi Zhang and Donggang Liu. 2010. DataGuard: Dynamic data attestation in wireless sensor networks. In *Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'10)*. IEEE Computer Society, Washington, DC, 261–270. DOI:http://dx.doi.org/10.1109/DSN.2010.5544307

Yun Zhou, Yuguang Fang, and Yanchao Zhang. 2008. Securing wireless sensor networks: A survey. *IEEE Commun. Surv. Tutor.* 10, 3 (2008), 6–28. DOI:http://dx.doi.org/10.1109/COMST.2008.4625802

Received December 2015; revised August 2016; accepted August 2016