IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 16, 2021, accepted October 4, 2021, date of publication October 8, 2021, date of current version October 26, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3119220

A Taxonomy and Review of Remote Attestation Schemes in Embedded Systems

WILLIAM A. JOHNSON[1], SHEIKH GHAFOOR[1], (Member, IEEE), AND STACY PROWELL[2], (Senior Member, IEEE)

[1]Department of Computer Science, Tennessee Technological University, Cookeville, TN 38505, USA
[2]National Security Sciences Directorate, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

Corresponding author: William A. Johnson (wajohnson43@tntech.edu)

This work was supported in part by the U.S. Department of Energy's Cybersecurity for Energy Delivery Systems Program; in part by the Cybercorps Scholarship for Service (SFS); and in part by the Tennessee Tech Cybersecurity Education Research and Outreach Center (CEROC).

ABSTRACT Embedded systems that make up the Internet of Things (IoT), Supervisory Control and Data Acquisition (SCADA) networks, and Smart Grid applications are coming under increasing scrutiny in the security field. Remote Attestation (RA) is a security mechanism that allows a trusted device, the verifier, to determine the trustworthiness of an untrusted device, the prover. RA has become an area of high interest in academia and industry and many research works on RA have been published in recent years. This paper reviewed the published RA research works from 2003-2020. Our contributions are fourfold. First, we have re-framed the problem of RA into 5 smaller problems: root of trust, evidence type, evidence gathering, packaging and verification, and scalability. We have provided a holistic review of RA by discussing the relationships between these problems and the various solutions that exist in modern RA research. Second, we have presented an enhanced threat model that allows for a greater understanding of the security benefits of a given RA scheme. Third, we have proposed a taxonomy to classify and analyze RA research works and use it to categorize 58 RA schemes reported in literature. Fourth, we have provided cost benefit analysis details of each RA scheme surveyed such that security professionals may perform a cost benefit analysis in the context of their own challenges. Our classification and analysis has revealed areas of future research that have not been addressed by researchers rigorously.

INDEX TERMS Remote attestation, embedded systems security, fileless malware.

I. INTRODUCTION

Embedded systems have become ubiquitous in our modern lives. They make up the backbone of several emerging systems including the Internet of Things (IoT), Smart Homes, Smart Metering, Supervisory Control and Data Acquisition (SCADA) systems that are used for industrial control and smart grids, and Body Area Networks (BAN)s. We take the definition of an embedded system as any general purpose computing device whose software is intended to behave in a defined, specific way [1]. Despite our reliance on these devices, security is often an afterthought in their design, development, and production. As a result they are vulnerable to exploitation by attackers for malicious purposes. The Maria botnet [2], the Stuxnet incident [3], and the Ukrainian power grid attack [4] all offer cautionary tales about unsecured embedded systems.

The associate editor coordinating the review of this manuscript and approving it for publication was Wen Chen.

The foundation for security in any system is trust. If a device cannot be trusted, then all other security applications will fail. Secret keys will be leaked to attackers, privileges will be escalated, and vital security features like firewalls and antiviruses will be disabled or misconfigured. The criteria a device must meet to be considered trustworthy change depending on the severity of a possible compromise. In the simplest case, if a device can prove that its hardware has not been altered, it might be considered trustworthy. But even if the hardware is trustworthy the system can be vulnerable to attack because the software may be compromised. In a more complex case, a device might have to prove that its executable files have not been changed. In the worst case, a device must prove that its Operating System (OS), file system, kernel, and volatile storage (RAM) have not been infected with malware.

Meeting these criteria can be difficult. One solution is to perform traditional computer forensics on the device in question. To do this, an analyst would power the device off and inspect its components to ensure that the device has

not been physically tampered with. The analyst would then mount the hard drive of the device to another computer and inspect the executable files on the device to ensure that they have not been infected. They would then search the hard drive for evidence that might suggest malware has staged on the hard drive to deploy in memory.

Traditional forensics can fail in embedded systems for four reasons. 1) Embedded systems are often deployed over large areas under hard to reach conditions. Examples might include sensors in a smart grid system or an oil pipeline. Under these circumstances, reaching each sensor to physically examine it may be too labor or cost intensive. 2) Some embedded systems, like smart home devices, are not designed to be disassembled. This could stop an attacker from physically tampering with the device, but it can also stop an analyst from performing traditional forensics. 3) Programmable Logic Controllers (PLCs) in SCADA systems have an intense need for trust as well as real time constraints. Since traditional forensics rely on human intervention, they are often too slow to meet real time constraints. 4) File-less malware is any malicious code that can deploy on a system without ever touching the hard drive of a device. Since it leaves no evidence behind, traditional forensics will be unable to detect it [5].

Remote Attestation (RA) is a security mechanism that assures a device can be trusted. A trusted third party (the verifier) is responsible for challenging a suspicious device (the prover) to prove that it has not been compromised. RA can offer a variety of assurances in different ways, each with their own benefits and drawbacks. Generally, RA is broken into categories based on how it establishes a root of trust in the message that the prover sends to the verifier. Without a root of trust, no evidence provided by the prover can be trusted by the verifier; for instance the prover might use old system state information to cheat attestation or call out to a malicious server to forge a response.

Categorizing RA works using only their roots of trust can miss a great deal of detail. Many features of an RA scheme are independent from the root of trust or are dependent on the root of trust as well as other design choices. Several surveys have been published in recent years to study RA schemes for embedded systems. These surveys focus on specific areas of RA without considering how each aspect of RA might impact others.

RA is a new area of research and it has seen rapid growth with new breakthroughs and techniques in recent years. As a relatively new area, many terms, assumptions, and definitions have not been used or defined uniformly. Many research challenges have been overlooked and new schemes have been built on incomplete solutions. In this paper we have proposed an updated threat model and taxonomy for RA schemes and provided a survey of existing RA research works based on our threat model and taxonomy.

In this work, we have reviewed 58 research papers that present RA schemes for embedded systems from 2003 to 2020. These articles were published in journals and conferences organized by IEEE, USENIX, ACM, Springer, and Wiley. We have omitted from this review papers that provide services like secure erasure [6] and proof of secure update [7] because they do not report on the system state of an embedded system; they simply force the embedded system into a new, trusted state. We have also omitted papers like Zhao *et al.* [8] which uses RA as a black box to provide further services, and Seshadri *et al.* [9] which provides a key agreement algorithm to assist RA. Finally we have selected 58 papers for in depth review and comparisons. The following are our contributions:

- We have presented an updated taxonomy for RA in embedded systems based on the work done by Steiner *et al.* [10]. We have focused our taxonomy on the challenges presented by an RA scheme and how the solution to each challenge affects the solutions that may be chosen for other challenges. In this way, we have presented a comprehensive review of RA in embedded systems.
- We have presented an enhanced threat model that combines previous adversarial assumptions [11] with a new metric: the malware model. Our malware model measures the assumed maximum level of compromise an adversary might achieve in an RA scheme such that the security of different RA schemes might be better compared and understood.
- Using our new taxonomy and our enhanced threat model, we have classified several popular RA works. We have then used these classifications to provide an analysis of trends in modern RA research and illuminate areas of future work.
- We have provided cost-benefit analysis details of each category in our taxonomy such that security administrators may better understand the cost benefit trade off of the RA solutions discussed here in the context of their own network environments.

The remainder of this paper is organized as follows. Section II provides a brief background over RA in embedded systems and covers our motivation. Section III presents our enhanced threat model. Section IV discusses the root of trust problem. Section V covers the challenges posed by evidence in RA research. Section VI discusses the particular problems of evidence collection. A review of the evidence packaging and verification problems is presented in section VII. Scalability is discussed in section VIII. We have provided cost benefit trade-off details in section IX. We have presented areas of future research as well as a summary of our taxonomy and review in section X. Finally, we have provided our conclusion in section XI.

II. BACKGROUND AND MOTIVATION

Remote attestation (RA) schemes offer many different assurances of the prover device through many different strategies. In general, however, RA begins when the verifier sends a challenge to the prover, as seen in Fig. 1. The challenge usually includes some randomness and asks the prover to generate evidence of its innocence. The evidence depends on the level of assurance the RA scheme tries to provide.

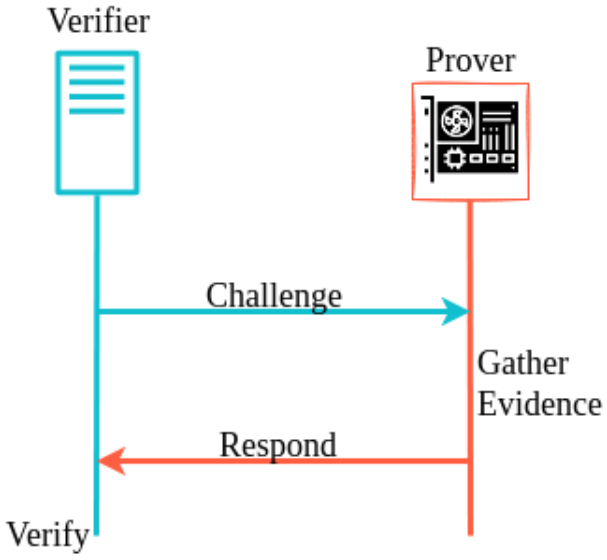

FIGURE 1. General form of remote attestation.

It could be proof that the prover's binaries have not been altered, that the boot sector has not been compromised, that no additional hardware has been added, or that no malware exists in memory. When the prover receives the challenge it computes a response and sends it back to the verifier. When the verifier receives the response, it examines the evidence to determine whether the prover has been compromised or not.

Five Major surveys have characterized the development of RA in embedded systems in recent years. In 2016, Abera *et al.* published a survey [11] over trust in the IoT setting. They provide an adversarial assumption, such that the security features of different RA schemes can easily be categorized and compared. The adversarial assumptions are as follows:

- **Remote Adversary** is able to launch remote code attacks against the prover.
- **Local Adversary** is close enough to the prover to snoop messages and interfere with communication.
- **Physically Non-Intrusive Adversary** is capable of side channel attacks against the prover.
- **Stealthy Physical Intrusive Adversary** can capture the prover and physically extract secrets from it.
- **Physically Intrusive Adversary** can modify the hardware components of the prover.

They also identify the root of trust as a major differentiation between RA schemes, and discuss the benefits and drawbacks of schemes that use secure hardware, no hardware, and hybrid methods. They then define a delineation between what they call static attestation (attesting only static parts of a prover's memory) and control flow attestation (which attests the control flow of a device). They point out that control flow attestation provides a greater security, but at the time of writing, very few attempts had been made.

In the same year, Steiner *et al.* [10] provide a novel taxonomy for RA in Wireless Sensor Networks (WSN). They identify eight categories by which an RA scheme can differ. Their "evidence acquisition" category is effectively the same as Abera's "root of trust," and the categorizations largely agree. Since Steiner *et al.* focus on RA schemes for WSN, the majority of the schemes they focus on use either a software or hybrid evidence acquisition, and as a result, the remaining 7 categories of their taxonomy largely focus on timing and memory traversal.

In 2017, Maene *et al* [12] present a review of secure hardware co-designs that can assist RA. They categorize each co-design based on security features, architectural features, and target market. They provide a summary of how each co-design functions and a comparison of each in terms of security and architectural features.

In 2018, Arias *et al.* [13] published a short survey to identify desirable qualities of an RA scheme. They redefine software and hardware attestation to refer to the form the evidence takes, not to the root of trust. They also present a new category of RA: swarm attestation, which they define to be any scheme that seeks to attest a swarm of devices rather than a single device.

In 2020, Ambrosin *et al.* [14] provide a survey over Collective RA (CRA). Although this area was mentioned by Arias *et al.* in their discussion of swarm RA, Ambrosin *et al.* discuss it in much greater detail. They begin by identifying 4 different strategies for CRA. Each strategy is either interactive or non-interactive, based on whether the verifier initiates attestation with a request. The strategies are also either self-attesting or not, based on whether the prover itself makes a judgment on its honesty based on a security architecture. They also provide an updated adversarial assumption based on the work done by Abera *et al.* [11] to include a new category of adversary proposed by Carpent *et al.* [15]. The new adversary is called the mobile software adversary, and it is capable of compromising a device and erasing all evidence of compromise. Ambrosin *et al.* then go on to present a short review of each CRA scheme discussed, and to compare them in terms of assumed adversary, network mobility, attack mitigation, and simulation parameters.

Each of these surveys provides a good review of the existing research works as well as unique insights. In general, however, these surveys do not review existing RA schemes holistically or from all angles to provide a complete understanding of RA. The adversarial assumptions provided by Abera *et al.* [11] and then modified by Carpent *et al.* [15] closely captures the behavior of an adversary, but they do not consider the behavior of a malicious prover. What capabilities might an adversary leverage through the prover once it is compromised? The Steiner taxonomy [10] provides an excellent framework to compare attestation schemes for WSN, but other applications of embedded systems are left out. WSN nodes are very computationally limited, so the scope of RA can be narrowed. The hardware review presented by Maene *et al.* [12] gives a strong understanding of how hardware security architectures compare, but do not discuss how they can be leveraged by new RA schemes and how

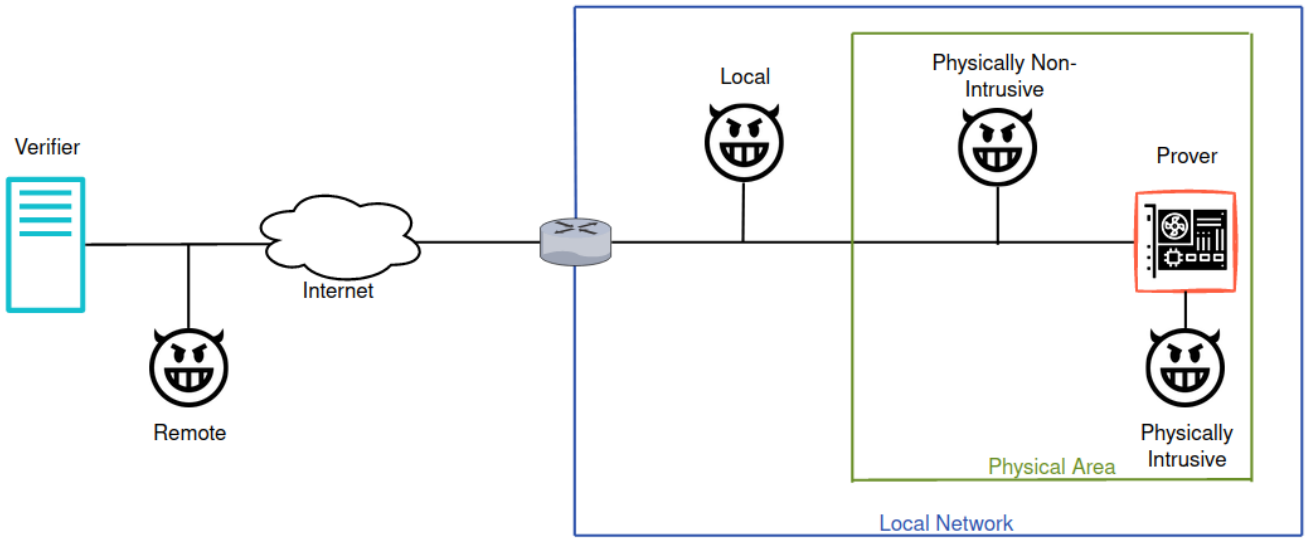

FIGURE 2. The first half of our proposed threat model: the adversarial assumption. As the adversary (marked with the devil images) gets closer to the prover, the adversary has more advantages when attacking it.

they compare to schemes that do not use hardware security architectures. Some language used by each survey (and by each of the works they review) differs substantially, so direct comparisons can be difficult. Ambrosin *et al.* [14] provide an excellent overview of CRA, but they do not discuss the implications of each design choice. How does self RA compare to a more traditional approach? How will verification differ if a non-interactive pattern is chosen? In all cases, the impact of a design choice on other design choices are left out. How does the root of trust affect evidence collection? How does evidence collection affect verification?

III. THREAT MODEL

An adversary's goal is to compromise an embedded system without being detected by the verifier. The threat models of most RA schemes are based on the adversary's capabilities in terms of control over the prover device and available attacks. We have added a new component to the threat model: the malware model. RA schemes assume that the prover may be compromised before the attestation protocol begins. As a result, threat models should assume that the prover will do everything it can to prevent detection. From this, we have concluded that the adversary's ability to communicate with the prover must be considered, but so must the prover's ability to subvert detection.

A. ADVERSARIAL ASSUMPTION

We define the adversarial assumption to be the level of access that the attacker has to the prover. We take inspiration from Abera *et al.* [11], but rather than expand their adversarial assumptions like Ambrosin *et al.* [14], we reduce them. Fig. 2 shows our adversarial assumptions and their hierarchy. It is as follows:

- **Remote Adversary** (A_R) is able to launch remote code attacks against the prover. This might be done over the internet or through a firewall. This adversary does not have any other access to the prover.
- **Local Adversary** (A_L) is on the same network as the prover, and can interfere with communications. The local adversary is capable of sniffing, spoofing, Man in the Middle (MitM), Replay, and other network based attacks.
- **Physically Non-Intrusive Adversary** (A_{PN}) is physically close to the device, but cannot interrupt its service. As a result, a physically non-intrusive adversary is capable of side channel analysis but cannot power the device off to physically tamper it.
- **Physically Intrusive Adversary** (A_{PI}) is in possession of the prover device. This adversary can power the device off and tamper with its hardware. Physically intrusive adversaries are capable of overclocking the processor, adding additional memory, and otherwise tampering with the hardware of the prover.

The adversarial assumptions are hierarchical with physically intrusive at the top and remote at the bottom. Any adversary in the hierarchy is capable of performing all the attacks of all the other adversaries who are lower in the hierarchy. For instance, the local adversary can perform all of the attacks that the remote adversary can. Similarly, the physically non-intrusive adversary can perform all attacks available to the local adversary and so on. Although the adversary model is hierarchical, many schemes do not address every aspect of a particular adversarial assumption. For instance, a scheme may address unique features of A_{PI} without considering A_R, A_L or A_{PN}. This is often because the proposed

scheme is intended to work with other RA schemes to provide a greater protection against attackers.

It is important to note that many schemes omit Denial of Service (DoS) attacks in which the adversary is in full control of the prover's network conditions and can drop any and all messages. This is often considered out of scope in RA research.

We have not considered the stealthy physically intrusive adversary in our work because its capabilities are already captured by the physically non-intrusive adversary, which can extract secrets from the prover through side-channel analysis [16], [17]. We also chose to omit the roaming adversary discussed by Carpent *et al.* [15] and Brasser *et al.* [18], because each paper proposes a different definition. Carpent *et al.* [15] define the roaming adversary to be capable of compromising a prover and removing evidence before attestation time. Brasser *et al.* [18] define the roaming adversary to be capable of performing a DoS attack by sending false attestation requests across the network. Each of these definitions is captured by previously used terminology; the former is often referred to as a Time of Check Time of Use (TOCTOU) attack (see subsection VI-A) and the latter is often referred to simply as a DoS attack.

B. MALWARE MODEL

We have included a malware model in our threat model. It is a measure of the dishonest prover's ability to subvert attestation. We consider it to be the maximum amount of compromise that an attacker can achieve within the prover device. The malware model can be measured in two ways: danger and footprint. The danger of a malware model is a qualitative measure of the malware's ability to subvert detection. The footprint is a measure of the total space in which the malware may leave evidence of compromise. No previous research works discussed here considers a malware model as part of their threat models; they only consider adversarial assumption. To the best of our knowledge this work is the first to consider malware as part of the threat model. The following is our proposed malware model:

- **Service File Malware** (M_{SF}) targets the service that runs on an embedded system by either changing the source code for the service, or by changing the settings of that service.
- **Service File-less Malware** (M_{SL}) targets a service that runs on an embedded system by infiltrating the running service in the volatile memory of the device.
- **Device File Malware** (M_{DF}) targets an embedded system directly by infecting system settings, boot files, or other system files.
- **Device File-less Malware** (M_{DL})targets an embedded system directly by infecting the device kernel, system call table, or other memory resident system resources.

There exist two major relationships between the malware models we discuss in this work. First, malware that targets a device (M_{DF} and M_{DL}) are generally more dangerous than malware that targets a service (M_{SF} and M_{SL}), because they have more control over the prover device. As an example, malware that infects the kernel will be better able to avoid detection than malware that infects a single process, because the kernel malware can better interfere with evidence collection. Second, fileless malware (M_{SL} and M_{DL}) is more difficult to detect than file malware (M_{SF} and M_{DF}), because no evidence of compromise exists outside of RAM and it cannot be detected with traditional computer forensics. When the device is powered off, fileless malware disappears. We consider fileless malware to be more dangerous and to have a smaller footprint than file malware in general. Despite these two relationships, it is difficult to determine a strict hierarchy between malware models like the adversarial assumption. Fig. 3 shows the relationships we have discussed here, but is not intended to be treated as a strict hierarchy.

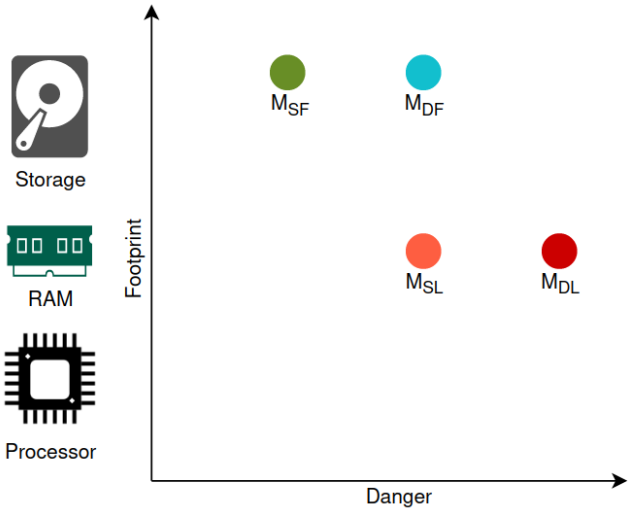

FIGURE 3. Relationship between footprint and danger in malware model.

In the following sections, we will discuss how the design choices for each category affect the threat model. In each case, we will discuss the attacks that are available to the assumed malware model and threat model.

IV. ROOT OF TRUST

A root of trust is the foundation for a Remote Attestation (RA) scheme. The root of trust is defined as what component of the prover (hardware, software or both) is used for gathering evidence. This component of the prover is trusted by the verifier. Without a root of trust, a prover could forge evidence or provide evidence that was generated by another device or network entity.

There are three major types of roots of trust in RA schemes: software, hardware, and hybrid. These terms are sometimes inconsistently used in literature. In this work, we take the definition for software root of trust from Steiner *et al.* [10] as any root of trust that does not rely on any additional hardware. Software roots of trust are generally more flexible than hardware or hybrid. However, software roots of trust usually make

strict assumptions such as predictable network conditions, or provers' operation environments that may not hold under realistic circumstances. In many works, hardware roots of trust are defined as using exclusively hardware to provide assurance over an embedded system. In this paper we define a hardware root of trust as any purpose build hardware such as redesigned processor, security co-processors, or accelerator in the system that gathers and provides evidence to the verifier for RA. Finally the hybrid root of trust is typically defined as using a combination of hardware and software features. We narrow this definition to be any root of trust that uses specific hardware features that are already available on certain embedded devices. This would include pre-existing Trusted Execution Environments (TEEs), such as the Intel SGX or the Trusted Computer Group's Trusted Platform Module (TPM) as well as Memory Protection Units (MPUs), write protected clocks, and ROM. We take the definition of a TEE as a tamper resistant processing environment that runs on a separation kernel [19].

A. SOFTWARE

The earliest root of trust for RA is software based. Early works such as Spinellis *et al.* [20] investigate a computer's ability to scan itself for malware. Software schemes do not require any additional hardware to establish a root of trust. If a software based system is compromised, it must be assumed that the adversary is in control of every secret, peripheral, and capability that the system possesses. As a result, any evidence that is computed using techniques such as cryptography, immutable execution, or atomic execution cannot be trusted by the verifier. To counteract this, software based RA schemes provide assurance over the embedded system in one of three ways. The first is virtualization with a hypervisor [21]. The embedded system virtualizes its OS using a secure hypervisor that intervenes between the OS and the hardware of the prover device. The hypervisor is able to ensure that evidence is collected honestly by monitoring the OS as it runs and by collecting information that would normally not be available. Virtualization is a popular choice in traditional computing systems because of these security features [22], but the additional latency and computational cost may violate the real time response requirement of many embedded systems, which usually have low computing capabilities. It is also important to note that virtualization is usually recommended with a TEE such as the TPM to allow the use of cryptographic keys and enhanced security [23].

The second method to provide a software root of trust is filling excess memory with random noise [24]–[28]. These schemes operate under the assumption that an attacker will need to either modify the embedded system's original program or upload a new one to compromise a device. To defend the device, these schemes use a pseudo random number generator to fill all unused program memory in the embedded system with incompressable noise that is only predictable by the verifier. This way, when the embedded system's memory is scanned it will directly match the verifier's expectation. Any alteration to the program memory will force the embedded system to erase some of the noise. Even after the adversary has compromised the device, they will not be able to return the original noise to the program memory, thereby leaving evidence of compromise and causing verification to fail. It is important to note that programs in memory can be compressed even if the noise cannot. Attackers may be able to leverage this to create extra space to stage malware in memory. Vetter *et al.* [24] provide a method to compress programs before they are loaded into an embedded system's program memory such that attackers cannot leverage the extra space.

The third software root of trust mechanism is timing. Timing based schemes use side channel information about the prover device to place tight timing constraints on its attestation response [29]–[34]. The verifier is assumed to know the exact latency of the prover's evidence gathering algorithm and the exact Round Trip Time (RTT) of the attestation request. If the prover takes too long or too short to respond, then it may have forged a response or received assistance from a malicious server. If the prover responds within an acceptable time frame then the verifier trusts that the evidence was collected legitimately. Equation 1 formalizes this, where $E(T_R)$ is the expected response time of the prover, T_E is the execution time of the prover's evidence gathering algorithm, and T_{RT} is the round trip time of a message between the verifier and the prover.

$$E(T_R) = T_E + T_{RT} \tag{1}$$

One of the weaknesses of software roots of trust is their assumptions, which often do not hold true in real systems and operating environments. Hypervisor schemes can provide a great deal of security if the additional computation and latency are tolerable on the embedded system in question and if the malware does not employ any anti-virtualization [35] or anti-sand-boxing methods [36]. Hypervisors that do not use a TEE are unable to leverage cryptography to provide a greater assurance that the hypervisor has not failed. Random noise schemes can prevent the attacker from loading new malware into program memory or altering existing program memory, but only if the task assigned to the prover is very narrow in scope. In random noise based RA schemes, any change in memory will cause the verification to fail. However, modern embedded systems have processes that may change memory while operating normally. This can cause the verifier to falsely assume the prover is guilty. Further, compressing programs for an embedded system is non-trivial [37] so a poor choice of compression algorithm by the embedded system designer could allow additional space for an adversary to launch an attack. Timing schemes are able to leverage trust over the entire system, but they rely on predictability in T_E and T_{RT}. Any way that an attacker can reduce T_E or T_{RT} can help them subvert detection. Several attacks have been proposed against timing based software roots of trust toward these ends. A listing of these attacks are as follows:

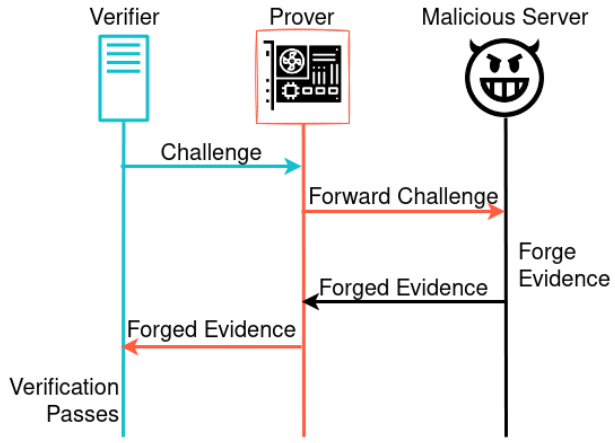

FIGURE 4. An example of a proxy attack. The compromised prover forwards the challenge to a malicious server who forges evidence of innocence.

- **Proxy** (M_{DF} **or** M_{DL}) [10] - When a compromised prover receives a challenge from the verifier, it calls out to a powerful malicious server with the challenge. The malicious server then forges part or all of the response faster than the compromised prover would have, and sends it to the compromised prover who forwards it back to the verifier within the acceptable time frame. An example proxy attack is shown in Fig. 4.
- **Precomputation** (M_{DF} **or** M_{DL}) [29] - If no randomness is included in the challenge, or if the randomness is predictable to the adversary, then a compromised prover can precompute part or all of a response.
- **Overclocking** (A_{PI}) [33] - If the adversary is physical, then they may be able to speed up the clock rate of the prover. This allows the prover additional clock cycles to forge evidence during attestation.
- **Optimization** (M_{DF} **or** M_{DL}) [33] - If the evidence gathering algorithm is not carefully compiled, it may be possible to optimize it. This can be done through multi-processing, multi-threading, or more efficient compilation. In this way, a prover might be able to leverage additional clock cycles to subvert the RA scheme.

Each of the above attacks can be defended through good cyber security practices such as including randomness in the challenge, restricting communications to the prover, careful compilation of the evidence gathering algorithm, and restricting physical access to the prover device. Under realistic network settings, however, the adversary may not need to perform any of these attacks to subvert a timing based software root of trust.

Network jitter is defined as a variation in delay of received packets in a network. More formally, we can say that network jitter is an unpredictable change to T_{RT}. Network jitter can be caused by network congestion or unexpected changes in network traffic and it is notoriously difficult to predict and counteract [38]. Many RA schemes have attempted to work around network jitter by either accurately predicting it [32] or by using relay nodes to record the attestation delay at each hop [30], [33].

B. HARDWARE

Hardware roots of trust are introduced by researchers to address the shortcomings of software roots of trust. Hardware roots of trust provide assurance over the prover by introducing additional hardware. New hardware can come in the form of a single purpose processor [39]–[46], novel MPUs [47]–[53], or Physically Unclonable Function (PUF) [54], [55]. Single purpose processor designs and novel MPUs share the goal of isolating software and resources needed for attestation from the rest of the processor. PUF designs are intended for the more narrow focus of binding an attestation report with the device that generated it, thereby defending against the proxy attack discussed in subsection IV-A.

It is important to note that some TEEs can provide an MPU as well as Local Attestation (LA); they can act as the verifier directly and report on the condition of the prover in a binary value. Maene *et al.* [12] provides a detailed review that covers the operation of these TEEs. For the purposes of our review, we include TEEs that provides Local Attestation (LA) in hardware roots of trust.

Hardware based RA schemes like Trustlite [48] and SMART [47] are purpose built and have become very popular in RA schemes because they are isolated, cannot be modified easily, are close to the processor, and can access the processor directly. Single purpose processor schemes and novel MPU schemes can provide immutability and atomicity of the evidence gathering algorithm. They can also provide access to cryptographic functions. These features make them immune to the attacks discussed in section IV-A. Single purpose processor schemes are able to provide very strong security because they expect to make invasive changes to the prover device. For instance, Abera *et al.* [39] propose a change to the processor of the prover device, such that information can be gleaned from the instruction pipeline.

While hardware roots of trust offer several benefits over other roots of trust, many of the purpose built hardware for RA proposed by researchers are not available as commodity products or have not adopted by manufacturers. As a result they are unlikely to be available in production embedded systems in the near future, although manufacturers have begun to offer their own devices that provide RA [56].

Embedded systems that are already in operation will not be able to take advantage of any new hardware root of trust. This could incur large costs for system administrators who have to replace several systems across their networks. Since hardware roots of trust intend to be implemented either in the motherboard or in the CPU of an embedded system, retro fitting existing embedded systems with the needed hardware may be cost prohibitive or nearly impossible.

One major problem with hardware roots of trust is that by design they cannot be updated with newer modified version. If a single purpose processor's algorithm is ever subverted or if an MPU's isolation is violated, then the entire prover

may be rendered obsolete. Attacks like Meltdown [57] and Spectre [58] show that even when security is built into the hardware of a device it can still be breached. If a hardware root of trust avoids subversion by a direct attack, the passage of time will advance hardware sophistication to the point that security primitives like cryptographic schemes might be easily broken. In any case, relying on a hardware root of trust may cause routine replacement of any and all prover devices.

C. HYBRID

Hybrid roots of trust are often defined as using hardware and software to accomplish assurances over the prover. We narrow this definition to be any root of trust that uses a pre-existing hardware features or easily pluggable hardware feature that are provided by manufacturers. Under our definition, any RA scheme that uses the minimal hardware features [59]–[67] discussed by Francillon *et al.* [68], or the isolation and MPU features of a TEE [15], [61], [69]–[84] would be classified as hybrid, even though the TEE itself is classified as a hardware solution. We make this distinction because it underscores the fact that hybrid schemes do not call for changes to a prover device but rather limit their scope to a specific class of embedded systems that are already in production.

Hybrid schemes can tap into desirable features of both hardware and software roots of trust. They have access to process immutability and atomicity. Cryptography can be used to validate attestation requests and a verifier can know that the prover's TEE ensures accurate evidence collection. These features indicate that hybrid roots of trust are not as limited in their assumptions as software schemes are. Hybrid schemes are more amenable to upgrade and modification than hardware schemes. ROM can be exchanged and secrets in MPUs can be re-written in the case that a vulnerability is discovered. Security administrators need not replace embedded systems as often in a hybrid scheme as they would in a hardware scheme.

While they inherit benefits of both hardware and software based schemes, hybrid schemes also inherit drawbacks of each. Hybrid schemes are able to leverage more resources than software based schemes, but they do not have the same level of access that hardware based schemes do. TEEs are also reliant on individual manufacturers in the same way that hardware based schemes are, and each manufacturer has developed TEEs with different goals and methods. Significant revisions must be made if an RA scheme will be used across several TEEs. Manufacturers, and by extension TEEs, also vary widely in transparency. For instance, SANCUS [44] is an open source TEE, while the Intel SGX [45] is not. It may be difficult to design an RA scheme for a TEE if information is not available outside of the manufacturing company. Another major limitation for hybrid roots of trust is the inability for the TEE to be updated. While the software deployed on the TEE can be updated, the TEE itself cannot. If a vulnerability in the TEE's design is exploited, then RA schemes could break down and network administrators could be forced to replace embedded systems. Examples of such exploits are given by Rosenberg *et al.* [85], Weichbrodt *et al.* [86], and Van Bulck *et al.* [16] against the Trustzone [87] and the Intel SGX [45] respectively.

V. EVIDENCE

Evidence is at the heart of any Remote Attestation (RA) scheme. There are many types of information that can be considered as evidence to prove the trustworthiness of the prover. Some example are system settings, binaries, and main memory contents. Whatever information is to be considered as evidence by the verifier depends on what threat model has been chosen. Evidence is often heavily influenced by the root of trust, but it should be considered separately from the root of trust because some attacks may compromise evidence only. As a result, a good RA scheme should consider both root of trust and evidence together.

Two major categories of evidence exist: static and dynamic. These terms have been used inconsistently by researchers. A static RA scheme can sometimes refer to a scheme that collects information that usually does not change over the life time of the prover (e.g. the boot sector of the hard drive). It can also refer to a scheme that collects information about the prover at a single moment in time. Likewise, the term dynamic RA can refer to a scheme that collects evidence that changes over time, or a scheme that collects evidence from a prover over a period of time. In this work, we use the terms static and dynamic to refer to the kind of evidence that is being collected, not how the evidence was collected. A static RA scheme collects information that does not change and a dynamic RA scheme collects information that does change over time. Examples of static evidence include the boot sector of memory, executable binary files, software configurations, and information about hardware components. Examples of dynamic evidence include RAM contents, running processes information, the contents of the instruction cache (I-cache) and the contents of the data cache (D-cache). We also note that device mobility can be classified as either static or dynamic. Node mobility is further discussed in section VIII.

Static evidence is able to detect compromise in the M_{SF} and M_{DF} malware models, because evidence of compromise exists in the storage of the prover. Likewise, Dynamic evidence is able to detect compromise in the M_{SL} and M_{DL} models. Static evidence is easy to collect, because it does not change and it is usually readily available to kernel space programs. Any root of trust can support the collection of static evidence, but many hardware and hybrid roots of trust have the advantage of Local Attestation (LA) compared to software roots of trust. If LA is available, a verifier can simply ask the prover whether attestation passes or fails. As long as the root of trust is not compromised, the verifier can trust the prover to attest itself.

Although it is easier to collect, schemes that commit to static evidence [31], [44], [45], [49]–[53], [66], [69], [71]–[74], [77]–[81], [88] lose a great deal of granularity in

their verification. If settings have been changed, or if a binary has been altered, static evidence will detect compromise. However, if malware infects a different file which was not part evidence collection, or if it infects a memory resident resource like the kernel or system call table, static evidence will not detect the compromise.

Dynamic evidence, on the other hand, can be very difficult to collect. Some dynamic evidence like cache contents can only be collected by hardware roots of trust. This is because some hardware root of trust schemes are capable of making a cache or buffer directly available to the security co-design [89]. Other dynamic evidence like memory contents are easier to collect, so any root of trust is able to collect them. Despite the difficulty of collection, dynamic evidence provides a more comprehensive picture of the state of a prover compared to static evidence [15], [21], [24]–[30], [32]–[34], [39]–[43], [46], [47], [54], [55], [59]–[61], [63]–[65], [70], [75], [76], [82]–[84], [90], [91]. Since dynamic evidence is usually collected from system resources that see all programs or instructions, it is theoretically able to capture evidence of any malware. All instructions pass through the cache and through program memory, so if evidence is collected from these resources, malware is much more likely to be detected.

VI. EVIDENCE COLLECTION

For a given threat model, choosing what evidence to collect alone is not sufficient, one needs to consider how that evidence is collected. If evidence is not collected securely, details may be missed and a malware may subvert detection. Evidence collection strategies can be broken into two major categories: discrete and continuous. Discrete RA schemes collect evidence from a prover at a particular moment in time. Continuous RA schemes collect evidence over a time period to provide more information and context. Sometimes these are referred to as static and dynamic in literature.

A. DISCRETE

Discrete RA schemes are well suited to static evidence, because no extra precautions need to be taken. If a M_{SF} malware model is chosen, then the evidence collection algorithm can simply scan the suspicious binary, file, or configuration and return the results to the verifier. If a M_{DF} malware model is chosen and if the device is compromised, then the verifier cannot rely on the discrete collection of static evidence. The adversary may be in control of the prover altogether, so the prover may attempt to subvert detection. Unless atomicity and immutability are guaranteed by a hybrid or hardware root of trust, several attacks can be performed against discrete evidence collection method.

On the other hand, dynamic evidence poses a problem for discrete collection strategies. For example if a RA scheme is scanning memory for evidence collection, it must choose an order to traverse the memory (a memory walk strategy). Since the evidence is constantly changing, it can be difficult to collect enough information to give an accurate picture of the state of the prover. To complicate the problem, prover

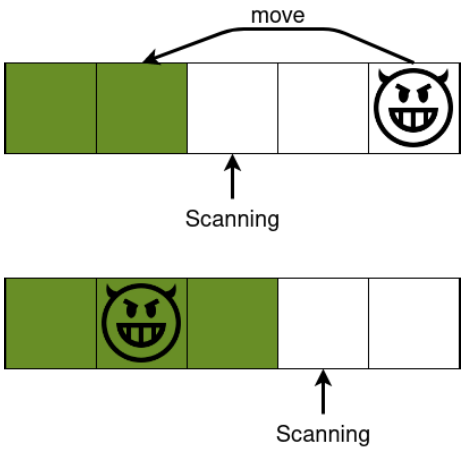

FIGURE 5. Example of a memory copy (File-less) attack. The prover scans memory linearly. Clean sectors are marked green. The malware identifies a sector that has already been scanned, and moves into it before its original memory sector is scanned.

memory is relatively large when compared to the size of the malware infection. Collecting enough memory can be difficult without scanning an entire device. To complicate the problem further, malware can employ intelligent strategies to avoid detection by hopping into different memory locations during evidence collection.

Many attacks against discrete memory collection have been proposed in recent literature [10], [32], [92]. The following are some reported examples:

- **Memory Copy: File** (M_{DL}) [93] - A dishonest prover saves a valid copy of the file to be scanned before it is altered. When an attestation request is made, the dishonest prover scans the honest version of the file rather than the copy that is running. When evidence is returned to the verifier, no malware is detected.
- **Memory Copy: Fileless** (M_{DL}) [29] - If the memory walk strategy is known, (either no randomness is used, or the randomness is too predictable) then the compromised prover may move malware from one area of memory that has not been scanned to another area that will not be scanned, or that has already been scanned. An example memory copy attack is shown in Fig. 5. As the prover's memory size increases, it becomes easier for the prover to move malware out of the way in time. Common defenses include filling all unused space in memory with random noise chosen by the verifier, or enforcing a random memory walk strategy. In the either case, careful consideration must be given to the memory walk strategy to make sure that a dishonest prover has little to no statistical chance of avoiding detection.
- **Compression** (M_{DL} **or** M_{SL}) [94] - Compression algorithms can be used to reduce the size of malware, legitimate processes, and even the evidence gathering algorithm of the prover. By using compression on any of these elements, a compromised prover may be able

to hide malware in a memory sector that will not be checked by the memory walk strategy. This attack can also be used to foil the random noise defense discussed in the previous attack. To prevent this attack, memory walk strategies should consider all areas of memory, and all legitimate code should already be compressed in RAM.

- **Split Translation Lookaside Buffer (TLB)** (M_{DL}) [92] - The TLB is part of the memory management unit, and is used to reduce the latency of acquiring data from memory. A compromised prover can split the TLB such that the correct instructions are run on the wrong parts of memory. By doing this, a compromised prover can alter the memory walk strategy without modifying the evidence gathering algorithm. A split TLB attack can be defended by forcing the evidence gathering algorithm to include information about execution, such as the program counter.
- **I-Cache Inconsistency** (M_{DL} **or** M_{SL}) [92] - Different processors have different degrees of cache consistency. Unfortunately, many embedded systems feature low end processors that do not guarantee any cache consistency. Beyond the negative implications on performance, this also means that a compromised prover may exploit an I-Cache inconsistency to subvert a memory walk. In short, if the malware is small enough, the prover can move it into the I-cache and delete it from memory. By doing this, malware may still be resident on the prover while avoiding detection. I-Cache inconsistency attacks can only be prevented by either disabling the I-cache altogether or upgrading to processors that guarantee some level of cache consistency.
- **Time of Check Time of Use (TOCTOU)** (M_{DL} **or** M_{DF}) [42] - Sometimes called future posted event attacks [92] or mobile malware [95], TOCTOU attacks work by predicting the time of a challenge and avoiding it. If a compromised prover can predict when a challenge will come (or if an exploit can be found in a timing based software root of trust) then the prover can delete any evidence before the challenge arrives. It then performs all of the expected steps and delivers proof of innocence. Once attestation has finished, the prover is re-infected. Future posted events, watch dog timers, and cron jobs are often used for this attack. The only defense against a TOCTOU attack are to either 1) use a continuous attestation scheme, or 2) keep the challenge time random and secret. The second option is particularly week in the case of a timing based software root of trust because of network jitter.
- **Return Oriented Programming (ROP)** (M_{DL} **or** M_{SL}) [96], [97] - ROP attacks exploit a pre-existing bug in a program to seize control flow. Typically an attacker would use this control to execute useful section of code (usually called gadgets) out of order. In this way, an attacker can perform totally legitimate operations in a totally legitimate executable file and still accomplish a compromise. In RA schemes, ROP attacks could be used to exploit a prover device while leaving no evidence for an RA scheme to detect. To defend against a ROP attack, evidence gathering algorithms should be designed and compiled with security in mind. For example, stack execution should be disabled in the compiler and no instructions with known weaknesses should be used.
- **Data Oriented Programming (DOP)** (M_{DL} **or** M_{SL}) [98] - DOP attacks identify critical variables to allow them to seize control of the prover's behavior. DOP attacks do not create an invalid control flow. Attackers may accomplish this by changing the value of a pointer to cause an incorrect, valid control flow through the prover's service. Hu *et al.* [98] show that DOP attacks can leverage Turing complete control over a device.

To defend against the above attacks, careful planning should be used in a discrete RA scheme. If the prover does not have tight real time constraints, a hardware or hybrid root of trust can stop normal operation and atomically execute the evidence gathering algorithm. If an RA scheme does not have a hardware or hybrid root of trust, randomness should be used in the memory walk strategy and challenge times. Evidence gathering algorithms should be written and compiled in such way that they cannot be compressed or optimized further so that the adversary cannot hide malware by reducing the size or the execution time of the evidence gathering algorithm.

B. CONTINUOUS

In contrast to discrete RA schemes, continuous RA schemes collect evidence from the prover device over a period of time. Continuous RA schemes are usually intended to work along with a static RA scheme to provide a greater security against ROP and DOP attacks. In general, these schemes monitor the behavior of a single service running on a prover device and report evidence if the service misbehaves. The earliest example in our review is a software based scheme by Zhang *et al.* [26]. Their proposed scheme places a unique cryptographic hash before and after every variable used by the service. If an attacker tries to overflow one of these variables, a hash will be destroyed. Since all hashes are known to the verifier, a memory scan will result in a failed attestation. The work done by Zhang *et al.* [26] uses a software root of trust. In contrast, most other schemes use a hardware or hybrid root of trust.

In general, continuous RA schemes collect evidence through the use of an intervening component like a Trusted Execution Environment (TEE) [39], [82] or an invasive change to the CPU of the prover [40], [41]. With the exception of Wang *et al.* [91] who provide a method for the Linux kernel to measure software behavior in a graph, these schemes measure the control flow of a single service running on a bare metal embedded system. The control flow of a service is defined as the order that instructions are processed. Control Flow Integrity (CFI) tools provide insight as to whether an invalid control flow (ROP attack) occurred [99]–[102], but

they lack context. No information about which control flow path was taken can be ascertained and CFI gives no defense against DOP attacks.

The first control flow RA scheme was proposed by Abera *et al.* [39] in C-FLAT. They provide a method to instrument the service such that whenever a control flow statement is reached, instead of executing it immediately, the Arm Trustzone intervenes and records the control flow statement in an isolated buffer. All control flow instructions are added to an aggregate hash that creates a fingerprint of the running service's control flow. Before attestation, the verifier generates a graph of all valid control flows through the service. During attestation, it is then able to use this fingerprint to determine if the measured control flow is or is not valid. Instrumentation of a service is difficult, however, because determining an accurate control flow graph is difficult for non-trivial programs. Abera *et al.* [39] note that latency is a concern with their scheme, since the Trustzone must intervene each time a control flow statement is reached. They argue that their scheme can mitigate this problem by only attesting the service when the verifier sends a request.

To mitigate these issues, Dessouky *et al.* [40] and Dessouky *et al.* [41] provide a hardware based RA scheme where the co-processor is able to collect control flow statements in parallel with the CPU as it executes them. The co-processor can access instructions as they enter the instruction pipeline of the CPU. Dessouky *et al.* [40] extend the functionality of their scheme by recording metadata from service execution such that the verifier may analyze results with more context. Dessouky *et al.* [41] then extend this work further by adding a data buffer to the hardware co-design such that DOP attacks can be defended. Since these schemes both use hardware co-designs to collect control flow statements, they do not require any code instrumentation. However, this type of co-design is difficult to implement, costly to manufacture, and far away from wide adoption in commodity devices.

Sun *et al.* [82] are similar to Abera *et al.* [39] in the sense that they use the Trustzone in a very similar way. They extend the functionality of their design to what they call Operation Execution Integrity (OEI). They do this by also instrumenting the prover's service to identify critical data variables. They then monitor the values of these data variables by ensuring that they do not change between known definitions and uses. This provides a guard against DOP attacks.

VII. EVIDENCE PACKAGING AND VERIFICATION

We define evidence packaging as the process of encoding or encrypting evidence for transfer between the prover and the verifier. Usually, a prover generates a message digest hash of the evidence and the verifier checks the hash [15], [21], [24]–[34], [44]–[47], [49]–[55], [59]–[61], [63]–[66], [70]–[81], [84], [88], [90]. Even if the collected evidence contains a lot of details, a hash cannot be analyzed. Any small change to the message creates a completely different hash. This can lead to unwarranted investigations into a prover device and service downtime. Evidence packaging has not been investigated rigorously by researchers. With the exception of some continuous RA schemes, no additional details are given outside of a cryptographic hash of the collected evidence.

Hash functions are excellent for packaging static evidence, because only one hash is needed per file. Verification is easy, because the verifier need only compare the evidence provided by the prover to a stored hash value. If any difference is detected, attestation fails. Computing and comparing hash values is computationally inexpensive.

Dynamic evidence is much more difficult to verify using a hash function. If evidence is collected from a complex, running service, there may be too many hashes to store and compare efficiently. To worsen the problem, hash functions usually employ diffusion and pre-image resistance [103]. Intelligent analysis of hash values is impossible, so identifying a hash of dynamic evidence efficiently is an open problem. Some RA schemes propose to use a Merkel Tree [104] to store related hashes such that finding valid hashes is simpler.

Continuous RA schemes are able to provide more context about execution than static schemes over dynamic evidence, but they still do not provide evidence to the verifier in a way that allows detailed analysis. Evidence is still stored in a cumulative hash function, so the verifier must keep track of all possible valid system state hashes. Dessouky *et al.* [41], Dessouky *et al.* [40], Sun *et al.* [82], and Agarwal *et al.* [21] are all able to somewhat reduce the number of possible evidence hashes that must be stored in the prover with contextual evidence, however.

VIII. SCALABILITY

Scalability is a measure of how efficiently an RA scheme can attest a group of embedded systems as the number of embedded systems increases. Many system employ embedded devices in large numbers. For instance, a SCADA network might employs tens of PLC devices and hundreds of sensors. Attesting each device in such a network can be a very time consuming process. Schemes that propose new ways to attest groups of devices are often referred to as Collective Remote Attestation (CRA) schemes. CRA schemes can be grouped into two categories: many-to-one [15], [34], [43], [46], [59]–[64], [66], [70], [72], [75], [80], [88], [105] and many-to-many [67], [69], [71], [73], [74], [76], [78], [79], [84]. Examples of each category can be seen side-by-side with a one-to-one example in Fig. 6. Many-to-one schemes provide a mechanism for several embedded devices to attest themselves to a single verifier while many-to-many schemes give a mechanism for embedded devices to attest themselves to each other. The scalability of a CRA scheme is affected by the root of trust, the evidence, evidence gathering, and verification.

Unlike previous design choices, CRA schemes are also affected by 2 new design choices: network mobility and homogeneity. The network mobility is often defined as the degree to which individual nodes are allowed to move in the

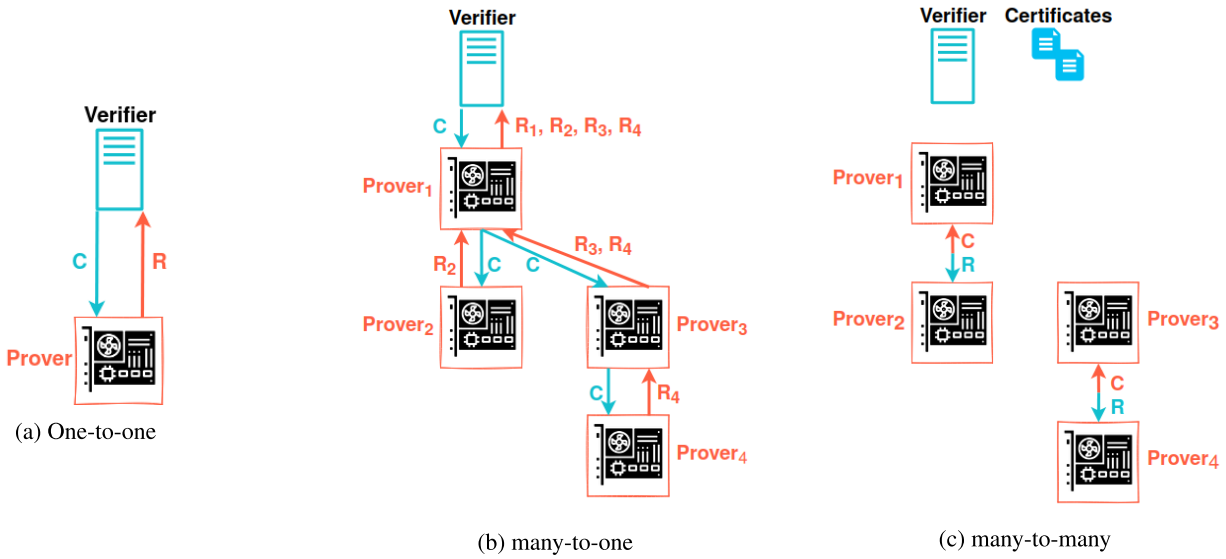

FIGURE 6. A side by side comparison of (a) one-to-one, (b) many-to-one, and (c) many-to-many. (b) is an example of a spanning tree CRA scheme, which propagates a single challenge from the verifier to each prover in the network. Each prover waits until its children have responded to the challenge before adding their own response, and sending it up the tree. (c) is an example of a single step of a peer verification CRA scheme. Each prover challenges a neighbor, and compares the results against a certificate published by the verifier.

network. Three types of network mobility have been reported in CRA literature: static, quasi-static, and dynamic. Static networks do not allow for any degree of movement of devices at all [43], [46], [61], [73], [79], [80], [84]. In quasi-static networks devices can move within the network but they do not leave the network during attestation [15], [34], [60], [63], [67], [70], [72], [75], [76], [88]. Examples might be herd mobility (in which case the entire network of provers moves together) and slow mobility (in which case devices move slowly enough as to not leave the network during attestation). In dynamic networks devices are highly mobile and changing [59], [62], [64], [66], [69], [71], [74], [78], [105]. Devices are assumed to leave and re-enter the network at any time.

Homogeneity is a measure of a system's ability to include prover devices of different classes and software states. In this work, we consider two cases: a system can be homogeneous or heterogeneous. Homogeneous networks are made up of only a single class of device with a single service and software states [34], [64], [73]–[76], [79], [79], [84]. Wireless Sensor Networks are often considered to be homogeneous. By contrast, heterogeneous networks are made up several different devices with different services and software states [15], [43], [46], [59]–[63], [66], [69]–[72], [78], [80], [105], [105]. Examples include smart homes and SCADA systems.

A. MANY-TO-ONE

Many-to-one schemes can attest either the device (M_{DF} or M_{DL}) or the service running on the device (M_{SF} or M_{SL}) just as 1-to-1 schemes can. Many-to-one schemes that assume a M_{DF} or M_{DL} malware model apply a variety of mechanisms to provide better scalability. If the network is assumed to be either static or quasi-static, then a spanning tree mechanism can be used. In spanning tree CRA schemes, the verifier sends a challenge to the highest node in the network. That node propagates the challenge to its children, who propagate the challenge down until it reaches the leaf nodes who calculate a response. When each node receives a response from its children, it aggregates those responses with its own and passes it to its parent. The verifier then receives an aggregate response over the whole network [15], [64], [70], [75], [88].

Another many-to-one scheme that has been proposed by researchers [34], [63], [66], [80], [105] allows network entities that are closer to the verifier to be treated with more trust. These devices either have a stronger root of trust than the nodes which are more distant, or they are attested before the node that are more than one hop away. In either case, these closer devices are used to measure the attestation responses of distant nodes. Once the far away nodes respond to attestation, the devices that issued the challenge aggregate the responses and send them to the verifier. We refer to these schemes as hop-by-hop attestation.

Rather than use intermediate nodes, Aggregation CRA schemes [72], [78] define a new cryptographic primitive to securely and efficiently aggregate the responses of each device in the network. Similarly, some CRA schemes use a secure broadcast mechanism that enables responses to be aggregated while also allowing the verifier to see individual responses [59], [60]. It is important to note that aggregation and secure broadcast are used in both many-to-one and many-to-many settings.

Some many-to-one CRA schemes adopt a M_{SL} malware model [43], [46] and provide service control flow across several embedded systems. These schemes extend the control

flow attestation schemes discussed in section VI, by building a control flow graph for a service running across several devices. These schemes include the messages sent from one device to another in the control flow graph, such that attacks can be traced to the specific device that was first compromised.

Many-to-one schemes that use a A_{PI} adversarial assumption generally employ a heartbeat mechanism to detect device absence [59], [62], [67], [76], [88], [105]. The rationale is that a physically intrusive adversary would need to power the prover device off for a relatively long period of time to perform their attack. The heartbeat mechanism sends messages out at regular intervals. If any device misses a message, then it is assumed that that device was offline, and could have fallen victim to a physically intrusive adversary. Heartbeat mechanisms are only effective in a static or quasi-static network model and can be applied in either a many-to-one or many-to-many setting.

B. MANY-TO-MANY

In consensus based schemes, provers must come to a consensus of which devices failed attestation [69], [73]. In group signature schemes, the TEE uses the attestation report and a secret key to calculate a group signature [71]. If the device's evidence matches other evidence in the network, it will reduce. If it does not, the signature becomes longer, indicating that the device failed attestation.

If the network can be assumed to be either homogeneous or mostly homogeneous, then peer verification can be used [74], [76], [79], [84]. In this case, provers directly attest themselves to each other. Rather than storing valid responses on each prover, many peer verification schemes use a centralized verifier to publish certificates that bind valid evidences to the device's class.

IX. COST BENEFIT ANALYSIS

To provide security administrators with the details they need to perform a cost benefit analysis of RA schemes in their own networks, we have investigated the costs and benefits of root of trust, evidence, evidence collections, verification, and scalability based on four measures: attack resistance, complexity, compatibility, and financial cost. For each of these measure we have assigned a score of either high, medium, or low depending on the goodness of a particular scheme/method. These evaluations are qualitative and subjective in nature because an objective quantitative measure would require a large scale test of several different embedded devices in different network configurations. Such a study is out of scope for this work.

We define the attack resistance of an RA scheme to be its ability to resist attacks from a variety of threat models. Complexity is measure of difficulty in terms of both setting up and operating the RA scheme. For example a whitelist approach of verification scored high in complexity because it is very difficult to identify all possible system states of a prover device, and it is computationally difficult to check each state against provided evidence. We use the term compatibility to capture how easy or difficult it is to deploy a scheme on multiple different embedded systems from different manufacturers. Finally, the last measure is financial cost which indicate how expensive a scheme is. Tables 1 through 5 show the cost benefit trade-off for root of trust, evidence, evidence gathering, verification, and scalability respectively.

TABLE 1. Cost benefit details of root of trust.

Root of Trust	Attack Resistance	Complexity	Compatibility	Financial Cost
Software	Low	Medium	High	Low
Hardware	High	Low	Low	High
Hybrid	Medium	Medium	Medium	Medium

TABLE 2. Cost benefit details of evidence.

Evidence	Attack Resistance	Complexity	Compatibility	Financial Cost
Static	Low	Low	High	Low
Dynamic	Medium	Medium	High	Medium

TABLE 3. Cost benefit details of evidence gathering.

Evidence Gathering	Attack Resistance	Complexity	Compatibility	Financial Cost
Discrete	Low	Medium	High	Low
Continuous	High	High	Low	High

TABLE 4. Cost benefit details of verification.

Verification	Attack Resistance	Complexity	Compatibility	Financial Cost
Whitelist	Low	High	High	Low
Control Flow Graph	High	High	Low	Medium
Data Flow Graph	High	High	Low	High

TABLE 5. Cost benefit details of scalability.

Scalability	Attack Resistance	Complexity	Compatibility	Financial Cost
One-to-One	High	High	Medium	High
Many-to-One	Medium	Medium	Medium	Medium
Many-to-Many	High	Medium	Low	High

In Table 1 we scored hardware root of trust high in attack resistance because it is immune to many attacks that target software based schemes, and because in some cases it can leverage access to the prover's CPU cache and busses. We have given it low in compatibility, high in financial cost, and low in complexity because, when compared with other roots of trust, it will only be available on certain devices,

TABLE 6. Our applied RA taxonomy. It is important to note that not every RA or CRA scheme addresses every part of our proposed taxonomy. Any scheme that does not consider a particular problem has that field marked N.A.

	Root of Trust	Root of Trust Mechanism	Attestation Type	Evidence	Gathering	Verification	Scalability
Carpent et al. [15]	Hybrid	SMART	Local	Dynamic	Discrete	Whitelist	Many-to-One
Agarwal et al. [21]	Software	Hypervisor	Remote	Dynamic	Discrete	Whitelist	One-to-One
Vetter et al. [24]	Hybrid	Random Noise	Remote	Dynamic	Discrete	Whitelist	One-to-One
Shaneck et al. [25]	Software	Random Noise	Remote	Dynamic	Discrete	Whitelist	One-to-One
Zhang et al. [26]	Software	Random Noise	Remote	Dynamic	Continuous	Whitelist	One-to-One
Abuhmed et al. [27]	Software	Random Noise	Remote	Dynamic	Discrete	Whitelist	One-to-One
Yang et al. [28]	Software	Random Noise	Remote	Dynamic	Discrete	Whitelist	One-to-One
de Castro et al . [29]	Software	Timing	Remote	Dynamic	Discrete	Whitelist	One-to-One
He et al. [30]	Software	Timing	Remote	Dynamic	Discrete	Whitelist	One-to-One
Seshradi et al. [31]	Software	Timing	Remote	Static	Discrete	Whitelist	One-to-One
Steiner & Lupu [32]	Software	Timing	Remote	Dynamic	Discrete	Whitelist	One-to-One
Yang et al. [33]	Software	Timing	Remote	Dynamic	Discrete	Whitelist	One-to-One
Jin et al. [34]	Software	Timing	Remote	Dynamic	Discrete	Whitelist	One-to-One
Abera et al. [39]	Hybrid	Trustzone	Remote	Dynamic	Continuous	CF Graph	One-to-One
Dessouky et al. [40]	Hardware	Isolation	Remote	Dynamic	Continuous	CF Graph & DF Digest	One-to-One
Dessouky et al. [41]	Hardware	Isolation	Remote	Dynamic	Continuous	CF Graph &DF Digest	One-to-One
Zeitouni et al. [42]	Hardware	Isolation	Remote	Dynamic	Both	CF Graph &Whitelist	One-to-One
Dushku et al. [43]	Hardware	Isolation	Remote	Dynamic	Continuous	CF Graph	Many-to-One
Noorman et al. [44]	Hardware	Isolation	Both	Static	Discrete	Whitelist	One-to-One
Costan et al. [45]	Hardware	Isolation	Both	Static	Discrete	Whitelist	One-to-One
Conti et al. [46]	Hybrid	Trustzone	Remote	Dynamic	Continuous	CF Graph	Many-to-One
el Defrawy et al. [47]	Hardware	Novel MPU	Remote	Dynamic	Discrete	Whitelist	One-to-One
Suh et al. [49]	Hardware	Isolation	Remote	Static	Discrete	Whitelist	One-to-One
Sumrall et al. [50]	Hardware	Isolation	Both	Static	Discrete	Whitelist	One-to-One
Strackx et al. [51]	Hardware	Isolation	Remote	Static	Discrete	Whitelist	One-to-One
Costan et al. [52]	Hardware	Isolation	Remote	Static	Discrete	Whitelist	One-to-One
Brasser et al. [53]	Hardware	Isolation	Remote	Static	Discrete	Whitelist	One-to-One
Kong et al. [54]	Hardware	PUF	Remote	Dynamic	Discrete	Whitelist	One-to-One
Schulz et al. [55]	Hardware	PUF	Remote	Dynamic	Discrete	Whitelist	One-to-One
Ammar et al. [59]	Hybrid	Minimum HW	Remote	Dynamic	Discrete	Whitelist	Many-to-One
Ammar et al. [60]	Hybrid	Minimum HW	Remote	Dynamic	Discrete	Whitelist	Many-to-One
Ibrahim et al. [61]	Hybrid	TEE	Remote	Dynamic	Discrete	Whitelist	Many-to-One
Kohnhäuser et al. [62]	Hybrid	Minimum HW	Remote	N.A.	N.A.	N.A.	Many to one
Kuang et al. [63]	Hybrid	Minimum HW	Remote	Dynamic	Discrete	Whitelist	Many-to-One
Meng et al. [64]	Hybrid	Minimum HW	Remote	Dynamic	Discrete	Whitelist	Many-to-One
Francillon et al. [65]	Hybrid	Minimum HW	Remote	Dynamic	Discrete	Whitelist	One-to-One
Ambrosin et al. [66]	Hybrid	Minimum HW	Remote	Static	Discrete	Whitelist	Many-to-One
Ibrahim et al. [67]	Hybrid	Minimal HW	Remote	N.A.	N.A.	N.A.	Many-to-Many
Ambrosin et al. [69]	Hybrid	TEE	Local	Static	Discrete	Whitelist	Many-to-Many
Asokan et al. [70]	Hybrid	TEE	Remote	Dynamic	Discrete	Whitelist	Many-to-One
Kohnhäuser et al. [71]	Hybrid	TEE	Local	Static	Discrete	Whitelist	Many-to-Many
Lee [72]	Hybrid	TEE	Remote	Static	Discrete	Whitelist	Many-to-One
Shivraj & Thankur [73]	Hybrid	TEE	Remote	Static	Discrete	Whitelist	Many-to-Many
Wedaj et al. [74]	Hybrid	TEE	Remote	Static	Discrete	Whitelist	Many-to-Many
Carpent et al. [75]	Hybrid	SMART	Remote	Dynamic	Discrete	Whitelist	Many-to-One
Ibrahim et al. [76]	Hybrid	SMART/Trustlite	Remote	Dynamic	Discrete	Whitelist	Many-to-Many
Detken et al. [77]	Hybrid	TPM	Remote	Static	Discrete	Whitelist	One-to-One
kohnhauser et al. [78]	Hybrid	TPM	Local	Static	Discrete	Whitelist	Many-to-Many
Tan et al. [79]	Hybrid	TPM	Local	Static	Discrete	Whitelist	Many-to-Many
Tan et al. [80] (parent nodes)	Hybrid	TPM	Local	Static	Discrete	Whitelist	Many-to-One
Tan et al. [80] (leaf nodes)	Software	Random Noise	Remote	Static	Discrete	Whitelist	Many-to-One
Kil et al. [81]	Hybrid	TPM	Remote	Dynamic	Discrete	Whitelist	One-to-One
Sun et al. [82]	Hybrid	Trustzone	Remote	Dynamic	Continuous	CF Graph &DF Digest	One-to-One
Wang et al. [83]	Hybrid	Intel SGX	Remote	Dynamic	Static	N.A.	One-to-One
Ibrahim et al. [84]	Hybrid	Minimum HW	Remote	Dynamic	Discrete	Whitelist	Many-to-Many
Yan et al. [88]	Hybrid	Isolation	Local	Static	Discrete	Whitelist	Many-to-One
Chen et al. [90]	N.A.	N.A.		Dynamic	Discrete	Whitelist	One-to-One
Wang et al. [91]	N.A.	N.A.	Remote	Dynamic	Continuous	Contextual Graph	One-to-One
Rabbani et al. [105]	N.A.	N.A.	Remote	N.A.	N.A.	N.A.	Many-to-One

it does not allow for easy updates, and because the isolation based designs are often simple and offer relatively simple operation. In contrast, software root of trust has received low in attack resistance, because it is vulnerable to many attacks that hardware and hybrid are immune to. We have scored it medium in complexity, high in compatibility and low in financial cost, because it is often fairly simple to implement and run, it is highly adaptable to many devices, and it does not require any additional hardware. Hybrid root of trust has scored medium in all categories, because it strikes a balance between hardware and software.

In Table 2 we ranked dynamic evidence medium in attack resistance, medium in complexity, high in compatibility, and medium in financial cost, because the quality of the evidence is very dependent on the way it is gathered. Dynamic evidence allows for more evidence to be collected, but depending on the collection algorithm, it may miss key details, have a high complexity, or have a high financial cost. By contrast, we have scored static evidence low in attack resistance, low in complexity, and low in financial cost, because it is vulnerable to known attacks, but does not add any complexity or hardware.

In Table 3 we have scored discrete collection with a low attack resistance, medium in complexity, high in compatibility, and low in financial cost, because discrete evidence collection is very easy to perform on many devices with a variety of methods. Continuous evidence collection, however, has been scored high in attack resistance, because it is capable of detecting Time Of Check Time Of Use (TOCTOU) and Return Oriented Programming (ROP) attacks inherently. Its enabling technology often requires additional, sophisticated hardware, which has driven its complexity to high, its compatibility to low, and its financial cost to high.

In Table 4 whitelist strategies have been scored low in attack resistance, because they do not allow any analysis of evidence which prevents strong attack resistance. They have received a score of high in complexity because setting up a list of allowable system states for an embedded system is a non-trivial task and comparing evidence to a given system state is also non-trivial. A whitelist approach does not require any additional hardware however, so we have scored it high in compatibility and low in cost. Control flow graphs and data flow graphs offer a greater attack resistance at the cost of a low compatibility and medium to high financial cost.

Finally, in Table 5 we have scored many-to-one schemes medium in attack resistance, medium in complexity, medium in compatibility, and medium in financial cost. This is due to the trade-offs that many-to-one designs make. They provide less security by relying on less hardware in the root of trust, they reduce complexity by reducing network overhead, they maintain compatibility by keeping their designs heterogeneous, and they reduce cost by reducing the number of verifiers necessary in a network. Many-to-many offers high attack resistance, because many schemes rely on additional hardware to allow devices to attest themselves to each other. As a result, they have been scored a medium in complexity, low in compatibility, and high in financial cost. A one-to-one approach has been scored high in attack resistance, because no consideration is given to the cost to the verifier. As a consequence, we have scored them high in complexity, medium in compatibility, and high in financial cost.

It is important to note that the details provided here must be considered in the context of a particular deployment scenario. For instance, a hardware root of trust, which has a high security, high complexity, and high cost may be appropriate for a critical service, like power distribution. In another context, like a smart home setting, a hardware root of trust may be overkill considering risk and financial cost.

X. SUMMARY AND RESEARCH GAP

Tables 6 and 7 summarize the existing RA literature based on our taxonomy as shown in Fig. 7. Table 6 applies the taxonomy that we discuss in previous sections to the 58 RA schemes surveyed here, where the Root of Trust Mechanism column gives further detail on the specific root of trust used by each scheme. It is important to note that some schemes have a half-circle for the M_{SL} category. This is because they only provide evidence for a limited period of time [39], or because they add memory protection to detect when a compromise occurs [26], [42] rather than collect evidence from a prover over a long period of time. Table 7 gives further detail on CRA schemes using the categories discussed in section VIII. The Scalability Mechanism column gives further detail on the particular mechanism used to provide scalability in each CRA scheme. Table 8 provides an overview of each scheme's security benefits in terms of the threat model, and their resistance to proxy, DoS, and TOCTOU attacks. We have treated these 3 attacks as special cases because they are often considered very difficult to defend against, and they can occur under almost any threat model.

The following are our major observations based on the information we have compiled in Tables 6, 7, and 8.

- Hardware and hybrid based RA schemes are able to leverage much greater security than software based schemes. They have access to cryptography, which allows them to easily defend against local adversaries, proxy attacks, and DoS attacks. In general, however, these RA schemes rely on specific features that are not applicable to all embedded systems. For instance, not every embedded system will have access to the Arm Trustzone or the Intel SGX. Additional research should be done to allow similar security features to a wider variety of embedded systems.
- Dynamic evidence gives a much clearer picture of the state of the prover device, but collection is challenging. Static collection mechanisms are vulnerable to TOCTOU, memory copy, compression, and other attacks. Dynamic collection mechanisms allow an RA scheme to monitor the control flow of a prover device, but little research has been done on embedded systems that do not run a service on bare metal. Static evidence

TABLE 7. Further detail on CRA schemes. It is important to note that not every CRA scheme addresses every part of our proposed taxonomy. Any scheme that does not consider a particular problem has that field marked N.A.

	Scalability	Scalability Mechanism	Network Mobility	Homogeneity	Absence Detection
Carpent et al. [15]	Many-to-One	Spanning Tree	Quasi-static	Heterogeneous	No
Jin et al. [34]	Many-to-One	Hop-by-hop	Quasi-static	Homogeneous	No
Dushku et al. [43]	Many-to-One	Service Control Flow	Static	Heterogeneous	No
Conti et al. [46]	Many-to-One	Service Control Flow	Static	Heterogeneous	No
Ammar et al. [59]	Many-to-One	Secure Broadcast	Dynamic	Heterogeneous	Yes
Ammar et al. [60]	Many-to-One	Secure Broadcast	Quasi-static	Heterogeneous	No
Ibrahim et al. [61]	Many-to-One	Non-Interactive	Static	Heterogeneous	No
Kohnhäuser et al. [62]	Many-to-One	Heartbeat	Dynamic	Heterogeneous	Yes
Kuang et al. [63]	Many-to-One	Hop-by-hop	Quasi-Static	Heterogeneous	No
Meng et al. [64]	Many-to-One	Spanning Tree	Dynamic	Homogeneous	No
Ambrosin et al. [66]	Many-to-One	Hop-by-hop	Dynamic	Heterogeneous	No
Ibrahim et al. [67]	Many-to-Many	Heartbeat	Quasi-static	N.A.	Yes
Ambrosin et al. [69]	Many-to-Many	Consensus	Dynamic	Heterogeneous	No
Asokan et al. [70]	Many-to-One	Spanning Tree	Quasi-static	Heterogeneous	No
Kohnhäuser et al. [71]	Many-to-Many	Group Signature	Dynamic	Heterogeneous	Yes
Lee [72]	Many-to-One	Aggregation	Quasi-Static	Heterogeneous	No
Shivraj & Thankur [73]	Many-to-Many	Consensus	Static	Homogeneous	No
Wedaj et al. [74]	Many-to-Many	Peer verification	Dynamic	Homogeneous	No
Carpent et al. [75]	Many-to-One	Spanning Tree	Quasi-static	Homogeneous	No
Ibrahim et al. [76]	Many-to-Many	Peer Verification	Quasi-static	Homogeneous	Yes
kohnhauser et al. [78]	Many-to-Many	Aggregation	Dynamic	Heterogeneous	No
Tan et al. [79]	Many-to-Many	Peer Verification	Static	Homogeneous	No
Tan et al. [80]	Many-to-One	Hop-by-hop	Static	Heterogeneous	No
Ibrahim et al. [84]	Many-to-Many	Peer verification	Static	Homogeneous	No
Yan et al. [88]	Many-to-One	Spanning Tree	Quasi-static	Heterogeneous	Yes
Rabbani et al. [105]	Many-to-One	Hop-by-hop	Dynamic	Heterogeneous	Yes

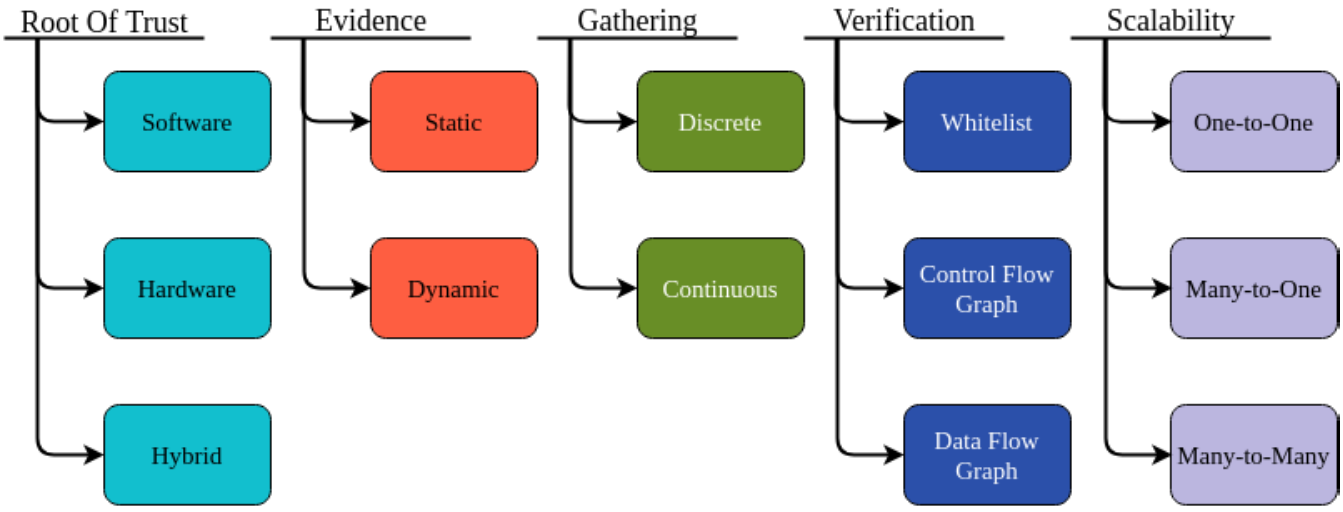

FIGURE 7. Overview of our proposed taxonomy.

collection should be investigated in greater detail to defend against the aforementioned attacks. Dynamic collection mechanisms for embedded systems with multiple services, non-bare metal deployment, or both should also be investigated.

- Very few RA schemes reviewed here give context for the evidence that they collect. The majority assume that the verifier can keep a whitelist of allowable system states and compare cryptographic hashes against the whitelist. This is impractical for dynamic evidence. As embedded systems become more advanced, their services will become more complex. Additional research should be conducted to allow the verifier to perform analysis of evidence.
- Scalability should be considered a high priority in RA research. Existing CRA schemes largely trade some amount of security for scalability. Several schemes assume that devices will either be capable of Local Attestation (LA), or devices will be homogeneous so evidence can be reduced. Neither case is ideal. LA is

TABLE 8. Security comparison of the schemes surveyed here.

	A_R	A_L	A_{PN}	A_{PI}	M_{SF}	M_{SL}	M_{DF}	M_{DL}	Proxy	DoS	TOCTOU
Carpent et al. [15]	●	●	○	○	●	○	●	○	●	●	○
Agarwal et al. [21]	●	●	○	○	○	●	○	●	○	○	●
Vetter et al. [24]	●	○	○	○	○	○	○	●	○	○	○
Shaneck et al. [25]	●	○	○	○	○	○	○	●	○	○	○
Zhang et al. [26]	●	○	○	○	○	◐	○	○	○	○	○
Abuhmed et al. [27]	●	○	○	○	○	○	○	●	○	○	○
Yang et al. [28]	●	○	○	○	○	○	○	●	○	○	○
de Castro et al . [29]	●	○	○	○	●	○	●	●	○	○	○
He et al. [30]	●	○	○	○	●	○	●	●	○	○	○
Seshradi et al. [31]	●	○	○	○	●	○	●	○	○	○	○
Steiner & Lupu [32]	●	○	○	○	●	○	●	●	○	○	○
Yang et al. [33]	●	○	○	○	●	○	●	●	○	○	○
Jin et al. [34]	●	○	○	○	●	○	●	●	○	○	○
Abera et al. [39]	●	●	○	○	○	◐	○	○	●	●	○
Dessouky et al. [40]	●	●	○	○	○	●	○	○	●	●	●
Dessouky et al. [41]	●	●	○	○	○	●	○	○	●	●	●
Zeitouni et al. [42]	●	●	●	○	●	◐	●	○	●	●	●
Dushku et al. [43]	●	●	○	○	○	●	○	○	●	●	●
Noorman et al. [44]	●	●	○	○	●	○	●	○	●	●	○
Costan et al. [45]	●	●	○	○	●	○	●	○	●	●	○
Conti et al. [46]	●	●	○	○	○	●	○	○	●	●	●
el Defrawy et al. [47]	●	●	○	○	●	○	●	○	●	○	○
Suh et al. [49]	●	●	○	○	●	○	●	○	●	●	○
Sumrall et al. [50]	●	●	○	○	●	○	●	○	●	●	○
Strackx et al. [51]	●	●	○	○	●	○	●	○	●	●	○
Costan et al. [52]	●	●	○	○	●	○	●	○	●	●	○
Brasser et al. [53]	●	●	○	○	●	○	●	○	●	●	○
Kong et al. [54]	●	○	○	○	●	○	●	○	●	○	○
Schulz et al. [55]	●	○	○	○	●	○	●	○	●	○	○
Ammar et al. [59]	●	●	○	○	●	○	●	○	●	○	○
Ammar et al. [60]	●	●	○	●	●	○	●	○	●	○	○
Ibrahim et al. [61]	●	●	○	○	●	○	●	○	●	●	○
Kohnhäuser et al. [62]	●	●	○	●	○	○	○	○	●	●	○
Kuang et al. [63]	●	●	○	○	●	○	●	●	●	●	○
Meng et al. [64]	●	●	○	○	●	○	●	●	●	●	○
Francillon et al. [65]	●	●	○	○	●	○	●	●	○	●	○
Ambrosin et al. [66]	●	●	○	○	●	○	●	○	●	●	○
Ibrahim et al. [67]	●	●	○	●	○	○	○	○	○	○	○
Ambrosin et al. [69]	●	●	○	○	●	○	●	○	●	●	○
Asokan et al. [70]	●	●	○	○	●	○	●	●	●	●	○
Kohnhäuser et al. [71]	●	●	○	●	●	○	●	○	●	●	○
Lee [72]	●	●	○	○	●	○	●	○	●	●	○
Shivraj & Thankur [73]	●	●	○	○	●	○	●	○	●	●	○
Wedaj et al. [74]	●	●	○	○	●	○	●	○	●	●	○
Carpent et al. [75]	●	●	○	○	●	○	●	●	●	●	○
Ibrahim et al. [76]	●	●	○	●	●	○	●	●	●	●	○
Detken et al. [77]	●	●	○	○	●	○	●	○	○	○	○
kohnhauser et al. [78]	●	●	○	○	●	○	●	○	●	●	○
Tan et al. [79]	●	●	○	○	●	○	●	○	●	●	○
Tan et al. [80] (parent nodes)	●	●	○	○	●	○	●	○	●	●	○
Tan et al. [80] (leaf nodes)	●	○	○	○	○	○	●	○	○	○	○
Kil et al. [81]	●	●	○	○	●	○	●	●	●	●	○
Sun et al. [82]	●	●	○	○	○	●	○	○	●	●	●
Wang et al. [83]	●	●	○	○	●	○	●	●	●	●	○
Ibrahim et al. [84]	●	●	○	○	●	○	●	●	●	●	○
Yan et al. [88]	●	●	○	●	○	○	○	○	●	●	○
Chen et al. [90]	○	○	○	○	○	○	○	●	○	○	○
Wang et al. [91]	○	○	○	○	○	●	○	●	○	○	○
Rabbani et al. [105]	●	●	○	●	○	○	○	○	●	●	○

largely constrained to static evidence which omits the possibility of M_{DL} and M_{SL} malware models. Many networks that employ embedded systems use a variety of different devices.

- Very few RA schemes are able to defend against a physical adversary of any kind. Some CRA schemes are able to leverage a heartbeat mechanism to detect device absence. In this way, they can defend against a A_{PI} adversarial assumption, but not a A_{PN} assumption. Additional research should be conducted to detect physical adversaries in both settings.
- TOCTOU attacks should also be considered a high priority. If ignored, an attacker can simply infect a device, accomplish their goal, and leave the device before a check occurs. Dynamic RA schemes are uniquely able to defend against a TOCTOU attack, because they collect evidence over a period of time. Given the limitations on dynamic collection schemes discussed above, they are not a blanket solution at this time. Static collection schemes may be able to defend against a TOCTOU attack through random challenge times, but further investigations should be conducted.

XI. CONCLUSION

We have reviewed 58 research articles on Remote Attestation (RA) in embedded systems. These articles covers a wide range issues related to RA schemes in embedded systems including novel roots of trust [32], [40], [43], discussions on novel attacks [10], [32], [33], [92] and proposed methods to improve scalability. We chose these works because they represent the most recent and relevant research for RA of embedded systems.

The following are the contributions of this survey paper. First, we have proposed an updated taxonomy for RA in embedded systems. Our work expands on the taxonomy provided by Steiner *et al.* [10] but it re-frames the problem of RA into the 5 major challenges: root of trust, evidence, evidence collection method, packaging and verification, and scalability. Design choice made to address one of these challenges affects the design choices that can be made for other challenges. Our proposed taxonomy considers all the challenges together, not in isolation as has been done by previous surveys. Second, we have presented an enhanced threat model to allow a better analysis of RA schemes for embedded systems. We have modified the existing adversarial assumptions given by Abera *et al.* [11] and introduced a new metric: the malware model. The malware model clarifies what type of malware will be defended against by the RA scheme. It measures the assumed maximum level of comprise that an RA scheme will defend against. Third, we have applied our proposed taxonomy over 58 recent RA schemes reported in literature. We have highlighted the research breakthroughs that have already been made, and point out areas that have yet to be investigated. Fourth, we give details concerning the benefit cost trade off of each category in our proposed taxonomy. We have chosen to give these details qualitatively rather than quantitatively such that the reader might make an informed decision about which RA scheme might be best for their particular network setting. Our cost benefit details are not intended to be conclusive, but to give the reader an idea of what each design choice might gain and sacrifice. We have provided a discussion on future research in the summary section.

Our planned future research work includes investigation of mechanisms to provide a root of trust that have the same security assumptions as many modern hardware and hybrid schemes without the reliance on specific hardware features that are not available to many embedded systems. We plan to investigate stronger memory collection and packaging methods such that dynamic evidence can be presented to the verifier with contextual information. We hope to allow the verifier to apply analysis over evidence, rather than a white-listing approach.

ACKNOWLEDGMENT

Notice: This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

REFERENCES

[1] A. Cui, "Embedded system security: A software-based approach," Ph.D. dissertation, Dept. Comput. Sci., Columbia Univ., New York, NY, USA, 2015.
[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, and D. Kumar, "Understanding the Mirai botnet," in *Proc. 26th Secur. Symp. (USENIX Secur.)*, 2017, pp. 1093–1110.
[3] J. P. Farwell and R. Rohozinski, "Stuxnet and the future of cyber war," *Survival*, vol. 53, no. 1, pp. 23–40, 2011.
[4] C.-W. Ten, K. Yamashita, Z. Yang, A. V. Vasilakos, and A. Ginter, "Impact assessment of hypothesized cyberattacks on interconnected bulk power systems," *IEEE Trans. Smart Grid*, vol. 9, no. 5, pp. 4405–4425, Sep. 2017.
[5] Sudhakar and S. Kumar, "An emerging threat fileless malware: A survey and research challenges," *Cybersecurity*, vol. 3, no. 1, pp. 1–12, Dec. 2020.
[6] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla, "SCUBA: Secure code update by attestation in sensor networks," in *Proc. 5th ACM Workshop Wireless Secur. (WiSe)*, 2006, pp. 85–94.
[7] D. Perito and G. Tsudik, "Secure code update for embedded devices via proofs of secure erasure," in *Proc. Eur. Symp. Res. Comput. Secur.* Heidelberg, Germany: Springer, 2010, pp. 643–662.
[8] J. Zhao, J. Liu, Z. Qin, and K. Ren, "Privacy protection scheme based on remote anonymous attestation for trusted smart meters," *IEEE Trans. Smart Grid*, vol. 9, no. 4, pp. 3313–3320, Jul. 2016.
[9] A. Seshadri, M. Luk, and A. Perrig, "Sake: Software attestation for key establishment in sensor networks," in *Proc. Int. Conf. Distrib. Comput. Sensor Syst.* Heidelberg, Germany: Springer, 2008, pp. 372–385.
[10] R. V. Steiner and E. Lupu, "Attestation in wireless sensor networks: A survey," *ACM Comput. Surveys*, vol. 49, no. 3, pp. 1–31, Dec. 2016.
[11] T. Abera, N. Asokan, L. Davi, F. Koushanfar, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "Invited–things, trouble, trust: On building trust in IoT systems," in *Proc. 53rd Annu. Design Autom. Conf.*, Jun. 2016, pp. 1–6.

[12] P. Maene, J. Götzfried, R. D. Clercq, T. Müller, F. Freiling, and I. Verbauwhede, "Hardware-based trusted computing architectures for isolation and attestation," *IEEE Trans. Comput.*, vol. 67, no. 3, pp. 361–374, Mar. 2017.

[13] O. Arias, F. Rahman, M. Tehranipoor, and Y. Jin, "Device attestation: Past, present, and future," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 473–478.

[14] M. Ambrosin, M. Conti, R. Lazzeretti, M. M. Rabbani, and S. Ranise, "Collective remote attestation at the Internet of Things scale: State-of-the-art and future challenges," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 4, pp. 2447–2461, Jul. 2020.

[15] X. Carpent, G. Tsudik, and N. Rattanavipanon, "ERASMUS: Efficient remote attestation via self-measurement for unattended settings," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 1191–1194.

[16] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *Proc. 27th Secur. Symp. (USENIX Secur.)*, 2018, pp. 991–1008.

[17] D. Moghimi, B. Sunar, T. Eisenbarth, and N. Heninger, "TPM-FAIL: TPM meets timing and lattice attacks," in *Proc. 29th Secur. Symp. (USENIX Secur.)*, 2020, pp. 2057–2073.

[18] F. Brasser, K. B. Rasmussen, A.-R. Sadeghi, and G. Tsudik, "Remote attestation for low-end embedded devices: The prover's perspective," in *Proc. 53nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[19] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, vol. 1, Aug. 2015, pp. 57–64.

[20] D. Spinellis, "Reflection as a mechanism for software integrity verification," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 1, pp. 51–62, Feb. 2000.

[21] N. Agarwal and K. Paul, "XEBRA: XEn based remote attestation," in *Proc. IEEE Region Conf. (TENCON)*, Nov. 2016, pp. 2383–2386.

[22] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, J. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen, "Principles of remote attestation," *Int. J. Inf. Secur.*, vol. 10, no. 2, pp. 63–81, 2011.

[23] R. Perez, R. Sailer, and L. van Doorn, "vTPM: Virtualizing the trusted platform module," in *Proc. 15th Conf. USENIX Secur. Symp.*, 2006, pp. 305–320.

[24] B. Vetter and D. Westhoff, "Simulation study on code attestation with compressed instruction code," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. Workshops*, Mar. 2012, pp. 296–301.

[25] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim, "Remote software-based attestation for wireless sensors," in *Proc. Eur. Workshop Secur. Ad-Hoc Sensor Netw.* Heidelberg, Germany: Springer, 2005, pp. 27–41.

[26] D. Zhang and D. Liu, "DataGuard: Dynamic data attestation in wireless sensor networks," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2010, pp. 261–270.

[27] T. AbuHmed, N. Nyamaa, and D. Nyang, "Software-based remote code attestation in wireless sensor network," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Nov. 2009, pp. 1–8.

[28] Y. Yang, X. Wang, S. Zhu, and G. Cao, "Distributed software-based attestation for node compromise detection in sensor networks," in *Proc. 26th IEEE Int. Symp. Reliable Distrib. Syst. (SRDS)*, Oct. 2007, pp. 219–230.

[29] C. G. D. Castro, S. D. M. Câmara, L. F. R. D. C. Carmo, and D. R. Boccardo, "EVINCED: Integrity verification scheme for embedded systems based on time and clock cycles," in *Proc. IEEE 15th Int. Conf. Dependable, Autonomic Secure Comput., 15th Int. Conf. Pervas. Intell. Comput., 3rd Int. Conf. Big Data Intell. Comput. Cyber Sci. Technol. Congr. (DASC/PiCom/DataCom/CyberSciTech)*, Nov. 2017, pp. 788–795.

[30] X. He, X. Yang, R. Li, and Q. Yang, "A novel delay-resilient remote memory attestation for smart grid," in *Proc. Int. Conf. Wireless Algorithms, Syst., Appl.* Heidelberg, Germany: Springer, 2013, pp. 88–99.

[31] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "SWATT: Software-based attestation for embedded devices," in *Proc. IEEE Symp. Secur. Privacy*, May 2004, pp. 272–282.

[32] R. V. Steiner and E. Lupu, "Towards more practical software-based attestation," *Comput. Netw.*, vol. 149, pp. 43–55, Feb. 2019.

[33] X. Yang, X. He, W. Yu, J. Lin, R. Li, Q. Yang, and H. Song, "Towards a low-cost remote memory attestation for the smart grid," *Sensors*, vol. 15, no. 8, pp. 20799–20824, Aug. 2015.

[34] X. Jin, P. Putthapipat, D. Pan, N. Pissinou, and S. K. Makki, "Unpredictable software-based attestation solution for node compromise detection in mobile WSN," in *Proc. IEEE Globecom Workshops*, Dec. 2010, pp. 2059–2064.

[35] R. R. Branco, G. N. Barbosa, and P. D. Neto, "Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-VM technologies," *Black Hat*, vol. 1, pp. 1–27, Jul. 2012.

[36] M. Polino, A. Continella, S. Mariani, S. D'Alessio, L. Fontana, F. Gritti, and S. Zanero, "Measuring and defeating anti-instrumentation-equipped malware," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Cham, Switzerland: Springer, 2017, pp. 73–96.

[37] M. Kozuch and A. Wolfe, "Compression of embedded system programs," in *Proc. IEEE Int. Conf. Comput. Design, VLSI Comput. Processors*, Oct. 1994, pp. 270–277.

[38] A. Sang and S.-Q. Li, "A predictability analysis of network traffic," *Comput. Netw.*, vol. 39, no. 4, pp. 329–345, 2002.

[39] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-FLAT: Control-flow attestation for embedded systems software," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 743–754.

[40] G. Dessouky, S. Zeitouni, T. Nyman, A. Paverd, L. Davi, P. Koeberl, N. Asokan, and A.-R. Sadeghi, "LO-FAT: Low-overhead control flow ATtestation in hardware," in *Proc. 54th Annu. Design Autom. Conf.*, Jun. 2017, pp. 1–6.

[41] G. Dessouky, T. Abera, A. Ibrahim, and A.-R. Sadeghi, "LiteHAX: Lightweight hardware-assisted attestation of program execution," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2018, pp. 1–8.

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

[43] E. Dushku, M. M. Rabbani, M. Conti, L. V. Mancini, and S. Ranise, "SARA: Secure asynchronous remote attestation for IoT systems," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3123–3136, 2020.

[44] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. Van Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens, "Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base," in *Proc. 22nd Secur. Symp. (USENIX Secur.)*, 2013, pp. 479–498.

[45] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016.

[46] M. Conti, E. Dushku, and L. V. Mancini, "RADIS: Remote attestation of distributed IoT services," in *Proc. 6th Int. Conf. Softw. Defined Syst. (SDS)*, Jun. 2019, pp. 25–32.

[47] K. Eldefrawy, G. Tsudik, A. Francillon, and A. Perito, "SMART: Secure and minimal architecture for (establishing dynamic) root of trust," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, vol. 12, Feb. 2012, pp. 1–15.

[48] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "TrustLite: A security architecture for tiny embedded devices," in *Proc. 9th Eur. Conf. Comput. Syst. (EuroSys)*, 2014, pp. 1–14.

[49] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "AEGIS: Architecture for tamper-evident and tamper-resistant processing," in *Proc. 25th Anniversary Int. Conf. Supercomput. Anniversary*, 2014, pp. 357–368.

[50] N. Sumrall and M. Novoa, "Trusted computing group (TCG) and the TPM 1.2 specification," in *Intel Developer Forum*, vol. 32. Santa Clara, CA, USA: Intel, Sep. 2003.

[51] R. Strackx, J. Noorman, I. Verbauwhede, B. Preneel, and F. Piessens, "Protected software module architectures," in *Proc. Securing Electron. Bus. Processes (ISSE)*. Wiesbaden, Germany: Springer, 2013, pp. 241–251.

[52] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. 25th Secur. Symp. (USENIX Secur.)*, 2016, pp. 857–874.

[53] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "TyTAN: Tiny trust anchor for tiny devices," in *Proc. 52nd Annu. Design Autom. Conf.*, Jun. 2015, pp. 1–6.

[54] J. Kong, F. Koushanfar, P. K. Pendyala, A.-R. Sadeghi, and C. Wachsmann, "PUFatt: Embedded platform attestation based on novel processor-based PUFs," in *Proc. 51st ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2014, pp. 1–6.

[55] S. Schulz, A.-R. Sadeghi, and C. Wachsmann, "Short paper: Lightweight remote attestation using physical functions," in *Proc. 4th ACM Conf. Wireless Netw. Secur. (WiSec)*, 2011, pp. 109–114.

[56] (Feb. 2021). *Stmicroelectronics Reveals Extreme Low-Power STM32U5 Microcontrollers With Advanced Performance and Cybersecurity*. [Online]. Available: https://newsroom.st.com/media-center/press-item.html/p4329.html

[57] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, and Y. Yarom, "Meltdown: Reading kernel memory from user space," in *Proc. 27th Secur. Symp. (USENIX Secur.)*, 2018, pp. 973–990.

[58] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 1–19.

[59] M. Ammar, M. Washha, G. S. Ramabhadran, and B. Crispo, "SlimIoT: Scalable lightweight attestation protocol for the Internet of Things," in *Proc. IEEE Conf. Dependable Secure Comput. (DSC)*, Dec. 2018, pp. 1–8.

[60] M. Ammar, M. Washha, and B. Crispo, "WISE: Lightweight intelligent swarm attestation scheme for IoT (the verifier's perspective)," in *Proc. 14th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2018, pp. 1–8.

[61] A. Ibrahim, A.-R. Sadeghi, and S. Zeitouni, "SeED: Secure non-interactive attestation for embedded devices," in *Proc. 10th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2017, pp. 64–74.

[62] F. Kohnhäuser, N. Büscher, S. Gabmeyer, and S. Katzenbeisser, "SCAPI: A scalable attestation protocol to detect software and physical attacks," in *Proc. 10th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, Jul. 2017, pp. 75–86.

[63] B. Kuang, A. Fu, S. Yu, G. Yang, M. Su, and Y. Zhang, "ESDRA: An efficient and secure distributed remote attestation scheme for IoT swarms," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8372–8383, Oct. 2019.

[64] W. Meng, T. Jiang, and J. Ge, "Dynamic swarm attestation with malicious devices identification," *IEEE Access*, vol. 6, pp. 50003–50013, 2018.

[65] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik, "Systematic treatment of remote attestation," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 713, Dec. 2012.

[66] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A.-R. Sadeghi, and M. Schunter, "SANA: Secure and scalable aggregate network attestation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 731–742.

[67] A. Ibrahim, A.-R. Sadeghi, G. Tsudik, and S. Zeitouni, "DARPA: Device attestation resilient to physical attacks," in *Proc. 9th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, Jul. 2016, pp. 171–182.

[68] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik, "A minimalist approach to remote attestation," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6.

[69] M. Ambrosin, M. Conti, R. Lazzeretti, M. M. Rabbani, and S. Ranise, "PADS: Practical attestation for highly dynamic swarm topologies," in *Proc. Int. Workshop Secure Internet Things (SIoT)*, Sep. 2018, pp. 18–27.

[70] N. Asokan, F. Brasser, A. Ibrahim, A.-R. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann, "SEDA: Scalable embedded device attestation," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 964–975.

[71] F. Kohnhäuser, N. Büscher, and S. Katzenbeisser, "A practical attestation protocol for autonomous embedded systems," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS P)*, Jun. 2019, pp. 263–278.

[72] J. Lee, "Collective attestation for manageable IoT environments," *Appl. Sci.*, vol. 8, no. 12, p. 2652, Dec. 2018.

[73] V. L. Shivraj, M. S. D. Thakur, and R. Ma, "A novel multi verifier device attestation scheme for swarm of devices," in *Proc. 32nd Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA)*, May 2018, pp. 124–129.

[74] S. Wedaj, K. Paul, and V. J. Ribeiro, "DADS: Decentralized attestation for device swarms," *ACM Trans. Privacy Secur.*, vol. 22, no. 3, pp. 1–29, Jul. 2019.

[75] X. Carpent, K. ElDefrawy, N. Rattanavipanon, and G. Tsudik, "Lightweight swarm attestation: A tale of two LISA-s," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Apr. 2017, pp. 86–100.

[76] A. Ibrahim, A.-R. Sadeghi, and G. Tsudik, "US-AID: Unattended scalable attestation of IoT devices," in *Proc. IEEE 37th Symp. Reliable Distrib. Syst. (SRDS)*, Oct. 2018, pp. 21–30.

[77] K.-O. Detken, M. Jahnke, T. Rix, and A. Rein, "Software-design for internal security checks with dynamic integrity measurement (DIM)," in *Proc. 9th IEEE Int. Conf. Intell. Data Acquisition Adv. Comput. Syst., Technol. Appl. (IDAACS)*, vol. 1, Sep. 2017, pp. 367–373.

[78] F. Kohnhäuser, N. Büscher, and S. Katzenbeisser, "SALAD: Secure and lightweight attestation of highly dynamic and disruptive networks," in *Proc. Asia Conf. Comput. Commun. Secur.*, May 2018, pp. 329–342.

[79] H. Tan, W. Hu, and S. Jha, "A remote attestation protocol with trusted platform modules (TPMs) in wireless sensor networks," *Secur. Commun. Netw.*, vol. 8, no. 13, pp. 2171–2188, 2015.

[80] H. Tan, G. Tsudik, and S. Jha, "MTRA: Multiple-tier remote attestation in IoT networks," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Oct. 2017, pp. 1–9.

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

[82] Z. Sun, B. Feng, L. Lu, and S. Jha, "OAT: Attesting operation integrity of embedded devices," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1433–1449.

[83] J. Wang, Z. Hong, Y. Zhang, and Y. Jin, "Enabling security-enhanced attestation with Intel SGX for remote terminal and IoT," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 88–96, Jan. 2017.

[84] A. Ibrahim, A.-R. Sadeghi, and G. Tsudik, "Healed: Healing & attestation for low-end embedded devices," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2019, pp. 627–645.

[85] D. Rosenberg, "QSEE trustzone kernel integer over flow vulnerability," in *Proc. Black Hat Conf.*, 2014, p. 26.

[86] N. Weichbrodt, A. Kurmus, P. Pietzuch, and R. Kapitza, "AsyncShock: Exploiting synchronisation bugs in Intel SGX enclaves," in *Proc. Eur. Symp. Res. Comput. Secur.* Cham, Switzerland: Springer, 2016, pp. 440–457.

[87] S. Pinto and N. Santos, "Demystifying arm TrustZone: A comprehensive survey," *ACM Comput. Surveys*, vol. 51, no. 6, pp. 1–36, Feb. 2019.

[88] W. Yan, A. Fu, Y. Mu, X. Zhe, S. Yu, and B. Kuang, "EAPA: Efficient attestation resilient to physical attacks for IoT devices," in *Proc. 2nd Int. ACM Workshop Secur. Privacy Internet Things (IoT S P)*, 2019, pp. 2–7.

[89] D. J. Sorin, M. D. Hill, and D. A. Wood, "A primer on memory consistency and cache coherence," *Synth. Lectures Comput. Archit.*, vol. 6, no. 3, pp. 1–212, Nov. 2011.

[90] B. Chen, X. Dong, G. Bai, S. Jauhar, and Y. Cheng, "Secure and efficient software-based attestation for industrial control devices with ARM processors," in *Proc. 33rd Annu. Comput. Secur. Appl. Conf.*, Dec. 2017, pp. 425–436.

[91] F. Wang, Y. Joung, and J. Mickens, "Cobweb: Practical remote attestation using contextual graphs," in *Proc. 2nd Workshop Syst. Softw. Trusted Execution*, Oct. 2017, pp. 1–7.

[92] Y. Li, Y. Cheng, V. Gligor, and A. Perrig, "Establishing software-only root of trust on embedded systems: Facts and fiction," in *Proc. Cambridge Int. Workshop Secur. Protocols*. Cham, Switzerland: Springer, 2015, pp. 50–68.

[93] G. Wurster, P. C. van Oorschot, and A. Somayaji, "A generic attack on checksumming-based software tamper resistance," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2005, pp. 127–138.

[94] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente, "On the difficulty of software-based attestation of embedded devices," in *Proc. 16th ACM Conf. Comput. Commun. Secur. (CCS)*, 2009, pp. 400–409.

[95] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks," in *Proc. 10th Annu. ACM Symp. Princ. Distrib. Comput.*, 1991, pp. 51–59.

[96] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)," in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, 2007, pp. 552–561.

[97] E. Buchanan, R. Roemer, H. Shacham, and S. Savage, "When good instructions go bad: Generalizing return-oriented programming to RISC," in *Proc. 15th ACM Conf. Comput. Commun. Secur. (CCS)*, 2008, pp. 27–38.

[98] H. Hu, S. Shinde, S. Adrian, Z. L. Chua, P. Saxena, and Z. Liang, "Data-oriented programming: On the expressiveness of non-control data attacks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 969–986.

[99] V. Kuznetzov, L. Szekeres, M. Payer, G. Candea, R. Sekar, and D. Song, "Code-pointer integrity," in *The Continuing Arms Race: Code-Reuse Attacks and Defenses*. San Rafael, CA, USA: Morgan & Claypool, 2018, pp. 81–116.

[100] P. Larsen, A. Homescu, S. Brunthaler, and M. Franz, "SoK: Automated software diversity," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 276–291.

[101] M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti, "Control-flow integrity principles, implementations, and applications," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 1, pp. 1–40, Oct. 2009.
[102] F. B. Cohen, "Operating system protection through program evolution," *Comput. Secur.*, vol. 12, no. 6, pp. 565–584, 1993.
[103] R. Sobti and G. Geetha, "Cryptographic hash functions: A review," *Int. J. Comput. Sci. Issues*, vol. 9, no. 2, p. 461, 2012.
[104] R. C. Merkle, "Method of providing digital signatures," U.S. Patent 4 309 569, Jan. 5, 1982.
[105] M. M. Rabbani, J. Vliegen, J. Winderickx, M. Conti, and N. Mentens, "SHeLA: Scalable heterogeneous layered attestation," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 10240–10250, Dec. 2019.

WILLIAM A. JOHNSON was born in Chattanooga, TN, USA, in 1995. He received the B.S. degree in computer engineering from Tennessee Tech University, in 2018, and the M.S. degree in computer science (as part of his direct admit Ph.D. program) from Tennessee Tech University, in 2021. He is currently pursuing the Ph.D. degree in computer science, with a focus in cyber-security.

He has worked as a temporary Lecturer at Tennessee Tech University as well as a Student Researcher at the Oak Ridge National Laboratory (ORNL). His research interests include security for embedded systems and cyber physical systems, penetration testing, ethical hacking, and secure hardware. In 2018, he received the National Science Foundation's Scholarship for Service (SFS).

SHEIKH GHAFOOR (Member, IEEE) received the Ph.D. degree in computer science from Mississippi State University, in 2007. He is currently a Professor of computer science at Tennessee Tech University. Prior to joining Tennessee Tech as a Faculty, he worked as a Research Associate at the Engineering Research Center (ERC), Mississippi State University. His research interests include high performance message passing, programming model, libraries, and tools for heterogeneous high performance computing environment. He is also actively engaged in research in embedded system security as well as computer science education.

STACY PROWELL (Senior Member, IEEE) received the Ph.D. degree in computer science from The University of Tennessee, in 1996. He is currently the Chief Cyber Security Research Scientist at the Oak Ridge National Laboratory. He is also an Adjunct Professor with the Computer Science Department, Tennessee Technological University. His primary research interests include the security of control systems in general and the power grid and vehicles in particular. He also researches novel methods in malware discovery and automated reverse engineering.