# INVITED
# Things, Trouble, Trust: On Building Trust in IoT Systems

Tigist Abera
TU Darmstadt

N. Asokan
Aalto University

Lucas Davi
TU Darmstadt

Farinaz Koushanfar
UC San Diego

Andrew Paverd
Aalto University

Ahmad-Reza Sadeghi
TU Darmstadt

Gene Tsudik
UC Irvine

## ABSTRACT

The emerging and much-touted Internet of Things (IoT) presents a variety of security and privacy challenges. Prominent among them is the *establishment of trust* in remote IoT devices, which is typically attained via remote attestation, a distinct security service that aims to ascertain the current state of a potentially compromised remote device. Remote attestation ranges from relatively heavy-weight secure hardware-based techniques, to light-weight software-based ones, and also includes approaches that blend software (e.g., control-flow integrity) and hardware features (e.g., PUFs). In this paper, we survey the *landscape* of state-of-the-art attestation techniques from the IoT device perspective and argue that most of them have a role to play in IoT trust establishment.

## Keywords

Trust Establishment, Remote Attestation, Internet of Things

## 1. INTRODUCTION

The increasingly popular paradigm of *"Internet of Things"* is the product of the current trend to interconnect all kinds of devices and systems. IoT encompasses a broad spectrum of application domains, ranging from large-scale smart energy grids to personal wearable devices. One distinguishing feature of IoT is that it involves cost- and/or resource-constrained devices: so-called *"things"*. These constraints, coupled with the scale of IoT devices, present significant security and privacy challenges.

One key challenge is the vulnerability of IoT devices to malware. Although malware is not a new threat, IoT broadens and amplifies its attack surface. Sophisticated malware exemplified by *Stuxnet* and *Duqu* demonstrate the impressive impact of attacks that target specialized embedded systems. Meanwhile, comparatively simpler DNS-focused (pharming) malware targeting residential routers shows just how close to home IoT attacks can reach.[1] Such threats clearly and urgently motivate a means of verifying that remote IoT devices are in the intended state and behave as expected. In other words, we need to ascertain whether IoT devices are currently *trustworthy*.

*Remote attestation* is a well-known and popular technique for verifying the state of remote computing devices. It appears to be well-suited for the IoT setting. For example, as discussed by Saroiu and Wolman [18], remote attestation can benefit a broad range of IoT-like applications that involve sensors, e.g., participatory sensing and energy consumption monitoring. However, we argue that integrating remote attestation into IoT devices is not a trivial matter. Numerous remote attestation techniques have been proposed over the years. They vary greatly in terms of complexity and adversarial models, as well as security, communication and device feature assumptions. Secure hardware-based techniques (e.g., using a Trusted platform Module (TPM)) are very effective and applicable to high-end devices that can accommodate the additional "real estate", monetary cost and power consumption. They are naturally less appealing to small(er) low-end devices. At the other end of the spectrum are software-based methods that involve minimal overall costs, while offering limited (and often uncertain) security guarantees and imposing other environmental restrictions. In between the two extremes are so-called *hybrid* approaches that blend hardware (e.g., PUFs) and software (e.g., control flow integrity) features; they tend to offer better security than software-based methods, though commensurate with limited security functionality.

This paper has two main goals: (i) to provide a broad overview of attestation in the IoT context, and (ii) to show how various types of attestation can complement one another in this setting. We begin by collating fundamental security requirements for remote attestation (Section 2) which serve as our frame of reference. We then overview current attestation approaches and show that each of them addresses only a part of the overall problem (Section 3). As the core of this work, we identify and justify attestation approaches that are well-suited for IoT. We then highlight their distinct advantages (Section 4). Finally, in Section 5, we present and discuss three new attestation-related challenges that arise in the IoT context: (1) scalability, (2) heterogeneity of attested devices, and (3) abuse of attestation functionality.



*DAC '16, June 05-09, 2016, Austin, TX, USA*



DOI: http://dx.doi.org/10.1145/2897937.2905020

[1] https://blog.trendmicro.com/trendlabs-security-intelligence/dns-changer-malware-sets-sights-on-home-routers

## 2. ATTESTATION PRELIMINARIES

*Attestation* is an interaction between two parties: a *prover* and a *verifier*, through which the latter ascertains the current state and/or behavior of the former. Regardless of the specifics, the objective of attestation is to provide the verifier with sufficient information to make a decision about whether the prover is in a trusted state. *Remote* attestation occurs in a setting where the prover and the verifier are distinct entities, possibly communicating over an open multi-hop network, such as the Internet. There are two main aspects of attestation: (1) the process for obtaining the evidence of the prover's current state (sometimes called the measurement process), and (2) the protocol for conveying that evidence to the verifier. Attestation techniques vary in terms of whether these two aspects are independent or inter-linked.

**Threat & Adversary Model.** The main threat model involves a compromised prover that lies about its current state and/or behavior to the verifier. Therefore, the primary attestation requirement is an *authentic* and *timely* representation of the *current* state and/or behavior of the genuine prover. Although this threat model is very general, it can stem from a variety of adversarial types:

- *Remote Adversary*'s goal is to remotely infect the prover with malware, e.g., as in Stuxnet.
- *Local Adversary* is sufficiently near the prover to be capable of eavesdropping on, and interfering with, the prover's communication.
- *Physical Non-Intrusive Adversary* is physically even closer to the prover, so as to be capable of mounting side-channel attacks.
- *Stealthy Physical Intrusive Adversary* can capture the prover and attempt to physically extract any information (including secrets) stored thereupon.
- *Physical Intrusive Adversary*, besides being able to physically capture the prover, can attempt to modify its state and/or hardware components (e.g., introduce additional memory).

One unifying feature of all aforementioned adversarial flavors is the goal to subvert the prover via one or more of: (1) infecting it with malware, i.e., modifying its software configuration, (2) extracting its secrets, and (3) modifying its hardware configuration.

As discussed below, attestation originated in the context of general-purpose computers: devices that can easily accommodate an additional trusted hardware component (e.g., a TPM) that can withstand attacks by all (or most) adversarial flavors. However, contemporary attestation-related research for *IoT devices* focuses on mitigating the *Remote Adversary*, for three reasons. First, it is the most likely (as illustrated by Duqu and Stuxnet) adversary type and the easiest to defend against, since it operates from afar. Second, it is also the most natural adversary type due to the potentially large scale of its impact.[2] Finally, IoT devices, at least those on the lower-end, can not bear the added costs (including monetary, power, and space) of dedicated secure hardware.

[2]If the adversary is local or physical, its attack scale is limited.

## 3. SECURE HARDWARE-BASED ATTESTATION

Remote attestation originally came to prominence as a feature of the TPM [4]. As standardized by the Trusted Computing Group (TCG), the TPM is a co-processor designed to protect cryptographic keys and record the software state of a computing platform. It achieves this by using a set of special-purpose Platform Configuration Registers (PCRs). Each PCR stores a single cryptographic hash which can be read by external software. Although direct over-writing of a PCR is not allowed, a PCR can be *extended* (via a special command) with a new value by hashing the latter together with the prior PCR value and storing the result in the PCR. During a *measured boot*, each binary in the boot chain computes a hash (a *measurement*) of the subsequent binary, records it in a measurement log, and extends it into the PCRs. The first piece of software is measured by a *trust anchor* (referred to as the "root of trust for measurement" in TCG terminology), typically the platform firmware. Integrity of the measurement log can be checked by comparing the sequence of measured values to current PCR values. PCRs can thus accumulate an unforgeable and easily verifiable chain of values that have been accumulated since the platform was last reset.

Since the PCR values are a representation of the system's state, they can be used as attestation evidence. To assure the verifier that the values are authentic, the TPM creates a *quote*: it signs the PCR values with its attestation key, which is certified to be held by a genuine TPM. To ensure that the quote represents the current state of the system, the verifier can provide a random challenge to be included in the quote. Gasmi et al. [12] discuss how to link such quotes to secure channel end-points.

With respect to the model in Section 2, TPM-based attestation is effective against all, except (perhaps) physical intrusive[3] adversarial flavors.

**Dynamic Root of Trust.** One drawback of the above is the potentially large number of measurements in a TPM quote, which can make verifying a quote time-consuming. For example, on a PC, a quote would include the system firmware, the bootloader, the OS and all running applications. The concept of a Dynamic Root of Trust for Measurement (DRTM) was introduced to address this issue by allowing the chain of measurements to begin at a user-defined point in the platform's operation. In a DRTM *late-launch*, the platform performs a partial CPU reset and resets a subset of the PCRs to an initial value. This prevents any software that was previously run on the platform from interfering with the platform's new state.

**Attestation in SGX.** Extending the idea of DRTM, Intel's recent Software Guard Extensions (SGX) [2] provide a hardware-enforced isolated execution environment (an *enclave*) for application software. The enclave isolates and protects its contents from all other software on the platform, and provides a means of attesting only the software inside the enclave to other enclaves, or other platforms. As with the TPM, attestation evidence consists of a hash of the software, signed by the CPU.

**Property-Based Attestation.** The attestation approaches

[3]This depends on the specifics of the TPM's tamper-resistance features.

described above are referred to as *binary attestation*, since attestation evidence is based on software binaries that have been executed on the platform. Binary attestation is brittle in the sense that any configuration changes or software upgrades result in different hashes of binaries, even if the platform remains in a trustworthy state. Verifiers are therefore required to maintain extensive lists of trusted values. To mitigate this, property-based attestation techniques, e.g. [17, 15], convert measurements of binaries into statements about system properties.

# 4. ATTESTATION OF THINGS

The distinguishing feature of secure hardware-based attestation is the reliance on explicit, purpose-built trust anchors, e.g., a TPM or an SGX-capable CPU. However, resource-constrained devices commonly found in IoT systems are unlikely to have these types of trust anchors due to cost and complexity considerations. First, a TPM increases the costs of materials and integration. Also, platforms that provide firmware TPMs or SGX are likely to be more expensive. Second, using explicit trust anchors requires additional software (e.g. TPM drivers and libraries), which increases overall system complexity. Therefore, although they may be part of the solution, secure hardware-based attestation techniques alone are insufficient in the IoT context. In the following sections we describe more recent attestation techniques designed for constrained devices and discuss how they fit into trust establishment for IoT systems.

## 4.1 Software-based Attestation

Software-based attestation verifies integrity of resource-constrained embedded devices which have no hardware security features to support attestation. Thus, software-based attestation cannot (and does not) assume any secrets on the prover device (since there is no secure place to keep them). It also cannot rely on the prover executing any specific code, since no code integrity can be assured. Instead, software-based attestation exploits side-channel information, such as precise time needed by the prover to perform specific computation.

Many software-based attestation techniques have been proposed in the academic literature. One early example is Pioneer [20]. It computes a checksum of the device memory using a function with run-time side-effects (e.g., status registers), such that any emulation of that function incurs additional timing overhead (i.e., extra delay) that is sufficient to detect cheating. Also, attestation that relies on time-based checksums has been adapted to embedded devices in [21]. However, several attacks on software-based attestation schemes have been demonstrated, e.g., Castelluccia et al. [7].

In general, all current software-based attestation techniques make strong assumptions about adversarial capabilities, and only work if the verifier communicates directly with the prover, with no intermediate hops. There are two reasons for this restriction: (1) a multi-hop path between the prover and the verifier usually incurs variable round-trip delay, thus skewing any timing measurements, and (2) a verifier must be able to detect attacks by a *local adversary* which can assist a malware-infected prover by computing (on the latter's behalf) the correct response to the attestation challenge. If these two limitations do not pose a problem, and referring to Section 2, software-based attestation can defend against all (except physical intrusive) adversarial flavors. In general, though it is the cheapest form of attestation, appealing in some very specific settings (e.g., attestation of legacy computer peripherals), software-based attestation is not viable in a network setting.

## 4.2 Hybrid Attestation

To overcome the aforementioned limitations of software-based attestation, various hybrid architectures have been proposed that employ software/hardware co-design to reduce the "footprint" of attestation on the prover. Hybrid architectures are also motivated by the inability to accommodate dedicated secure hardware (such as a TPM) on low-end devices. The objective of hybrid attestation is to resist all, except physical invasive, adversaries in a network setting (i.e., multiple hops between the prover and the verifier) while minimizing hardware changes.

**Minimal Trust Anchors.** The SMART architecture [9] provides a dynamic root of trust for low-end devices without specialized memory management or protection features. Its software/hardware co-design attempts to minimize hardware changes to the prover device. SMART has four main components: 1) a demarcated *attestation read-only memory* code region in ROM; 2) a *secure key storage* region that can only be accessed from SMART code in ROM; 3) MCU *access controls* that prevent non-SMART code from accessing secure key storage and interrupting SMART code execution; and 4) *reset and memory erasure* if any error is reported by these components. Upon a challenge from the verifier, the ROM-resident attestation code computes a cryptographic checksum of the prover's memory region and returns it to the verifier. SMART's viability has been demostrated via prototypes on two common low-end MCU platforms.

Building on the design of SMART, follow-up work by Francillon et al. [10, 11] presents a more precise specification of minimal hardware features needed to support attestation on low-end provers. The issue of verifier authentication (as well as DoS attacks on the prover) is also highlighted, as discussed in Section 5.2.

TrustLite [13] is a generic security architecture for low-end embedded systems. It allows OS-independent isolation of specific software modules, called *trustlets*. TrustLite introduces the *Execution-Aware Memory Protection Unit* (EA-MPU) as a generalization of simple means of memory protection, such as SMART. Though similar to a memory management unit (MMU), a memory protection unit (MPU) is primarily designed for lightweight access control and does not provide virtual memory. While an MPU makes access control decisions based on accessed memory address, an EA-MPU is *execution-aware* – it also considers the address of the currently executing instruction.

In TrustLite, an EA-MPU enforces that a given trustlet's data can only be accessed by that trustlet's code. This mechanism can also be used to control access to hardware peripherals. Unlike SMART, TrustLite also handles memory access violations and hardware interrupts. TrustLite can be instantiated in several configurations, and can scale from providing a simple protected firmware runtime to more advanced functionality, such as attestation and trusted execution of user-space tasks.

The EA-MPU concept is also used in the more recent TyTAN system [5]. Compared with TrustLite, TyTAN provides more flexibility by enabling dynamic loading and un-

loading of multiple tasks at runtime, real-time scheduling guarantees and secure interprocess communication (IPC), with sender and receiver authentication.

**PUF-based attestation.** As discussed earlier, software-based attestation is only effective in restricted settings where the prover and the verifier communicate directly. Moreover, it cannot explicitly authenticate underlying hardware components of the prover, making it vulnerable to impersonation attacks [19]. Meanwhile, in the presence of physical stealthy or physical invasive adversaries, even hybrid attestation methods (such as SMART or TrustLite) are ineffective since device keys can be obtained and the prover can be impersonated and/or cloned.

A Physically Unclonable Function (PUF) is physical structures that generates a unique hardware-specific output (*response*) to each input (*challenge*), by exploiting side-effects in chip manufacturing. PUFs are a promising technology that binds attestation to the specific physical hardware of a particular prover [19]. Thus, PUFs can be used instead of access-restricted device keys in hybrid architectures. For example, [14] extended the lightweight PUF remote attestation scheme of [19] and presented the design and implementation of the ALU PUF, a novel minimalist hardware trust anchor based on manufacturing variations in commodity processors.

However, PUFs typically offer only limited resilience against operational and environmental influences, such as temperature, power supply variations or silicon aging effects. Hence, it is necessary to integrate effective error correction mechanisms at the prover while minimizing the PUF's hardware overhead. Furthermore, current PUF-based techniques typically require: (1) a costly enrollment phase for each PUF, and (2) maintaining a large database of PUF challenges and responses (needed for authentication).

## 4.3 Control-Flow Attestation

Trustworthiness of code also pertains to the runtime behavior of attested code. Static attestation discussed in Section 3), although efficient, only provides assurance of the integrity of binaries and not of their execution. Static attestation completely fails to capture software corruptions that hijack a program's control flow. Such attacks tamper with state information on the application's stack or heap to arbitrarily divert the execution flow while the binaries remain the same. Today's state-of-the-art memory corruption attacks (or runtime exploits) use code-reuse techniques, such as return-oriented programming (ROP), that dynamically generate malicious programs based on code snippets (gadgets) in benign binaries, *without* injecting any malicious code [16]. Consequently, measurements computed over static binaries remain unchanged.

A more fine-grained approach would be a hybrid attestation approach combining static and runtime attestation. We are currently working on a runtime attestation scheme which provides precise attestation of the execution path of a program for the case of embedded devices [22]. The execution path of an application can be encoded either as a trace of taken branches, executed instructions, function calls, or a cumulative hash over the execution path. In either case, the execution path serves as a fingerprint of software execution allowing a remote device to attest the dynamic execution state of running processes. This represents an important step towards tackling the open problem of runtime attestation.

Runtime attestation has several benefits over other runtime mitigation technologies, such as control-flow integrity (CFI) [1]. CFI is enforcement-based, ensuring that no control-flow violation has occurred. However, runtime attestation requires reporting the executed control-flow path rather than only whether an attack occurred. Also, control-flow attestation captures non-control-data attacks [8] that corrupt data values (e.g., authentication variables) to follow a privileged path in the control-flow graph (CFG) without violating the legitimate CFG. Finally, it is context-sensitive since it captures the control flow from beginning to end while CFI typically validates a branch without considering previous flows.

# 5. FURTHER CHALLENGES

The IoT context highlights some further important challenges for attestation. First, attestation mechanisms must be sufficiently scalable to handle a potentially large number of devices in an IoT system, e.g., in a factory, or a building automation scenario. On a related note, a large IoT system might include a range of devices, each with a distinct set of hardware security features. Thus, attestation of multiple *heterogeneous* devices, becomes the next challenge. Also, the presence of attestation support on resource-constrained low-end IoT devices can be exploited, e.g., via verifier impersonation and/or denial of service attacks against honest provers. We discuss some potential means of addressing these challenges in the following sections.

## 5.1 Scalability

Most attestation techniques, including those described in the preceding sections, operate in a setting with one prover and one verifier. However, many current and emerging scenarios require a verifier to attest a (possibly large) group of provers. For example, in industrial control systems, many autonomously operating devices collaborate to monitor and control safety-critical processes [3]. Such systems are often referred to as *device swarms*. The difficulty of attesting a device swarm is compounded by potentially dynamic topology and membership, e.g. an ad-hoc vehicular network where nodes move around as well as join and leave the swarm.

The first attempt to provide attestation for swarms is *SEDA:* Scalable Embedded Device Attestation [3]. The design of SEDA was informed by two requirements [3]: (1) the attestation scheme must verify the integrity of the swarm as a whole, in a more efficient manner than attesting each individual device, and (2) it must be independent of the underlying integrity measurement mechanism(s) used by individual devices. The latter allows the swarm attestation scheme to support multiple single-prover attestation techniques, such as those described in Section 4.

After an initial setup phase, a device (e.g., $D_8$ in Figure 1) can join swarm $\mathcal{S}$ or change its position using the `join` protocol, which establishes pairwise attestation keys with each of its neighbours. The principal idea behind SEDA is that attestation is performed recursively throughout the swarm, starting with any initiator node (e.g. $D_1$) selected by the verifier. The verifier sends $D_1$ an unpredictable challenge $N$, and $D_1$ generates a new global session identifier $q$.

Using the `attdev` protocol, $D_1$ sends an attestation request containing $N$ and $q$ to each of its neighbours. When a node receives an `attdev` request with a new $q$, it forwards this to all of its own neighbours and awaits their responses. This way, the request propagates along the edges of a span-

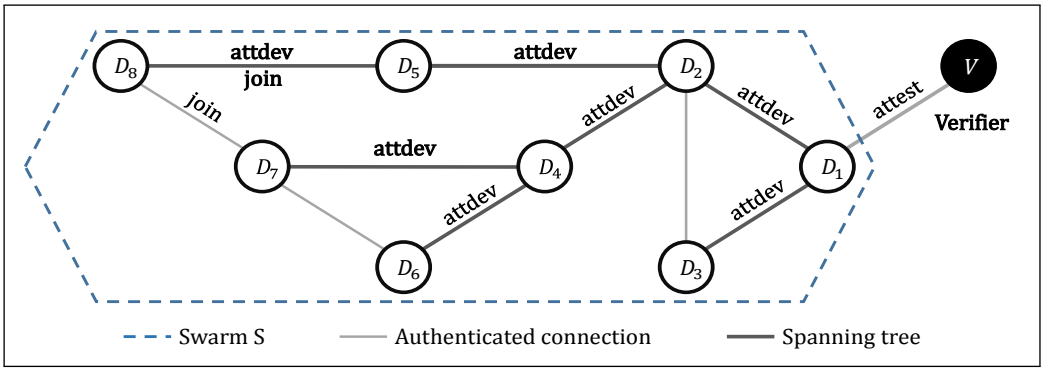


**Figure 1: Swarm attestation (adapted from [3])**

ning tree, rooted at $D_1$. After all neighbours respond or time-out, the node aggregates their responses with its own attestation report and returns the result to the parent. Attestation reports are thus accummulated as they flow from the leaves of the tree towards the root, $D_1$. This is clearly more efficient than attesting each individual node. Once the aggregated attestation reaches $D_1$, the aggegated response is sent to the verifier. To demonstrate its viability, SEDA has been successfully prototpyed over SMART and and TrustLite architectures; both discussed in Section 4.2.

## 5.2 The Prover's Perspective

Remote attestation techniques typically consider a compromised prover, while assuming that the verifier is trusted. However, in a real-world scenario, the verifier can be impersonated by the adversary, resulting in unauthorized invocation of attestation functionality on the prover and leading to, e.g., DoS attacks. To this end, Brasser et al. [6] analyzed the potential impact of abusing remote attestation (on the prover) as a vector for DoS attacks. For example, an attestation scheme that computes a message authentication code (MAC) over the entire memory of the prover running on a typical low-end MCU could consume about 754 ms per attestation request [6]. This is an important problem, largely ignored by prior attestation schemes. It occurs in part due to significant asymmetry in the amount of work performed by the prover and the verifier, as well as asymmetry in their computational abilities (since the verifier is generally a more powerful entity).

Brasser et al. [6] proposed several simple techniques to mitigate DoS attacks on the prover. Intuitively, the verifier must authenticate itself to the prover, using using either public or symmetric key cryptography. However, since the former is computationally intensive, its use could exacerbate the problem by providing yet another vector for DoS attacks. Thus, symmetric cryptography should be used. However, the prover must also detect replays of previous legitimate attestation requests. There are several standards for this: nonces, counters and timestamps. Using nonces is unworkable, since a large amount of non-volatile storage is required to keep track of previous nonces. Monotonically-increasing counters also require non-volatile storage, albeit only a small amount. (They also cannot help in detecting delayed requests). Although timestamps offer the best security, their use requires the prover to have a reliable real-time clock.

Brasser et al. [6] also considered a stronger *roaming adversary* model, which can temporarily compromise the prover and alter its dynamic state. For example, a roaming adversary can reset the prover's counters or change the prover's clock. The proposed counter-measure involves execution-aware memory access control (EA-MAC) to protect the counters, clock and other critical aspects of the prover's attestation functionality. An experimental evaluation shows that this can be achieved with low additional costs.

# 6. CONCLUSION

Remote attestation can mitigate some important security threats in IoT systems. Secure hardware-based attestation techniques are not well-suited for the IoT context, since IoT devices: (1) can be resource-constrained, (2) may not have specialized hardware *trust anchors*, and (3) often operate in (possibly large and heterogeneous) groups. Meanwhile, software-based attestation techniques, while appropriate for very specific settings, are not applicable in remote scenarios, i.e., where the prover and the verifier are not communicating directly. Fortunately, other types of attestation protocols can fill this gap. In this paper, we surveyed some attestation techniques that might be suitable for the IoT context. We also overviewed recent results that overcome specific IoT attestation challenges, such as scalability as well as protection against malicious verifiers and DoS attacks. We believe that the presented approaches are likely to play a role in trust establishment in IoT systems. They also form a solid foundation for further research in this area. There remain several open challenges, including (1) new system architectures to efficiently attest groups (or swarms) of *heterogeneous* devices, (2) new schemes to attest runtime properties and behavior, and (3) robust trust anchors for IoT systems.

# 7. ACKNOWLEDGEMENTS

This work was supported in part by the German Science Foundation (project S2, CRC 1119 CROSSING), European Union's Seventh Framework Programme (609611), Academy of Finland (283135), National Security Agency (H98230-15-1-0276), Department of Homeland Security (under subcontract from the HRL Laboratories), National Science Foundation (CNS-1059416), and a Multidisciplinary University Research Initiative grant (FA9550-14-1-0351/Rice 14-0538).

# 8. REFERENCES

[1] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-flow integrity: Principles, implementations,

and applications. *ACM Trans. Inf. Syst. Secur.*, 13(1), 2009.

[2] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. Innovative Technology for CPU Based Attestation and Sealing. In *Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[3] N. Asokan, F. Brasser, A. Ibrahim, A.-R. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann. SEDA: Scalable Embedded Device Attestation. In *ACM Computer and Communications Security (CCS)*, 2015.

[4] B. Balacheff, L. Chen, S. Pearson, D. Plaquin, and G. Proudler. *Trusted computing platforms: TCPA technology in context*. Prentice Hall Professional, 2003.

[5] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl. TyTAN: Tiny Trust Anchor for Tiny Devices. In *Design Automation Conference (DAC)*, 2015.

[6] F. Brasser, A.-R. Sadeghi, K. B. Rasmussen, and G. Tsudik. Remote Attestation for Low-End Embedded Devices: the Prover's Perspective. In *Design Automation Conference (DAC)*, 2016.

[7] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente. On the difficulty of software-based attestation of embedded devices. In *ACM Computer and Communications Security (CCS)*, 2009.

[8] S. Chen, J. Xu, E. C. Sezer, P. Gauriar, and R. K. Iyer. Non-control-data attacks are realistic threats. In *USENIX Security Symposium*, 2005.

[9] K. El Defrawy, A. Francillon, D. Perito, and G. Tsudik. SMART : Secure and Minimal Architecture for ( Establishing a Dynamic ) Root of Trust. In *NDSS*, 2012.

[10] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik. Systematic Treatment of Remote Attestation. *Cryptology ePrint Archive*, 2012.

[11] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik. A Minimalist Approach to Remote Attestation. In *Design, Automation & Test in Europe (DATE)*, 2014.

[12] Y. Gasmi, A.-R. Sadeghi, P. Stewin, M. Unger, and N. Asokan. Beyond secure channels. In *ACM workshop on Scalable trusted computing (STC)*, 2007.

[13] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan. TrustLite: A Security Architecture for Tiny Embedded Devices. In *European Conference on Computer Systems (EuroSys)*, apr 2014.

[14] J. Kong, F. Koushanfar, P. K. Pendyala, A.-R. Sadeghi, and C. Wachsmann. PUFatt: Embedded Platform Attestation Based on Novel Processor-Based PUFs. In *Design Automation Conference (DAC)*, 2014.

[15] K. Kostiainen, N. Asokan, and J.-E. Ekberg. Practical property-based attestation on mobile devices. In *International conference on Trust and Trustworthy Computing (TRUST)*, 2011.

[16] R. Roemer, E. Buchanan, H. Shacham, and S. Savage. Return-oriented programming: Systems, languages, and applications. *ACM Trans. Inf. Syst. Secur.*, 15(1):2:1–2:34, 2012.

[17] A.-R. Sadeghi and C. Stüble. Property-based Attestation for Computing Platforms: Caring about properties, not mechanisms. In *Workshop on New Security Paradigms (NSPW)*, 2005.

[18] S. Saroiu and A. Wolman. I am a sensor, and I approve this message. In *HotMobile*, New York, New York, USA, 2010.

[19] S. Schulz, A.-R. Sadeghi, and C. Wachsmann. Short Paper: Lightweight Remote Attestation using Physical Functions. In *ACM conference on Wireless network security (WiSec)*, 2011.

[20] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: Verifying integrity and guaranteeing execution of code on legacy platforms. In *Symposium on Operating Systems Principles (SOSP)*, 2005.

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

[22] T. Abera et al. C-FLAT: Control flow attestation for embedded systems software. Work in Progress, 2016.