Towards Systematic Design of Collective Remote Attestation Protocols

Ivan De Oliveira Nunes, Ghada Dessouky, Ahmad Ibrahim, Norrathep Rattanavipanon, Ahmad-Reza Sadeghi and Gene Tsudik

Abstract—Networks of and embedded (IoT) devices are becoming increasingly popular, particularly, in settings such as smart homes, factories and vehicles. These networks can include numerous (potentially diverse) devices that collectively perform certain tasks. In order to guarantee overall safety and privacy, especially in the face of remote exploits, software integrity of each device must be continuously assured. This can be achieved by Remote Attestation (RA) – a security service for reporting current software state of a remote and untrusted device. While RA of a single device is well understood, collective RA of large numbers of networked embedded devices poses new research challenges. In particular, unlike single-device RA, collective RA has not benefited from any systematic treatment. Thus, unsurprisingly, prior collective RA schemes are designed in an ad hoc fashion.

Our work takes the first step toward systematic design of collective RA, in order to help place collective RA onto a solid ground and serve as a set of design guidelines for both researchers and practitioners. We explore the design space for collective RA and show how the notions of security and effectiveness can be formally defined according to a given application domain. We then present and evaluate a concrete collective RA scheme systematically designed to satisfy these goals.

I. INTRODUCTION

An increasing number and variety of embedded (and IoT) devices are being deployed and connected to the Internet. In 2020, 50 millions of such devices are expected to operate across the globe [9] performing safety- and security-critical operations (e.g., sensing and actuation), and exchanging private information. IoT devices are expected to work collaboratively and autonomously in large systems designed to provide various services and enable new applications. They have been utilized in different smart domains, including: homes, offices, factories, warehouses, public venues, and vehicles. Examples of current IoT devices include smart door locks, WiFi lighting, Bluetooth trackers, smoke/CO₂ detectors, and smart switches.

Due to the limited resources imposed by cost, size, and energy constraints, IoT devices lack necessary security features to defend against attacks, such as memory management and anti-malware mechanisms. Furthermore, these devices represent attractive targets for remote attacks due to their interconnectivity. In fact, IoT devices have already been targeted by reverse engineering [15, 17] and malware infestation attacks [14, 37]. In recent years, Stuxnet [36] demonstrated the gravity and physical damage inflicted by large-scale malware attacks, while Mirai [4] used malware-infected IoT devices (routers, DVRs and cameras) to mount massive-scale Distributed Denial-of-Service (DDoS) attacks.

Remote malware attacks aim to compromise devices by modifying their software to a malicious state without requiring physical access (or close proximity) to these devices. To detect such attacks, various Remote Attestation (RA) schemes have been proposed. RA is an interaction between two parties: an untrusted and potentially compromised prover, and a remote verifier that aims to measure the former’s current software integrity. Collective Remote Attestation (cRA) is a means of securely and efficiently applying RA to multitudes of interconnected provers, i.e., large IoT networks or swarms. Next, we review existing cRA protocols and motivate the need for a methodology to systematically design them.

A. Background & Related Work

Single-Device Attestation architectures fall into three categories: software-based, hardware-based, or hybrid. Security of software-based attestation [18, 21, 26, 32–34] relies on strong assumptions about precise timing and constant communication delays, which are unrealistic in the IoT ecosystem. Hardware-based methods [25, 27, 28, 30, 31, 35] rely on security provided by dedicated hardware components (e.g., TPMs [35]). However, the cost of such hardware is normally prohibitive for lower-end IoT devices. Hybrid RA [5, 11–13, 16, 22] aims to achieve security equivalent to hardware-based mechanisms, yet with lower hardware cost. It leverages minimal hardware support while relying on software to reduce the complexity of additional hardware. Systematization of collective attestation presented in this paper focuses on hybrid architectures, since they are the most suitable for large groups of low-end IoT devices [1].

Collective Remote Attestation aka cRA aims to scale single-device attestation to large IoT networks. SEDA [3] is the first cRA protocol that allows a central verifier to assess trustworthiness of up-to-million-device networks in order of seconds. To achieve this, attestation burden is distributed across the network, allowing neighbors to attest each other and to aggregate attestation results. SANA [2] includes a novel cryptographic primitive, Optimistic Aggregate Signatures (OAS), to construct a cRA protocol, which is more resilient to physical and DoS attacks. LISA [7] proposes two protocols for synchronous and asynchronous collective attestation, while reducing implementation, bandwidth, and computation complexity as well
as defining metrics for quality of cRA, DARPA [19] extends cRA to support detection of physical node capture attacks using a periodic presence confirmation. SCAPI [23] improves on DARPA by reducing bandwidth complexity and providing robustness against network delays. SALAD [24] is a cRA scheme for highly dynamic device topologies.

B. Problem Statement & Contributions

All aforementioned cRA schemes are designed in an ad hoc fashion, i.e., based on identifying efficiency [3], and/or security [2],[19] problems of their predecessors, and informal selection of features to construct a solution. Their key shortcomings are: (1) no concrete security guarantees; (2) not designed in a systematic or principled manner; and (3) lack formal definitions for assumptions and guarantees makes it is hard to decide suitability for a particular use-case or application.

Contributions. We take the initial step toward placing cRA onto solid ground and present the first model to analyze the cRA efficiency, soundness, and security requirements. Further, we construct a concrete scheme that adheres to these requirements. Our contributions are three-fold:

1) Formal Model for cRA: We define an example cRA use-case and construct a corresponding formal model (TCA-Model) encompassing desirable efficiency, soundness, and security notions for such a use-case. TCA-Model is the first formal model for analyzing cRA requirements.

2) Systematic Design: We construct the first systematically designed cRA protocol (SAP) and show that it adheres to the desirable requirements defined by TCA-Model.

3) Proof-of-Concept & Performance Analysis: We present a prototype implementation of SAP and empirically evaluate its performance via simulation.

We anticipate that our methodology would serve as guidelines for defining new cRA models (by modifying or extending TCA-Model), to formally define and systematically design other cRA protocols targeting different applications with various requirements.

II. DESIGN SPACE FOR COLLECTIVE ATTESTATION

We now explore the design space of cRA. There are many inter-related design parameters that should be ideally taken into consideration when designing a cRA protocol. Some are fundamental and explicit, i.e., they directly impact protocol design choices, while others are subtle and specific to the use-case in question. We refer to these parameters as principal and secondary, respectively.

Principal Parameters. We first consider network configuration and topology. Assuming a static topology simplifies protocol design in certain security aspects, such as key management among adjacent devices due to mobility. Whereas, cRA for a dynamic topology network requires additional assumptions on the network setting, e.g., a routing protocol and/or an upper bound on the communication delay between any two devices, which is necessary to determine protocol timeouts for instantiating cRA. On the other hand, it is also important to consider concrete device properties. For example, device homogeneity (in both hardware and software terms) makes attestation simpler, while heterogeneity requires considering different times to complete attestation on each device type; this is also relevant for estimating timeouts and vulnerability windows in case of TOCTOU attacks.

Another principal parameter is the adversarial model. Most cRA protocols assume a remote adversary, only capable of modifying binaries. Given a collection of connected devices, the adversary might be capable of physically attacking one or more of them. However, mitigating physical attacks greatly complicates protocol design, as shown in [19], as well as considering more sophisticated runtime attacks.

Secondary Parameters. Some network and device properties might appear cRA-irrelevant, e.g., network connectivity (redundancy), device architecture, and real-time (or safety-critical) nature of devices. However, such properties can indeed influence design choices. cRA functionality is typically realized in an uninterruptible process, which takes a device away from performing its original task for a non-negligible amount of time. Such distractions might be unacceptable for real-time or safety-critical systems that cannot dedicate a valuable time slot to attestation. Thus, we must take this aspect into consideration, e.g., by enabling interrupts or minimizing attestation time. Device connectivity might be useful in safety-critical setting, allowing parallel attestation of multiple devices, while their peers take care of safety-critical tasks. Network redundancy can also influence the overall resiliency to attacks, e.g., maximum tolerated number of physically compromised devices.

cRA design is also influenced by non-security of hardware and/or software features, e.g., single- vs multi-core architectures, support for direct memory access (DMA) and ability to host a hypervisor or a microkernel. Different hardware architectures might require different hardware security features for cRA functionality. Also, device software stack imposes additional requirements on attestation, e.g., whether all traces of secret key(s) are securely erased.

On the other hand, design choices are directly related to the desired cRA outcome, which are in turn influenced by the exact requirements in the target use-case. One important attestation metric is the so-called quality of attestation (QoA) [7]. It encompasses a broad spectrum of choices, ranging from a binary flag that reflects the entire network’s state (good/bad) to explicit state of individual device.

Finally, there is the usual trade-off between security, efficiency, and scalability, which is determined by the specific use-case. While security is characterized by the adversarial model described above, performance and scalability include runtime, power consumption, computational, memory, and communication overheads. Tiny devices with 8-bit processors might not have sufficient secure (restricted-access, non-volatile) memory for storing large public/private keys, or enough computational power for public key operations. Battery-powered devices must minimize power consumption due to computation and communication. Safety-critical systems need low network congestion.
discusses how the same
Description discusses how TCA-
Time interval between IV
Arbitrary function polynomial in 
Arbitrary function negligible in 
Time taken by a given device to compute 
Attestation token of 
provides a generic 
Set of valid states for 
Attestation token of 

Finally, a cRA scheme can be systematically designed and proved to achieve these goals.

to ensure timeliness of critical messages.

Based on the above, many parameters can influence cRA design. We only discussed a few, since considering all would lead to combinatorial explosion of possible use-case scenarios. More importantly, a single formal model cannot account for all of such parameters. Through the rest of this paper, we show how to construct a formal model for a specific cRA use-case and how a cRA protocol can be systematically designed accordingly. Section VIII discusses how the same methodology can be adapted and applied to different use-cases.

III. cRA PROTOCOLS

We informally characterize Collective Remote Attestation (cRA) as a scheme involving: (1) group of devices \( S \), and (2) trusted verifier \( Vrf \). The goal of cRA is to allow \( Vrf \) to measure trustworthiness of \( S \), i.e., obtain authenticated software integrity measurement(s) from \( S \)'s members, or a subset thereof. Definition 1 provides a generic cRA specification.

Section IV defines TCA-Model, along with assumptions and goals. Later, in Section V, we systematically design a cRA protocol that adheres to 3 notions: (1) TCA-Efficiency – sets performance requirements for device and network models, and delay/communication constraints, (2) TCA-Soundness – defines the expected result of honest cRA execution in TCA-Model, and (3) TCA-Security – models adversarial capabilities using a security game.

IV. TCA-MODEL

Systematically designing a cRA scheme involves first defining an abstract machine model for \( S \) devices and a network model compatible with the application in the context of which devices are to be deployed. Such abstract models are necessary to formally reason about the effect of design choices in a specific application domain. Then, one can formulate security and performance requirements according to application goals. Finally, a cRA scheme can be systematically designed and proved to achieve these goals.

To illustrate one possible systematic design, we next define the Timely Collective Attestation Model (TCA-Model) composed of three formal notions: TCA-Efficiency, TCA-Soundness, and TCA-Security. The purpose of TCA-Model is to introduce (by example) a methodology for systematically designing and proving properties of cRA protocols. For the sake of clarity, TCA-Model targets a simple use-case, considering a static topology and homogeneous devices. As the name suggests, we require attestation to run synchronously on all devices, i.e., all devices execute attest at the same time. The final verification result is a binary flag: either stating that the entire \( S \) is healthy or that at least one device is compromised (or unresponsive, or failed). The network is assumed to be reliable, unless it is under adversarial influence. We also assume that entire program memory of all devices is to be attested. Section VIII discusses how TCA-Model could be extended to incorporate different dimensions of cRA. Our notation is reflected in Table I.

A. TCA Machine Model

In TCA, we choose to model groups of low-end devices based on low-power, single-core microcontrollers. We model features of such a device as Random Access Machines (RAM) comprising two components: Memory \( M \), and one Central Processing Unit CPU. CPU includes a set of registers \( R \) (as well as Program Counter – PC) and an Instruction Set (IS). Communication between \( M \) and CPU occurs in fetch-execute cycles, which are referred to as execution rounds. In a given round, the address within \( M \) pointed by PC determines the instruction \( i \in IS \) to be executed along with the \( i \)'s inputs/outputs, when applicable. We assume there is no Direct Memory Access (DMA); thus, modifications to \( M \) and \( R \) occur through CPU instructions. A valid instruction \( i \in IS \) can be of three types:

1) **Read:** Modify \( R \)'s state based on \( M \)'s current values.
2) **Write:** Modify \( M \)'s state based on \( R \)'s current values.

The only change to \( R \) is incrementing PC to point to the
next instruction.

3) Execute: Modify R’s state based on the current values in R itself. Those reflect algebraic operations (e.g., add and mul) as well as interrupts and branching operations that modify PC. (Recall that PC ∈ R).

M consists of four sections:

1) DMEM - standard random access memory, including General Purpose Input/Output (GPIO).

2) PMEM - standard executable code program memory.

3) ROM - read-only memory.

4) ProMEM - protected memory that can only be read/written according to access control policies. These policies are implemented and enforced by trusted hardware (or operating system) which monitors, at each execution cycle, PC and M locations accessed by CPU.

B. TCA Network Model

In cRA, the network comprises a large number of interconnected members, which interact and collaborate to achieve a certain task. We denote an Attestation Group with N members by S = m1, · · · , mN. We consider homogeneous groups of devices, and thus all members are assumed to abide the same machine model:

∀m i ∈ S : {m i = RAM[CPU, M]} (1)

where RAM[CPU, M] denotes the machine model described in Section IV-A. Vrf is the trusted entity responsible for assessing S’s trustworthiness. Any entity that is interested in S’s services, and thereby wants to assess its trustworthiness, must do so through Vrf. Let degree(mi) denote the number of members directly connected to mi, sPath(mi, mj) denote the number of hops in the shortest path between mi and mj, and rate(mi, mj) denote the transmission rate in bits/sec, between two directly connected nodes. TCA network model assumes:

∀m i ∈ S : degree(mi) > 0 (2)

∀m i ∈ S : {∃j ∈ [1, N−1] : sPath(mi, Vrf) = h} (3)

∀m i ∈ S : {sPath(mi, mj) = 1 → rate(mi, mj) = µ} (4)

In other words, all members can communicate with at least one neighbor; there exists a path between Vrf and all mi ∈ S, and the transmission rate between two directly connected members is constant and equal to µ. Propagation, queuing, and processing delays are assumed negligible. Hence, communication delay is dominated by transmission delays. Messages are routed through the shortest path. Thus, the delay to transmit d bits between two arbitrary members in S is:

delay(mi, mj, d) = sPath(mi, mj) × d/µ (5)

C. TCA-Efficiency

Efficiency requirements are the fundamental difference between cRA and RA. If efficiency parameters such as topology restrictions, verification delays, and network utilization are not taken into account, one can simply design a secure cRA protocol by having Vrf individually attest each member in S. In TCA-Model, we consider three efficiency metrics: cRA delay, network utilization, and node degree.

The first represents the overall time for S to complete one execution of the cRA protocol. Let taṭt denote the time when attest is supposed to execute synchronously on all mi ∈ S and trexp denote the time when Vrf receives Hs. We define cRA delay TCA as the elapsed time between these two times:

TCA = trexp − taṭt (6)

It is worth noting that TCA excludes the time for Vrf to verify Hs, because the report verification can be done off-line. Vrf is assumed to know a set of valid states (VS) of all mi’s, and thus can use this information to pre-compute the expected attestation token for S.

The Network Utilization, denoted as UC,A, is defined as the total amount of data (in bits) transmitted across all links in S during the interval between the time Vrf issueschal (tchal) and the time it receives Hs (trexp). Let L denote all direct links between two devices in S and D(l, t0, te) be the amount of data (in bits) transmitted through link l between times t0 and te. UC,A is defined as:

UC,A = ∑ ∀l ∈ L D(l, tchal, trexp) (7)

Using these metrics, we establish the TCA-Efficiency goals in Definition 2.

Several other requirements might be of interest when defining cRA efficiency goals, such as power consumption, link utilization, memory utilization, and geographical coverage of the swarm, which are very use-case-specific. In TCA-Model, we select a set of typical metrics that are desirable for most domains. Naturally, as the number of efficiency requirements grows, the more challenging the protocol design becomes.

D. TCA-Soundness and TCA-Security

We consider an adversary, Adv, that can control the entire software state (i.e., code and data) of any number of devices in S. Adv can modify any writable memory and read any memory (e.g., secrets) that is not protected by access control rules (ProMEM). It might also re-locate malware from one memory segment to another, in order to evade detection. TCA goal is to allow Vrf to measure the software state of all devices in S synchronously at a time of choice: tatt, TCA does not require Vrf to be able to identify which device is infected, but only to ascertain trustworthiness of S as a whole (see Section VIII for discussion on Quality of Attestation).

1) Notation: Without loss of generality, we use m(mi, l) to denote the content of mi’s entire memory at time t. Conversely, PMEM(mi, t) refers specifically to the contents of mi’s PMEM. The same notation extends to DMEM, ROM and ProMEM. We
use $PC \in \text{PMEM}$ to denote that $PC$ points to a memory cell inside PMEM. This notation (e.g., $PC \in \text{ProMEM}$) applies to the other memory sections. The contents of $\text{PMEM}$ of all $S$ members is denoted as $\text{PMEM}^i(S, t)$ and $\text{PMEM}^i(S, t) = \{\text{PMEM}(m_1, t), ..., \text{PMEM}(m_N, t)\}$. In addition, $\text{PMEM}^i(S, t) \equiv VS$ is used to denote that $S$ is in one of the possible valid states according to $VS$. $\text{PMEM}^i(S, t) \neq VS$ means at least one member is not in a valid state. Lastly, $\neg\text{Adv}$ denotes a cRA protocol execution without adversarial intervention, i.e., all members are in a benign state ($\text{PMEM}^i(S, t) \equiv VS$) and $\text{Adv}$ does not interfere with the network communication.

2) Definitions: We now formally define TCA soundness and security subsuming the aforementioned goal and adversarial model. Definition 3 specifies that verification should succeed with overwhelming probability as long as $\neg\text{Adv}$ does not interfere with the communication between $S$ and Vrf and does not compromise any $m_i \in S$. Definition 4 captures the adversarial capabilities in a security game.

\begin{definition}[TCA-Soundness] Let $H_S$ be a cRA report received by Vrf in response to the execution of request. A cRA scheme is TCA-Sound iff:
$$\Pr[\neg\text{verify}(H_S, VS) = 0 | \neg\text{Adv} < \text{negl}(l)]$$
where $\text{negl}$ is a negligible function, and $l$ is the security parameter.
\end{definition}

\begin{definition}[TCA-Security] A protocol is TCA-Secure if there is no ppt $\text{Adv}$ capable of winning TCA-Security-Game with probability $\Pr[\text{Adv, TCA-Security-Game} > \text{negl}(l)]$.
\end{definition}

V. SAP: Systematic Design in TCA-Model

Based on the formal definition in TCA-Model we can now reason about the goals of our cRA protocol at its design. In this section we systematically design SAP = (setup, request, attest, report, verify): a cRA protocol compliant with TCA-Model. SAP stands for Synchronous Attestation Protocol and is illustrated in Figure 1. In Section VI, we prove that SAP is TCA-Efficient, TCA-Sound, and TCA-Secure according to the definitions of Section IV. We now proceed to the specification of SAP sub-protocols.

A. setup:

- **Key Distribution:** At deployment time, one symmetric key $K_{m_i, \neg\text{Vrf}}$ is generated for each $m_i \in S$, securely provisioned to $m_i$, and shared with Vrf.

- **Topology:** All $m_i \in S$ are organized in a balanced binary tree topology rooted on Vrf. Each device communicates with up to two children and one parent.

- **Valid States (VS):** $S$ is considered in a valid state at time $t$ if $\forall m_i \in S : PRMC(m_i, t) = \text{cfg}_i$. Value: $VS = \{\text{cfg}_1, ..., \text{cfg}_N\}$ (8)

B. request:

- **Challenge Generation:** Vrf chooses a time $t_{\text{att}}$ set $\text{chal} = t_{\text{att}}$ and forward $\text{chal}$ to its children. $t_{\text{att}}$ determines the point in time where all $m_i \in S$ should call $\text{attest}$ routine. Let $t_{\text{chal}}$ denote the time when Vrf forwards $\text{chal}$. To allow enough time for the challenge propagation, Vrf’s choice for $t_{\text{att}}$ is restricted to:
$$t_{\text{att}} > t_{\text{chal}} + (\text{log}(2(N + 2) - 1)) \times |\text{chal}|/\mu \quad (9)$$

where $|\text{chal}|$ denotes the size of $\text{chal}$ in bits and $|\text{chal}| = O(l)$.

- **Challenge Propagation** (executed by all $m_i \in S$):
  1) Upon receipt of $\text{chal}$ from a parent, forward $\text{chal}$ immediately to all children.
  2) Schedule a call to $\text{attest}$ implementation at time $t = \text{chal}$.

C. $\text{attest}^m_i$:

- $\text{time} = \text{readSecureClock}()$
- If ($\text{chal} \neq \text{time}$), $h_{m_i} = 0$.
- Otherwise: $h_{m_i} = \text{HMAC}_{K_{m_i, \neg\text{Vrf}}}(\text{PMEM}(m_i, \text{chal})|\text{chal})$.
- $|h_{m_i}| = l$. ($l$ is the security parameter)

$\text{attest}$ implementation relies on three sub-properties that must be guaranteed by cRA design:

1) **Network-Wide Secure Synchronized Clock:** All devices in $S$ including Vrf must be synchronized in time to schedule $\text{attest}$ invocation to the same time ($t = \text{chal}$) at the end of request. The clock module used for synchronization must be read-only, such that malware can not spoof $\text{readSecureClock}()$ result.

2) **Local Attestation Correctness:** execution of $\text{attest}$ must result in the correct computation of $h_{m_i}$. This includes ensuring (1) that the HMAC is properly implemented, and (2) that the result of the HMAC computation $h_{m_i}$ is temporally consistent with the attested memory [8], i.e., the resulting $\text{HMAC}$ must reflect a snapshot of PMEM (in its entirety) at the time when $\text{attest}$ was invoked (i.e., $\text{PMEM}(m_i, t = \text{chal})$).

3) **Local Attestation Security:** Output of $\text{attest}$ execution must be unforgeable. This requires that: (1) $K_{m_i, \neg\text{Vrf}}$ can never be leaked and can only be used by $\text{attest}$ and (2) $\text{chal}$ is not repeated, implying a sufficiently large buffer for a monotonically increasing (secure and synchronized) clock.

Sections VI and VII discuss hardware and software features for $S$ members to achieve these sub-properties.
1) If \( m_i \) is a leaf: Set \( \text{token}_m = h_m \) and forward it to parent.

2) Otherwise: upon receiving children’s tokens, \( \text{token}_{\text{child}_1} \) and \( \text{token}_{\text{child}_2} \), compute 
\[ \text{token}_m = h_m \oplus \text{token}_{\text{child}_1} \oplus \text{token}_{\text{child}_2} \] 
forward \( \text{token}_m \) to parent.

- Executed by Vrf: upon receiving \( \text{token}_{\text{child}_1} \) and \( \text{token}_{\text{child}_2} \), compute 
\[ H_S = \text{token}_{\text{child}_1} \oplus \text{token}_{\text{child}_2} \].

E. verify:

- Compute:
  1) \( \forall i \in \{1, ..., N\} \), \( \text{res}_i = HMACK_{n_i \rightarrow \text{Vrf}}(\text{cfg}_i | \text{chal}) \)
  2) \( \text{RES}_S = \text{res}_1 \oplus ... \oplus \text{res}_N \)

- If \( H_S = \text{RES}_S \), output 1. Otherwise, output 0.

VI. SAP ANALYSIS IN TCA MODEL

SAP was designed with TCA-Model in mind. The methodology adopted allows us to take a step back to analyze SAP with respect to TCA-Efficiency, TCA-Soundness and TCA-Security, defined in Section IV.

A. TCA-Efficiency

To show that SAP complies with TCA-Efficiency, we need to prove three following lemmas:

**Lemma 1.** In SAP: \( \forall m_i \in S : \text{degree}(m_i) = O(1) \).

**Proof:** SAP’s setup procedure deploys \( S \) as a balanced binary tree rooted on Vrf. The topology remains static from this point onwards. In a binary tree, leaves have degree = 1, the root has degree = 2, and all other nodes have degree = 3. Therefore: \( \forall m_i \in S : \text{degree}(m_i) \leq 3 = O(1) \)

**Lemma 2.** Let \( U_{\text{CA}}(\text{SAP}) \) denote network utilization in SAP. 
\( U_{\text{CA}}(\text{SAP}) = O(N \times l) \), where \( N \) is the number of devices in \( S \) and \( l \) is the security parameter.

**Proof:** In SAP, only request and report procedures involve network communication.

1) From setup, \( S \)'s topology is a balanced binary tree which contains \( \Theta(N) \) communication links.

2) In request procedure, each \( m_i \in S \) receives \( \text{chal} \) from its parent and forwards it to its two children. Thus, each link in the binary tree topology transmits a total of \( |\text{chal}| = l \) bits.

3) report involves transmitting aggregated (XOR-ed) \( h_m \) values. From attest specification, we know that \( |\text{token}_m| = l \) for all \( \text{token}_m \), and the XOR operation does not change the bit-length of its inputs. Therefore, each link transmits \( l \) bits during report execution.

From observations 1, 2 and 3 above and \( U_{\text{CA}} \) definition in Equation 7, we can compute \( U_{\text{CA}}(\text{SAP}) \) as:
\[
U_{\text{CA}}(\text{SAP}) = \sum_{\forall i, j \in \mathcal{L}} D(i, t_{\text{chal}}, t_{\text{resp}}) = \sum_{\forall i, j \in \mathcal{L}} (|\text{chal}| + |\text{token}_m|) = \sum_{\forall i, j \in \mathcal{L}} (2 \times l) = \Theta(N) \times 2 \times l = O(N \times l)
\]

**Lemma 3.** Let \( T_{\text{CA}}(\text{SAP}) \) denote cRA delay of SAP. 
\( T_{\text{CA}}(\text{SAP}) = O(|\log(N) \times (l/\mu + T_{\text{agg}}) + T_{\text{att}}|) \).

**Proof:** According to Equation 6 and SAP description, we can break down \( T_{\text{CA}}(\text{SAP}) \) into 2 delays: \( T_{\text{att}} \) and \( T_{\text{report}} \). They correspond to the execution time of attest and report respectively.

1) From setup, we know that \( S \)'s topology is a balanced binary tree. Let \( m_j \) denote a node that is farthest away from the root. The maximum number of hops in \( S \) then can be represented as a shortest path between \( m_j \) and Vrf, which in turn equals the depth of the tree, i.e.:
\[
\max_{1 \leq i \leq N} sP a\text{th}(\text{Vrf}, m_i) = sP a\text{th}(\text{Vrf}, m_j) = \log_2(N + 2) - 1 = \Theta(\log(N)) \tag{10}
\]

2) Since all members invoke attest at the same time, the delay for \( S \) to execute attest is the same as the time for each individual \( m_i \) to perform attest or:
\[
T_{\text{att}} = T_{\text{att}} \tag{11}
\]

3) report’s delay is dominated by the time for \( m_j \)'s token to be aggregated and forwarded up to Vrf. The propagation time of such token takes \( \text{delay}(m_j, \text{Vrf}, |\text{token}|) \) while the total aggregation time takes \( \text{Path}(m_j, \text{Vrf}) \times T_{\text{agg}} \), where \( T_{\text{agg}} \) is the time for each member to perform token aggregation. Thus, report’s execution time is:
\[
T_{\text{report}} = \text{delay}(m_j, \text{Vrf}, |\text{token}_{m_j}|) + \text{Path}(m_j, \text{Vrf}) \times T_{\text{agg}} = \text{Path}(m_j, \text{Vrf}) \times \left( |\text{token}_{m_j}| / \mu + T_{\text{agg}} \right) \tag{12}
\]

Observations 2 and 3 allow us to compute cRA as:
\[
T_{\text{CA}}(\text{SAP}) = T_{\text{att}} + T_{\text{report}} = T_{\text{att}} + \text{Path}(m_j, \text{Vrf}) \times \left( |\text{token}_{m_j}| / \mu + T_{\text{agg}} \right) \tag{12}
\]

From report description and observation 1, we know that \( \forall m_i \in S : |\text{token}_m| = l \) and \( \text{Path}(m_j, \text{Vrf}) = \Theta(\log(N)) \). Equation 12 becomes:
\[
T_{\text{CA}}(\text{SAP}) = \Theta(|\log(N)|) \times (l/\mu + T_{\text{agg}}) + T_{\text{att}} = O(|\log(N)| \times (l/\mu + T_{\text{agg}}) + T_{\text{att}})
\]

**Theorem 1.** SAP is TCA-efficient according to Definition 2.

**Proof:** Follows directly from Lemmas 1, 2 and 3.

B. TCA-Soundness

To argue that SAP is TCA-Sound, we must show that in the case when \( S \)'s devices are all in a healthy state, verify will output 1.

**Theorem 2.** SAP is TCA-Sound according to Definition 3.

**Proof:** The proof of soundness is based on four observations about SAP specification:
1) From request specification, we can easily see that there is always enough time for chal to reach all \( m_i \in S \), between \( t_{chall} \) and \( t_{att} = chal \). That is because:

\[
\max_{1 \leq j \leq N} \text{delay}(Vrf, m_j, chal) = \max_{1 \leq l \leq N} sPath(Vrf, m_l) \times |chal|/\mu \\
= (\log(N+2) - 1) \times \mu/\mu \\
< |chal| - t_{chall} \quad \text{(from Equation 9)}
\]

Thus, we assume that attest is implemented securely and correctly. Then we go over the leaking any information about SAP TCA-Security-Game.

2) TCA-soundness has a pre-condition of \( \neg \text{Adv} \). Thus, it holds that:

\[
\forall m \in S : \text{PMEM}(m, t_{att}) = cfg
\]

We further break down this scenario into three cases.

a) Adv calls attest \( m \) to produce res\( S \). This option does not buy Adv anything. From attest’s specification, h\( m \) for a given chal = t\( att \) can only be computed successfully if attest is invoked at the proper time. That is because the value of chal is checked against readSecureClock() at the beginning of attest’s execution. Since PMEM\( (m_i, t = chal) \neq cfg \), it is easy to see that \( Pr[|h_m = \text{res}_i| = \text{negl}(l)] \), due to HMAC’s collision resistance.

b) Suppose there exists exactly one \( m \) such that PMEM\( (m_i, t = chal) \neq cfg \). To win the game, Adv must be able to guess res\( S \), i.e., \( Pr[|\text{guess} = \text{res}_i| \geq \text{negl}(l)] \). However, this is not possible because:

\[
\forall \text{guess} \in \{0,1\}^l : Pr[|\text{guess} = \text{res}_i| \geq \text{negl}(l)] = Pr[|\text{HMAC}_{K_m^{\text{res}}}^{\text{res}}(\text{PMEM}, m_i, t = t_{att}) \Rightarrow \text{guess} | \geq \text{negl}(l)] \\
\leq Pr[|\text{rd} = \text{guess}| = 1/2^l = \text{negl}(l)]
\]

where \( \text{rd} \leftarrow \{0,1\}^l \).

c) Suppose there are multiple \( m \) in \( S \) such that PMEM\( (m_i, t = chal) \neq cfg \). It is clear that guessing multiple independent res\( S \)-s is at least as hard as guessing one of them because:

\[
\forall \{\text{guess}_1, \ldots, \text{guess}_l\} \in \{0,1\}^l : \text{Pr}[\forall \text{res}_i = \text{guess}_i] = \prod_{j=1}^{l} \text{Pr}[\text{guess}_j = \text{res}_i] \leq \text{Pr}[\text{res}_i = \text{guess}_i]
\]

Since \( \text{Pr}[\text{res}_i = \text{guess}_i] = \text{negl}(l) \), the probability that Adv winning the game in this case is also negligible.

C. TCA-Security

First, we prove SAP is TCA-Secure by contradiction while assuming the attest’s Trusted Computing Base (TCB) is implemented securely and correctly. Then we go over the security requirements for attest’s implementation, such as memory access control and controlled invocation of attest. In other words, we assume that attest correctly computes h\( m \) = HMAC\( K_{m_i}^{\text{res}}(\text{PMEM}, m_i, t = t_{att}) \) without leaking any information about K\( _{m_i} \). In short:

**Theorem 3.** SAP is TCA-Secure according to Definition 4.

**Proof:** Recall from Definition 4 that, in order to win TCA-Security-Game, Adv must be able to provide H\( \text{Adv} \) that is equal to RES\( S \) with non-negligible probability (\( > \text{negl}(l) \)), while having at least one \( m_i \in S \) for which PMEM\( (m_i, t = chal) \neq cfg \).

Assume that Adv wins TCA-Security-Game. This means that Adv is able to perform one of the following:

1) Adv is able to directly guess the value of RES\( S \) correctly with non-negligible probability, i.e., \( \text{Pr}[\text{guess} = \text{RES}_S] > \text{negl}(l) \). However, under the assumption that the result of an HMAC is indistinguishable from a random bit string:

\[
\forall \text{guess} \in \{0,1\}^l : \text{Pr}[|\text{guess} = \text{RES}_S| > \text{negl}(l)]
\]

This contradicts the assumption.

2) For some \( m_i \) in \( S \) such that PMEM\( (m_i, t = chal) \neq cfg \), Adv is able to obtain res\( S \) with non-negligible probability. We further break down this scenario into three cases.

a) Adv calls attest\( m \) to produce res\( S \). This option does not buy Adv anything. From attest’s specification, h\( m \) for a given chal = t\( att \) can only be computed successfully if attest is invoked at the proper time. That is because the value of chal is checked against readSecureClock() at the beginning of attest’s execution. Since PMEM\( (m_i, t = chal) \neq cfg \), it is easy to see that \( \text{Pr}[|h_m = \text{res}_i| = \text{negl}(l)] \), due to HMAC’s collision resistance.

b) Suppose there exists exactly one \( m \) such that PMEM\( (m_i, t = chal) \neq cfg \). To win the game, Adv must be able to guess res\( S \), i.e., \( \text{Pr}[|\text{guess} = \text{res}_i| > \text{negl}(l)] \). However, this is not possible because:

\[
\forall \text{guess} \in \{0,1\}^l : \text{Pr}[|\text{guess} = \text{res}_i| > \text{negl}(l)] = \text{Pr}[|\text{HMAC}_{K_m^{\text{res}}}^{\text{res}}(\text{PMEM}, m_i, t = t_{att}) \Rightarrow \text{guess} | > \text{negl}(l)]
\]

where \( \text{rd} \leftarrow \{0,1\}^l \).

c) Suppose there are multiple \( m \) in \( S \) such that PMEM\( (m_i, t = chal) \neq cfg \). It is clear that guessing multiple independent res\( S \)-s is at least as hard as guessing one of them because:

\[
\forall \{\text{guess}_1, \ldots, \text{guess}_l\} \in \{0,1\}^l : \text{Pr}[\forall \text{res}_i = \text{guess}_i] = \prod_{j=1}^{l} \text{Pr}[\text{guess}_j = \text{res}_i] \leq \text{Pr}[\text{res}_i = \text{guess}_i]
\]

Since \( \text{Pr}[\text{res}_i = \text{guess}_i] = \text{negl}(l) \), the probability that Adv winning the game in this case is also negligible.

The Proof above shows that Adv can not succeed by guessing RES\( S \) or res\( S \), nor by calling attest. However, Adv might exploit bugs and vulnerabilities on attest to derive the correct h\( m \), while PMEM\( (m_i, t = chal) \neq cfg \). This might be done in three ways.

(a) Learning K\( _{m_i} \) corresponding to some device m\( i \). This allows Adv to generate valid replies for any chal, since the HMAC’s secret is now known by Adv.

(b) Violating the temporal consistency of attest, by chang-
ing PMEM during the execution of \texttt{attest}, making \texttt{attest} result equals to $HMAC_{K_m, -Vrf}(\text{PMEM}\|[\text{chal}])$ where $\text{PMEM} \neq \text{PMEM}(m_i, t = \text{chal})$.

(c) Tampering with the system clock by setting it to \texttt{chal} before the actual real-world time is indeed $t = \text{chal}$. This will cause \texttt{attest} to run before $t = \text{chal}$, making \texttt{attest} result equal to $HMAC_{K_m, -Vrf}(\text{PMEM}(m_i, t = t')\|[\text{chal}])$ where $t' \neq \text{chal}$. In this case, \texttt{Adv} wins for whenever $\text{PMEM}(m_i, t = t') = cf_g_i$ independently from the value of $\text{PMEM}(m_i, t = \text{chal})$ which could be different from $cf_g_i$.

To prevent (a), only \texttt{attest} should have access $K_{m, -Vrf}$. \texttt{attest} should not be modifiable, otherwise malware could modify it to, for example, return $K_{m, -Vrf}$, instead of an $HMAC$ result. Moreover, \texttt{attest}’s execution should not leak any information about $K_{m, -Vrf}$ (e.g., leakage by dead memory) and correctly implement an $HMAC$ function. To guarantee (b), \texttt{attest} should be uninterruptable, therefore making it impossible for malware to modify $\text{PMEM}$ during \texttt{attest}’s execution. Let $r_4$ and $r_6$ denote regions in $\text{ProMEM}$ for storing \texttt{attest}’s implementation and $K_{m, -Vrf}$, respectively (see Figure 2). Aforementioned guarantees can be enforced with the following rules:

\begin{align}
\forall t : & \quad r_4 = \texttt{attest} \\
\forall t : & \quad r_6 = K_{m, -Vrf} \\
\text{Read}(r_6) \rightarrow & \quad PC \in r_4 \\
PC(t + 1) \in r_4 \land \neg (PC(t) \in r_4) \rightarrow & \quad PC(t) = \text{first}(r_4) \\
PC(t) \in r_4 \land \neg (PC(t+1) \in r_4) \rightarrow & \quad PC(t) = \text{last}(r_4) \\
PC \in r_4 \land \neg (\text{interrupt}) 
\end{align}

where $\text{first}(r_4)$ and $\text{last}(r_4)$ refer to the first and last instructions stored in $r_4$. $PC(t+1)$ denotes the address pointed by PC one execution cycle after $PC(t)$.

Equations 15 and 16 imply that \texttt{attest} implementation and the attestation key are the same at all times, i.e., immutable. Equation 17 specifies that the key memory region can only be accessed during the execution of \texttt{attest}. Equations 18 and 19 guarantee that \texttt{attest} execution always starts from its first instruction in memory and ends at its last instruction (controlled invocation). Finally, Equation 20 ensures that \texttt{attest} cannot be interrupted.

Such rules, in addition to the functional correctness of \texttt{attest}’s software implementation (including the fact that $K_{m, -Vrf}$ is not leaked from its execution) must be enforced by the attestation TCB. We do not discuss how these properties can be implemented in a provably secure way and refer to [11] for an example of $\mathcal{RA}$ architecture formally verified with respect to attestation correctness and security, including aforementioned rules. In addition to these conditions, the architecture should rely on a read-only, and synchronized clock, hence preventing (c). We describe next our prototype implementation of SAP and its evaluation.

VII. SAP Implementation and Evaluation

A. Implementation

We implemented SAP on top of TrustLite [22], a recently proposed $\mathcal{RA}$ architecture for low-end embedded devices. TrustLite uses a Secure Boot feature to ensure integrity and immutability of SAP’s code and $K_{m, -Vrf}$ (Equations 15 and 16). $k_{\text{plat}}$ is the secret used by the Secure Boot’s integrity ensuring function. Memory protection is provided by a lightweight execution-aware memory protection unit (EAMA) which is programmed with $\text{ProMEM}$’s access control rules (Equation 17) as well as the required access rules for the Secure Boot. Controlled invocation and uninterruptability of \texttt{attest} (represented by Equations 18, 19, and 20) are enforced by TrustLite’s trusted operating system. The Secure Boot also ensures that the operating system is correctly loaded. The \texttt{attest}’s HMAC is based on SHA-1, which is already implemented by TrustLite. Figure 2 depicts our implementation on TrustLite.

To enable secure synchronous attestation, we have extended TrustLite’s hardware with a secure read-only clock. Secure clocks are typically used in $\mathcal{RA}$ schemes that support non-interactivity [20], to mitigate denial of service (DoS) attacks [6], and to detect physical attacks [19]. The secure clock must be write-protected and have a large enough counter, which does not wrap around during the expected device lifetime. On TrustLite with 24 MHz, we use a 32-bit register that is incremented every 250,000 cycles (i.e., every 11 ms), and hence would wrap around in almost 2 years, which we consider enough for most applications.

B. Implementation Overhead

Software Complexity. Since the HMAC’s implementation is already present in TrustLite, SAP incurs only a few additional Lines-of-Code (LoC), corresponding to the code for request and report. This results in less than 200 Bytes increase in code size.

Hardware Costs. The additional hardware comprises a secure clock and one MPU rule to restrict access to $K_{m, -Vrf}$. Table II shows the hardware utilization of baseline TrustLite and that of TrustLite with the hardware extensions required for SAP in terms of required registers and Look-up Tables. SAP incurs an overhead of 2.45% and 1.41% over that of baseline TrustLite.

<p>| Table II: SAP’s Hardware cost |</p>
<table>
<thead>
<tr>
<th>Register</th>
<th>Look-up Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrustLite [22]</td>
<td>6038</td>
</tr>
<tr>
<td>SAP</td>
<td>6335</td>
</tr>
</tbody>
</table>

C. Efficiency Evaluation

To assess the efficiency of SAP in practice, we have used OMNeT++ network simulator [29]. We also compare SAP with SEDA [3], a state-of-the-art $\mathcal{RA}$ scheme. We report...
on cR\textsubscript{A} Execution Time and Network Utilization. These correspond to \( T_{C_A} \) and \( U_{C_A} \) in TCA-Model. For completeness, Section VII-D also reports on the overall power consumption.

We account for cryptographic operations by adding delays that are based on real measurements from our implementation on TrustLite. We simulated networks with up to 1,000,000 devices. Recall that the time to compute \textbf{attest} in SAP is determined by \( t_{att} > t_{chal} + (\log_2(N+2) - 1) \times |chall|/\mu \). We set \( t_{chal} \) to \( t_{chal} + (\log_2(N+2) - 1) \times |chall|/\mu + \tau(N) \), where \( \tau(N) = 0.001 \cdot (\log_2(N+2) - 1) \); \( \tau(N) \) accounts for the delay on each hop which is set to 1 ms per hop. Further, per \textbf{setup} specification, the network topology is deployed as a binary tree. Finally, we executed our simulations assuming that all members are low-end embedded devices (e.g., our 24 MHz TrustLite), and that PMEM size on each device is 50 KB. Simulation results are presented in Figure 3.

As shown in Figure 3(a), execution time of SAP is logarithmic in \( \mathcal{S} \) size and significantly lower than that of the most efficient cR\textsubscript{A} protocol – SEDA. SAP run-time is 0.6 seconds in a network of 1,000,000 devices, as opposed to 1.4 seconds for the same in SEDA. Unlike SEDA, SAP does not use public key cryptography. Communication overhead of SAP is half that of SEDA, and aggregation in SAP is based on XOR-ing MACs, compared to hop-by-hop MAC verification in SEDA.

Execution of SAP can be split into three main phases (see Figure 3(b)): (1) Inbound execution, when attestation challenges are generated and spread across the network. This has logarithmic runtime in network size; (2) Measurement, during which every member computes \textbf{attest}. This time is constant (because all \( m_i \in \mathcal{S} \) run \textbf{attest} in parallel at time \( t_{att} \);
and (3) Outbound execution, when reports are gathered and aggregated en route to Vrf. This also has logarithmic runtime in network size. Finally, there is an additional delay caused by the difference between the end of outbound execution, and the start of measurement, i.e., delay function $\tau(N)$. This delay is logarithmic and it is required to ensure that all members receive the challenge before attestation time.

Figure 3(c) presents SAP’s network utilization, which is linear in the size of the network. The network utilization in a network formed of 1,000,000 devices is around 40 MBytes (40 Bytes per device). The logarithmic execution time and linear increase in network utilization demonstrate that, in practice, SAP performs as expected from its systematic design with respect to SAP-Efficiency (see Definition 2).

D. Power Consumption

Let $P_{\text{attest}}$, $P_{\oplus}$, $P_{\text{send}}$, and $P_{\text{recv}}$ denote power consumption for computing: attest, XOR-ing attest results during report, sending one byte, and receiving one byte, respectively. Then, power consumption for a leaf device and inner devices to execute SAP ($P_{\text{leaf}}$ and $P_{\text{node}}$ respectively) can be estimated as:

$$
\begin{align*}
P_{\text{leaf}} & \leq (|\text{chal}| + |\text{token}|) \cdot P_{\text{send}} + |\text{chal}| \cdot P_{\text{recv}} + P_{\text{attest}} \\
P_{\text{node}} & \leq (|\text{chal}| + |\text{token}|) \cdot P_{\text{send}} + (|\text{chal}| + 2 \cdot |\text{token}|) \cdot P_{\text{recv}} + P_{\text{attest}} + 2 \cdot P_{\oplus}
\end{align*}
$$

Considering such metrics, power consumption of SAP is presented in Table III. Computation is based on previously reported power consumption for sensor nodes MICAz and TelosB [10], which belong to the same low-end device class targeted by TrustLite.

Table III: Power consumption (in mW) of SAP

<table>
<thead>
<tr>
<th>Device</th>
<th>Leaf</th>
<th>Inner node</th>
</tr>
</thead>
<tbody>
<tr>
<td>MICAz</td>
<td>0.3372</td>
<td>0.5516</td>
</tr>
<tr>
<td>TelosB</td>
<td>0.369</td>
<td>0.6282</td>
</tr>
</tbody>
</table>

VIII. Other cRA Dimensions & Their Systematic Analysis

In this work, we define TCA-Model targeting a specific domain. To model other cRA use-cases, it needs to be modified or extended. We now describe a non-exhaustive list of sample application settings and outline how our definitions for cRA’s security, soundness and efficiency could be adapted.

Mitigation of DoS Attacks. Authenticating Vrf’s requests is needed to prevent computational DoS attacks for settings where Adv controls network communication and can spoof cRA requests. One way to integrate Vrf’s authentication into TCA-model is to adapt the security game in Definition 4, to allow Adv to win if attest is called on an unauthenticated request. Mitigation of other types of attacks (e.g., physical or runtime attacks) can also be considered by similarly modifying the security game.

Quality of Attestation. In some application domains, a binary cRA outcome may not be sufficient. For example, if $S$ covers a large physical area, it might be important to identify which members are infected, so that a quick in-place reaction can be taken. This was defined in [7] as Quality of Attestation (QoA). To incorporate QoA, Definition 4 should be modified to account for the granularity of information that can be learned about $S$’s state from verify’s result.

Lossy Networks. TCA-Model assumes a lossless network. To consider lossy networks, the soundness notion in Definition 3 must be relaxed to consider that packet loss might be caused by natural reasons (e.g., interference) and not only due to adversarial activities. Accounting for packet losses and retransmissions might also significantly complicate the definition of efficiency goals (Definition 2) and its analysis.

Resource-Constrained Devices. Different application domains might involve groups of devices with different constraints, such as energy, memory, and computational power. For example, for battery-powered devices, energy is a primary concern, and power consumption must be minimized. For devices with limited program memory, the size of cRA TCB can be critical. To account for these issues, the notion of efficiency, in Definition 2, must be modified or expanded.

Safety-Critical Systems. There can be multiple aspects to consider in this domain. First, it is important to ensure that the attestation functionality does not deviate a device away from its original task. Thus, attest execution time may need to be minimized. This would require a modification to the efficiency notion to include upper bounds for local attestation runtime. Also, the soundness notion needs to be adapted to reflect that such safety-critical task can still be successfully executed even when the cRA protocol requires the execution of attest.

IX. Conclusions

Collective attestation (cRA) is an important security service that allows efficiently securing large groups of IoT devices. However, lack of a systematic treatment for cRA makes it hard to derive any formal guarantees and all previous cRA protocols were designed in an ad hoc fashion. This paper presents the first formal model for analyzing cRA protocols, TCA-model, which captures cRA requirements in three categories: efficiency, soundness and security. Based on TCA-Model, we construct SAP – the first systematic design of a cRA protocol with provable guarantees. We believe that the methodology for systematically designing cRA protocols can be naturally extended to other use-cases and serve as a guideline for future cRA designs.

Acknowledgments. Work by TU Darmstadt co-authors was funded by the German Research Foundation (DFG) within CRC 1119 CROSSING projects S2 and P3 and the Intel Collaborative Research Institute for Collaborative Autonomous & Resilient Systems (ICRI-CARS). Work by UCI co-authors was supported by ARO under contract #: W911NF-16-1-0536, and NSF WiFiUS Program Award #: 1702911.
REFERENCES


