TOCTOU Resilient Attestation for IoT Networks

Pavel Frolikov, Youngil Kim, Renascence Tarafder Prapty, Gene Tsudik
UC Irvine, CA, USA
{pavel,youngik2,rprapty,gene.tsudik}@uci.edu

Abstract

Internet-of-Things (IoT) devices are increasingly common in both consumer and industrial settings, often performing safety-critical functions. Although securing these devices is vital, manufacturers typically neglect security issues or address them as an afterthought. This is of particular importance in IoT networks, e.g., in the industrial automation settings.

To this end, network attestation – verifying the software state of all devices in a network – is a promising mitigation approach. However, current network attestation schemes have certain shortcomings: (1) lengthy TOCTOU (Time-Of-Check-Time-Of-Use) vulnerability windows, (2) high latency and resource overhead, and (3) susceptibility to interference from compromised devices. To address these limitations, we construct TRAIN (TOCTOU-Resilient Attestation for IoT Networks), an efficient technique that minimizes TOCTOU windows, ensures constant-time per-device attestation, and maintains resilience even with multiple compromised devices. We demonstrate TRAIN's viability and evaluate its performance via a fully functional and publicly available prototype.

CCS Concepts

• **Security and privacy → Embedded systems security**; **Hardware-based security protocols**; **Security protocols**; • **Computer systems organization → Embedded systems**.

Keywords

IoT Security, Remote Attestation, Swarm Attestation, Network Attestation, TOCTOU, Malware Resistance Operation

ACM Reference Format:

Pavel Frolikov, Youngil Kim, Renascence Tarafder Prapty, Gene Tsudik. 2025. TOCTOU Resilient Attestation for IoT Networks. In *The 23rd ACM Conference on Embedded Networked Sensor Systems (SenSys '25), May 6–9, 2025, Irvine, CA, USA*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3715014.3722057

1 Introduction

Rapid expansion and popularity of the Internet of Things (IoT) devices and Cyber-Physical Systems (CPS) have resulted in the deployment of vast numbers of Internet-connected and inter-connected devices. Such networks, composed of numerous devices, collaboratively execute sensing and/or actuation tasks in diverse settings, such as smart factories, warehouses, agriculture, and environmental monitoring. However, the resource-constrained nature of IoT devices makes them vulnerable to remote attacks. This poses significant risks: malicious actors can compromise data integrity or even jeopardize safety within critical control loops. Given the safety-critical functions they perform and the sensitive data they collect, protecting IoT devices against such attacks is essential. Remote attestation ($\mathcal{RA}$), a well-established security service, detects malware on remote devices by verifying the integrity of their software state [15, 32, 39]. However, applying single-device $\mathcal{RA}$ techniques to large IoT networks incurs high overhead. Many techniques, including [4, 7, 10, 28, 47], made progress towards efficient network (aka swarm) attestation ($\mathcal{NA}$). Nonetheless, they have substantial limitations, which form the motivation for this work.

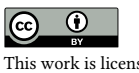
Time-of-Check to Time-of-Use (TOCTOU)**:** Prior techniques do not guarantee simultaneous (synchronized) attestation across all networked devices. Network structure, potential mobility, intermittent connectivity, and congestion can lead to staggered reception of $\mathcal{RA}$ requests, thus widening the time window for discrepancies in $\mathcal{RA}$ timing. Also, even if networked devices are all of the same type, varying memory sizes and application task scheduling can result in different execution times of $\mathcal{RA}$. These factors lead to a potentially long TOCTOU window, where the state of network devices is captured over an interval of time rather than at the same time. This increases the risk of undetected transient malware presence. The TOCTOU problem arises in two cases:

TOCTOU$_{\mathcal{RA}}$ – the window of vulnerability between two successive $\mathcal{RA}$ instances performed by a device, during which the state of the software is unknown and potentially compromised without detection; colored orange in Figure 1(a). TOCTOU$_{\mathcal{RA}}$ can be exploited by transient malware that: (1) infects a device, (2) remains active for a while, and (3) erases itself and restores the device software to its "good" state, as long as (1)-(3) occur between two successive $\mathcal{RA}$ instances.

TOCTOU$_{\mathcal{NA}}$ – the inter-device TOCTOU window, i.e., the time variance between the earliest and the latest $\mathcal{RA}$ performed across networked devices. colored red in Figure 1(a).

Consider a situation where, the verifier performs network attestation. Device-A receives an attestation request, performs its attestation at time t_0 and replies to the verifier. At time $t_1 > t_0$, the verifier receives device-A's attestation report, checks it, and decides that device-A is benign. However, device-A is compromised at time $t_2 > t_1$. Meanwhile, due to network delay, device-B performs its attestation at $t_3 > t_2$ and replies. The verifier receives device-B's attestation report at $t_4 > t_3$ and (erroneously) concludes that both devices are now benign.

Synchronized Attestation – An important requirement for $\mathcal{NA}$ is that all attestation reports should accurately reflect current system



SenSys '25, May 6–9, 2025, Irvine, CA, USA

ACM ISBN 979-8-4007-1479-5/25/05
https://doi.org/10.1145/3715014.3722057

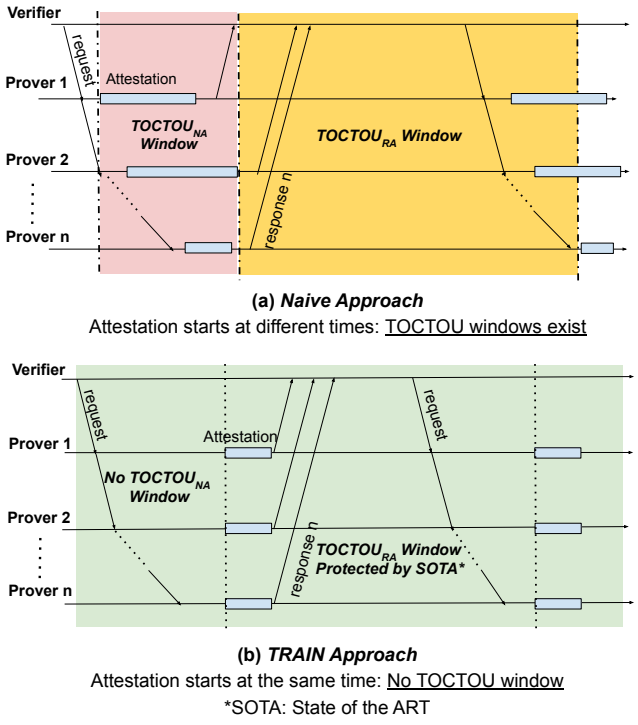


Figure 1: TOCTOU Window Minimized by TRAIN

state. If devices are attested at different times, the verifier cannot determine if the network as a whole is (or was) in a secure state, even if all individual $\mathcal{R}$A reports reflect the benign state. This undermines trust in current attestation methods, motivating the need for more synchronized network-wide attestation.

Performance Overhead: Attesting the entire software state of a device is computationally expensive. For safety-critical IoT devices, minimizing time spent on non-safety-critical tasks (e.g., $\mathcal{R}$A) is crucial to maintain responsiveness and real-time performance. Even a lightweight $\mathcal{R}$A, which is typically based on a device computing a relatively fast Message Authentication Code (MAC) (usually implemented as a keyed hash) requires doing so over the entire application program memory. This introduces a non-negligible delay which is a function of memory size. For example, a TI MSP430 microcontroller unit (MCU) running at 8MHz takes $\approx$ 450ms to compute an SHA2-256 HMAC over 4KB of program memory [40]. This delay is significant for real-time or safety-critical systems with tight timing constraints.

Energy Overhead: Execution of $\mathcal{R}$A consumes energy on battery-powered or energy-harvesting IoT devices. This is particularly problematic for devices deployed in remote or inaccessible locations where battery replacement is difficult or infeasible. Reducing power consumption is therefore both beneficial and important.

Unreliable Communication: Malware-infected devices can subvert the attestation process by dropping or modifying attestation requests and replies. Prior techniques do not adequately address this problem. To this end, we construct TRAIN: TOCTOU Resilient Attestation for IoT Networks. It offers two protocol variants: TRAIN_A – for devices equipped with real-time clocks (RTCs), and TRAIN_B – for devices without such clocks. TRAIN is designed to work with low-end IoT devices that have a small set of security features, based on RATA [15] or CASU [14] $\mathcal{R}$A techniques which were originally developed for a single-device $\mathcal{R}$A setting. TRAIN pairs these $\mathcal{R}$A techniques with GAROTA [3] – another recent technique that constructs a minimal **active** Root-of-Trust (*RoT*) for low-end devices and guarantees operation even if a device is fully malware-compromised. Specifically, TRAIN uses NetTCB and TimerTCB of GAROTA which ensure, respectively: (1) timely sending and receiving messages, and (2) starting attestation on time, with no interference from any other software.

Contributions of this work are:

(1) **Reduced** TOCTOU **Window**: TRAIN employs time synchronization (using RTCs or a depth-based mechanism) to ensure nearly simultaneous attestation across all devices in the network, substantially reducing the $\text{TOCTOU}_{\mathcal{NA}}$ window. The $\text{TOCTOU}_{\mathcal{RA}}$ window is mitigated by the use of CASU or RATA security features. Figure 1(b) shows decreased TOCTOU windows by TRAIN.

(2) **Efficient and resilient** $\mathcal{R}$A: TRAIN combines a few *RoT* constructions to minimize $\mathcal{R}$A-induced performance overhead and power consumption for individual devices, while guaranteeing timely $\mathcal{R}$A execution by isolating it from any potential malware interference.

(3) **Open-Source Implementation**: TRAIN's practicality and cost-effectiveness is confirmed via a fully functional prototype on a popular low-end IoT device platform – TI MSP430 microcontroller, using FPGA [45].

2 Background

2.1 Targeted Devices

We focus on resource-constrained devices using low-end MCUs, such as Atmel AVR ATmega and TI MSP430, which are low-power single-core platforms with limited memory. These devices have 8-bit or 16-bit CPUs, 1-16MHz clock frequencies, and typically $\leq$ 64KB of addressable memory. Data memory (DMEM) ranges from 4 to 16KB, while the rest is program memory (PMEM). Software executes in-place from PMEM. It runs on "bare metal", with no memory management for virtual memory or isolation.

A representative architecture for targeted devices includes a CPU core, DMA controller, and interrupt controller connected via a bus to memory regions: ROM, PMEM, DMEM, and peripheral memory. ROM holds the bootloader and immutable software. Device software resides in PMEM, and DMEM is used for the stack and heap. The device may incorporate both internal peripherals (timers) and external peripherals (sensors, actuators).

2.2 Remote Attestation ($\mathcal{R}$A)

As mentioned earlier, $\mathcal{R}$A is used for malware detection on a remote device. It is typically achieved via a challenge-response protocol that enables a trusted entity called a verifier ($\mathcal{V}$rf) to remotely verify the software state of an untrusted remote device ($\mathcal{P}$rv):

(1) $\mathcal{V}$rf sends an $\mathcal{R}$A request with a challenge ($\mathcal{C}$hal) to $\mathcal{P}$rv.

(2) $\mathcal{P}$rv generates an unforgeable attestation report, i.e., an authenticated integrity check over PMEM, including the software, and $\mathcal{C}$hal, and sends it to $\mathcal{V}$rf.

(3) $\mathcal{V}$rf verifies the report to determine whether $\mathcal{P}$rv is in a valid state.

The report includes either a Message Authentication Code (MAC) or a signature, depending on the type of cryptography used. In the former case, $\mathcal{P}$rv and $\mathcal{V}$rf must share a unique secret key – $\mathcal{K}_{Dev}$, while in the latter, $\mathcal{K}_{Dev}$ is a unique private key of $\mathcal{P}$rv. In either case, $\mathcal{K}_{Dev}$ must be stored securely and be accessible only to the trusted attestation code on $\mathcal{P}$rv.

A large body of research [2, 14, 15, 17–20, 26, 38, 40, 41, 49, 56] explored $\mathcal{R}$A for low-end devices. Prior work can be split into **passive** and **active** techniques. The former only **detects** compromise and offers no guarantee of the device responding to an $\mathcal{R}$A request. Whereas, the latter either **prevents** compromise and/or guarantees small security-critical tasks (e.g., an $\mathcal{R}$A response).

2.3 Network Attestation ($\mathcal{N}$A)

Unlike single-device $\mathcal{R}$A, which involves one $\mathcal{V}$rf and one $\mathcal{P}$rv, $\mathcal{N}$A deals with a potentially large number (network, group, or swarm) of $\mathcal{P}$rv-s. This opens new challenges. First, naïve adoption of single-$\mathcal{P}$rv $\mathcal{R}$A techniques is inefficient and even impractical. Also, $\mathcal{N}$A needs to take into account topology discovery, key management, and routing. This can be further complicated by mobility (i.e., dynamic topology) and device heterogeneity. Moreover, TOCTOU$_{\mathcal{N}\mathsf{A}}$ (inter-device TOCTOU) emerges as a new problem.

2.4 Building Blocks

RATA [15] is a passive Root-of-Trust (*RoT*) architecture that mitigates TOCTOU$_{\mathcal{R}\mathsf{A}}$ with minimal additional hardware. RATA securely logs the last PMEM modification time to a protected memory region called Latest Modification Time (LMT), which can not be modified by any software. $\mathcal{P}$rv's attestation report securely reflects the integrity of its software state indirectly through the LMT. This approach is based on the principle that any modification to the software in PMEM would necessitate an update to the LMT. Thus, by attesting the LMT, RATA effectively attests the software state without needing to read the entire PMEM contents. This minimizes $\mathcal{R}$A computational overheads of $\mathcal{P}$rv by attesting only a fixed-size (32-byte) LMT (plus the $\mathcal{V}$rf's challenge of roughly the same size), instead of attesting its entire software in PMEM.

CASU [14] is an active *RoT* architecture that provides run-time software immutability and authenticated software updates. It defends against code injection (into PMEM) and data execution attacks by preventing (1) unauthorized modification of PMEM and (2) code execution from DMEM. CASU monitors several CPU hardware signals (e.g., program counter, write-enable bit, and destination memory address) and triggers a reset if any violation is detected. The only means to modify PMEM is via secure update. CASU inherently prevents TOCTOU$_{\mathcal{R}\mathsf{A}}$ since PMEM cannot be overwritten by malware.

GAROTA [3] is another active architecture which guarantees execution of trusted and safety-critical tasks. These tasks are triggered based on arbitrary events captured by hardware peripherals (e.g., timers, GPIO ports, and network interfaces), even if malware is present on the device. GAROTA provides two hardware properties: "guaranteed triggering" and "re-triggering on failure". The former ensures that a particular event of interest always triggers execution of GAROTA TCB tasks, while the latter ensures that if TCB execution is interrupted for any reason (e.g., attempts to violate execution integrity), the device resets, and the TCB task is guaranteed to be executed first after the boot. GAROTA has 3 flavors: TimerTCB, NetTCB, and GPIO-TCB,. In this paper, we are interested in the first two: (1) **TimerTCB** – A real-time system where a predefined safety-critical task is guaranteed to execute periodically, and (2) **NetTCB** – A trusted component that guarantees to process commands received over the network, thus preventing malware on the MCU from intercepting and/or discarding commands destined for the *RoT*.

2.5 Hash Chains for Authentication

Hash chains provide a secure, scalable, and efficient means of authentication, originally proposed by Lamport [31]. Over the last 40+ years, they have been used in numerous settings where one party (signer/sender) needs inexpensive (though limited or metered) authentication to a multitude of receivers.

An m-link hash chain is constructed by repeatedly applying a cryptographic hash function H, starting with the initial secret value x_0, such that:

$$H(x_0) = x_1, H(x_1) = x_2, ..., H(x_{m-1}) = x_m, \text{ i.e., } x_m = H^m(x_0)$$

To set up the operation of an m-link hash chain, signer (sender) retains x_0 (**root**) and shares final x_m (**anchor**) with all receivers. Given a value x_i where $(0 \leq i \leq m)$, it is computationally infeasible to compute x_{i-1} or any previous value x_k for $k < i$. Conversely, calculating x_{i+1} or any subsequent value x_j where $j > i$ is straightforward; x_j can be computed by repeatedly applying the hash function $H()$ to x_i (j-i times).

For the first authentication round, signer reveals x_{m-1} and all receivers can easily authenticate it by comparing $H(x_{m-1}) \stackrel{?}{=} x_m$. In the second round, the signer reveals x_{m-2}, and so on. This continues until the hash chain is exhausted, at which point a new hash chain is generated and shared. See Section 4.3 for the use of hash chains in TRAIN.

Suppose that a receiver is de-synchronized: it currently has x_i and, instead of the expected x_{i-1}, it next receives authenticator x_j where $j < (i-1)$. This means that this receiver missed $i-j-1$ successive authenticators: $x_{i-1}, ..., x_{j+1}$. Nonetheless, a receiver can quickly re-synchronize by authenticating x_j via computing $H^{(i-j)}(x_j)$ and checking if it matches x_i.

3 Design Overview

3.1 System Model

Network: We assume a single verifier ($\mathcal{V}$rf) and a network of multiple low-end embedded devices as $\mathcal{P}$rv-s. $\mathcal{V}$rf is assumed to be trusted and sufficiently powerful. The network is assumed to be: (1) connected, i.e., there is always a path between $\mathcal{V}$rf and any of $\mathcal{P}$rv-s, and (2) quasi-static during attestation, i.e., its topology can change as long as the changes do not influence the path of message propagation. TRAIN is network-agnostic and can be realized over any popular medium (e.g., WiFi, Bluetooth, Cellular, Zigbee, Matter).

$\mathcal{R}$A Architecture in $\mathcal{P}$rv: All $\mathcal{P}$rv-s must support RATA or CASU architecture: in a given deployment, either all support the former or all support the latter, i.e., no mixing.[1] As mentioned in Section 2, an

[1]This is not a hard requirement, meaning that a mix of RATA and CASU devices would work as well; however, it makes the presentation simpler.

attestation token in RATA is computed as a keyed hash over a small fixed-size input.

In contrast, CASU prevents any PMEM modification (except via secure update), thus obviating the entire need for $\mathcal{R}$A. However, CASU does not offer $\mathcal{P}$rv liveness. Note that, in any secure $\mathcal{R}$A technique, an attestation token returned by $\mathcal{P}$rv provides both attestation and $\mathcal{P}$rv liveness. The latter is important for detecting whether $\mathcal{P}$rv is operational, i.e., not powered off, destroyed/damaged, or physically removed. To this end, CASU supports a "secure heartbeat" feature, whereby $\mathcal{V}$rf periodically issues a random challenge and $\mathcal{P}$rv simply computes (and returns) a keyed hash over that challenge. This costs about the same as attestation token computation in RATA. We discuss various use-cases of RATA and CASU in Section 7.2.

Network Interface in $\mathcal{P}$rv: The primary network interface of each $\mathcal{P}$rv is placed within TRAIN's Trusted Computing Base (TCB). This ensures that TRAIN protocol messages are handled with the highest priority, even in the presence of malware or run-time attacks. TRAIN uses two special attestation-specific packet types: request and report. Normal software outside TCB is prevented from sending or receiving these packet types; this is achieved by inspecting each incoming/outgoing packet header in order to prevent tampering with, and forgery of, TRAIN messages. Furthermore, TRAIN packets are always handled with higher priority than other tasks. This approach is based on NetTCB of GAROTA [3]. Besides, we adopt TimerTCB from GAROTA to guarantee (nearly) synchronized attestation start times.

With these security features, RATA-enabled $\mathcal{P}$rv-s are safeguarded against full compromise and malware-based disruption of the attestation process. The benefit is more subtle in the case of CASU: although CASU guarantees no malware, software running on CASU-enabled $\mathcal{P}$rv-s can still be susceptible to control-flow attacks, which would prevent, or delay, receiving of $\mathcal{V}$rf attestation requests and generation of secure heartbeats. The above features ensure that this does not occur.

$\mathcal{P}$rv TCB: TRAIN TCB includes both hardware and software components, i.e., akin to RATA and CASU, TRAIN is a **hybrid** architecture. In addition to the trusted software of either RATA or CASU, TRAIN software includes TimerTCB, NetTCB, and $\mathcal{N}$A logic described in Section 4. The primary network interface (NetTCB) is shared between the TRAIN software and other non-TCB software. Incoming messages cause an interrupt via NetTCB. TRAIN software prioritizes TRAIN protocol messages. It forwards other incoming messages to the intended application (outside TCB) and outgoing messages to the destination. TCB hardware components are:

- NetTCB – Network interface for TRAIN messages
- TimerTCB – Timer used for simultaneous attestation
- DMEM segment reserved for running TRAIN software
- Part of ROM reserved for TRAIN software, key shared with $\mathcal{V}$rf, and hash chain data

3.2 Adversary Model

In line with other network attestation ($\mathcal{N}$A) techniques, TRAIN considers software-only remote network attacks. We assume an adversary ($\mathcal{A}$dv) that can inject malware and exercise full control over a compromised $\mathcal{P}$rv, except for its TCB. $\mathcal{A}$dv can manipulate any non-TCB peripherals and external components, such as Direct Memory Access (DMA), sensors, actuators, and other (non-primary) network interfaces. Also, $\mathcal{A}$dv has comprehensive knowledge of software (i.e., non-TRAIN software) running on $\mathcal{P}$rv, including its memory vulnerabilities. Thus, it can launch run-time (e.g., control-flow) attacks.

We also consider a network-based $\mathcal{A}$dv represented by a malicious (non-TRAIN) entity in the $\mathcal{P}$rv network. Consequently, all packets exchanged between $\mathcal{V}$rf and $\mathcal{P}$rv-s can be manipulated by $\mathcal{A}$dv: based on the Dolev-Yao model [16], $\mathcal{A}$dv can eavesdrop on, drop, delay, replay, modify, or generate any number of messages.

DoS Attacks: TRAIN prevents DoS attacks that attempt to "brick" $\mathcal{P}$rv-s via malware, or control-flow attacks. However, DoS attacks that jam the network or attempt to inundate specific $\mathcal{P}$rv's network interfaces are out of scope. For countermeasures, we refer to well-known techniques, such as [34, 37, 57].

Physical Attacks: TRAIN does not offer protection against physical attacks, both invasive (e.g., via hardware faults and reprogramming through debuggers) and non-invasive (e.g., extracting secrets via side-channels). Such attacks can be mitigated, at considerable cost, via well-known tamper-resistance methods [42, 48].

3.3 Protocol Elements

As mentioned in Section 1, we construct two TRAIN variants, based on the availability of a real-time clock (RTC) on $\mathcal{P}$rv-s:

(1) **TRAIN$_\mathbf{A}$:** Each $\mathcal{P}$rv has an RTC. In an attestation request, $\mathcal{V}$rf includes the exact time when all $\mathcal{P}$rv-s should perform attestation.

(2) **TRAIN$_\mathbf{B}$:** $\mathcal{P}$rv-s do not have RTC-s. In an attestation request, $\mathcal{V}$rf provides the height of the spanning tree, composed of all $\mathcal{P}$rv-s. Each $\mathcal{P}$rv estimates the time to perform attestation using spanning tree height and its own secure timer.

TRAIN$_\text{A}$ is designed for an ideal best-case scenario where each $\mathcal{P}$rv is assumed to have a synchronized RTC. TOCTOU$_{\mathcal{N}\text{A}}$ window is completely removed in TRAIN$_\text{A}$. On the other hand, TRAIN$_\text{B}$ is intended for a more realistic scenario where each $\mathcal{P}$rv has a timer. Although TRAIN$_\text{B}$ can not offer precisely synchronized attestations on $\mathcal{P}$rv-s, it still significantly reduces TOCTOU$_{\mathcal{N}\text{A}}$. Section 6.1 provides further details.

TOCTOU$_{\mathcal{N}\text{A}}$ resilience Due to the availability of RTC in TRAIN$_\text{A}$ and the spanning tree's height in TRAIN$_\text{B}$, all $\mathcal{P}$rv-s perform attestation almost simultaneously. Figure 1(b) shows the eliminated TOCTOU window in TRAIN$_\text{A}$.

Attestation Regions: Unlike prior $\mathcal{N}$A schemes which perform attestation over the entire PMEM, TRAIN is built on top of either RATA or CASU, which enables $\mathcal{P}$rv to compute a MAC over a short fixed size including: (1) LMT_{Dev} and $\mathcal{V}$rf's challenge, in RATA, or (2) merely $\mathcal{V}$rf's challenge, in CASU. Section 4 provides details about other parameters included in the MAC computation.

Authentication of Attestation Requests: Most prior work in network (or swarm) attestation does not take into account authentication of attestation requests. While this may or may not be an issue in a single $\mathcal{P}$rv $\mathcal{R}$A setting[2], it certainly becomes a concern in $\mathcal{N}$A. If requests are not authenticated, $\mathcal{A}$dv can readily mount a DoS attack whereby $\mathcal{A}$dv floods all $\mathcal{P}$rv-s with bogus requests, each of

[2] $\mathcal{V}$rf authentication in a single $\mathcal{P}$rv setting is thoroughly discussed in [9].

which causes **all** $\mathcal{P}$rv-s to perform attestation and generate numerous useless replies.

This issue is deceptively simple. The naïve approach to address the problem is for $\mathcal{V}$rf (which already shares a unique symmetric key with each $\mathcal{P}$rv) to send an individual attestation request to every $\mathcal{P}$rv, authenticated with each shared key. This is unscalable for obvious reasons.

Another intuitive approach is to assume that every $\mathcal{P}$rv knows $\mathcal{V}$rf's public key and $\mathcal{V}$rf simply signs each attestation request with a timestamp. Despite scaling well, this approach opens the door for a simple DoS attack whereby $\mathcal{A}$dv floods the network with attestation requests with fake signatures, forcing all $\mathcal{P}$rv-s to verify them, and due to failed verification, discard the requests. This incurs heavy collective computational overhead on the entire network.

Yet another trivial method is to assume a separate group key (shared among $\mathcal{V}$rf and all $\mathcal{P}$rv-s) that is used exclusively for authenticating $\mathcal{V}$rf-issued attestation requests. This is quite efficient since a simple MAC (realized as a keyed hash) would suffice. However, a key shared among a potentially large number of $\mathcal{P}$rv-s raises the risk of its eventual compromise, which would have unpleasant consequences.

Also, managing the group and key revocation becomes increasingly complex as the network grows.

TRAIN uses hash chains to authenticate attestation requests. Hash chains, as described in Section 2.5, are well-known constructs used in numerous similar settings where symmetric keys are unscalable and traditional public key signatures are too expensive.

They provide forward security and efficient verification, while offering relatively simple key management. Although hash chains suffer from some fragility in terms of synchronization and timing requirements, these issues are more palatable than those that stem from managing large numbers of shared keys.

4 TRAIN Protocols

This section describes two protocol variants [3]. The notation used in the rest of the paper is summarized in Table 1.

Assumptions: As mentioned above, we assume that each $\mathcal{P}$rv shares a unique symmetric key ($\mathcal{K}_{Dev}$) with $\mathcal{V}$rf. Also, throughout a single attestation instance, $\mathcal{V}$rf is assumed to be within the broadcast range of at least one $\mathcal{P}$rv, and the entire $\mathcal{P}$rv network must remain connected during this time. Furthermore, all $\mathcal{P}$rv-s have a parameter ($t_{maxDelay}$) that denotes the maximum attestation report (Att_{report}) propagation delay in the network. In the absolute worst case of a line topology, it can be set as: $t_{maxDelay} = n * t_{report}$, where n is the number of $\mathcal{P}$rv-s and t_{report} is the Att_{report} propagation delay. We also assume that the attestation request ($Att_{request}$) propagation delay ($t_{request}$) and the $Att_{request}$ verification time (t_{hash}) are known to all $\mathcal{P}$rv-s. $t_{maxDelay}$ is needed to limit the time when each $\mathcal{P}$rv forwards other $\mathcal{P}$rv-s' attestation results towards $\mathcal{V}$rf. For the sake of simplicity, we assume that no new attestation requests are issued while one is being served.

NOTE: As mentioned at the end of Section 3.3, the use of hash chains for $\mathcal{V}$rf authentication is optional; a separate group key shared between $\mathcal{V}$rf and all $\mathcal{P}$rv-s could be used instead, albeit with the risk of its possible leak.

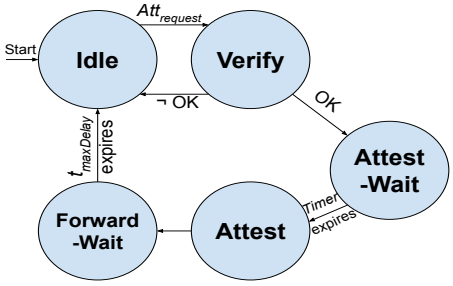


Figure 2: $\mathcal{P}$rv State Machine

4.1 TRAIN$_A$: RTC-Based $\mathcal{N}$A Technique

Commodity RTCs, such as MCP7940MT-I/SM [54], are now readily available for under $0.60 per unit. This affordability marks a significant shift from the past, when real-time security features were often too costly for IoT devices. This motivates our design of an $\mathcal{N}$A protocol for devices with RTCs. We begin by presenting this simple variant of the core ideas of TRAIN. An alternative variant without the RTC requirement is described in Section 4.2. TRAIN$_A$ has two message types:

Attestation Request ($Att_{request}$): Generated by $\mathcal{V}$rf, it contains: $Hash_{New}$, $HashInd_{New}$, and t_{attest}, which are used to authenticate $Att_{request}$. Note that $Hash_{New}$ is used as a challenge for this $\mathcal{N}$A. $Att_{request}$ also includes the packet type field: *"req"* and the identifier ID_{Snd} of either $\mathcal{V}$rf that originated it (for the first hop), or a $\mathcal{P}$rv that forwards it (for subsequent hops). ID_{Snd} is used by each receiving $\mathcal{P}$rv to learn its parent in the spanning tree.

Attestation Report (Att_{report}): Generated by each $\mathcal{P}$rv, this message carries the attestation report. It contains an authentication token ($Auth_{report}$), which provides message integrity. $LMT_{Dev}{}'$ is included in the calculation of $Auth_{report}$ and in Att_{report} only for RATA-enabled $\mathcal{P}$rv-s. Similar to $Att_{request}$, Att_{report} also includes the packet type field: *"rep"* and the identifier of $\mathcal{P}$rv that generated this report. Also, Att_{report} includes $Hash_{New}$ that was received in $Att_{request}$ and the actual time ($t_{attest}{}'$) when attestation is performed. We now describe $\mathcal{P}$rv operation as a state machine with five states, as shown in Figure 2: *Idle, Verify, Attest-Wait, Attest,* and *Forward-Wait*.

P1. Idle: $\mathcal{P}$rv runs normally. Upon receiving an $Att_{request}$, it proceeds to **Verify**. Any Att_{report} received in this state is discarded.

P2. Verify:

P2.1: $\mathcal{P}$rv checks if $HashInd_{Cur} > HashInd_{New}$ and $t_{attest} > T$, where T is its current RTC value. If either check fails, it discards $Att_{request}$ and returns to **Idle**.

P2.2: $\mathcal{P}$rv computes and checks whether $H^s(Hash_{New}) \stackrel{?}{=} Hash_{Cur}$, where $s = HashInd_{Cur} - HashInd_{New}$.[4] If not, it discards $Att_{request}$ and returns to **Idle**. (Note that a $\mathcal{P}$rv might receive duplicate $Att_{request}$-s from multiple neighbors; it simply discards them.)

[3] For clarity and completeness, a detailed pseudo-code implementation of both TRAIN$_A$ and TRAIN$_B$ is provided in the extended version of this paper, available at https://www.arXiv:2502.07053.

[4] Recall that it is possible for $s > 1$, (as discussed at the end of Section 2.5) meaning that $\mathcal{P}$rv became de-synchronized. Also, $HashInd_{New}$ is decremented by one in every $\mathcal{R}$A instance.

Notation	Meaning
ID_{Dev}	Identifier of responding $\mathcal{P}$rv
ID_{Par}	Identifier of responding $\mathcal{P}$rv's parent
ID_{Snd}	Identifier of the sending device
$H()$	Cryptographic hash function (e.g., SHA2-256) used in hash chain computation
$H^s(x)$	Denotes $s > 1$ repeated applications of $H()$ starting with initial input x
$Hash_{New}$	Hash value sent by $\mathcal{V}$rf that authenticates it to all $\mathcal{P}$rv-s; it also serves as the challenge for this $\mathcal{N}$A instance
$Hash_{Cur}$	Current hash value stored by $\mathcal{P}$rv
$HashInd_{New}$	Index of $Hash_{New}$ sent by $\mathcal{V}$rf
$HashInd_{Cur}$	Index of $Hash_{Cur}$ stored by $\mathcal{P}$rv
$Height_{Net}$	Network spanning tree height
$Height_{Cur}$	Height of $\mathcal{P}$rv in the spanning tree
LMT_{Dev}	Last Modification Time (of PMEM), only used in RATA, stored on $\mathcal{V}$rf
$LMT_{Dev}{}'$	Last Modification Time (of PMEM), only used in RATA, stored on $\mathcal{P}$rv
$\mathcal{K}_{Dev}$	Shared key between $\mathcal{P}$rv and $\mathcal{V}$rf, securely stored on $\mathcal{P}$rv and restricted to its trusted attestation code
$Att_{request}$	Attestation request message ($\mathcal{V}$rf $\rightarrow$ $\mathcal{P}$rv): $["req", ID_{Snd}, Hash_{New}, HashInd_{New}, t_{attest}]$
Att_{report}	Attestation report message ($\mathcal{V}$rf $\leftarrow$ $\mathcal{P}$rv): $["rep", ID_{Dev}, ID_{Par}, t_{attest}{}', Hash_{New}, \{LMT_{Dev}\}, Auth_{report}]$
$Auth_{report}$	Authentication of attestation report in Att_{report}: $\text{MAC}(\mathcal{K}_{Dev}, ID_{Par}, t_{attest}{}', Hash_{New}, \{LMT_{Dev}{}'\})$
$t_{request}$	propagation delay of $Att_{request}$
t_{report}	propagation delay of Att_{report}
t_{hash}	Computation time for $Att_{request}$ verification
t_{MAC}	Computation time for MAC generation
t_{slack}	Additional slack time
$t_{maxDelay}$	Max delay to receive an Att_{report} from a descendant $\mathcal{P}$rv
t_{attest}	Time to begin attestation, set by $\mathcal{V}$rf
$t_{attest}{}'$	Time when a given $\mathcal{P}$rv actually performed attestation
$t_{timeout}$	$\mathcal{V}$rf's timeout for receiving all attestation replies

Table 1: Notation Summary

P2.3: $\mathcal{P}$rv replaces: $Hash_{Cur}$ with $Hash_{New}$, and $HashInd_{Cur}$ with $HashInd_{New}$. Then, $\mathcal{P}$rv stores ID_{Snd} as ID_{Par}, sets ID_{Snd} field of received $Att_{request}$ to its ID_{Dev}, and broadcasts modified $Att_{request}$.

P2.4: $\mathcal{P}$rv sets (using its RTC) a secure timer (TimerTCB) to t_{attest} and transitions to **Attest-Wait**.

P3. Attest-Wait: $\mathcal{P}$rv runs normally while the timer is ticking. If any $Att_{request}$ is received in this state, it is discarded.

P4. Attest: When the timer matches t_{attest}, $\mathcal{P}$rv sets $t_{attest}{}'$ to the current RTC value, computes $Auth_{report}$, and composes Att_{report} as defined above. It then uni-casts Att_{report} to ID_{Par}, sets the timer to $t_{maxDelay}$, and transitions to **Forward-Wait**.

P5. Forward-Wait: $\mathcal{P}$rv runs normally while the timer is ticking. If $\mathcal{P}$rv receives an Att_{report}, it checks whether the report's $Hash_{New}$ matches that previously received in **Verify**. If not, it is discarded. Otherwise, $\mathcal{P}$rv uni-casts received Att_{report} to its parent and remains in **Forward-Wait**. When the timer matches $t_{maxDelay}$, $\mathcal{P}$rv transitions to **Idle**. Note that any $Att_{request}$ received while in this state is discarded.

Whereas, as shown in Figure 3, $\mathcal{V}$rf's state machine has four states: *Idle, Initiate, Collect,* and *Tally*.

V1. Idle: $\mathcal{V}$rf waits for an external signal to begin an attestation instance. When it occurs, $\mathcal{V}$rf transitions to **Initiate**.

V2. Initiate: $\mathcal{V}$rf assigns t_{attest}, as described in Section 4.4, which accounts for request propagation and network height. (t_{attest} is computed by each $\mathcal{P}$rv in TRAIN$_B$.) It then initializes *Attest*=*Fail*=$\emptyset$, and *NoRep*=$\{ID_{Dev}$-s of all $\mathcal{P}$rv-s$\}$. Next, $\mathcal{V}$rf sets ID_{Snd} to $\mathcal{V}$rf, composes $Att_{request}$, and broadcasts it. (Recall the assumption that at least one $\mathcal{P}$rv must be within broadcast range of $\mathcal{V}$rf at this time.) It then sets a local timer to $t_{timeout}$, as detailed in Section 4.4, which factors into network size and delays, and then transitions to **Collect**.

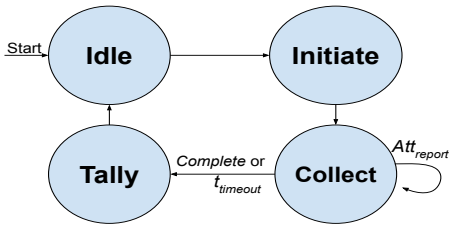


Figure 3: $\mathcal{V}$rf State Machine

V3. Collect: $\mathcal{V}$rf waits for Att_{report}-s. Upon receipt of an Att_{report}, $\mathcal{V}$rf first checks that $Hash_{New}$ contained in Att_{report} matches that in the currently pending $Att_{request}$; otherwise, it is discarded. Next, $\mathcal{V}$rf validates Att_{report} by looking up the corresponding $\mathcal{K}_{Dev}$ shared with $\mathcal{P}$rv (identified by ID_{Dev}) and recomputing the MAC. If MAC validation fails, Att_{report} is discarded. Otherwise:

V3.1 (CASU): ID_{Dev} is moved from *NoRep* to *Attest*.

V3.2 (RATA): $\mathcal{V}$rf maintains the last valid LMT_{Dev} for each $\mathcal{P}$rv. When processing an Att_{report} from a given $\mathcal{P}$rv, $\mathcal{V}$rf compares received $LMT_{Dev}{}'$ with the stored LMT_{Dev} for that $\mathcal{P}$rv. A mismatch signifies failed attestation and ID_{Dev} is added to *Fail*. Otherwise, it is added to *Attest*. In either case, ID_{Dev} is removed from *NoRep*.

V3.3 If *NoRep* = $\emptyset$, $\mathcal{V}$rf transitions to **Tally**.

V3.4 If $t_{timeout}$ timer expires, $\mathcal{V}$rf transitions to **Tally**.

V4. Tally: $\mathcal{V}$rf outputs *Attest*, *Fail*, and *NoRep*. It then returns to **Idle**.

4.2 TRAIN$_B$: Clockless $\mathcal{N}$A Technique

Despite its relatively low cost, an RTC might still not be viable for some IoT deployments. This leads us to construct a TRAIN variant without RTCs.

There are still just two message types, $Att_{request}$ and Att_{report}, of which only $Att_{request}$ differs from TRAIN$_A$:

Attestation Request ($Att_{request}$): Generated by $\mathcal{V}$rf, $Att_{request}$ includes two extra fields: $Height_{Cur}$ and $Height_{Net}$ which represent the height of the sender ($\mathcal{V}$rf or $\mathcal{P}$rv) and the spanning tree height of the network, respectively. $Height_{Cur}$ is essentially a hop counter during the propagation of $Att_{request}$ throughout the network. It is initialized to 0 by $\mathcal{V}$rf and incremented by each forwarding $\mathcal{P}$rv.

$\mathcal{P}$rv's state machine has five states, three of which are identical to those in TRAIN$_A$. Only Verify and Attest differ, as follows:

P2. Verify:

P2.1: $\mathcal{P}$rv checks whether $HashInd_{Cur} > HashInd_{New}$. If this check fails, it discards $Att_{request}$ and returns to **Idle**.

P2.2: $\mathcal{P}$rv computes and checks whether $H^s(Hash_{New}) \stackrel{?}{=} Hash_{Cur}$, where $s = HashInd_{Cur} - HashInd_{New}$. If not, it discards $Att_{request}$ and returns to **Idle**. Duplicate $Att_{request}$-s from multiple neighbors are also discarded.

P2.3: $\mathcal{P}$rv replaces: $Hash_{Cur}$ with $Hash_{New}$, and $HashInd_{Cur}$ with $HashInd_{New}$. Then, $\mathcal{P}$rv stores ID_{Snd} as ID_{Par}, sets ID_{Snd}

field to its ID_{Dev}, increments $Height_{Cur}$, and broadcasts modified $Att_{request}$.

P2.4: $\mathcal{P}$rv sets a secure timer (TimerTCB) to:

$$attestWait = (Height_{Net} - Height_{Cur}) * (t_{request} + t_{hash}) \quad (1)$$

and transitions to **Attest-Wait**.

P4. Attest: Identical to TRAIN_A, except that $\mathcal{P}$rv sets $t_{attest}{}'$ to its current secure timer value, to be later validated by $\mathcal{V}$rf. The degree of reduction of $\text{TOCTOU}_{\mathcal{N}\text{A}}$ depends on the accuracy and functionality of $\mathcal{P}$rv's secure timer. Also, the propagation delay from $\mathcal{V}$rf to each $\mathcal{P}$rv affects $\text{TOCTOU}_{\mathcal{N}\text{A}}$. This is discussed in more detail in Section 6.1.

Note that t_{attest} in TRAIN_B is a timer value (increases from 0), unlike that in TRAIN_A, which represents the current time. $\mathcal{V}$rf's state machine has four states identical to that of TRAIN_A.

Protocol Trade-offs: TRAIN's two variants address distinct deployment constraints. TRAIN_A uses RTCs to synchronize attestation timing globally via precise timestamps (t_{attest}), thus minimizing $\text{TOCTOU}_{\mathcal{N}\text{A}}$ with marginal hardware costs. Whereas, TRAIN_B eliminates RTC dependencies by deriving attestation timing from the network topology ($Height_{Net}$), thus sacrificing $\text{TOCTOU}_{\mathcal{N}\text{A}}$ precision for broader applicability. These synchronization implications are addressed in Section 7.3.

4.3 Renewing Hash Chains

As typical for any technique utilizing hash chains, the issue of chain depletion must be addressed. An m-link hash chain is depleted after m authentication instances (m $\mathcal{N}$A instances in our context). To address this issue and ensure long-term operation, we need a mechanism for refreshing the hash chain.

Recall the well-known Lamport hash chain construct from Section 2.5. Suppose that the current hash chain of length m being used is $\mathcal{X}$:

$$H(x_0) = x_1, H(x_1) = x_2, ..., H(x_{m-1}) = x_m$$

Suppose that we have already used up $m-2$ links of the chain for all $\mathcal{P}$rv-s. This means that only two links in the chain remain, and the entire chain will be depleted when $\mathcal{V}$rf reveals x_1 and then x_0 in the next two $\mathcal{N}$A instances. Knowing this, $\mathcal{V}$rf wants all $\mathcal{P}$rv-s to switch over to a new hash chain $\mathcal{X}'$:

$$H(x'_0) = x'_1, H(x'_1) = x'_2, ..., H(x'_{m-1}) = x'_m$$

To do so, it includes in the next $Att_{request}$ ($Att_{request_{m-1}}$) two extra values/fields:

$$Att_{request_{m-1}} = [\text{“}req\text{”}, ID_{Snd}, Hash_{New}{=}x_1, HashInd_{New}{=}1, t_{attest}, \textbf{NewChain=}x'_m\textbf{, Auth=}MAC(x_0, x'_m)\ ]$$

These two new fields convey the anchor of the new hash chain **NewChain** and its authenticator **Auth** computed as a MAC over NewChain using, as a key, still-unreleased next link in the current chain – x_0. Upon receiving such an $Att_{request}$, in addition to the usual $Att_{request}$ processing, a $\mathcal{P}$rv stores NewChain and Auth. Obviously, at this time, a $\mathcal{P}$rv has no way to verify Auth since it does not yet know x_0. A $\mathcal{P}$rv continues to process this $Att_{request}$, as detailed earlier.

However, at this stage, each $\mathcal{P}$rv maintains a current hash $\mathcal{X}$, where $HashInd_{Cur} = 1$ and $Hash_{Cur} = x_1$. A $\mathcal{P}$rv waits for the next $\mathcal{N}$A instance, wherein $Att_{request_m}$ should convey x_0. Upon receiving $Att_{request_m}$, a $\mathcal{P}$rv obtains x'_0, which may differ from the original x_0 if it was modified by $\mathcal{A}$dv in transit. As part of its normal processing, a $\mathcal{P}$rv first verifies that $H(x'_0) = Hash_{Cur} = x_1$. A $\mathcal{P}$rv recomputes Auth′ using the newly received x'_0 and its stored NewChain value. If Auth′ matches the previously stored Auth, a $\mathcal{P}$rv completes the switchover to the chain $\mathcal{X}'$ by setting $HashInd_{Cur} = m$ and $Hash_{Cur} = x'_m$.

This simple renewal technique is secure, lightweight, and trivial to implement. However, two factors contribute to its fragility.

Timing: It must hold that the time difference between $\mathcal{V}$rf sending $Att_{request_{m-1}}$ and $Att_{request_m}$ is sufficiently long to avoid forgeries of NewChain and Auth. However, even when the time difference is reasonably long, $\mathcal{A}$dv can delay the delivery of $Att_{request_{m-1}}$ to one or more targeted $\mathcal{P}$rv-s. If $Att_{request_m}$ is sent by $\mathcal{V}$rf when at least one $\mathcal{P}$rv has not yet received $Att_{request_{m-1}}$, $\mathcal{A}$dv can learn x_0 from $Att_{request_m}$. It can then change the NewChain field in $Att_{request_{m-1}}$ from x'_m to y_m, and Auth field – from $MAC(x_0, x'_m)$ to $MAC(x_0, y_m)$, where y_m is the anchor of $\mathcal{A}$dv-selected hash chain.

This issue is not unique to the present technique. It is indeed quite similar to the timing requirement in the well-known TESLA protocol for secure multicast and its many variants [46]. TESLA also uses the delayed key disclosure mechanism and makes reasonable assumptions about timing.[5] The timing issue can be further mitigated if $\mathcal{V}$rf switches the chain in $Att_{request_m}$ only if it has received legitimate responses from all $\mathcal{P}$rv-s upon sending $Att_{request_{m-1}}$.

DoS on $\mathcal{P}$rv-s: Upon observing $Att_{request_{m-1}}$, $\mathcal{A}$dv (present in the network) can modify NewChain and/or Auth fields. Each $\mathcal{P}$rv would then duly store these two values. Once the subsequent $Att_{request_m}$ arrives in the next $\mathcal{N}$A instance, each $\mathcal{P}$rv would fail to verify stored NewChain and Auth, thus ending up being unable to process any further $Att_{request}$-s. Although there is no full-blown fix for this problem, one way to side-step it is for $\mathcal{V}$rf to begin switching to the new hash chain prior to a few links being left in the old chain, i.e., when $Hash_{Cur} = (m-k)$ for some reasonably small k. In this case, Auth = $MAC(x_{m-k-1}, x'_m)$, which can be verified in the successive attestation, $Att_{request_{m-k-1}}$. Then, $\mathcal{V}$rf can decide to switch to $\mathcal{X}'$ when it receives valid Att_{report}-s from all $\mathcal{P}$rv-s, indicating that all have the identical NewChain, x'_m.

4.4 Timeouts

The overall attestation timeout (on $\mathcal{V}$rf) is set as follows:

$$t_{timeout} = n * (t_{request} + t_{hash} + t_{report}) + t_{MAC} + t_{slack} \quad (2)$$

where n is the total number of $\mathcal{P}$rv-s in the network. $\mathcal{V}$rf sets the attestation time in TRAIN_A as follows:

$$t_{attest} = Height_{Net} * (t_{request} + t_{hash}) + t_{slack} + t_{current} \quad (3)$$

where $t_{current}$ is $\mathcal{V}$rf's current time.

t_{attest} must be large enough for every $\mathcal{P}$rv to receive $Att_{request}$ before the actual attestation begins. Note that an inflated t_{attest} does not influence $\text{TOCTOU}_{\mathcal{N}\text{A}}$; it only incurs $\mathcal{V}$rf's waiting time. In the worst case (line topology), the total request propagation time would be: $n * (t_{request} + t_{hash})$. Once all devices receive the request, they perform attestation at (ideally) the same time t_{attest}, taking t_{MAC}. Finally, Att_{report}-s from all $\mathcal{P}$rv-s need to be returned to $\mathcal{V}$rf, which

[5] See Section 2.2 in IETF RFC 4082: https://www.ietf.org/rfc/rfc4082.txt).

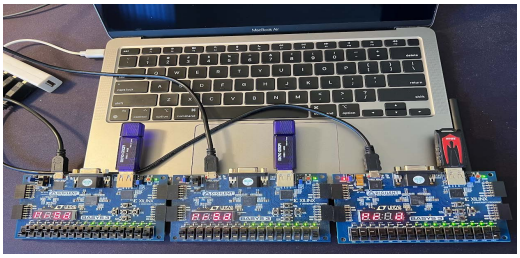


Figure 4: TRAIN **Proof-Of-Concept with Three $\mathcal{P}$rv-s**

takes at most $n{*}t_{report}$ in the worst case. Note that t_{report} may differ from $t_{request}$ due to network congestion caused by simultaneous response transmissions from all $\mathcal{P}$rv-s. An additional tolerance value t_{slack} helps account for unexpected delays.

5 Implementation

TRAIN is prototyped atop openMSP430[43], an open-source implementation of TI MSP430 MCU, written in Verilog HDL. OpenMSP430 can execute software generated by any MSP430 toolchain [55] with near-cycle accuracy. We extended both RATA and CASU architectures to support TRAIN. In this implementation, $\mathcal{P}$rv and $\mathcal{V}$rf are connected via UART.

5.1 TRAIN Software

Using the native msp430-gcc toolchain, TRAIN software on $\mathcal{P}$rv is compiled to generate software images compatible with the memory layout of the modified openMSP430. TRAIN software, responsible for processing TRAIN protocol messages and generating attestation responses, is housed in ROM. NetTCB is triggered whenever a TRAIN protocol message is received; this is determined by the cleartext message type in the header.

Also, TimerTCB is triggered to start attestation whenever the timer expires in the **Attest-Wait** state. For cryptographic operations we use a formally verified cryptographic library, HACL* [25]. It provides high-assurance implementations of essential cryptographic primitives, such as hash functions and MAC-s. SHA2-256 and HMAC are used for hash and MAC, respectively. Both RATA and CASU implement their respective cryptographic operations using HACL*.

To emulate $\mathcal{V}$rf, we developed a Python application with $\approx$ 200 lines of code, as described in Sections 4.1 and 4.2. The application runs on an Ubuntu 20.04 LTS laptop with an Intel i5-11400 processor @2.6GHZ with 16GB of RAM.

5.2 TRAIN Hardware

As mentioned earlier, $\mathcal{P}$rv-s in TRAIN can adopt either CASU or RATA architecture, possibly equipped with different system resources (e.g., CPU clock, memory, peripherals). We refer to CASU-based $\mathcal{P}$rv-s as $\textbf{TRAIN}_{\text{CASU}}$ and RATA-based $\mathcal{P}$rv-s as $\textbf{TRAIN}_{\text{RATA}}$. We implemented and evaluated both as part of the proof-of-concept.

The design is synthesized using Xilinx Vivado 2023.1, a popular logic synthesis tool. It generates the hardware implementation for the FPGA platform. The synthesized design is then deployed on a Basys3 Artix-7 FPGA board for prototyping and evaluating hardware design.

Figure 4 shows a proof-of-concept implementation of TRAIN. In it, three $\mathcal{P}$rv-s (implemented on Basys3 FPGA boards) are connected to $\mathcal{V}$rf. For the sake of simplicity, $\mathcal{P}$rv-s are deployed using a star topology for signal routing. All three $\mathcal{P}$rv-s in Figure 4 are $\text{TRAIN}_{\text{CASU}}$ devices. However, we also implemented TRAIN with $\text{TRAIN}_{\text{RATA}}$ devices for performance evaluation.

6 Evaluation

6.1 Security Analysis

Network-based $\mathcal{A}$dv: This adversary ($\mathcal{A}$dv) is a malicious (not a TRAIN $\mathcal{P}$rv) physical network entity, e.g., a non-compliant IoT device or a computer.

$Att_{request}$ in TRAIN_{A} includes: *"req"*, ID_{Snd}, $Hash_{New}$, $HashInd_{New}$, t_{attest}, while $Att_{request}$ in TRAIN_{B} also includes $Height_{Cur}$ and $Height_{Net}$. $\mathcal{P}$rv authenticates each $Att_{request}$ by verifying $HashInd_{New}$, t_{attest}, and checking if $H^s(Hash_{New}) = Hash_{Cur}$, where $s = HashInd_{Cur} - HashInd_{New}$. The $Hash_{New}$ is known only to $\mathcal{V}$rf, and recovering it from $Hash_{Cur}$ is computationally infeasible, so $\mathcal{A}$dv cannot forge $Hash_{New}$. However, $\mathcal{A}$dv can modify other fields (such as t_{attest}, $Height_{Cur}$, and $Height_{Net}$) affecting $\mathcal{P}$rv's attestation time. Nonetheless, this is later detected by $\mathcal{V}$rf, as $t_{attest}{}'$ is included in $Auth_{report}$ within each $\mathcal{P}$rv's Att_{report}.

$\mathcal{A}$dv can also alter the ID_{Snd} field in $Att_{request}$, supplying an incorrect ID_{Par} to $\mathcal{P}$rv. This may obstruct valid Att_{report}-s from benign $\mathcal{P}$rv-s. However, $\mathcal{V}$rf will notice the absence of Att_{report} from affected $\mathcal{P}$rv-s.

Att_{report} includes: *"rep"*, ID_{Dev}, ID_{Par}, $t_{attest}{}'$, $Hash_{New}$, $\{LMT_{Dev}{}'\}$, and $Auth_{report}$, with authenticity and integrity ensured by $Auth_{report}$, computed as:
MAC($\mathcal{K}_{Dev}$,ID_{Par},$t_{attest}{}'$,$Hash_{New}$,$\{LMT_{Dev}{}'\}$). Manipulation of ID_{Dev} is detectable by $\mathcal{V}$rf since $ID_{Dev}$is used to retrieve the corresponding key.

$\mathcal{A}$dv can forge an Att_{report} only if: (1) $\mathcal{A}$dv forges $Auth_{report}$ without knowing $\mathcal{K}_{Dev}$, which is infeasible with a secure MAC function, or (2) $\mathcal{A}$dv learns $\mathcal{K}_{Dev}$ and constructs an authentic $Auth_{report}$, which is infeasible since $\mathcal{K}_{Dev}$ is in the TCB and is only accessible to TRAIN software.

Malware-based $\mathcal{A}$dv: TRAIN remains secure despite malware presence on any number of $\mathcal{P}$rv-s due to: (1) $\mathcal{P}$rv's TCB ensuring $\mathcal{K}_{Dev}$ secrecy, (2) NetTCB enforcing receiving and forwarding of $Att_{request}$ and Att_{report} (3) TimerTCB ensuring timely Att_{report} generation, and (4) $\mathcal{P}$rv's TCB blocking non-TCB software from sending Att_{report} or $Att_{request}$ messages. These measures prevent DoS attacks from $\mathcal{P}$rv-resident malware.

$\text{TOCTOU}_{\mathcal{NA}}$ & $\text{TOCTOU}_{\mathcal{RA}}$: TRAIN_{A} eliminates $\text{TOCTOU}_{\mathcal{NA}}$ as long as the RTC is accurately synchronized with the $\mathcal{V}$rf. Meanwhile, minimizing $\text{TOCTOU}_{\mathcal{NA}}$ in TRAIN_{B} depends on: (a) the secure timer, and (b) propagation delay from $\mathcal{V}$rf to each $\mathcal{P}$rv. Two scenarios relating to the former could increase $\text{TOCTOU}_{\mathcal{NA}}$ in TRAIN_{B}: (a-1) $\mathcal{A}$dv tampering with a $\mathcal{P}$rv's secure timer, or (a-2) timer drift due to physical imperfections, disrupting the attestation schedule.

To address (a-1), TRAIN_{B} uses TimerTCB to: (1) prioritize the timer's Interrupt Service Routine (ISR) for timely attestation, and (2) protect timer configurations from unauthorized changes. Although (a-2) can't be fully addressed, TRAIN_{B} significantly reduces $\text{TOCTOU}_{\mathcal{NA}}$ compared to unsynchronized schemes. For example,

Notation	Description
PC	Program Counter pointing to the current instruction being executed
W_{en}	1-bit signal that represents whether MCU core is writing to memory
D_{addr}	Memory address being accessed by MCU core
DMA_{en}	1-bit signal that represents whether DMA is active
DMA_{addr}	Memory address being accessed by DMA
$reset$	Signal that reboots the MCU when it is set to logic '1'
TCR	Trusted code region, a fixed ROM region storing $\mathsf{TRAIN}_{\mathsf{CASU}}$ SW
ER	Executable region, memory region where authorized (normal) software is stored
EP	Executable pointer, a fixed memory region storing current ER boundary
$IVTR$	Reserved memory region for the MCU's interrupt vector table
M_{TRAIN}	Memory region protected by $\mathsf{TRAIN}_{\mathsf{CASU}}$ hardware, including ER, EP, and $IVTR$
gie	Global interrupt enable, 1-bit signal that represents whether interrupts are globally enabled
irq	1-bit signal that represents if an interrupt occurs
IRQ_{cfg}	Set of registers in DMEM used to configure of interrupts, e.g., timer deadline and UART baudrate
ISR_T	Timer interrupt service routine: $ISR_T = [ISR_{T_{min}}, ISR_{T_{max}}]$
ISR_U	UART interrupt service routine: $ISR_U = [ISR_{U_{min}}, ISR_{U_{max}}]$

Table 2: Notation Summary

with a propagation delay $t_{request}$ = 1ms and $Height_{Net}$= 10,000, $\mathcal{P}$rv-s wait for up to 10s. A timer drift of 100ppm results in a 1ms drift, reducing $\mathsf{TOCTOU}_{\mathcal{NA}}$ from 10,000ms to 1ms in $\mathsf{TRAIN}_{\mathsf{B}}$.

Recall that $\mathsf{TRAIN}_{\mathsf{B}}$ assumes identical network propagation delays. However, in reality, variations may occur due to congestion or connectivity changes. For instance, with $Height_{Net}$= 10,000 and $t_{request}$ = 1ms, if the delay between $\mathcal{P}\mathsf{rv}_i$ and $\mathcal{P}\mathsf{rv}_j$ is 1.5ms, $\mathcal{P}\mathsf{rv}_i$ starts attestation 0.5ms earlier than its descendants. To minimize this, (b), $t_{request}$ should average all network propagation delays. Note that $Height_{Net}$ over-estimation by $\mathcal{V}$rf doesn't affect $\mathsf{TOCTOU}_{\mathcal{NA}}$; it only delays attestation start on $\mathcal{P}$rv-s.

6.2 $\mathsf{TRAIN}_{\mathsf{CASU}}$ Formal Verification

We formally specify $\mathsf{TRAIN}_{\mathsf{CASU}}$ with $\mathsf{TRAIN}_{\mathsf{B}}$ security goals using Linear Temporal Logic (LTL). Formal verification plays a crucial role by showing that $\mathsf{TRAIN}_{\mathsf{CASU}}$ adheres to well-specified goals. It assures that it does not exhibit any unintended behavior, especially in corner cases, rarely encountered conditions and/or execution paths, that humans tend to overlook. By employing computer-aided tools, we define and validate LTL rules that govern $\mathsf{TRAIN}_{\mathsf{CASU}}$ operation. The use of LTL enables precisely capturing temporal dependencies and expected behavior over time, ensuring that $\mathsf{TRAIN}_{\mathsf{CASU}}$ meets stringent security standards. Table 2 describes the notation used in this section. We use regular propositional logic, such as conjunction $\wedge$, disjunction $\vee$, negation $\neg$, and implication $\rightarrow$. A few other temporal quantifiers are used as well:

- **X**Φ (ne**X**t) – holds if Φ=true at the next system state.
- **F**Φ (**F**uture) – holds if there is a future state when Φ=true.
- **G**Φ (**G**lobally) – holds if for all future states Φ=true.

Figure 5 formally describes $\mathsf{TRAIN}_{\mathsf{CASU}}$ hardware security properties using propositional logic and temporal quantifiers. Recall that $\mathsf{TRAIN}_{\mathsf{CASU}}$ is based on CASU combined with GAROTA. All such properties must hold at all times to achieve $\mathsf{TRAIN}_{\mathsf{CASU}}$'s security goals.

LTL 4 states that any modifications to M_{TRAIN}, including ER, EP, and $IVTR$, trigger a reset when $\mathsf{TRAIN}_{\mathsf{CASU}}$ software is not running. ER is a region in PMEM, where normal device software resides, while EP is a fixed region in PMEM that points to ER. Upon a secure update, EP is updated to point to the new verified software version. $IVTR$ also resides in PMEM and contains the ISR addresses. As stated in LTL 5, the MCU cannot execute any code outside ER or $\mathsf{TRAIN}_{\mathsf{CASU}}$ code in read-only memory (ROM).

LTL 6 ensures that, if the timer or the UART peripheral configurations are modified by any software (other than the timer or UART ISR-s), a reset is triggered. LTL 7-9 specify atomic operation of timer ISR, LTL 7 and LTL 8 guarantee that $ISR_{T_{min}}$ and $ISR_{T_{max}}$ are the only legal entry and exit points, respectively. Also, LTL 9 states that DMA and other interrupts must remain inactive while timer ISR executes. Similarly, LTL 10-12 enforce UART ISR atomicity. Finally, LTL 13 guarantees that gie can be disabled only if the timer or UART ISR-s are running. Any violations result in a device reset. Note that we slightly modified CASU and GAROTA to realize $\mathsf{TRAIN}_{\mathsf{CASU}}$:

(1) $\mathsf{TRAIN}_{\mathsf{CASU}}$ employs both TimerTCB and NetTCB, while GAROTA uses them individually in each case.
(2) *Trusted PMEM Updates* rule from GAROTA is integrated to Equation 4.
(3) GAROTA's *Re-Trigger on Failure* property is not viable since $\mathsf{TRAIN}_{\mathsf{CASU}}$ cannot retain a consistent timer value upon a failure (e.g., a reset) in $\mathsf{TRAIN}_{\mathsf{B}}$.

To verify the above LTL rules, we convert the Verilog code described at the Register Transfer Level (RTL) to Symbolic Model Verifier (SMV) [36] using Verilog2SMV [24]. The SMV output is in turn fed to the NuSMV [13] model-checker for specified LTL rule validation. NuSMV works by checking LTL specifications against the system finite-state machine for all reachable states. This comprehensive approach ensures that $\mathsf{TRAIN}_{\mathsf{CASU}}$'s security goals are thoroughly validated, offering robust assurance against potential vulnerabilities. See [45] for further proof details.

6.3 Hardware Overhead

Recall that underlying hardware RoT for $\mathcal{P}$rv-s in TRAIN is either CASU or RATA with additional hardware support from GAROTA. Table 3 compares the hardware overhead of $\mathsf{TRAIN}_{\mathsf{CASU}}$ and $\mathsf{TRAIN}_{\mathsf{RATA}}$ implementations with the baseline openMSP430, CASU, and RATA architectures. $\mathsf{TRAIN}_{\mathsf{CASU}}$ implementation requires 0.46% more Look-Up Tables (LUTs) and 0.55% more registers over CASU. Also, $\mathsf{TRAIN}_{\mathsf{RATA}}$ implementation needs 0.05% LUTs and 0.69% registers over RATA. Numbers of additional LUTs and registers are under 15, implying minimal overheads incurred by NetTCB and TimerTCB.

Comparison with Other Hybrid *RoT*: We compare TRAIN with other hybrid *RoT* constructions leveraging $\mathcal{R}$A: VRASED [40], RATA [15], CASU [14], GAROTA [3], and APEX [41]. Note that RATA, CASU, APEX are implemented based on VRASED, and all the above architectures are (in turn) based on openMSP430. Results are shown in Figure 6. APEX has a higher overhead than others due to additional hardware properties required for generating proofs-of-execution.

• *Security Properties Stemming from* CASU

- Software Immutability in PMEM:

$$\mathbf{G} : \{\text{modMem}(M_{\text{TRAIN}}) \wedge (PC \notin TCR) \rightarrow reset\} \quad (4)$$

- Unauthorized Software Execution Prevention:

$$\mathbf{G} : \{(PC \notin ER) \wedge (PC \notin TCR) \rightarrow reset\} \quad (5)$$

• *Security Properties Stemming from* GAROTA

- IRQ Configuration Protection:

$$\mathbf{G} : \{[\neg(PC \in TCR) \wedge W_{en} \wedge (D_{addr} \in IRQ_{cfg})] \vee [DMA_{en} \wedge (DMA_{addr} \in IRQ_{cfg})] \rightarrow reset\} \quad (6)$$

- Timer ISR Execution Atomicity:

$$\mathbf{G} : \{\neg reset \wedge \neg(PC \in ISR_T) \wedge (\mathbf{X}(PC) \in ISR_T) \rightarrow \mathbf{X}(PC) = ISR_{T_{min}} \vee \mathbf{X}(reset)\} \quad (7)$$

$$\mathbf{G} : \{\neg reset \wedge (PC \in ISR_T) \wedge \neg(\mathbf{X}(PC) \in ISR_T) \rightarrow PC = ISR_{T_{max}} \vee \mathbf{X}(reset)\} \quad (8)$$

$$\mathbf{G} : \{(PC \in ISR_T) \wedge (irq \vee DMA_{en}) \rightarrow reset\} \quad (9)$$

- UART ISR Execution Atomicity:

$$\mathbf{G} : \{\neg reset \wedge \neg(PC \in ISR_U) \wedge (\mathbf{X}(PC) \in ISR_U) \rightarrow \mathbf{X}(PC) = ISR_{U_{min}} \vee \mathbf{X}(reset)\} \quad (10)$$

$$\mathbf{G} : \{\neg reset \wedge (PC \in ISR_U) \wedge \neg(\mathbf{X}(PC) \in ISR_U) \rightarrow PC = ISR_{U_{max}} \vee \mathbf{X}(reset)\} \quad (11)$$

$$\mathbf{G} : \{(PC \in ISR_U) \wedge (irq \vee DMA_{en}) \rightarrow reset\} \quad (12)$$

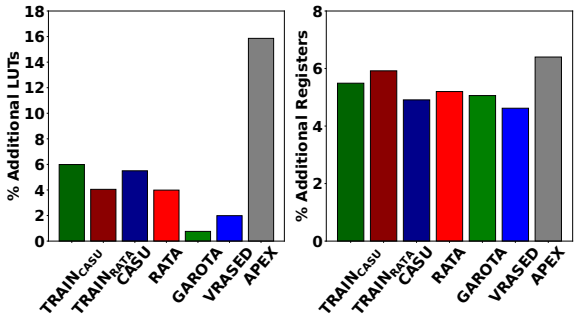
- Interrupt Disablement Protection:

$$\mathbf{G} : \{\neg reset \wedge gie \wedge \neg\mathbf{X}(gie) \rightarrow (\mathbf{X}(PC) \in (ISR_T \vee ISR_U)) \vee \mathbf{X}(reset)\} \quad (13)$$

Figure 5: $\text{TRAIN}_{\text{CASU}}$ **Hardware Security Properties**

Architecture	Look-Up Tables	Registers
openMSP430	1854	692
CASU	1956	726
$\text{TRAIN}_{\text{CASU}}$	1967 (+11)	740 (+14)
RATA	1928	728
$\text{TRAIN}_{\text{RATA}}$	1935 (+7)	737 (+9)

Table 3: TRAIN **Hardware Overhead**



(a). Additional LUTs (%) (b). Additional Reg-s (%)

Figure 6: Hardware Overhead Comparison

Architecture	Request Verification Time (ms)	Report Generation Time (ms)
$\text{TRAIN}_{\text{CASU}}$ (@ 8MHz)	13.0	29.5
$\text{TRAIN}_{\text{RATA}}$ (@ 8MHz)	12.9	29.8
SEDA Initiator (SMART) (@ 8MHz)	N/A	$56900 + 256 * g$
SEDA participating devices (SMART) (@ 8MHz)	N/A	$96 + 256 * (g - 1)$
SEDA Initiator (TRUSTLITE) (@ 24MHz)	N/A	$347.2 + 4.4 * g$
SEDA participating devices (TRUSTLITE) (@ 24MHz)	N/A	$0.6 + 4.4 * (g - 1)$
SCRAPS (LPC55S69) (@ 150MHz)	N/A	2109.1
SCRAPS (ATmega1284P) (@ 20MHz)	N/A	40147.4
DIAT (@ 168MHz)	N/A	835
SANA (@ 48MHz)	921.5	3125.8

Table 4: Run-time Overhead Comparison (g: number of neighbors of a device)

6.4 Run-time Overhead

Since $\mathcal{V}$rf is not a resource-constrained device, we focus on the overheads incurred on $\mathcal{P}$rv. Table 4 provides an overview of the run-time overhead for TRAIN and a comparison with prominent prior $\mathcal{NA}$ techniques: SEDA [7], SCRAPS [47], DIAT [1], and SANA [4].

Generating the attestation report (Att_{report}) is quite fast for both $\text{TRAIN}_{\text{CASU}}$ and $\text{TRAIN}_{\text{RATA}}$ $\mathcal{P}$rv types, since the overhead is dominated by the computation of an HMAC over a minimal fixed-length region. In comparison, initiators in SEDA have to sign the entire aggregated report, resulting in a significantly longer timing overhead compared to TRAIN. The report generation time of other $\mathcal{P}$rv-s is also higher than TRAIN as they must attest the whole program memory and verify neighbors' reports. Moreover, report generation

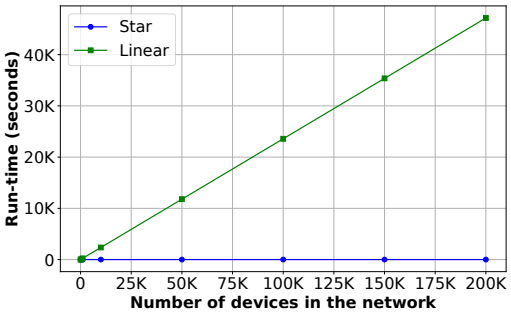


Figure 7: TRAIN **Simulation for Line/Star Topologies**

time in SEDA grows (almost) linearly, relying on the number of neighbors, denoted by g.

We also examine run-time overhead of SCRAPS, DIAT, and SANA. These schemes perform relatively complex tasks as part of attestation and thus incur high run-time overhead despite being implemented on more powerful devices.

In summary, compared to DIAT, SCRAPS, and SANA, TRAIN is lightweight in terms of run-time overhead.

6.5 Energy Consumption

Dynamic power consumption measurements from Xilinx Vivado show that $\text{TRAIN}_{\text{CASU}}$ and $\text{TRAIN}_{\text{RATA}}$ consume $115mW$, of which $111mW$ is consumed by either CASU or RATA. This represents a 2% increase in total on-chip power. Total time spent by TRAIN (request verification and report generation) is 42.5ms for $\text{TRAIN}_{\text{CASU}}$ and 42.7ms for $\text{TRAIN}_{\text{RATA}}$. Therefore, energy consumption per attestation instance is $\approx$ 0.00221mWh for $\text{TRAIN}_{\text{CASU}}$ and $\text{TRAIN}_{\text{RATA}}$, which is negligible.

6.6 Scalability Eval via Network Simulation

We conduct network simulations using the OMNeT++ [44]. Since TRAIN_{A} and TRAIN_{B} protocols are similar, only the former is simulated. Simulations are performed at the application layer. Cryptographic operations are simulated using delays that correspond to their actual execution times on $\text{TRAIN}_{\text{CASU}}$ and $\text{TRAIN}_{\text{RATA}}$ $\mathcal{P}$rv-s. We exclude $\mathcal{V}$rf's verification time from the simulations and set the communication rate between $\mathcal{P}$rv-s to 250Kbps. This rate matches the standard data rate for ZigBee – a common communication protocol for IoT devices. Simulations are conducted with various spanning tree topologies: line, star, and several types of trees, with degrees ranging from 2 (binary) to 12. We also vary the number of devices from 10 to $1,000,000$. Simulation results for $\text{TRAIN}_{\text{CASU}}$ and $\text{TRAIN}_{\text{RATA}}$ are almost identical, thus, only $\text{TRAIN}_{\text{CASU}}$ results are shown in Figures 7 and 8.

As evident from Figure 7, the run-time of TRAIN is constant with the star topology and grows linearly with the linear topology. This is because, in the former, $\mathcal{P}$rv can start attestation almost immediately (as there is no forwarding to descendants), while each $\mathcal{P}$rv waits until the farthest-away $\mathcal{P}$rv is ready to perform attestation in the latter. The actual run-time for the star topology is 343ms. For a network with a tree topology, TRAIN run-time overhead is logarithmic in the number of $\mathcal{P}$rv-s since the tree height governs it. Simulation results show that TRAIN is efficient in both small and large networks with various topologies.

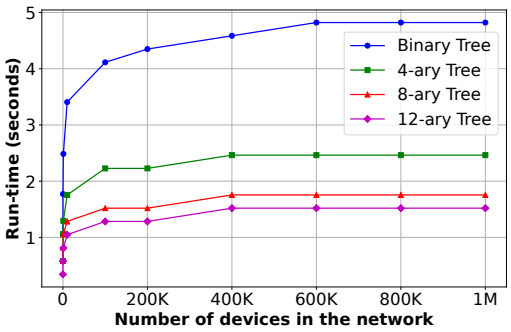


Figure 8: TRAIN **Simulation for Various Tree Topologies**

7 Discussion

7.1 TRAIN Compatibility

The rationale behind our choice of $\mathcal{P}$rv $\mathcal{R}$A platforms (i.e., CASU and RATA) is due to their minimal $\mathcal{R}$A overhead (HMAC over minimal fixed size input), $\text{TOCTOU}_{\mathcal{R}\text{A}}$ mitigation, and extensibility, which facilitates the construction of TimerTCB and NetTCB with a few hardware modifications. However, TRAIN is also compatible with other $\mathcal{R}$A platforms to minimize the $\text{TOCTOU}_{\mathcal{N}\text{A}}$ window,while losing the benefits of CASU and RATA. Some examples of compatible devices are:

- Devices with custom hardware *RoT*, e.g., Sancus [38] or TrustVisor [35].
- Devices with off-the-shelf TEE, such as TrustZone-A or TrustZone-M [6].
- Devices with hybrid (HW/SW) *RoT*, such as SMART [17], VRASED [40], TyTAN [8], TrustLite [27].
- Devices without any hardware *RoT*. In this case, the device OS must be trusted.

7.2 RATA vs. CASU

Given that RATA operates as a passive *RoT* and CASU functions as an active *RoT*, it is natural to question the necessity of RATA and why CASU is not utilized exclusively. The justification for employing RATA over CASU stems from three primary reasons: (1) Memory Constraints: In CASU, only half of the program memory (PMEM) can store authorized software, while the other half is reserved for the secure update process. This significant (50%) PMEM reservation can be prohibitive for low-end devices with limited memory. (2) Access to Non-Volatile Memory: CASU prevents normal software from modifying PMEM. However, some software may require access to non-volatile memory (e.g., flash) for benign purposes, such as storing text or image files. RATA allows such access and is preferred in these circumstances. (3) Hardware Overheads: RATA has slightly lower hardware overheads compared to CASU.

7.3 $\text{TOCTOU}_{\mathcal{N}\text{A}}$ Minimization in TRAIN_{B}

Even though TRAIN_{B} cannot achieve perfect synchronization without RTCs, it significantly reduces the $\text{TOCTOU}_{\mathcal{N}\text{A}}$ window compared to naïve approaches where the window scales with both spanning tree depth and network congestion. Recall that Section 6.1 illustrates the reduction in $\text{TOCTOU}_{\mathcal{N}\text{A}}$ window through a concrete example. By computing attestation timing based on the network topology, TRAIN_{B} effectively eliminates the spanning tree traversal

$\mathcal{RA}$ Method	Type	Passive/Active	TOCTOU$_{\mathcal{RA}}$	Network TCB	Attestation Time	Platform
RealSWATT [53]	SW	Passive	✓	✘	$O(n)$	ESP32
SANCUS [38]	HW	Passive	✘	✘	$O(n)$	openMSP430
IDA [5]	Hybrid	Passive	✓	✘	$O(n)$	openMSP430
RATA [15]	Hybrid	Passive	✓	✘	$O(1)$	openMSP430
GAROTA [3]	Hybrid	Active	✘	✓	$O(n)$	openMSP430
CASU [14]	Hybrid	Active	✓	✘	$O(1)$	openMSP430
TRAIN	**Hybrid**	**Passive/Active**	✔	✔	$O(1)$	**openMSP430**

Table 5: Comparison with Other Individual Attestation Schemes (n: attested area size)

$\mathcal{NA}$ Method	TOCTOU$_{\mathcal{NA}}$	Simulator	Underlying Platform	Remark
SEDA [7]	✘	OMNeT++	SMART/TrustLite	Provides pioneering scheme using secure hop-by-hop aggregation
SANA [4]	✘	OMNeT++	TyTan	Extends SEDA with aggregate signatures and sub-networks
LISA [10]	✘	CORE	Unspecified	Introduces neighbor-based communication and quality metric
SeED [22]	✘	OMNeT++	SMART/TrustLite	Extends SEDA with self-initiated $\mathcal{RA}$
TRAIN	✔	**OMNeT++**	**CASU/RATA**	Minimizes TOCTOU window, $\mathcal{RA}$ overhead, and isolates $\mathcal{RA}$ functionality

Table 6: Comparison with Other Network Attestation Schemes

component of the TOCTOU$_{\mathcal{NA}}$ window, leaving only network delay as a factor influencing the imperfection of the synchronization.

8 Related Work

Individual Device Attestation ($\mathcal{RA}$) is an extensively studied topic and numerous schemes have been proposed in the literature. These techniques generally fall into three categories: software-based, hardware-based, and hybrid. Software-based $\mathcal{RA}$ [32, 50, 51, 53] is only viable for legacy devices with no secure hardware features. It uses request-to-response time (between $\mathcal{V}$rf and $\mathcal{P}$rv) to establish confidence in the integrity of the attestation report. Nonetheless, network limitations (e.g. intermittent connection, network congestion) on $\mathcal{P}$rv introduce noise to the request-to-response time, making software-based $\mathcal{RA}$ impractical.

In contrast, hardware-based $\mathcal{RA}$ techniques [11, 12, 33, 35, 38, 52] either (1) embed $\mathcal{P}$rv attestation functionality entirely within dedicated hardware, or (2) require substantial changes to the underlying hardware to support isolated execution of trusted software, e.g., SGX [23] or TrustZone [6]. However, such hardware features are often too complex and costly for low-end devices constrained by size, energy, and cost.

Given the limitations of both hardware- and software-based approaches in low-end embedded platforms, software/hardware co-design (hybrid) [5, 8, 17, 27, 40, 41] has recently emerged as a promising solution. It aims to provide equivalent security guarantees to hardware-based $\mathcal{RA}$ while minimizing modifications to the underlying hardware. Table 5 compares various software, hardware, and hybrid $\mathcal{RA}$ methods.

Network Attestation ($\mathcal{NA}$) enables scalable attestation for large groups of interconnected devices. Few prior work [1, 4, 7, 10, 21, 22, 28–30, 39, 47] refers to this process as Swarm Attestation; we employ the term Network Attestation to denote the same concept. Table 6 shows a comparison with other $\mathcal{NA}$ schemes.

The first scheme, SEDA [7], employs secure hop-by-hop aggregation of $\mathcal{RA}$ reports. Initially, $\mathcal{V}$rf broadcasts an attestation request to $\mathcal{P}$rv-s. Each $\mathcal{P}$rv attests its children nodes and forwards aggregated $\mathcal{RA}$ reports to its parent. Finally, $\mathcal{V}$rf verifies only the last $\mathcal{RA}$ reports to assess the status of all $\mathcal{P}$rv-s. SANA [4] extends SEDA with a novel aggregate signature scheme, ensuring low verification overhead with minimal trust anchor. It partitions $\mathcal{P}$rv-s into subnetworks and aggregates $\mathcal{RA}$ results across the entire network, facilitating public verification by multiple $\mathcal{V}$rf-s. LISA [10] introduces neighbor-based communication to propagate $\mathcal{RA}$ reports. $\mathcal{P}$rv-s verify $\mathcal{RA}$ reports of other $\mathcal{P}$rv-s before forwarding them to prevent replay attacks, and a quality metric for $\mathcal{NA}$ techniques captures the information from each $\mathcal{P}$rv. SeED [22] enhances the efficiency of SEDA and resilience against DoS attacks by enabling $\mathcal{P}$rv-s to self-initiate $\mathcal{RA}$.

9 Conclusion

This paper constructs a TOCTOU-resilient $\mathcal{NA}$ protocol (TRAIN) for networks of low-end IoT devices. It facilitates simultaneous attestation across the network while minimizing runtime/energy overhead by computing HMAC over minimal fixed-size input. Two variants of the protocol, based on the availability of real-time clocks, are present. An open-source prototype implemented on TI MSP430 demonstrates the practicality of TRAIN on commodity hardware.

Acknowledgments

We thank SenSys 2025 reviewers for constructive feedback. This work was supported in part by funding from NSF Award SATC-1956393, NSA Award H98230-22-1-0308, and a 2024 Guggenheim Fellowship.

References

[1] Tigist Abera, Raad Bahmani, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Matthias Schunter. 2019. DIAT: Data Integrity Attestation for Resilient Collaboration of Autonomous Systems.. In *NDSS*.

[2] Abdulla Aldoseri, Tom Chothia, Jose Moreira, and David Oswald. 2023. Symbolic modelling of remote attestation protocols for device and app integrity on Android. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*.

[3] Esmerald Aliaj, Ivan De Oliveira Nunes, and Gene Tsudik. 2022. {GAROTA}: generalized active {Root-Of-Trust} architecture (for tiny embedded devices). In *31st USENIX Security Symposium (USENIX Security 22)*.

[4] Moreno Ambrosin, Mauro Conti, Ahmad Ibrahim, Gregory Neven, Ahmad-Reza Sadeghi, and Matthias Schunter. 2016. SANA: Secure and scalable aggregate network attestation. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*.

[5] Fatemeh Arkannezhad, Justin Feng, and Nader Sehatbakhsh. 2024. IDA: Hybrid Attestation with Support for Interrupts and TOCTOU. In *31th Annual Network and Distributed System Security Symposium, NDSS 2024*.

[6] Arm Ltd. 2018. Arm TrustZone. https://www.arm.com/products/security-on-arm/trustzone/.

[7] Nadarajah Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. 2015. Seda: Scalable embedded device attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*.

[8] Ferdinand Brasser, Brahim El Mahjoub, Ahmad-Reza Sadeghi, Christian Wachsmann, and Patrick Koeberl. 2015. TyTAN: tiny trust anchor for tiny devices. In *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*.

[9] Ferdinand Brasser, Kasper Bonne Rasmussen, Ahmad-Reza Sadeghi, and Gene Tsudik. 2016. Remote attestation for low-end embedded devices: the prover's perspective. In *Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, Austin, TX, USA, June 5-9, 2016*.

[10] Xavier Carpent, Karim ElDefrawy, Norrathep Rattanavipanon, and Gene Tsudik. 2017. Lightweight swarm attestation: A tale of two lisa-s. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*.

[11] Guoxing Chen and Yinqian Zhang. 2022. {MAGE}: Mutual Attestation for a Group of Enclaves without Trusted Third Parties. In *31st USENIX Security Symposium (USENIX Security 22)*.

[12] Guoxing Chen, Yinqian Zhang, and Ten-Hwang Lai. 2019. Opera: Open remote attestation for intel's secure enclaves. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*.

[13] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. 2002. Nusmv 2: An opensource tool for symbolic model checking. In *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings 14*. Springer, 359–364.

[14] Ivan De Oliveira Nunes, Sashidhar Jakkamsetti, Youngil Kim, and Gene Tsudik. 2022. Casu: Compromise avoidance via secure update for low-end embedded systems. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*.

[15] Ivan De Oliveira Nunes, Sashidhar Jakkamsetti, Norrathep Rattanavipanon, and Gene Tsudik. 2021. On the TOCTOU problem in remote attestation. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*.

[16] D. Dolev and A. Yao. 1983. On the security of public key protocols. *IEEE Transactions on Information Theory* (1983).

[17] Karim Eldefrawy, Gene Tsudik, Aurélien Francillon, and Daniele Perito. 2012. SMART: Secure and Minimal Architecture for (Establishing Dynamic) Root of Trust. In *NDSS*.

[18] Dmitry Evtyushkin, Jesse Elwell, Meltem Ozsoy, Dmitry Ponomarev, Nael Abu Ghazaleh, and Ryan Riley. 2014. Iso-x: A flexible architecture for hardware-managed isolated execution. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*.

[19] Erhu Feng, Xu Lu, Dong Du, Bicheng Yang, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. 2021. Scalable memory protection in the {PENGLAI} enclave. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*.

[20] Hamid Reza Ghaeini, Matthew Chan, Raad Bahmani, Ferdinand Brasser, Luis Garcia, Jianying Zhou, Ahmad-Reza Sadeghi, Nils Ole Tippenhauer, and Saman Zonouz. 2019. {PAtt}: Physics-based Attestation of Control Systems. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*.

[21] Ahmad Ibrahim, Ahmad-Reza Sadeghi, Gene Tsudik, and Shaza Zeitouni. 2016. Darpa: Device attestation resilient to physical attacks. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*.

[22] Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Shaza Zeitouni. 2017. SeED: secure non-interactive attestation for embedded devices. In *Proceedings of the 10th ACM conference on security and privacy in wireless and mobile networks*.

[23] Intel. [n.d.]. Software Guard Extensions (Intel SGX). https://software.intel.com/en-us/sgx/.

[24] Ahmed Irfan, Alessandro Cimatti, Alberto Griggio, Marco Roveri, and Roberto Sebastiani. 2016. Verilog2SMV: A tool for word-level verification. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1156–1159.

[25] J. Protzenko J.-K. Zinzindohoué, K. Bhargavan and B. Beurdouche. 2017. "Hacl*: A verified modern cryptographic library, In "Hacl*: A verified modern cryptographic library. *CCS*.

[26] Sashidhar Jakkamsetti, Youngil Kim, and Gene Tsudik. 2023. Caveat (IoT) Emptor: Towards Transparency of IoT Device Presence. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*.

[27] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. 2014. TrustLite: A security architecture for tiny embedded devices. In *EuroSys*.

[28] Florian Kohnhäuser, Niklas Büscher, Sebastian Gabmeyer, and Stefan Katzenbeisser. 2017. Scapi: a scalable attestation protocol to detect software and physical attacks. In *Proceedings of the 10th ACM conference on security and privacy in wireless and mobile networks*.

[29] Florian Kohnhäuser, Niklas Büscher, and Stefan Katzenbeisser. 2018. Salad: Secure and lightweight attestation of highly dynamic and disruptive networks. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*.

[30] Boyu Kuang, Anmin Fu, Shui Yu, Guomin Yang, Mang Su, and Yuqing Zhang. 2019. ESDRA: An efficient and secure distributed remote attestation scheme for IoT swarms. *IEEE Internet of Things Journal* (2019).

[31] Leslie Lamport. 1981. Password Authentication with Insecure Communication. In *Communications of the ACM 24.11*.

[32] Yanlin Li, Jonathan M McCune, and Adrian Perrig. 2011. VIPER: Verifying the integrity of peripherals' firmware. In *Proceedings of the 18th ACM conference on Computer and communications security*.

[33] Zhen Ling, Huaiyu Yan, Xinhui Shao, Junzhou Luo, Yiling Xu, Bryan Pearson, and Xinwen Fu. 2021. Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT Nodes. *Journal of Systems Architecture* (2021).

[34] Marwa Mamdouh, Mohamed AI Elrukhsi, and Ahmed Khattab. 2018. Securing the internet of things and wireless sensor networks via machine learning: A survey. In *2018 International Conference on Computer and Applications (ICCA)*.

[35] Jonathan M McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. 2010. TrustVisor: Efficient TCB reduction and attestation. In *2010 IEEE Symposium on Security and Privacy*.

[36] Kenneth L McMillan and Kenneth L McMillan. 1993. The SMV system. *Symbolic Model Checking* (1993), 61–85.

[37] Rajani Muraleedharan and Lisa Ann Osadciw. 2006. Jamming attack detection and countermeasures in wireless sensor network using ant system. In *Wireless Sensing and Processing*.

[38] Job Noorman, Pieter Agten, Wilfried Daniels, Raoul Strackx, Anthony Van Herrewege, Christophe Huygens, Bart Preneel, Ingrid Verbauwhede, and Frank Piessens. 2013. Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In *22nd USENIX Security Symposium (USENIX Security 13)*.

[39] Ivan De Oliveira Nunes, Ghada Dessouky, Ahmad Ibrahim, Norrathep Rattanavipanon, Ahmad-Reza Sadeghi, and Gene Tsudik. 2019. Towards systematic design of collective remote attestation protocols. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*.

[40] Ivan De Oliveira Nunes, Karim Eldefrawy, Norrathep Rattanavipanon, Michael Steiner, and Gene Tsudik. 2019. {VRASED}: A verified {Hardware/Software}{Co-Design} for remote attestation. In *28th USENIX Security Symposium (USENIX Security 19)*.

[41] Ivan De Oliveira Nunes, Karim Eldefrawy, Norrathep Rattanavipanon, and Gene Tsudik. 2020. {APEX}: A verified architecture for proofs of execution on remote devices under full software compromise. In *29th USENIX Security Symposium (USENIX Security 20)*.

[42] Johannes Obermaier and Vincent Immler. 2018. The past, present, and future of physical security enclosures: from battery-backed monitoring to puf-based inherent security and beyond. *Journal of hardware and systems security* (2018).

[43] Olivier Girard. 2009. OpenMSP430. https://opencores.org/projects/openmsp430/.

[44] OpenSim Ltd. [n.d.]. OMNeT++ Discrete Event Simulator. https://omnetpp.org/.

[45] P. Frolikov, Y. Kim, R. Prapty, G. Tsudik. [n.d.]. TRAIN source code. https://github.com/sprout-uci/TRAIN.

[46] Adrian Perrig, JD Tygar, Adrian Perrig, and JD Tygar. 2003. TESLA broadcast authentication. *Secure Broadcast Communication: In Wired and Wireless Networks* (2003).

[47] Lukas Petzi, Ala Eddine Ben Yahya, Alexandra Dmitrienko, Gene Tsudik, Thomas Prantl, and Samuel Kounev. 2022. {SCRAPS}: Scalable Collective Remote Attestation for {Pub-Sub}{IoT} Networks with Untrusted Proxy Verifier. In *31st USENIX Security Symposium (USENIX Security 22)*.

[48] Srivaths Ravi, Anand Raghunathan, and Srimat Chakradhar. 2004. Tamper resistance mechanisms for secure embedded systems. In *VLSI Design*.

[49] Nader Sehatbakhsh, Alireza Nazari, Haider Khan, Alenka Zajic, and Milos Prvulovic. 2019. Emma: Hardware/software attestation framework for embedded systems using electromagnetic signals. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*.

[50] Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. 2006. SCUBA: Secure code update by attestation in sensor networks. In *Proceedings of the 5th ACM workshop on Wireless security*.

[51] Arvind Seshadri, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. 2004. SWATT: Software-based attestation for embedded devices. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*.

[52] Raoul Strackx, Frank Piessens, and Bart Preneel. 2010. Efficient isolation of trusted subsystems in embedded systems. In *Security and Privacy in Communication Networks: 6th Iternational ICST Conference, SecureComm 2010, Singapore, September 7-9, 2010. Proceedings 6*.

[53] Sebastian Surminski, Christian Niesler, Ferdinand Brasser, Lucas Davi, and Ahmad-Reza Sadeghi. 2021. Realswatt: Remote software-based attestation for embedded devices under realtime constraints. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*.

[54] Microchip Technology. [n.d.]. MCP7940M: Low-Cost I2C Real-Time Clock/Calendar with SRAM. https://ww1.microchip.com/downloads/en/DeviceDoc/MCP7940M-Low-Cost%%20I2C-RTCC-with-SRAM-20002292C.pdf

[55] Texas Instruments. 2016. MSP430 GCC User's Guide. https://www.ti.com/tool/MSP430-GCC-OPENSOURCE/.

[56] Jinwen Wang, Yujie Wang, Ao Li, Yang Xiao, Ruide Zhang, Wenjing Lou, Y Thomas Hou, and Ning Zhang. 2023. {ARI}: Attestation of Real-time Mission Execution Integrity. In *32nd USENIX Security Symposium (USENIX Security 23)*.

[57] Wu Zhijun, Li Wenjing, Liu Liang, and Yue Meng. 2020. Low-rate DoS attacks, detection, defense, and challenges: A survey. *IEEE access* (2020).