TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214
DOI: 10.26599/TST.2025.9010052

Research Article

SeCA: Software Behavior-Enforced Collective Attestation for Highly Dynamic Device Networks

Ziyu Wang, Cong Sun, Qingsong Yao*, and Jingwei Li

Abstract: The Internet of Things and collaborative embedded device networks, such as vehicular networks, robots, and unmanned aerial vehicle swarms, are increasingly used in the real world. Such devices confront severe physical attacks and software compromises beyond the network threats. Collective attestation is a popular approach for verifying the security and integrity of remote devices. However, existing collective attestation approaches mitigating physical attacks focus only on attesting the devices' static code integrity. They cannot resist the runtime compromises that hijack the control flow of the task programs. We propose SeCA, an efficient collective attestation framework that uses the trusted execution environment based self-attestation to withstand the runtime control-flow hijacking on the task programs of the networked devices. SeCA's collective attestation protocol achieves efficient attestation evidence aggregation and uses a heartbeat mechanism to mitigate the physical attacks in highly dynamic networks. We implement SeCA on the Raspberry Pi 3b board. Compared to the state-of-the-art control-flow attestation approach, SeCA's program instrumentation reduces the code size and runtime attestation overhead. Additionally, SeCA's attestation report aggregation scheme reduces the per-device computational, storage, and communication overhead. Overall, SeCA is the first collective attestation that can resist both physical attacks and runtime control-flow hijacking, with competitive attestation performance compared to other collective attestation methods.

Key words: collective attestation; remote attestation; embedded system; physical attack; control-flow integrity; security protocol

1 Introduction

Embedded systems are ubiquitous and crucial components of numerous devices in the real world. They connect, measure, and control various actuators and sensors, monitor and automate diverse processes to enable a wide range of functionalities, and perform various safety-critical tasks in smart cities, industrial control systems, medical devices, automobiles, and unmanned aerial vehicles[1−3]. Despite their vital role, embedded devices are prone to various security vulnerabilities[4−8]. Remote attestation is a crucial technology that ensures the healthiness of remote devices. However, in many application scenarios, embedded devices form self-organizing networks, collaboratively executing tasks as a group. In such scenarios, the collective attestation model is necessary.

SEDA[9], a pioneer in collective remote attestation, leverages a verifier-rooted prover generation tree for proofs; each device, as a prover, verifies its child nodes

• Ziyu Wang, Cong Sun, and Qingsong Yao are with School of Cyber Engineering, Xidian University, Xi'an 710071, China. E-mail: wangziyu@stu.xidian.edu.cn; suncong@xidian.edu.cn; qsyao@xidian.edu.cn.
• Jingwei Li is with the State Key Laboratory of Integrated Services Networks, School of Telecommunications Engineering, Xidian University, Xi'an 710071, China. E-mail: jingweili@xidian.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2024-10-31; revised: 2025-02-10; accepted: 2025-04-02

and reports evidence to its parent node in the tree. Nevertheless, tree-based proofs are unsuitable for highly dynamic networks, such as Unmanned Aerial Vehicle (UAV) networks. From this perspective, recent approaches for dynamic[10, 11] and even disruptive networks[12−17] are effective. These approaches' adversarial application scenarios pose a great need for physical attack resilience. PASTA[10] detects physical attacks by recording and sharing communication timestamps, but confronts significant storage and communication overhead. LICAPA[17] improves the precision of physical attack detection by generating a tamper-proof signed timestamp but requires substantial computational overhead due to the extensive signature operations.

On the other hand, most collective attestation schemes only ensure the security of the software behaviors by measuring the integrity of static code. These approaches fail to capture the runtime behavior of the task programs, thereby being unable to guarantee software runtime correctness, especially control-flow integrity. Although C-FLAT[18] and OAT[19] implement control-flow attestation for individual devices and safeguard the task software's runtime integrity, these approaches are inapplicable to the collective attestation. Therefore, new collective attestations are required to report the runtime integrity of the provers' software to the verifier or network owner. DIAT[20] achieves dynamic and modular data integrity attestation against runtime software compromises for highly dynamic devices, e.g., drones. The execution paths obtained by DIAT can support control-flow attestation and have been further extended to the collective attestation scenarios. Nevertheless, DIAT's collective attestation implementation raises significant storage overhead on devices, and the physical attacks are out of DIAT's scope.

We propose SeCA, a Software behavior-enforced Collective Attestation for highly dynamic and disruptive device networks. SeCA is designed to withstand communication and physical adversaries, as well as the more potent software adversaries, including those capable of hijacking the runtime control flow of the prover's task program. SeCA's attestation procedure consists of two components: local control-flow self-attestation and collective attestation. The self-attestation, protected by Trusted Execution Environment (TEE), employs an efficient control-flow attestation to prevent software from being compromised by runtime control-flow hijacking. The self-attestation periodically measures the runtime control-flow integrity of the task programs and updates the local attestation report with the fresh control-flow attestation result. The self-attestation results are aggregated with the attestation evidence from other devices in the collective attestation procedure. Each device's TEE holds a hardware-protected, online-updated heartbeat count for detecting and mitigating physical attacks. Moreover, each device protects its public key with TEE as a disruption salt to alter the Message Authentication Code (MAC) key of the self-attestation failure device, preventing such software-compromised devices from impacting other healthy devices. Our main contributions are as follows:

- We propose SeCA, the first collective attestation framework that is resilient against both control-flow hijacking and physical attacks. SeCA implements a novel and efficient TEE-based control-flow self-attestation to enforce the correctness of the runtime software behavior for the networked devices. To make the self-attestation efficient, we propose a new program instrumentation approach. Applying this instrumentation to the provers' task programs allows the control-flow paths to be profiled with fewer instrumentation points, thereby reducing the computational overhead of control-flow attestation. The overall per-device control-flow attestation cost can be reduced by 0.27%−7.34% compared with OAT[19].
- SeCA efficiently aggregates attestation evidence among devices and enforces network-wide control-flow integrity for device software. Our innovative attestation evidence aggregation scheme compresses the device's control-flow integrity status into ternary space. Specifically compared with LICAPA[17], our aggregation scheme reduces the size of the attestation report by 77.8%, thereby reducing the overall collective attestation cost by 10.1%.

2 Related Work

Remote attestation is a critical system security mechanism that employs a series of security protocols and monitoring techniques to verify the remote platform or device's authenticity and the integrity of its software or data. This mechanism provides fundamental protection in the realm of cloud computing, Internet of Things (IoT), and distributed systems, as it ensures trust relationships between remote principals and mitigates threats such as

malware, unauthorized access, or data tampering. This section first outlines the fundamental concepts and architectures of remote attestation. Then, we present the state-of-the-art collective attestation methodologies to delineate the scope of SeCA. Lastly, we briefly summarize the self-attestation techniques, which are crucial for implementing the control-flow attestation within SeCA.

2.1 Individual device attestation

Within the remote attestation framework, a trusted verification entity receives a recent attestation report detailing the current software configuration and status of a potentially compromised device (prover). It proves its healthiness at a remote location. To ensure the credibility and integrity of the proof process, the prover relies on the trust anchor mechanism to prevent the attestation code from being tampered with. Traditionally, trust anchors have been implemented in hardware, e.g., using a Trusted Platform Module (TPM) as its core component[21]. However, due to the complexity and high cost of TPM and similar hardware components[22−24], their deployment in resource-constrained low-end embedded devices is challenging. In contrast, software-based remote attestations[25, 26] require no hardware-level security features and are more suitable for attesting cost-sensitive embedded devices. However, software-based solutions have stronger security assumptions, and their feasibility and robustness in realistic deployment environments are questionable[27]. Hybrid attestation mechanisms[28, 29] have emerged to balance security and overhead, flexibly and efficiently providing vigorous security enforcement while requiring only minimal hardware security support. It is worth noting that all of the above remote attestations focus on verifying an individual prover device. In network scenarios with multiple provers, extending the protocol and attestation evidence measurements is essential.

2.2 Collective attestation

Collective attestation aims to provide an efficient network-wide attestation service and ensure network-wide runtime integrity. Collective attestation techniques mitigate the individual-device attestation's limitation on scalability and differ from the access control approaches that address the private data interaction among devices, e.g.,[30]. The security protocols widely adopt the hybrid-attestation device architecture. They are categorized based on network characteristics into stable-topological and dynamic collective attestation. In stable network scenarios, the collective attestation approaches[9, 31−35] assume robust network connections, and the network topology remains static throughout the entire attestation period. These schemes leverage the spanning tree overlay strategy to restructure the network topology into a balanced binary tree, establishing explicit node hierarchy among devices. During the attestation procedure, devices engage in synchronous interactions, i.e., each device validates the state of its child nodes and submits attestation reports to its parent node, which then aggregates the reports upwards. Xiong et al.[36] designed an aggregator on top of a static network and reduced the computational cost by lifting the complex computations of the aggregator to the cloud. To overcome the restriction of stable networks, dynamic collective attestations[11−17, 37] integrate remote attestation with consensus mechanisms[38] or more robust evidence aggregation techniques to adapt to highly dynamic and disruptive network scenarios. In the dynamic attestation procedure, devices engage in continuous communication for attestation knowledge sharing. With consensus mechanisms, these works ensure that all devices ultimately agree on a common set of knowledge, thereby effectively attesting to the networked devices.

2.3 Self-attestation

Self-attestation is a security mechanism that allows a device to prove locally that its software and hardware states have not been tampered with. In this attestation procedure, the device system periodically and autonomously attests to its integrity and reports any runtime deviations from the expected states without relying on any external verifier. This mechanism is particularly useful in scenarios that require continuous monitoring and lightweight communication overheads, e.g., IoT networks and edge computing. Researchers recently proposed various self-attestation schemes to address different security threats and application requirements, e.g., SeED[39], PADS[10], and ERASMUS[40], achieving lightweight communication through self-measurement and enhancing the robustness of the attestation procedure.

3 Attacker Model

We specify the adversary models for SeCA to precisely

define its resilience against potential attackers and the scope of our security objectives. In general, the objective of the adversary is to alter one or more prover devices' hardware, software behaviors, and configurations while evading the network owner's detection.

3.1 Communication adversary

The communication adversary Adv_{com} has complete control over the communication channels between devices. Adv_{com} can conduct the following attacks:

(1) Man-In-The-Middle (MITM) attack[41]: Intercepting and tampering with the data transmission between two communicating principals (provers or the network owner), thereby undermining the communication confidentiality or integrity between the principals.

(2) Impersonation attack[42]: Propagating crafted attestation evidence by forging attestation reports to impersonate a healthy device.

(3) Replay attack[43]: Replaying the obsolete healthy evidence of the target device to confuse other principals from the genuine status of this device.

(4) Denial of Service (DoS) attack[39]: Launching many attestation requests or crafted packets to exhaust the device's service resource, causing the device to be unable to provide regular service.

3.2 Software adversary

The software adversary Adv_{sw} can access memory outside the trust anchor and read unprotected memory locations, including user-space code. Adv_{sw} can exploit the user-space memory vulnerabilities of the device system to hijack the control flow at runtime, introducing the execution path along invalid control-flow edges that do not belong to the original program's Control-Flow Graph (CFG). This exploit, known as control-flow hijacking, can be achieved through techniques such as Return-Oriented Programming (ROP) or Return2Libc for code reuse attacks. Through these attacks, Adv_{sw} can gain complete control over code execution outside the trust anchor. Besides, Adv_{sw} can also hijack the control flow of the user-space task program to disrupt its normal execution and discard the aggregated attestation reports instead of propagating them, achieving a black-hole attack[44] at the software level.

3.3 Hardware adversary

The hardware adversary Adv_{hw} is capable of conducting physical attacks[45]. On a device that has been physically compromised, Adv_{hw} can circumvent hardware security measures, modify code, and access secrets stored within the TEE. However, physical tampering requires taking the device offline for a minimum time interval, denoted as T_{att}, during which the device cannot communicate with other devices.

4 Design of SeCA

This section elaborates on the design of the SeCA framework. In a typical scenario of SeCA, there are two categories of mobile entities: the provers and a network owner, i.e., owner for short. The provers are homogeneous devices equipped with a trusted execution environment. Each device has a limited communication range. The owner is trusted. The owner can reach at least one prover at any moment to launch the network-wide attestation procedure and obtain the aggregated attestation report to summarize the network security status.

The SeCA framework consists of two components: local control-flow self-attestation and collective attestation. On the one hand, the local control-flow self-attestation is designed on each prover device to continuously monitor the runtime status of the prover's task software, ensuring the control-flow integrity of its operations. This procedure can efficiently detect potential control-flow hijacking attacks. The TEE of each device can mitigate the impact of such control-flow compromises on other devices. On the other hand, the collective attestation procedure aggregates the locally generated control-flow attestation evidence to enforce the network-wide behavior trustworthiness towards the owner against the threats from the communication and hardware adversaries. Specifically, each prover periodically attests to the provers it communicates with and detects the physically compromised or control-flow hijacked provers. The attestation reports are propagated and aggregated throughout the attested provers to the network owner. The attested provers cooperatively prevent the compromised prover from obtaining sensitive data or tampering with more devices.

4.1 Control-flow self-attestation

General-purpose control-flow attestations, e.g., ReCFA[46], usually have significant communication overhead thus are inapplicable to the highly dynamic

and disruptive network scenario. We resort to the self-attestation approach for the control-flow attestation to reduce the communication and storage costs. As a result, we implement a local control-flow verifier, i.e., $\mathcal{V}^{\ell}$, in the TEE of each prover device to attest to the behavior of the vanilla prover software. The prover software has numerous control-flow paths. Under a self-attestation scheme, the prover device only stores the measurement of its own control-flow traces and requires no control-flow knowledge of other devices. Thus, SeCA's $\mathcal{V}^{\ell}$ significantly reduces storage costs. The task software starts execution in response to user command or environmental input. Such executions iterate, and $\mathcal{V}^{\ell}$ makes a control-flow integrity decision at the end of each iteration. Therefore, $\mathcal{V}^{\ell}$ regularly produces posterior control-flow evidence and updates the evidence into an aggregatable attestation report, which will be aggregated in the network-wide collective attestation (Section 4.2).

We adapt the design feature of the operation-scope Control-Flow Integrity (CFI) component of OAT[19] and make new improvements to design and implement the local control-flow verifier of SeCA. For the example in Fig. 1, the CFG is presented in Fig. 2. Figure 2a presents the CFG nodes instrumented by OAT. These nodes involve secure/normal-world switching and high computational cost. The operation-scope CFI of OAT instruments nodes N_2, N_3, N_6, N_5, and N_{11}. Specifically, the conditional-branch nodes (i.e., N_2, N_3, and N_6) are instrumented to transfer a

```
fp = func;                          //N1
while (cond1) {                     //N2
    if (cond2) {                    //N3
        (*fp)();                    //N4, N5
    } else {
        if (cond3) {                //N6
            BBL_A                   //N7
        } else {
            BBL_B                   //N8
        }
    }
    BBL_C                           //N9
}
BBL_D                               //N10
…
func(){
    BBL_E                           //N11
}
```

Fig. 1 Motivating example. BBL_A–BBL_E denote a basic block in CFG, $cond_1$–$cond_3$ are branching conditions, fp is a function pointer of the function func(), N_1–N_{11} are nodes of CFG.

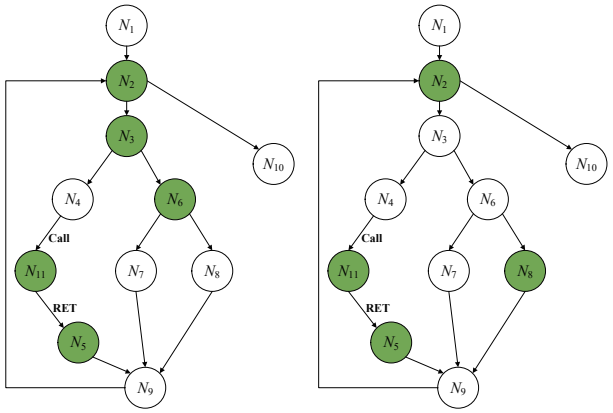


Fig. 2 Comparison of CFG node instrumentation. RET: Return.

binary flag to the secure world. The indirect call target N_{11} is recorded, and the backward-edge target N_5 is accumulated in a hash value delivered into the secure-world. SeCA's $\mathcal{V}^{\ell}$ follows OAT to make instrumentation for the indirect branch targets and backward-edge targets, i.e., N_{11} and N_5 in Fig. 2b. However, SeCA makes more efforts to reduce the number of other CFG nodes to be instrumented. Specifically, SeCA uses the following policies to instrument the CFG nodes other than the indirect branch targets and backward-edge targets:

Rule I. The program exit node (without any successor) should not be instrumented.

Rule II. The loop-condition node should be instrumented.

Rule III. For a node with n predecessors, if the node is not a loop-condition node and some of its predecessors are undecided yet, we find the undecided predecessor with the largest out-degree. We skip this predecessor and instrument the other $n-1$ predecessors.

Rule IV. We classify the predecessors of the loop-condition node into two categories: in-loop predecessors and out-loop predecessors. For each category of predecessors, we apply Rule III on each category.

We develop a backward breadth-first traversal over the CFG to decide the instrumented nodes other than the indirect branch and backward-edge targets. In Fig. 2b, the procedure starts from N_{10}, which is decided uninstrumented with Rule I. N_2 is instrumented according to Rule II. The in-loop predecessor set of N_2 is $\{N_9\}$, and the out-loop predecessor set of N_2 is $\{N_1\}$. According to Rule IV,

both N_1 and N_9 are not instrumented since they are the exclusive in/out-loop predecessor of N_2. It is worth noting that even though N_2 has the exclusive out-loop predecessor N_1 in our example, multiple out-loop predecessors will be common in real programs. For N_9's predecessors, N_5 is decided as being instrumented, and we randomly decide to instrument N_8 and leave N_7 uninstrumented since N_7 and N_8 have the same out-degree (Rule III). Then, nodes N_6, N_3, and N_4 are decided uninstrumented. As a result, SeCA only instruments nodes N_2, N_5, N_8, and N_{11} to measure the path features for the control-flow attestation.

The instrumented code takes charge of transferring the control-flow evidence to the TEE. Therefore the CFG nodes with instrumented code generally have higher runtime overheads than the uninstrumented nodes. Reducing the number of instrumented nodes reduces the number of context switchings between the secure and normal world. Table 1 elaborates on the TEE-switching CFG nodes with regard to typical runtime sub-traces of the motivating example program. In all the loop-related sub-traces (T_1, T_2, and T_3), SeCA has fewer TEE-switching CFG nodes than OAT. Considering the program execution produces a trace in the form of $N_1 \rightarrow (T_1|T_2|T_3)^* \rightarrow N_2 \rightarrow N_{10}$, SeCA is more efficient than OAT in control-flow evidence collection.

The minimized TEE-switching CFG nodes have to cooperate with a new control-flow verifier design in the secure world, as presented in Fig. 3. The instrumented task program of the prover calls the trampolines in the trampoline library to deliver the control-flow evidence to the measurement engine in TEE. Like OAT, the task program delivers each indirect branch target address (e.g., N_{11} of Fig. 2b). The prover accumulates the return target address (e.g., N_5 in Fig. 2b) into a hash value and delivers the hash value into TEE at a specific point or the end of the execution. However, differently from OAT, SeCA does not instrument to deliver the binary flags at the conditional branches. Instead, SeCA delivers the address of loop-condition nodes (e.g., N_2 in Fig. 2b) and the instrumented predecessor nodes (e.g., N_8 in Fig. 2b). Therefore, the measurement engine receives a stream of the CFG-node addresses and an accumulated hash value from the normal world.

The measurement engine statically reconstructs the control-flow path from the evidence received from the prover. To realize this reconstruction without the conditional branch information, we use an offline phase to build the address correspondences between the adjacent instrumented nodes. A pair of addresses (addr_1, addr_2) is an address correspondence if addr_1 and addr_2 are the addresses of the instrumented CFG node, and by following the valid control flow paths from addr_1, we can reach node addr_2 without passing other instrumented nodes. For the example of Fig. 2b, the address correspondence set is {(start, N_2), (N_2, end), (N_2, N_{11}), (N_2, N_2), (N_2, N_8), (N_{11}, N_2), (N_8, N_2)}. It is worth noting that N_5 is not considered in the address correspondence since the instrumented code in N_5 conducts the accumulated hash. The correspondence between the indirect call and its backward edge is controlled by the call stack, which is also used by OAT[19]. The reconstruction procedure refers to the address correspondences and the received address stream to reconstruct the control-flow trace. For the example in Fig. 2b, if the prover delivers an address stream {N_2, N_{11}, N_2, N_2, N_{11}, N_2} and an accumulated hash value h_x, the measurement engine can reconstruct the control-flow trace $N_1 \rightarrow N_2 \xrightarrow{T_1} N_2 \xrightarrow{T_2} N_2 \xrightarrow{T_1} N_2 \rightarrow N_{10}$ with the call stack. In this procedure, the measurement engine hashes the return address of N_5 twice accumulatively because the sub-trace T_1 appears twice in this reconstructed control-flow trace. The measurement engine compares the hashing result with the received accumulated hash value h_x. If they are identical, we deem this execution conforms with CFI. Otherwise, we raise a CFI violation for this execution. Both the conformance and violation decisions raised by TEE are used to generate the evidence in aggregation reports (Section 4.2).

4.2 Collective attestation

4.2.1 Secure-world elements and primitives

The aggregation report is the main evidence propagated

Table 1 TEE-switching CFG nodes of typical runtime sub-traces of motivating example.

Sub-trace ID	Loop-related sub-trace	TEE-switching CFG nodes	
		OAT	SeCA
T_1	$\cdots \rightarrow N_2 \rightarrow N_3 \rightarrow N_4 \rightarrow N_{11} \rightarrow N_5 \rightarrow N_9 \rightarrow \cdots$	{N_2, N_3, N_{11}, N_5}	{N_2, N_{11}, N_5}
T_2	$\cdots \rightarrow N_2 \rightarrow N_3 \rightarrow N_6 \rightarrow N_7 \rightarrow N_9 \rightarrow \cdots$	{N_2, N_3, N_6}	{N_2}
T_3	$\cdots \rightarrow N_2 \rightarrow N_3 \rightarrow N_6 \rightarrow N_8 \rightarrow N_9 \rightarrow \cdots$	{N_2, N_3, N_6}	{N_2, N_8}

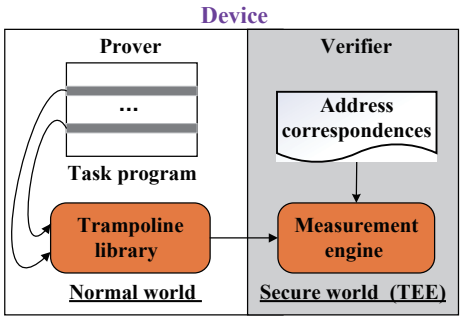


Fig. 3 Self-attestation workflow.

between the provers. For a network with n provers $P_1, P_2, \ldots, P_n$, the aggregation report ε is defined as $\varepsilon \equiv \{e^1, e^2, \ldots, e^n\}$, s.t., the node evidence e^i holds the knowledge of the prover P_i's healthiness. For a specific aggregation report, prover P_i's node evidence e^i is decided by the newest result of the local control-flow self-attestation of P_i. Meanwhile, other provers' node evidences are decided with the evidences aggregated from other devices and the execution procedure of the collective attestation protocol.

The local control-flow self-attestation avoids including concrete control-flow information, e.g., code addresses, into the aggregation report. Thus, we follow the principle of Ref. [17] to use bit vectors to represent the node evidence and the aggregation report. This design can significantly reduce the communication and storage overhead. To be more accurate than Ref. [17], we use a two-bit encoding to represent the node evidence in a ternary space of device status, as shown in Table 2. We use the bitwise-or as the aggregation operation over node evidence. This aggregation operation is intuitive since one device communicating with a network member indicates its communication with the network ($(00|*) = *$), and one device decided compromised by a network member should be strictly identified as compromised to the network ($(11|*) = 11$). Here $*$ means any two-bit binary. We define the operation primitives on the aggregation reports as Eqs. (1)–(3),

$$\text{AggRep}(\varepsilon_i, \varepsilon_j) = \{(e_i^k | e_j^k) \mid \forall k = 1, 2, \ldots, n\} \quad (1)$$

$$\text{InitRep}(\cdot) = \{e^k = 00 \mid \forall k = 1, 2, \ldots, n\} \quad (2)$$

$$\text{UpdateRep}(i, \varepsilon) = \varepsilon[e^i \leftarrow \text{CFA}(\cdot)] \quad (3)$$

Note that the prover P_i can only update its own node evidence in the aggregation report ε through a periodical local control-flow attestation procedure (CFA operation in Eq. (3)) in UpdateRep(i, ε) and keep other nodes' evidence unchanged. Each prover's task software executes iteratively at its own pace with different user commands or environmental inputs. The TEE updates the control-flow integrity status at the end of each execution, making the CFA operation asynchronous to the UpdateRep operation. Therefore, each time UpdateRep takes the newest result of CFA asynchronously.

The keys and the security primitives other than the report-related primitives are defined in Table 3. In our security settings, each device holds its own secret/public key pair, other devices' public keys, and the network owner's public key in the TEE. We require a session key for each device pair's secure communication. The session key can be established through pre-installation or updated through a key exchange scheme, e.g., Elliptic Curve Diffie Hellman (ECDH). We deem the generation and updating of the session keys out of our scope. The MAC key for each device pair is temporarily composed of other security elements in our collective attestation. To mitigate the physical attack of Adv_{hw}, all the devices are equipped with a synchronized real-time clock time. Their local time T_i is updated with time. Moreover, each device's TEE holds a heartbeat count δ which is increased at the synchronous and periodic time points when the device is online. For the time points $t_0, t_1, \ldots, t_{k-1}, t_k, t_{k+1}, \ldots$ synchronized among devices, if a device gets offline during $[t', t'']$, s.t., $t_k \in [t', t''] \subseteq (t_{k-1}, t_{k+1})$, then this device will have $\delta = k$ at t_{k+1} while other keeping-online devices have $\delta = k+1$ at t_{k+1}. Besides, we develop a trusted hash operation over each device's stored public keys, i.e., the disruption salt. In our collective attestation, the hashing-based update over

Table 2 Encoding of node evidence.

Encoding of e^i	Definition
00	Prover p_i has not been communicated with any other prover in the network.
01	Prover p_i is healthy. It has not been physically compromised by Adv_{hw} and the CFI of its software is enforced to resist Adv_{sw}.
11	Prover p_i is unhealthy. It has been physically compromised by Adv_{hw} or its software's control flow get hijacked by Adv_{sw}.

the disruption salt, i.e., pk← Hash(pk), is an instant operation to invalidate the MAC generated by the control-flow hijacked device.

4.2.2 Deployment phase

During the deployment phase, all the devices are deployed with a globally unique attestation iteration counter pid initialized to 0. The heartbeat count δ is initialized to 0 on each device. Furthermore, each device initializes an aggregation report ε_i with InitRep. Then, this aggregation report will be aggregated with other reports using AggRep or updated with UpdateRep based on the control-flow self-attestation results in the collective attestation.

4.2.3 Collective attestation phase

SeCA's collective attestation comprises multiple attestation iterations, identified by the iteration counter pid. To simplify our demonstration, we define a compound primitive SendRepMac in Table 3 to generate and send aggregation report with proper MAC to the specific receiver. A parameter message msg_0 is attached before the message delivery.

In the collective attestation procedure, the network owner $\mathcal{O}$ broadcasts an initialization message (❶ to ❷ in Fig. 4) to all the provers within its communication range (e.g., P_i) to start an attestation iteration. Upon receiving the iteration initialization message, P_i disseminates its newest aggregation report to the nearby provers (e.g., P_j) within its communication range (❷ to ❸ in Fig. 4). Any prover can participate in the current attestation iteration as a new participant (❸a)) or an old participant (❸b)) to receive attestation reports from other provers. Note that each prover has the functionalities of ❷ and ❸ to receive messages from either the network owner or other provers. At the end of each iteration, the network owner obtains the report aggregating the attestation evidence of the

Table 3 TEE-stored security primitives and elements.

Notation	Description
$(\text{sk}_i, \text{pk}_i)$	Secret and public key pair of device i
ck_{ij}	Session key between devices i and j
mk_{ij}	MAC key between devices i and j
GenSig(sk, msg)	Signature generation for message
VerSig(pk, sig, msg)	Signature verification for message
GenMac(mk, msg)	MAC generation for message
VerMac(mk, mac, msg)	MAC verification for message
send(i, msg)	Send message to device i (prover or owner)
SendRepMac(i, j, msg_0)	Compound primitive defined by algorithm: SendRepMac(i, j, msg_0){ $\text{mk}_{ij} \leftarrow \text{ck}_{ij}\|\|\delta_i\|\|\text{pk}_i\|\|\text{pid}_i$; $\text{mac}_{ij} \leftarrow$GenMac($\text{mk}_{ij}$, $\varepsilon_i\|\|T_i\|\|\text{pid}_i$); $\text{msg}_{\text{rep}} \leftarrow \varepsilon_i\|\|T_i\|\|\text{mac}_{ij}$; send($j$, $\text{msg}_0\|\|\text{msg}_{\text{rep}}$)}

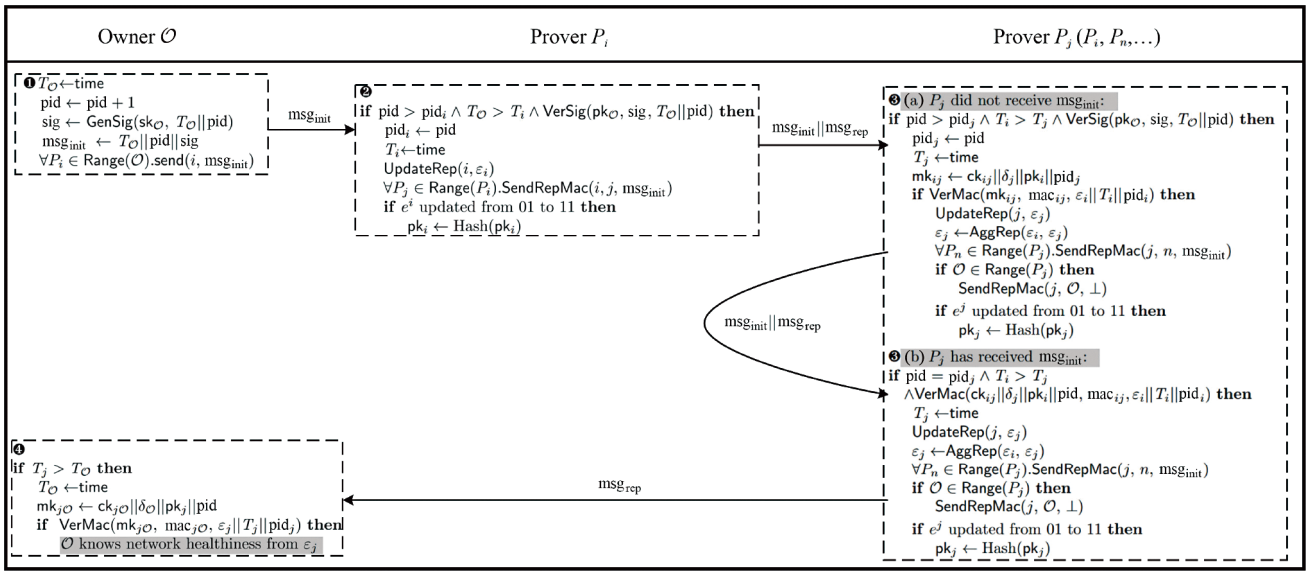


Fig. 4 Security protocol of SeCA.

involved provers, which reflects the trustworthiness of the network (❸ to ❹ in Fig. 4). We elaborate on the protocol details as follows.

❶ The network owner $\mathcal{O}$ initiates a new attestation iteration marked by a unique counter pid. The start timestamp $T_{\mathcal{O}}$ of this new iteration is combined with pid and signed by $\mathcal{O}$ using its private key. The signed message, msg_{init}, is broadcasted as the initialization message to all devices within $\mathcal{O}$'s communication range.

❷ The prover P_i verifies $\mathcal{O}$'s signature and knows it is notified to participate in a new iteration ($\text{pid} > \text{pid}_i$). Thus, P_i updates its iteration counter and local aggregation report ε_i with the newest result of control-flow self-attestation. Then, P_i generates the deliverable message and sends it to the nearby provers with SendRepMac. The message includes the aggregation report ε_i and the current timestamp T_i. The message is also attached with the msg_{init} of this attestation iteration and a message authentication code mac_{ij}. The message authentication code is obtained with a temporary MAC key, which is decided by the session key, the disruption salt, the local heartbeat count, and the current iteration counter (see Table 3). Using disruption salt (P_i's public key) to produce the MAC key allows TEE to invalidate subsequent MAC generations of the CFI-violated device by hashing the disruption salt ($\text{pk}_i \leftarrow \text{Hash}(\text{pk}_i)$). Therefore, after ❷, the message from a CFI-violated prover P_i cannot be accepted by other provers that use the correct public key pk_i to verify the MAC.

❸ P_j is a prover receiving messages from other provers in the attestation iteration. If P_j never receives msg_{init} and is a new participant in this iteration, P_j first validates the freshness and authenticity of the initialization message msg_{init}. After updating its iteration counter pid_j and timestamp T_j, P_j verifies the MAC for the authenticity of the received aggregation report with an appropriately generated MAC key. The MAC key contains the local heartbeat count to invalidate the message from a physically compromised device. P_j then updates its local aggregation report ε_j with the newest control-flow self-attestation result and aggregates the received report ε_i into its own report ε_j. If the current device's task software gets CFI-violated, then after broadcasting such CFI-violation evidence, the final hashing operation of ❸a in Fig. 4 on the disruption salt invalidates the subsequent MAC generations. In contrast, if P_j has received msg_{init} and thus is an old participant of this iteration (❸b in Fig. 4), P_j first confirms ε_i's freshness and authenticity and the absence of physical attacks through MAC verification. Then, P_j updates its timestamp and directly aggregates the received ε_i into its report ε_j. Similarly to ❷, in both cases, P_j sends the timestamped aggregation report along with its MAC using SendRepMac. When targeted on prover devices, the iteration initialization message msg_{init} is attached. The subsequent provers receiving the aggregation reports perform attestation in the same manner as P_j until the reports reach the network owner $\mathcal{O}$.

❹ The network owner $\mathcal{O}$ checks the received report ε_j for authenticity and physical-attack absence, and knows the network healthiness. The owner can further aggregate different aggregation reports sent by different provers to gain a more comprehensive knowledge of network status before launching another attestation iteration.

5 Security Analysis

We analyze SeCA's security against the adversarial model presented in Section 3. We follow Refs. [11, 31] to define a security experiment Exp_{Adv}. The adversary Adv follows the attacker model of Section 3 to conduct Exp_{Adv} and interact with the provers and the network owner during the protocol execution. We assume ℓ_{ck} and ℓ_{pk} are the lengths of the session key and public key, respectively. The function $g(\cdot)$ is polynomial in ℓ_{ck} and ℓ_{pk}. The outcome of the Exp_{Adv} is a binary value r. $r = 1$ means that the network owner accepts the attestation evidences, while $r = 0$ represents the network owner rejecting the evidence. SeCA's collective attestation protocol is secure if it is computationally infeasible for a polynomial attacker to produce fake attestation evidence in ε that the network owner accepts. Formally speaking,

$$\Pr\left[r = 1 | \text{Exp}_{\text{Adv}}\left(1^{g(\ell_{\text{ck}}, \ell_{\text{pk}})}\right) = r\right] < \text{negl}_{\text{Adv}} \tag{4}$$

where negl_{Adv} represents the negligible threshold of the assumed attacker. Formula (4) means that the probability (Pr) for the adversary (adv) to infer the attestation evidence acceptable by the network owner through the polynomial number of steps (in terms of the key lengths) of the security experiment (Exp_{Adv}) can be negligible ($< \text{negl}_{\text{Adv}}$).

5.1 Communication adversary

MITM attack: Adv_{com} can conduct the MITM attack.

In this attack, Adv_{com} attempts to intercept and tamper with the data transmission between two communicating principals, e.g., P_i and P_j. Adv_{com} endeavors to modify the message msg_{rep}. Adv_{com} cannot infer the MAC key mk_{ij} because mk_{ij} and the session key ck_{ij} are protected by TEE and cannot be obtained by Adv_{com} (see SendRepMac of Table 3). The probability of guessing the correct ck_{ij} in polynomial time by Adv_{com} is negligible. Consequently, the probability that Adv_{com} can successfully generate a valid and fresh MAC for msg_{rep} is negligible in $g(\ell_{\text{ck}}, \ell_{\text{pk}})$, i.e., $\Pr\left[r = 1 | \text{Exp}_{\text{MITM}}\left(1^{g(\ell_{\text{ck}}, \ell_{\text{pk}})}\right) = r\right]$ being negligible. Without mk_{ij}, the MITM attacker cannot tamper with the attestation evidence, and SeCA is secure against the MITM attack. Within a dynamic network environment, each device propagates its attestation evidence along multiple routines. Although Adv_{com} can intercept messages, the overall healthiness of the network remains unaffected by the activity of Adv_{com}.

Impersonation attack: Adv_{com} attempts to propagate crafted attestation evidence by forging attestation reports to impersonate a healthy device. However, due to the compound structure of the MAC key and TEE's protection, Adv_{com} cannot synthesize a valid MAC for the current attestation iteration. Similarly to the security against the MITM attack, Adv_{com}'s probability of forging the attestation report to make it accepted by other healthy devices is negligible.

Replay attack: Adv_{com} preserves the healthy attestation evidence of the attack-target device. After this device gets compromised, Adv_{com} replays the obsolete healthy evidence and confuses other devices from the genuine compromised status of the target device. The synchronized real-time clock time resists the replay attack. Each device compares the timestamp of the received message with its local timestamp. In each step of the collective attestation, if the device's local timestamp is newer than the received message's timestamp, it indicates a potential replay of the message. The device discards the message straightforwardly. Otherwise, the collective attestation proceeds, and the device updates its local timestamp to make the previously received message obsolete and concentrate on receiving more newer messages. Forging the timestamp in a message requires obtaining the MAC key mk, which is prohibited by the protection of ck and mk.

DoS attack: Adv_{com} continually injects forged messages or replays obsolete messages, aiming to exhaust the devices' communication bandwidth and computing resources. From the device's perspective, the DoS attack induced by message replaying is mitigated by the early rejection to the violation of timestamp comparison (e.g., $T_{\mathcal{O}} > T_i$, $T_i > T_j$, and $T_j > T_{\mathcal{O}}$) in each step of Fig. 4. Such early rejections avoid more time-consuming aggregation operations involved in DoS request processing. From the network perspective, the highly dynamic network scenario of SeCA avoids stable connection maintenance between devices. The attestation evidence can be propagated through multiple routes, thus mitigating the impact of one device's service failure on the overall collective attestation procedure.

5.2 Software adversary

Runtime control-flow hijacking: SeCA's control-flow self-attestation is designed to precisely identify the unexpected indirect control-flow branches triggered by ROP or Return2Libc. Specifically, SeCA's $\mathcal{V}^{\ell}$ reconstructs the control-flow path from the control-flow evidence of the prover's task software. The mismatch between the $\mathcal{V}^{\ell}$-generated hash value and the received accumulated hash value indicates the control-flow hijacking. The TEE ensures the control-flow self-attestation results are reflected in the device's node evidence e^i of the local aggregation report, which will be aggregated and the device's trustworthiness will be known by the network owner. Moreover, when detecting the control-flow integrity violation, the TEE conducts a hashing-based update to the disruption salt, i.e., the public key of the current device. This operation invalidates the subsequent MAC generated by this device since pk is a component of the MAC key. Furthermore, the other devices use the offline-deployed public key of this compromised device to verify the MAC, thus rejecting the message from this device. This design forbids this device's reports from being aggregated by other devices.

Black-hole attack: This attack is analogous to a software-induced DoS attack on some routing-critical device. Adv_{sw} can compromise the device to disrupt the normal execution of the prover's task program and to discard the aggregated attestation reports instead of propagating them. Due to the dynamic network scenario, the robustness of the multi-routes attestation evidence propagation significantly mitigates this threat.

5.3 Hardware adversary

Physical attack: Adv_{hw} employs physical intrusion to access the target device, circumvent hardware security measures, modify the code of the task program, and access secrets stored in the TEE. After successfully extracting the device's keys, Adv_{hw} may tamper with or forge messages. Because the device is not operational to keep updating the heartbeat count δ under the physical attack, δ easily becomes obsolete. Although Adv_{hw} can obtain the secrets in TEE, Adv_{hw} cannot tamper with δ in TEE. When the compromised device resumes from under physical attack to participating in the collective attestation, the device uses the obsolete δ as the component of the MAC key to generate the MAC of messages, which will be decided invalid by other devices that hold the correct heartbeat count.

Compared to LICAPA, SeCA's heartbeat-based physical attack detection has its restriction. The synchronous heartbeat has fixed interval T_δ, thus SeCA can only detect the physical attacks with $T_\delta < T_{\text{att}}$. In contrast, LICAPA detects physical attacks by generating a timestamp that a physical attacker cannot alter. Thus, LICAPA imposes no assumptions on the time required for a physical attack.

We summarize the attack mitigation capabilities of different approaches in Table 4.

5.4 Tamarin simulation

We utilize the security protocol verifier Tamarin[47] to conduct an automatic security analysis to prove the correctness and authenticity of the proposed collective attestation protocol. We adopt the Dolev−Yao adversary model, where the adversary has complete control over the communication channels, intercepts and stores transmitted messages, manipulates stored messages, or forges new messages to interact with other principals, thereby realizing typical attacks such as impersonation, MITM, and replay attacks. Furthermore, the adversary can be a legitimate entity that actively initiates the attestation iteration.

Tamarin models the protocol workflow by defining rules and restrictions. Based on the adversary model, when modeling the collective attestation protocol steps between the network owner and prover, we establish two roles, owner and prover, each with the keys defined in Table 3 between them. When modeling the protocol steps among provers, we establish two roles, ProverI and ProverJ, each with the keys between them.

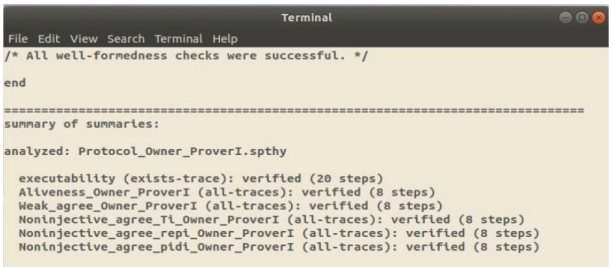
Tamarin requires lemmas to specify the protocol executability and various security properties. In our lemma specifications, Lemmas executability specify the correctness of the protocol steps. To specify Lowe's hierarchy of authentication properties[48], Lemmas aliveness specify the aliveness property between the principals. Lemmas weak_agree specify the weak agreement property between principals. Lemmas Noninjective_agree_$*$ specify the non-injective agreement property between principals, regarding variables T_i, T_j, pid_i, pid_j, ε_i, and ε_j, respectively. These three security properties specify the one-way authentication between principals jointly and progressively. Figure 5 presents the verification results, indicating that the collective attestation protocol successfully enables mutual authentication among principals.

6 Implementation and Evaluation

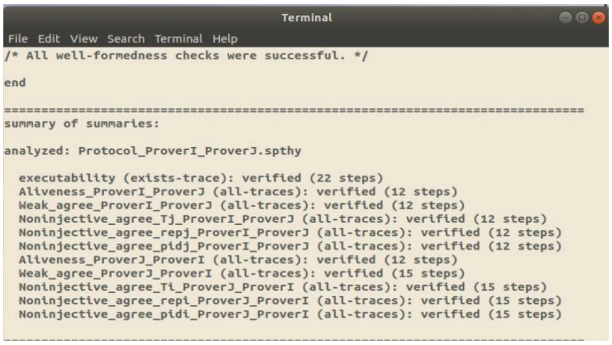
This section first introduces the implementation details and our network simulation setups. Then, we compare our control-flow self-attestation with the state-of-the-art control-flow attestation approach OAT[19] on the instrumented code size and the runtime overhead. To evaluate the collective attestation, we compare SeCA

Table 4 Effect of attack mitigation (strong mitigation (●), weak mitigation (◐), and no mitigation (○) against specific adversary).

Adversary type	PADS[10]	US-AID[45]	SALAD[14]	FADIA[16]	PASTA[12]	Delica[13]	LICAPA[17]	SeCA
MITM	●	●	●	●	●	●	●	●
Impersonation	●	●	●	●	●	●	●	●
Replay	●	●	○	●	●	●	●	●
DoS@network	●	◐	●	◐	●	●	●	●
DoS@device	●	●	◐	●	●	●	●	●
Runtime CF Hijack	○	○	○	○	○	○	○	●
Black-hole	●	◐	●	◐	●	●	●	●
Physical attack	○	◐	○	◐	◐	◐	●	◐



(a) Results between network owner and prover P_i



(b) Results between provers P_i and P_j

Fig. 5 Tamarin verification results.

with other collective attestation approaches, i.e., SALAD[14], PASTA[12], Delica[13], and LICAPA[17], on per-device computational and communication cost and overall attestation performance.

6.1 Implementation

We implement SeCA on a Raspberry Pi 3b board equipped with a Broadcom SoC 1.4 GHz ARM Cortex-A53 (ARMv8) CPU, 1 GB of RAM, and the Linux 4.19.97-v7+ kernel (Raspbian GNU/Linux 10). The TEE is implemented using Open-TEE-22.04. The cryptographic primitives in Table 3 are implemented with OpenSSL-1.1.0. We use the BLAKE-2s hash function in the instrumented code and SeCA's $\mathcal{V}^{\ell}$ verifier to implement the online procedure of the local control-flow self-attestation. In the collective attestation protocol, we use SHA-256 as the hash function, HMAC-SHA-256 for MAC operations, Elliptic Curve Digital Signature Algorithm (ECDSA) for digital signatures, and ECDH for encryption and key exchange. To evaluate the adaptability of SeCA, we also deploy SeCA on a HiKey Kirin 620 board equipped with a 1.2 GHz ARM Cortex-A53 (ARMv8) CPU, 1 GB of RAM, and the Linux 4.19.5-hikey kernel (Debian GNU/Linux 9.13). We test the efficiency of control-flow self-attestation on Raspberry Pi 3b and HiKey Kirin 620 boards.

We employ the OMNeT++[49] discrete event simulator to simulate SeCA under a highly dynamic network environment, using the same configurations as LICAPA[17]. We deploy the Zigbee wireless communication protocol to simulate broadcast communication between devices. We use the built-in measurement function of the simulator to measure the communication overheads. The nodes are deployed within a 5×5 km^2 area. In initialization, the provers are randomly distributed within the area. Their movements during the simulation follow the random waypoint mobility model, i.e., each prover repeatedly selects a random destination within the area and then moves towards that destination at a specified speed (10–20 m/s). The communication rate of the provers is set to 250 Kilo-bytes per second (Kbps), with a communication range of 50 m.

6.2 Efficiency of control-flow self-attestation

We demonstrate the efficiency of SeCA's control-flow self-attestation by comparing it with OAT[19]. As presented in Section 4.1, SeCA outperforms OAT in both the instrumented code size and runtime overhead. We use three open-source embedded programs from Ref. [19] as the benchmarks, as shown in Tables 5 and 6. In these programs, we instrument the same operations as described in Ref. [19]. We instrument and build the code into binary. We use a crafted indirect-branch injection to test the functionality of the control-flow attestation of both approaches. Then, we present the number of instrumented basic blocks and the binary code size of instrumented programs, as shown in Table 5. Compared with OAT, SeCA reduces the number of instrumented basic blocks by 12.28% to 27.24%. Since we instrument different code into the basic blocks, we observe that SeCA causes the code

Table 5 Comparison of number of instrumented basic blocks and code size.

Program	Number of instrumented BBLs		Code size (Kb)			Reduction (%)
	OAT	SeCA	Original	OAT	SeCA	
House alarm system (alarm)	279	203	25.54	28.87	27.96	3.56
Rover controller (rc)	57	50	8.02	8.91	8.73	2.24
Syringe pump (sy)	2457	2136	10.83	12.23	12.05	1.66

size increment by 8.85%–11.27%, and compared with OAT, SeCA reduces the code size increment by 1.66%–3.56%. Moreover, we compared the runtime overhead of the control-flow attestation between OAT and SeCA, as shown in Table 6. SeCA reduces the attestation time overhead by 0.30%–5.86% on Raspberry Pi 3b and 0.27%–7.34% on HiKey Kirin 620.

6.3 Collective attestation

6.3.1 Per-device measurement and cost

Figure 6 presents the comparison results of the aggregation report size of different approaches. In LICAPA, healthy devices accurately detect and reject the messages from physically compromised devices. Consequently, LICAPA's attestation report requires only one-byte evidence and one-bit attestation status for each device. SALAD and Delica have the same attestation report design. In their attestation reports, each device has the same length of device identifier (two bytes for upper to 65 536 devices) and attestation evidence (one byte). PASTA uses multi-signatures to generate evidence and requires additional space to store the commitments for multi-signatures. Thus, PASTA's reports are much longer. In SeCA, whether a device has been physically compromised is decided through the heartbeat count delivered to other devices, which is unrelated to the aggregation report. Each two-bit node evidence e^i contained in the aggregation report reflects the control-flow integrity of devices' task programs, leading to $\lceil n/4 \rceil$-byte aggregation report, where n is the number of provers. As a result, compared with LICAPA, SeCA's report size is reduced by 77.8%.

We measure the time cost of the implemented cryptographic primitives on the real board, as presented in Table 7. We also investigate the time costs of the critical aggregation report operations AggRep and UpdateRep. Figure 7 compares the time cost of report aggregation operations of different approaches. Specifically, the healthy device identifier merging, deduplication, and the corresponding report aggregation operations are more costly for other approaches than for SeCA. According to Fig. 7, PASTA has extremely high report aggregation costs. In contrast, SeCA's attestation report aggregation is most efficient with a complexity linear to the device number. The time costs for updating the attestation report are compared in Table 8. All the approaches in Table 8 require generating or updating the devices' attestation evidence in each attestation iteration for the verifier's verification. Differently from other approaches, the time cost of SeCA's UpdateRep should be affected by the control-flow self-attestation cost, as defined in Eq. (3). In SeCA's implementation, the control-flow

Table 6 Runtime attestation overhead.

Board	Program	Self-attestation time (ms)		Reduction (%)
		OAT	SeCA	
Raspberry Pi 3b	alarm	138.69	135.04	2.63
	rc	111.64	111.31	0.30
	sy	189.18	178.09	5.86
HiKey Kirin 620	alarm	142.22	138.71	2.47
	rc	116.49	116.17	0.27
	sy	196.83	182.38	7.34

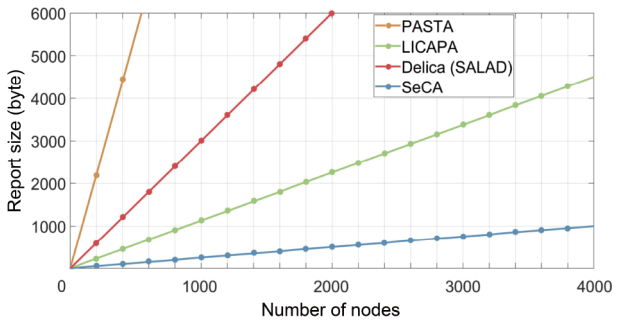


Fig. 6 Aggregation report size comparison of different approaches.

Table 7 Performance of attestation primitives of SeCA.

Attestation primitive	Time cost (ms)
GenSig/VerSig	9.03/8.72
GenMac/VerMac	3.56/3.89

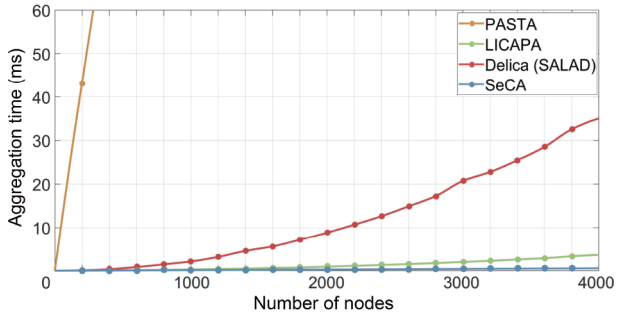


Fig. 7 Report aggregation time comparison of different approaches.

Table 8 Time cost comparison of different approaches on report updating.

(ms)

SALAD	PASTA	Delica	LICAPA	SeCA
2.24	23.87	2.11	1.73	0.16

self-attestation procedure is asynchronous to minimize the impact of self-attestation on the report updating operation. Thus, the time cost in Table 8 is the time to update the newest and existing control-flow attestation result into the aggregation report.

Based on the above cost measurements, we estimate the per-device computational cost in Table 9 according to the collective attestation protocols of different approaches. In the formulas of Table 9, n represents the number of provers within the communication range of a prover (or an aggregator of Delica). The formulas estimate the upper bound of one device's computational cost since we assume the device successfully makes one-step protocol progress with all the other devices in its range to propagate the evidence. To be more concrete, we present the per-device computational cost of different approaches, correlated with varying densities of nodes in Fig. 8. Because Delica and SALAD have identical report aggregation costs, SALAD's per-device computational cost is almost identical to Delica's aggregator; thus we ignore SALAD's curve in Fig. 8. We also ignore Delica's prover cost in Fig. 8 since it is minor and loosely related with node density. We observe that SeCA's per-device computational cost tends to be significantly lower than other approaches.

6.3.2 SeCA network performance simulation

This section compares the per-device communication cost and the overall performance of collective attestation between SeCA and other state-of-the-art approaches. We also investigate the impact of network dynamics on SeCA's attestation efficiency by comparing the overhead of default SeCA with a device-motionless setting of SeCA.

Average per-device communication cost. Figure 9a presents the average per-device communication cost of the collective attestation in simulated highly dynamic networks with different scales. The aggregated attestation reports are transmitted between devices, and the report size significantly impacts the communication costs. Besides, the aggregation scheme is a key factor influencing the communication overhead of each approach. Since Delica categorizes its nodes into provers and aggregators, we set the proportion of aggregators to 20%, following the optimal ratio described in Ref. [13]. Figure 9a demonstrates that SeCA has the lowest per-device communication overhead compared with other approaches, leading to a reduction of 31.3%–76.2% compared with LICAPA. To further investigate the impact of network dynamics on attestation efficiency, Fig. 9b compares the average per-device communication cost of SeCA with that of a device-motionless setting of SeCA. In this evaluation and the following evaluations that require static network settings, we confirm that the random and static device-location deployment ensures the network forms a connected graph. From Fig. 9b, we can observe that the overhead in dynamic networks is significantly higher than in static networks. The average overhead per device in static networks is only 6.2%–24.1% of that in dynamic networks because dynamic networks lacking stable topology leads to increased communication interactions and consequent higher overhead.

Overall performance of collective attestation in normal network. The overall performance of collective attestation is measured in the time required for the verifier or network owner to obtain the attestation evidence from all devices. Considering the practical application scenario of the UAV network and the network scale setting of the related work[14], we conduct our evaluation on a highly dynamic network comprising 3000 devices. The topology of dynamic networks is unstable. Thus, we measure the time cost 20 times for each approach and compared the average costs. For fairness, we assume that none of the devices

Table 9 Comparison on per-device computational cost estimation (VM: VerMac, GM: GenMac, GS: GenSig, VS: VerSig, AR: AggRep, UR: UpdateRep).

	Time cost estimation
SALAD	$T_{\mathrm{VS}} + T_{\mathrm{UR}}^{\mathrm{SALAD}} + (T_{\mathrm{VM}} + T_{\mathrm{AR}}^{\mathrm{SALAD}} + T_{\mathrm{GM}}) \cdot (n-1)$
PASTA	$T_{\mathrm{VS}} + T_{\mathrm{UR}}^{\mathrm{PASTA}} + (T_{\mathrm{AR}}^{\mathrm{PASTA}} + T_{\mathrm{GS}}) \cdot (n-1)$
Delica	Prover: $T_{\mathrm{VS}} + T_{\mathrm{VM}} + T_{\mathrm{AR}}^{\mathrm{Delica}} + T_{\mathrm{GM}} + T_{\mathrm{UR}}^{\mathrm{Delica}}$, Aggregator: $T_{\mathrm{VS}} + T_{\mathrm{GS}} + T_{\mathrm{UR}}^{\mathrm{Delica}} + (T_{\mathrm{VM}} + T_{\mathrm{AR}}^{\mathrm{Delica}} + T_{\mathrm{GM}}) \cdot (n-1)$
LICAPA	$T_{\mathrm{VS}} + T_{\mathrm{UR}}^{\mathrm{LICAPA}} + (T_{\mathrm{VM}} + T_{\mathrm{VS}} + T_{\mathrm{AR}}^{\mathrm{LICAPA}} + T_{\mathrm{GS}} + T_{\mathrm{GM}}) \cdot (n-1)$
SeCA	$T_{\mathrm{VS}} + T_{\mathrm{UR}}^{\mathrm{SeCA}} + (T_{\mathrm{VM}} + T_{\mathrm{AR}}^{\mathrm{SeCA}} + T_{\mathrm{GM}}) \cdot (n-1)$

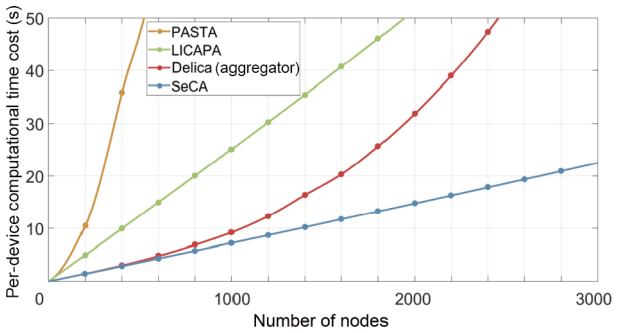
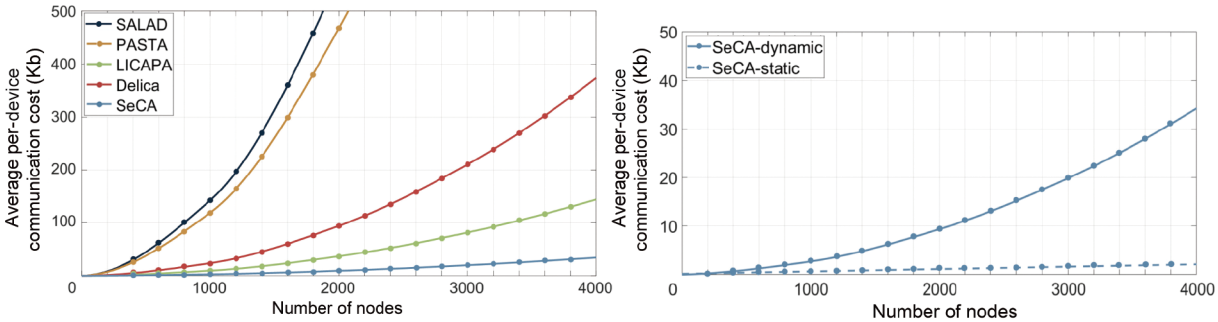


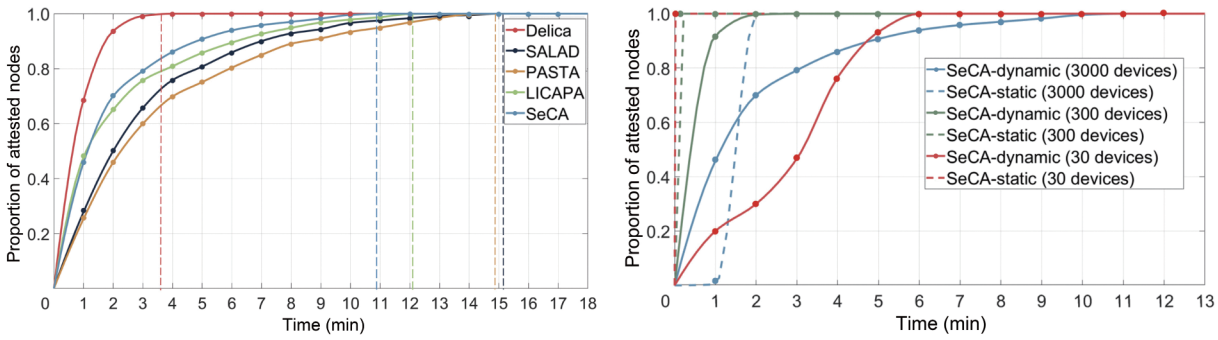
Fig. 8 Per-device computational time cost comparison of different approaches.

are compromised in this evaluation. The results are presented in Fig. 10a. SeCA takes a shorter time to complete the current attestation iteration than SALAD, PASTA, and LICAPA because SeCA's aggregation report only contains two-bit evidence for each device, thus reducing the communication and aggregation cost. Compared with SALAD, PASTA, and LICAPA, SeCA reduces the overall collective attestation cost by 27.6%, 25.2%, and 10.1%, respectively. However, SeCA falls short in its overall performance compared with Delica. This is because Delica restricts the evidence aggregation to the aggregators and the provers do not communicate with each other, thus reducing the overall number of messages in each attestation iteration. Delica's overall efficiency is obtained at the cost of weaker robustness that relies on capable aggregators. Furthermore, we investigate the impact of network dynamics on the overall network attestation performance by comparing SeCA with the device-motionless SeCA in networks with 30, 300, and 3000 devices, respectively. The results are presented in Fig. 10b. For these networks, it is evident that the attestation cost in static networks is significantly lower than in dynamic networks. The unstable network topology substantially impacts the overall network attestation cost. Besides, the network density also impacts the network attestation cost. We observed that the attestation time cost for the 30-device dynamic network is even longer than that for the 300-device dynamic network. Intuitively, the 30-device dynamic network is too sparse, requiring more cost to establish communication with other devices. When the network density increases to a certain extent, the overall network attestation cost becomes positively correlated with the number of devices; thus, attesting the 300-



(a) Comparison among different schemes (b) Static and dynamic network comparison of SeCA

Fig. 9 Average per-device communication cost of collective attestation.



(a) Comparison between different schemes (n = 3000) (b) Static and dynamic network comparison of SeCA

Fig. 10 Overall collective attestation performance. The vertical dashed lines in (a) indicate when a node obtained the attestation evidence of all the network devices using different approaches. Ideally, this node can communicate with the verifier or the network owner to complete the current collective attestation iteration.

device network takes a shorter time than attesting the 3000-device network.

Performance comparison in networks with different delays. In real-world environments, network delays affect the devices' communication efficiency. We present the impact of network delays on the attestation efficiency in Fig. 11. Considering that the typical network delay for the Zigbee protocol ranges from 10 to 30 ms, we set the network delays in our simulations within the intervals of 10–20 ms and 20–30 ms for both static and dynamic networks. The attestation efficiency results in Fig. 11 indicate that in the 3000-motionless-device network, the attestation costs are increased by 100.0% under the 10–20 ms network delay and 156.5% under the 20–30 ms network delay. In contrast, for the dynamic network, the attestation costs are increased by 52.0% under the 10–20 ms network delay and 85.1% under the 20–30 ms network delay. These results demonstrate that the network delays have a considerable positive correlation with the attestation overheads of the SeCA network.

6.3.3 Effectiveness of SeCA under attacks

SeCA performance under physical attack. Physical attacks pose the most significant threat to collective attestation because the hardware adversary can obtain all the secrets stored in the hardware through the attack. Since a device is forced to leave the network during a physical attack, we assume that a physical attack takes longer than the minimum time interval T_{att}. SeCA detects physical attacks through the compromised device's missing heartbeats. Therefore, the heartbeat interval T_δ should be less than T_{att}. Otherwise, the physical attack cannot be detected. To quantify the effects of physical attack in simulation, we use two metrics: the proportion of physically compromised devices and whether the physical attack is detectable ($T_\delta < T_{\text{att}}$ or $T_\delta \geqslant T_{\text{att}}$). In the simulations, we set T_{att} to two minutes. We chose different T_δ to ensure the attack is detectable or undetectable in different simulation scenarios for the comparison. We raise the physical attack on a specific proportion of devices at the end of the first minute. The results are presented in Fig. 12. When $T_\delta > T_{\text{att}}$, SeCA erroneously identifies the compromised devices as healthy devices, allowing all devices to continue the communication. The collective attestation efficiency is affected moderately due to the temporary offline of the compromised devices. In contrast, when $T_\delta < T_{\text{att}}$, the physical attacks are detected, and the compromised devices are excluded from subsequent communication and transmission, leading to more significant increases in the collective attestation cost.

The physical attack detection capability of SeCA relies on a synchronized real-time clock time. However, in the real-world network, clock drift may occur, leading to unsynchronized real-time clock and incorrect detection results. As a result, SeCA can mistakenly identify a healthy device as being under physical attack. In contrast, the physical attack detection approach of LICAPA[17] is less affected by the clock drift as explained in Section 5.3. To investigate such differences in clock-drift robustness, we measure the time cost for identifying the false alarms caused by the clock drifts. We set T_{att} to 120 s and T_δ to 60 s. We set the clock drift to 90 s. The simulations are conducted on a 30-device network in a 1×1 km^2 area. We randomly select five devices from the network to raise the clock drift. From Fig. 13, we observe that because the clock drift is longer than T_δ, SeCA identifies the five clock-drifted devices as being

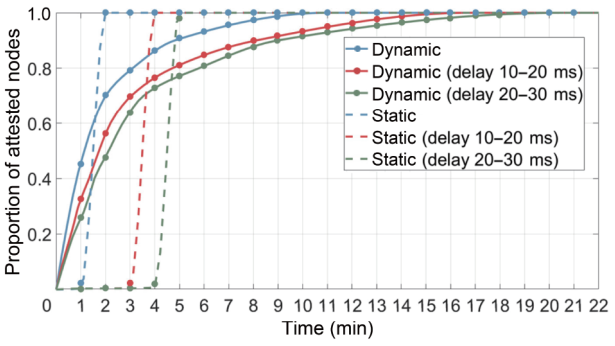


Fig. 11 Collective attestation efficiency comparison on static and dynamic networks with different network delays (n = 3000).

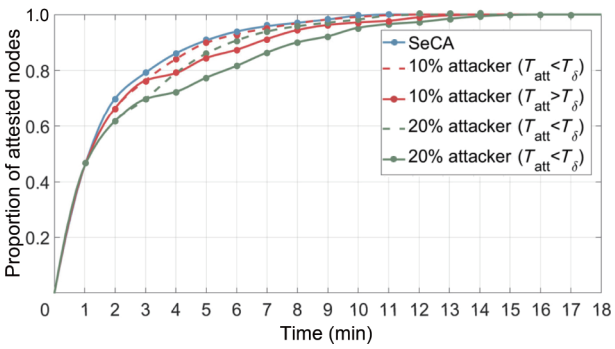


Fig. 12 Effectiveness of SeCA against different levels of physical attacks.

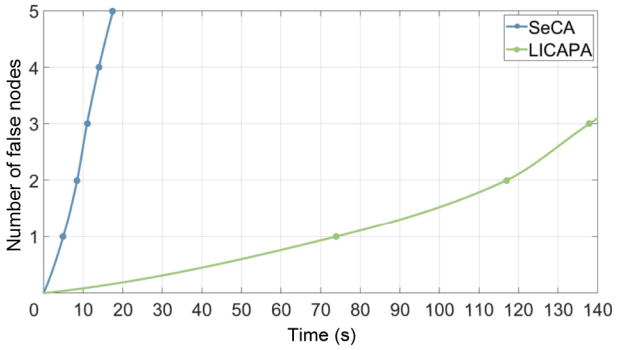


Fig. 13 False positive physical attack detection caused by clock drifts.

physically compromised very soon. In contrast, for LICAPA, if the clock-drifted device can communicate with other devices in 30 s (120 s − 90 s), such online fact can be propagated and potentially avoid the clock-drifted device being identified as under physical attack. Therefore, LICAPA raises the false alarms much slower than SeCA, as shown in Fig. 13. These results demonstrate the comparison in Table 4.

SeCA performance under impersonation and replay attacks. Impersonation and replay attacks are representative communication attacks threatening the effectiveness of SeCA. Under the default dynamic network setting, we investigate the impact of impersonation attacks on the collective attestation cost by simulations respectively on 30, 300, and 3000-device networks. We assume that an attacker device launched 1000 impersonation-attacking actions on the network devices. The overall attestation time costs under the impersonation attack are illustrated in Fig. 14. By comparing Fig. 14 with the dynamic networks' results in Fig. 10b, we conclude that the 1000 impersonation attacking actions have very slight impact on the overall network attestation cost. This is because when any healthy device is attacked, the attacker cannot obtain the correct MAC key, leading the device to early reject further interactions during the VerMac step in Fig. 4. When it comes to the replay attacks, the simulation results show that even less deviation in attestation cost from the attack-absent cases, because the replay attacks can be detected efficiently by verifying the timestamp freshness. In conclusion, both impersonation and replay attacks do not distinctly affect the overall attestation cost of SeCA. Moreover, we consider SeCA's defense success rate against the impersonation attack. Because the attacker cannot obtain the MAC key but can only guess the key with a probability of $1/2^{256}$, the defense success rate against the 1000 attacking attempts is $(1-1000/2^{256}) \approx 1$, demonstrating SeCA's defense effectiveness against impersonation attacks.

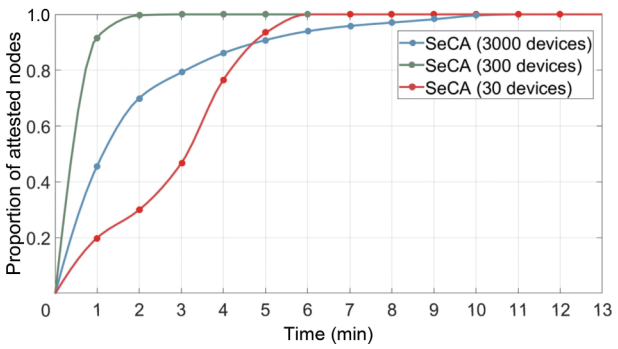


Fig. 14 Collective attestation performance under impersonation attack.

7 Discussion

The typical application scenario of SeCA is the UAV network. Therefore, we use a network scale with thousands of devices (typically 3000) in our network simulations. Adapting SeCA to a very large network scale is left for future work. A security-related limitation of SeCA is the proper settings of the minimum time interval and the synchronized heartbeat interval. Under the requirement of $T_\delta < T_{\text{att}}$ for detecting physical attacks, excessively long T_δ can result in undetected physical attacks. On the other hand, shorter T_δ can lead to increased runtime overheads. It will be preferred to set T_δ and T_{att} based on the specific application scenario. For example, physically compromising a UAV and making it rejoin the network can take at least several minutes. Implementing adaptive heartbeat intervals based on the network scale, density, and transmission speed will facilitate the robustness of SeCA, which is left for future work.

8 Conclusion

SeCA is an efficient collective attestation framework designed to detect and mitigate runtime control-flow hijacking attacks on the device's software and physical attacks in highly dynamic device networks. SeCA's collective attestation employs an efficient attestation report aggregation scheme with the TEE-facilitated control-flow self-attestation to reduce the network's overall communication overhead and achieve high overall and per-device efficiency. SeCA's program

instrumentation approach reduces the target program code size and the runtime overhead of control-flow self-attestation compared with the state-of-the-art control-flow attestation OAT. To the best of our knowledge, SeCA is the first collective attestation framework resisting both physical attacks and runtime control-flow hijacking.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (Nos. 62272366 and 62232013), the Key Research and Development Program of Shaanxi Province (No. 2023-YBGY-371), and the Basic Natural Science Research Program of Shaanxi Province (No. 2022JQ-691).

Declaration of competing interest

The authors have no competing interests to declare that are relevant to the content of this article.

References

[1] W. He, M. Golla, R. Padhi, J. Ofek, M. Dürmuth, E. Fernandes, and B. Ur, Rethinking access control and authentication for the home Internet of Things (IoT), in *Proc. 27th USENIX Conf. Security Symp*., Baltimore, MD, USA, 2018, pp. 255–272.

[2] J. Zhang, Y. Lu, Y. Wu, C. Wang, D. Zang, A. Abusorrah, and M. Zhou, PSO-based sparse source location in large-scale environments with a UAV swarm, *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 5, pp. 5249–5258, 2023.

[3] Y. Liu, J. Wang, Y. Zhang, L. Cheng, W. Wang, Z. Wang, W. Xu, and Z. Li, Vernier: Accurate and fast acoustic motion tracking using mobile devices, *IEEE Trans. Mob. Comput.*, vol. 20, no. 2, pp. 754–764, 2021.

[4] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, A large-scale analysis of the security of embedded firmwares, in *Proc. 23rd USENIX Conf. Security Symp*., San Diego, CA, USA, 2014, pp. 95–110.

[5] N. Corteggiani, G. Camurati, and A. Francillon, Inception: System-wide security testing of real-world embedded systems software, in *Proc. 27th USENIX Conf. Security Symp*., Baltimore, MD, USA, 2018, pp. 309–326.

[6] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, SoK: Security evaluation of home-based IoT deployments, in *Proc. IEEE Symp. Security and Privacy*, San Francisco, CA, USA, 2019, pp. 1362–1380.

[7] F. Dang, Z. Li, Y. Liu, E. Zhai, Q. A. Chen, T. Xu, Y. Chen, and J. Yang, Understanding fileless attacks on Linux-based IoT devices with HoneyCloud, in *Proc. 17th Annual Int. Conf. Mobile Systems*, *Applications*, *and Services*, Seoul, Republic of Korea, 2019, pp. 482–493.

[8] D. Dai, Z. An, and L. Yang, Inducing wireless chargers to voice out for inaudible command attacks, in *Proc. 2023 IEEE Symp. Security and Privacy*, San Francisco, CA, USA, 2023, pp. 1789–1806.

[9] N. Asokan, F. Brasser, A. Ibrahim, A. R. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann, SEDA: Scalable embedded device attestation, in *Proc. 22nd ACM SIGSAC Conf. Computer and Communications Security*, Denver, CO, USA, 2015, pp. 964–975.

[10] M. Ambrosin, M. Conti, R. Lazzeretti, M. M. Rabbani, and S. Ranise, PADS: Practical attestation for highly dynamic swarm topologies, in *Proc. 2018 Int. Workshop on Secure Internet of Things*, Barcelona, Spain, 2018, pp. 18–27.

[11] S. Frontera and R. Lazzeretti, Bloom filter based collective remote attestation for dynamic networks, in *Proc. 16th Int. Conf. Availability*, *Reliability and Security*, Vienna, Austria, 2021, p. 80.

[12] F. Kohnhäuser, N. Büscher, and S. Katzenbeisser, A practical attestation protocol for autonomous embedded systems, in *Proc. 2019 IEEE European Symp. Security and Privacy*, Stockholm, Sweden, 2019, pp. 263–278.

[13] Z. Wang, C. Sun, Q. Yao, D. Ding, and J. Ma, Delica: Decentralized lightweight collective attestation for disruptive IoT networks, in *Proc. 27th Int. Conf. Parallel and Distributed Systems*, Beijing, China, 2021, pp. 364–371.

[14] F. Kohnhäuser, N. Büscher, and S. Katzenbeisser, SALAD: Secure and lightweight attestation of highly dynamic and disruptive networks, in *Proc. 2018 on Asia Conf. Computer and Communications Security*, Incheon, Republic of Korea, 2018, pp. 329–342.

[15] M. Conti, P. Kaliyar, M. Rabbani, and S. Ranise, Attestation-enabled secure and scalable routing protocol for IoT networks, *Ad Hoc Netw.*, vol. 98, p. 102054, 2020.

[16] M. Mansouri, W. Ben Jaballah, M. Önen, M. Rabbani, and M. Conti, FADIA: Fairness-driven collaborative remote attestation, in *Proc. 14th ACM Conf. Security and Privacy in Wireless and Mobile Networks*, Abu Dhabi, UAE, 2021, pp. 60–71.

[17] Z. Wang and C. Sun, LICAPA: Lightweight collective attestation for physical attacks detection in highly dynamic networks, *Pervasive Mob. Comput.*, vol. 99, p. 101903, 2024.

[18] T. Abera, N. Asokan, L. Davi, J. E. Ekberg, T. Nyman, A. Paverd, A. R. Sadeghi, and G. Tsudik, C-FLAT: Control-flow attestation for embedded systems software, in *Proc. 2016 ACM SIGSAC Conf. Computer and Communications Security*, Vienna, Austria, 2016, pp. 743–754.

[19] Z. Sun, B. Feng, L. Lu, and S. Jha, OAT: Attesting operation integrity of embedded devices, in *Proc. 2020 IEEE Symp. Security and Privacy*, San Francisco, CA, USA, 2020, pp. 1433–1449.

[20] T. Abera, R. Bahmani, F. Brasser, A. Ibrahim, A. R. Sadeghi, and M. Schunter, DIAT: Data integrity attestation for resilient collaboration of autonomous systems, in *Proc. 26th Annu. Network and Distributed*

System Security Symp., San Diego, CA, USA, 2019. https://www.ndss-symposium.org/ndss-paper/diat-data-integrity-attestation-for-resilient-collaboration-of-autonomous-systems.

[21] D. Fu and X. Peng, TPM-based remote attestation for Wireless Sensor Networks, *Tsinghua Science and Technology*, vol. 21, no. 3, pp. 312–321, 2016.

[22] T. P. Stremlau, Integrating post-quantum cryptography into trusted computing group standards, in *Proc. 10th Int. Conf. Information Systems Security and Privacy*, Rome, Italy, 2024, p. 15.

[23] X. Kovah, C. Kallenberg, C. Weathers, A. Herzog, M. Albin, and J. Butterworth, New results for timing-based attestation, in *Proc. 2012 IEEE Symp. Security and Privacy*, San Francisco, CA, USA, 2012, pp. 239–253.

[24] S. Zeitouni, G. Dessouky, O. Arias, D. Sullivan, A. Ibrahim, Y. Jin, and A. R. Sadeghi, ATRIUM: Runtime attestation resilient under memory attacks, in *Proc. 2017 IEEE/ACM Int. Conf. Computer-Aided Design*, Irvine, CA, USA, 2017, pp. 384–391.

[25] Y. Li, J. M. McCune, and A. Perrig, VIPER: Verifying the integrity of peripherals' firmware, in *Proc. 18th ACM Conf. Computer and Communications Security*, Chicago, IL, USA, 2011, pp. 3–16.

[26] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, SWATT: Software-based attestation for embedded devices, in *Proc. IEEE Symp. Security and Privacy*, Berkeley, CA, USA, 2004, pp. 272–282.

[27] F. Armknecht, A. R. Sadeghi, S. Schulz, and C. Wachsmann, A security framework for the analysis and design of software attestation, in *Proc. 2013 ACM SIGSAC Conf. Computer and Communications Security*, Berlin, Germany, 2013, pp. 1–12.

[28] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, SMART: Secure and minimal architecture for (establishing dynamic) root of trust, in *Proc. 19th Annu. Network and Distributed System Security Symp*., San Diego, CA, USA, 2012. https://www.ndss-symposium.org/ndss2012/ndss-2012-programme/smart-secure-and-minimal-architecture-establishing-dynamic-root-trust.

[29] P. Koeberl, S. Schulz, A. R. Sadeghi, and V. Varadharajan, TrustLite: A security architecture for tiny embedded devices, in *Proc. 9th European Conf. Computer Systems*, Amsterdam, The Netherlands, 2014, p. 10.

[30] S. Luo, T. Hu, N. Han, and Y. Qian, A smart-contract-based policy-domain access control framework for distributed industrial IoT, *IEEE Internet Things J.*, vol. 11, no. 7, pp. 11427–11443, 2024.

[31] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A. R. Sadeghi, and M. Schunter, SANA: Secure and scalable aggregate network attestation, in *Proc. 2016 ACM SIGSAC Conf. Computer and Communications Security*, 2016, pp. 731–742.

[32] A. Ibrahim, A. R. Sadeghi, G. Tsudik, and S. Zeitouni, DARPA: Device attestation resilient to physical attacks, in *Proc. 9th ACM Conf. Security and Privacy in Wireless and Mobile Networks*, Darmstadt, Germany, 2016, pp. 171–182.

[33] A. Diop, M. Laurent, J. Leneutre, and J. Traoré, CoRA: A scalable collective remote attestation protocol for sensor networks, in *Proc. 6th Int. Conf. Information Systems Security and Privacy*, Valletta, Malta, 2020, pp. 84–95.

[34] F. Kohnhäuser, N. Büscher, S. Gabmeyer, and S. Katzenbeisser, SCAPI: A scalable attestation protocol to detect software and physical attacks, in *Proc. 10th ACM Conf. Security and Privacy in Wireless and Mobile Networks*, Boston, MA, USA, 2017, pp. 75–86.

[35] M. Ammar, M. Washha, G. S. Ramabhadran, and B. Crispo, SlimIoT: Scalable lightweight attestation protocol for the Internet of Things, in *Proc. 2018 IEEE Conf. Dependable and Secure Computing*, Kaohsiung, China, 2018, pp. 1–8.

[36] H. Xiong, Q. Mei, Y. Zhao, L. Peng, and H. Zhang, Scalable and forward secure network attestation with privacy-preserving in cloud-assisted Internet of Things, *IEEE Sens. J.*, vol. 19, no. 18, pp. 8317–8331, 2019.

[37] L. Petzi, A. E. Ben Yahya, A. Dmitrienko, G. Tsudik, T. Prantl, and S. Kounev, SCRAPS: Scalable collective remote attestation for pub-sub IoT networks with untrusted proxy verifier, in *Proc. 31st USENIX Security Symp*., Boston, MA, USA, 2022, pp. 3485–3501.

[38] S. F. J. J. Ankergård, E. Dushku, and N. Dragoni, State-of-the-art software-based remote attestation: Opportunities and open issues for Internet of Things, *Sensors*, vol. 21, no. 5, p. 1598, 2021.

[39] A. Ibrahim, A. R. Sadeghi, and S. Zeitouni, SeED: Secure non-interactive attestation for embedded devices, in *Proc. 10th ACM Conf. Security and Privacy in Wireless and Mobile Networks*, Boston, MA, USA, 2017, pp. 64–74.

[40] X. Carpent, G. Tsudik, and N. Rattanavipanon, ERASMUS: Efficient remote attestation via self-measurement for unattended settings, in *Proc. 2018 Design, Automation and Test in Europe Conf. Exhibition*, Dresden, Germany, 2018, pp. 1191–1194.

[41] M. Conti, E. Dushku, and L. V. Mancini, RADIS: Remote attestation of distributed IoT services, in *Proc. 6th Int. Conf. Software Defined Systems*, Rome, Italy, 2019, pp. 25–32.

[42] N. Marastoni and M. Ceccato, Remote attestation of IoT devices using physically unclonable functions: Recent advancements and open research challenges, in *Proc. 5th Workshop on CPS&IoT Security and Privacy*, Copenhagen, Denmark, 2023, pp. 25–36.

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

[44] T. Li, J. Ma, and C. Sun, NetPro: Detecting attacks in MANET routing with provenance and verification, *Sci. China Inf. Sci.*, vol. 60, no. 11, p. 118101, 2017.

[45] A. Ibrahim, A. R. Sadeghi, and G. Tsudik, US-AID:

Unattended scalable attestation of IoT devices, in *Proc. 37th Symp. Reliable Distributed Systems*, Salvador, Brazil, 2018, pp. 21–30.

[46] Y. Zhang, X. Liu, C. Sun, D. Zeng, G. Tan, X. Kan, and S. Ma, ReCFA: Resilient control-flow attestation, in *Proc. 37th Annu. Computer Security Applications Conf.*, 2021, pp. 311–322.

[47] The Tamarin Team, Tamarin-Prover Manual, https://tamarin-prover.com/manual/master/tex/tamarin-manual.pdf, 2024.

[48] G. Lowe, A hierarchy of authentication specifications, in *Proc. 10th Computer Security Foundations Workshop*, Rockport, MA, USA, 1997, pp. 31–43.

[49] A. Varga and R. Hornig, An overview of the OMNeT++ simulation environment, in *Proc. 1st Int. Conf. Simulation Tools and Techniques for Communications, Networks and Systems and Workshops*, Marseille, France, 2008, p. 60.

Ziyu Wang received the BEng degree in information and computing sciences from Xi'an Shiyou University, China, in 2016. He is currently pursuing the PhD degree at School of Cyber Engineering, Xidian University, China. His research interests include remote attestation and security protocol design.

Cong Sun received the BEng degree in computer science from Zhejiang University, China, in 2005 and the PhD degree in computer science from Peking University, China, in 2011. He is a full professor at School of Cyber Engineering, Xidian University, China. His research interests include software security, UAV system security, and program analysis.

Qingsong Yao received the PhD degree in computer science and technology from Xi'an Jiaotong University, China, in 2012. He is currently an associate professor at School of Cyber Engineering, Xidian University, China. His research interests mainly encompass AI-based security in audio and image processing, as well as IoT security.

Jingwei Li received the PhD degree from Xi'an Jiaotong University, China, in 2020. She is currently a lecturer at the State Key Laboratory of Integrated Services Networks, School of Telecommunications Engineering, Xidian University, China. Her main research interests include federated learning, distributed machine learning, and data center networking