PReFeR : Physically Related Function based Remote Attestation Protocol

ANUPAM MONDAL, SHREYA GANGOPADHYAY, and DURBA CHATTERJEE, Indian Institute of Technology Kharagpur, India
HARISHMA BOYAPALLY, Temasek Labs @ Nanyang Technological University, Singapore
DEBDEEP MUKHOPADHYAY, Indian Institute of Technology Kharagpur, India

Remote attestation is a request-response based security service that permits a trusted entity (verifier) to check the current state of an untrusted remote device (prover). The verifier initiates the attestation process by sending an attestation challenge to the prover; the prover responds with its current state, which establishes its trustworthiness. Physically Unclonable Function (PUF) offers an attractive choice for hybrid attestation schemes owing to its low overhead security guarantees. However, this comes with the limitation of secure storage of the PUF model or large challenge-response database on the verifier end. To address these issues, in this work, we propose a hybrid attestation framework, named PReFeR, that leverages a new class of hardware primitive known as Physically Related Function (PReF) to remotely attest low-end devices without the requirement of secure storage or heavy cryptographic operations. It comprises a static attestation scheme that validates the memory state of the remote device prior to code execution, followed by a dynamic run-time attestation scheme that asserts the correct code execution by evaluating the content of special registers present in embedded systems, known as hardware performance counters (HPC). The use of HPCs in the dynamic attestation scheme mitigates the popular class of attack known as the time-of-check-time-of-use (TOCTOU) attack, which has broken several state-of-the-art hybrid attestation schemes. We demonstrate our protocol and present our experimental results using a prototype implementation on Digilent Cora Z7 board, a low-cost embedded platform, specially designed for IoT applications.

CCS Concepts: • **Security and privacy → Embedded systems security;**

Additional Key Words and Phrases: Remote attestation, dynamic attestation, fuzzy extractor, hardware performance counters

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES), 2023.
The work is partially supported by the project entitled "Development of Secure Hardware and Automotive Cyber-Physical Systems", funded by the National Mission on Interdisciplinary Cyber-Physical Systems, Department of Science and Technology (DST), Govt. of India.
Authors' addresses: A. Mondal, S. Gangopadhyay, D. Chatterjee, and D. Mukhopadhyay, Indian Institute of Technology Kharagpur, Kharagpur, West Bengal, India, 721302; emails: anupam17it@gmail.com, gangopadhyay.shreya09@gmail.com, durba.chatterjee94@gmail.com, debdeep.mukhopadhyay@gmail.com; H. Boyapally, Temasek Labs @ Nanyang Technological University, Singapore; email: harishma.boyapally@ntu.edu.sg.

1539-9087/2023/09-ART109 $15.00
https://doi.org/10.1145/3609104

ACM Reference format:
Anupam Mondal, Shreya Gangopadhyay, Durba Chatterjee, Harishma Boyapally, and Debdeep Mukhopadhyay. 2023. PReFeR : Physically Related Function based Remote Attestation Protocol. *ACM Trans. Embedd. Comput. Syst.* 22, 5s, Article 109 (September 2023), 23 pages.
https://doi.org/10.1145/3609104

1 INTRODUCTION

The rapid development of the Internet of Things (IoT) and Cyber-Physical Systems (CPS) has ensued the usage of embedded devices in various applications such as industrial automation, smart homes and wearable devices. While the proliferation of these low-end devices has increased their accessibility, it has also enhanced the attack surface. Since most market-ready devices do not have in-built security guarantees, owing to time, cost, and resource constraints, they are often prime targets of attacks such as malware injections that modify the software state of an embedded system. One such attack, *Stuxnet* [22] targeted the Programmable Logic Controllers (PLCs) in the Uranium enrichment facility. Such attacks necessitate the development of attestation mechanisms that verify the integrity of the internal state of a remote-embedded device. An attestation scheme involves a trusted party, known as a verifier that inspects the software state of an embedded device, also known as a prover. The protocol involves two phases, namely *setup* and *attestation*. The setup phase is executed in a secure setting before the device is deployed in the respective application. The verifier initiates the attestation process by sending a challenge to the prover, which generates a token indicating its current internal state. The verifier then compares the received token with the value generated at the verifier end using some prover-specific attributes pre-computed during the setup phase. For instance, in software-based attestation schemes, it is assumed that a verifier knows the memory content of the prover, which is used in the verification procedure [8, 38].

Current attestation mechanisms span the entire range from software-based attestation requiring no hardware support [38] to hardware-assisted attestation schemes involving secure co-processors or trusted execution platforms [10, 17]. Although software-based attestation schemes require no additional hardware implementation, it often relies on strong assumptions such as accurate computation time measurement and single-hop communication between the prover and verifier. On the other hand, hardware-based attestation protocols involve trust anchors such as Trusted Platform Module (TPM) [10] or SGX [17] that are inappropriate for low-end embedded devices and can not be scaled for IoT or CPS applications. To address these issues, a new genre of attestation schemes also known as hybrid attestation is proposed, which employs lightweight primitives like PUFs along with schemes implemented in software such as [32, 37]. PUF is a hardware-based security primitive that leverages the uncontrollable intrinsic device properties to realize a random and unpredictable function [24]. It takes a binary input as a challenge and generates a binary output known as the response. Since the PUF response depends on the given input as well as its physical characteristics, it is a popular candidate to bind the software state of the prover with the device-specific hardware properties.

Another class of hybrid attestation schemes known as Device Identifier Composition Engine (DICE)-based schemes [26] involve a hardware root of trust and classical cryptographic primitives. DICE-based schemes perform layered attestation of components using a set of sequentially generated keys. These keys are generated using symmetric key primitives. However, these schemes are known to incur significant hardware and integration overhead.

Most of the existing remote attestation protocols assume that the verifier is a trusted entity, that verifies the integrity of the resource-constrained prover device. Therefore, the PUF-based remote attestations protocols assume that it is secure to store the challenge-response database or a

software model of the PUF (embedded in the prover device) [8, 32, 37] on the verifier. Since the verifier is trusted, the prover trusts the attestation request originating from an authentic verifier. Another variant of remote attestation protocols [14] is proposed from the prover's perspective, where the adversary may impersonate the verifier and perform replay attacks, while the prover is benign. In this model, if the challenge-response database on the verifier is compromised, then the security of all the devices in the system is also jeopardized. Eventually, the prover ends up communicating with a fake verifier. In this work, we aim to build hybrid remote attestation protocols for IoT device owners, who wish to check if their device code is compromised. The protocol ensures that the user can verify the authenticity of the source of the attestation request along with verifying the correctness of the code on their device.

However, existing PUF-based attestation protocols suffer from the following limitations: (i) In case of schemes employing PUF models or CRP database, since the security of the protocol depends on the unpredictability of the PUF functionality, loss of the database jeopardizes the scheme as the adversary can use the responses from the database. Thus the PUFs become obsolete and the corresponding nodes need to be revoked and brought back to the server to replace the PUFs. (ii) The verifier in the PUF-based schemes are assumed to be resourceful in order to store the CRP database or the PUF model, which may not be scalable for collective or swarm attestations. In swarm attestation, a verifier attests several prover devices and in some cases, a prover can also act as a verifier for a set of nodes in the network [11]. To facilitate this, the prover and the verifier nodes are considered to be homogeneous in terms of resources. Additionally the operations performed at the verifier node must be lightweight so that it can be performed by a lightweight prover node. This necessitates the use of a lightweight hardware primitive that reduces the resource overhead of a verifier as well as prevents the revocation of prover nodes in case the verifier is compromised. To address these limitations, we adopt *physically related functions* (PReF) [16] that eliminates the requirement of CRP storage and enables building symmetric protocols. PReFs are the physical realization of *related* functions in hardware that enable a pair of devices each embodying a PUF instance to generate a similar response for a given set of challenges. In the proposed attestation framework, both the prover and the verifier embody a PReF instance and share a set of publicly available challenges, for which both entities produce similar responses.[1] The related challenges are stored in an immutable cloud or a public ledger. Since the related challenges do not reveal any information about the PReF responses, even if an adversary compromises the challenge database, it cannot obtain the PReF responses without physical access to the prover device. Additionally due to the immutability of the challenges, an adversary accessing the challenge database cannot alter it. In this protocol, PReF forms the hardware root-of-trust, which eliminates the requirement for secure storage of challenge-response database at verifier end. Since PReFs leverage the intrinsic properties of the host device, it binds the hardware characteristics to the attestation process, thereby authenticating the prover node as well.

In this work, we propose two attestation protocols - *static* attestation that verifies the software state of the prover node before the target code execution and *dynamic* attestation scheme that verifies the code execution on the remote node using the content of its hardware performance counters. Hardware performance counters (HPCs) are special registers that capture the statistics of low-level hardware events such as cache-hit, cache-miss, number of branch instructions, etc., thereby providing proof of execution of attested code. Due to the high granularity of these events, HPCs have been used to identify the presence of malicious code in low-end devices [7, 29]. In this work, we employ HPCs to verify the code execution flow in a remote device. To summarize, the major contributions of this work are:

[1]We refer to these challenges as related challenges in the following sections.

- We propose a lightweight PReF-based hybrid attestation framework termed PReFeR comprising a static and a dynamic attestation scheme, targetted for low-end embedded devices. The static scheme verifies the integrity of the prover's memory state by performing a pseudo-random walk on its main memory, where the next step is decided by the PReF response. The dynamic attestation verifies the code execution using PReF outputs and HPCs reflecting the footprint of the executed code.
- We employ PReF to couple the hardware characteristics of the embedded device with the attestation schemes, thereby thwarting man-in-the-middle, replay as well as runtime memory-based and TOCTOU attacks.
- We realize PReF using a Double Arbiter PUF and implement the attestation schemes on a lightweight embedded platform Digilent Cora Z7, comprising an FPGA and a microprocessor. The PReF is implemented on the Artix-7 FPGA and the software components of the attestation protocols are implemented on the ARM Cortex A9 processor.
- We analyse the security of the proposed schemes under realistic assumptions and present our evaluation results.

The remainder of the paper is organized as follows. Section 2 introduces the basic concepts required in this work. Section 3 briefly discusses some of the hybrid attestation schemes. In Section 4, we present a framework for PReF-based hybrid attestation, followed by a description of our proposed lightweight attestation protocols in Section 5. Section 6 presents the implementation details and experimental results of our schemes. Section 7 concludes the paper.

2 PRELIMINARIES

This section presents the necessary background on Physically Related Function, Fuzzy Extractor and Hardware Performance Counters.

2.1 Physically Related Function (PReF)

Physically Related Function (PReF) [16] is a cryptographic hardware primitive that realizes random functions in hardware, having similar functional behaviour over a subset of inputs. A pair of devices (D_A, D_B) embodying functions $f_A, f_B : \mathcal{X} \rightarrow \mathcal{Y}$ form a PReF if there exists a subset $\mathcal{X}_{A,B} \subseteq \mathcal{X}$ over which f_A and f_B generate similar outputs. For the remaining inputs, the outputs of f_A and f_B are uncorrelated. The distance between the outputs is computed using the Hamming Distance (HD) metric. Mathematically, for any $x \in \mathcal{X}_{A,B}$, $\text{HD}(f_A(x), f_B(x)) \leq \delta$ for a predefined distance δ and these inputs are called "related inputs". Note that although the probability of finding related inputs depends on the bit mismatch probability of the underlying construction, one can find related inputs even for devices with ideal uniqueness (50%). This is due to the fact that for every pair, there will exist a subset of inputs for which the outputs are similar (have almost δ HD), as has been validated theoretically in [16].

In [16], this construction is realized in hardware using 5-4 Double Arbiter PUF (DAPUF). It also proposes a mechanism to identify the related inputs using machine learning (ML) models and SAT solver. The idea is to construct accurate mathematical models of the pair of PUF instances using the interim outputs in the construction. We describe the steps briefly. A 5-4 DAPUF comprises 5 delay chains followed by 20 arbiters, where each arbiter takes a pair of delay paths emanating from two delay chains. The 20 outputs are combined using 4 XOR gates to produce a 4-bit response as depicted in Figure 6. In the setup phase, a designer has access to the 20 intermediate arbiter outputs, which are used to create accurate mathematical models using ML algorithms. The resultant model parameters are used to formulate the ML algorithm prediction logic as a satisfiability problem. In [16], the authors employ Logistic Regression (LR) as it produces accurate ML models and

the prediction logic (scalar product of weight vector and challenge parity vector) can be easily represented as a first-order mathematical equation, that is then fed to an SMT solver. The solver iteratively produces the challenges for which the predicted responses are within the predefined threshold δ. We refer the reader to [16] for more details. The construction has been employed to develop a lightweight node-to-node authentication protocol leveraging the 'related' functional behaviour.

A set of PReF devices are said to form a *PReF network* if each pair of devices is related over a disjoint set of *related* inputs. In a PReF network, the related inputs need to be unique, i.e., only one pair of PReF devices can produce similar outputs for a given related input. To obtain unique related inputs, the trusted third party removes the intersection of the related input sets between the new pair of devices and previously determined inputs. Note that, due to the large challenge set of a Strong PUF, even after removing the intersection of the related sets, each device has a significant number of unique related inputs. Conversely, this implies that a PReF network can comprise of a significantly large number of devices. The maximum number of devices that can be added to a PReF network can be determined by bounding the minimum number of unique related challenges required by the application. For details, we refer the reader to [16].

2.2 Fuzzy Extractor

A fuzzy extractor (FE) is a cryptographic primitive that is used to transform a noisy non-uniformly distributed output to a uniform key. It comprises two procedures described below:

- Generation procedure (**Gen**()): It takes as input a noisy non-uniform bit stream (r) and produces a uniform and random key (k) along with a helper data (h). Mathematically, $(k, h) \leftarrow FE.\mathbf{Gen}(r)$
- Reproduction procedure (**Rep**()): It takes a noisy bit stream (r') as input along with the helper data generated in the last step to reconstruct the key. Mathematically, $k \leftarrow FE.\mathbf{Rep}(r', h)$

2.3 Hardware Performance Counters

Hardware Performance Counters (HPCs) are special registers that measure the counts of low-level hardware events of any system during its operation. These registers are inbuilt into the processor and are primarily used to monitor the system performance; however, recently they have also been included in security applications such as malware detection [12]. Over the years, various malware detection mechanisms have been developed leveraging HPCs [27, 33, 35]. In [34], the authors proposed integrity checking of legitimate code using the HPC events of the system. In ARM Cortex-A9 processor, there are a set of 66 hardware performance events corresponding to different configurations [1]. These events correspond to low-level hardware operations executed by the processor, such as the number of instructions, number of reads and write cycles, cache reference, cache misses, number of branches, branch miss, bus cycle, etc. These event details are stored in special purpose registers such as program counter, current stack pointer, zero registers, and link register [2]. For our analysis we choose the events that exhibit significant variations upon code execution.

3 RELATED WORKS ON HYBRID REMOTE ATTESTATION

Hybrid attestation schemes are a class of remote attestation that achieves better security guarantee than software-based attestation while using less computational resources compared to hardware-based schemes. This class of protocols combine the physical properties of the resource-constrained modules such as PUFs with software attestation protocols, to mitigate the limitations of heavy cryptographic operations or expensive hardware modules [8, 9, 13, 31, 32, 37]. We categorize the related works into the following three categories on the basis of employed hardware primitives and attestation techniques:

PUF-based hybrid attestation schemes: One of the seminal works proposes a lightweight attestation protocol that binds the software checksum computation in the protocol with the response obtained from a PUF implemented in hardware [37], thereby binding the software computation with the hardware. In a subsequent work [32], a remote attestation protocol is proposed that involves a novel processor-based PUF construction built from ALUs present in modern processors. Recently another hybrid scheme involving PUFs, termed HAtt is proposed [8] targetting IoT devices. It attests the memory of the prover node in parts, thereby reducing the attestation time.

However, all the above-mentioned schemes require the storage of PUF-related database at the verifier end. In [9], the authors address this issue and propose an attestation scheme leveraging timing information for industrial IoT devices, which requires accurate time measurement of operations executed during the protocol. Subsequently, attestation schemes have been proposed that make extensive modifications to the hardware to provide software isolation [13, 31]. These protocols require extensive modifications to the underlying hardware to provide software isolation using a memory protection unit (MPU). In [25], the authors propose a physics-based attestation mechanism specific to an industrial control system, that combines software-based attestation with the physical process of the programmable control logic (PLC). Herein the verifier stores a database comprising the model and the expected set of reading for different commands given to the PLC, which incurs significant storage overhead.

Device Identifier Composition Engine (DICE)-based schemes: DICE [26] is a hardware-software architecture that comprises a hardware root-of-trust that is integrated with the target device to establish a chain of trust using a set of keys generated using classical symmetric key primitives in a sequential manner. The key generated in each layer represents the component identity and the integrity of the component. Based on this primitive, several attestation schemes have been developed targetted for IoT applications. However, these schemes also incur significant hardware and integration overhead. Moreover, DICE-based schemes have been recently shown to be vulnerable against TOCTOU attacks [28].

Control-Flow attestation schemes: While the above-mentioned schemes check the integrity of the binaries, it does not perform runtime integrity checks. This is handled by another class of attestation schemes known as control flow attestation that checks the integrity of code execution, thereby mitigating runtime attacks and time-of-check-time-of-use (TOCTOU) attacks. It involves the construction of a control-flow data structure at the verifier end during the set-up phase, which is used during attestation to validate the correct code execution flow [5]. However, it was subsequently shown to be vulnerable to TOCTOU attack [39], which was followed by the proposition of a hardware attestation system integrated with the processor that allows instruction-level access to the attestation, thereby allowing the detection of run-time attacks. In [19], a hardware-based control flow attestation scheme leveraging processor features is proposed. In [18], the author proposed a hardware-assisted tracking mechanism for control flow attestation, which monitors the control as well as the data flow during code execution, thereby preventing runtime attacks. However, these solutions incur significant implementation overhead and are difficult to integrate with low-cost embedded devices.

4 PREFER: PREF-BASED REMOTE ATTESTATION FRAMEWORK

In this section, we present PReFeR, a lightweight attestation framework targetting low-resource embedded devices. The overview of the attestation framework is depicted in Figure 1. The attestation framework comprises two components: static attestation protocol and dynamic attestation protocol. The static attestation protocol verifies the integrity of the software code in the prover device, after the device is deployed and prior to target code execution. On the other hand, the dynamic attestation code checks the integrity of the code execution during runtime. Both the schemes

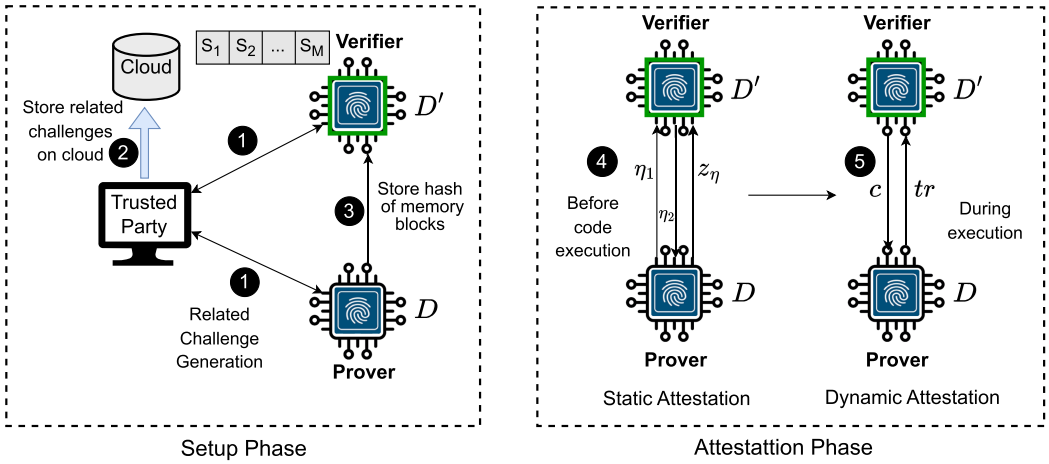


Fig. 1. Overview of PReFeR attestation framework.

employ a hardware-based security primitive called PReF (introduced in Section 2.1.) that binds the device characteristics to the attestation process.

System Model: We consider a network of interconnected lightweight devices ($\mathcal{N} = \{N_1, \ldots, N_M\}$) each of which embodies a PReF instance $D_i, \forall i \in [M]$, where M is number of devices connected in the network. These devices communicate with each other without the involvement of a resourceful third party or a central entity. In the set of nodes, some nodes are assumed to be trusted and are responsible for verifying the integrity of the memory content and code execution flow in the prover nodes. Such nodes are termed as verifiers. A verifier node can attest several prover nodes. Note that, at a subsequent point of time, a prover node can be delegated the task of verifying another subset of nodes in the network, as is common in collective attestation schemes [6]. Thus, both prover and verifier are assumed to be lightweight. In a given node, the attestation/verification code is assumed to be isolated from the target code.

Prior to deployment of the nodes in the network, as part of one-time setup phase, a trusted entity generates the related inputs corresponding to PReF devices embedded in each pair of prover-verifier nodes using the solver-based method proposed in [16]. For every verifier, the trusted entity computes disjoint related inputs corresponding to each prover by removing the inputs over which more than one prover node is related to the particular verifier. The disjoint related challenges are then uploaded to a public immutable cloud (ref. Figure 1). For ease of understanding, Figure 1 depicts the setup and attestation phase for a single pair of prover verifier nodes. The PReF instances corresponding to the prover and verifier device are denoted by D and D' respectively. Note that the trusted entity is required in the setup phase only. The feasibility and existence of such distinct related inputs are entailed in [16]. The prover node(s) are then deployed in-field after the initiation of the setup phase.

Adversary Model: An adversary can control the communication channel and eavesdrop, manipulate, and delay messages transmitted between the prover and verifier. It can access the prover devices remotely, and alter the software state and memory content of the devices by injecting malicious code. In this work, we assume that the adversary performs software-based attacks. An attempt at altering the prover's device hardware by means of invasive attacks to obtain the internal outputs of the hardware primitive, modifies the underlying characteristics of the device, thereby affecting the challenge-response behaviour of the PReF. The verifier is assumed to be trusted and secure. We assume the attestation codes are executed in a secure manner and cannot

be targetted by the attacker. Thus, main memory access by the attestation schemes are considered to be legitimate. An adversary can modify the memory state of the prover device after the target code gets loaded in the main memory. An adversary can perform runtime attacks on the target code execution. Interrupts in the attestation schemes are not considered in this work. We consider the adversary to be stealthy, i.e., its goal is to compromise the system whilst evading detection.

Security Properties: A PReF-based hybrid attestation scheme must achieve correctness and soundness properties. Correctness property ensures that an honest prover that has the same memory state as stored with the verifier and executes the intended target code should be successfully attested by the verifier. Soundness ensures that a prover having a memory state different from the original memory state or executing a code different from the target code should be successfully attested by the verifier with only negligible probability. Additionally, the PReF-based attestation scheme must ensure mutual authentication between the prover and verifier nodes.

5 PROPOSED ATTESTATION SCHEMES

The PReFeR framework comprises two hybrid attestation schemes, namely static and dynamic attestation. Both schemes require a one-time setup which is executed in a trusted and secure environment before node deployment. The steps involved in the setup phase of both schemes are described in their respective subsections.

5.1 Static Attestation Scheme

The static attestation comprises two phases:

Setup Phase: This phase is executed in a secure and trusted environment prior to the deployment of the prover node. Herein, a third party generates the related challenges for devices D and D' using the methodology described in [16] and uploads it in a publicly accessible cloud. Next, the verifier stores the hash of consecutive words of the prover's memory (termed as a memory block) $H(B_i)$ subsequently denoted as S_i along with the corresponding start index i. Here, an index i refers to the i^{th} block. Before the start of the attestation protocol, the prover downloads the related challenges and stores it in the local memory inbuilt with the prover node or secondary memory such as SD Card.

Attestation Phase: We describe the attestation scheme depicted in Figure 2 as follows.

(1) The attestation process is initiated by the prover that generates a nonce η_1 and sends it to the verifier.
(2) The verifier fetches the related challenge using η_1 as the index, computes the helper data h and the key k by applying the **Gen**() function on the PReF response. It also generates a nonce η_2. The verifier then concatenates the challenge, key and η_1, computes its hash using $\mathcal{H}$ and sends it along with η_2 and h to the prover node.
(3) The prover uses the nonce η_1 to get the first related challenge. Using the reproduce **Rep**() function of the fuzzy extractor it generates the key. It then computes the hash of the concatenation of the related challenge, key and nonce. The prover begins the attestation protocol only if this value is equal to the hash value sent by the verifier, implying that the verifier is authenticated.
(4) Once the verifier is authenticated, η_2 is used as the first memory block index. This ensures that in each run of the protocol, the prover starts with a randomly chosen memory block. Next, the prover fetches a related challenge x corresponding to the index i using **getRelChal**() function. It uses the **Gen**() function to produce a helper data h_j and key k_j for the PReF response $D(x)$. A one-way and collision-resistant hash function $\mathcal{H}_1$ is used to generate a hash of the concatenation of $\mathcal{H}(B_i)$, nonce η_2 and the key generated in the previous

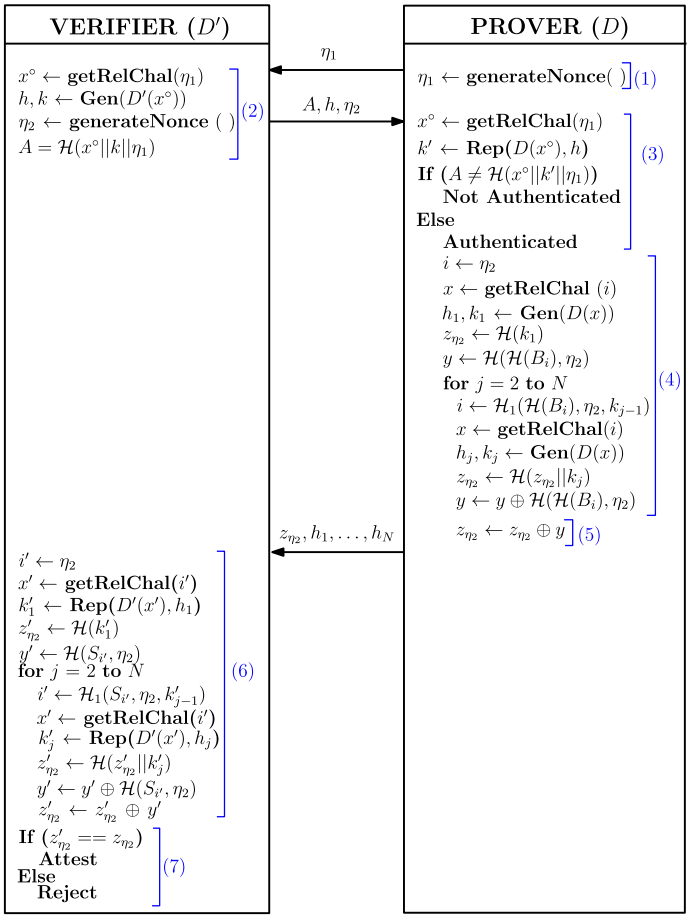


Fig. 2. Static remote attestation using PReF.

iteration k_{j-1}. The hash value is used as index to the next memory block. This process goes on for N iterations. Through the iterations, z_{η_2} stores the hash of previous z_{η_2} and the key k_j generated in the current iteration and y stores the XOR of the hash of every hashed memory block along with η_2.

(5) At the end, the prover sends the XOR of z_{η_2} and y along with the helper data $h_1, h_2, \ldots, h_N$ generated in all the iterations to the verifier. Note that the next memory block is dependent on the PReF response, which tightly integrates the hardware characteristics with the attestation process. The indices generated iteratively emulate a pseudorandom walk over the prover's memory state, as depicted in Figure 3.

(6) The verifier receives the values $z_{\eta_2}, h_1, h_2, \ldots, h_N$ from the prover. Next, it goes through N iterations, wherein it uses its PReF response $D'(x')$ as well as the helper data h_j to compute k'_j using the **Rep**() function.

(7) Using the previous key k'_{j-1}, the hashed values stored at the verifier end $S_{i'}$ and η_2, the verifier produces the next index i' with the same hash function $\mathcal{H}_1$ employed at the prover end. The verifier hashes all the reproduced keys k'_j iteratively and XORs it with hash $\mathcal{H}$ of $S_{i'}$ and the nonce η_2.

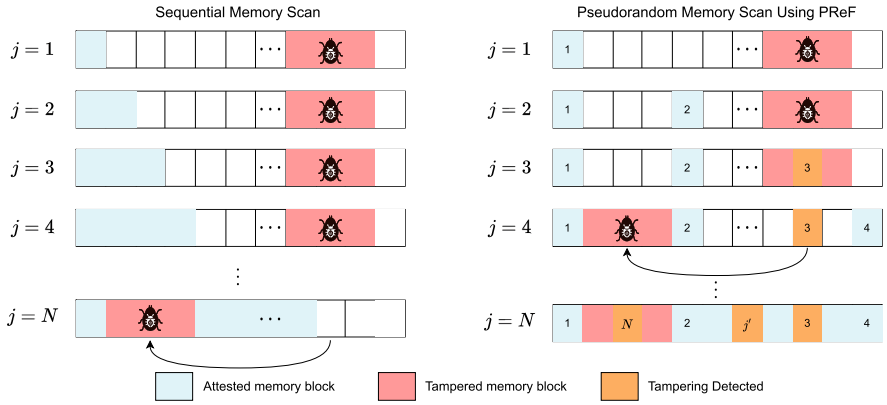


Fig. 3. Difference between sequential and pseudorandom memory scan for static attestation.

The verifier successfully attests the prover if the calculated value z'_{η_2} is equal to the value sent by the prover z_{η_2}. Else, it rejects the prover. This check ensures that the prover's current memory state has not been modified prior to the execution of the code.

The protocol is efficient in terms of computational complexity since it requires only PReF and hash operations and does not involve any heavy crypto-operations such as symmetric-key operations. To account for the unreliability of the hardware primitive PReF, we use the fuzzy extractor, which helps to correct the mismatches between the PReF responses.

The prover sends the nonce η_1 forcing the verifier to generate A for a previously unknown and fresh nonce. In the existing PUF-based protocols, the loss of CRPs from the verifier will require the prover nodes to go back to the setup phase, to remove the current PUF and embed a new PUF. However, for the PReF-based protocol, even if the verifier's PReF device is compromised, the functionality of the PReF on the prover device is still secure. The adversary cannot learn the prover's PReF functionality only using the related inputs. Thus, if a verifier is compromised, the security of all the nodes is not compromised and the attestation protocol can be instantiated with a different verifier embedded with another PReF device. In this case, the nodes will be notified with a new set of unique related challenges, which can be published on a cloud for the prover to download.

Security Analysis of the Static Attestation Scheme: This section discusses the security assessment of our PReF-based static attestation protocol.

- *Pseudo-random memory traversal*: In the proposed scheme, the next memory index i depends on the hash of nonce, the previous iteration key k_{j-1} generated from the PReF response, thereby ensuring a pseudo-random memory traversal. The pseudo-random memory traversal prevents an attacker to determine the next memory block that is to be attested. On the other hand, in sequential memory access where each block is accessed consecutively, an attacker can place the unintended/malicious code in the memory blocks that have already been attested. On the contrary, the random traversal ensures that the attacker cannot identify *safe* memory blocks with more than negligible probability as the formerly attested memory blocks can be revisited.
- *Resistant to replay and impersonation attacks:* In each execution of the static protocol, since the computation of y and the following memory block index depends on the nonce sent by the verifier, it ensures that an adversary cannot replay any stored values for y. The first message A sent by the verifier authenticates the verifier node as it is linked to the

intrinsic characteristics of the verifier PReF. The PReF also ensures the authenticity of the prover node by binding the hardware characteristics specific to a particular node in the computation of the next memory block index, thereby thwarting impersonation-based attacks. An adversary trying to impersonate using another remote device cannot succeed as it can not obtain the same PReF response as the verifier over the specified set of related challenges. Note that the static attestation protocol targets integrity of the main memory prior to code execution. Thus, it cannot directly deal with TOCTOU attacks, which are handled by dynamic attestation discussed in the next subsection.

Besides PReF, the scheme included hash functions and error correction codes (ECC). In our schemes, we employ a collision-resistant hash function. Thus, we do not consider attacks on hash functions such as collision or preimage attacks. In the case of ECCs, we address it by extending the length of the PReF response. This approach ensures that the amount of information or entropy leaked from the ECC is minimal, making it difficult for an attacker to deduce the key from the helper data during the reproduction phase.

Estimation of Number of Iterations of Static Protocol: Next, we estimate the number of iterations (N) that must be executed in the static attestation protocol to identify an attack on the prover's memory. Let us consider that the prover's memory is split into M blocks each consisting of n consecutive memory words. Let us assume that the words modified by an adversary during an attack corrupt Q memory blocks. The number of iterations should be calculated such that an attack can be identified with a probability of at least γ. Herein, we consider that each of the memory blocks verified during the static attestation is chosen randomly, as the index generated from the PReF response is pseudorandom.

Given these parameters, we compute the number of iterations as follows:

The probability that a randomly chosen memory block is corrupted is given by $\frac{Q}{M}$. Then the probability that no corrupted block is identified in any of the previous N iterations is given by $(1 - \frac{Q}{M})^N$. Thus the probability of identifying an attack is equal to the probability of finding a corrupted block in N iterations is given by $1 - (1 - \frac{Q}{M})^N \geq \gamma$.

From this, we obtain the minimum number of iterations to be $\log_{(1-\frac{Q}{M})}(1-\gamma)$. Assuming the memory to be split into 2^{16} blocks, and assuming γ to be 0.9. If an attacker corrupts 2^8 blocks, the number of iterations must be at least 589 to identify an attack with a probability of γ. Note that, even if the attacker corrupts a single word in the memory, the entire block is corrupted. In the worst case, if an attacker corrupts words in a single block of memory, i.e., $Q/M = 2^{-16}$, then the number of iterations in the static protocol needs to be at least 196326, which is significantly less for a main memory of size 512MB [4].

5.2 Dynamic Attestation Scheme

This scheme targets the execution of the target code residing in the prover's memory and validates the intended execution using HPCs that capture various instruction-level hardware events. Although the use of HPCs for malware detection in embedded devices has been studied intensively [12, 21, 30, 40], it has never been employed in remote attestation schemes. In this work, we present the first remote attestation protocol that leverages the granularity of HPC values to monitor the runtime-integrity of software code in resource-constrained devices, hence termed as *dynamic* attestation. The proposed protocol employs PReF responses to obfuscate the HPC measurement at the prover end. Since the responses are only known to the intended prover-verifier pair over their unique related inputs, only the intended verifier can decode the measurement using its PReF response. Since another third node, be it a verifier or a rogue node, does not share the

related inputs, they cannot obtain the HPC values. This step binds the device characteristics to the scheme, thereby authenticating the prover-verifier nodes.

The dynamic attestation protocol resides in the prover's memory and is initiated with the attestation token z_η obtained at the end of the static scheme. Thus the dynamic attestation scheme is executed only if the prover's memory state is attested. We describe the scheme as follows:

Setup Phase: In the setup phase, a set of T inputs $(c_1, c_2, \ldots, c_T)$ are fed to the target code residing in the prover device and their corresponding HPC traces $(HPC_{c_1}, HPC_{c_2}, \ldots, HPC_{c_T})$ are stored with the verifier stores the HPC traces obtained from the prover for a set of T inputs that are fed to the target code residing in the prover device. Note that the setup phase of the dynamic attestation scheme is executed after the setup phase of the static protocol in a secure and trusted environment. First, we present a dynamic attestation protocol that uses multiple HPC traces to verify the integrity of the target code.

Attestation Phase: Figure 4 depicts the dynamic attestation scheme executed at runtime. The steps of the first dynamic scheme are as follows:

(1) The verifier initiates the protocol by sending the T inputs of the target code $(c_1, c_2, \ldots, c_T)$ to the prover.
(2) The prover node executes the target code (stored in its main memory) with the received inputs and collects traces $(tr_1, tr_2, \ldots, tr_T)$ where each trace comprises a set of HPC values.[2]
(3) The key concatenation z_η established at the end of the static scheme is used as a seed to a Linear Feedback Shift Register (LFSR) to generate indices $\{i_1, i_2, \ldots, i_k\}$. These indices are used to access related challenges. Note that, the use of LFSR to index related challenges reduces the communication overhead.
(4) The hardware performance signature $(tr_1, tr_2, \ldots, tr_T)$ is encoded using an error correction code (ECC) and XORed with the PReF responses corresponding to the related challenges selected in the previous step.
(5) The verifier retrieves the previously stored traces $(tr'_1, tr'_2, \ldots, tr'_T)$ corresponding to the chosen inputs, generated by the prover in the setup phase.
(6) Next, it generates a concatenation of related responses y' to decode the output received from the prover. Since the dynamic signature depends on hardware performance statics during run-time, there is a chance of a mismatch between the traces of the same code running at the prover and the verifier.
(7) We employ statistical tests to compute the distance between the two traces and accepts if the distance is less than a pre-defined threshold value τ. Else, it rejects the node.

However, the scheme has the limitation of using several HPC traces to determine the attestation outcome. A prover has to run the target code several times with all the inputs provided by the verifier. To address this drawback, we propose another dynamic attestation scheme that works with a single trace. For this scheme, the prover executes the correct code in the setup phase of T inputs. The mean (μ_C) and variation (σ_C) corresponding to these T traces are then stored by the verifier. Additionally, the verifier also stores the mean (μ_M) and variation (σ_M) corresponding to a set of malicious/unintended codes.[3] We describe the process of generating malicious traces in Section 6.3. Note that the verifier requires only the mean and variance and need not to store the individual traces. Thus, choosing a large value of T does not increase the memory overhead. The steps depicted in Figure 5 are described as follows:

[2]To ensure that a potentially malicious code does not impact the prover device, the target code is executed in a sandbox.
[3]For low-end embedded devices, the target code is assumed to be fixed as it is used for a particular set of operations. In this context, any code execution deviant from the pre-defined device functionality is considered to be malicious.

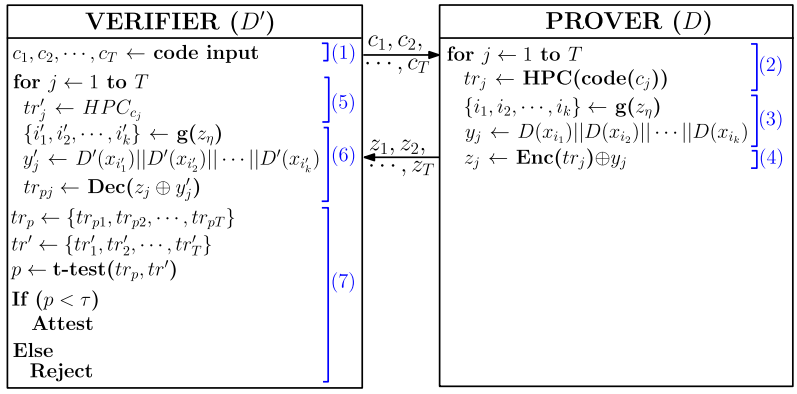


Fig. 4. Dynamic remote attestation using PReF.

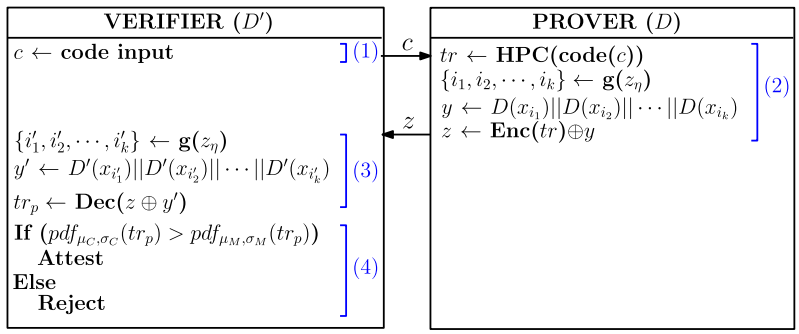


Fig. 5. Dynamic remote attestation using PReF and a single HPC trace.

(1) The attestation phase is initiated by the verifier by sending a single input c to the prover.
(2) The prover executes the target code with the input and generates a single trace tr comprising the chosen HPC values. It encodes the trace and XORes with the PReF responses y. The related challenges corresponding to the PReF responses are generated using an LFSR g that uses the token z_η generated at the end of the static protocol as seed.
(3) Upon receiving z from the prover, the verifier generates related challenges using z_η as seed and uses their corresponding responses to decode z and obtain tr_p.
(4) Next it computes the likelihood of the trace belonging to the correct or malicious distributions.[4] The verifier accepts if the probability of the trace lying in the correct distribution is higher than that of the malicious distribution for all the HPC values under consideration. Else, it rejects the node.

Note that the choice of HPCs is crucial to this application, as has been reported in previous works on malware detection. We elaborate on the choice of HPCs in the following section. The details of the performance counters used in the protocols are mentioned in Table 1.

Security Analysis of the Dynamic Attestation Scheme: In this section, we discuss the security analysis of the two dynamic attestation schemes.

[4]In Figure 5, pdf_C and pdf_M denote the probability density function of the correct and malicious distribution respectively.

Table 1. Hardware Performance Event Details

Hardware Performance Event	Event Description
XPM_EVENT_INSRFETCH_CACHEREFILL	Instruction fetch that causes a refill at the lowest level of instruction cache or unified cache.
XPM_EVENT_DATA_CACHEACCESS	Data read or write operation that causes a refill at the lowest level of data or unified cache.
XPM_EVENT_DATA_READS	Data read instructions accepted by Load Store Unit including speculative and aborted Load Register (LDR)/Load Multiple Register (LDM) instructions.
XPM_EVENT_DATA_WRITE	Data write instructions accepted by Load Store Unit including speculative and aborted Store Register (STR)/Store Multiple Register (STM) instructions.
XPM_EVENT_SW_CHANGEPC	Changes in Program Counter (PC) during execution, excluding the PC changes due to exceptions
XPM_EVENT_IMMEDBRANCH	Immediate branch executed by processor internally (taken or not taken).
XPM_EVENT_BRANCHMISS	The number of mispredicted or not-predicted branches.
XPM_EVENT_CLOCKCYCLES	Clock cycles counter for Cortex-A9 processor that are not in Wait for Event (WFE)/Wait for Interrupt (WFI).
XPM_EVENT_BRANCHPREDICT	Branches or other changes in program flow that could have been predicted by the branch prediction resources of the processor.
XPM_EVENT_NODISPATCH	The number of cycles where the issue stage does not dispatch any instruction because it is empty or cannot dispatch any instructions.
XPM_EVENT_ISSUEEMPTY	Number of cycles where the issue stage is empty.
XPM_EVENT_INSTRRENAME	Instructions going through the register renaming stage.
XPM_EVENT_MAINEXEC	Instructions executed in the main execution pipeline of the processor, the multiply pipeline and Arithmetic Logic Unit (ALU) pipeline.
XPM_EVENT_SECEXEC	Instructions executed in the processor second execution pipeline (ALU).
XPM_EVENT_LDRSTR	Instructions executed in the Load/Store Unit.
XPM_EVENT_DE_CLKEN	Number of clock cycles during which the data engine clock is enabled.

- *Resistant to TOCTOU attacks*: We assess the dynamic protocol against runtime based attacks and time-of-check-time-of-use (TOCTOU) attacks. In TOCTOU or runtime attacks, an adversary targets the execution flow of the target code. A malicious code changes the program memory of the prover and reverts it to its previous states between two attestations while the prover node has no information about it. As the PReF-based dynamic attestation starts with the final static attestation report (z_{η_2}), it verifies the absence of malicious code in the prover memory prior to target code execution. TOCTOU attack is launched using techniques such as return-oriented programming that try to alter the control flow of the code execution by using additional conditional statements. Since the HPCs monitor instruction level events including branch instructions and provide runtime statistics of code execution, they can detect changes in the execution flow of the code in the prover device due to any run-time attacks.
- *Resistant to replay and impersonation attack*: Since the prover is a resource-constrained device, with limited memory, the possibility of an attacker replaying previously-stored trace values is negligible. In the case of the first protocol, the attacker needs to store the traces corresponding to T inputs where each trace comprises of multiple HPC values, which requires significant storage overhead. In the case of the second protocol, since the code input is sent by the verifier, the adversary comprising the prover node is unaware of the input. In order to perform a replay attack, it needs to keep track of all the code inputs and their corresponding traces collected over several sessions which also incurs significant memory overhead. The adversary cannot perform an impersonation attack as the device characteristics of the prover node are bound to the attestation scheme by virtue of PReF.

- *Resistant to spoofing attack*: In spoofing attack, an adversary tries to tamper the HPC values transmitted to the verifier in order to evade an illegitimate code execution. Since the HPC values are transmitted after obfuscating with the PReF response, it is computationally infeasible for an eavesdroppping adversary to determine the actual HPC values without the PReF response. As per our adversary model, the adversary can only attack device software. Thus, it cannot determine the performance counters transmitted, neither can it spoof the values.

Comparison with PUF-based Attestation Schemes: PUF-based attestation schemes require secure storage of large challenge-response database or mathematical models at the verifier end. On the contrary, in PReF-based schemes, PReF forms the hardware root-of-trust at both the prover and verifier end, which eliminates the requirement for secure storage. The related challenges for a pair of prover-verifier devices can be stored in a public immutable ledger. Note that the responses need not be stored at the verifier end. Additionally, PReF allows the replacement of the verifier - in case it is compromised - without the prover nodes being revoked. This step involves the computation of related challenges corresponding to the new verifier and prover nodes which are to be updated in the public ledger. While computing the related challenges, the trusted party removes the related challenges that intersect with the related challenges corresponding to the compromised verifier. This ensures that a compromised node cannot impersonate a legitimate verifier. Since the challenge response dataset of a PReF is exponential in the input length, even after removing the intersecting challenges, a new verifier has a significant number of related inputs. Since the challenges are public, the loss of a verifier node does not compromise the system.

While the PReF-based attestation scheme may seem similar to PUFatt [32], we would like to highlight the following key differences. (i) The underlying primitives in these two protocols are different. PReFeR is the first PReF-based attestation scheme that eliminates the critical drawbacks of PUF-based attestation schemes such as secure storage of challenge response database, limited number of attestations and lack of scalability due to bounded storage. (ii) PUFatt relies on time check mechanism which is hard to scale in multihop networks such as IoT. PReFeR was developed to mitigate this limitation and rely solely on the hardware primitive. (iii) While PUFatt leverages classical cryptographic primitives such as stream cipher RC4 for generating address for random memory access, PReFeR relies on the psuedorandomness of the PReF response for the next address computation, thereby eliminating use of classical random number generators.

Comparison with DICE-based schemes: Comparing the DICE scheme to the proposed schemes, in the static attestation scheme, the PReF forms the hardware root-of-trust which is used to generate a series of tokens depicting the memory state in an iterative manner. The final token comprises the integrity information of all the previously traversed memory locations. Furthermore, all the proposed PReF-based schemes are lightweight compared to DICE-based schemes. While DICE employs symmetric key primitives and PRNG, the proposed schemes use hash and error-correcting codes.

Scalability of the proposed schemes: As per our system model, a verifier can attest several prover devices. Thus for a given verifier, no two provers should share a related input. Thus, on increasing the number of prover nodes, the number of related inputs for each prover-verifier pair reduces. This implies that the scalability of PReFeR depends on the number of PReF instances (M) that can be connected in a mesh topology. The relationship between the maximum permissible network size and minimum number of related inputs for each pair of nodes in the network is given by $\frac{1}{2}(\frac{\log \epsilon - \log \epsilon'}{\log(\frac{1}{1-\epsilon})})$ where ϵ and ϵ' denote the fraction of related and disjoint-related challenges respectively [16]. In static attestation, the number of related inputs is equal to the number of iterations. Thus, considering the calculation of number of iterations from Section 5.1, we consider

$N \geq 196326$ for a memory of size $512MB$. Thus, assuming each pair of prover verifier nodes require approximately 196326 challenges the maximum size of the network of devices can be 2^{25}. We refer the reader to Section IV.F in [16] for the derivation of the network size. This proves that the PReFeR framework is scalable to IoT scenarios.

6 IMPLEMENTATION AND EXPERIMENTAL RESULTS

In this section, we describe the implementation setup followed by experimental evaluation of the PReFeR attestation framework.

6.1 Implementation Setup

The implementation setup comprises two low-end embedded devices, corresponding to a prover and a verifier. For proof-of-concept implementation, we consider a single prover and a verifier as the attestation scheme is executed between a pair of nodes at a time. This can be extended to multiple devices as well, with an additional step of computing the disjoint related inputs. In this scenario, the verifier node stores the related inputs for each of the prover nodes. We use a Digilent Cora Z7 board that integrates a dual-core 667MHz ARM Cortex-A9 processor with a Xilinx Artix-7 FPGA and 512 MB DDR3 RAM. Note that the implementation does not assume any aspect specific to the chosen board and this can be implemented on any embedded platform with a microprocessor (to run the software components of the attestation framework and the target code) and an FPGA (to implement the PReF construction). The PReF construction proposed in [16] is implemented in Verilog language on Artix-7 FPGA using Xilinx Vivado 2021.2 tool. The attestation schemes are implemented on bare metal in C language and are executed on the ARM processor using the Xilinx Vitis tool. The main memory is partitioned into two parts: one for the execution of the attestation schemes and the other for the target code that is supposed to be executed at the prover end. The verifier device uses an additional secondary memory (SD Card) to store the hash of the prover memory blocks. We employ a trusted party to compute the related challenges for the PReFs embedded in the prover and verifier node in the setup phase. The related challenges are stored in a publically accessible cloud. Note that this step is performed only once for a pair of PReFs. The prover device downloads the related challenges and stores in the secondary memory in order to reduce the communication overhead with the cloud. The number of related challenges required by a prover node for one execution of the attestation protocols is equal to the number of iterations which depends on the size of the memory to be attested. Since the first index of the related challenge is determined by the nonce generated by the prover, in each execution of the static protocol, the first chosen related input will be different. This in turn will lead to a different pseudorandom memory traversal depending on the subsequent PReF responses. This crucial aspect of the protocol allows a prover node to reuse the set of related inputs for multiple sessions. In our implementation, we consider 512MB of the DDR4 memory that includes data and code blocks. Note that, this can be extended to a generic memory layout assuming that specific memory regions like memory-mapped I/O, etc can be accessed using standard instructions and explicit OS utilities (like `mmap` in the context of Linux).

6.2 PReF Implementation Details and Results

Before proceeding to the experimental evaluation of the PReFeR framework, we present the implementation details of PReF on Artix-7 FPGA.

- *Implementation details of PReF*: We follow the implementation details mentioned in [16] and realize PReF using 5-4 Double Arbiter PUF (DAPUF). The PUF takes a 64-bit challenge as input and produces a 4-bit response. It comprises 5 delay chains followed by 20 arbiters

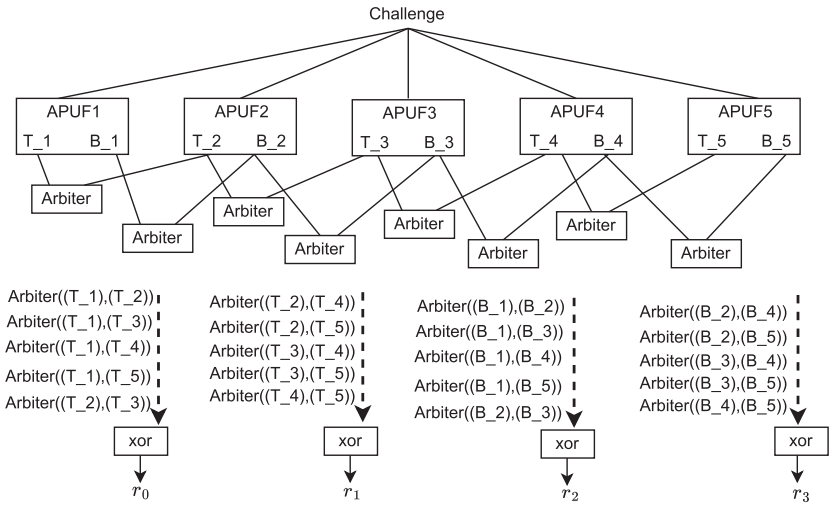


Fig. 6. Schematic Representation of 5-4 Double Arbiter PUF.

whose outputs are combined using 4 XOR gates to produce a 4-bit response as depicted in Figure 6. To generate the mathematical models required to compute the related challenges, we output the internal bits emanating from the arbiters in the construction along with the final response. These outputs are required only during the setup phase and can be fused prior to deployment. To determine the related inputs, we model individual arbiter outputs using LR, as it is known to produce the most accurate models for APUFs [36]. The modelling accuracy of individual arbiter outputs surpasses 95% for a training set of $20K$ randomly chosen CRPs. These are fed to the Z3 solver to compute related challenges using the steps described in Section 2.1 (for more details refer to Algorithm 1 [16]). We use these related challenges in the protocols discussed in the following sections.

- *Robustness of PReF against ML-based modeling attacks*: We characterize the PReF construction using $50K$ randomly generated challenges responses pairs (CRPs) obtained from 4 Cora Z7 boards. The uniformity of the 4 response bits is 43.70%, 58.34%, 35.66% and 56.47% respectively. The PReF has a uniqueness of 34.99% and the bitwise reliability values are 94.83%, 98.56%, 96.85% and 98.34% respectively. For reliability calculation, we compute the reference response by performing majority voting over 5 measurements of each response. We also investigate the modeling resilience of PReF using classical ML algorithms like Logistic regression (LR), Support Vector machine (SVM) and Random Forest (RF). We implement the algorithms in Python3 using the `scikit – learn` library. The training and test dataset comprise $400K$ and $100K$ CRPs respectively chosen uniformly at random. We apply 5-fold cross-validation over the training dataset to prevent overfitting. The bitwise modelling accuracy of LR, SVM with Linear kernel, SVM with Polynomial kernel, and RF is 58.88%, 53.45%, 56.26% and 52.50% respectively, averaged over all 4 bits.

6.3 PReFeR Implementation Details and Results

In this section, we present the details of our proof-of-concept implementation details of the proposed attestation schemes and the evaluation results.

6.3.1 Static Attestation Scheme. The static attestation scheme involves accessing main memory content which is performed using standard function calls (**`Xil_In32`**() function that reads the

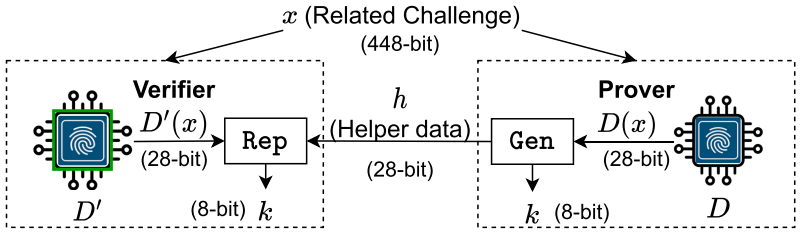


Fig. 7. Generate and Reproduce functions of Fuzzy Extractor adapted to PReF. The key k corresponds to only one iteration in the static attestation scheme.

content of one 32-bit block) present in Xilinx IO library. The order of block accesses is determined by the response obtained from the PReF implemented on the FPGA. The communication between the processor and the FPGA is established using AXI GPIO modules. We use SHA-256 from Libgcrypt library to implement all the hash functions in the scheme.

In our proof-of-concept implementation, each memory block index (depicted by i in Figure 2) has a length of 32-bits. Thus to compute the index i, we use H_1 that returns 32-bits from the least significant bit (LSB) of the SHA3 output. Each iteration involves hash computation of 2^{10} consecutive memory words (comprising one block) using the function H. The hash of the memory block $H(B_i)$ is used to retrieve the next related challenge as well as for the computation of y. The variable y stores the hash of all memory blocks along with the nonce η_2. Consecutively, z_{η_2} stores the keys generated from the FE. At the end of the iterations, the prover sends z_{η_2} which has a length of 256-bits. The prover sends z_{η_2} along with the helper data generated in each iteration to the verifier. At the verifier's end, it compares the 256-bit string z_{η_2} with z'_{η_2} computed using the previously stored hash values and the PReF responses. For FE implementation, we employ a $(7, 4, 1)$-BCH scheme. Since the codeword is 7-bits long and the PReF returns a 4-bit response, we concatenate the responses for 7 related challenges to generate a 28-bit string that is fed to the **Gen**() function as shown in Figure 7. Note that the key (k) corresponds to one iteration of memory access. The same procedure is executed for N iterations, where N can be computed as described in Section 5.1. To verify that the indices generated in each iteration are random, we perform autocorrelation over a list of indices [23]. We obtain the autocorrelation coefficient to be close to zero, indicating a negligible correlation between the consecutively generated indices, thereby corroborating with the pseudorandom walk model. This also ensures that an adversary cannot guess the next index solely by observing the previously generated indices. We run the protocol 1000 times with different nonce values and obtain a false positive rate close to zero, as an incorrect index calculation in any of the iterations deviates the memory traversal from the intended memory block.

6.3.2 Dynamic Attestation Scheme. For evaluation of the dynamic attestation schemes, the target code and the attestation code are executed in an interleaved manner in the prover device. In the setup phase, the verifier device stores the HPC traces obtained from the prover node for a set of T code inputs. For each code input, the prover provides one HPC trace. A trace comprises 16 HPC values, corresponding to each hardware performance event. The number of code inputs (T) has to be large enough so that each performance counter can capture the behaviour of correct code execution. For our proof-of-concept implementation, we use 1000 code inputs. The prover sets the hardware events that it wants to monitor during the execution of the target code using the **Xpm_SetEvents**() function. Next, it runs the target code with the input provided by the verifier and accumulates the event counts using **Xpm_GetEventCounters**() function. In our proof-of-concept

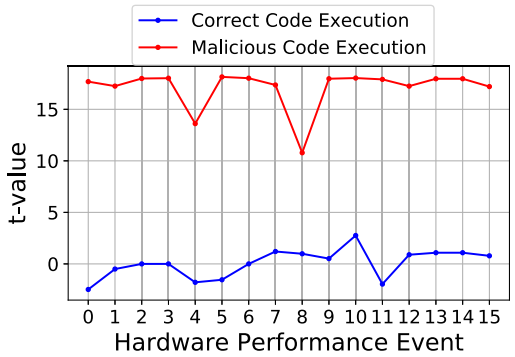


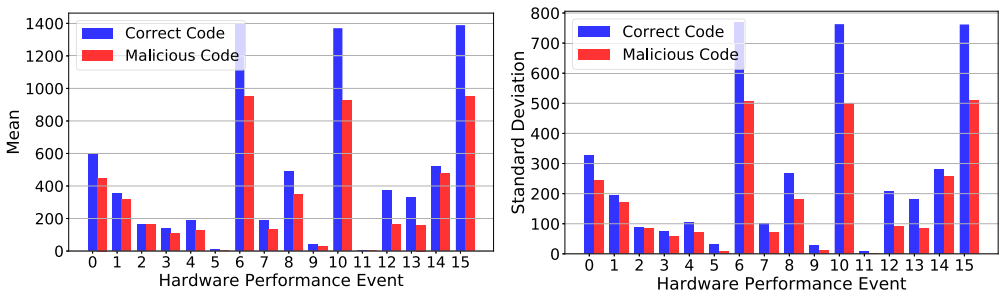
Fig. 8. Correct and Malicious code execution in Prover 1 and Verifier 1.

implementation, we implement the prover code and verifier code on the same architecture however that is not necessary as the verifier does not perform any architecture-specific operations. This also validates that the operations executed at the prover end are lightweight and thus are suitable for resource-constrained embedded devices. Since the verifier compares the previously stored traces with real-time traces obtained from a legitimate prover, the t-value is less as depicted by the blue line in Figure 8.

Note that during attestation the target code can be executed in a sandbox to ensure the execution of a potentially malicious code does not impact the prover device. There are several sandboxes that can be easily integrated with the ARM Cortex-A9 processor [3]. The ARM Cortex A9 processor provides a suite of 66 HPCs [2]. We choose only those counter values which reflect significant changes during the code execution. The considered counters capture hardware events such as cache-access, cache-miss, data read and write operations, number of branch instructions and number of load-fetch instructions which are impacted during the code execution. The details of the considered hardware performance events are given in Table 1.

To validate the first dynamic scheme, we perform the following experiment. First, at the prover end, we collect the HPC values $\{tr_1, tr_2, \ldots, tr_{16}\}$ corresponding to the correct target code when executed with 100 different test input values. Here each tr_i comprises a set of 1000 HPC values one for each input. At the verifier end, we retrieve the HPC dataset $\{tr'_1, tr'_2, \ldots, tr'_{16}\}$ for the same chosen code inputs where each tr'_i comprises of 1000 HPC values. Next, we compute the paired t-test between tr and tr' for each of the 16 HPC values. We repeat this experiment for a malicious code (with only one additional if statement compared to the correct code). Note that for low-end embedded devices, since the device operates a fixed number of operations, the target code executed on the node is assumed to be fixed. Thus any code execution deviant from the legitimate code is considered to be malicious. To test the proposed method, we generate new *malicious* codes by introducing minimal changes in the target code (such as the addition of an if statement compared to the correct code or changes in constant variables) so that it remains undetected in the statistical tests. We execute the dynamic attestation scheme using these codes. Figure 8 depicts the t-values for correct and malicious code execution for each of the hardware events, thereby establishing the distinguishability of malicious code execution from legitimate code. However, this scheme has the drawback that a prover node needs to run the target code 1000 times before the target code can be attested.

To mitigate this limitation, we propose another dynamic attestation scheme that helps to determine the runtime integrity of the target code using a single trace. Herein, the verifier sends a single test input to the prover. The prover computes the HPC trace $\{tr_1, tr_2, \ldots, tr_{16}\}$ corresponding to that particular execution and sends it to the verifier after encoding and XORing with the PReF



(a) Mean (b) Standard Deviation

Fig. 9. Mean and Standard deviation of HPC values computed over $10K$ traces.

Table 2. Performance of Dynamic Attestation Protocol using Single Trace from Prover

		Prover	
		Correct Trace	**Incorrect Trace**
Verifier	**Correct Distribution**	94.4%	0%
	Malicious Distribution	5.6%	100%

response. In both the dynamic protocols, the related challenges used to query the PReF are obtained from an LFSR, which uses the token generated at the end of the static protocol as the Initialization Vector (IV). The use of an LFSR helps in reducing the communication cost of the dynamic protocol.

During the setup phase of the protocol, the verifier computes the mean and variance of each of the 16 HPC values corresponding to correct code execution as well as malicious code execution. The mean and standard deviation of each HPC event for correct and malicious code execution is given in Figure 9. The set of hardware performance events considered for this scheme is tabulated in Table 1. We observe a difference in the mean and standard deviation of the HPC values, where the deviation between the means is significant for events such as XPM_EVENT_BRANCHMISS, XPM_EVENT_ISSUEEMPTY, XPM_EVENT_DE_CLK EN.

In the attestation phase, the verifier computes the probability of the trace lying in either of the two distributions by comparing the probability that a given HPC value lies in the correct or malicious distribution. The verifier attests a code only if the probability of a value lying in the correct distribution is higher for all the 16 HP events. We report the true positive (probability that a correct execution trace lies in the correct trace distribution), true negative (probability that a malicious execution trace follows the malicious distribution), false positive (probability that a malicious trace lies in the correct trace distribution) and false negative (probability that a correct trace lies in the malicious trace distribution) values in Table 2. We observe that this protocol identifies a correct trace with a probability of 0.944 (true positive rate) and a malicious trace with a probability of 1 (true negative rate), thereby depicting the accuracy of the dynamic protocol.

6.3.3 Performance Comparison with Existing Hybrid Attestation Schemes. Next, we present a comparison of PReFeR with existing lightweight hybrid attestation schemes in terms of communication overhead, and memory and hardware resource utilization.

- Communication Overhead: The communication overhead of our PReFeR protocol compared with [8, 9, 32] are shown in Table 3. For one round of attestation processes, we send 28-bit helper data and 256-bit for the final attestation report. In PUFatt [32] have the lowest communication overhead with only two message exchanges, but it suffers from a reliance on

Table 3. Comparison of Hybrid Attestation Schemes

Attestation Scheme	Hardware Platform	Features					Performance Metrics	
		Lightweight Operations	No Time Check	Interrupt-free Execution	Requires Expensive Hardware	Device Availability	Hardware and Memory Overhead	Number of Messages Transferred
SMART [20] (2012)	TI MSP430 and Atmel AVR Microcontroller	Yes	Yes	Yes	No	No	42 KB(AVR), 46 KB(MSP430)	2
PUFatt [32] (2014)	Virtex-5 FPGA and Atmel AVR Microcontroller	Yes	No	Yes	No	No	9207 LUTs, 2921 Registers	2
TrustLite [31] (2014)	Intel Siskiyou Peak(Xilinx Virtex-6 FPGA)	Yes	Yes	No	No	No	14361 LUTs, 5528 Registers	4
TyTAN [13] (2015)	Intel Siskiyou Peak(Xilinx Spartan-6 FPGA)	Yes	Yes	No	No	No	244 KB(TyTAN's OS)	4
ATT-auth [9] (2018)	Atmel Atmega Microcontroller	Yes	No	Yes	No	Yes	Not Mentioned	2
SMARM [15] (2018)	ARM MX6-SabreLite board	No	Yes	No	No	No	256 MB	4
HAtt [8] (2020)	Raspberry Pi & Arduino Mega	Yes	Yes	Yes	No	Yes	412 KB(ATMega2560)	4
PReFeR (this work)	**Digilent Cora Z7(Xilinx Artix-7 FPGA)**	**Yes**	**Yes**	**Yes**	**No**	**Yes**	1442 **LUTs,** 1351 **Registers**	3 **(Static)** 2 **(Dynamic)**

a time-out timer (TOT). It needs to be completed within a specified time constraint, which is an unrealistic assumption in practice. While, HAtt [8] protocol requires four message exchanges, the static and dynamic attestation protocols of PReFeR requires only 3 and 2 message transfers respectively.

- Resource Overhead: Finally, we compare the resource consumption of the proposed schemes with respect to hardware and memory overheads as shown in Table 3. We specifically compared the attested memory regions of SMARM [15], TyTan[13], and HAtt [8] with our proposed scheme called PReFeR. During runtime, PReFeR attests a larger memory size of 512MB, while SMARM, TyTan, and HAtt attest smaller memory sizes of 256MB, 244KB, and 412KB respectively. In terms of hardware design, PUFatt [32] required a significant number of resources with 9207 LUTs and 2921 registers. TrustLite [31] also had a high resource requirement, utilizing 14361 LUTs and 5528 registers. In contrast, our proposed PReFeR scheme demonstrated superior efficiency with only 1442 LUTs and 1351 registers, showcasing a substantial reduction compared to the baseline architecture. PUFatt [32] relies on strong architectural support, specifically the synchronization of two ALU cores, in order to behave as PUF. But in real-time scenarios where the Verifier and Prover are engaged in resource-heavy tasks such as control signal generation, resource allocation, sensing, and actuation, the ALU cores may not be synchronized so the attestation cannot be guaranteed to operate effectively.

7 CONCLUSION

The development of secure lightweight remote attestation protocols has been a prevalent research problem. Although several remote attestation schemes have been proposed in the literature, they are accompanied by limitations of heavy computational or storage overhead. PUF-based protocols do not address the issue of adversaries attacking the verifier to gain knowledge of the secure storage contents (CRP database of PUFs). Additionally, they assume that the attestation request is originating from an authentic verifier and do not have a mechanism to verify its authenticity. In this work, we address these issues and propose PReFeR-a hybrid remote attestation framework based on a new cryptographic primitive called PReF. The framework comprises two attestation schemes: a static protocol that validates the static memory content of the prover node before code execution, and a dynamic scheme that verifies correct execution flow using HPCs. We also propose an improved version of the dynamic scheme that attests the prover node accurately using HPC values of only one code execution. We implement PReF on Artix-7 FPGA and validate both schemes on Digilent Cora Z7 platform. Finally, we experimentally validate the performance of both the protocols.

REFERENCES

[1] 2017. Xilinx Performance Counter. https://github.com/Xilinx/embeddedsw/blob/master/lib/bsp/standalone/src/arm/cortexa9/xpm_counter.h

[2] 2019. Arm Cortex A9 Special Purpose Register. https://developer.arm.com/documentation/dui0473/m/overview-of-the-arm-architecture/arm-registers

[3] 2021. OpenOCD Sandbox. https://github.com/tarek-bochkati/openocd
[4] 2023. Cora Z7 Programmable Logic Reference Manual. https://digilent.com/reference/programmable-logic/cora-z7/reference-manual
[5] Tigist Abera, N. Asokan, Lucas Davi, Jan-Erik Ekberg, Thomas Nyman, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. 2016. C-FLAT: Control-flow attestation for embedded systems software. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 743–754.
[6] 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*.
[7] Manaar Alam, Debdeep Mukhopadhyay, Sai Praveen Kadiyala, Siew-Kei Lam, and Thambipillai Srikanthan. 2020. Improving accuracy of HPC-based malware classification for embedded platforms using gradient descent optimization. *J. Cryptogr. Eng.* 10, 4 (2020), 289–303. https://doi.org/10.1007/s13389-020-00232-9
[8] Muhammad Naveed Aman, Mohamed Haroon Basheer, Siddhant Dash, Jun Wen Wong, Jia Xu, Hoon Wei Lim, and Biplab Sikdar. 2020. HAtt: Hybrid remote attestation for the Internet of Things with high availability. *IEEE Internet of Things Journal* 7, 8 (2020), 7220–7233.
[9] Muhammad Naveed Aman and Biplab Sikdar. 2018. ATT-Auth: A hybrid protocol for industrial IoT attestation with authentication. *IEEE Internet of Things Journal* 5, 6 (2018), 5119–5131.
[10] Will Arthur, David Challener, and Kenneth Goldman. 2015. *A Practical Guide to TPM 2.0: Using the New Trusted Platform Module in the New Age of Security*. Springer Nature.
[11] Alexander Sprogø Banks, Marek Kisiel, and Philip Korsholm. 2021. Remote attestation: A literature review. *arXiv preprint arXiv:2105.02466* (2021).
[12] Kanad Basu, Prashanth Krishnamurthy, Farshad Khorrami, and Ramesh Karri. 2019. A theoretical study of hardware performance counters-based malware detection. *IEEE Transactions on Information Forensics and Security* 15 (2019), 512–525.
[13] 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*. 1–6.
[14] 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*. ACM, 91:1–91:6. https://doi.org/10.1145/2897937.2898083
[15] Xavier Carpent, Norrathep Rattanavipanon, and Gene Tsudik. 2018. Remote attestation of IoT devices via SMARM: Shuffled measurements against roving malware. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST'18)*. IEEE, 9–16.
[16] Durba Chatterjee, Harishma Boyapally, Sikhar Patranabis, Urbi Chatterjee, Aritra Hazra, and Debdeep Mukhopadhyay. 2022. Physically related functions: Exploiting related inputs of PUFs for authenticated-key exchange. *IEEE Transactions on Information Forensics and Security* (2022).
[17] 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 CCCS*. 2317–2331.
[18] Ghada Dessouky, Tigist Abera, Ahmad Ibrahim, and Ahmad-Reza Sadeghi. 2018. Litehax: Lightweight hardware-assisted attestation of program execution. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'18)*. IEEE, 1–8.
[19] Ghada Dessouky, Shaza Zeitouni, Thomas Nyman, Andrew Paverd, Lucas Davi, Patrick Koeberl, N. Asokan, and Ahmad-Reza Sadeghi. 2017. LO-FAT: Low-overhead control flow attestation in hardware. In *Proceedings of the 54th Annual Design Automation Conference 2017*. 1–6.
[20] Karim Eldefrawy, Gene Tsudik, Aurélien Francillon, and Daniele Perito. 2012. Smart: Secure and minimal architecture for (establishing dynamic) root of trust. In *Ndss*, Vol. 12. 1–15.
[21] Rana Elnaggar, Kanad Basu, Krishnendu Chakrabarty, and Ramesh Karri. 2021. Runtime malware detection using embedded trace buffers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 1 (2021), 35–48.
[22] Nicolas Falliere, Liam O. Murchu, and Eric Chien. 2011. W32. stuxnet dossier. *White Paper, Symantec Corp., Security Response* 5, 6 (2011), 29.
[23] Emil Faure, Iryna Myronets, and Artem Lavdanskyi. 2020. Autocorrelation criterion for quality assessment of random number sequences. In *CMIS*. 675–689.
[24] Blaise Gassend, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. 2002. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 148–160.
[25] 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'19)*. 165–180.
[26] Trusted Computing Group. 2018. https://trustedcomputinggroup.org/work-groups/dice-architectures/

[27] Zhangying He, Tahereh Miari, Hosein Mohammadi Makrani, Mehrdad Aliasgari, Houman Homayoun, and Hossein Sayadi. 2021. When machine learning meets hardware cybersecurity: Delving into accurate zero-day malware detection. In *2021 22nd International Symposium on Quality Electronic Design (ISQED'21)*. IEEE, 85–90.

[28] Stefan Hristozov, Moritz Wettermann, and Manuel Huber. 2022. A TOCTOU attack on DICE attestation. In *Proceedings of the Twelveth ACM Conference on Data and Application Security and Privacy*. 226–235.

[29] Sai Praveen Kadiyala, Manaar Alam, Yash Shrivastava, Sikhar Patranabis, Muhamed Fauzi Bin Abbas, Arnab Kumar Biswas, Debdeep Mukhopadhyay, and Thambipillai Srikanthan. 2020. LAMBDA: Lightweight assessment of malware for emBeddeD architectures. *ACM Trans. Embed. Comput. Syst.* 19, 4 (2020), 23:1–23:31. https://doi.org/10.1145/3390855

[30] Sai Praveen Kadiyala, Pranav Jadhav, Siew-Kei Lam, and Thambipillai Srikanthan. 2020. Hardware performance counter-based fine-grained malware detection. *ACM Transactions on Embedded Computing Systems (TECS)* 19, 5 (2020), 1–17.

[31] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. 2014. TrustLite: A security architecture for tiny embedded devices. In *Proceedings of the Ninth European Conference on Computer Systems*. 1–14.

[32] Joonho Kong, Farinaz Koushanfar, Praveen K. Pendyala, Ahmad-Reza Sadeghi, and Christian Wachsmann. 2014. PUFatt: Embedded platform attestation based on novel processor-based PUFs. In *DAC 2014*. IEEE, 1–6.

[33] Charalambos Konstantinou, Xueyang Wang, Prashanth Krishnamurthy, Farshad Khorrami, Michail Maniatakos, and Ramesh Karri. 2022. HPC-based malware detectors actually work: Transition to practice after a decade of research. *IEEE Design & Test* (2022).

[34] Corey Malone, Mohamed Zahran, and Ramesh Karri. 2011. Are hardware performance counters a cost effective way for integrity checking of programs. In *Proceedings of the sixth ACM Workshop on Scalable Trusted Computing*. 71–76.

[35] Nisarg Patel, Avesta Sasan, and Houman Homayoun. 2017. Analyzing hardware based malware detectors. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC'17)*. IEEE, 1–6.

[36] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. 2010. Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*. 237–249.

[37] Steffen Schulz, Ahmad-Reza Sadeghi, and Christian Wachsmann. 2011. Short paper: Lightweight remote attestation using physical functions. In *Proceedings of the fourth ACM Conference on Wireless Network Security*. 109–114.

[38] 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.* IEEE, 272–282.

[39] Shaza Zeitouni, Ghada Dessouky, Orlando Arias, Dean Sullivan, Ahmad Ibrahim, Yier Jin, and Ahmad-Reza Sadeghi. 2017. Atrium: Runtime attestation resilient under memory attacks. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'17)*. IEEE, 384–391.

[40] Boyou Zhou, Anmol Gupta, Rasoul Jahanshahi, Manuel Egele, and Ajay Joshi. 2018. Hardware performance counters can detect malware: Myth or fact?. In *Proceedings of the 2018 on AsiaCCS*. 457–468.

Received 23 March 2023; revised 2 June 2023; accepted 13 July 2023