The $\mathcal{S}$A^{4}P Framework: Sensing and Actuation as a Privilege

Piet De Vaere[*]
Felix Stöger[*]
piet@devae.re
felix.stoeger@inf.ethz.com
ETH Zurich
Switzerland

Adrian Perrig
adrian.perrig@inf.ethz.ch
ETH Zurich
Switzerland

Gene Tsudik
gts@ics.uci.edu
University of California, Irvine
USA

ABSTRACT

Popular consumer Internet of Things (IoT) devices provide increasingly diverse sensing and actuation capabilities. Despite their benefits, such devices prompt numerous security concerns. Typically, security is attained at device-level granularity, which relies upon device trustworthiness. However, if a device is compromised (e.g., via remote exploits), this approach fails. To this end, we construct $\mathcal{S}$A^{4}P: Sensing and Actuation as a Privilege, a framework that decouples IoT devices from their physical environment. In $\mathcal{S}$A^{4}P, whenever any software on a device wants to access a sensing or actuation peripheral, it must be authorized to do so. This is achieved by the inclusion of an on-board component, Peripheral Guard ($\mathcal{P}$EG), that physically guards peripherals. Besides providing strong security guarantees, $\mathcal{S}$A^{4}P motivates developers to consider sensing and actuation as valuable resources. $\mathcal{S}$A^{4}P' design is modular, lightweight, and formally verified. It also does not require any hardware modifications for trusted execution environment (TEE)-equipped devices, while imposing only modest changes for other devices.

CCS CONCEPTS

• **Security and privacy → Access control**; **Embedded systems security**; *Distributed systems security.*

KEYWORDS

Distributed Reference Monitor, IoT, sub-device level access control

ACM Reference Format:
Piet De Vaere, Felix Stöger, Adrian Perrig, and Gene Tsudik. 2024. The $\mathcal{S}$A^{4}P Framework: Sensing and Actuation as a Privilege. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3634737.3657006

1 INTRODUCTION

Specialized embedded devices of various sorts and purposes increasingly permeate numerous aspects of everyday life. Commonly referred to as IoT devices, these gadgets differ from general-purpose computers (servers, desktops, laptops, tablets, and smartphones), since their primary purpose is to provide one or more service(s) that involve sensing and/or controlling (via actuation) various aspects of the physical (analog) world.

Depending on its type and purpose(s), an IoT device can pose certain risks, e.g., privacy risks for leaked sensed data, security risks for tampered sensed data, and safety risks for malicious actuation commands. These risks are not imagined or over-hyped; they are quite real, as evidenced by a continuous stream of real-world incidents and research results [7, 19, 22, 29]. Some issues are intuitive, e.g., privacy leaks from audio and video sampling devices, or safety concerns with garage/gate/door-lock controllers. Whereas, the situation with simpler gadgets is more subtle, e.g., a compromised humidity sensor (or a motion-activated light fixture) can leak information about room occupancy [20].

Aforementioned risks are caused by device interactions with the physical environment, i.e., sensing and/or actuation. Hence, by controlling (and minimizing) device's access to the physical environment, risks caused by that device can be also minimized. Therefore, this work aims to achieve fine-grained control over all device interactions with the physical world. In other words, the goal is to turn a device's sensing and actuation abilities into carefully managed privileges. This is in contrast with the currently dominant approach where access to sensors and actuators is enforced at device-level granularity, meaning that the entire device is treated as a monolithic entity access to which is controlled. This works as long as the device is benign, i.e., fully trusted. However, if a device is compromised (via an exploit or backdoor), this approach becomes ineffective, since the adversary is now inside the device, free to sense and actuate at will.

A more robust approach is to perform access control at the device component level. Prior work includes SeCloak [30] and a proposal by Brasser et al. [11]. These schemes enable/disable access for extended periods of time and focus on powerful mobile devices, such as smartphones. In contrast, we focus on fine-grained access control, at the level of a single peripheral access. Also, since IoT devices are often deployed as a system, access control policies can be imposed and enforced collectively, rather than on each device individually. One example of a system-based approach is ContextIot [52], a context-aware access control system for apps on IoT cloud platforms. Since ContextIoT operates at the cloud-platform level, its trusted computing base (TCB) includes IoT devices as well as the entire cloud platform.

This paper constructs $\mathcal{S}$A^{4}P: Sensing and Actuation as a Privilege, an IoT framework that allows deployment-wide access control at the device component level. As illustrated in Fig. 1, $\mathcal{S}$A^{4}P places

[*]Both authors contributed equally to this research.

a Peripheral Guard ($\mathcal{P}$EG) between each device's software runtime ($\mathcal{R}$untime) and its sensing and actuating peripherals, thereby decoupling the former from the physical environment.

Each $\mathcal{P}$EG is managed by the *deployment manager*, a trusted entity that makes all access control decisions for a given IoT deployment, e.g., a household or office. The deployment manager can run on a remote or a local server, e.g., home router, controller, or owner's smartphone. Together with $\mathcal{P}$EGs, the deployment manager effectively constitutes a distributed reference monitor wherein a centralized entity (deployment manager) controls all physical-world interactions occurring across an entire IoT deployment.

By default, $\mathcal{P}$EG disallows all sensor and actuator access. Whenever any software on a device wants to interact with the physical environment, it must first send a request to the deployment manager. The main benefit of this approach is its preventative nature: data that is not sampled cannot be leaked, and attempted (yet not performed) actuation causes no harm. In addition to allowing the deployment manager to make fine-grained access control decisions, this approach also enables centralized, deployment-wide, logging and auditing of all interactions with the physical world.

This centralized control enables security mechanisms, which are currently limited to individual devices, to be consistently applied on a deployment-wide scale. For example, 6thsense [52], which monitors sensor access patterns to detect adversarial activity, could be extended to observe cross-deployment sensor access activity. $\mathcal{S}$A^{4}P also supports work on automated IoT privacy assistants [15], by providing a robust and efficient mechanism to collect sensor activity information and enforce user privacy policies.

To facilitate $\mathcal{S}$A^{4}P's ease of deployment, $\mathcal{P}$EG is lightweight and modular. The former facilitates deployment in constrained environments, while the latter allows $\mathcal{P}$EG to be efficiently adapted to individual settings. Moreover, its lightweight structure and modularity facilitate formal verification.

On devices with TEE support (e.g., ARM TrustZone), $\mathcal{P}$EG can be instantiated with no additional hardware requirements. On simpler devices, a stand-alone $\mathcal{P}$EG component can be added. We provide implementations for both device types. Combining $\mathcal{S}$A^{4}P with orthogonal mechanisms like traffic monitoring and remote attestation can further increase the level of security.

The main contributions of this work are as follows:

- $\mathcal{S}$A^{4}P, an IoT security framework that includes a distributed reference monitor, isolating computational resources from the physical environment, thus turning sensing and actuation into a *privilege.*
- Design and formal verification of Peripheral Guard ($\mathcal{P}$EG), the key building block in $\mathcal{S}$A^{4}P deployment.
- Implementation and evaluation of two $\mathcal{P}$EG instantiations: one using a dedicated microcontroller unit (MCU), and another using ARM TrustZone on Cortex-M.

All verification and implementation code is available at: https://github.com/SA4P.

2 PRELIMINARIES

Devices The term "IoT" refers to a wide variety of devices, ranging from tiny sensors/actuators to powerful industrial controllers. In this paper, we only consider devices that interact with the physical environment. We exclude purely computational or general-purpose devices, e.g., smartphones. Also, although we aim for compatibility with as many device classes as possible, some are too constrained for our purposes. We use RFC 7228 [10] to define a set of minimum requirements for targeted devices. RFC 7228 lists six types of constraints: code complexity (i.e., flash memory size), size of state (i.e., size of RAM), processing power, user interfaces, connectivity, and available electrical energy. In this work, we consider all devices that can: execute a simple state machine; store tens of bytes of persistent state; perform basic cryptographic operations; capture user input (e.g., through a button press); communicate with a central entity on a regular basis; and have enough energy to support the demands listed above. The required level of security and the available energy dictate the frequency of communication with the central entity. We expect frequencies of multiple times per second to once every few days. Examples of targeted device include: smart speakers, motion sensors, CO_2 sensors, door locks, and household appliances.

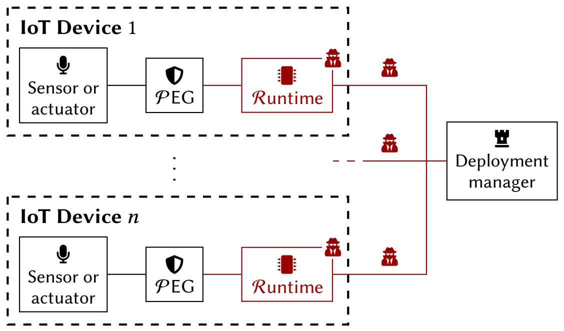


Figure 1: $\mathcal{S}$A^{4}P **architecture.** 🕵 **indicates an untrusted component or link.**

Deployment context Although IoT devices are deployed in many diverse settings, we focus on private or semi-private ones, such as: (i) private homes, (ii) office spaces, and (iii) hotel rooms and short-term private rentals, e.g., AirBnB In such spaces, users have expectations of personal security and privacy. For example, a homeowner might want to be assured that security cameras are disabled when they are home, or that an adversary cannot turn on the stove when they are sleeping. Similarly, participants in a confidential meeting in a hotel room want to ensure that the audio or video system is not snooping.

Deployment manager We assume that each deployment has a trusted *deployment manager* – a logical entity that runs on a local or remote server, e.g., on an existing home gateway. Devices can be paired with the deployment manager. Acting on behalf of the primary user of the deployment space (e.g., homeowner, renter, or hotel guest), the deployment manager evaluates access policies. The details of the deployment manager are out of scope of this paper. We refer to prior work on IoT policy management [48] and privacy assistants [15, 28].

3 TRUST & ADVERSARY MODEL

Considering targeted deployment settings, the adversary's goal is to access a sensor or an actuator without the deployment manager's permission.

The $\mathcal{S}$A^{4}P trust model considers IoT device manufacturers to be trusted, while all code running on IoT devices is untrusted. We refer to this code – which includes both application(s) and OS/hypervisor – as **software runtime** or just $\mathcal{R}$untime. Its untrusted status is motivated by (i) a myriad of vulnerabilities seen in IoT devices [14, 49], and (ii) widespread use of third-party software [64]. Nonetheless, we assume that the manufacturer can introduce a (small) trusted component into the device, either in the form of additional hardware for TEE-less devices, or software for TEE-equipped devices. Given its limited footprint, this component is assumed to be trusted.

We consider two types of adversaries:

Base adversary: Controls all network traffic to/from a device, on all of its interfaces. It may be local and/or remote. It can arbitrarily record, drop (including jam), modify, and insert packets. Moreover, it has control over the device's application code, obtained either through an exploit, or through infection at production or deployment/provisioning time. However, it does not have physical access to the device.

Non-invasive physical adversary: Beyond capabilities of the base adversary, this adversary has physical access to the device. It can use the device's standard interfaces, e.g., press buttons, turn it of/off, read any external markings, and use any wired (e.g., USB) interfaces. This adversary models malicious visitors to the private space. We do not consider physically invasive adversaries, as they could simply install their own malicious sensors or actuators.

Attacks against the TEEs (if present) or the code running therein are out of scope. We also do not consider attacks against the deployment manager, since, we expect it to be secure and not resource-constrained.

Our high-level security goals are:

SEC-1 A base adversary cannot access device sensing/actuation peripherals unless when permitted by the deployment manager.

SEC-2 A non-invasive physical adversary cannot access device sensing/actuation peripherals without being detected.

4 $\mathcal{S}$A^{4}P OVERVIEW

A $\mathcal{S}$A^{4}P deployment consists of one or more $\mathcal{S}$A^{4}P-enabled IoT devices and a *deployment manager*. When a device enrolls in a deployment, its $\mathcal{R}$untime is decoupled from the physical environment, i.e., by default, devices cannot access their sensors or actuators. Upon request, the deployment manager *may* grant a device temporary access to its sensing or actuation peripherals. The manager keeps a log of all granted and denied access requests. For conciseness, we refer to sensors and actuators collectively as *interaction peripherals*, or, depending on context, simply as *peripherals*.

The key enabler of $\mathcal{S}$A^{4}P is the Peripheral Guard or $\mathcal{P}$EG, an architectural component that enables the deployment manager to enforce peripheral access policies within a device, even if that device is compromised. Concretely, $\mathcal{P}$EGis placed between interaction peripherals and $\mathcal{R}$untime, as Fig. 1 depicts. $\mathcal{P}$EG responds to commands from the deployment manager, allowing the latter to enforce access control policies. Although $\mathcal{S}$A^{4}P does not require synchronized clocks, $\mathcal{P}$EG needs to have a notion of elapsed time.

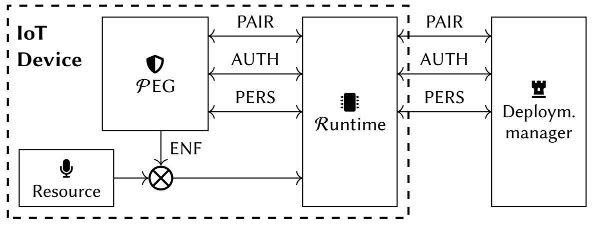


Figure 2: An overview of the $\mathcal{P}$EG architecture.

To make $\mathcal{S}$A^{4}P widely applicable, the $\mathcal{P}$EG design must be *inclusive*, i.e., compatible with as many device classes as possible. This is especially important for highly constrained devices. $\mathcal{S}$A^{4}P achieves inclusiveness by applying two design strategies: complexity reduction and functional modularization.

Complexity reduction A central goal of the design is to minimize complexity of $\mathcal{P}$EG. This allows it to operate even in highly challenging environments and on constrained hardware platforms. Lower complexity also simplifies formal verification, and a small TCB ensures auditability. This is especially important since IoT devices, once deployed, might never experience a firmware update.

Functional Modularization Given the wide variety of IoT devices, no single $\mathcal{P}$EG design can cover all use-cases. Therefore, we identify four core $\mathcal{P}$EG system functions. By specifying the interface between these functions, we can implement them in separate and independent *modules* which can be individually tailored to the requirements of a specific deployment scenario.

5 $\mathcal{P}$EG DESIGN

We now overview four $\mathcal{P}$EG modules, their functionalities, and interactions. We then discuss them in more detail and provide concrete designs. Fig. 2 illustrates the high-level architecture of a $\mathcal{P}$EG-enabled device, and the functionality of four $\mathcal{P}$EG modules.

First and foremost, we need a mechanism to associate $\mathcal{P}$EG (and its host device) with the deployment manager. This mechanism is implemented by the *pairing* module (PAIR, Section 5.1), which sets up an association in the form of shared cryptographic keying material.

Then, the deployment manager can receive access requests from a $\mathcal{P}$EG-enabled device, and grant or deny access based on its policies. The format and behavior of access requests and access grants are defined by the *authorization* module (AUTH, Section 5.2). As shown in Fig. 2, communication between $\mathcal{P}$EG and the deployment manager is proxied by $\mathcal{R}$untime, reducing $\mathcal{P}$EG complexity and TCB size. However, since $\mathcal{P}$EG traffic is exposed to both the untrusted $\mathcal{R}$untime and the network, it must provide integrity and origin authenticity. For that, AUTH relies on the keys earlier established by PAIR.

An access grant tells $\mathcal{P}$EG about allowing access to protected peripherals. It is then the *enforcement* module's (ENF, Section 5.3) responsibility to enforce this policy.

Finally, the *persistence* module (PERS, Section 5.4) ensures that devices cannot be stealthily removed from $\mathcal{S}$A^{4}P deployment. As discussed in Section 5.4, PERS is especially important with respect to physical adversaries.

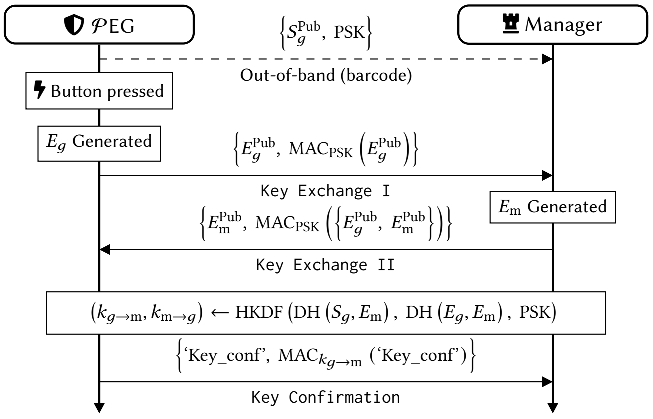

$\mathcal{P}$EG
Manager
$\left\{S_g^{\text{Pub}},\ \text{PSK}\right\}$
Out-of-band (barcode)
Button pressed
E_g Generated
$\left\{E_g^{\text{Pub}},\ \text{MAC}_{\text{PSK}}\left(E_g^{\text{Pub}}\right)\right\}$
Key Exchange I
E_{m} Generated
$\left\{E_{\text{m}}^{\text{Pub}},\ \text{MAC}_{\text{PSK}}\left(\left\{E_g^{\text{Pub}},\ E_{\text{m}}^{\text{Pub}}\right\}\right)\right\}$
Key Exchange II
$(k_{g\to\text{m}}, k_{\text{m}\to g}) \leftarrow \text{HKDF}\left(\text{DH}\left(S_g, E_{\text{m}}\right),\ \text{DH}\left(E_g, E_{\text{m}}\right),\ \text{PSK}\right)$
$\left\{\text{'Key\_conf'},\ \text{MAC}_{k_{g\to\text{m}}}\left(\text{'Key\_conf'}\right)\right\}$
Key Confirmation

Figure 3: The SIPA handshake.

5.1 Pairing Module

PAIR is responsible for establishing shared keying material between $\mathcal{P}$EG and its the deployment manager.

Because of the wide variety of IoT devices, no single pairing scheme can satisfy the requirements of each deployment setting. Nonetheless, for the sake of completeness, we design and implement *semi-identified, PSK-aided authentication (SIPA)*, a 3-message pairing protocol inspired by the Noise framework [46]. Since SIPA is applicable to even highly constrained settings, it is applicable to most deployment scenarios. We present SIPA below, and assume its use for the remainder of this paper.

SIPA is illustrated in Fig. 3. In order to participate in a SIPA pairing, each $\mathcal{P}$EG must be provisioned with (i) a static public-private key-pair, and (ii) a pre-shared key (PSK) at the time of manufacturing. These keys must be stored in non-volatile, read-only memory (ROM). Each device containing a $\mathcal{P}$EG, must have both the public and pre-shared keys of its $\mathcal{P}$EG printed or etched on the outside of the device, e.g., using a 2D barcode. Additionally, each device must have a button or similar interface that can be used to instruct the $\mathcal{P}$EG to enter the pairing mode.

A SIPA handshake is executed as follows:

(1) The user, acting on behalf of the deployment manager scans (e.g., via QR code) public and pre-shared keys of the device.
(2) The user pushes the $\mathcal{P}$EG button of the device. This enables the $\mathcal{P}$EG pairing mode.
(3) The $\mathcal{P}$EG and the deployment manager perform a Noise protocol-inspired handshake as follows:
(4) (a) Both $\mathcal{P}$EG and the deployment manager generate ephemeral key-pairs and exchange public keys.
    (b) They perform a Diffie–Hellman key exchange for each of the two key-pairs (static and ephemeral) of $\mathcal{P}$EG, and the ephemeral keys of the deployment manager.
    (c) The result of these Diffie-Hellmann operations and PSK, are used as input to the HKDF key derivation function [26], which yields two uni-directional session keys.
(5) If the above succeeds, $\mathcal{P}$EG erases all old pairing information and adopts new keys. It then sends a key confirmation message to the deployment manager.
(6) Upon receipt of the key confirmation message, the deployment manager adopts new keys.

The SIPA pairing module provides the following properties:

PAIR-1: $\mathcal{P}$EG authentication Whenever the deployment manager believes to have established a session key with a $\mathcal{P}$EG G, $\mathcal{P}$EG G established the same session key. Hence, $\mathcal{P}$EG is authenticated to the deployment manager and, by extension, to the user. This ensures, even while pairing over wireless interfaces, that $\mathcal{P}$EG which performed the pairing is, in fact, sam $\mathcal{P}$EG in the scanned device.

PAIR-2: Weak manager authentication Whenever $\mathcal{P}$EG G establishes a key with the deployment manager M, the latter (or its delegate) must have earlier scanned the data inscribed on the device in which G is present. This is accomplished by mixing PSK in the handshake, thereby verifying that the party performing the pairing had, at some point, physical access to the device.

PAIR-3: Key secrecy Whenever the deployment manager establishes a key with a benign $\mathcal{P}$EG, that key is not known to the adversary.

PAIR-4: Verification of intent No new keys can be established without a time-adjacent physical interaction. This ensures that an entity with current physical device access intends to pair the device.

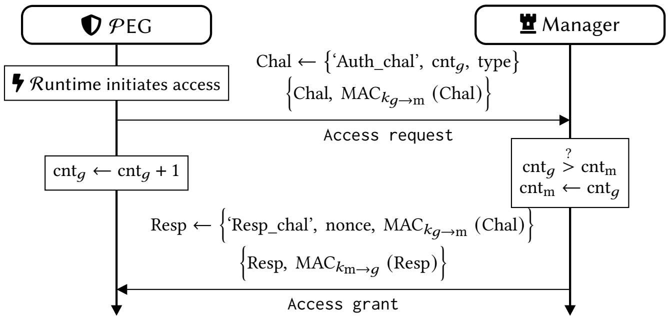
As its name suggests, SIPA does not fully authenticate both parties involved in the pairing process: while $\mathcal{P}$EG is strongly authenticated using its long-term private key, the deployment manager is only weakly authenticated using PSK. The motivation behind this asymmetry is twofold. First, there is the practical information limit on $\mathcal{P}$EG: given its restricted user interfaces, providing $\mathcal{P}$EG with the identity of the deployment manager is cumbersome and unnecessary. Second, there is no need for stronger authentication since, at the end of a SIPA handshake, the deployment manager, acting on behalf of the user, is assured about the identity of $\mathcal{P}$EG. Since $\mathcal{P}$EG adopts the new keying material *before* sending out the key confirmation message, it already established a session with the deployment manager. This one-sided knowledge is sufficient to satisfy required high-level security properties.

5.2 Authorization Module

After shared keying material is established, $\mathcal{P}$EG must be able to receive instructions from the deployment manager about when to allow or restrict access to the protected peripheral. This functionality is provided by the authorization module.

$\mathcal{P}$EG uses a two-message challenge-response protocol illustrated in Fig. 4. When access to a protected peripheral is requested by $\mathcal{R}$untime, $\mathcal{P}$EG generates an authenticated challenge consisting of a counter, access type, and MAC computed using the session key over the first two fields. The counter achieves replay protection, while access type allows one $\mathcal{P}$EG to protect multiple peripherals. This challenge is then sent to $\mathcal{R}$untime, which forwards it to the deployment manager.

Upon receipt, the deployment manager checks the counter value against previous values, evaluates the request against its policy, and, if access is granted, returns an authenticated response. The latter consists of a random nonce, the original authentication tag of



$\mathcal{P}$EG
Manager
$\mathcal{R}$untime initiates access
Chal ← {'Auth_chal', cnt$_g$, type}
{Chal, MAC$_{k_{g\to m}}$ (Chal)}
Access request
cnt$_g$ ← cnt$_g$ + 1
cnt$_g \overset{?}{>}$ cnt$_m$
cnt$_m$ ← cnt$_g$
Resp ← {'Resp_chal', nonce, MAC$_{k_{g\to m}}$ (Chal)}
{Resp, MAC$_{k_{m\to g}}$ (Resp)}
Access grant

Figure 4: The authorization protocol flow.

the challenge, and a new MAC computed using the session key over the first two fields. The nonce prevents the manager from being used as a MAC oracle.

When $\mathcal{P}$EG receives a valid response within a fixed time T_{chal}, it grants access to the protected peripheral for T_{auth}. Both T_{chal} and T_{auth} are configurable parameters. T_{chal} can be different for each protected peripherals. Multiple access types can be associated with the same peripheral, each with a different T_{auth} value.

The authorization module provides the following properties:

AUTH-1: Bounded-window authorization It is guaranteed that access to a protected peripheral is only granted within well-defined time windows. Each time window starts when the deployment manager transmits a grant and ends at most $T_{\text{chal}} + T_{\text{auth}}$ time units later.

AUTH-2: Bounded-duration authorization The protected peripheral can be accessed for at most T_{auth} time units per granted access.

5.3 Enforcement Module

The enforcement module implements low-level access control to peripherals. Because of the wide variety of peripheral interfaces, different authorization modules must be used for different types of interfaces. We provide implementation examples for three types:

Open collector (Fig. 5a) Communication over open collector-based interfaces can be interrupted by pulling the signal lines to ground potential using transistors. $\mathcal{P}$EG forces all signal lines to ground, unless peripheral access is granted.

Push-Pull (Fig. 5b) Unlike open collector type interfaces, forcing signal lines of push-pull-based interfaces to ground may cause physical damage. Instead, tri-state digital buffer elements can be used to decouple signal lines on both sides of the interface.

Analog (Fig. 5c) Digital buffers cannot be used for analog interfaces. However, similar functionality can be achieved using an operational amplifier in a non-inverting configuration. A transistor in the feedback circuit can be used to clamp the analog output to the positive supply bus voltage, inhibiting communication. Alternatively, an analog buffer could be used.

Regardless of the implementation, an enforcement module should provide the following property:

ENF-1: Peripheral isolation The enforcement module ensures that $\mathcal{P}$EG can isolate the protected peripheral from $\mathcal{R}$untime. Specifically, the enforcement module can prevent $\mathcal{R}$untime from interacting with the protected peripheral.

5.4 Persistence Module

Although the pairing module guarantees the pairing state of $\mathcal{P}$EG at the time of the pairing handshake, it does not prevent $\mathcal{P}$EG from being re-associated[1] with another the deployment manager at a later time. However, when combined with the pairing module's verification of intent (i.e., the requirement for the user to press a button), this suffices to meet our security goals for the base adversary (**SEC-1**).

To contend with a non-invasive physical adversary, additional guarantees are needed. Such an adversary could press the pairing button and re-associate the device with a malicious the deployment manager. The current the deployment manager would not learn this, leading to a violation of security goal **SEC-2**.

The persistence module mitigates this attack by providing guarantees about the continued pairing state of a $\mathcal{P}$EG. For example, the persistence module could require the current the deployment manager's approval before $\mathcal{P}$EG can be unpaired. Although desirable, this approach is fragile since it hinges upon availability/reachability of current $\mathcal{P}$EG. We favor an approach based on liveness, shown in Fig. 6. In it, the deployment manager periodically sends an authenticated nonce to $\mathcal{P}$EG, which re-authenticates to the former. Although depicted as a stand-alone protocol in Fig. 6, its messages can be piggybacked onto the messages of the authentication protocol.

The persistence protocol provides the following properties:

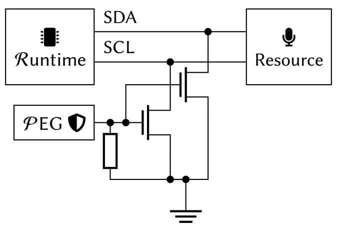
PERS-1: Retroactive proof of pairing state Whenever the deployment manager M receives nonce N from $\mathcal{P}$EG G, it is confirmed that $\mathcal{P}$EG G was paired with M when nonce N was transmitted by M.

Thus, this protocol provides retroactive confirmation of pairing state. Moreover, when this protocol is periodically repeated, lack of response from $\mathcal{P}$EG indicates that an un-pairing event occurred. This means that devices that leave a given $\mathcal{S}$A^{4}P deployment are detected within a bounded amount of time P – the protocol's periodicity. Furthermore, when $\mathcal{P}$EG disallows un-pairing for a period $P' > P$ after each received persistence challenge, the deployment manager can detect an unresponsive device before any interactions with the physical world can occur. Thus, even stronger properties are attained, albeit at the cost of reduced usability.

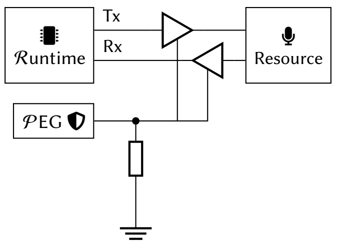
6 SECURITY ANALYSIS

We leverage a combination of formal and circuit analysis methods to analyze security of $\mathcal{P}$EG modules. Formal methods are used to analyze protocol-based modules, and circuit analysis is used to evaluate enforcement modules. All Tamarin models are available on GitHub [9]. We also evaluate $\mathcal{S}$A^{4}P against reference monitor NIST criteria [36].

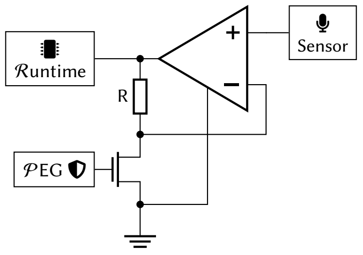
[1]For clarity's sake, we use the term *re-associated* rather than *re-paired.*



(a) Open collector (e.g., I2C).



(b) Push-Pull (e.g., UART).



(c) Analog voltage (e.g., microphone).

Figure 5: Example enforcement module implementations for different signal types.

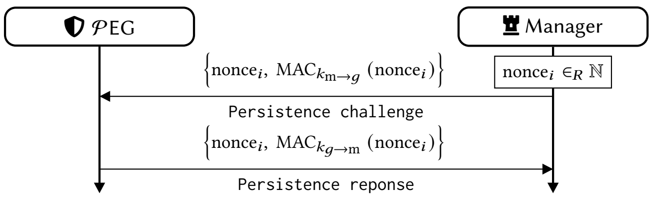

PEG
Manager
$\{\text{nonce}_i,\ \text{MAC}_{k_{m\to g}}(\text{nonce}_i)\}$
$\text{nonce}_i \in_R \mathbb{N}$
Persistence challenge
$\{\text{nonce}_i,\ \text{MAC}_{k_{g\to m}}(\text{nonce}_i)\}$
Persistence reponse

Figure 6: The persistence protocol flow.

Table 1: An overview of symbolic notation.

Symbol	Description
T_x	Name of a timer or duration
t_x	Time when event x occurs
$t_x^{\text{start}}, t_x^{\text{end}}$	Time when timer x is started or expires, respectively
S_x, E_x	Static and ephemeral key pairs of entity x, respectively
$S_x^{\text{Pub}}, E_x^{\text{Pub}}$	Static and ephemeral public key of entity x, respectively

Due to 𝒫EG's modular design, each module can be analyzed independently: properties provided by authentication and persistence modules are only dependent on session keys being correctly established by the pairing module. The enforcement module has no dependencies. Therefore, when individual modules are updated or replaced, only those modules need to be re-verified.

6.1 Pairing Module

We analyze the pairing module using Tamarin [33]. We assume that all long-term (i.e., static) public keys are known to the adversary. We model the scanning of the public and pre-shared keys inscribed on the device using two rules: (1) `Scan_device` rule, which generates a fact representing the deployment manager's knowledge of inscribed keys, and (2) `Scan_device_adversary` rule which adds inscribed keys to adversary's knowledge. Also, the model includes a rule that leaks 𝒫EG's internal long-term secrets.

Further, the model does not limit the number of 𝒫EG or the deployment managers in a trace and supports re-associating after a completed handshake. The model also allows for an ongoing pairing operation to be aborted when a new pairing operation starts. Moreover, for each operation, the internal key management of 𝒫EG is modeled.

Using the model described above, we validate properties **PAIR-1**–**PAIR-4** and find that all of them hold.

6.2 Authorization Module

We model the authorization protocol using Tamarin. Our model accounts for re-association events, key disclosure, counter reveals, timer expirations, and 𝒫EG reboots.

Because Tamarin's discrete time system only models order, and not elapsed time, properties **AUTH-1** and **AUTH-2** cannot be proven directly. Instead, we use a hybrid approach" we prove base statements using Tamarin, from which we then derive **AUTH-1** and **AUTH-2** using conventional logic. To this end, we instrument the model with action facts that indicate when the T_{chal} and T_{auth} timers are started or when they expire. We then prove the following properties, referring to Table 1 for notation.

T-1 A T_{auth} timer cannot start after its associated T_{chal} timer expired: $t_{\text{auth}}^{\text{start}} < t_{\text{chal}}^{\text{end}}$.
T-2 A T_{chal} timer must be started before the deployment manager can issue a corresponding access grant: $t_{\text{chal}}^{\text{start}} < t_{\text{grant}}$.
T-3 A T_{auth} timer can only start after the deployment manager issued the corresponding access grant: $t_{\text{auth}}^{\text{start}} > t_{\text{grant}}$.
T-4 A peripheral interaction can only occur between the start and expiry of a T_{auth} timer: $t_{\text{auth}}^{\text{start}} < t_{\text{interaction}} < t_{\text{auth}}^{\text{end}}$.
T-5 At most one T_{auth} timer can be started per access grant.

From these properties, we show **AUTH-1** as follows: From **T-1**, we know that: $t_{\text{auth}}^{\text{start}} < t_{\text{chal}}^{\text{end}} = t_{\text{chal}}^{\text{start}} + T_{\text{chal}}$ Therefore, $t_{\text{auth}}^{\text{end}} = t_{\text{auth}}^{\text{start}} + T_{\text{auth}} < t_{\text{chal}}^{\text{start}} + T_{\text{chal}} + T_{\text{auth}}$.

Combined **T-2**, this yields $t_{\text{auth}}^{\text{end}} < t_{\text{grant}} + T_{\text{chal}} + T_{\text{auth}}$ In other words, T_{auth} expires at most $T_{\text{chal}} + T_{\text{auth}}$ after the deployment manager issued the corresponding access grant. Similarly, **T-3** states that T_{auth} must be started after the deployment manager's access grant. Combining these two results with **T-4** yields that**AUTH-1** holds. □

We show **AUTH-2** by combining **T-5** and **T-4**. This directly yields **AUTH-2**. □

6.3 Enforcement Module

Unlike other modules, the enforcement module is not protocol-based, which rules out Tamarin. Instead, we verify **ENF-1** using circuit analysis.

Open Collector (Fig. 5a) When 𝒫EG pulls data lines to ground, the drain source resistance R_{ds} of MOSFETs induces a small voltage drop across each MOSFET. Assuming a common 2N7000 MOSFET [44] with $R_{ds} = 6.0\,\Omega$, a system voltage $V_{cc} = 3.3\,\text{V}$, and a data pull-up resistor value of $2\,\text{k}\Omega$, the resulting voltage on data lines will be $\frac{6\,\Omega}{2\,\text{k}\Omega + 6\,\Omega} \cdot 3.3\,\text{V} \approx 10\,\text{mV}$. Assuming that the line driver has perfect switching capabilities, it could toggle data lines between 0 and 10 mV.[2] When connected to a digital input, both values will be read as low; thus, no communication is possible. However, care must be taken to ensure that no other (more sensitive) sampling hardware can be connected to the bus, e.g., by reassigning data pins to an analog-to-digital converter (ADC).

Push-Pull (Fig. 5b) Digital buffers provide much better isolation than the circuit in Fig. 5a. For example, the MC74VHC541 buffer [45] lists a maximum tri-state leakage current of 0.25 µA. This makes it extremely unlikely that signal could be recovered using IoT device hardware.

Analog (Fig. 5c) When $\mathcal{P}$EG disables the signal, the feedback factor of the opamp is $b = \frac{R_{ds}}{R_{ds}+R}$. Again, assuming the 2N7000 MOSFET [44], and $R = 10\,\text{k}\Omega$ this results in $b \approx 6 \cdot 10^{-4}$. Assuming a LMV358 opamp [55] with a large signal differential voltage gain $A_{vd} = 10^4$, the closed loop gain becomes $A = b^{-1} \cdot \left(1 + (A_{vd} \cdot b)^{-1}\right)^{-1} \approx 1429$ [23]. Assuming 0 V and 3.3 V supply rails, this means that any input signal above 2.3 mV would be clamped to the 3.3 V rail, rendering the output meaningless.

6.4 Persistence Module

We model the persistence protocol using Tamarin. Our model covers re-association events, key disclosure, multiple $\mathcal{P}$EG transmissions with the same nonce, and loss of nonce state on the guard, e.g., caused by a reboot. We only model stand-alone persistence messages and defer the formal verification of the piggyback-based persistence module to future work due to current limitations in formal verification tools. We find that property **PERS-1** holds.

6.5 Reference Monitor Properties

We show that $\mathcal{S}$A^{4}P satisfies three defining properties of a reference monitor [36]: (i) non-bypassability, (ii) tamper proofness, and (iii) verifiability. Non-bypassability prevents unauthenticated peripheral access, tamper proofness prevents manual or programatic modifications of the reference monitor, and verifiability prescribes a sufficiently small TCB that can be analyzed and proven correct.

Non-Bypassability Peripheral access is restricted by the enforcement module (**ENF-1**), unless permitted by the authorization module (**AUTH-1** and **AUTH-2**). Access is only permitted by the authorization module for T_{auth} seconds in response to a fresh and authentic access grant from the deployment manager. Freshness is ensured by the counter in each request and T_{chal} window, while authenticity is ensured by the MAC tag. The deployment manager only grants access to fresh and authentic requests permitted by its policy.

Tamper-Proofness Trusted $\mathcal{P}$EG is implemented either inside ARM TrustZone or on a physically separate MCU. Recall that we only consider software and non-invasive physical tampering, while invasive physical tampering is out of scope. TrustZone, similar to a physically separate MCU, provide strong isolation between secure $\mathcal{P}$EG and untrusted $\mathcal{R}$untime. By connecting any physical interface directly to pins controlled by TrustZone's *Secure World*[3], or directly to the separate MCU, $\mathcal{R}$untime cannot tamper with any physical interactions. **PERS-1** ensures attempts to re-associate a device are detected.

Verifiability Authorization, pairing, and persistence modules are formally verified, and can be implemented using simple primitives. Circuit verification was used to verify the enforcement module. The deployment manager is more complex as it also implements access control policy. Prior work on policy analyzers, however, provides powerful mechanisms for auditing access policies [37].

7 IMPLEMENTATION

To assess $\mathcal{S}$A^{4}P's practicality, we implement two $\mathcal{P}$EG variants. The first uses a dedicated MCU to implement a stand-alone $\mathcal{P}$EG and the second uses a TrustZone-enabled Cortex-M MCU as $\mathcal{R}$untime, placing $\mathcal{P}$EG logic in the secure TrustZone partition on the same MCU. Both are publicly available [9]. We instantiate them on an ultra-low-power STM32L552ZE MCU. For the stand-alone $\mathcal{P}$EG, we fully disable TrustZone. STM32L552ZE runs at 110 MHz, has 256 kB of SRAM, and 512 kB of flash. We set $T_{\text{chal}} = 20\,\text{ms}$ and $T_{\text{auth}} = 10\,\text{s}$.

7.1 Stand-alone Peripheral Guard

Stand-alone $\mathcal{P}$EG is implemented on a dedicated MCU. Although it is possible to independently connect this $\mathcal{P}$EG to the network, we refrain from doing so, in favor of proxying all commands over $\mathcal{R}$untime using a UART connection. This has two advantages: (i) obviates the need for a (complex) networking stack on the $\mathcal{P}$EG, significantly reducing TCB size; and (ii) makes it easier for $\mathcal{R}$untime to stay in sync with $\mathcal{P}$EG, since $\mathcal{R}$untime can monitor all $\mathcal{P}$EG communication. Although this allows $\mathcal{R}$untime to perform denial-of-service (DoS) attacks against $\mathcal{P}$EG, doing so would be strictly against its own interests.

Message Format To facilitate communication over the serial link, we implement a serial message format to carry protocol messages from Figs. 3, 4 and 6. For serial transmission, a 3 B header – consisting of a 1 B type field and a 2 B length field – prefixes each message. To reduce the number of messages, we combine authorization and persistence protocol messages.

Cryptography Asymmetric key exchanges are realized using *compact25519* library [27] on Curve25519. All symmetric keys are 256 bits. MACs are implemented using HMAC with SHA256. HMAC and randomness generation are hardware-supported.

Counters and Persistent Memory The authorization protocol uses a monotonic counter to prevent replays. It can be implemented using either persistent memory or a secure clock. Since the latter is expensive and persistent memory is already required to store session keys, we opt for the former. However, counter update events are expected to be frequent, unlike re-association events. Thus, care must be taken not to wear out MCU's memory prematurely via excessive writes. For example, flash memory of STM32L552ZE MCU is specified as being able to withstand at least 10 000 writecycles. This number is unlikely to be reached via re-association events.

[2]Most components do not do this, e.g., standard-complying I2C implementations interpret low bus voltage as an ongoing transmission and refrain from sending [42].

[3]See: https://developer.arm.com/documentation/100935/0100/The-TrustZone-hardware-architecture-

However, with only five counter updates per day, this number can be reached in 5 years.

To avoid memory wear, we use a hybrid 8 B counter consisting of two 4 B sub-counters: upper and lower, referred to as *reboot* and *request* counters, respectively. The latter functions as expected, incremented each time a challenge is generated. However, instead of persistent memory, it is stored in RAM and initialized to zero at each device boot. Meanwhile,reboot counter *is* stored in persistent memory, and incremented each time the device boots. Hence, the combined counter remains strictly incremental, and only requires a write to persistent memory once per device boot cycle, thus mitigating memory write constraints. To ensure integrity of reboot counter in case of boot interruptions, it is replicated three times in memory. A corrupted value can thus always be recovered using majority voting. Alternatively, we could use a specialized memory type, e.g., ferroelectric RAM (FRAM), for counter storage, though such memory types are much less common.

Timeouts Because the state of T_{chal}only needs to be checked when processing an incoming message, it is implemented using system ticks. Conversely, T_{auth} uses a dedicated hardware timer that triggers an interrupt when it expires. This ensures that access to protected peripherals is always promptly removed.

Enforcement Module Our $\mathcal{P}$EG implementation writes a binary signal representing the current peripheral enforcement state to a General Purpose Input/Output (GPIO) output pin. An external enforcement circuit (e.g., one of the circuits in Fig. 5) can be connected to this pin.

7.2 TrustZone Peripheral Guard

Unlike the dedicated $\mathcal{P}$EG, the TrustZone variant shares processing hardware with the $\mathcal{R}$untime. Concretely, one TrustZone enabled Cortex-M MCU is flashed with two binaries: secure and non-secure. We use the former for $\mathcal{P}$EG functionality, and the latter for $\mathcal{R}$untime code. Isolation of $\mathcal{P}$EG is ensured by TrustZone.

Implementing $\mathcal{P}$EG using TrustZone has the primary advantage of not requiring additional hardware. Also, TrustZone-based $\mathcal{P}$EGs should be applicable to current TrustZone-capable MCUs types.[4]

API Another advantage of a TrustZone-based $\mathcal{P}$EG is that no (slow) communication bus between $\mathcal{P}$EG and $\mathcal{R}$untime is needed. Instead, TrustZone-based $\mathcal{P}$EG exposes a narrow API to $\mathcal{R}$untime, consisting of one call per incoming message type. Messages are passed as function arguments and return values.

Enforcement Module Contrary to dedicated $\mathcal{P}$EG, TrustZone-based $\mathcal{P}$EG does not need external enforcement circuitry. Instead, enforcement can dynamically modify the MCU's TrustZone configuration to provide or revoke $\mathcal{R}$untime's access to the interface to which the protected peripheral is connected [53].

Preventing Interference Having $\mathcal{R}$untime and $\mathcal{P}$EG share a computation platform increases the risk of interference by $\mathcal{R}$untime into $\mathcal{P}$EG operations. Although default TrustZone behavior suffices to protect $\mathcal{P}$EG's secrets, additional care must be taken so that untrusted $\mathcal{R}$untime code cannot interfere with the $\mathcal{P}$EG's notion of time. Most importantly, the `PRIS` bit in the Application Interrupt and Reset Control Register (AIRCR) must be set to ensure that the code running in TrustZone's *Normal World* ($\mathcal{R}$untime) cannot prevent $\mathcal{P}$EG code in *Secure World* from executing by masking its interrupt service routines [31].

[4]No general statements can be made about feasibility of this approach, since it is largely dependent on the design of individual products.

Table 2: Sizes of messages exchanged between the $\mathcal{P}$EG and $\mathcal{R}$untime, including 3 B serial encapsulation header.

Message	Generated by	Size [B]
Pairing		
Key exchange I	$\mathcal{P}$EG	67
Key exchange II	Manager	67
Key confirmation	$\mathcal{P}$EG	35
Authorization & persistence		
Access initialization	$\mathcal{R}$untime	3
Access request	$\mathcal{P}$EG	61
Access grant	Manager	51
Access confirmation	$\mathcal{P}$EG	6

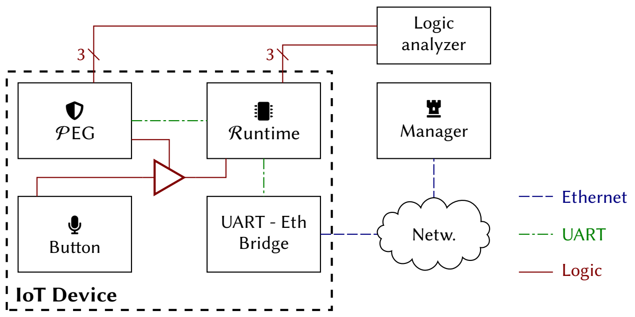


Figure 7: Schematic overview of the macro evaluation setup.

8 EVALUATION

To evaluate our $\mathcal{P}$EG implementations, we appl;y a macro-scale benchmark which measures overall system performance of a $\mathcal{P}$EG-enabled device. To gain further insights, we perform a series of micro-scale benchmarks that evaluate performance of the $\mathcal{P}$EGs in isolation.

8.1 Macro-scale Evaluation

Modifying devices to adopt the $\mathcal{S}$A^{4}P philosophy will inevitably have an impact on system operations. Concretely, the requirement to obtain permission ahead of peripheral interactions introduces both bandwidth overhead and access latency. Given that pairing events are expected to be rare, we focus this evaluation on access requests and grants.

Bandwidth Overhead (Access Event) The communication overhead in terms of message sizes can be determined directly from the protocol specification (Sections 5.1, 5.2 and 5.4) and the serial message format (Section 7.1). The resulting message sizes are shown in Table 2.

Latency Overhead (Access Event) To quantify the latency overhead, we performed end-to-end system measurements using the setup shown in Fig. 7. The setup consists of a proof-of-concept $\mathcal{P}$EG-enabled device that communicates with a deployment manager over a standard Ethernet network. The $\mathcal{P}$EG-enabled device

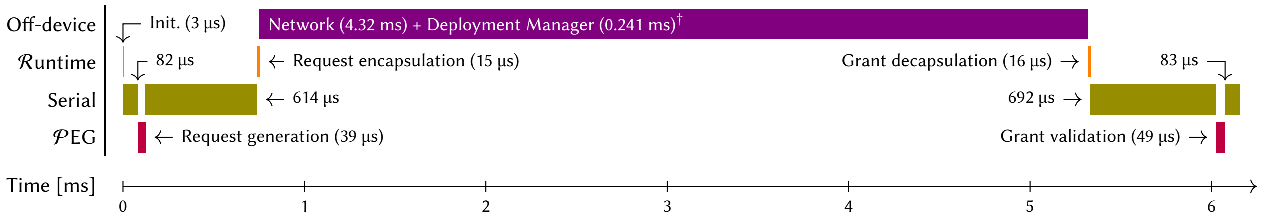


Figure 8: Timeline of an access requests to a dedicated $\mathcal{P}$EG. Averaged values, $N = 468$. †Deployment manager processing times were measured during a separate experiment with $N = 100$ and an independent time base.

consists of a button functioning as a rudimentary sensor, a STM32 as dedicated $\mathcal{P}$EG, and a second, identical, MCU functioning as $\mathcal{R}$untime. Communication between the $\mathcal{P}$EG and $\mathcal{R}$untime uses a UART bus running at 921 kbaud. $\mathcal{R}$untime is connected to the Ethernet network using a WIZnet W7500S2E-R1 UART to Ethernet bridge, running at 461 kbaud. The manager has a grant-all policy. It is implemented in Golang and runs on a commodity laptop.

To extract timing information, $\mathcal{P}$EG and $\mathcal{R}$untime are connected to the same logic analyzer sampling at 16 MHz, each using a 3-bit logic bus. The code running on both platforms is instrumented to write code points to the bus, providing insight into internal operations and timing.

We use the setup described above to measure the overall system performance of an access request (including the piggy-backed persistence exchange). The results are shown in Fig. 8.

As can be seen in Fig. 8, it takes, on average, 6.16 ms from the point where $\mathcal{R}$untime initiates an access request to the point at which $\mathcal{R}$untime is informed that peripheral access has been provided by the $\mathcal{P}$EG. This duration is primarily dominated by serial (1.47 ms) and network[5] (4.32 ms) delays, which sum up to 5.79 ms. We attribute the majority of the network delays to the relatively slow WIZnet module.

Processing times on the protocol endpoints ($\mathcal{P}$EG and deployment manager) are an order of magnitude shorter, with a combined processing time of 0.09 ms measured on the $\mathcal{P}$EG, and 0.24 ms measured on the deployment manager. The latter being higher due to the use of a general purpose operating system on the deployment manager. The forwarding overhead on $\mathcal{R}$untime was measured to sum up to 0.03 ms, which we consider to be negligible.

Pairing Events The sizes of pairing messages are shown in Table 2. We informally measured a pairing event to take around 9 s. More detailed measurements are given in the following section on micro evaluation.

8.2 Micro-scale Evaluation

To gain further insight into $\mathcal{P}$EG performance, we continue with a set of micro benchmarks. Similar to the macro evaluation setup, we connect the $\mathcal{P}$EG to a logic analyzer using a 3-bit bus, and instrument the $\mathcal{P}$EG code to write code points to this bus at significant points in its program.

Dedicated $\mathcal{P}$EG (Access Event Timing) We start by taking a more detailed measurement of the processing times required to generate an access request and validate a grant on the dedicated $\mathcal{P}$EG. The results of this measurement are shown in Table 3. As can be seen from the table, the processing times are highly deterministic, with low spread. The small discrepancy between the values in Fig. 8 and Table 3 can be attributed to slight differences in code instrumentation and compile-time optimization.

Table 3: Processing times for access events. Values in µs.

	Mean	Std. Dev.	Min	Max
Dedicated $\mathcal{P}$EG (N=288)				
Request generation	40.34	0.11	40.25	41.25
Grant validation	50.28	0.18	50.17	51.17
TrustZone $\mathcal{P}$EG (N=324)				
Request generation	33.52	0.34	32.62	34.63
Grant validation	49.98	0.54	48.63	52.87

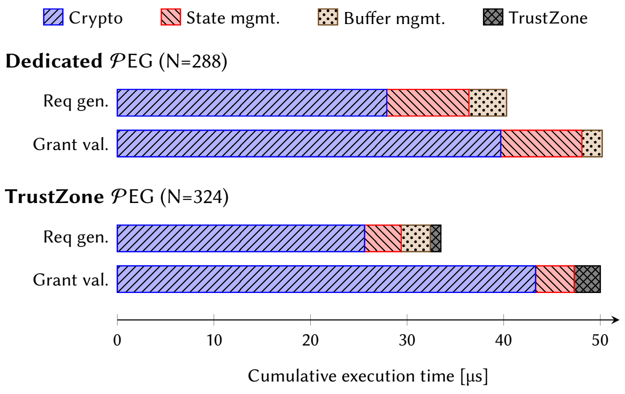


Figure 9: $\mathcal{P}$EG load profiles for access-event messages.

Dedicated $\mathcal{P}$EG (Access Event Load Profile) To further break down the numbers shown in Table 3, Fig. 9 shows relative load profiles for the $\mathcal{P}$EG. We see that for both messages, the workload is dominated by the cryptographic tag verification, which is to be expected.

TrustZone $\mathcal{P}$EG (Access Event Timing) Next, we compare the performance of the dedicated $\mathcal{P}$EG to the TrustZone-based $\mathcal{P}$EG. Table 3 and Fig. 9 show the execution times of `get_challenge()` and `put_response()` $\mathcal{P}$EG API calls as observed by $\mathcal{R}$untime code. These calls respectively initiate an access request, or supply the $\mathcal{P}$EG with a response from the manager. We see that the latencies

[5]Network delays include serial transmissions to the UART to Ethernet bridge.

Table 4: Dedicated $\mathcal{P}$EG processing times for pairing message processing and generation. Values in ms.

Outgoing message	Mean	Std. Dev.	N
Key exchange I	2891	0.30	281
Key confirmation	5737	0.92	155

Table 5: Dedicated $\mathcal{P}$EG processing times for pairing-related cryptographic operations. Values in ms.

Operation	Mean	Std. Dev.	N
Ephemeral key pair generation	2891	0.30	281
Diffie-Hellmann derivation	2868	0.46	155
HKDF computation	0.08	< 0.01	155

of the two implementations are similar. However, TrustZone numbers already include the $\mathcal{P}$EG to $\mathcal{R}$untime communication, whereas the dedicated $\mathcal{P}$EG relies on time-intensive serial communication. Therefore, the system performance of the TrustZone $\mathcal{P}$EG is superior. That said, as is shown in Table 3, the response times of the TrustZone $\mathcal{P}$EG are less stable than those of the dedicated $\mathcal{P}$EG. This is to be expected, as the TrustZone implementation shares a processor core with $\mathcal{R}$untime.

Dedicated $\mathcal{P}$EG (Pairing Event Timing) Although pairing events are expected to be rare, we performed measurements to get an understanding of their performance. Table 4 shows the time required to process the pairing messages on the $\mathcal{P}$EG. Table 5 lists the measured duration of individual cryptographic operations, providing more insight in the composition of the delays shown in Table 4. We see that (as expected) the processing times are dominated by the cryptographic operations. We note that two Diffie-Hellmann operations take place for each key confirmation message.

8.3 Memory Footprint

We measure the memory footprint of both $\mathcal{P}$EG implementations. The dedicated $\mathcal{P}$EG has a binary size of 14.97 kB and a RAM footprint of 3.18 kB. The TrustZone-based $\mathcal{P}$EG has a binary size of 6.62 kB and a RAM footprint of 1.9 kB.

9 DISCUSSION

9.1 Peripheral Access Latency

The results from Section 8 confirm the feasibility of the $\mathcal{S}$A^{4}P approach. Although the newly introduced peripheral access latency ($\leq$ 6 ms) is non-negligible, it is not of a prohibitive nature for the applications we envision; system response times below 100 ms are generally perceived as instantaneous by users [39]. Moreover, significant performance improvements are still achievable. Concretely, we expect that when a TrustZone-based $\mathcal{P}$EG is used, and our proof-of-concept networking setup is replaced by a networking stack running directly on $\mathcal{R}$untime, an access latency below 1 ms is attainable. Additionally, $\mathcal{P}$EG operations have shown to be both fast and deterministic, which facilitates access request scheduling.

9.2 T_{chal}, T_{auth}, and Network Overhead

The timer durations T_{chal} and T_{auth} should be set based on the characteristics of the deployment: T_{chal} should be set to sum of the largest expected round trip time (RTT) from the $\mathcal{P}$EG to the deployment manager, and the highest expected deployment manager response time. In most deployments this value will be on the order of milliseconds or less.

Setting T_{auth} is more complicated, as it requires a trade-off between control and network overhead: setting T_{auth} high results in coarse access control and low overhead, whereas a low T_{auth} provides fine-grained control at the cost of higher network overhead. To quantify this overhead, consider a deployment with $T_{\text{auth}} = 10$ s, and in which access requests are sent with a 100 ms overlap to ensure access continuity. Assuming our $\mathcal{S}$A^{4}P instantiation, the resulting network overhead during peripheral access events would average $\frac{1}{10-0.1}\text{req/s} \times (61+51)\text{B/req} = 90\,\text{bps}$. Although we expect this data rate to be acceptable in most application settings, this might not hold for some highly constrained scenarios. Strategies to reduce the network overhead include:

- Increasing T_{auth}. In some deployment scenarios significantly longer values of T_{auth} can be considered. For example, in wireless sensor networks where the primary goal of $\mathcal{S}$A^{4}P deployment is to remove peripheral access when the deployment is compromised, values of T_{auth} up to multiple hours may be appropriate.
- The length of the various protocol fields can be reduced, albeit at the cost of weakened cryptographic properties.
- Multiple access types associated with the same peripheral, but with different associated T_{auth} values, can be used.

9.3 High-Level Context

Although the low-level nature of the $\mathcal{P}$EG makes the design inclusive and robust, it limits the amount of high-level contextual information that is inherently available to base access decisions on. This means that the deployment manager must actively collect such information from across the deployment. How this collection should be implement is beyond the scope of this paper. Additionally, the $\mathcal{S}$A^{4}P design philosophy is not intended to replace existing high-level access control mechanism, but rather to complement them.

Example: Combining with Attestation One type of high-level context that could be gathered, is the attestation state of the $\mathcal{P}$EG-enabled devices. By requiring $\mathcal{R}$untime to present an attestation proof together with each access request, the deployment manager can ensure that peripheral access is rapidly removed after a device has been compromised. Doing so would remove significant adversarial utility from compromised devices, effectively rendering them to be generic network-connected computational nodes.

9.4 Post-Sampling Access Control

$\mathcal{S}$A^{4}P is designed to prevent unauthorized interactions between the physical and virtual domain. It does not provide any mechanisms to control or limit data dissemination in the virtual domain after data has been sampled from the physical domain. If such mechanisms are required, they can be deployed together with $\mathcal{S}$A^{4}P.

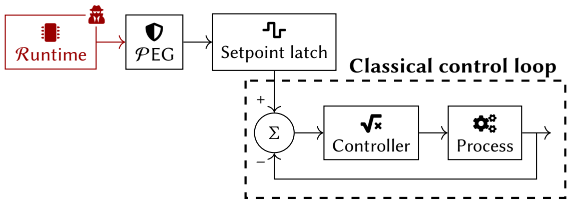


Figure 10: A $\mathcal{S}A^4P$ design only requiring authorization for setpoint adjustments. 🕵 indicates untrusted components.

9.5 Non-Binary Enforcement Modules

In Section 6.3 we only considered *binary* enforcement modules, i.e., enforcement modules that either allow for full peripheral access or non at all. However, it is also possible to design non-binary enforcement modules, e.g., by integrating a low-pass filter into their design. Doing so would allow the $\mathcal{P}$EG to provide more fine-grained access control. For example, it has been shown that the high frequency components in a humidity signal can reveal information about human presence [20]. A low-pass filter enabled $\mathcal{P}$EG could selective provide access to the average room humidity while filtering out sensitive room occupancy information.

9.6 Protecting Complex Actuators

For simple actuators, our $\mathcal{P}$EG design is directly applicable. In fact, as part of this research project we have created a demo that uses our $\mathcal{P}$EG design to control access to an electric strike plate, giving the deployment manager control over when a door can be opened.

For other actuators (e.g., lights), a designer might opt to place a latch or a flip flop behind the $\mathcal{P}$EG, so that an access permission is only required to toggle the actuator. More generally, a setpoint-based control loop could be used, thus only requiring manager authorization for setpoint adjustments. Such an architecture is depicted in Fig. 10. The primary challenge associated with this approach is that all components placed behind the $\mathcal{P}$EG must be trusted.

9.7 Information Leakage

The $\mathcal{P}$EG design does not encrypt network traffic, since, unlike data authenticity, data secrecy is not required for our security goals. However, this means that passive network adversaries can observe access requests in cleartext, including the access type field. This exposure can be addressed by creating an encrypted tunnel between $\mathcal{R}$untime and the deployment manager. However, encryption offers limited protection: previous work has shown that even encrypted IoT traffic leaks significant private information [2, 6, 57].

10 RELATED WORK

Remote attestation The use of remote attestation techniques to secure IoT devices has been widely studied. In their most essential form, attestation techniques provide integrity guarantees about binaries. Such *static* techniques are not only widely used in research (e.g., in the Sancus security architecture for IoT devices [40]), but have also matured enough to see applications in industry, most commonly in combination with secure boot [5, 8].

However, static attestation cannot provide guarantees about control-flow integrity. C-FLAT [1] and LO-FAT [16] address this issue by designing IoT-compatible control-flow attestation schemes. C-FLAT does so by relying on TrustZone for Cortex-A, while LO-FAT introduces custom hardware. SIMPLE [4] continues this line of work by proposing a software-only remote attestation mechanism that implicitly guarantees control flow integrity. In contrast, OAT [54] maintains a dependency on TrustZone, but adds data integrity guarantees.

Although all of these results provide strong guarantees about the code being executed on an IoT device, they cannot prevent a compromised device from accessing its interaction peripherals. Therefore they provide only limited privacy guarantees. In summary, device attestation is an orthogonal research direction to ours, and we discuss in Section 9.3 how device attestation and $\mathcal{S}A^4P$ can complement each other.

IoT recovery mechanisms A number of prior efforts studied how IoT devices can be efficiently recovered after compromise. For example, FIRE [51]) sends code update requests and blacklists devices when a successful recovery cannot be confirmed. Cider [62] presents a mechanism that guarantees that updates are installed within a bounded time frame. Lazarus [21] provides similar properties as Cider, though it targets more constrained devices and uses TrustZone on Cortex-M to eliminate some of Cider's hardware requirements. Verify&Revive [3] presents a pure-software-based device healing scheme, albeit at the cost of weaker availability guarantees.

This work is similar to ours in that they can be used to prevent an adversary from sampling sensors after a breach has been detected. However, they cannot be used to regulate sensor access during normal deployment operations.

Managing Sensor Access, Notifications, and Logging Brasser et al. use TrustZone on Cortex-A and remote memory writes to restrict peripheral access [11]. Unlike this work, it targets feature-rich devices (e.g., smartphones or laptops) and does not provide fine-grained temporal control. Instead, the peripheral access policy can only be updated during check-in and check-out events. SeCloak [30] targets similar devices as Brasser et al. and also relies on TrustZone. However, it is designed to provide on-device control, i.e., SeCloak does not delegate access policy management to a remote server.

VERSA [41] uses a modified MCU to ensure that only attested and explicitly authorized routines can access sensors, based on remotely-issued authorization tokens. Although VERSA requires the remote verifier to issue a new token for each invocation of the sampling routine, it does not provide the same bounded-time guarantees as our work. Moreover, VERSA requires custom hardware and its reliance on attestation requires significantly closer integration between the remote verifier and the VERSA-enabled device.

Ditio [34] leverages TrustZone on Cortex-A to provide secure logs of sensor activities to a remote server. Viola [35] uses OS- or hypervisor-level checks to enforce a one-to-one relationship between on-device sensor notifications (e.g., using a notification LED) and sensor activity. Neither Ditio nor Viola target constrained devices.

In the industrial setting, C^2 [32] monitors programmable logic controller (PLC) control and sensor signals to verify that a preset policy is followed. If a violation is detected, a safe fall-back system is executed. M2Mon [24] implements a non-distributed peripheral reference monitor for unmanned vehicles.

Camera Privacy Much research has focused on camera privacy, typically by performing video anonymization by obfuscating sensitive video regions [61]. Contributions in this area can be categorized based on when the obfuscation is performed: before [47, 58, 63], during [59]), or after [12, 60] capture time. The first two of these categories are of special interest, as, similar to our work, they perform access regulation before $\mathcal{R}$untime. Zhang et al. [63] and Pittaluga et al. [47] propose to place dynamic, privacy-preserving optics in front of image sensors. CamShield [58] takes a different approach by fully obscuring the view of the main image sensor, and instead expose it to a pre-anonymized digital video stream. TurstEYE.M4 [59] proposes a sensor unit with an integrated privacy filter, resulting in a pre-filtered video stream being received by $\mathcal{R}$untime.

Contrary to our work, all of this work focus on image privacy and none of them have provisions for interactions with a deployment manager.

Device-Level IoT Access Management Many results consider device-level IoT access control. This includes both research papers (e.g., [17, 18, 25, 38, 43, 56]) and industry standards (e.g., [13, 50]). Unlike the present work, these techniques operate at the device or agent level. However, some of them, e.g., AoT's [38] full-lifecycle key management mechanism, could be used to instantiate a $\mathcal{P}$EG pairing module.

11 CONCLUSION

As IoT devices are becoming increasingly ubiquitous and intertwined with our daily life, their vulnerabilities are creating severe security and privacy risks for their users. The $\mathcal{S}$A^{4}P framework introduces a new approach to address this challenge: by controlling a device's ability to interact with the physical environment, $\mathcal{S}$A^{4}P can significantly reduce the risks of rogue IoT devices. Our implementations demonstrate that $\mathcal{S}$A^{4}P is technically feasible and practical, thus making it a promising approach for securing common types of IoT devices.

REFERENCES

[1] 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*. ACM. https://doi.org/10.1145/2976749.2978358

[2] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. 2020. Peek-a-boo: I see your smart home activities, even encrypted!. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM. https://doi.org/10.1145/3395351.3399421

[3] Mahmoud Ammar and Bruno Crispo. 2020. Verify&Revive: Secure Detection and Recovery of Compromised Low-end Embedded Devices. In *Annual Computer Security Applications Conference*. ACM. https://doi.org/10.1145/3427228.3427253

[4] Mahmoud Ammar, Bruno Crispo, and Gene Tsudik. 2020. SIMPLE: A Remote Attestation Approach for Resource-constrained IoT devices. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE. https://doi.org/10.1109/iccps48487.2020.00036

[5] Analog Devices. [n. d.]. *The Fundamentals of Secure Boot and Secure Download: How to Protect Firmware and Data within Embedded Devices*. Technical Report.

[6] Noah Apthorpe, Dillon Reisman, Srikanth Sundaresan, Arvind Narayanan, and Nick Feamster. 2017. Spying on the Smart Home: Privacy Attacks and Defenses on Encrypted IoT Traffic. https://doi.org/10.48550/ARXIV.1708.05044

[7] Ionut Arghire. 2022. Nuki Smart Lock Vulnerabilities Allow Hackers to Open Doors. https://www.securityweek.com/nuki-smart-lock-vulnerabilities-allow-hackers-open-doors/.

[8] ARM. 2009. *ARM Security Technology: Building a Secure System using TrustZone Technology*. Technical Report.

[9] SA4P Authors. 2023. GitHub repositories accompanying this paper. https://github.com/SA4P.

[10] Carsten Bormann, Mehmet Ersue, and Ari Keränen. 2014. Terminology for Constrained-Node Networks. RFC 7228. https://doi.org/10.17487/RFC7228

[11] Ferdinand Brasser, Daeyoung Kim, Christopher Liebchen, Vinod Ganapathy, Liviu Iftode, and Ahmad-Reza Sadeghi. 2016. Regulating ARM TrustZone Devices in Restricted Spaces. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM. https://doi.org/10.1145/2906388.2906390

[12] Ankur Chattopadhyay and T.E. Boult. 2007. PrivacyCam: a Privacy Preserving Camera Using uCLinux on the Blackfin DSP. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. https://doi.org/10.1109/cvpr.2007.383413

[13] Connectivity Standards Alliance, Inc. 2022. *Matter Specification Version 1.0*. Technical Report.

[14] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. 2014. A Large-Scale Analysis of the Security of Embedded Firmwares. In *Proceedings of the 23rd USENIX Security Symposium*.

[15] Anupam Das, Martin Degeling, Daniel Smullen, and Norman Sadeh. 2018. Personalized Privacy Assistants for the Internet of Things: Providing Users with Notice and Choice. *IEEE Pervasive Computing* 17, 3 (July 2018). https://doi.org/10.1109/mprv.2018.03367733

[16] 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*. ACM. https://doi.org/10.1145/3061639.3062276

[17] Sebastian Echeverria, Grace A. Lewis, Dan Klinedinst, and Ludwig Seitz. 2019. Authentication and Authorization for IoT Devices in Disadvantaged Environments. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE. https://doi.org/10.1109/wf-iot.2019.8767192

[18] Geovane Fedrecheski, Laisa Caroline Costa De Biase, Pablo C. Calcina-Ccori, Roseli de Deus Lopes, and Marcelo Knorich Zuffo. 2022. SmartABAC: Enabling Constrained IoT Devices to Make Complex Policy-Based Access Control Decisions. *IEEE Internet of Things Journal* 9, 7 (apr 2022), 5040–5050. https://doi.org/10.1109/jiot.2021.3110142

[19] Eileen Guo. 2023. Roomba testers feel misled after intimate images ended up on Facebook. https://www.technologyreview.com/2023/01/10/1066500/.

[20] Jun Han, Abhishek Jain, Mark Luk, and Adrian Perrig. 2007. Don't Sweat Your Privacy: Using Humidity to Detect Human Presence. In *Proceedings of the International Workshop on Privacy in UbiComp (UbiPriv)*. /publications/papers/han_jain_luk_perrig_privacy.pdf

[21] Manuel Huber, Stefan Hristozov, Simon Ott, Vasil Sarafov, and Marcus Peinado. 2020. The Lazarus Effect: Healing Compromised Devices in the Internet of Small Things. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. ACM. https://doi.org/10.1145/3320269.3384723

[22] Umar Iqbal, Pouneh Nikkhah Bahrami, Rahmadi Trimananda, Hao Cui, Alexander Gamero-Garrido, Daniel Dubois, David Choffnes, Athina Markopoulou, Franziska Roesner, and Zubair Shafiq. 2022. Your Echos are Heard: Tracking, Profiling, and Ad Targeting in the Amazon Smart Speaker Ecosystem. https://doi.org/10.48550/ARXIV.2204.10920

[23] Jim Karki. 2021. *Application Report: Understanding Operational Amplifier Specifications*. Technical Report. Texas Instruments.

[24] Arslan Khan, Hyungsub Kim, Byoungyoung Lee, Dongyan Xu, Antonio Bianchi, and Dave (Jing) Tian. 2021. M2MON: Building an MMIO-based Security Reference Monitor for Unmanned Vehicles. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 285–302. https://www.usenix.org/conference/usenixsecurity21/presentation/khan-arslan

[25] Jun Young Kim, Wen Hu, Dilip Sarkar, and Sanjay Jha. 2017. ESIoT: enabling secure management of the internet of things. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM. https://doi.org/10.1145/3098243.3098252

[26] Dr. Hugo Krawczyk and Pasi Eronen. 2010. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869. https://doi.org/10.17487/RFC5869

[27] Davy Landman. 2022. compact25519: A compact portable X25519 + Ed25519 implementation. https://github.com/DavyLandman/compact25519.

[28] Marc Langheinrich. 2002. A Privacy Awareness System for Ubiquitous Computing Environments. In *ACM Conference on Ubiquitous Computing*. https://doi.org/10.1007/3-540-45809-3_19

[29] Ralph Langner. 2011. Stuxnet: Dissecting a Cyberwarfare Weapon. *IEEE Security & Privacy Magazine* 9, 3 (may 2011), 49–51. https://doi.org/10.1109/msp.2011.67

[30] Matthew Lentz, Rijurekha Sen, Peter Druschel, and Bobby Bhattacharjee. 2018. SeCloak: ARM Trustzone-based Mobile Peripheral Control. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. ACM. https://doi.org/10.1145/3210240.3210334

[31] ARM Ltd. 2020. ARM Cortex-M33 Devices Generic User Guide. https://developer.arm.com/documentation/100235/0100. Version r1p0.

[32] Stephen McLaughlin. 2013. CPS: Stateful Policy Enforcement for Control System Device Usage. In *Proceedings of the 29th Annual Computer Security Applications Conference* (New Orleans, Louisiana, USA) *(ACSAC '13)*. Association for Computing Machinery, New York, NY, USA, 109–118. https://doi.org/10.1145/2523649.2523673

[33] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Computer Aided Verification*. Springer Berlin Heidelberg, 696–701. https://doi.org/10.1007/978-3-642-39799-8_48

[34] Saeed Mirzamohammadi, Justin A. Chen, Ardalan Amiri Sani, Sharad Mehrotra, and Gene Tsudik. 2017. Ditio: Trustworthy Auditing of Sensor Activities in Mobile & IoT Devices. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM. https://doi.org/10.1145/3131672.3131688

[35] Saeed Mirzamohammadi and Ardalan Amiri Sani. 2016. Viola: Trustworthy Sensor Notifications for Enhanced Privacy on Mobile Systems. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM. https://doi.org/10.1145/2906388.2906391

[36] National Institute of Standards and Technology. 2020. *Security and Privacy Controls for Information Systems and Organizations*. Technical Report. https://doi.org/10.6028/nist.sp.800-53r5

[37] Timothy Nelson, Christopher Barratt, Daniel Dougherty, Kathi Fisler, and Shriram Krishnamurthi. 2012. The Margrave Tool for Firewall Analysis. (05 2012).

[38] Antonio L. Maia Neto, Artur L. F. Souza, Italo Cunha, Michele Nogueira, Ivan Oliveira Nunes, Leonardo Cotta, Nicolas Gentille, Antonio A. F. Loureiro, Diego F. Aranha, Harsh Kupwade Patil, and Leonardo B. Oliveira. 2016. AoT: Authentication and Access Control for the Entire IoT Device Life-Cycle. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. ACM. https://doi.org/10.1145/2994551.2994555

[39] Jakob Nielsen. 1993. *Usability Engineering*. AP Professional.

[40] Job Noorman, Jo Van Bulck, Jan Tobias Mühlberg, Frank Piessens, Pieter Maene, Bart Preneel, Ingrid Verbauwhede, Johannes Götzfried, Tilo Müller, and Felix Freiling. 2017. Sancus 2.0: A Low-Cost Security Architecture for IoT Devices. *ACM Transactions on Privacy and Security* 20, 3 (jul 2017), 1–33. https://doi.org/10.1145/3079763

[41] Ivan De Oliveira Nunes, Seoyeon Hwang, Sashidhar Jakkamsetti, and Gene Tsudik. 2022. Privacy-from-Birth: Protecting Sensed Data from Malicious Sensors with VERSA. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE. https://doi.org/10.1109/sp46214.2022.9833737

[42] NXP Semiconductors. 2021. *UM10204: I2C-bus specification and user manual*. Technical Report.

[43] Se-Ra Oh, Young-Gab Kim, and Sanghyun Cho. 2019. An Interoperable Access Control Framework for Diverse IoT Platforms Based on OAuth and Role. *Sensors* 19, 8 (apr 2019), 1884. https://doi.org/10.3390/s19081884

[44] ON Semiconductor. 2011. *2N7000G: Small Signal MOSFET 200 mApms, 60 Volts*. Technical Report.

[45] ON Semiconductor. 2014. *MC74VHC541: Octal Bus Buffer*. Technical Report.

[46] Trevor Perrin. 2018. The Noise Protocol Framework. https://noiseprotocol.org/noise.pdf. Revision 34.

[47] Francesco Pittaluga and Sanjeev Jagannatha Koppal. 2017. Pre-Capture Privacy for Small Vision Sensors. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39, 11 (nov 2017), 2215–2226. https://doi.org/10.1109/tpami.2016.2637354

[48] Jing Qiu, Zhihong Tian, Chunlai Du, Qi Zuo, Shen Su, and Binxing Fang. 2020. A Survey on Access Control in the Age of Internet of Things. *IEEE Internet of Things Journal* 7, 6 (jun 2020), 4682–4696. https://doi.org/10.1109/jiot.2020.2969326

[49] D. Quarta, M. Pogliani, M. Polino, F. Maggi, A. M. Zanchettin, and S. Zanero. 2017. An Experimental Security Analysis of an Industrial Robot Controller. In *IEEE Symposium on Security and Privacy*.

[50] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig. 2022. *Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)*. Technical Report. https://doi.org/10.17487/rfc9200

[51] Arvid Seshadri, Mark Luk, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. 2004. *Using FIRE and ICE for Detectin and Recovering Compromised Nodes in Sensor Networks*. Technical Report. Carnegie Mellon University.

[52] Amit Kumar Sikder, Hidayet Aksu, and A. Selcuk Uluagac. 2017. 6thSense: A Context-aware Sensor-based Attack Detector for Smart Devices. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 397–414. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/sikder

[53] Dimitrios Slamaris. 2022. Embedded Systems Security and TrustZone. https://embeddedsecurity.io. Accessed: 2023-02-07.

[54] Zhichuang Sun, Bo Feng, Long Lu, and Somesh Jha. 2020. OAT: Attesting Operation Integrity of Embedded Devices. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. https://doi.org/10.1109/sp40000.2020.00042

[55] Texas Instruments. 1999. *LMV3xx Low-Voltage Rail-to-Rail Output Operational Amplifier*. Technical Report.

[56] Piet De Vaere and Adrian Perrig. 2019. Liam: An Architectural Framework for Decentralized IoT Networks. In *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE. https://doi.org/10.1109/mass.2019.00056

[57] Yinxin Wan, Kuai Xu, Feng Wang, and Guoliang Xue. 2022. IoTAthena: Unveiling IoT Device Activities From Network Traffic. *IEEE Transactions on Wireless Communications* 21, 1 (jan 2022), 651–664. https://doi.org/10.1109/twc.2021.3098608

[58] Zhiwei Wang, Yihui Yan, Yueli Yan, Huangxun Chen, and Zhice Yang. 2022. CamShield: Securing Smart Cameras through Physical Replication and Isolation. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 3467–3484. https://www.usenix.org/conference/usenixsecurity22/presentation/wang-zhiwei

[59] Thomas Winkler, Adam Erdelyi, and Bernhard Rinner. 2014. TrustEYE.M4: Protecting the sensor — Not the camera. In *2014 11th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE. https://doi.org/10.1109/avss.2014.6918661

[60] Thomas Winkler and Bernhard Rinner. 2010. TrustCAM: Security and Privacy-Protection for an Embedded Smart Camera Based on Trusted Computing. In *2010 7th IEEE International Conference on Advanced Video and Signal Based Surveillance*. IEEE. https://doi.org/10.1109/avss.2010.38

[61] Thomas Winkler and Bernhard Rinner. 2014. Security and Privacy Protection in Visual Sensor Networks. *Comput. Surveys* 47, 1 (may 2014), 1–42. https://doi.org/10.1145/2545883

[62] Meng Xu, Manuel Huber, Zhichuang Sun, Paul England, Marcus Peinado, Sangho Lee, Andrey Marochko, Dennis Mattoon, Rob Spiger, and Stefan Thom. 2019. Dominance as a New Trusted Computing Primitive for the Internet of Things. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. https://doi.org/10.1109/sp.2019.00084

[63] Yupeng Zhang, Yuheng Lu, Hajime Nagahara, and Rin ichiro Taniguchi. 2014. Anonymous Camera for Privacy Protection. In *2014 22nd International Conference on Pattern Recognition*. IEEE. https://doi.org/10.1109/icpr.2014.715

[64] Binbin Zhao, Shouling Ji, Jiacheng Xu, Yuan Tian, Qiuyang Wei, Qinying Wang, Chenyang Lyu, Xuhong Zhang, Changting Lin, Jingzheng Wu, and Raheem Beyah. 2022. One Bad Apple Spoils the Barrel: Understanding the Security Risks Introduced by Third-Party Components in IoT Firmware. https://doi.org/10.48550/ARXIV.2212.13716