Exploiting the DICE specification to ensure strong identity and integrity of IoT devices

Exploiting the DICE specification to ensure strong identity and integrity of IoT devices

Enrico Bravi
Politecnico di Torino
Dip. Automatica e Informatica
Torino, Italy
enrico.bravi@polito.it

Silvia Sisinni
Politecnico di Torino
Dip. Automatica e Informatica
Torino, Italy
silvia.sisinni@polito.it

Antonio Lioy
Politecnico di Torino
Dip. Automatica e Informatica
Torino, Italy
antonio.lioy@polito.it

***Abstract*—IoT devices are becoming widely used in several contexts, and nowadays billions of devices are deployed in different scenarios, some of which are very critical to people's privacy and safety. For these reasons, it is very important to provide capabilities for guaranteeing the correct behaviour of the devices. Remote attestation is a technique traditionally used to monitor the integrity status of nodes and to determine if they are behaving as expected. This technique requires that the device is equipped with Roots of Trust, that are the set of hardware and software features that make the platform capable of providing reliable integrity reports even when it has been compromised. This paper proposes a solution that permits to identify and attest devices in a dynamic context, such as Smart Cities or Smart Homes, where unknown devices can connect to the network and perform several actions. The proposed security schema is based on the Device Identity Composition Engine (DICE), which represents a set of specifications designed by the Trusted Computing Group (TCG) to enhance security and privacy of devices with minimal silicon requirements.**

***Index Terms*—IoT, DICE, MUD, Remote Attestation, Implicit Attestation, Authentication, Authorization, Trusted Computing**

I. Introduction

IoT technology is rapidly and continuously expanding, with countless service providers, different platforms, and millions of new devices becoming available yearly. This is because IoT devices have the potential to improve our lives in every field, enabling more effective monitoring of production (Smart Manufacturing), and logistics (Smart Logistics) systems, new generation healthcare (Smart Health), urban services that improve the quality of life (Smart City), safer streets thanks to autonomous vehicles, greener and more comfortable houses (Smart Home), and more. On the other hand, the advent of the IoT has broadened the horizon of threats, that concern not only risks for data and IT systems, but also attacks to physical infrastructures that may put people's safety at risk. Cybersecurity related to IoT ecosystems is a fundamental aspect to avoid data loss and/or compromise, secure the business from huge economic losses, guarantee the safety of individuals. However, most IoT devices have little or no security mechanisms at all, so that, on average, they are attacked within 5 minutes of being connected to the network [1].

The seriousness of the situation emerged in October 2016 with the Mirai botnet attack [2], when hundreds of thousands of low-cost devices were used to perform a Distributed Denial of Service (DDoS) against the dynDNS provider. For eight hours the clients of Amazon, Twitter, Netflix, Wall Street Journal, and more were unable to use their services.

There are various reasons that make IoT ecosystems vulnerable. The number of connected devices inevitably increases the attack surface of IT systems, making them more exposed to breaches by adversaries. The ubiquity of IoT devices, often deployed in uncontrolled environments, makes it difficult to protect them from physical attacks. The great heterogeneity of IoT devices creates difficulties in designing security solutions suitable to protect different device categories. The software deployed on IoT devices often remains unpatched for long periods of time, so any discovered vulnerability is easily exploited by attackers for the creation of botnets. Finally, to reduce production costs and/or to have very small devices (e.g. wearable devices), IoT devices often have a limited amount of resources, with stringent constraints on memory, power, and computational capacity. This challenges the creation and maintenance of an optimal security posture over time.

In a scenario where IoT devices become increasingly intertwined with our lives, we must be confident that they are trustworthy. *Remote Attestation* (RA) techniques were developed to allow an external entity (the Verifier) to check that a remote device (the Attester or Prover) has not been compromised by an attacker. The RA relies on the presence, in the attesting system, of a *trust anchor* which allows to measure the integrity status of the device and sign reports containing the integrity evidence, reliable even when the platform has been compromised.

In this paper, we propose the design of a security mechanism capable of guaranteeing authenticity and integrity to the hardware and software deployed in low-end IoT devices. This category of devices is challenging to treat for several reasons: it has too limited resources to run traditional OSes such as Linux or Windows 10 IoT Core; it may use a single CPU protection mode; it may lack a Memory Management Unit (MMU); it can not rely on fully-fledged TEE environments

This work has received funding from the SPIRS (Secure Platform for ICT Systems Rooted at the Silicon Manufacturing Process) project with Grant Agreement No. 952622 under the European Union's Horizon 2020 research and innovation programme. This work was also partially supported by the project SERICS (PE00000014) under the NRRP MUR program funded by the European Union - NextGenerationEU.

or dedicated security co-processors, like the Trusted Platform Module (TPM). The proposed scheme targets constrained RISC-V devices and aims to build a trusted anchor for them with minimal silicon requirements, by relying only on Read Only Memory (ROM) and the Physical Memory Protection (PMP) primitive, a security extension that allows the creation of memory barriers by restricting access to certain physical memory regions defined in PMP entries. Moreover, we implement the trusted anchor following *Device Identifier Composition Engine* (DICE) [3], a specification proposed by the Trusted Computing Group (TCG) for providing a strong device identity even to platforms without the TPM. Our proposal also integrates the *Manufacturer Usage Description* (MUD) [4] specification within the protocol that allows the IoT device to access the public network, providing the overall IoT system with a high level of protection from co-optation into botnets.

The rest of the paper is organized as follows. Section II describes the technologies used in the solution. Section III presents our proposed security solution. Section IV reviews some related works. Finally, Section V concludes the paper and proposes the work that needs to be done in the future.

II. Background

A. Device Identifier Composition Engine (DICE)

The Device Identity Composition Engine [5] specification has been proposed by the Trusted Computing Group (TCG) in the context of IoT devices, which typically have very limited resources and no presence of special hardware for attestation.

The idea is based on a *Unique Device Secret* (UDS), which is a statistically unique value that identifies the platform and permits to derive all other values used during the process. This secret must be kept in secure storage, where it cannot be rewritten, and it should be at least of 256 bits.

At the reset of the platform, the *Device Identity Composition Engine* (DICE), which is the only component that can access the UDS, uses it, in combination with the measurement of the first mutable code running on the platform, to generate the *Compound Device Identifier* (CDI), as depicted in figure 1. The CDI can also include measurements of configuration data that influence the execution of the first mutable code, and/or hardware information, and must be calculated in a way that makes it infeasible to derive back the UDS; this can be accomplished using a secure hash algorithm (H) [6], or using a Hash-based Message Authentication code (HMAC). The first value needed is the first mutable code measurement:

$$M = \mathbf{H}(first\ mutable\ code)$$

that will be used to compute the CDI. Using a secure hash algorithm, the CDI can be computed by hashing the concatenation of the UDS and the first mutable code measurement:

$$CDI = \mathbf{H}(UDS \mid\mid M)$$

Using HMAC, accordingly to [7], which is a more sophisticated alternative, the CDI can be computed as follow:

$$CDI = \mathbf{H}((UDS \oplus opad) \mid\mid \mathbf{H}((UDS \oplus ipad) \mid\mid M))$$

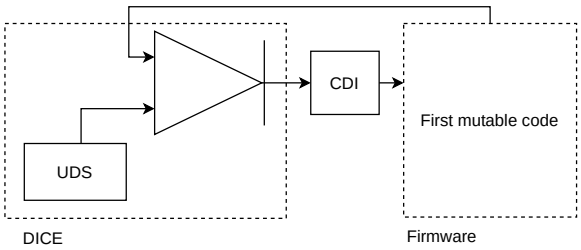

Fig. 1. CDI derivation

where *opad* is a data block composed by the repetition of the $5c_h$ character as many times as it is necessary to reach the block size of the chosen hash function, and *ipad* is built in the same way using the 36_h character. This process permits to obtain a different CDI if the first mutable code is in any way modified or compromised, and the malicious code cannot have available the precedent CDI.

The DICE can be a one-time programmed component or can support the possibility to be securely upgradable by the DICE manufacturer.

Once the CDI has been calculated, it is passed to the mutable code, which also takes control of the platform, and can use this value to derive keys, for example using a Key Derivation Function (KDF).

B. DICE Layering Architecture

The DICE specification can be extended to the case of a multi-layered Trusted Computing Base (TCB). This architecture [3] allows a lower TCB layer to provision trusted functionality to a higher TCB level. The process starts from the DICE hardware Root of Trust, which in this architecture is implemented in the base hardware layer, which is assumed to be in a trustworthy state. By convention, the layer above the DICE implementation is layer 0, and it receives the CDI from DICE. Once a layer is assumed as trusted, it is possible to pass the execution to the next layer.

In order to build a chain of trust, each TCB layer must have trusted access to a set of TCB capabilities, which have to be protected in terms of the execution environment. These capabilities have the purpose to produce some values that are used as the identity of each TCB layer. At each step, there is the need to compute:

TCB Component Identifier (TCI): This value permits to describe a specific TCB component, performing a hash on the code that will be executed by the component and could include some other information, such as vendor name, version, etc;

Compound Device Identifier (CDI): This is the value received from the previous layer, computer combining, using a one-way function (OWF), the CDI of the previous layer and the measurement (TCI) of the current layer (figure 2). The only exception is for the DICE, which uses the UDS, instead of a CDI, because in that case there is no previous context;

Fig. 2. CDI derivation in the case of architecture composed by three layers.

Fig. 3. Certi cate hierarchy with embedded CA

One-Way Function(OWF): This is a pseudo-random function, according to [8], which accepts seed and data values, that in this case are respectively the CDI of the layer and the TCI of the next layer.

C. Certi cation and Key Types

Using a DICE layered architecture, it is possible to link a TCB layer with the next one, issuing a certi cate or a token for the keys of the next layer. The certi cation can be obtained in two different ways:

1) a TCB layer generates keys for the next layer, then issues the certi cates for these keys, and nally passes the keys and the certi cates to the next layer;
2) another possibility is that a TCB layer generates its own keys and then sends a certi cate request to the TCB layer below. This layer will create the certi cate and will send it to the upper layer that requested it.

The generated keys can be different, based on their specific purpose: they can be symmetric or asymmetric.

Asymmetric Keys

Attestation keys: These are the keys used, during the process of remote attestation, to sign integrity measurements. They are used to sign only data known by the TCB, and they cannot sign opaque data from outside the TCB layer;
Identity Keys: These are keys used to sign authentication challenges, in this case, they may sign opaque data;
Embedded Certi cate Authorities (ECA) Keys: An ECA key is used by a TCB component to issue certi cates for keys belonging to the current layer or higher TCB layer. Also in this case the key can sign only data known by the TCB layer;
DeviceID Key: This asymmetric key is a long-lived key certi ed by the manufacturer, computed by the DICE and it depends on the UDS and measurement of layer 0. It can be used to sign certi cates of keys belonging to a higher TCB layer, so it behaves as an ECA Key, in addition, it can be used to sign certi cates containing some attestation evidence, so it can also behaves as an Attestation Key;
Alias Key: This key is derived at the last TCB layer using its CDI. It is an Attestation Key because is used to sign integrity measurements of the top-level device rmware.

D. Credential Types

These credentials are conformed with the IEEE802.1AR standard [9]:

Initial Device Identi er (IDevID): This credential contains a DeviceID and is typically generated by layer 0, which has the constraint to be immutable or upgradable following a process controlled by the device manufacturer.
Local Device Identi er(LDevID): This credential is typically created at deployment time on the owner's network, by a layer after layer 0.

E. Layered Certi cation

On a DICE layered architecture is possible to expand a CA hierarchy starting from the Device Manufacturer certi cate of the DICE Hardware Root of Trust (HRoT). It is possible to associate an Embedded Certi cate Authority (ECA) to a TCB layer (gure 3), which is able to issue certi cates for a higher layer's keys.

In the layered certi cation architecture, each layer has computed its identity key, and each key will require a certi cate that can assert the trustworthiness of the key. The certi cate related to layer 0 is issued by the device manufacturer. Each TCB layer contains an ECA, which can issue certi cates for higher layer's key, for this reason, a consumer of an ECA-issued certi cate, to verify it, will have to trace the trust dependencies through TCB layers to the device manufacturer.

F. RISC-V Physical Memory Protection (PMP)

RISC-V [10] is an open-source Instruction Set Architecture (ISA) that provides a set of features to enhance the security and reliability of several kinds of devices. One of these features is the Physical Memory Protection (PMP) [11]. The PMP provides a hardware mechanism to protect memory regions (up to 16 regions) from unauthorized access to read or write data. It is implemented using a set of hardware registers where is possible to de ne the boundaries of the memory regions to protect and it can be exploited only in Machine mode (M-mode). Each region is identi ed specifying the start address and the size. For each region is possible to specify the access privileges (read, write, execute) to grant permission to the other execution modes: Supervisor mode (S-mode) and User mode (U-mode), and in case of a PMP violation, there will be a CPU exception. It is possible to de ne memory regions

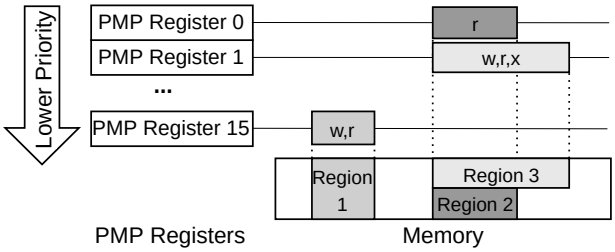

Fig. 4. Example of PMP registers' priority

that overlap with each other because each PMP register has a priority where the highest priority is owned by the register 0, so it will be applied the permissions related to the highest priority PMP register that refers to that specific memory region (figure 4). It is possible to enable an execute-only option, for a specific memory region, in order that even when running in M-mode, is not possible to modify that particular region, which can be useful to prevent some possible modifications that can cause runtime errors.

G. Manufacturer Usage Description (MUD)

Manufacturer Usage Description (MUD) [4] is a recently developed standard that provides a framework for describing and classifying the network behavior of Internet of Things (IoT) devices. It was developed by the Internet Engineering Task Force (IETF) to address the security and privacy concerns associated with IoT devices. MUD enables network administrators to enforce security policies by identifying and controlling the network behavior of IoT devices.

MUD is a JSON file that provides information about the type of device, the ports it uses, the protocols it supports, and the services it provides. The MUD file is generated by the manufacturer and is digitally signed to ensure its authenticity. The MUD file is then distributed to network administrators who can use it to configure their network security policies.

MUD enables network administrators to create policies that allow or deny specific types of traffic to and from IoT devices based on their behavior. This is particularly useful in situations where the network administrator may not have complete knowledge of the devices on their network. By using MUD, network administrators can ensure that IoT devices are only able to access the resources they require and that they are unable to perform any malicious actions. The component which performs these actions is the MUD Controller, which analyzes the MUD file related to an IoT device and manages access policies for it.

One of the main advantages of MUD is its simplicity. The MUD file is a standardized format that can be easily generated by manufacturers and read by network administrators. This simplicity enables widespread adoption of the standard, which is essential for ensuring the security and privacy of IoT devices.

III. Design of the Solution

This work contributes to the IoT security area by proposing a solution that combines the verification of the hardware and firmware identity of an IoT device and the enforcement of network security through MUD specification. The following sections describe the threat model and the design of the proposed solution in detail.

A. Threat Model

We consider a privileged software attacker who has the goal to gain persistent control of the IoT device by exploiting vulnerabilities present in the code. The attacker thus manages to arbitrarily compromise portions of the memory not protected by hardware-assisted mechanisms, i.e. ROM and PMP primitive. Moreover, the attacker may operate in M mode, which is the only execution mode supported by some low-end IoT devices. We rely on the correct implementation of RISC-V PMP primitive and ROM unit to ensure the integrity and confidentiality of the device's DICE trust anchor, containing the *Unique Device Secret* (UDS), the *Device Root Key* (DRK) certificate and the *Core Root of Trust for Measurement* (CRTM) of the device. In this context, we do not consider hardware attacks aimed at altering the execution flow and bypassing PMP protection, such as physically altering system conditions through fault injection.

B. Proposed architecture for the IoT device

The internal architecture of the IoT device requires the implementation of a DICE-based trust anchor, residing in the ROM of the device, while the user function, which represents Layer 0 first mutable code, is mapped in RAM or resides in Flash memory, as shown in the figure 5. At power-on, code in ROM executes basic hardware configuration routines, after which it sets a PMP entry to assign execute-only permissions to the physical memory area in ROM containing the DICE code. Furthermore, this entry is configured so that it can no longer be modified after being initialized. In this way, the integrity of the functions contained in the root of trust (RoT) (i.e. CRTM, IDevID key generation, Authentication) are guaranteed by the ROM, while the confidentiality of the UDS, present in the RoT, is guaranteed by the PMP entry which

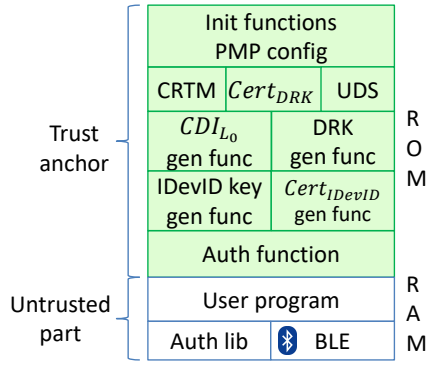

Fig. 5. ROM and RAM/Flash configuration of the IoT device.

does not allow to read the memory area of the RoT. Upon initialization of the PMP entry, DICE, using a Key Generation Function (KGF), generates an asymmetric key pair from the UDS, which is a statistically unique value configured by the device manufacturer or derived from a Physical Unclonable Function (PUF). The generated key is hardware unique and is called *Device Root Key* (DRK) in this work. A trusted manufacturer certificate authority (CA) can sign a certificate for the DRK (e.g. an X.509 certificate), thus ensuring the reliability of the RoT in the DICE architecture.

The DICE Core maps the user function into RAM and calculates the IDevID key, which represents the hardware and firmware identity of the device and can be used to sign authentication challenges during network authentication protocols, e.g. TLS extension. To generate the IDevID key, it performs the following steps:

the CRTM calculates a digest on the physical memory region containing the user program;
the measurement obtained from the CRTM is combined, via an OWF, with the UDS to obtain the CDI_{L0} relating to the user program;
the CDI_{L0} is used as the seed of a key generation function to derive the IDevID key.

Then, DICE generates a certificate, behaving as an ECA, (e.g., via X.509) for the IDevID public key (IDevID.pk), signed with the private part of the DRK (DRK.sk), inserting in the certificate:

the measure calculated by the CRTM;
the URL of the IoT device manufacturer server from which to download the trusted reference measurements;
the URL of the IoT device manufacturer server from which to download the MUD file.

Finally, DICE copies the certificate for the created IDevID key into the user function's memory, before passing the control over to it. Figure 6 represents at a high level the steps performed by DICE when booting the IoT device.

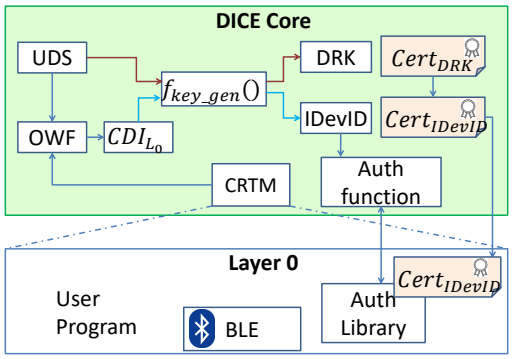

Fig. 6. IoT device internal architecture. DICE Core represents the Trust Anchor, and Layer 0 represents untrusted components.

The authentication library, embedded in the user program, will now be able to rely on the IDevID key certificate, generated by the RoT, to provide its identity to the end-points with which it has to exchange data. However, since

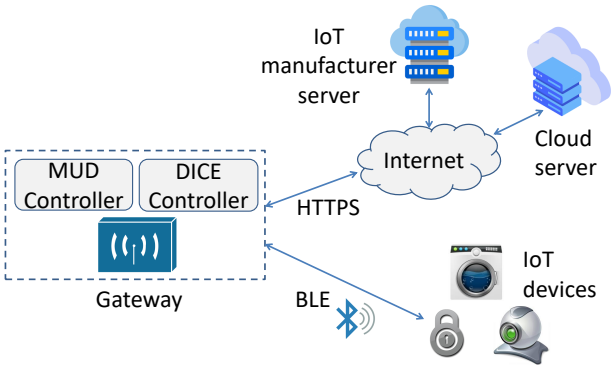

Fig. 7. Network architecture of an IoT system.

the user program does not know the IDevID.sk, it will invoke an *Authentication Function* embedded in the device's RoT to decrypt opaque challenges from remote-peers and prove that the device has a trusted identity. Once invoked, the authentication function regenerates the current IDevID key, using a new measurement of the user program performed by the CRTM: in this way, if an adversary has managed to compromise the user program mapped in RAM, the generated IDevID key will be different from the one certified at boot-time and the remote-peer automatically knows that it cannot trust the device it is communicating with.

C. Network architecture of the IoT system

IoT ecosystems are composed of a large variety of elements, including embedded devices and sensors, which are responsible for acquiring data, gateways, which aggregate the data acquired by the sensors to transmit them to other devices over the network, network communication protocols, which deal with connecting the various devices present in the IT infrastructure, cloud infrastructures, which deal with cleaning up data and carrying out analysis processes, made available in the form of reports to end users. IoT platforms can assume various configurations, ranging from managing a few devices at a household level to an incredibly large number, such as in a smart city or smart factory.

Figure 7 shows a typical IoT architecture, made up of a set of devices connected via low-power radio technology to a gateway or access point, which provides them with connectivity to the public network. Radio technology includes short-range protocols (e.g., Bluetooth Low Energy (BLE), Z-Wave), suitable for systems with a limited number of devices, such as smart homes, medium-range protocols (e.g., ZigBee), used in industrial environments, and long-range protocols (e.g., LoRaWAN, Sigfox), which guarantee the connectivity of millions of low-power devices, as happens in Smart Cities. The gateway then communicates with the IoT server in the cloud, typically using REST APIs. Typically, IoT platform components trust each other, creating a large attack surface within the system.

The architecture proposed in this work allows the gateway to perform a *trusted identity verification* of the IoT device,

both before granting it access to the public network and subsequently, during its operation. Furthermore, thanks to the integration with the MUD specification, the gateway has the possibility of configuring a firewall that allows only the traffic permitted by the device manufacturer to pass through.

The steps involving the proposed solution are described in detail below. Once started, the IoT device communicates with the gateway, providing it with its identity defined by the IDevID certificate. The gateway passes this certificate to the DICE Controller, a sub-component responsible for carrying out checks regarding the trustworthiness of the IoT device. The DICE Controller verifies the validity of the certificate, checking that it has been signed with a DRK key certified by a trusted device manufacturer. The DICE Controller then extracts the measurement of the user program and the URL of the IoT manufacturer server from the certificate, downloads the trusted reference measurements from the server and verifies that the measurement of the user program running on the IoT device matches one of the golden reference measures. If this check fails, the DICE Controller denies the IoT device access to the public network, keeping it isolated and without the possibility of causing damage to the outside; otherwise, the DICE Controller extracts the URL corresponding to the MUD file from the certificate and passes it to the MUD Controller. The MUD Controller downloads the MUD file from the IoT manufacturer server, interprets its policies, and configures a firewall that only allows traffic authorized by the device manufacturer to cross the gateway and access the public network.

An IoT device that has gained network connectivity from the gateway is able to establish trusted channels with the cloud server based on the IDevID key. In particular, the server is able to carry out the same checks on the trustworthiness of the IoT device performed by the gateway. Moreover, since the IoT device regenerates the IDevID key when it is to be used to decrypt authentication challenges, the cloud server is able to verify that the device it is communicating with is really trusted since it has not been compromised by an attacker.

Furthermore, the gateway can periodically challenge the device to prove its identity and, if this is modified with respect to the one established in the IDevID certificate, the gateway revokes the device's permissions and isolates it from the network so that, if it has been co-opted into a botnet, it is no longer able to participate in a DDoS attack. In this way, we use the authentication response as an implicit statement of the integrity status of the device.

The proposed scheme is able to verify the trustworthiness level of unknown devices, since it is not based on the a priori knowledge of trusted information, such as a whitelist of reference measures or public keys of IoT devices allowed in the platform, but obtains this information by extracting it from certificates, relying on a Public Key Infrastructure (PKI). The proposed architecture is therefore suitable for use in dynamic environments such as smart homes or smart cities, where unknown devices can enter the IoT network and need to be authorized before gaining access to platform resources.

IV. Related work

C. Shepherd et al. propose LIRA-V [12], an explicit remote attestation scheme between constrained RISC-V devices, based on the PMP security primitive. Differently from this work, our scheme allows to realize implicit remote attestation between unknown nodes, without needing to know a-priori the trusted status of a node.

Regarding DICE specification, some implementations have already been proposed in literature. Jäger et al. [13] propose a symmetric attestation protocol based on DICE, and evaluate its applicability to secure commercial off-the-shelf (COTS) IoT devices. Choi et al. [13] propose a method based on DICE to detect and identify faulty IoT devices through the extraction of their context, known as the sensor correlation and the transition probability between sensor states.

V. Conclusion

This paper proposed an architecture to secure IoT platforms containing RISC-V constrained devices. Our proposal defines a trust anchor for the IoT device, based on PMP and ROM hardware mechanisms, without requiring the presence of TEEs or other secure elements in the device. We describe a protocol that allows the creation of trusted channels between IoT devices and cloud servers, relying on an implicit attestation mechanism of the trust state of the IoT device. The device attests its trustworthiness by demonstrating that the IDevID key, which represents its hardware and firmware identity, has not been changed since boot.

Future work concerns the in-field evaluation of the performance of the proposed solution, the implementation of the presented schema using lightweight cryptography, and the evaluation of post-quantum cryptosystems that can be adopted in constrained devices.

References

[1] Netscout, "Down of the terrorbit era." https://www.netscout.com/sites/default/files/2019-02/SECR_001_EN-1901 - NETSCOUT Threat Intelligence Report 2H 2018.pdf.

[2] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[3] TCG, "DICE Layering Architecture, version 1.0," July 2020.

[4] E. Lear, R. Droms, and D. Romascanu, "Manufacturer Usage Description specification." RFC-8520, May 2019.

[5] TCG, "Hardware requirements for a device identifier composition engine," 2018.

[6] Q. Dang, "Secure Hash Standard (SHS)." NIST FIPS.180-4, 2012.

[7] Q. Dang, "The keyed-hash message authentication code (hmac)." NIST FIPS.198-1, 2008.

[8] R. D. Elaine Barker, Lily Chen, "Recommendation for key-derivation methods in key-establishment schemes." NIST SP-800-56, 2020.

[9] IEEE 802.1 working group, "IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity," *IEEE Std 802.1AR-2018 (Revision of IEEE Std 802.1AR-2009)*, pp. 1–73, 2018.

[10] A. Waterman, *Design of the RISC-V Instruction Set Architecture*. PhD thesis, EECS Department, University of California, Berkeley, Jan 2016.

[11] A. Waterman, K. Asanovic, and J. Hauser, "The RISC-V instruction set manual," *Volume II: Privileged Architecture*, 2021.

[12] C. Shepherd, K. Markantonakis, and G.-A. Jaloyan, "LIRA-V: Lightweight Remote Attestation for Constrained RISC-V Devices," in *2021 IEEE Security and Privacy Workshops*, pp. 221–227, may 2021.

[13] L. Jäger, R. Petri, and A. Fuchs, "Rolling dice: Lightweight remote attestation for cots iot hardware," in *12th International Conference on Availability, Reliability and Security (ARES'17)*, August 2017.