Hardware security for Internet of Things identity assurance

André Cirne*‡, Patrícia R. Sousa§, João S. Resende*¡, Luís Antunes*†

*Dep. de Ciência de Computadores, Faculdade de Ciências, Universidade do Porto, Rua do Campo Alegre s/n, 4169-007 Porto, Portugal †TekPrivacy, Lda; Faculdade de Ciências da Universidade do Porto, Rua do Campo Alegre 1021-1055; 4160-007 Porto, Portugal ‡INESC TEC, Campus da Faculdade de Engenharia da Universidade do Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal §INSIGHTSEC, Lda; Aveiro, Portugal

***Abstract*—With the proliferation of Internet of Things (IoT) devices, there is an increasing need to prioritize their security, especially in the context of identity and authentication mechanisms. However, IoT devices have unique limitations in terms of computational capabilities and susceptibility to hardware attacks, which pose significant challenges to establishing strong identity and authentication systems. Paradoxically, the very hardware constraints responsible for these challenges can also offer potential solutions. By incorporating hardware-based identity implementations, it is possible to overcome computational and energy limitations, while bolstering resistance against both hardware and software attacks.**

This research addresses these challenges by investigating the vulnerabilities and obstacles faced by identity and authentication systems in the IoT context, while also exploring potential technologies to address these issues. Each identified technology underwent meticulous investigation, considering known security attacks, implemented countermeasures, and an assessment of their pros and cons. Furthermore, an extensive literature survey was conducted to identify instances where these technologies have effectively supported device identity.

The research also includes a demonstration that evaluates the effectiveness of hardware trust anchors in mitigating various attacks on IoT identity. This empirical evaluation provides valuable insights into the challenges developers encounter when implementing hardware-based identity solutions. Moreover, it underscores the substantial value of these solutions in terms of mitigating attacks and developing robust identity frameworks.

By thoroughly examining vulnerabilities, exploring technologies, and conducting empirical evaluations, this research contributes to understanding and promoting the adoption of hardware-based identity and authentication systems in secure IoT environments. The findings emphasize the challenges faced by developers and highlight the significance of hardware trust anchors in enhancing security and facilitating effective identity solutions.

***Index Terms*—IoT, device's identity, identity, hardware-based identity, hardware trust anchors, hardware attacks**

I. Introduction

The Internet of Things (IoT) is a vast ecosystem of interconnected devices that rely on the Internet to share data. Almost any device can be connected to this network, including smartphones, wearables, motion sensors, cars, and smart home appliances. As the number of IoT devices connected to the Internet continues to grow [1], they are becoming an integral part of our daily lives. Moreover, many industries are increasingly using them to replace humans in factories, farms, and other jobs [2]. This trend shows no signs of slowing down; in fact, it is expected to accelerate further due to the emergence of advanced technologies such as 5G [3], Big Data [4], and Fog Computing [5]. These technologies enhance connectivity to end devices, increase network bandwidth, and provide more storage and computing resources, making it easier to handle the massive amounts of data generated by IoT devices. Consequently, the number of IoT devices is likely to continue its significant growth in the coming years [3].

As the number of IoT devices increases, the need for robust security policies and controls throughout the entire IoT life cycle also grows [6]. IoT devices are known to be vulnerable, and developing effective security solutions is considered one of the foremost research challenges in this field [7], [8]. This problem primarily arises from a general lack of security standards and the pursuit of cost-effective systems, which often leads developers to deprioritize security during the development process [9]–[11].

Among the various security challenges in this domain [12], [13], device identity stands out as one of the most critical factors for establishing a secure system. Authentication is the process of verifying the identity or origin of an object or person, which is particularly relevant in the context of IoT. Thus, Identity Management (IdM) forms the foundation for secure authentication methods. By establishing a reliable and trustworthy IdM system, IoT devices can be securely authorized and authenticated, facilitating their communication and data exchange with other devices while mitigating the risks of unauthorized access or data breaches. IdM entails the creation, maintenance, and control of digital identities for IoT devices, users, and applications. Without these foundational elements, designing a secure IoT system becomes unattainable, as it hinders the ability to control access to resources within a system and ensure the accuracy of information received from a device [14]. Despite these facts, the previously mentioned issues hinder the implementation of identity and authentication mechanisms. This often leads to custom solutions without peer review, reliance on slow or outdated cryptographic algorithms, and, consequently, suboptimal security.

Moreover, IoT devices are more susceptible to physical attacks than other devices [15], and this has significant implications for the development of IoT authentication and identity systems. Unlike computers and servers, IoT devices are often

Manuscript received April 19, 2021; revised August 16, 2021.

deployed in unprotected locations where attackers can gain unrestricted physical access. Consequently, IoT devices are more prone to physical attacks, such as tampering, theft, or destruction, which can jeopardize their security and the entire IoT network. Given this challenge, it is evident that safeguarding IoT devices against physical attacks is essential to ensure their security and reliability [15].

IoT devices consist of two primary components: software and hardware. The software implements the device's logic, while the hardware supports its execution and facilitates interactions with the physical world. Therefore, there is an intrinsic relationship between these components, with the software running on top of the hardware. When it comes to identifying and addressing vulnerabilities in IoT devices, a range of tools and techniques are available to handle issues in the software component. These tools include code auditors, fuzzers, debuggers, and static analyzers, which can help detect and remediate software related issues. However, addressing hardware vulnerabilities can be more challenging due to the limited availability of tools for this purpose. As a result, hardware vulnerabilities may take longer to be detected and resolved compared to their software counterparts. Despite this challenge, IoT manufacturers and developers must remain vigilant in identifying and mitigating both software and hardware vulnerabilities to ensure the security and reliability of their devices [16].

Attackers employ both software and hardware attacks to exploit these vulnerabilities. However, owing to the intricate relationship between software and hardware, the nature of an attack may not necessarily align with the nature of the target. For example, a hardware attack may target software, and vice versa. Furthermore, many software attacks can be executed remotely, while hardware attacks require physical access to the device [17]. Consequently, addressing hardware attacks must be a priority for manufacturers, given the susceptibility of IoT devices to physical attacks.

In contrast, a Root-Of-Trust (RoT) is a process that can safeguard the identity of an IoT device from potential software and hardware attacks. A RoT begins with an immutable (unchangeable) hardware identity deeply integrated into the IoT device. Essentially, it serves as the foundation for a chain of trust, providing an unalterable starting point that can be relied upon to authenticate subsequent actions and transactions. While a hardware RoT can be exceptionally valuable for highly sensitive applications, providing a hardware-based trust anchor is also a fundamental element for any secure IoT device, since its inclusion can deliver a higher level of assurance and resilience compared to purely software-based security mechanisms.

Despite the widespread belief that a RoT can provide device security against software and hardware attacks, research suggests that this assumption may not always hold true [18]. While hardware-based RoT are generally considered robust and resistant to tampering, they can still be vulnerable to sophisticated hardware attacks capable of compromising the entire chip. These vulnerabilities enable attackers to bypass identity authentication and data encryption, potentially leading to the theft of valuable intellectual property and causing significant application security issues. Therefore, while hardware-based roots of trust offer certain security advantages, their design must account for sophisticated attacks and ensure adequate protection against them. Consequently, there is a growing demand for the implementation of hardware-level security features [16], [18]–[20].

A. Related research overview

Several literature reviews have been published on hardware security and identity, and some of them share similarities with our work. For instance, Ehret et al. [21] conducted a survey of hardware-based security techniques applicable to IoT devices. Their research delves into various components of an IoT device, such as its processing unit or memory, and addresses the hardware security threats associated with these components, along with potential mitigations, encompassing both hardware and software countermeasures.

Hu et al. [22] pursued a research direction similar to that of Ehret et al., but their work stands out by providing a comprehensive review of security tools designed to validate the security of devices. Specifically, Hu et al. focused on evaluating tools capable of scrutinizing device security by analyzing information flows within the hardware board. Moreover, they emphasized the importance of verifying whether the device's implementation aligns with its intended design.

Michailidis et al. [23] also conducted a survey of hardware attacks concerning device security and listed possible countermeasures, including those based on machine learning. There are also works that solely analyze hardware attacks without encompassing countermeasures [24]. Akter et al. [25] analyzed current trends in hardware security from both physical and software perspectives. Their work explores countermeasures for these attacks, with a particular focus on PCB design techniques aimed at minimizing associated risks. Yang et al. [26] reviewed different technologies that can be used to support the identification and authentication of IoT devices. They explored how each technology can serve as a building block for new systems and examined potential security attacks on these technologies. Although Yang et al.'s research is not specifically focused on identity, it acknowledges the significance of device identity for overall device security and highlights some technologies that can be utilized for this purpose. In comparison to these works, our analysis focuses specifically on hardware security for the purpose of identifying and authenticating IoT devices via hardware. Consequently, we present a threat analysis tailored to this specific purpose and establish the relationship between these threats and how they can compromise the device's identity.

Cheruvu et al. [27] presented a comprehensive guide to IoT system security in their work, titled "Demystifying Internet of Things Security: Successful IoT Device/Edge and Platform Security Deployment". Their coverage encompasses a broader range of topics compared to our work. They provide practical advice on deploying security solutions for IoT devices and platforms, offer threat modeling for specific IoT application domains, and compare security features of various IoT Operating Systems (OSs). While both Cheruvu et al. and our

work emphasize the significance of hardware-based solutions and how they can leverage identity in IoT, there are notable distinctions. Cheruvu et al. dedicate a chapter entitled "Base Platform Security Hardware Building Blocks" to an exploration of technologies involved in securing an IoT device. They also discuss the importance of identity for secure systems and address the threats these technologies may mitigate. However, unlike our work, Cheruvu et al. restrict this discussion to Intel technologies and product lines, focusing on software threats while omitting consideration of hardware attacks. This exclusive focus on Intel products limits the applicability of their findings since IoT devices commonly employ alternative Central Processing Unit (CPU) architectures like ARM or RISC-V for their energy efficiency and cost-effectiveness — architectures not included in Intel's product lineup. In contrast, our research adopts a vendor and CPU architecture agnostic approach. Furthermore, our emphasis centers on analyzing hardware threats rather than software threats, providing a distinct perspective on device security.

Shepard et al. [28] delved into hardware security by reviewing technologies that facilitate safe and reliable execution in IoT systems. Throughout their analysis, they established a threat model and evaluation criteria specifically tailored to the IoT use case, enabling them to compare various technologies. Their threat model encompassed adversaries with physical access to the device. Furthermore, the criteria used to assess different technologies considered resilience against hardware attacks, such as protection against fault injections, secure storage, and tamper-resistant hardware. However, it is important to note that the article did not explore these attacks in detail; instead, they were merely mentioned as characteristics that these technologies must possess to support secure and trusted execution in IoT.

Compared to our work, Shepard et al. pursued a broader research question: how to achieve secure and trusted execution in IoT. Consequently, many of the technologies analyzed in their work are also mentioned in ours. Nonetheless, our analysis differs significantly as we concentrate on identity and the specific threats to hardware that can compromise it. Additionally, we thoroughly explored different attacks that can target these technologies and developed a demonstration environment to illustrate some of these attacks along with potential mitigations.

In conclusion, our study has centered on hardware security with a specific focus on the identification and authentication of IoT devices through hardware-based methods. We conducted a thorough threat analysis tailored to this specific objective, uncovering potential risks and their implications for compromising device identity. Furthermore, we delved into various hardware-based technologies that can facilitate the implementation of robust device identity measures. Throughout our review, we diligently examined the current state of research and found no other work of significant note in this specific area. What distinguishes our work is the comprehensive assessment of threats that target device identity and the corresponding hardware technologies essential for supporting identity implementation. Lastly, to illustrate the practical implications of our research, we conducted an experiment using an actual device to demonstrate how its identity can be targeted and to propose potential mitigation strategies against such attacks.

B. Case study

This work employs a case study approach to explore various aspects of our research, with a specific focus on smart meters. Smart meters serve as measuring instruments for household energy consumption and are typically owned by energy companies, with a paramount requirement that they remain tamper-proof.

Smart meters, also known as Advanced Metering Infrastructure (AMI), offer a range of benefits, including the ability to remotely measure household energy consumption and manage the unit. This management includes services such as disconnection and power supply adjustments, making them vital for utility companies. Additionally, smart meters play a critical role in the implementation of smart grids by providing real-time data that enhances energy distribution and informed decision-making [29].

Communication between smart meters and the broader grid can be established through two distinct approaches: wireline and wireless. Wireline technologies utilize cable infrastructure to create bidirectional communication channels between smart meters and the utility company's infrastructure. This technology can use power lines as a communication medium or employ dedicated cables for data transmission, such as fiber optics or telephone infrastructure. Conversely, wireless technologies rely on radio communication, with various solutions available for implementing wireless communication, including cellular communications and mesh networks [29].

For our case study, we will consider a wireless mesh network as the communication infrastructure for our smart meter. This network comprises multiple meters that transmit measurements and relay messages from neighboring meters to the nearest concentrator. The concentrator, functioning as a gateway for the mesh network, is connected to the internet and ensures that the meter's traffic reaches the utility company's back office [30] (see Figure 1).

C. Contributions

The objective of our research is to offer a comprehensive analysis of hardware trust anchors and their potential to support the implementation of identity systems for IoT devices. In summary, our research delivers the following key contributions:

- Threat analysis: We identify physical risks to IoT identity, define security assets and goals, describe threat actors, and outline potential threats and corresponding countermeasures. This in-depth analysis aids in designing and implementing effective hardware-based identity solutions.
- Technology analysis: We scrutinize various technologies that can bolster device identity, evaluating their advantages and disadvantages, security concerns, and providing examples of their use. This analysis offers valuable insights into selecting and deploying suitable hardware trust anchors for IoT devices.

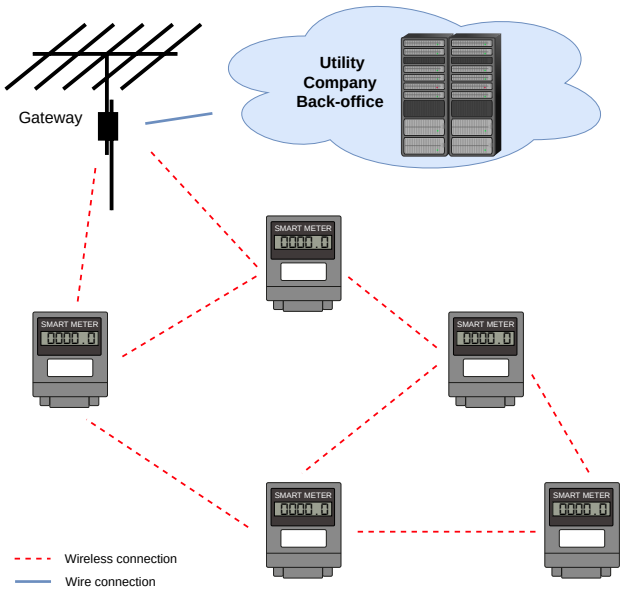


Fig. 1: Advanced metering infrastructure network

- Experimental evaluation: We carry out experiments to assess the effectiveness of hardware trust anchors in mitigating different attacks on IoT identity. This evaluation furnishes empirical evidence regarding the challenges faced by developers when implementing hardware-based identity solutions and underscores their value in countering attacks and establishing effective identity solutions.
- Discussion of challenges: We discuss the challenges associated with the adoption of hardware trust anchors, encompassing both technical and economic factors. Additionally, we highlight potential avenues for future research that can address these challenges and enhance the security of IoT identity systems.

Overall, our research provides a comprehensive and practical guide for designing and implementing hardware trust anchors for IoT identity systems. The ultimate aim is to enhance the security and privacy of IoT devices and their users.

D. Outline

The structure of the rest of this article is as follows:

Section II briefly introduces identity and authentication and their current state of development within the context of IoT. In this section, we analyze the distinctions between traditional and IoT-based IdM, enumerate various research challenges found in the literature related to device identity and authentication processes, and provide a comparative analysis between two identity systems - one traditional and the other purpose-built for IoT. Section III explores the selection of the appropriate hardware trust anchor technology for each application. To do so, we elaborate on a threat analysis focused on the physical risks to device identity and analyze various technologies that can be employed to support it. Building upon the knowledge presented in the preceding sections, Section IV demonstrates how hardware can effectively support device identity. In Section V, we reflect on the current limitations and explore future research directions that we encountered throughout our work. Section VI provides a summary of our research.

Figure 2 summarizes the outline of this work.

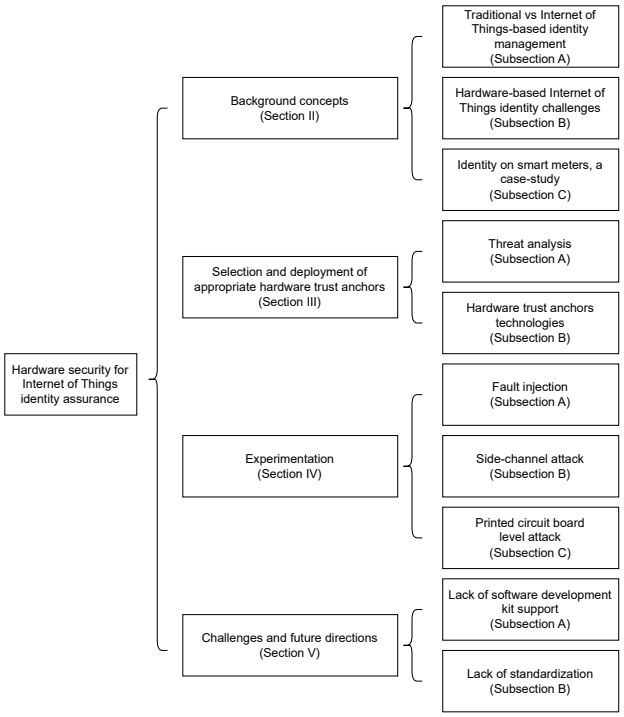


Fig. 2: Article outline

II. Background Concepts

One of the challenges in the area of IoT is how to effectively identify devices and implement secure authentication mechanisms. These two aspects are fundamental in designing a secure system by default, as they serve as essential building blocks for implementing additional security features [13].

The concept of identity pertains to a set of distinct characteristics that enable the recognition and differentiation of an entity or an individual. There are numerous implementations within various application domains [31]. For example, in the context of human identity, it may involve biometric features such as fingerprints or Personal Identification Numbers (PINs) like social security numbers. Similarly, in the IoT context, identity can be established using a unique serial number or cryptographic key.

Authentication refers to the ability of an entity to prove that it is genuinely the entity it claims to be. Thus, authentication verifies the identity of an entity. In the field of cryptography, authentication can be classified into two main categories: data source authentication and entity authentication. Authentication of data origin pertains to situations where it is necessary to ensure that both the information and its source remain unchanged. As a result, data source authentication implies data integrity. In contrast, entity authentication involves confirming an entity's identity and does not include any message other than the assertion of being a specific entity [32].

According to the literature, there is a lack of consensus on whether entity authentication is equivalent to identification. Some authors regard identification as a separate concept, which involves asserting a specific identity without providing conclusive evidence [32], [33]. In this study, we will use the terms "entity authentication" and "entity identification" interchangeably.

A. Traditional vs Internet of Things-based identity management

In the context of identity and authentication, we encounter IdM, which is the process of managing identity information and providing authentication and access control for information systems. IdM systems manage the relationships among various parties, including entities, Service Provider (SP), and Identity Provider (IdP). The entity is the one making an identity claim, the SP offers services to the entity, and the IdP serves three primary functions: entity registration, identity storage, and authentication. This means the IdP is responsible for enrolling new entities and handling the authentication process whenever an entity seeks to access a service [34], [35]. This pivotal role makes the IdP the core component of an IdM system.

In the context of an IdM system, an entity can possess multiple identities distinguished by different identifiers falling into three categories: something known only to the entity and the IdP (e.g., a password), something possessed by the entity (e.g., a serial number), and a physical characteristic of the entity (e.g., fingerprints) that can be used for identification [34].

With the growing usage of IdM systems, they started adopting an isolated model, which represents the traditional identity model where the SP and IdP functions are combined. Consequently, the identification and authentication processes are performed directly within the SP itself. Isolated models pose a management challenge for organizations with multiple services since each entity requires separate identities [35]. In the case of human identity, multiple credentials are needed, reducing user convenience and compromising overall security.

To address these issues, IdM systems began simplifying user experiences and management by introducing the centralized model. In this model, the SP is separated from the IdP. Multiple SPs can use the same IdP for authentication, and entities maintain a single identity across multiple servers, making management easier. However, despite the advancements achieved with the centralized model and its paradigms, two problems persist. IdP servers face scalability issues as the number of identities increases, which results in greater computational and storage requirements. Additionally, the centralized model does not support inter-domain authentication, which poses a usability challenge for large enterprises [35].

The federated model resolves these issues by integrating multiple IdPs within a single authentication domain known as the federated authentication domain. This model is implemented through a set of agreements, standards, and technologies that enable an SP to recognize identities from IdPs in different authentication domains or to create mappings between identities from various IdPs [35]–[37]. Under this model, various protocols have emerged, including Security Assertion Markup Language (SAML) [38], OpenID Connect [39], and comprehensive IdM systems such as Keycloak [40] and Shibboleth [41]. It is worth noting that numerous other frameworks and systems are available, but the ones mentioned here are among the most commonly used and well-known in the field [42].

The SAML [38] is an XML-based protocol for exchanging authentication and authorization data between an SP and an IdP, even when they belong to different authentication domains. This protocol relies on the SAML Assertion message, an XML document containing all the information required by the SP about the entity and cryptographically signed by the IdP. The SP uses the IdP's public key to verify the authenticity of the message. Shibboleth [41] leverages the SAML protocol to implement a complete IdM solution with federated Single Sign-On (SSO) capabilities.

OpenID Connect [39] is an authentication and authorization framework built on top of the OAuth 2.0 authorization framework [43]. It adds an identity layer that facilitates the exchange of identity information [39]. This protocol employs a REST Application Programming Interface (API) to delegate conditional access to entity data. The entity obtains an access token from the IdP, which the SP uses to access its identity information. Keycloak [40] is an example of an IdM system based on OpenID Connect.

With the increasing number of online users and accounts, IdM models have evolved to become more user-centric. Both protocols exemplify this paradigm. In a user centric approach, users have control over the information exchanged between the SP and IdP. This allows users to have different identifiers linked to their identity, which can be shared with SPs based on their consent. Additionally, this paradigm is explored to create SSO experiences, where users only need to authenticate once to access multiple services without having to re-enter their credentials [44]. Currently, research continues to follow the user-centric paradigm, addressing privacy concerns such as a lack of control over the dissemination of personal data [45]. Some initiatives move away from the classic centralized model and focus on decentralized IdMs to preserve user privacy [46]–[48].

The need for specific IoT IdM solutions has also grown with the proliferation of IoT devices [49]. IoT devices differ from humans in that they lack identifiers, making it challenging to develop solutions. The work of Lam et al. [50] outlines four types of characteristics that can be employed to identify an IoT device: *inheritance*, *association*, *knowledge*, and *context*. Among these, *inheritance* relies the most on hardware and is immutable, while *context* depends the least on hardware and is subject to change.

The *inheritance* category is akin to human biometric identifiers, containing information that relies on the device's hardware and is unique to each device. Examples of identifiers in this category include physical unclonable functions (PUFs), which are hardware-based cryptographic primitives leveraging random manufacturing variations to generate a unique and irreversible response for each device. These variations are challenging to reproduce, making PUFs resistant to cloning

or copying, and they serve as a robust foundation for device authentication and identification. Further details on PUFs can be found in Subsection III-B6 of this paper.

The *association* category is founded on the relationships between devices, which are vital to their functionality. In this category, an identifier is derived from the connections between devices. For instance, if a wearable device relies on a connection to a smartphone to communicate with the internet, the smartphone can serve as an identifier for the wearable device.

The *knowledge* category is akin to the "something you know" concept for humans. The device possesses specific information that only it knows, such as a password or a secret key. However, unlike human memory, the level of security assurance differs significantly since the device needs a mechanism to securely store this information, which introduces risks like theft or hacking.

Lastly, the *context* category employs IoT-based sensing data as identifiers. For instance, sensor readings generated by GPS sensors can be considered raw sensor data. However, it is crucial to consider the quality of context, which depends on the quality of the physical sensor, the context data, and the effectiveness of the delivery process. Consequently, these identifiers may have relatively lower quality than others and introduce challenges, such as when a device has an owner and multiple users or when interactions with the devices evolve over time. Both factors contribute to changes in the identifiers, making them challenging to utilize [50].

The absence of a universal identifier for IoT devices presents a significant challenge in the development of seamless IoT solutions. While each resource on the Internet typically has a unique domain name or public IP address managed by international organizations, the lack of a standardized approach to device identification and authentication in the IoT realm hinders the creation of a universal solution [51]. Nevertheless, researchers are actively working on effective methods to authenticate and identify IoT devices.

Most IoT systems employ cryptographic-based entity authentication, which means that device identifiers are used in conjunction with cryptographic algorithms to facilitate identity verification [52], [53]. An example of this type of authentication is *attribute-based authentication* schemes, where device attributes are used to generate a secret key within a public-key encryption scheme. This approach allows devices to be authenticated without the need to share secrets like passwords or keys between the device and the server. Instead, the device's attributes are used to create a unique key that only the server can replicate. Whenever the device requires authentication, it encrypts a challenge sent by the server using its key. The server, in turn, decrypts this message using the expected attributes to reproduce the device key. The device is authenticated if the server can recover its challenge from the encrypted message [50]. Other cryptographic-based entity authentication approaches involve the use of private keys and Public Key Infrastructure (PKI) certificates for each device [54], or modified versions of IdM systems tailored for IoT, particularly when authentication of devices and users is necessary [55].

Blockchain-based solutions are gaining popularity in IoT security due to their decentralized and secure approach to managing and storing device identities. By harnessing blockchain technology [13], [42], IdMs systems can achieve enhanced fault tolerance, as the distributed nature of the blockchain ensures the absence of a single point of failure. Furthermore, blockchain-based solutions have the potential to promote interoperability among devices from various brands by facilitating the use of unique identifiers that are globally recognized and accepted [42].

As previously mentioned, most protocols rely on asymmetric cryptography and assume the availability of secure storage, which imposes limitations on their use in IoT. Asymmetric encryption can be overly resource-intensive for low-end IoT processors, resulting in slow and energy-consuming encryption operations [26], [56]. Additionally, the majority of IoT devices lack access to secure storage due to inherent cost constraints or the expertise required to implement such features [19]. Therefore, to continue using solutions that depend on standard cryptographic algorithms, IoT must be complemented by hardware components that streamline their execution and establish the necessary security conditions.

B. Hardware-based Internet of Things identity challenges

Since the beginning of IoT, device identity has consistently been identified as an ongoing research challenge [12], [57]. Considering the requirements presented in the preceding subsection, we can pinpoint three primary research opportunities for IdM in IoT: **lightweight cryptography** [7], [8], [12], [57], **object identification** [8], [12], [49], [51], [57], and **secure storage** [51], [57], [58].

A significant limiting factor in IoT is its constrained resources, which restrict the implementation of identity and authentication mechanisms. Several authors have suggested that **lightweight cryptography** can provide a solution to this challenge [7], [8], [12]. Lightweight cryptography comprises encryption algorithms or protocols explicitly designed for resource-constrained devices. These solutions are assessed based on criteria such as energy consumption, implementation size, RAM, and computational power [59]. It is important to note that lightweight cryptography does not necessarily entail compromising security for efficiency. Some researchers aim to develop innovative approaches to cryptographic problems while respecting device constraints, while others work on optimizing known algorithms and protocols to align with the requirements of resource-limited devices [59].

Before designing any security system, it is imperative to establish a reliable means of identifying each device. An ideal identification solution should encompass the unique characteristics of the device in its identification process [12]. For instance, in light of the notion that IoT devices can connect to the Internet from virtually anywhere and at any time, their identity should encapsulate these properties [60]. Furthermore, IoT holds the promise that devices will seamlessly communicate regardless of their manufacturer. However, the absence of standardization poses a challenge to this vision. Hence, establishing a vendor-independent identifier becomes

a top priority in addressing this issue. The challenge of **object identification** is being tackled through two distinct approaches. First, researchers and international organizations are endeavoring to create a global naming scheme that multiple manufacturers can employ to identify a device, even when it is not directly connected to the Internet (for example, a sensor connected to a Bluetooth gateway) [57], [60]. On the other hand, researchers are exploring how identity can be defined by assessing the requisite resources and available technologies.

The need for **secure storage** resources has grown in response to the urgency of addressing the object identification challenge. As previously mentioned in this section, many identities necessitate the storage of cryptographic keys, and regardless of the method employed, securely storing the identity on the device is a crucial requirement. Consequently, researchers are actively seeking solutions to this issue, ranging from creating encrypted storage to dynamically generating cryptographic keys using the intrinsic characteristics of the device [33], [58].

The solutions proposed for addressing identity in IoT can be categorized into two main approaches: the utilization of existing IdM systems with computationally intensive cryptographic algorithms and the adoption of new IdM systems based on lightweight cryptography. Hardware can play a substantial role in either of these approaches. The technologies discussed in this paper hold the potential to offer valuable contributions toward overcoming these challenges.

C. Identity on smart meters, a case study

The identity of smart meters is strongly influenced by the evolution and challenges discussed in previous sections. Initially, smart meters began using traditional IdM systems, such as PKIs [61], [62]. In addition, smart meters have also implemented adapted versions of Kerberos and LDAP [62]. Kerberos is a network authentication protocol that utilizes secret keys to authenticate clients and servers, while LDAP is a protocol for managing the authentication and authorization process. By adapting these protocols to the specific requirements of smart meters, it becomes possible to create a more secure and efficient IdM system.

In the realm of smart metering, the implementation of PKI relies on *knowledge* to establish a device's identity. This is achieved by assigning a private key and a signed certificate containing the device's information, such as model and serial number, by the manufacturer's trust authority. The utility company is responsible for defining policies that ensure trust and security among various components of the smart grid [61]. The identity of the smart meter comprises a global identifier from the certificate and a private key.

Unfortunately, this type of solution brings multiple challenges for IoT devices. This identifier is based on *knowledge*, which introduces security challenges. PKIs require that the device securely manage its private keys, which implies the need for secure storage and trusted computing capabilities. These requirements are introduced by Metke et al. [61] as prerequisites for applying PKIs to Smart Grids. Additionally, public-key cryptography expects devices to support these algorithms, which may be challenging for low-end devices [63].

Furthermore, PKIs can have functional limitations as well. Communication channels may have limited bandwidth, as is the case with mesh networks, making these solutions unfeasible. Finally, if these certificates are used to establish peer-to-peer communications, mechanisms for revoking certificates are required. Typically, PKIs use online certificate status protocols to announce revoked certificates, but they necessitate internet access, which may not be available in many smart grid components [63].

Considering the challenges associated with using PKI as an IdM solution for smart meters, researchers have proposed alternative IdM systems to address these limitations. One promising approach explored in recent studies involves non-interactive key distribution [64]. This technique allows the secure distribution of encryption keys to devices without the need for interaction with a central authority. For instance, Seferian et al. [65] introduced a framework that applies this concept to smart meters.

The framework proposed by Seferian et al. [65] aims to address some of the challenges associated with PKIs in smart meters. It focuses on creating a more scalable way to identify devices and manage their keys with reduced bandwidth requirements. The scheme identifies two types of interactions that must be secured in smart meter networks: encryption of peer-to-peer communications between nodes and authentication of nodes by the gateway. The framework utilizes public data, such as the device's MAC address, as the device's public key to reduce communication overhead. Its private key is generated outside of the device by the utility company using the device's public key and a secret managed by the utility company. It is then installed in the device before deployment using a Physically Unclonable Functions (PUF) to ensure secure storage. Each device has a PUF that can encrypt its private key during provisioning and storage. This identity-based key distribution framework aims to provide a more efficient and scalable way to manage smart meter identities and keys while reducing communication overhead.

Nonetheless, this scheme is still rooted in identity-based cryptography, relying on cryptographic algorithms that must be supported by the device. Furthermore, it necessitates that each device possesses a PUF and does not provide a solution for key revocation. When compared to the previous IdM solution, this new approach is better aligned with the requirements of IoT devices. It imposes lower bandwidth demands, permits the utility company to update the device's private key as needed, and ties the device's identity to its hardware. The device's identity in the proposed solution comprises two identifiers: a global identifier represented by the device's MAC address and a private key. Unlike PKIs, the private key in this solution cannot be categorized as a *knowledge*-based identifier since it is safeguarded by the PUF. The PUF generates a unique key based on the device's physical characteristics. Any attempt to tamper with the PUF would render the private key irretrievable, thus adding an additional layer of protection. Consequently, this private key can be classified under the *association* category since it is linked to the device's PUF.

For the remainder of this work, we will use the identity framework proposed by Seferian et al. [65] as a reference for

our case study.

III. Selection and deployment of appropriate hardware trust anchors

Establishing trust is a fundamental requirement for ensuring the integrity and confidentiality of IoT devices. Hardware trust anchors are indispensable components that guarantee the security and reliability of cryptographic operations, providing a strong foundation for the device's identity and authentication. Therefore, the critical task of selecting and deploying suitable hardware trust anchors must not be underestimated.

This section delves into the process of selecting the appropriate hardware trust anchor technologies through a combination of threat analysis on the device's identity and an examination of available technologies. The threat analysis, is designed to inform the reader about potential attacks and countermeasures that must be taken into account while constructing the device's security model. The insights gained from this analysis are then used to scrutinize trust anchor technologies, identifying their characteristics, strengths, and limitations, thus enabling readers to make informed decisions based on their specific requirements.

A. Threat analysis

As previously mentioned , IoT devices are vulnerable to hardware attacks. However, this trend can be mitigated by designing resilient devices to withstand such attacks, which involves implementing measures to protect the components and design of the devices from physical tampering, as suggested by Loukas et al. in their research [15].

A threat analysis is a systematic process used to identify potential security risks and requirements for a given system. This process typically involves four main steps: (1) system characterization, which entails identifying the scope, boundaries, and functions of the system; (2) threat identification, which involves recognizing and categorizing the possible threats that may target the system; (3) threat mitigation, which involves assessing and addressing the identified threats to minimize their potential impact; and (4) validation of the threat model, which verifies the completeness of the threat analysis results. The ultimate goal of a threat analysis is to provide a comprehensive understanding of the potential security risks faced by the system and to develop effective strategies to mitigate them.

Different strategies are available for each of these steps [66]. In this threat analysis, we will employ the Common Criteria (CC) [67] approach, which is a standard for the security evaluation of products. Its model refers to the product being evaluated as the Targets of Evaluation (TOE) and defines two key actors: the TOE owner and the *attacker*[1]. The TOE owner is an individual who aims to mitigate the risks associated with the TOE. This individual is responsible for characterizing the TOE, identifying its *assets* (elements of value that require protection), and its *assumptions* (focus of the analysis). The *attacker* is an actor whose objective is to compromise the identified *assets* and has characteristics that influence the *threats* they impose on the *assets*. These *threats* increase the security risks to the assets and can be minimized with *countermeasures* (see Figure 3) [68]. After developing all of these components, a TOE owner gains a clear understanding of the threats they may encounter and how to reduce the risk associated with the TOE.

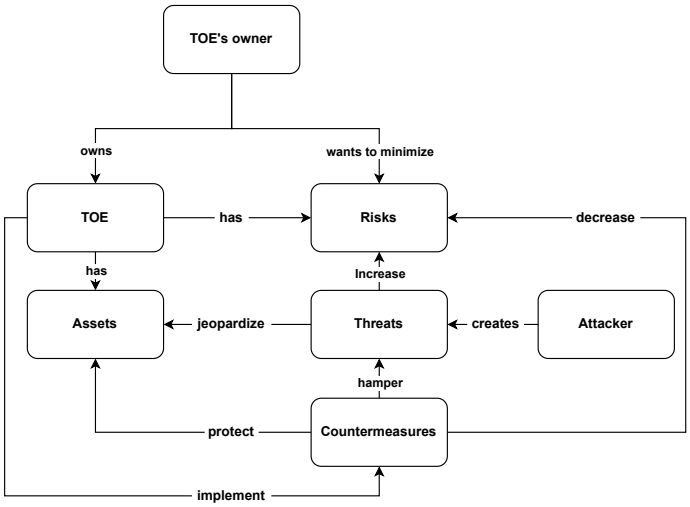


Fig. 3: Relationship between the different actors of a threat analysis

The CC introduced a document called the Security Target [67] to express all this information in a standardized manner. While this document provides a comprehensive and concise depiction of the various components of a threat analysis, its strict structure and extensive requirements may pose challenges for readers unfamiliar with the specifics of such documents. Therefore, to overcome this potential limitation and make the content more accessible to a wider audience, this section will take inspiration from the format of the Security Target as proposed by the CC, while making necessary adaptations to streamline and simplify its content.

The primary aim of this section is to investigate the measures that can be implemented to ensure the secure identity of a device. This investigation includes identifying potential threats and corresponding countermeasures, as well as exploring the security context of a specific case study - smart meters. To achieve this objective, this section is structured as follows: firstly, we will characterize the TOE and outline a set of *assumptions* that will guide our analysis. Subsequently, we will identify the assets of the TOE, enumerate the possible *attackers*, and describe their capabilities and motivations. Finally, we will examine the *threats* these attackers may impose upon the device and propose appropriate *countermeasures*.

1) System characterization and assumptions:

IoT devices can be broadly divided into two major components - hardware and software. While hardware attacks are directed toward the device's hardware, it is essential to

[1]The CC refers to *threat agents* as *attackers*, and we will use the two terms interchangeably.

note that the relationship between the hardware and software components also makes the latter vulnerable to such attacks. As a result, any successful hardware attack has the potential to significantly impact the software and overall functionality of the IoT device. This underscores the importance of ensuring the security of IoT devices' hardware and software components to prevent attacks and ensure optimal performance.

Like any computer, IoT hardware features a CPU that provides computational capabilities, RAM for program storage, Read Only Memory (ROM) housing the boot program connected by a CPU bus, and multiple buses for peripherals, such as persistent storage. However, unlike traditional computers, IoT devices consolidate all these components on a single chip known as a System On a Chip (SoC) [69]. Additionally, an IoT device includes a Printed Circuit Board (PCB), which facilitates the soldering of various components and establishes reliable electrical connections between them.

Smart meters are typically more complex devices with multiple SoCs that share the same PCB [30], [70]. One of the SoCs is responsible for measuring electricity consumption, while the other interacts with the external environment, such as communicating via radio with the utility company [71]. The latter SoC manages interactions between the device and its external environment and is, therefore, responsible for operations related to the device's identity.

In IoT devices, firmware plays a pivotal role in device functionality. Firmware refers to embedded software within a hardware device. Before any software on a device operates, the bootloader is the initial software to run. This bootloader may exist in on-chip ROM, meaning it is hardcoded into the SoC, or it may reside in external memory. Its primary responsibility is initializing various device components and transferring control to the user image. Notably, some SoCs may be equipped with multiple bootloaders to offer added flexibility.

The user image in an IoT device can either be an embedded OS, such as a Real-Time Operating System (RTOS), or a bare-metal application [72]. In the case of an embedded OS, the OS manages various processes and events and includes a hardware abstraction layer (HAL). Moreover, the embedded OS is responsible for executing applications. Conversely, if a bare-metal application is loaded, the application must handle and interact directly with the hardware. The choice between an OS and a bare-metal application depends on the device's capabilities.

The proper functioning of a device relies on the interdependence between its software and hardware components. A defective element within either of these components can render the device inoperable. In the event of a compromised software stack, it does not necessarily imply that the hardware has been compromised. Conversely, hardware attacks can lead to software compromise. Additionally, software vulnerabilities can often be patched through over-the-air (OTA) updates, while hardware vulnerabilities necessitate a new hardware revision for a resolution. This means that certain devices may never receive a fix for hardware vulnerabilities [17]. Hardware attacks require specialized knowledge and tools. Consequently, devices that do not require high assurance do not minimize these risks. Furthermore, security certifications that address these threats do not assess whether the risks are fully mitigated but evaluate whether the device has countermeasures to disrupt and delay the attacker [73].

In the context of device security, any of its components can serve as an entry point for compromising its identity. While software attacks may not directly impact the device's identity, they may target the authentication protocol or necessitate further exploits and lateral movement to attack it. To maintain the focus of this work on identity assurance through hardware, we will only address identity threats related to hardware, including hardware attacks and attacks that hardware can mitigate. These threats will help establish the security features and capabilities of the components discussed in Subsection III-B. Moreover, in this analysis, we assume that the attacker possesses complete physical access to the device and an indefinite amount of time to carry out the attack. Additionally, we will only consider the availability requirement for any *asset* we identify since we assume that, with physical access, the attacker can render a service unavailable by disconnecting the device from power.

2) Assets:

In a threat analysis, an asset is something that holds value for the company and must be protected. For example, in the case of a pay-TV network, a smart card with a decryption key that controls access to the network is considered an asset as it plays a crucial role in the company's revenue [74]. In our threat analysis, there is no specific TOE defined. However, we will focus on the technical assets required to establish identity and authentication and their relevance to our case study.

As we stated at the beginning of Section II, identifiers are essential for supporting the identity of IoT devices. We have listed four identifiers: *context*, *association*, *knowledge*, and *inheritance*. *Context* identifies a device by analyzing its relationships with other devices and its environment. *Association* signifies a symbiotic relationship the device has with a component or another device. In contrast, *knowledge* and *inheritance* rely solely on the device's characteristics to establish identity. The *inheritance* category uses hardware characteristics, while the *knowledge* category encompasses information that only the device knows, such as an authentication token or cryptographic key. Therefore, these last two identifiers are susceptible to compromise through hardware attacks. Additionally, hardware attacks can also endanger identifiers from the *association* category.

In our case study, we have two identifiers: a MAC address, which serves as a global identifier and falls under the *knowledge* category, and the device's private key, which belongs to the association category due to its symbiotic relationship with the device's PUF. The MAC address functions as a public key and, as such, does not require any confidentiality requirement. However, it must remain unchanged to maintain its integrity. On the other hand, the device's private key must ensure both confidentiality and integrity.

Furthermore, since the device's private key is securely stored using a PUF, attackers may attempt to clone the device to steal its identity. Therefore, the **Integrated Circuit (IC) design** and the device's **firmware** must remain confidential.

Additionally, attackers may also attempt to tamper with the device to bypass security features, so the **IC design** must protect its integrity.

Regarding firmware integrity, the primary security objective is to prevent the execution of unauthorized code. This is crucial because even if the device's identity remains secure, an attacker who can execute unauthorized code can perform actions using the device's identity. Ensuring the integrity of firmware, requires consideration of three attack vectors that can compromise security. Firstly, the attacker should not be able to modify the firmware when it is stored in persistent storage. Secondly, if the persistent storage is compromised, the device must not execute any tampered instructions. Finally, the attacker should not be able to induce temporary changes in the execution of instructions.

Moreover, it is important to consider any **secret** encountered within the device as an asset, not only those related to the device's identity. These secrets may include other cryptographic keys, authentication tokens, or passwords that are not directly used as identifiers but play a role in keeping information secure or authenticating data traffic. For example, public keys are typically used to verify the authenticity of traffic and Over-the-air (OTA) updates. If a public key is altered, it can disrupt the signature verification process, leading to various attacks, from Man-In-The-Middle (MITM) to malicious OTA updates. Therefore, for **secrets** like public keys, at the very least, integrity must be addressed. In other cases, both confidentiality and integrity need to be ensured.

To conclude, devices must secure three assets : **IC design**, **firmware**, and **secrets**, to protect the device's identity and authentication capabilities.

3) Attackers:

Many techniques presented throughout this section require expensive equipment, in-depth system knowledge, and execution time. Therefore, not all are accessible to all potential attackers. A threat agent can be either a person or a group that poses a threat [75]. Threat agents can be characterized based on their capabilities and motivation. The capabilities of threat agents encompass the combination of resources and knowledge available to carry out an attack. The threat agent's motivation relates to their desire to perform a particular action and their expectations of success (confidence) [76]. The threat agent's motivation level is tied to the time and effort they are willing to invest. A less motivated attacker in more likely to abandon their target quickly if the device has numerous countermeasures that require time to circumvent. IoT devices face threats from five primary threat agents: criminal enterprises, industrial competition, nation-states, ethical hackers, and lay attackers [17].

Criminal enterprises are primarily motivated by financial gain resulting from an attack. These organizations typically target vulnerabilities that offer tangible benefits for their illegal activities. For instance, they may exploit remote code execution vulnerabilities in IoT devices to enhance the effectiveness of their botnets for conducting distributed denial of service attacks. Due to their characteristics, these entities allocate substantial funds for such attacks. However, their illegal status restricts their ability to acquire human resources and equipment. Given this situation, malicious entities may attempt to launch attacks against smart meters, with the goal of distributing tampered devices that produce inaccurate measurements or even holding a smart meter hostage as a form of ransomware, demanding payment from utility companies to restore energy services to their customers.

Industrial competitors seek a competitive advantage by acquiring information about their rivals' devices. This involves reverse engineering the device to uncover its internal workings. Such attackers have access to a highly skilled workforce and substantial financial resources to obtain all the necessary tools and equipment for executing an attack.

Nation-states have motivations such as espionage and sabotage. This group possesses all the necessary means to carry out complex attacks, including an abundance of professionals, tools, and ample planning time. In the past, nation-states have targeted power grids in the context of cyber warfare. Therefore, smart meters are a prime target for this kind of initiative since they could disrupt the power grid in a distributed manner [77].

Layperson attackers are typically individuals or small groups that engage in hacking to extort money from companies or gain a reputation. These groups generally have limited resources and expertise, so they will seek other targets if they encounter effective countermeasures.

Ethical hackers are driven by curiosity about how a system works or the possibility of monetary rewards. Unlike lay attackers, they do not seek illegal ways to make money but opt for legal initiatives, such as bug bounties. They may possess the expertise to carry out complex attacks but often have a limited budget and time. At the same time, they can access more expensive equipment through universities and hackerspaces. Compared to other attackers, ethical hackers pose a different type of risk. They do not intend to harm a company. However, if not handled properly, they can impact consumer trust in a company. If a vulnerability in a well-known company is disclosed before it is resolved or properly explained by the company, customers may lose trust in the company.

Table I summarizes the motivation and capabilities of each attacker. The attacker's capabilities are analyzed based on two characteristics: knowledge, and resources, which are classified as low, moderate, and extensive. Similarly, motivation is classified as low, high, and extreme based on each attacker's level of persistence. These characteristics directly influence the types of attacks each attacker can execute.

Attackers	Capability		Motivation
	Knowledge	Resources	
Criminal enterprise	Moderate	Moderate	High
Industry competition	Extensive	Extensive	High
Nation-states	Extensive	Extensive	Extreme
Ethical hackers	Moderate	Moderate	High
Layperson attackers	Low	Low	Low

TABLE I: Attacker characterizations

4) Threats and possible countermeasures:

Hardware attacks can be categorized into two main types based on their physical impact on the device: invasive and non-invasive. Non-invasive attacks do not require any prior preparation of the device, which means the attacker can access all necessary components without modifying the device and without leaving any evidence of tampering. In contrast, invasive attacks are more destructive and entail the removal of the chip package to target the internal components. These types of attacks demand expensive tools, complex techniques, and the ability to operate at a miniature scale. Furthermore, many invasive attacks can result in the destruction of the chip itself.

In addition to these categories, some authors have proposed an intermediary class between non-invasive and invasive attacks, referred to as semi-invasive attacks [78], [79]. Semi-invasive attacks are a subset of invasive attacks that involve removing the chip package but do not require contact with its internal lines, reducing their complexity. For the sake of simplicity in this work, we will not distinguish these attacks from invasive attacks.

Another way to categorize these attacks is based on their objectives. This taxonomy classifies attacks into three main categories [79]: reverse engineering, fault injection, and side-channel attacks. Each of these categories encompasses multiple attack techniques (see Figure 4).

In this section, our aim is to identify potential threats and present countermeasures that a device can employ to hinder their exploitation. These countermeasures can be implemented in either software or hardware, but we will primarily focus on hardware-based solutions.

Reverse engineering

Reverse engineering is the process of analyzing a fully functional system to develop specifications describing the system [80]. In the context of IoT, reverse engineering focuses on two main areas: hardware (components and PCB) and firmware. Attackers seek to gain a comprehensive understanding of the device's inner workings, identify vulnerabilities, or create device clones.

To reverse engineer a device's hardware, attackers often employ invasive techniques. These techniques include decapsulation, depackaging, and delayering, which use chemicals to dissolve the chip package. Decapsulation partially dissolves the package while keeping the chip functional, whereas depackaging completely removes the package, rendering the chip non-operational. Delayering involves removing individual layers of the chip for in-depth analysis. These techniques allow attackers to expose the chip, making it possible to analyze it using high-resolution images or Scan Electron Microscope (SEM). [81].

Incorporating active metal shields [82] or a defensive PCB design pattern [83], [84] can deter decapsulation and SEM-based attacks. Active metal shields are conductive layers in the PCB that shield crucial circuit elements, and they can range from simple conductive layers to complex meanders of conductive lines with resistance sensors for detecting tampering attempts. Defensive PCB designs employ various techniques, such as routing critical signals on deeper PCB layers, overlapping them with other electrical paths, or introducing structural obstacles to disrupt the device's operation [83]–[85].

These techniques can also be applied to reverse engineering embedded memory to extract stored information. For instance, information stored on a masked ROM can be decoded, after decapsulation using an optical microscope. Additionally, techniques like microprobing can monitor buses to extract information or bypass encrypted buses by reverse engineering the chip design [86].

Microprobing entails attaching probes inside a chip to measure (side-channel attacks and eavesdropping) or inject voltage (voltage glitching) into an electrical line. However, microprobing requires a decapsulated chip, making it an invasive technique and sometimes necessitating probe pad creation with a Focused Ion Beam (FIB) [17]. FIB involves using a beam of ions to either remove parts of a chip or deposit material. With FIB, attackers can cut or reroute wires at a nanometer scale. While microprobing and FIB techniques are complex, they can bypass many hardware security measures, including active shields, when executed successfully. FIB can be used to tamper with a chip or support other attacks [17], [87]. The success of microprobing depends on the preceding chip decapsulation. A defensive PCB design that obstructs access to critical signals can impede microprobing techniques.

Eavesdropping is the act of passively intercepting information from buses connecting the device's components. Using microprobing for eavesdropping makes this an invasive attack [88]. If buses are exposed, the attack can take a non-invasive form, as an attacker can simply attach a logic analyzer to the bus. Regardless, the attacker must reverse engineer the signals and convert them into meaningful information.

A defensive PCB design can thwart simpler forms of eavesdropping attacks by ensuring that critical signals are not routed on the top or bottom PCB layers. This forces the attacker to use invasive techniques like decapsulation and microprobing to achieve the same goal: eavesdropping on a signal. For critical device signals, designers should evaluate the need for encryption [84].

When attackers aim to reverse engineer firmware, they must first obtain it from the device. Attackers can leverage their access to the device's hardware to perform PCB or logical attacks. The PCB interconnects the various device components. To extract the firmware, an attacker may try to directly connect to persistent external storage using probes or desolder the memory and use a debug tool to read it. If successful, the attacker can write a tampered version of the firmware to the device. Another approach involves analyzing the PCB connections to discover possible features that electrical connections can unlock. SoC devices are often bootable with security features disabled by grounding or pulling up logical pins.

Logical attacks exploit logical interfaces to communicate directly with device firmware, bypassing all hardware security features. Many IoT devices have exposed logical ports like Joint Test Action Group (JTAG), Serial Wire Debug (SWD), or Universal asynchronous Receiver/Transmitter (UART) that allow direct interaction with firmware or attachment of a

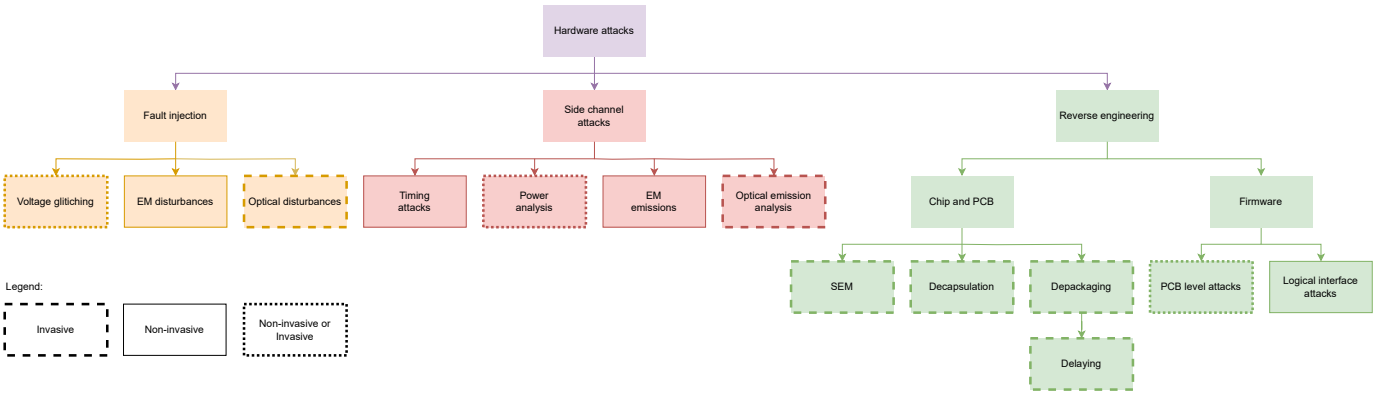


Fig. 4: Taxonomy of hardware attacks

debugger. While these ports can be disabled in software, they are often left enabled. If an attacker can communicate with the JTAG or SWD interface successfully, they can extract the firmware and interact with its execution. In the case of UART, further exploitation may be required to obtain the device's firmware.

Once an attacker has access to the device firmware, the reverse engineering process resembles software reverse engineering. However, researchers must understand the device's architecture and how the firmware interacts with the hardware, as these aspects are often overlooked due to the abstractions provided by OSs.

Fault injection

Fault injection refers to an attack method in which deliberate processing errors are introduced into a processor. These errors are designed to manipulate the normal execution of instructions or alter the data stored in registers within the processor.

There are multiple ways to create these faults. The more common ones are voltage glitching, electromagnetic (EM) interference, and optical disturbances, often referred to as laser glitching. These attacks are generally non-invasive, except for optical disturbances, which qualify as invasive attacks. However, their shared objective remains consistent, even as the attack vectors differ.

Voltage glitching is a technique employed by attackers to rapidly manipulate the voltage of a device's components in order to disrupt its normal operation. By inducing sudden changes in voltage, the attacker aims to affect the device's functioning, potentially exploiting vulnerabilities or gaining unauthorized access. This attack targets a device's power supply or clock signal. Unlike invasive methods, this attack does not require physical modifications to the component's package. Instead, it involves manipulating the voltage or timing characteristics externally to disrupt the device's normal operation and potentially exploit security vulnerabilities. This attack aims to push the hardware to induce an error in the software.

An illustrative example of this type of attack is documented by Colin O'Flyn [89]. In this case, the author used a paper clip to perform a power glitch on a Philips Hue Bridge 2.0's Electrically Erasable Programmable Read-Only Memory (EEPROM) to interrupt the communication between the memory module and the processor. By introducing a deliberate fault in the communication between the memory module and the processor, he could interact with the locked bootloader shell. It is worth noting that while this example showcases a relatively simple attack, many fault injection attacks are more intricate and aim to manipulate the processor to divert its execution flow or change the content of registers in more sophisticated ways.

Usually, voltage glitching is a non-invasive technique. However, it can also be performed at a nanometer scale by attacking voltage lines inside a chip, a process known as microprobing. Unlike the non-invasive approach, microprobing involves the invasive process of decapsulating the chip. This means the chip's protective packaging is removed to gain direct access to its internal components. Consequently, microprobing turns the attack into an invasive one.

EM disturbances occur when an attacker intentionally generates EM signals and directs them toward a target system to induce faults. This is possible because changes in a magnetic field near a chip induce alterations in the voltage, which can temporarily cause flips in the logical levels of a data line.

Optical disturbances leverage the fact that when a transistor is illuminated with a photon-intense light pulse, it conducts current, which can be used to generate localized faults. This attack requires the decapsulation of the component and lasers to emit light pulses.

Fault injection attacks jeopardize the code integrity of the device by executing the code in an unintended way. These attacks are momentary and not persistent by nature but can be leveraged to produce persistent errors, for instance, by attacking the storage interface [90]. Overall, fault injection attacks require knowledgeable and highly motivated attackers since these attacks need to be tuned by experimentation according to the target hardware, which is time-consuming. Moreover, EM and optical disturbances involve high voltages and lasers, which can harm the attacker if the necessary safety measures are not taken.

Generally, devices can prevent this type of attack by applying software or hardware countermeasures. In the realm of software development, there are various recommendations that developers can implement to increase security. For instance,

random delays can be added to the code to deflect exploitation, or critical information may be checked multiple times during execution (for instance, two copies of the same information stored in different memory regions) to detect any exploitation attempt [91]. The duplication principle may also be applied to hardware. For example, the device can implement the same function in multiple places and compare their output to detect tampering attempts. Nevertheless, this type of approach is expensive [85].

In addition to these general countermeasures, specific techniques exist for each threat. Devices may have voltage sensors or monitor the clock signal to detect voltage glitching attacks [92], [93]. Power or clock lines are distributed across the PCB, meaning that any attack targeting these lines will propagate throughout the entire network of lines, making it relatively easy to detect such attacks. In contrast, EM attacks tend to be localized, making them more challenging to detect [94]. EM sensors may be used to detect EM fault injections [93], [95]. However, their placement must be properly analyzed due to their limited range. On the other hand, EM fault injections can be avoided by using an active metal shield that protects the SoC from EM waves and optical disturbances [85], [96].

Side-channel attacks

Side-channel attacks constitute a class of non-invasive attacks aimed at extracting confidential information from a system by analyzing various physical parameters, such as time, power consumption, or EM emissions [97]. Most of these attacks necessitate knowledge of both the plaintext and corresponding ciphertext.

Timing attacks capitalize on data-dependent execution time differences to uncover secret data [97]. Let us consider a scenario where a password verification system compares an attacker's input with the correct password character by character. During this process, the system provides timely feedback on the correctness of each character. In this situation, an attacker can measure the time it takes from entering a password to receiving feedback. By carefully measuring these time intervals, the attacker can exploit variations in execution times, which depend on the number of correct characters in their input. This knowledge enables the attacker to optimize a brute-force attack strategy. Instead of attempting to guess the entire password in one go, the attacker can iteratively and efficiently brute-force each password character. This is achieved by leveraging the different execution times associated with the varying number of correct characters, significantly reducing the effort required to crack the password. Depending on the target, an attacker may measure the time taken by a software system to respond to specific inputs or operations [98] or count CPU cycles [99]. To mitigate this threat, developers must address it at the software level. One approach is to ensure consistent response times regardless of the correctness of the input. By implementing measures that maintain uniform timing and response patterns, developers can minimize information leakage through side channels and enhance the system's overall security.

The power consumption of a processor depends on its current activity, particularly when there are changes in the state of its components. Precise measurement of power consumption allows an attacker to identify the current instruction and estimate changes in memory bits [78]. Many power analysis techniques can be employed to attack cryptographic systems, with the two primary techniques being Simple Power Analysis (SPA) and Differential Power Analysis (DPA) [78].

SPA involves direct observation of power consumption and demands the attacker's specific knowledge of the cryptographic algorithm implementation to succeed. DPA is a technique that does not require prior knowledge, relying on statistical analysis to extract information from a data set of power traces [100]. Despite not necessitating expensive equipment, power analysis does require a skilled attacker.

An example of power analysis utilization is featured in the work of Ronen et al. [101]. In this study, a team of researchers used power analysis to extract the cryptographic keys responsible for encrypting and verifying firmware updates in a smart bulb. This analysis subsequently allowed the authors to upload a malicious OTA update, compromising the integrity and security of the device. To prevent these attacks, boards may employ voltage regulators to maintain steady power consumption independently of running operations. However, attackers may bypass this by probing inside the chip (microprobing) [93]. This process, though, would require decapsulation.

Integrated Circuits emit electromagnetic waves during their operation. The principle behind EM analysis is quite similar to power analysis. By using EM probes, it is possible to identify events by analyzing the EM signals around a device. Moreover, in EM analysis, you can also pinpoint the location of a specific activity by locating the source of that radiation, a capability not available in power analysis. Several techniques exist to analyze these measures, with the main ones being simple EM analysis and differential EM analysis, similar to their power analysis counterparts [97].

Optical emission analysis focuses on studying emitted photons by transistors that change state. Once again, there are two main techniques: simple and differential analysis [102], which can be used to extract cryptographic keys. Additionally, optical emission analysis allows attackers to locate the source of photon emissions, supporting reverse engineering efforts. These attacks require direct observation of the chip's components , which necessitates chip decapsulation. Furthermore, these attacks require custom-built tools, increasing the knowledge required to execute them [17], [103].

One possible mitigation for EM and optical emission analysis is to use an active metal shield to protect critical components. These shields hinder EM and optical injections and prevent internal emissions from propagating to the outside, thus preventing leakages [85], [104].

Software developers can implement measures, such as introducing random delays during critical operations, to mitigate the risk of successful side-channel attacks [91]. These delays disrupt timing patterns, making it harder for attackers to gather information. By incorporating random delays, the timing side-channel becomes less reliable as an avenue for exploitation.

Side-channel attacks primarily affect the confidentiality of identity data. The actors who perform these attacks

Threats	Identity assets						Required capabilities			Hardware countermeasure
	Secrets		Firmware		Integrated circuit design		Knowledge	Resources	Motivation	
	Confidentiality	Integrity	Confidentiality	Integrity	Confidentiality	Integrity				
Voltage glitching	No	No	No	Yes	No	No	Moderate	Low	High	- Voltage sensors - Clock signal sensors
Electromagnetic disturbances	No	No	No	Yes	No	No	Moderate	Moderate	High	- Multiple voltage sensors - Active metal shield - Electromagnetic sensor
Optical disturbances	No	No	No	Yes	No	No	Extensive	Extensive	High	- Active metal shield
Timing attacks	Yes	No	Yes	No	No	No	Moderate	Low	High	
Power analysis	Yes	No	No	No	No	No	Moderate	Low	High	- Voltage monitoring
Electromagnetic emissions analysis	Yes	No	No	No	No	No	Moderate	Low	High	- Active metal shield
Optical emission analysis	Yes	No	No	No	No	No	Extensive	Extensive	High	- Active metal shield
Scanning electron microscope	No	No	No	No	Yes	No	Extensive	Extensive	High	- Active metal shield - Defensive printed circuit board design
Decapsulation	No	No	No	No	Yes	Yes	Extensive	Extensive	High	- Active metal shield - Defensive printed circuit board design
Depackaging	No	No	No	No	Yes	Yes	Extensive	Extensive	High	- Defensive printed circuit board design
Delaying	No	No	No	No	Yes	Yes	Extensive	Extensive	High	- Defensive printed circuit board design
Printed circuit board level attacks	Yes	Yes	Yes	Yes	No	No	Low	Low	Low	
Logical interface attacks	Yes	Yes	Yes	Yes	No	No	Low	Low	Low	
Eavesdropping	Yes	no	Yes	no	No	No	Moderate	Low	Low	- Defensive printed circuit board design
Microprobing	NA	NA	NA	NA	NA	NA	Extensive	Extensive	High	- Defensive printed circuit board design
Focused ion beam	NA	NA	NA	NA	NA	NA	Extensive	Extensive	High	

TABLE II: Relation between threats, assets, and required capabilities

require extensive knowledge of statistical analysis and the target's cryptographic implementation. In terms of resource requirements, time, power, and EM emission analysis share a common need for equipment such as oscilloscopes and appropriate probes. However, optical emission analysis deviates from this requirement, as it necessitates specialized equipment and the process of chip decapsulation.

5) Summary:

In this section, we have examined how hardware-based attacks can pose a threat to the identity and authentication capabilities of IoT devices. We have discussed the various security objectives needed to maintain the security of IoT identity and authentication. Additionally, we have provided a comprehensive list of potential attacks that could be used to compromise these devices, accompanied by a detailed analysis of the knowledge and resources required for each attack. These insights play a crucial role in understanding the tactics that a specific threat actor may employ to compromise IoT device security.

Table II summarizes the various attacks by mapping the requirements needed for the attacker to succeed. Additionally, it identifies the compromised assets and outlines hardware countermeasures. In this table, we included microprobing and FIB as they can be applied to multiple attacks and significantly affect the requirements.

In the following subsection, we will use the identified countermeasures and attacks to assess the security level of each technology against hardware attacks.

B. Hardware trust anchors technologies

We have identified six key technologies that serve as fundamental building blocks for addressing the current open research challenges in IoT: True Random Number Generator (TRNG)s, ROMs, crypto accelerators, Secure Element (SE)s, Trusted Execution Environment (TEE)s, and PUFs. In the following sections, we will examine the advantages and disadvantages of each of these technologies in supporting device identity. We will also analyze prevalent security attacks and the commonly implemented countermeasures. Furthermore, we will explore the systems in which these technologies have already been employed to facilitate device identity.

Before analyzing each one, it is essential to consider their nature and relationships. These technologies can be categorized into two distinct groups: basic and composite building blocks. Basic blocks provide elementary resources, while composite blocks offer multiple resources, and can be broken down into smaller building blocks. For instance, a TRNG is a basic block that solely provides random numbers. On the other hand, a SE is a composite building block because it offers various features such as cryptographic operations and random number generation, which are provided by basic blocks, like a cryptographic coprocessor and TRNG.

1) True random number generator:

Encryption serves as the foundation of identity and authentication solutions. These systems heavily rely on the generation of unpredictable and non-reproducible key streams for cryptographic key creation and the generation of authentication challenges, among other essential functions. There are two primary types of random number generators: True Random Number Generator (TRNG) and Pseudorandom Number Gnerator (PRNG). However, PRNGs are not recommended for cryptographic operations due to their lack of entropy [105].

A TRNG is a random number generator capable of producing truly random numbers, free from any predictable patterns or periodicity, derived from physical sources. What sets TRNGs apart from the PRNGs commonly used in most systems is the quality of the generated numbers. A PRNG employs algorithms to generate a sequence of numbers that depend on an initial seed provided to the algorithm. The numerical sequence generated by a PRNG is deterministic, meaning that if the seed is known, an attacker can calculate the entire PRNG sequence.

On IoT devices, the generation of seeds for cryptographic purposes often faces limitations in terms of available entropy sources. Additionally, attackers can gain physical access to the device, potentially disrupting these sources. As a solution to these challenges, TRNGs can be employed to extract entropy from the device's environment, such as electronic noise [26]. An example of this electronic noise is the variations

in signals produced by electronic oscillators, which can be sampled, filtered for potential interference, and quantified as digital bits [106]. However, constructing such TRNGs may require the use of multiple oscillators to produce high-quality random numbers [105], which can increase production costs and energy consumption [26].

Electronic noise, however, can be susceptible to external interference, which may affect the quality of the generated random numbers. To address this issue, researchers have turned to quantum theory to develop new TRNG constructions. In quantum mechanics, each choice is inherently random and independent of others. Based on this principle, researchers have utilized methods such as analyzing the choices of individual photons or measuring the time intervals between the radioactive decay of elements to create TRNGs [107].

Security attacks and countermeasures

TRNGs, typically embedded within a device, often do not incorporate their own security countermeasures but delegate this responsibility to the device itself. Depending on the type of TRNG, some attacks may exploit environmental biases inherent in these components, potentially leading to the generation of weak random numbers. For example, TRNGs based on Ring Oscilator (RO)s can be biased with EM fault injections [107], and attackers may perform EM side-channel analysis to retrieve information about the TRNG's internal state [108].

Advantages and disadvantages for identity assurance

IoT devices commonly face challenges related to a lack of available entropy. In this context, TRNGs can provide a solution by offering high-entropy numbers within an IoT platform. However, TRNGs also come with notable drawbacks. These drawbacks include increased device costs and higher power consumption. Additionally, conventional TRNGs that do not rely on quantum physics may be susceptible to environmental biases. This vulnerability implies that adversaries with physical access to the device can exploit such biases and launch attacks against the generated random numbers.

Implementations

TRNGs are often included as a foundational component within more complex systems. For instance, any identity system that utilizes a Trusted Platform Module (TPM) or SE inherently relies on a TRNG since TRNGs are a fundamental part of these components. An explicit example of TRNG use in identity systems is demonstrated in the work of Yang Su et al. [109], where a decentralized machine identifier for electric vehicles was developed using a TRNG module to generate vehicle identification.

2) Masked read-only memory and one-time-programmable memories:

Non-Volatile Memory (NVM) is a type of memory used in devices to store information persistently. Different families of NVM distinguish themselves by the technology they employ and the number of times they can be rewritten. Among the various types, those allowing only a single write operation can be used to support security operations in a system. For example, they can be employed to implement a RoT or store public keys [110].

There are four types of one-time writable memories: masked ROMs, floating-gate One-Time-Programmable (OTP)s, fuse OTPs, and anti-fuse OTPs. The last three are collectively referred to as OTP memories because they can be programmed in the field, offering more flexibility compared to a masked ROM, which can only be set during the fabrication process [111].

Masked ROMs have the information hardwired into the chip design, requiring knowledge of the data to be stored before the component's production. This type of memory offers a significant advantage in terms of low production costs when a large quantity of memories storing the same information is needed. However, it can be a disadvantage for small-scale deployments [110], [112].

Floating-gate memory is a type of memory that utilizes floating-gate transistors to store information, enabling it to be reprogrammed and erased in the field [113]. This is possible because the packages of these memories have a quartz window that can be uncovered to irradiate the floating-gates with ultraviolet (UV) radiation, effectively erasing the memory [110]. Floating-gate OTPs follow the same working principle as floating-gate memories but have the floating-gates shielded to ensure they are not reset by radiation.

Electric-Fuse (eFuse) OTP memory is constructed with a set of fuses that are blown to represent data. This operation is performed by applying a high voltage to the fuse, which can be done in the field [114]. The working principle behind an eFuse is electromigration, a process in which material is gradually transported in a conductor. eFuses consist of conductive metal lines that increase their resistance when exposed to high voltages, causing the circuit to open due to electromigration [115]. However, in terms of data retention, electromigration introduces disadvantages, as eFuses are susceptible to re-growth issues, where metal lines unintentionally connect, altering the stored data [116].

Anti-fuse OTP memory is a variant of fuse OTP that incorporates an anti-fuse element in its construction. An anti-fuse is an electronic component designed to exhibit non-conductive behavior under normal conditions. However, when subjected to high voltage, it undergoes a permanent transformation, becoming conductive. This transition is used as a means to store information in memory permanently [114].

Finally, table III summarizes the differences between read-only and one-time-programmable memories.

Security attacks and countermeasures

The primary objective of read-only memories is to make data unchangeable. With the exception of floating-gate OTPs, there are no known attacks against this type of memory. However, as mentioned earlier, since floating-gate OTPs are based on floating-gate memories, attackers may attempt to bypass their shield and expose them to UV radiation to overwrite the data [117]. Even though eFuses suffer from re-growth issues, no literature has been found exploiting this fact to create an attack to compromise the OTP.

Technology	Pros and cons
Masked ROM	+ Hardwired data − Programmed during manufacturing
Floating-gate OTP	+ Programmable after manufacturing − Vulnerable to ultraviolet attacks
eFuse OTP	+ Programmable after manufacturing − Limited data retention
Anti-fuse	+ Programmable after manufacturing

TABLE III: Comparison between different memory technologies

However, given that these memories often contain information that could be of interest to an attacker, it is essential to assess how easily the stored information can be retrieved. The discussed memories share very similar protection levels. To access the information, an attacker would need to depackage and delayer the chip, then employ a high-resolution optical microscope to read its content [86], [117]–[119]. Another option is to use an SEM to retrieve the information, though this is not possible for anti-fuse OTPs [115]. All these techniques are invasive, requiring expensive equipment and manual work to extract the memory data. To mitigate these risks, device designers often employ active shields to protect the memories, erasing their content or damaging the device if any tampering attempt is detected. However, if not backed up by batteries, this protection will only function when the device is powered on and will not prevent offline attacks.

Advantages and disadvantages for identity assurance

Masked ROM and OTP memories offer cost-effective options to store information that can only be read. Nevertheless, they also come with several security risks that must be addressed for high-assurance deployments. An attacker with physical access to the device can replace these memories with similar ones containing different content, as devices typically lack mechanisms to ensure the integrity of the memory itself. Furthermore, specific attacks can be leveraged for each memory type to alter their content [118], [120]. As a result, masked ROM and OTP memories bring advantages for systems requiring low-security assurance. However, these security risks must be mitigated if these components are used for higher assurance levels.

Implementations

Masked ROM and OTP memories are foundational components in more complex systems. One of the most common uses of these memories is to provide a RoT, which is why they are included in most SEs [121]. More recently, researchers have employed these components as RoTs in the ARM Trust Zone system, as they do not provide a default secure way to store information [122]. Another application for these memories is enabling and disabling features in a device. For example, many SoCs use fuses to disable debug capabilities, such as a JTAG or UART port [19].

3) Crypto accelerators:

Crypto accelerators exhibit a remarkable capacity for performing cryptographic operations at a high throughput rate. It is worth noting that these components may or may not include countermeasures to mitigate known attacks. In the literature, they go by various names, such as custom processors and crypto arrays, as discussed in the work by Bossuet et al. [123]. Following the taxonomy presented by Bossuet et al., we categorize these components into four types: General Purpose Processor (GPP)s with crypto acceleration, hardware crypto-coprocessors, crypto-processors, and crypto arrays.

GPPs with crypto acceleration are CPUs that come equipped with dedicated instructions for cryptographic operations. These instructions allow programs to utilize specialized hardware for cryptographic operations, which is significantly faster than running these operations on general-purpose hardware. However, this type of solution does not offer additional security features and relies on the GPP for secure storage and protection against hardware attacks. Crypto acceleration in these GPPs is typically achieved through specialized Arithmetic Logic Units integrated into the GPP, providing a low overhead connection to the rest of the CPU. Examples of GPPs with crypto acceleration include Intel CPUs with the AES-NI instruction set [124] and ARM CPUs with the ARMv8 Cryptography Extension [125].

Hardware crypto-coprocessors are dedicated logic devices or hardware modules designed solely for executing cryptographic operations. These coprocessors cannot be programmed and depend entirely on a processor to function, even to the extent of lacking storage for secrets. However, they offer greater flexibility than GPPs with crypto acceleration, as they allow for reconfiguration of cryptographic algorithms, often implemented on Field-Programmable Gate Array (FPGA)s [123].

A crypto-processor is an independent processor specialized in cryptographic operations. Unlike GPPs, these processors protect their secret keys by generating them internally, storing them in a dedicated memory, and transporting them via a dedicated bus. These measures are implemented to ensure that the system can only interact with the keys by performing cryptographic operations, offering a higher level of security [123], [126].Nevertheless, countermeasures against more intrusive attacks may vary depending on the specific model. An example of a crypto-processor commonly found in computers is the TPM.

Hardware TPMs are secure crypto-processors that adhere to specification created by the Trusted Computing Group (TCG) to establish trust in a computing system. Hardware TPMs must incorporate the necessary hardware protections to provide three RoTs: storage, measurement, and reporting [127], [128]. TPMs include registers for storing measurements of each software component that runs during the system's boot process, providing a chain of trust that enables the detection of any tampering attempt during boot. Additionally, they feature a set of asymmetric key pairs used for encryption and signing [127]. These features enable the production of signed reports on the system's software configuration and, the decryption of data only when the system matches a specific state, forming the

basis of trusted computing. TPMs also offer a secure random number generator and cryptographic engines, with the most recent version, 2.0, supporting cryptography engines for RSA with keys of at least 2048 bits, ECC with a 256-bit key using the US NIST Curve P-256, and AES with keys of 128-bit keys in CFB mode. All these features are backed by tamper-resistant hardware [129], [130].

Finally, a crypto array is a crypto accelerator that comprises multiple cryptographic processing elements working in conjunction with a GPP to provide fast parallel computation of cryptographic algorithms. These components are primarily used in virtual private networks, where the simultaneous handling of multiple encrypted connections is essential [123]. Consequently, they are not typically directed toward or used in IoT applications, which do not require the simultaneous handling of multiple connections.

Table IV presents the pros and cons of each one of these technologies.

Technology	Pros and cons
Cryptographic instruction sets	+ No additional hardware + Easy integration
Crypto-coprocessors	− Complex integration − Dedicated hardware
Crypto-processor	+ Easy integration − Dedicated hardware
TPM	+ Easy integration − Dedicated hardware
Crypto array	+ Complex integration − Dedicated hardware − Expensive hardware + High throughput

TABLE IV: Comparison between different crypto accelerators

Security attacks and countermeasures

In general, crypto accelerators do not inherently provide security countermeasures against hardware attacks since they are typically integrated into more complex systems. Consequently, crypto accelerators have been targeted by various attacks. For instance, it is known that Intel AES-NI is vulnerable to voltage glitching [131] and side-channel attacks [132]. An example of an attack against crypto-coprocessors is the power fault injection attack against the coprocessor in the PlayStation Vita [133]. However, it is important to note that some of these components may offer security features. For instance, TPMs typically share processors with SEs, inheriting features like active metal shields and voltage monitors.

Nevertheless, it is crucial to recognize that the level of protection provided by TPMs can vary. According to the TCG definition, TPMs need to be certified with Common Criteria Evaluation Level 4, indicating that the component undergoes rigorous design, testing, and review against the TCG security profile [134]. The TCG security profile defines TPMs as being resilient against physical hacking attempts. Additionally, TPMs are certified with the FIPS 140-3 certification [135], which encompasses multiple assurance levels. Level 1 of this certification does not mandate any specific physical security mechanism, while level 2 requires the component to show evidence of tampering attempts on plaintext cryptographic keys and critical security parameters. Mechanisms for detecting and responding to hardware attacks are only enforced at level 3 of the certification [135]. Typically, TPMs are certified at level 1 or 2, which means TPM designers do not have to incorporate protections against invasive attacks but must make the component tamper-evident.

Advantages and disadvantages for identity assurance

One limitation when implementing security features in IoT devices is the constraint of a limited development budget and minimal hardware capabilities. Crypto accelerators can be a solution for IoT devices that lack the computational power to run cryptographic algorithms. GPPs with crypto acceleration are the most cost-effective and straightforward way to incorporate crypto acceleration into an IoT device since they are embedded within the device's main CPU. To leverage these capabilities, developers only need to ensure that their operating system and cryptographic libraries are optimized to utilize these specialized instruction sets [19]. This makes it relatively easy to integrate at the software level with the rest of the system.

Another approach, albeit requiring more adaptation, is to use crypto-processors. These components are standalone and must be added to the device, typically connecting to a general-purpose bus. At the software level, they usually provide a software stack that enables developers to leverage their capabilities.

Conversely, cryptographic coprocessors present certain disadvantages. They necessitate low-level integration with access to the internal buses of a GPP. However, they offer the advantage of reconfigurability when required.

Implementations

GPPs with crypto acceleration are commonly found in commercial processors. As previously mentioned, examples of these include AES-NI and ARMv8 Cryptography Extension. Consequently, many IoT identity systems, whether intentionally or not, already take advantage of these extensions to accelerate their cryptographic operations.

Crypto-coprocessors and crypto-processors are instrumental in expediting cryptographic operations, particularly when employed alongside resource-constrained controllers. An illustration of their application can be observed in the work of Pearson et al. [136], where a microchip cryptographic coprocessor was used to accelerate authentication and encryption operations in cloud-based environments. Beyond their acceleration capabilities, such coprocessors offer the added advantage of providing storage for cryptographic keys, which was utilized to securely store authentication keys. Additionally, the TPM is a noteworthy technology in this context. Given its widespread adoption, various solutions incorporate TPMs to attest the boot process and securely store device cryptographic keys [137]–[139].

4) Secure elements:

A Secure Element (SE) represents a tamper-resistant component offering a range of security features such as secret management and secure application execution [140]. They are known for their resistance to forgery or copying and are equipped with a unique identifier [141]. The concept of SEs was introduced by the GlobalPlatform, an initiative involving various industry stakeholders aimed at creating specifications and standardization for secure components [140]. These components are also commonly referred to as smart cards, and both terms will be used interchangeably in this work.

A SE is essentially a SoC with its own independent CPU, RAM, EEPROM, and ROM, all packaged in a compact form. Inexpensive smart cards typically have storage capacities ranging from 12 to 144 kilobytes of EEPROM, 6 kilobytes of RAM, and 200 kilobytes of ROM [142]. Moreover, modern smart cards are equipped with various interfaces for interacting with the external world. Initially, relying on serial communication, contemporary smart cards often incorporate Near Field Communication (NFC) or even Bluetooth interfaces due to the prevalence of smartphones [143].

What sets SEs apart from common embedded systems is their multiple layers of defense. For example, any attempt to decap the SE's package would encounter an active current-carrying layer that, if compromised, would result in the destruction of card's information [121]. To thwart probing attacks, SEs employ encrypted buses between their components, and the paths on the PCB are typically scrambled to deter reverse engineering attempts [144].

SEs are susceptible to side channel and fault injection attacks, with the preferred attack vector being power analysis [145]. To counter these attacks, SEs typically employ software-based measures, such as designing cryptographic algorithms with constant execution times or introducing random delays in execution (e.g., processing a stream of bytes XORed with a key in a random order) [121]. Furthermore, they can detect or impede fault injection attacks by using sensors to identify fault conditions, like abnormal events in the voltage or clock supplied to the card. Additional software measures, such as checksums to prevent memory alterations and variable redundancy involving multiple copies of the same information in different locations, can be implemented to detect tampering attempts. The specific measures employed by each SE manufacturer depend on the desired assurance level.

In the past, since SEs were primarily used for a single application, the application and OS were developed together and stored in ROM. However, this approach posed challenges in development, requiring specific knowledge about the smart card's intrinsic features. Additionally, the final product was model-dependent and unable to receive updates as it was stored in ROM. Nowadays, smart cards deploy the OS and applications independently to overcome these limitations. Smart card OSs are minimal, provide hardware abstraction to applications, and are typically deployed in ROM, making them unchangeable after production. On the other hand, applications use the APIs provided by the OS and are stored in Electrically-Erasable Programmable Read-Only Memory (EERPOM), enabling updates over time. Furthermore, many smart card OSs support virtualization, allowing the deployment of multiple applications within the same smart card independently [146].

GlobalPlatform standards have significantly contributed to the development of multi-application smart cards, promoting security and interoperability, regardless of the OS used. Among their comprehensive set of standards, the GlobalPlatform Card Specification stands out. This specification defines a detailed framework comprising logical components designed to facilitate secure multi-application smart card operation. It includes various procedures and APIs specifically designed for efficient application management and installation within the dynamic context of multi-application environments [146].

Smart card models offer a range of security features supported by specialized hardware coprocessors. Typically, this list includes asymmetric and symmetric cryptographic algorithms, hash functions, and a TRNG. The availability of algorithms may vary depending on the specific smart card model [121].

Security attacks and countermeasures

Despite their limited computing power and memory capacity, SEs operate under high-security requirements. The SE's threat model assumes that the information stored inside the card must remain secure even when attackers have unlimited access. As a result, smart cards employ the security countermeasures outlined in Subsection III-A4. SEs use multiple anomaly sensors to detect unintended execution conditions, including variations in temperature, voltage, clock, and EM fluctuations. When an unusual condition is detected, the SE responds accordingly. Depending on the specific devices and sensors involved, the response may involve an automatic reset or halting execution until normal operating conditions are restored [121].

Over the years, numerous side-channel attacks have been launched against smart cards. To combat these threats, SE software is designed with multiple layers of defense to prevent information leakage. This includes measures such as ensuring constant execution times for cryptographic operations, masking memory when critical information is present, and introducing randomized data manipulation by the user. In addition to software countermeasures, SEs employ active metal shields to thwart certain side-channel attacks, as described in Subsection III-A4. These measures in combination with randomized elements in the SE's internal design further discourage reverse engineering and make invasive attacks or SEM more challenging [121].

The security features of SEs are guaranteed by two certifications: Common Criteria and FIPS 140-3 [135]. Typically, SEs are certified at Assurance Level 6 in the Common Criteria, signifying a partially formally verified and tested design with a security profile specifically created for this purpose [147]. They are also certified at Level 3 of FIPS 140-3, ensuring tamper protection and response to attacks [135].

Advantages and disadvantages for identity assurance

Incorporating SEs in IoT devices offers a practical solution for addressing the hardware limitations of highly constrained devices when it comes to performing security operations. SEs

provide a significant advantage by delivering robust security assurance while consuming minimal energy. Additionally, there is a wide variety of smart cards to choose from, each with its own set of characteristics. In more complex setups, devices can make use of multiple application cards within the SE, allowing them to support a diverse range of applications. In simpler systems, you can employ smart cards with limited functionality to efficiently meet specific requirements.

However, depending on the system's design, the device may require the storage of a PIN to unlock the smart card's functionalities. Therefore, in a production system, using an SE will necessitate additional security mechanisms to address this issue. Moreover, despite different form factors, adding an SE means introducing another component, which increases the device's size and complexity. Finally, if you need to develop an SE application, the development team will need to familiarize themselves with a new technology and development kit.

Implementations

SEs have found utility among numerous researchers and solutions for securely storing cryptographic keys that identify devices and offloading cryptographic operations [148]–[150]. For instance, in the academic realm, Jeon et al. [149] introduced the use of SEs in LoRaWAN nodes to mitigate the risk of communication key leakage within the LoRaWAN protocol. This approach significantly enhances the security of the communication process.

In the industrial sector, an exemplary application can be seen in Bosch security cameras, where SEs are deployed to securely store cryptographic keys and facilitate the secure handling of firmware updates [150]. Additionally, some researchers have harnessed SEs with NFC interfaces to create solutions that support both remote and local identification [151], [152]. This enables an operator to physically identify a device using a smartphone equipped with NFC, mitigating issues related to labels that could be tampered with or eavesdropped on by potential attackers [152].

Furthermore, the application of SEs in military Unmanned Aerial Vehicle (UAV)s has also been subject to study. Any information stored in a military UAV must remain secure, even if the UAV falls into enemy hands. With the ongoing development of autonomous UAV fleets, researchers have suggested incorporating an SE into each drone within the fleet to store any information that could compromise its mission or the overall fleet's security [153].

5) Trusted execution environment:

A Trusted Execution Environment (TEE) comprises a combination of software and hardware designed to provide isolated execution and storage environments separate from the main operating system. Its primary objective is to ensure information security and privacy even in cases where the device is compromised [154]. This is accomplished through five key features: isolated execution, secure storage, remote attestation, secure provisioning, and trusted paths [155].

Isolated execution allows applications to run independently of other code, their own address space and system resources. This isolation can be implemented in various ways, either at the OS level using a hypervisor or within a parallel environment with separated components.

Secure storage ensures data confidentiality and integrity, even when the device is powered off. While the OS can maintain isolation based on the required assurance level, stronger secure storage solutions employ separate components to manage access control independently of the OS. Alternatively, the RoT can be utilized to store cryptographic keys, which are then used to securely encrypt and decrypt data. However, in this scenario, it is critical to prevent data from being rolled back to a previous version.

Remote attestation serves two essential purposes: verifying the origin of a message remotely and ensuring the correct loading of the TEE. It guarantees the correct loading of the TEE's firmware, protecting against persistent threats. However, it is important to note that remote attestation does not defend against runtime compromises or provide information about the device's ongoing proper functioning.

Secure provisioning involves sending data to a specific TEE, while maintaining secrecy and integrity in communication. This mechanism is typically used for secure updates or changing device settings, leveraging remote attestation and unique cryptographic keys for each device.

A trusted path enables secure access to physical peripherals. For example, if an application running within a TEE requires access to a keyboard for user interaction, it should be impossible to interfere with the connection between the TEE and the keyboard in any way, including eavesdropping attempts.

At the core of these features, lies the concept of the Trusted Computing Base (TCB), which encompasses the set of software and hardware components explicitly trusted to ensure the security properties expected from a TEE [155]. The TCB essentially serves as the RoT for these platforms. It defines two distinct environments: the TEE, which is the execution environment provided by the TCB, and the Rich Execution Environment (REE), provided by untrusted components [141].

Much like SEs, the GlobalPlatform initiative plays a critical role in establishing TEE standards. It proposes TEE architectures, defines APIs for communication between REE applications and those running within the TEE, and promotes the development of TEE applications that can operate independently of the underlying TEE implementation [141]. Additionally, it introduces the concept of a Trusted User Interface API. As mentioned earlier, TEE applications often require user input, which is why trusted paths are integrated into the TEE architecture. The GlobalPlatform aims to address this challenge by advocating for an input/output peripheral within the TCB [156].

The GlobalPlatform TEE system architecture specification [154] puts forth three different architectures: one with shared memory between the REE and an isolated component inside the SoC where the TEE operates; an architecture in which all resources are shared with the REE but with an isolation level between the two environments; and an architecture introducing an external security SoC in the device that communicates with the main SoC to provide

TEE capabilities.

Hardware-based trusted execution environment enabling technologies

The concept of TEE encompasses various implementations with unique characteristics, limitations and development approaches. By delving into ARM TrustZone and Intel Software Guard Extensions (SGX), we can gain valuable insights into the specific attributes, constraints, and development opportunities offered by these TEE technologies. This subsection aims to provide an analysis of two TEE-enabling technologies: ARM TrustZone [157] and Intel SGX [158]. These technologies have been selected for examination due to their market availability and support for third-party development.

However, it is crucial to note that at the time of writing, the version of Intel SGX detailed here is deprecated in consumer-grade CPUs. It is anticipated that only server-grade CPUs will continue to provide support for Intel SGX [159]. Furthermore, there have been rumors of a new version of Intel SGX, but specific details are currently limited, and its future remains uncertain [160].

ARM TrustZone

ARM TrustZone [157] is a set of hardware security extensions present in a wide array of ARM processors, ranging from cost-effective and less powerful chips to high-end processors. This technology permits applications to operate in either a secure state (TEE) or a non-secure state (REE), with the processor exclusively executing in one of these states at any given time. The underlying system ensures a secure context switch between these two states and regulates access to its resources. Notably, this architecture does not rely on separate hardware components for each environment. Instead, a secure monitor manages the context switch between the two worlds (REE and TEE) at the hardware level. Depending on the processor generation, this secure monitor may be an independent component within the processor or integrated directly into the processor's logic [161].

While ARM TrustZone is primarily designed as a security technology, it also functions as hardware-supported virtualization technology [161]. This means that each execution environment within TrustZone can host its own OS, offering flexibility to developers. Depending on the application, developers may create a software library that resides in the TEE and is invoked from the REE or utilize a Secure OS designed for the TEE's purpose, often featuring a streamlined set of features to keep the TCB as small as possible. In contrast, the REE typically utilizes conventional OSs [141].

Developers often use a TEE built upon ARM TrustZone as a development framework for their applications rather than implementing applications directly on ARM TrustZone itself. The GlobalPlatform standards facilitate this process because ARM TrustZone implements multiple GlobalPlatform standards, which enable interoperability among different TEEs [162]. Examples of TEEs used in conjunction with ARM TrustZone include OP-TEE [163], SierraTEE and OpenTEE [164].

Sharing hardware components between the TEE and REE introduces inherent risks, as REE applications can disrupt TEE applications. For instance, REE applications may interfere with TEE execution by generating interruptions, forcing context switches, potentially resulting in denial-of-service (DoS) scenarios. Additionally, the shared CPU cache becomes susceptible to side-channel attacks, wherein malicious applications exploit cache vulnerabilities to extract sensitive information. To mitigate DoS attacks, ARM TrustZone allows for interrupt prioritization configuration, giving precedence to secure world interrupts. However, developers must activate this feature explicitly [161].

ARM TrustZone CPUs share their cache between applications in both worlds, though cache access is controlled by a tag bit that indicates the cache's assigned world. While non-secure world applications cannot access the cache assigned to secure user world applications, this setup can still be exploited in various ways, from rootkit attacks that evade detection [165] to multiple side-channel attacks capable of retrieving cryptographic secrets from the rich world by monitoring cache activity [166]–[168].

It is important to note that ARM TrustZone secure world has full access to the memory of the untrusted world, introducing the possibility of boomerang attacks [169]. These vulnerabilities allow a non-secure world application to exploit a TEE application to access memory it should not have access to.

Lastly, ARM TrustZone does not specify a RoT or a secure storage method, placing the onus on system designers to develop effective solutions for these challenges. This complexity can result in a lack of authenticity and integrity guarantees in devices that lack dedicated hardware modules for these critical functions [161], [170].

Intel Software Guard Extensions

Intel SGX consist of a set of Intel CPU instructions designed to offer integrity and confidentiality to computations, even in the event of a compromise of privileged software, such as the kernel or hypervisor [158]. At the core of Intel SGX is a trusted container, referred to as an enclave, protected by trusted hardware. Enclaves only accept applications signed by a trusted entity, currently Intel, and can undergo remote attestation. Each CPU can host multiple enclaves, and each enclave can house multiple applications.

The data and code of an enclave are stored in the Processor Reserved Memory (PRM), a subset of DRAM reserved for enclaves. Inside the PRM, there is an Enclave Page Cache (EPC), divided into multiple pages, with each page assignable to a single enclave. Enclaves cannot access pages assigned to other enclaves. Pages may have various types, from those mapped to the enclave's address space to metadata used in their lifecycle [158].

To secure an EPC when it must reside in untrusted memory, SGX encrypts and signs it to guarantee confidentiality and integrity. Untrusted applications can only write to the PRM during the loading stage of an application into the enclave, a process that is cryptographically hashed and used for software attestation.

Enclave virtual memory may incorporate memory mapped from the outside world, allowing enclaves to use existing libraries from the non-secure world or act as libraries for processes outside the enclave. In such cases, non-enclave software cannot access PRM memory [158]. However, when transitioning from an enclave to an application outside the enclave, the CPU saves its state to a predefined area and clears its registers to prevent data leakage.

Intel SGX enclaves run at the lowest possible privilege level (user mode), making enclave application development similar to non-enclave applications. Developers have a set of libraries and a Software Development Kit (SDK) to compile and deploy applications [171]. Additionally, multiple SDKs exist that build upon Intel's SDK to facilitate secure SGX application development [172], [173].

Enclaves must adhere to the same security constraints as non-enclave applications and are restricted from direct interaction with computer devices [158]. However, the inherent secrecy of enclave software raises security concerns. Traditional antivirus software typically scans executables, files, and memory for patterns indicative of malicious behavior, which SGX technology can potentially evade, rendering a malicious actor residing within an enclave undetectable [174].

From a physical security perspective, SGX's threat model excludes hardware attacks on the CPU chip but considers attacks on the DRAM, its bus, and debugging ports [175]. Attacks on CPU chips are complex and require expensive equipment, making them rare. However, researchers have analyzed Intel SGX-related patents, and while the technology's intrinsic characteristics are not publicly known, they have identified countermeasures to increase the difficulty of CPU chip attacks [158]. These measures may include hardcoding keys with fuses in the CPU circuit or using PUFs to generate keys.

Intel SGX is susceptible to several side-channel attacks, some of which are specific to Intel processors. These attacks leverage processor features like hyper-threading and speculative execution, designed to optimize instruction execution. Hyper-threading partitions physical cores into logical cores, allowing simultaneous execution of multiple threads, with these logical cores sharing essential resources. Speculative execution optimizes instruction pipelines by processing steps in parallel for different instructions , even before the outcome of branch instructions is known [176].

These optimizations have been exploited to create various attacks against SGX. Branch prediction attacks target the component responsible for predicting execution flow before a branch instruction. These attacks have been used to extract a private key stored within an enclave [172], [176]. Vulnerabilities related to speculative execution, like: Spectre [177] and Meltdown [178], have initially targeted non-enclave applications but have also been adapted to attack SGX enclaves, exploiting the same principles [179], [180].

More recently, microarchitectural data sampling attacks have introduced a new category of attacks capable of breaching established security boundaries, including enclaves. These attacks exploit vulnerabilities in undocumented buffers to leak information [181]–[183]. Fortunately, these vulnerabilities can be addressed through microcode updates to the processor [173].

SGX establishes a RoT to ensure the confidentiality and integrity of the TEE. Only properly signed enclaves can be installed. SGX also offers attestation capabilities, but neither of these features is supported by hardware. Each SGX-enabled CPU comes with a privileged enclave, the *quoting enclave*, installed by Intel. This enclave measures the data and code loaded into each enclave and offers remote attestation capabilities. The measurements it provides are similar to those offered by TPMs, but the signature algorithms are different and are not implemented in tamper-resistant hardware [184].

To sum up, Table V summarizes the different advantages and disadvantages of each TEE technology.

Technology	Pros and cons
Intel SGX	+ No additional hardware − Deprecated on Intel Core processors − Requires an expensive CPU
Arm TrustZone	+ No additional hardware + Available in a wide range of CPUs − No standard secure storage or RoT

TABLE V: Comparison between Intel SGX and Arm TrustZone

Security attacks and countermeasures

As previously mentioned TEEs generally share their hardware resources with the rest of the computing environment, which introduces certain security risks. In this subsection, we will examine the consequences of this, affecting both the TEE implementations discussed earlier.

First and foremost, it is essential to recognize that TEEs do not protect against software vulnerabilities. Consequently, if a security flaw exists in the implementation of the TEE or its associated SDK, the security of the TEE can be compromised. Over the course of TEE development, both ARM TrustZone and Intel SGX have faced software vulnerabilities. For example, Bulck et al. [185] identified several vulnerabilities in Intel SGX SDKs. Researchers have also identified several software vulnerabilities in ARM TrustZone TEEs [161]. At the time of writing, the NIST vulnerability database had documented 73 vulnerabilities related to ARM TrustZone [186]. It is important to note that many of these vulnerabilities affect TEE implementations rather than ARM TrustZone itself, as ARM TrustZone is a bare-metal technology that developers use to create their TEE implementations [187].

On the hardware side of TEEs, side-channel attacks pose a significant threat. Cache-based attacks, in particular, target TEEs. In such attacks, a malicious application running in the untrusted environment shares the same processor core with the TEE application. The malicious application fills the processor cache with its data and waits for the TEE application to execute, causing the TEE's data to be evicted from the cache. Subsequently, the malicious application accesses the

same data and measures the time it takes to retrieve the information. By exploiting the faster access to data stored in the CPU's cache compared to RAM, the malicious application can infer access patterns of the TEE. With knowledge about the TEE's code, it can further deduce information about the TEE's execution [173]. These principles have been used to develop various attacks, posing risks to both Intel SGX and ARM TrustZone [166]–[168], [188], [189].

Additionally, TEEs are vulnerable to other types of side-channel and hardware attacks. The literature indicates that both Intel SGX and ARM TrustZone are susceptible to attacks like EM attacks, power analysis attacks, and fault injections [158], [161]. Researchers have demonstrated these attacks in practice. For instance, Bukasa et al. [190] conducted EM attacks against ARM TrustZone, and Chen et al. [191] performed a voltage glitch attack on an Intel SGX enclave. In both cases, the researchers were able to recover cryptographic keys from the TEE.

Intel SGX's threat model excludes physical threats against the CPU chip due to the substantial cost of hardening a general-purpose CPU against such attacks. However, the threat model does include threats against the bus connecting Random Access Memory (RAM) to the CPU, primarily due to the risk of eavesdropping attacks. Consequently, any data that SGX needs to store in RAM is appropriately encrypted and signed. Nevertheless, information inside the internal CPU buses is transmitted in clear text [158], [192].

In summary, TEEs offer valuable security features to enhance resilience against software attacks. Nevertheless, TEEs lack protection against most hardware attacks, and if compromised, they can be used as a persistence method for attackers [155].

Advantages and disadvantages for identity assurance

The primary advantage of using a TEE for identity assurance is the ability to establish a secure execution environment without the need for additional hardware components, which results in cost savings and reduced power consumption compared to alternative solutions. However, it is important to note that in the case of ARM TrustZone, additional hardware adaptations may be necessary to ensure a RoT and enhance the overall security of the system. These adaptations may involve incorporating supplementary hardware elements to reinforce the security foundation provided by the TEE.

TEEs can offer better performance compared to solutions like SEs [193], particularly in terms of CPU, RAM, and storage capabilities. For example, CPUs that incorporate ARM TrustZone technology can have gigabytes of RAM, whereas SEs typically have much smaller RAM capacities. However, developers must acquire proficiency in a new software stack to develop solutions using TEEs. Furthermore, they must have a clear understanding of the limitations of TEE technology. While it provides security benefits, it does not offer complete defense against all hardware attacks. Additionally, sharing components with the non-secure world introduces new risks that need to be carefully considered.

Implementations

TEE solutions have primarily been used for storing cryptographic keys, providing remote attestation, identity verification, and ensuring the security and resilience of applications even if the device is compromised. Most of these solutions are based on public key cryptography, where each device has a private key used for device identification. Several solutions follow these principles. For instance, Ling et al. [194] developed a system that provides secure boot and remote attestation for IoT devices, leveraging ARM TrustZone. This research uses ROM and eFuses to overcome the secure storage limitations of ARM TrustZone and ensure the security of a RoT.

Lesjak et al. [193] conducted a study to compare two authentication systems: one based on an ARM TrustZone-enabled CPU and the other on an SE. The study aimed to analyze the advantages and disadvantages of each approach, and concluded by proposing a hybrid system that combines the strengths of both technologies to mitigate the security risks associated with ARM TrustZone.

Wang et al. [195] developed a solution using Intel SGX that offers a lightweight alternative to remote attestation compared to TPMs. This solution provides a secure environment for running applications and offers the benefit of reduced complexity and overhead.

Durand et al. [196] developed a lightweight communication system backed up by hardware. However, since Intel SGX CPUs are too expensive for most IoT devices, they used an SE in the device and an Intel SGX enclave in the server to securely receive the device's communications.

6) PUFs:

PUFs are physical primitives designed to generate a unique response when provided with a specific input, known as a "challenge" [197]. The uniqueness and unpredictability of the response arise from the unique hardware characteristics of each device, which result from variations in the physical manufacturing process [198]. The pair of challenge and corresponding response is referred to as a Challenge-Response-Pair (CRP) [199]. PUF is a generic concept encompassing various systems from different application fields. However, some constructions outside the field of hardware security engineering are not referred to as PUFs [18].

Many researchers consider PUFs a technology that can help address device identity challenges in the IoT [200]–[202]. This is due to the potential of PUFs to provide a cost-effective and secure means of generating and storing cryptographic keys compared to EEPROM solutions with similar assurance levels [203].

PUF constructions are evaluated using two key metrics: intra-distance and inter-distance. Intra-distance measures the Hamming distance between two responses from the same PUF instance when the same challenge is applied. Inter-distance, on the other hand, qualifies the Hamming distance between responses from different PUF instances when the same challenge is used [18].

In addition to these metrics, researchers also assess PUF reproducibility and uniqueness. The distribution of intra-distance responses provides insights into PUF reproducibility. PUFs are not mathematical functions because a single input

(challenge) can produce multiple outputs due to changes in the physical environment and random noise in response generation [204]. Therefore, reproducibility is a crucial characteristic of PUF-based systems. PUF solutions often employ fuzzy extractors to handle these variations and produce a stable response [204]. The distribution of inter-distance responses assesses the uniqueness of a PUF. Different PUFs challenged with the same input should yield distinct responses. Ideally, the inter-distance responses should be around 50% for a PUF to be considered a true random generator [203]. If a PUF is both reproducible and unique, it is also identifiable. This means that using a PUF response to identify a device is feasible because the response is both unique and stable [18].

The security promise of PUFs is based on three essential characteristics: tamper-resistance, unclonability, and unpredictability. Tamper-resistance involves the ability to resist unauthorized physical modifications aimed at extracting information or bypassing security protections. PUF constructions rely on precise measurements of physical features, and any slight variation in these features results in a change in the CRPs, effectively creating a new PUF instance [18]. PUFs are unclonable due to these precise measurements. In any cloning attempt, the attacker cannot reproduce all of the PUF characteristics because they stem from variations in the manufacturing process [18].

From a security perspective, the predictability of PUFs distinguishes them into Weak PUFs and Strong PUFs. This categorization is based on the system's resistance to an attacker attempting to predict all possible CRPs [205], [206]. Strong PUFs possess the property of being resistant to prediction, even when subjected to extensive attacks over an extended period. In contrast, Weak PUFs do not meet this requirement, making them susceptible to prediction by an attacker [18].

Strong PUFs have a large set of CRPs, preventing the creation of a comprehensive database with all possible pairs. Even if attackers have knowledge of a significant subset of CRPs, they cannot predict the unknown ones. Furthermore, if the attacker physically possesses the PUF, the security of a Strong PUF is not compromised [203], [207]. Weak PUFs, on the other hand, may have only a single CRP. Consequently, the security system is compromised if an attacker obtains its response [203], [208]. Moreover, most Weak PUFs having a single CRP, allow device cloning [207].

Strong PUFs offer higher security guarantees than Weak PUFs. However, some Strong PUF constructions have become vulnerable to modeling attacks due to advancements in new technologies. These attacks require a substantial number of CRPs to model the PUF response. Researchers have been aware of these attacks since the inception of this research field. Nevertheless, with the advancement of machine learning techniques, Strong PUF constructions, previously considered secure are now considered vulnerable to these attacks [209]–[213].

The vulnerability of Strong PUFs to these attacks has spurred significant interest in recent research. As a result, some authors argue that the development of Strong PUFs remains an ongoing research challenge, as these attacks have compromised the expected security features. There is a need for further advancements and refinements to enhance the robustness and resilience of Strong PUFs against these emerging attack vectors [18], [214], [215].

Security attacks and countermeasures

PUFs offer a potential security improvement in storing cryptographic keys, particularly by eliminating offline attacks because the information is not stored. However, it is important to note that PUFs lack countermeasures against hardware attacks. Any attempt to tamper with or inject a fault into the device would induce changes in the PUF's response, making it detectable. However, depending on the specific construction of PUFs, they may still be vulnerable to side-channel attacks or reverse engineering attempts to model the PUF's response.

For example, some delay-based PUFs, like the arbiter PUF, which rely on measuring delays between two competing signal paths, can be vulnerable to power-side channel analysis [216] and optical side-channel attacks [217]. Similarly, RO PUFs can be vulnerable to EM side-channel attacks [218].

The complexity of PUFs is often cited as a factor that makes them resistant to reverse engineering attempts [18].

However, researchers have demonstrated that some PUFs, like SRAM PUFs, can be attacked using SEM to model their responses. SRAM PUFs exploit the probabilistic behavior of memory cells after power-on/power-off cycles [219]. While they have the advantage of being low-cost and simple to implement, these advantages can also make them easier for attackers to compromise [220].

To mitigate these vulnerabilities, PUFs can be improved through design modifications or the incorporation of countermeasures. For instance, new SRAM PUF constructions might include an asynchronous reset mechanism for memory cells to decrease information exposure [221]. Researchers have also suggested changes in the design of RO PUFs to reduce the emanation of EM radiation [218]. In summary, while the reliance on physical characteristics protects PUFs against tampering attempts, they remain vulnerable to side-channel attacks, which require additional countermeasures for mitigation.

Advantages and disadvantages for identity assurance

PUFs offer several advantages in the context of IoT key management. They are a cost-effective solution when compared to other methods, such as anti-tampering EEPROMs, for storing cryptographic keys. PUFs also require less circuit space and energy to operate [203], making them practical choice for resource-constrained IoT devices. Additionally, when Strong PUFs successfully withstand modeling attacks, they can provide lightweight authentication and identification protocols that do not rely on resource-intensive cryptographic algorithms [222].

PUFs are particularly suitable for managing device identity and ensuring the security of IoT devices against invasive attacks like tampering or key extraction. However, as mentioned earlier, both Strong and Weak PUFs may have vulnerabilities that need to be addressed. Therefore, any system using this technology should be aware of these risks and implement appropriate safeguards to mitigate them.

It is also crucial to consider the security of the server where the CRPs are stored when using strong PUFs. In many systems, the assumption is made that the server is secure [223]. However, in certain scenarios, the risk of a compromised server may be deemed unacceptable, as it could lead to the compromise of the confidentiality of all CRP pairs and enable attackers to impersonate any device within the system. This highlights the importance of securing not only the PUFs but also the infrastructure that handles their data.

Implementations

In identity assurance, we observe the use of PUFs in two primary scenarios: the generation of secure keys and authentication. Specifically, Strong PUFs are employed for authentication, while Weak PUFs are utilized for key generation [203].

A typical authentication protocol entails conducting multiple secure challenges on a manufactured PUF. The resulting responses, known as CRPs, are securely stored and used for device authentication. To prevent replay attacks, each challenge is employed only once, ensuring the reliability of the authentication process [215], [224]. This type of authentication protocol has been in use since the inception of this technology. Early examples include Gasend et al.'s silicon PUF [225] and Lim et al.'s arbiter-based PUF [226]. However, the threat model of these initial implementations was overly restrictive, rendering them vulnerable to modeling attacks [227].

To thwart modeling attacks, the subsequent generation of PUF constructions started using one-way functions applied to the PUF response to hinder direct access to the PUF CRP [224], [228]–[230]. This type of PUF is generally referred to as a control PUF. However, further research uncovered vulnerabilities in this construction due to reversible one-way functions and pattern matching [210], [231].

Current research is focused on enhancing existing constructions to make them more resilient to modeling attacks. The primary approach is to secure the PUF interface by implementing mutual authentication [232]–[235]. Research efforts have also been dedicated to preventing modeling attacks while improving the authentication protocol.

Other researchers, such as Chatterjee et al. [236] and Qureshi et al. [223], have proposed solutions that eliminate the assumption of a secure CRP database. This means that even if the authentication server is compromised, the CRPs remain protected as they are not stored in plain text.

Ebrahimabadi et al. [237] have made advancements in mitigating the eavesdropping of CRPs and addressing modeling attacks. They have devised an authentication protocol that scrambles and divides the communications between the server and a node (the device with a PUF that needs to be authenticated) into multiple packets. These packets are sent through multiple nodes to obscure the actual destination of the message. With this approach, the authors anticipate that attackers cannot correlate the CRPs with a specific device, making it virtually impossible to perform a successful modeling attack.

Hang et al. [238] employ the classic authentication protocol with Strong PUFs. However, they combine multiple PUFs on the same device to create a fingerprint that changes if its components are tampered with. This construction uses a configurable RO PUF as a hardware security primitive and a latch structure to extend the key space of the responses, thereby increasing its resilience. This solution follows a similar authentication protocol, where a server stores multiple CRPs and performs queries to authenticate the device.

On the other hand, Weak PUFs used in IoT systems do not attempt to replace traditional authentication systems. Instead, they aim to enhance the storage of cryptographic keys on affordable IoT devices. A Weak PUF generates the same secret key every time the device runs. Consequently, this key does not need to be stored, mitigating the risk of an attacker extracting the key when the device is not operational. For this generation process to be reliable, the secret key extraction from a Weak PUF response requires an additional step: a *helper data algorithm* [2]. Any alteration in the cryptographic key is unacceptable for most encryption protocols. Therefore, this algorithm facilitates the extraction of a secret key from a noisy or non-uniform response [239], [240].

The generation process of a secret key from a Weak PUF typically consists of two phases: enrollment and reconstruction. During enrollment, a new secret key is generated, while reconstruction is the process of retrieving the same secret key after its initial creation. In the enrollment phase of a PUF, a secret key and helper data are generated from the PUF response. Keeping the secret key confidential is essential, but the helper data does not require the same level of secrecy and can be stored in a non-secure NVM. The reconstruction phase utilizes the obtained helper data to generate the same secret key when presented with another PUF response from the same PUF [241], [242]. The first practical work utilizing PUFs to securely manage cryptographic keys was carried out by Škorić et al. [241] and Suh et al. [243]. Over the years, new research in this field has emerged. Some of this research involves the use of novel types of PUFs for secret key generation [219], [244], while others focus on creating re-configurable PUFs that enable the change of secret keys over time [242], [245], [246]. In this body of research, Strong PUFs are also employed due to the abundance of CRPs, which enable the generation of multiple keys on the same PUF instance [239]. More recently, researchers have been developing PUF-based systems capable of generating a shared key among different resource-constrained devices to facilitate multiparty communication [247].

In summary, there are two primary approaches to enhancing authentication and identity on IoT devices using PUFs: lightweight authentication protocols using CRP databases and secret key generation. Authentication protocols with CRP databases rely on the resilience of the PUF against security attacks. However, when properly implemented, this technology meets the needs of resource-constrained IoT devices by providing an energy-efficient way to authenticate a device without requiring intensive computation. On the other hand, PUF-based key generation serves as a security primitive for existing identity and authentication systems. While it does not

[2] A *fuzzy extractor* is a specific type of *helper data algorithm* [239].

Technologies		Lightweight cryptography	Object identification	Secure storage
TRNG		•		
Masked ROM and OTP memories	Masked ROM			•
	Floating gate OTP			•
	Fuse OTP			•
	Anti-fuse OTP			•
Crypto accelerators	Crypto instruction sets	•		
	Crypto-coprocessors	•		
	Crypto-processor	•	•	•
Secure element		•	•	•
TEEs	Intel SGX	•	•	•
	Arm TrustZone	•		
PUF			•	•

TABLE VI: Technologies and associated research challenges

eliminate the need for substantial computation, it offers a cost-effective and secure means of generating and storing unique secret keys.

C. Summary

The various hardware technologies discussed in this section have the potential to contribute to the development of IoT identity systems. However, it is crucial to acknowledge that these technologies differ significantly in their capacity to support such systems. Additionally, when deciding which technology to incorporate into a device, designers must carefully evaluate the strengths and weaknesses of each option. In this assessment, we examined how these technologies can benefit IoT identity systems and contribute to addressing the challenges associated with IoT identity. We also considered their respective advantages and disadvantages.

Section II-B identified three key challenges for developing an identity system: lightweight encryption, object identification, and secure storage. The relationship between these challenges and various technologies is summarized in Table VI.

Each encryption accelerator, SEs, and TEEs can provide ways to address the need for lightweight encryption algorithms. All these technologies can efficiently run cryptographic algorithms, making these operations faster and more energy-efficient. Among these technologies, it is essential to highlight the cryptographic instruction sets (a subtype of the cryptographic accelerator) because they can potentially standardize the implementation of cryptographic algorithms in hardware. Strong PUFs and TRNGs, albeit indirectly, also contribute to the research challenge in lightweight cryptography. TRNGs may not optimize cryptographic algorithm execution but provide high-entropy numbers, essential for any asymmetric encryption scheme. PUFs can be leveraged for lightweight authentication systems, alleviating the reliance on cryptographic algorithms.

In terms of object identification, Intel SGX and SEs offer an isolated execution environment and secure storage capabilities that can be leveraged to develop object identification systems. Another technology with similar potential is TPMs. However, developers are constrained by the features specified in the TPM standard. Taking a different approach, we have strong PUFs, which rely entirely on the hardware characteristics of the device and can be used to uniquely identify a device.

Masked ROM and OTP memories provide read-only memory options for storing RoTs. However, they do not provide data-at-rest protection, unlike other technologies such as Intel SGX and SEs. TPMs and PUFs can also help address the challenge of secure storage but offer a limited set of features. TPMs are primarily designed for storing cryptographic keys, while Weak PUFs are even more limited as they cannot import cryptographic keys generated outside of the PUF.

Therefore, there are several candidate technologies capable of supporting new identity and authentication systems, as they can address some of the challenges hindering identity and authentication in IoT. It is now up to developers to understand these limitations and integrate these technologies into new systems.

IV. Experimentation

In previous sections, we discussed various attacks that could compromise IoT authentication and device security, as well as different technologies that can be employed to mitigate these attacks and address the challenges highlighted in earlier research. This section explores how these technologies can be integrated to create a device that leverages multiple components to overcome the challenges and threats mentioned earlier.

Our case study focuses on a smart meter connected to a utility company via a mesh network. In many instances, such devices incorporate two processors [30], [70]. The first processor is primarily responsible for accurately measuring electricity consumption, while the second facilitates external interactions by enabling communication with the utility company via radio, establishing a means for transmitting relevant data and receiving instructions. This processor may also include a Home Area Network (HAN) port [71], a physical interface through which device settings can be changed, and measurements can be obtained.

Our primarily focus is the processor responsible for external communication. Based on the technology employed in the mesh network, these processors can incorporate different types of radios. Commonly, these networks use protocols based on IEEE *802.15.4* [248], such as Zigbee and 6LoWPAN [70], [249]. For this purpose, we have selected the Espressif ESP32-C3 [250], a 32-bit RISC-V microcontroller with several hardware security features, including cryptographic accelerators.

We opted for this microcontroller as an alternative to the Espressif ESP32-H2 [251], which offers similar capabilities and support for 802.15.4 but was unavailable for purchase at the time of writing.

The Espressif ESP32-C3 features cryptographic accelerators supporting both symmetric and asymmetric encryption, hash algorithms, a TRNG, and OTP memory. These features serve as fundamental building blocks for implementing advanced functionalities such as secure boot or external flash encryption. However, it is important to note that leveraging these features and primitives requires explicit enabling and utilization by the device's developer. At the core of the ESP32-C3's security features is the eFuse OTP memory located inside the processor. This memory is divided into 11 blocks, each containing 256 bits, serving various purposes. Some blocks are reserved for system-related functions, while others store essential information like the device's MAC address or disabling debugging capabilities. Additionally, certain blocks are allocated for storing cryptographic keys and user data. By working in conjunction with the ESP32-C3's eFuse controller, it is possible to configure access controls for memory blocks. For example, certain eFuses can be configured to be unreadable from outside the chip or only readable by cryptographic accelerators [252].

Secure boot and flash encryption rely on the utilization of these features alongside cryptographic accelerators, enabling a hardware-centric implementation. This allows for the use of standard cryptographic algorithms, such as AES and RSA, without compromising device performance or power consumption. The secure storage of cryptographic keys is ensured through the application of eFuse blocks and an eFuse controller, guaranteeing key immutability and restricting key access solely to the required cryptographic accelerators. Additionally, the integrated TRNG enables the generation of cryptographic keys internally within the device [252].

In addition to these features, the ESP32-C3 includes defenses against voltage glitching attacks. The ESP32-C3's clock can be driven by an external crystal, which may be susceptible to clock glitch attacks. In this type of attack, the attacker introduces a momentarily higher-frequency signal to disrupt the regular functioning of the targeted device. The microcontroller includes a sensor that detects abnormal clock pulses and resets the board to prevent the exploitation of this attack [252]. For voltage glitches against its power supply, the device has a brownout detector. This mechanism senses the voltage provided, and if the voltage falls below a specific threshold, it resets the board.

The ESP32-C3 does not have a TEE developed by Espressif, but it does have a physical memory protection unit that can be used by software to restrict access to specific regions of memory [252]. This feature has been leveraged by researchers to implement a TEE for the ESP32-C3 [253]. Therefore, despite being an affordable SoC, the ESP32-C3 provides a wide range of security features and defenses against hardware attacks, as detailed in Table VII.

Our goal is to explain how hardware attacks can be executed on an off-the-shelf board and how hardware can be leveraged to mitigate these attacks. For our experiments, we will use an Ai-Thinker ESP-C3-13 [254], a development board equipped with an Espressif ESP32-C3 SoC [250].

Security primitives	- eFUSE OTP memory - True random number generator - Crypto accelerators (AES, RSA, SHA, HMAC)
Security features	- Secure boot - Flash encryption - TEE (third-party support)
Countermeasures	- Brownout detector - Clock signal sensor

TABLE VII: Security capabilities of the ESP32-C3

Subsection III-A4 analyzed hardware attacks, categorizing them into three distinct groups. For the purpose of the conducted study, we executed one attack from each group to effectively demonstrate their impact and feasibility, namely: voltage glitching (fault injection), timing attack (side-channel attacks), and PCB-level attack (reverse engineering). To facilitate the execution of these attacks, a custom firmware was developed on the software side, which provided an interface accessible through serial communication. This firmware aimed to emulate the functionalities typically found in a Home Area Network (HAN) port, which simulates a restricted set of features intended for technicians and safeguarded by a key. Given the conventional exposure of such ports to external access, they represent a logical target for potential attackers.

Espressif provides two development frameworks for this device: ESP-IDF and Arduino-esp32 [255]. Unfortunately, both are based on RTOS, which hinders the execution of our experiments since concurrent threads are running in the device. Therefore, to ease the reproducibility of our demonstration, we will create bare-metal software using *mdk* SDK [256], a third-party SDK for bare-metal development on the ESP32-C3.

The software employed in this demonstration utilizes a user-input password verification mechanism. The program waits for the user to input a password, compares each character of the inserted password with a predetermined value that is hardcoded within the firmware, and provides feedback regarding the accuracy of the entered password. The complete source code of this software can be accessed from the GitHub repository located at https://github.com/MrSuicideParrot/esp32-c3-attacks. Furthermore, this repository is a comprehensive resource for reproducing the demonstration, encompassing detailed instructions, attack scripts, and firmware implementations incorporating countermeasures against the demonstrated attacks. Researchers and readers are encouraged to refer to this repository as an auxiliary tool to understand the subsequent subsections.

A. Fault injection - voltage glitch

Voltage glitching is a specific case of fault injection. This attack occurs when malicious actors modify a circuit's voltage to induce software errors. In our case, we will attack the SoC's power supply with the goal of under-volting the device in a way that induces a malfunction.

The first step in performing this attack is understanding the target's power domains. The ESP32-C3 Technical Reference Manual [252] specifies that this SoC has nine power domains

organized into three groups: *Real Time Clock (RTC)*, *digital*, and *analog*. The *RTC* group, as the name implies, includes the RTC and the power management unit. The *digital* group is responsible for powering the digital part of the SoC, which includes different cryptographic accelerators, the CPU, and digital pins. Finally, the *analog* group is in charge of the embedded radios. The ESP32-C3 datasheet [250] describes how these groups manifest in the physical world. The ESP32-C3 has four power supply input pins: *VDDA1*, *VDDA2*, *VDD3P3_RTC*, and *VDD3P3_CPU*, which directly correspond to the power domain groups.

The power supply for various groups within the system is as follows:

- *VDDA1* and *VDDA2* provide power to the *analog* group.
- *VDD3P3_RTC* powers the *RTC* group.
- *VDD3P3_CPU* is responsible for powering the *digital* group.

Given our objectives, a fault should be injected into the energy source responsible for powering computation operations, which in this case is the *VDD3P3_CPU*. Additionally, to successfully execute this attack, this pin should be isolated from the rest of the board to inject a faulty voltage, and any decoupling capacitor that may affect this power line must be removed. A decoupling capacitor suppresses high-frequency noise in power supply signals, which means any temporary drops in its voltage will be rectified. Therefore, any decoupling capacitor would decrease our probability of success since it will suppress any slight variation in the voltage feed to the circuit.

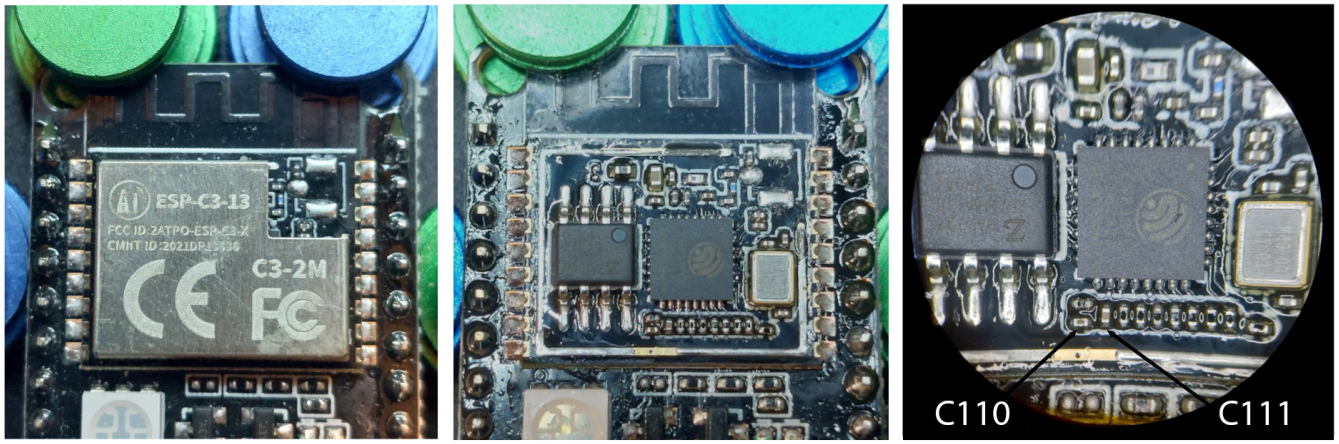
Typically, SoC datasheets explain the different decoupling capacitors that should be present to ensure the reliable operation of an SoC. This should be the starting point for any attacker. In our case, it is no different; the ESP32-C3's datasheet [250] states that *VDD3P3_CPU* should have a decoupling capacitor attached to it, and it is also responsible for supplying power to *VDD_SPI*. Consequently, any capacitors on this line should be removed to minimize the potential for interference in our attack. The ESP-C3-13 development board specification includes a diagram of the connections and the associated decoupling capacitors identified as C110 and C111 (see Schematic 5 - on page 14 of [254]). Unfortunately, in our case, the PCB is not labeled, which means it is necessary to manually identify the components we mentioned earlier and remove them.

The components can be identified with the help of a multimeter in continuity mode to check for electrical continuity and an optical microscope to follow the trace connected to the SoC's pin that corresponds to our target. Using this process, we identified the two decoupling capacitors, C110 and C111 (Figure 5C), that should be removed with a soldering iron or a hot air rework station. In the case of the ESP-C3-13, we had to perform an intermediary step before proceeding with the component's identification, which involved removing the metal shield protecting these components against electromagnetic interference (Figure 5A and B).

The next step to prepare for the execution of a fault injection is to isolate the target's pin from the rest of the board. The idea is to find a place on the top or bottom PCB layer where the power of the *VDD3P3_CPU* is being routed and with enough room to use a scalpel to cut the electrical connection. This procedure is similar to the one used to find decoupling capacitors, where we should probe the board with the multimeter for electrical continuity and follow the electrical traces until we find a suitable place to cut it. In the ESP-C3-13, we found this place on the bottom layer of the board (Figure 6B) near a via placed beneath the ESP-C3 chip that connects to the *VDD3P3_CPU* pad (Figure 6A).

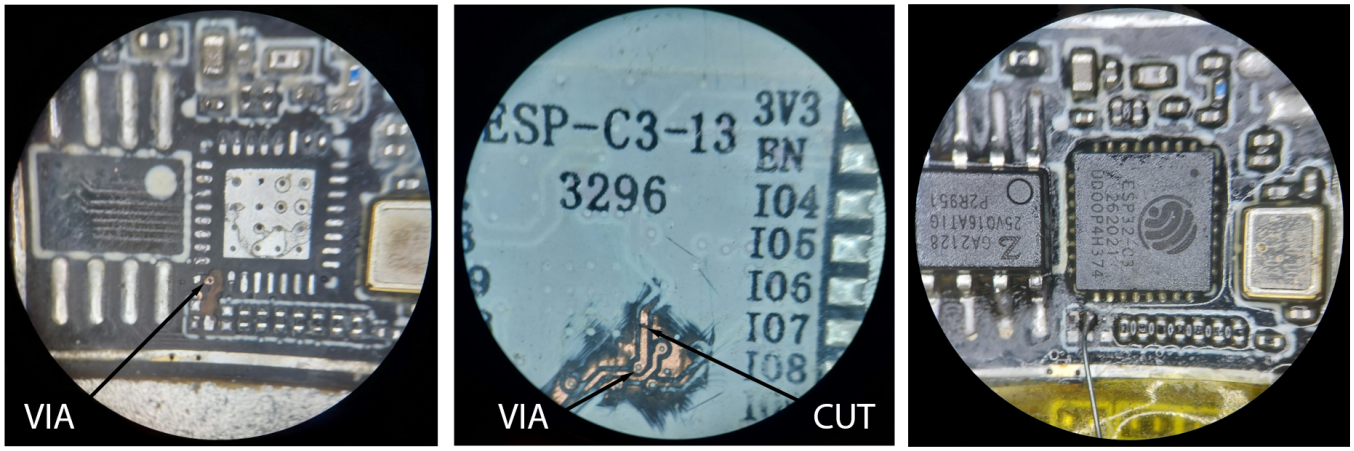
At this point, we have made every physical modification needed to perform this attack, and we should solder a small cable to the target pin's pad to be able to inject power externally (Figure 6C).

On the hardware side, our setup involves the utilization of two analog switches: the TS12A4514P and TS12A4515P [257]. These switches play a crucial role in enabling the transition from the normal operating voltage to the faulty voltage during the fault injection process. In



A - ESP-C3-13 with metal shield B - ESP-C3-13 after removing metal shield C - Decoupling capacitors: C110 and C111

Fig. 5: Steps involved in the identification of decoupling capacitors



A - Top view of the PCB B - Bottom view of the PCB C - Cable soldered to the *VDD3P3_CPU*

Fig. 6: Identification and isolation of the printed circuit board (PCB) trace that feeds power to the *VDD3P3_CPU*

our experimental configuration, we incorporated two external power supplies. The first power supply provides a fixed voltage of 3.3 volts, while the second allows variable voltage adjustments (Figure 7). Additionally, we employ a ChipWhisperer-Nano [258], a dedicated hardware platform designed to receive signals from the target board and control the power supply based on predefined parameters.

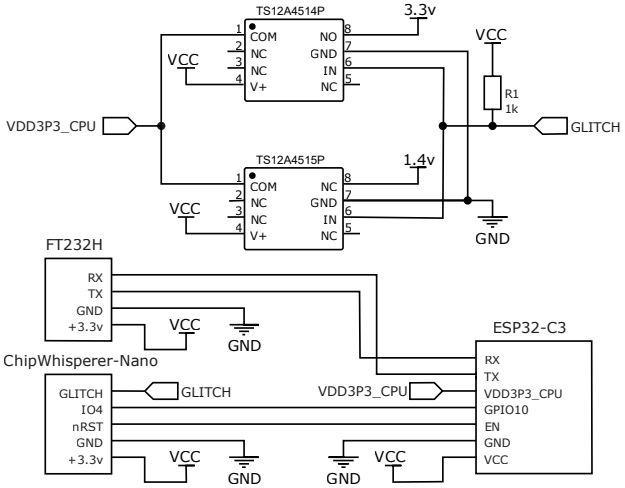


Fig. 7: Schematic showing the connection of the two analog switches, ChipWhisper and the target board

The TS12A4514P and TS12A4515P are two low-voltage analog switches with similar specifications but with opposite behavior. The TS12A4514P is a *normally open* switch, and the TS12A4515P is a *normally closed* switch, which means that when connected, one of them has the switch opened, and the other is permanently closed.

The ChipWhisperer-Nano [258] is a platform developed to ease the education of embedded security. We leveraged its glitch capabilities to trigger the voltage switch. This board is able to receive a trigger in a General-Purpose Input/Output (GPIO) pin and make another GPIO pin change from high to low during the duration and delay we defined. This pin will be connected to the analog switches to change between the two power supplies. However, any microcontroller could be used to control these switches. It is necessary to have firmware that ensures reduced delay between the trigger and the voltage change and offers an API that allows us to customize the duration of the glitch programmatically. We opted for the ChipWhisperer-Nano because it already has optimized firmware for reduced delay between the trigger and glitch and a Python API that enables customization of the different attack parameters.

Finally, to successfully execute an under-voltage glitch attack, we need to manipulate three key variables, namely:

- *Voltage*: This refers to the specific voltage level that is injected into the target device during the glitch attack.
- *Glitch width*: This represents the duration of the glitch itself, signifying the span of time for which the voltage injection is maintained at the chosen level.
- *Glitch offset*: The glitch offset corresponds to the duration between receiving the trigger signal to initiate the glitch attack and the actual commencement of the voltage injection.

All these variables need to be determined through experimentation. An attacker may utilize a development board that shares the same SoC as the target device to pinpoint the precise variables required for a reliable exploit. Using this development board, the attacker can create custom firmware specifically designed to detect malfunctions or abnormalities within the device. Typically, this firmware includes a loop that increments a variable and, at the end, a verification step that checks the result of the algebraic operation and loop counter. Any deviations from the expected behavior indicate a fault or an unintended alteration in the system's execution [17]. In our case, we have full access to the device allowing us to flash a loop and determine the correct parameters. We created firmware that implements the loop test for the ESP-C3. [3] Furthermore, in this firmware, an ESP-C3 pin is set

[3]https://github.com/MrSuicideParrot/esp32-c3-attacks/blob/main/attacks/voltage-glitch/loop-test/main.c

to a "high" logic level when the code enters the loop. This deliberate action serves to facilitate the synchronization of the glitch injection with the code execution.

To discover the *voltage* value to inject, one approach is to test a low-*voltage* value at which the device would reset with the smallest *glitch width* possible, and then gradually increase this value until it no longer resets at the smallest *glitch width*. In our case, we began with a value of 1 volt and ended up with 1.4 volts. We will use this value for the remainder of this experiment. However, it is essential to keep in mind that if we do not achieve the desired results in the next phase, we may need to repeat this step and experiment with other *voltage* values.

Identifying the *glitch width* follows a similar process. We can test the attack against the loop test with incremental widths multiple times. After each execution, we observe and record which parameters did not affect the device and which ones created a malfunction. Additionally, we can experiment with different combinations of different *glitch offsets* to test the effect of the glitch on different instructions.

In our case, we set a *glitch offset* known to target the middle of the test loop (4.15 microseconds) and tested *glitch widths* ranging from 8.3 nanoseconds to 83 nanoseconds, with increments of 8.3 nanoseconds. For each width, we executed the test 50 times. Ultimately, we observed glitches with a *glitch width* of 49,8 nanoseconds. To confirm that our *glitch offset* aligned with the test loop, we used a logic analyzer to monitor the trigger and glitch pin, ensuring that the glitch started while the trigger was active. [4] By following these steps, we successfully identified the parameters required to execute a successful glitch attack on this board, allowing us to bypass the authentication process. [5]

Countermeasures

On the hardware side, the primary countermeasure against this type of attack is a voltage sensor available in the SoC, referred to by this vendor as a brownout detector. This detector resets the board if the voltage provided falls below a specific threshold, which is 2.7 volts by default. One advantage of this sensor is that it is enabled by default, eliminating the need for developers to enable it explicitly. However, it is worth noting that this sensor does not detect over-voltage, meaning an attacker may still be able to exploit the device using this attack vector. In the past, attackers used over-voltage to bypass the device's secure boot [259]. This issue was addressed with a new hardware revision that hardened the device against voltage fault injection attacks. Unfortunately, no further details are provided about the changes that were introduced. Similar to the brownout detector, as mentioned before, the ESP-C3 also includes a clock sensor to detect clock glitches, which is enabled by default.

Nevertheless, we can also implement some software countermeasures in our code. It is important to remember that software countermeasures may introduce performance penalties and often only protect specific code functionality. For instance, to protect key verification against fault injections, we can attempt to deflect or detect the attacks [91]. Attack deflection involves running our check at a random point in time, decreasing the reliability of any attack. Attack detection entails performing the operation we aim to protect multiple times, preferably implemented in different ways. In our case, this would mean verifying the key twice at different times. In summary, both techniques aim to reduce the success rate of any attack rather than completely mitigating it.

A modified version of the firmware incorporating these mitigations, including the activation of the brownout detection, is accessible on the dedicated GitHub repository. [6]

B. Side-channel attack - timing attack

A timing attack is a specific type of side-channel attack that leverages small differences in device response times to extract information about the target. This family of attacks is broad and can encompass any device interface with the exterior or specific signals activated by a particular behavior. For instance, it can involve scenarios such as turning on an LED when password verification is correct or writing to EEPROM at the end of an operation.

Our target has a serial interface protected by a password. We will monitor the time the device takes to respond to our login attempt. If the response time depends on the correctness of our input, we can potentially brute-force each character of the password individually.

For our setup, we used a Saleae Logic 8 logic analyzer [260] to monitor the serial lines of the target device, enabling us to capture and analyze the communication between the device and other connected entities. Additionally, we utilized an FTDI FT232H serial UART adapter [261] to interact with the serial interface and the ChipWhisperer-Nano [258] to reset the target board after each attempt. The hardware setup is similar to the one presented in Figure 7, with the addition of attaching Saleae Logic 8's probes to the *TX* and *RX* lines of the device.

We developed a script [7] that uses the FT232H serial UART adapter to perform authentication attempts on the target device and analyzes the device's response time to our authentication requests with the logic analyzer. The methodology for this attack follows a systematic approach. Attackers select a pool of characters that may potentially be part of the password. Subsequently, they iterate through each index of the password and perform a brute-force attack, attempting all possible characters for that specific index independently. This step-by-step process allows for an exhaustive search for the correct password combination.

To illustrate the process, an attacker initiates the brute force attack by systematically testing all possible characters for the first position of the password, and each character is tested five times. The time the device takes to respond to the

[4] https://github.com/MrSuicideParrot/esp32-c3-attacks/blob/main/attacks/voltage-glitch/LoopTestDiscovery.ipynb

[5] https://github.com/MrSuicideParrot/esp32-c3-attacks/blob/main/attacks/voltage-glitch/VoltageGlitchExploit.ipynb

[6] https://github.com/MrSuicideParrot/esp32-c3-attacks/tree/main/attacks/timing-attack/patched_firmware

[7] https://github.com/MrSuicideParrot/esp32-c3-attacks/blob/main/attacks/timing-attack/TimeBasedSideChannelAttackExploit.ipynb

authentication request is measured, and these measurements are recorded and averaged. After completing this initial step, the collected response times are analyzed to identify the correct character, which corresponds to the longest response time. This character is considered the correct character for the first position of the password, and the process is repeated for the subsequent password positions. At each position, the correct characters from the previous positions have already been determined, allowing for the concatenation of the known partial password with the attempts for the new position. This iterative process continues until all password positions have been successfully identified, resulting in the recovery of the complete password through a series of timing measurements and analysis. With this technique, we were able to brute-force the password that gives access to the debug interface.

Countermeasures

Side-channel attacks can be mitigated by eliminating any information leakage that could be exploited. In our demonstration, we conducted a timing attack based on the information leaked by the response time of the board. However, since this vulnerability stems from software, hardware-based solutions alone cannot provide mitigation. This issue must be addressed at the software level by ensuring consistent response times or introducing randomized delays to prevent information leakage. Even though we only explored this program to perform a timing attack, it is also vulnerable to SPA. The time it takes to execute can be observed in a power trace of the CPU, and even if we implement the countermeasures described above, an attacker can still identify the different patterns of password verification in the power consumption, bypassing this countermeasure. The device should monitor its power consumption and adjust it to ensure that information about the executing operations is not leaked.

The ESP-C3-13 does not include any form of voltage monitoring or a metal shield that could prevent some types of side-channel attacks on the hardware side. Moreover, this problem affects not only the normal execution of code but also the cryptographic coprocessors, which leak information through power consumption. As evidence of this concern, it is documented that attackers have previously exploited similar vulnerabilities in microcontrollers from the same family [262].

Furthermore, an additional software-based mitigation approach involves comparing the user input not directly with the correct key but instead with a masked version of the user input, as proposed by Standaert et al. [263]. For instance, you can store the key hashed, apply the same hash function to the user input, and compare the two hashes. This way, there is no direct relation between the user input and the time it takes to verify the key.

Additionally, the security hardware integrated into this board offers potential support for implementing this solution. The board's cryptographic accelerator can generate a hash of the user input, and the resulting hashed key can be securely stored in the eFuse OTP memory. A patched firmware example that incorporates this mitigation against the side-channel attack, utilizing the cryptographic accelerator, is available on our GitHub repository. [8]

C. Printed circuit board level attack

PCB-level attacks leverage existing features of the PCB to enable reverse engineering of the firmware or its signals. In this case, we will exploit our physical access to the board to interact directly with its EEPROM and read any firmware or secrets stored there.

The first step in executing this attack is to identify the EEPROM on the board in question. The simplest method is to refer to the SoC's datasheet, determine the pins responsible for connecting external memory, and then trace these connections on the board until we find a component. Normally, EEPROMs have their part number engraved on their package. An attacker should use this reference to find its datasheet and select the necessary hardware to interact with it. In the case of the ESP32-C3, we found an EEPROM with the inscription "25VQ16ATIG" (Figure 8), which is a memory supporting the Serial Peripheral Interface (SPI) protocol and works with voltages from 2.3 to 3.6 volts, capable of storing 16 megabits.

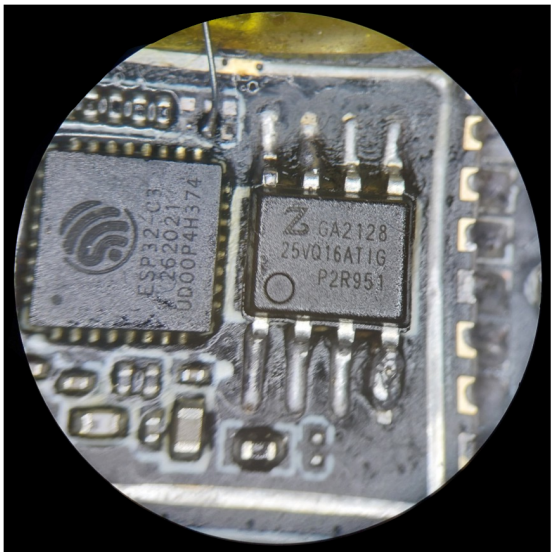


Fig. 8: ESP32-C3's EEPROM

Extracting data from an EEPROM requires a tailored approach, involving a hardware board capable of interfacing with SPI, commonly called a programmer board, and suitable software to control it. Each control software typically has a set of EEPROMs with which it is compatible and should be the basis for our choice. The programmer needs to be compatible with the chosen software and work with the operating voltage of the EEPROM. To read the ZB25VQ16, we selected the XGecu TL866II Plus [264] EEPROM programmer and its companion software, as they support this specific EEPROM model.

From the physical perspective, there are two ways to attach a programmer to the EEPROM: unsolder the IC from the PCB or perform in-circuit programming [265], both with advantages

[8] https://github.com/MrSuicideParrot/esp32-c3-attacks/tree/main/attacks/timing-attack/patched_firmware

and disadvantages. Unsoldering the IC ensures that the read operations of the programmer are not interrupted by actions of the device's CPU. However, this is a more intrusive operation and may damage the IC and the PCB if performed by an unskilled person. On the other hand, in-circuit programming aims to interact with the IC without unsoldering it from the PCB. It is simpler than unsoldering the IC, as for standard ICs, some adapters can be used to attach to the IC's pins without removing it from the PCB (Figure 9). Unfortunately, this approach comes with a few drawbacks. When performing the reading procedure, it is necessary to power on the EEPROM. However, since the EEPROM is still connected to the PCB, powering it on may inadvertently activate other components. If the CPU is powered on during this operation, it can interact with the EEPROM and disrupt the programmer's operations, leading to undesirable interference.

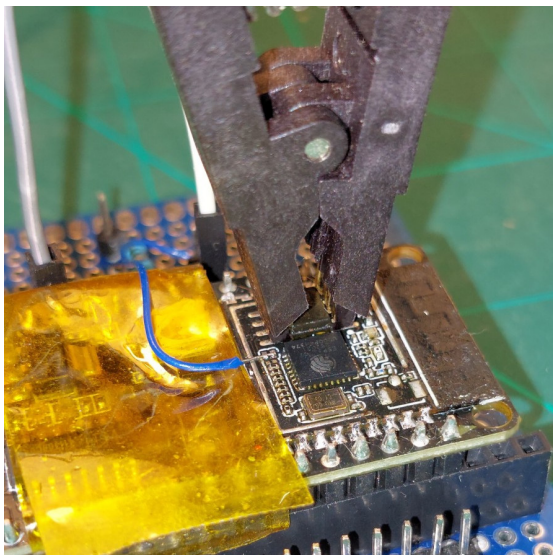


Fig. 9: SOP8 adapter attached to the *ZB25VQ16* to perform in-circuit programming

We opted for performing in-circuit programming. To prevent any interference from the CPU during this operation, we will reuse the board prepared for the voltage glitch since it has its *VDD3P3_CPU* separated from the PCB, and therefore the CPU will not power on. However, the same result can be achieved with built-in features, even if its power supply pin is not isolated. Most CPUs have ways to keep the CPU halted even when it has power. Typically, this is implemented as a GPIO pin that, when in a specific logic level, prevents the CPU from booting. In the case of the ESP32-C3, the pin is the *CHIP_EN*, and when it is grounded, the chip will remain powered off [250].

Upon successfully retrieving the EEPROM data, an attacker transitions to the next attack phase, reverse engineering. The EEPROM dump has different memory regions mapped on it, from program data to file systems that are used to store information. Our goal is to bypass user authentication. Therefore, we must find and disassemble the program data to recover the password. This task is very similar to reverse engineering an executable from a computer. However, computer programs have a standard format that can be easily imported into disassemblers, and firmware requires a custom approach according to our target. We will use the Ghidra disassembler [266] for this attack because it is open-source and supports a wide range of CPU architectures, including RISC-V. Additionally, this software includes a decompiler, which eases the reverse engineering task.

During execution, the microcontroller abstracts the interactions with the EEPROM by incorporating it into its address space. This behavior is known as memory-mapped peripherals and is employed to interact with any peripheral. For example, if the microcontroller wishes to interact with its cryptographic accelerators, it must read and write to a specific memory region. The disassembler needs to be aware of these memory regions and their associations with peripherals in order to disassemble the firmware. The memory mapping information is available in the CPU's technical manual, allowing it to be manually defined in the disassembler.

Additionally, it is crucial to understand the boot process of the microcontroller to load the firmware into memory correctly. The ESP32-C3 has a first-stage bootloader stored in its internal ROM, which cannot be altered. This bootloader initializes the CPU and transfers control to the second-stage bootloader [255]. The second-stage bootloader is stored at offset 0 in the EEPROM and is loaded into memory by the first-stage bootloader. Therefore, the second-stage bootloader can be customized and must be the starting point of our reverse engineering efforts.

We are using the *mdk* [256] SDK to implement our program, so we analyzed this SDK to understand the behavior of the second-stage bootloader. Regardless of our prior knowledge, in many cases, the SDK being utilized can be identified by analyzing the visible strings present in the firmware. The firmware produced by the *mdk* starts with an eight-byte header, with the last four bytes representing the address where the execution of the second-stage bootloader should begin. Subsequently, there is a 16-byte header containing metadata about the firmware. Finally, the firmware consists of multiple memory segments, each delineated by information specifying its memory address and size, facilitating its proper allocation in memory.

With all the gathered information, it is possible to import the firmware into Ghidra as a raw file, create the necessary memory regions, and ensure the correct placement of the firmware in memory. With this, Ghidra can disassemble the firmware accurately, allowing us to delve into the firmware and ultimately retrieve the password. Further instructions and code samples can be found in the corresponding GitHub repository [9] for reference and in-depth analysis.

Countermeasures

With the exception of the embedded EEPROM on the SoC, it will always be possible to interact with the EEPROM by unsoldering it from the PCB or performing in-circuit programming. From a physical perspective, preventing this type of attack is difficult. We can choose EEPROM packages

[9]https://github.com/MrSuicideParrot/esp32-c3-attacks/blob/main/attacks/pcb-level-attack/

that are more challenging to use for in-circuit programming, but a skilled attacker can always unsolder the IC.

Therefore, it is crucial to employ encryption mechanisms to ensure the security of firmware and information stored in the EEPROM. However, implementing flash encryption can be complex, especially on devices that lack native support. One of the primary challenges involves finding a secure location to store the encryption key and having sufficient computational capabilities to perform the encryption and decryption operations on the flash memory.

As stated at the beginning of this section, the ESP32-C3 supports flash encryption based on eFuse OTP memory to store the encryption key and an AES cryptographic accelerator to encrypt and decrypt the flash memory. By default, the flash encryption feature is typically disabled and needs to be explicitly enabled by the developer. This feature operates independently of the running application but requires specific configurations, such as a different partition table and bootloader. Currently, only the ESP-IDF SDK supports this operation [267]. The lack of availability of flash encryption on the ESP32-C3 or other microcontrollers is not due to restrictions imposed by a specific framework. Instead, it is primarily because other frameworks have not yet implemented the necessary software components to leverage the hardware features of the ESP32-C3 for flash encryption. Additionally, an increased level of protection against attacks targeting the EEPROM can be achieved by enabling secure boot. This security measure requires the second-stage bootloader and application to be signed by a public key securely stored in eFuse memory.

V. Challenges and future directions

During our research, we investigated various technologies that can be used to facilitate identity and authentication operations in IoT systems. Our examination revealed the existence of functional prototypes for identity and authentication systems, and some of these solutions have even been implemented in real-world production environments. However, despite the availability of these technologies, we observed a lack of widespread adoption in IoT devices.

In the past, the cost of incorporating specialized hardware has often been cited as a hindrance to adopting identity and authentication solutions in IoT devices. However, as pointed out by other researchers [19], this notion is a misconception. Our analysis aligns with this perspective, demonstrating that hardware solutions are available across various price ranges, including options suitable for smaller budgets.

The lack of adoption of hardware-based solutions in IoT devices can be attributed, in part, to a disregard for security considerations. However, it is worth noting that some devices implement robust software-level protection measures while choosing not to incorporate hardware trust anchors. In such cases, the threat model for these devices likely accepts that the risk of hardware attacks or the cost associated with hardware solutions outweighs the value of the information being protected.

Regardless of the threat model in place for a specific device, the challenges listed in Section II-B affect all devices. As mentioned throughout this work, technologies can be employed to overcome these challenges. Consequently, it is essential to explore alternative factors that contribute to the limited adoption of these technologies, as hardware cost alone does not account for the prevailing situation. Based on our analysis, we identify two challenges that may be hindering the widespread use of hardware-based solutions, even when the required hardware is affordable or already included in the device but remains unused: the lack of SDK support and standardization.

A. Lack of software development kit support

Each technology in the IoT landscape typically comes with its own library or software stack that provides low-level APIs for interacting with it. These APIs allow developers to leverage the capabilities of the technology to build their solutions. This involves working at a lower level, interacting directly with the technology's APIs to implement the desired identity functionality. Therefore, increases the cost of development due to the time spent developing these features and the required specialized labor.

Moreover, designers often do not develop device software from scratch but instead use an existing SoC as a foundation for adding their features. This fact decreases the production cost and enables quicker development cycles. However, if the SDKs do not provide support for a particular technology or feature, it can indeed discourage software designers from incorporating that technology into their devices for the same reasons we mentioned earlier. Therefore, SDKs are the key to the widespread adoption of these technologies since they can facilitate the integration of security technologies and features.

One way SDKs could decrease the integration complexity of these technologies is by providing Hardware Abstraction Layer (HAL)s. A HAL is a set of APIs that abstract the complexity of the hardware underneath it [268]. These APIs are shared by different components that have the same function. Thus, developers trying to integrate these components into their system only need to be familiar with a set of APIs independent of the component manufacturer.

Another direction that SDKs can take is to provide authentication and identity frameworks that incorporate these technologies and abstract the complexity of the underlying hardware. These frameworks can leverage HALs to support various technologies. However, this should remain transparent to the user, and the integration of these frameworks and technologies must be seamless. Without this seamless integration, designers may prefer traditional systems over implementing their custom solutions with hardware security, which demands experienced personnel and is error-prone.

During our experimentation phase, we encountered a lack of support and difficulty in using hardware technologies. Among the two frameworks developed by Espressif, only one fully supports all available security features. However, even in this framework, it only offers low-level access. This means that a developer wishing to harness these technologies must possess knowledge of them and understand how to use them.

The Zephyr RTOS serves as an example of an SDK that already incorporates some of these suggestions. This SDK

and its RTOS are built on top of a HAL for most operations, simplifying the developer's experience with just one API to learn. However, this approach is not extended to all components. For instance, while there is an API to streamline the use of TRNGs and cryptographic operations, other technologies, like eFuses lack abstraction and necessitate hardware-specific APIs. Therefore, there is still room for improvement.

B. Lack of standardization

One way to encourage hardware-based IoT solutions is to integrate these technologies into a common SDK. However, when it comes to high-level features, such as authentication and identity frameworks, the lack of standardization in IoT devices hampers their inclusion in SDKs. There are no standards for these features, and each vendor tends to implement its own framework, resulting in increased development diversity and effort or not being included at all, with its complete development delegated to the system designer.

A consequence of the lack of standardization in some of the technologies presented is the shortage of products with a PUF capable of CRP authentication. Strong PUFs promise a way to uniquely identify a device with an identity tied to its hardware without requiring energy- or computation-expensive cryptographic operations during its authentication. This approach is a novelty compared to the other solutions presented in this work that use classical identity schemes.

Despite its potential, when we searched for PUFs in the market, we only found off-the-shelf components that incorporate PUFs for key generation or storage. This limitation has already been observed in prior research [269] and remains unchanged. Therefore, while PUF technology holds great promise for the IoT field, current products only tap into a fraction of their potential by not utilizing CRP schemes for authentication. This absence presents a significant challenge in fully harnessing this technology. Furthermore, identity and authentication frameworks based on CRP are not standardized, making them unsuitable for environments that require compliance with security standards such as FIPS or NIST.

One initiative that tries to oppose this fact is the Trusted Firmware initiative by Linaro [270], which provides a reference implementation of a secure processing environment for microcontrollers. Despite being aimed at ARM processors, some of its components are architecture-independent, which means they can be applied to any microcontroller. Moreover, Linaro states that it is committed to using this project to help create conditions for secure processing across IoT, independently of their architectures [271]. An example of this commitment is the MCUboot [272], a secure bootloader that is independent of the OS and hardware and is already being used by multiple OSs. This project offers a standardized way to update the device's firmware while ensuring its integrity and authenticity during boot.

On the other hand, when standardization is applied to hardware components, it helps move the burden of integrating hardware security technologies from SDK developers to manufacturers. Standardizing component APIs is also advantageous for system integrators, SDK developers, and manufacturers. System integrators can easily change suppliers since, with a common API, components may be easily swapped with others with the same specifications. SDK developers only need to handle a single API for multiple components, and technology manufacturers can decrease their development costs and make their products more competitive. This approach is already being employed by some technologies, such as TEEs and SEs standards promoted by the GlobalPlatform [140], [156]. Unfortunately, these initiatives target a subset of the components presented earlier.

In general, standardization initiatives encourage the widespread adoption of high-level security features and security hardware components. The successful integration of these mechanisms into the IoT context heavily depends on their incorporation into widely used SDKs. Standardizing these mechanisms would streamline their inclusion in a common SDK, thereby boosting their adoption. Additionally, international bodies should examine emerging authentication methods, standardize them, and incorporate them into existing certifications, such as those that leverage Strong PUFs.

In summary, the cost of hardware security in IoT devices is not solely attributed to the additional hardware itself but rather to the integration process with the rest of the device. To address this prevailing trend, SDKs must encompass hardware-based authentication and identity frameworks, simplifying the incorporation of hardware RoT solutions into new systems. Furthermore, the standardization of these frameworks is essential to avoid fragmentation among manufacturers' frameworks and minimize the learning curve associated with deploying these technologies.

VI. Conclusion

IoT devices interact with our personal lives and manage critical infrastructures. Thus, keeping them secure is a priority. Identity and authentication play a vital role in the security of these devices. Without them, it is impossible to guarantee the device's security, as we would be unable to assure the veracity of any information. Nevertheless, identity and authentication are considered open research challenges in IoT.

Resource-constrained devices, a lack of standardization, and exposure to hardware attacks are only some of the reasons that make identity and authentication in IoT so challenging. Over the years, multiple researchers have advocated using hardware to address these challenges. However, the widespread adoption of hardware technologies supporting identity and authentication has yet to be seen.

During our work, we focused on hardware trust anchors and their security features that can be exploited to develop new identity and authentication systems.

We analyzed physical risks for IoT identity and identified possible countermeasures. Our findings revealed that hardware trust anchors must employ protections like multiple sensors, active metal shields, and a defensive PCB design to safeguard themselves against physical risks. Additionally, we explored how challenging these risks are, as we cannot completely mitigate them but can increase the difficulty of an attack.

Considering these security features and challenges for IoT identity in mind, we reviewed technologies available to designers to develop new identity and authentication systems. In this analysis, we included the following technologies: TRNGs, masked ROMs and OTP memories, crypto accelerators, SE, TEEs, and PUFs.

In addition to the theoretical analysis and exploration of technologies, this research paper presents an experimental evaluation demonstrating hardware trust anchors' effectiveness in mitigating attacks on IoT identity. The empirical evidence from these experiments reinforces the significance of hardware trust anchors in enhancing security and developing effective identity solutions for IoT devices. Moreover, the practical implementation showcased through the case study of a smart meter connected to a utility company via a mesh network exemplifies the successful integration of multiple technologies and demonstrates the value of hardware trust anchors in the real-world.

To sum up, there are multiple candidate technologies that might support new identity and authentication systems, aiming at different price points. These technologies can overcome some of the challenges holding back identity and authentication in IoT by using common cryptographic algorithms in low-power devices and offering resilience against hardware attacks. Unfortunately, the complex integration process of some of these technologies, the required knowledge to effectively use them, and the lack of standardization continue to hinder the widespread use of hardware trust anchors in IoT.

REFERENCES

[1] A. Nordrum *et al.*, "Popular internet of things forecast of 50 billion devices by 2020 is outdated," *IEEE spectrum*, vol. 18, no. 3, 2016.

[2] D. Hanes, G. Salgueiro, P. Grossetete, R. Barton, and J. Henry, *IoT fundamentals: Networking technologies, protocols, and use cases for the internet of things*. Cisco Press, 2017.

[3] D. Wang, D. Chen, B. Song, N. Guizani, X. Yu, and X. Du, "From iot to 5g i-iot: The next generation iot-based intelligent algorithms and 5g technologies," *IEEE Communications Magazine*, vol. 56, no. 10, pp. 114–120, 2018.

[4] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa, and I. Yaqoob, "Big iot data analytics: Architecture, opportunities, and open research challenges," *IEEE Access*, vol. 5, pp. 5247–5261, 2017.

[5] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. Goren, and C. Mahmoudi, "Fog computing conceptual model," Tech. Rep., mar 2018.

[6] N. Yousefnezhad, A. Malhi, and K. Främling, "Security in product lifecycle of IoT devices: A survey," *Journal of Network and Computer Applications*, vol. 171, p. 102779, dec 2020.

[7] A. R. H. Hussein, "Internet of things (iot): Research challenges and future applications," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 6, pp. 77–82, 2019.

[8] H. U. Rehman, M. Asif, and M. Ahmad, "Future applications and research challenges of iot," in *2017 International conference on information and communication technologies (ICICT)*. IEEE, 2017, pp. 68–74.

[9] S. A. Al-Qaseemi, H. A. Almulhim, M. F. Almulhim, and S. R. Chaudhry, "Iot architecture challenges and issues: Lack of standardization," in *2016 Future Technologies Conference (FTC)*. IEEE, 2016, pp. 731–738.

[10] M. Nawir, A. Amir, N. Yaakob, and O. B. Lynn, "Internet of things (iot): Taxonomy of security attacks," in *2016 3rd International Conference on Electronic Design (ICED)*. IEEE, 2016, pp. 321–326.

[11] A. Cirne, P. R. Sousa, J. S. Resende, and L. Antunes, "Iot security certifications: Challenges and potential approaches," *Computers & Security*, vol. 116, p. 102669, 2022.

[12] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh, "Iot security: ongoing challenges and research opportunities," in *2014 IEEE 7th international conference on service-oriented computing and applications*. IEEE, 2014, pp. 230–234.

[13] X. Zhu and Y. Badr, "A survey on blockchain-based identity management systems for the internet of things," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, jul 2018.

[14] K. Chen, S. Zhang, Z. Li, Y. Zhang, Q. Deng, S. Ray, and Y. Jin, "Internet-of-things security and vulnerabilities: Taxonomy, challenges, and practice," *Journal of Hardware and Systems Security*, vol. 2, no. 2, pp. 97–110, 2018.

[15] G. Loukas, *Cyber-physical attacks: A growing invisible threat*. Butterworth-Heinemann, 2015.

[16] S. Sidhu, B. J. Mohd, and T. Hayajneh, "Hardware security in iot devices with emphasis on hardware trojans," *Journal of Sensor and Actuator Networks*, vol. 8, no. 3, p. 42, 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8761062

[17] C. O. Jasper van Woudenberg, *The Hardware Hacking Handbook*. Random House LCC US, Dec. 2021. [Online]. Available: https://www.ebook.de/de/product/31189064/jasper_van_woudenberg_colin_o_flynn_the_hardware_hacking_handbook.html

[18] M. Roel, "Physically unclonable functions: Constructions, properties and applications," *Katholieke Universiteit Leuven, Belgium*, 2012.

[19] B. Pearson, L. Luo, Y. Zhang, R. Dey, Z. Ling, M. Bassiouni, and X. Fu, "On misconception of hardware and cost in iot security and privacy," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–7.

[20] I. Butun, A. Sari, and P. Österberg, "Hardware security of fog end-devices for the internet of things," *Sensors*, vol. 20, no. 20, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/20/5729

[21] A. Ehret, K. Gettings, B. R. Jordan, and M. A. Kinsy, "A survey on hardware security techniques targeting low-power soc designs," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019, pp. 1–8.

[22] W. Hu, C.-H. Chang, A. Sengupta, S. Bhunia, R. Kastner, and H. Li, "An overview of hardware security and trust: Threats, countermeasures, and design tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 6, pp. 1010–1038, 2021.

[23] E. T. Michailidis, D. G. Kogias, and I. Voyiatzis, "A review on hardware security countermeasures for iot: Emerging mechanisms and machine learning solutions," in *Proceedings of the 24th Pan-Hellenic Conference on Informatics*, ser. PCI '20. New York, NY, USA: Association for Computing Machinery, 2021, p. 268–271. [Online]. Available: https://doi.org/10.1145/3437120.3437322

[24] I. Tudosa, F. Picariello, E. Balestrieri, L. De Vito, and F. Lamonaca, "Hardware security in iot era: the role of measurements and instrumentation," in *2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4.0&IoT)*, 2019, pp. 285–290.

[25] S. Akter, K. Khalil, and M. Bayoumi, "A survey on hardware security: Current trends and challenges," *IEEE Access*, pp. 1–1, 2023.

[26] K. Yang, D. Blaauw, and D. Sylvester, "Hardware designs for security in ultra-low-power IoT systems: An overview and survey," *IEEE Micro*, vol. 37, no. 6, pp. 72–89, nov 2017.

[27] S. Cheruvu, A. Kumar, N. Smith, and D. M. Wheeler, *Demystifying internet of things security: successful iot device/edge and platform security deployment*. Springer Nature, 2020.

[28] C. Shepherd, G. Arfaoui, I. Gurulian, R. P. Lee, K. Markantonakis, R. N. Akram, D. Sauveron, and E. Conchon, "Secure and trusted execution: Past, present, and future - a critical review in the context of the internet of things and cyber-physical systems," in *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, aug 2016, pp. 168–177.

[29] Y. Kabalci, "A survey on smart metering and smart grid communication," *Renewable and Sustainable Energy Reviews*, vol. 57, pp. 302–318, 2016.

[30] P. Mlynek, J. Misurec, Z. Kolka, J. Slacik, and R. Fujdiak, "Narrowband power line communication for smart metering and street lighting control," *IFAC-PapersOnLine*, vol. 48, no. 4, pp. 215–219, 2015.

[31] ITU-T, "Y.2720 : Ngn identity management framework," International Telecommunication Union, Tech. Rep., 2009. [Online]. Available: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-Y.2720-200901-I!!PDF-E&type=items

[32] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, dec 2018.

[33] R. Maes, *PUF-Based Entity Identification and Authentication*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 117–141. [Online]. Available: https://doi.org/10.1007/978-3-642-41395-7_5

[34] P. Angin, B. Bhargava, R. Ranchal, N. Singh, M. Linderman, L. B. Othmane, and L. Lilien, "An entity-centric approach for privacy and identity management in cloud computing," in *2010 29th IEEE Symposium on Reliable Distributed Systems*. IEEE, oct 2010.

[35] Y. Cao and L. Yang, "A survey of identity management technology," in *2010 IEEE International Conference on Information Theory and Information Security*. IEEE, dec 2010.

[36] M. Gaedke, J. Meinecke, and M. Nussbaumer, "A modeling approach to federated identity and access management," in *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*, ser. WWW '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 1156–1157. [Online]. Available: https://doi.org/10.1145/1062745.1062916

[37] D. W. Chadwick, *Federated Identity Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 96–120. [Online]. Available: https://doi.org/10.1007/978-3-642-03829-7_3

[38] S. Cantor, J. Moreh, R. Philpott, and E. Maler, "Metadata for the oasis security assertion markup language (saml) v2. 0," 2005.

[39] N. Sakimura, J. Bradley, M. Jones, B. De Medeiros, and C. Mortimore, "Openid connect core 1.0," *The OpenID Foundation*, p. S3, 2014.

[40] D. Divyabharathi and N. G. Cholli, "A review on identity and access management server (keycloak)," *International Journal of Security and Privacy in Pervasive Computing (IJSPPC)*, vol. 12, no. 3, pp. 46–53, 2020.

[41] S. Cantor and T. Scavo, "Shibboleth architecture," *Protocols and Profiles*, vol. 10, p. 16, 2005.

[42] P. R. Sousa, J. S. Resende, R. Martins, and L. Antunes, "The case for blockchain in IoT identity management," *Journal of Enterprise Information Management*, vol. ahead-of-print, no. ahead-of-print, jun 2020.

[43] D. Hardt, "The OAuth 2.0 Authorization Framework," RFC 6749, Oct. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6749

[44] A. Jøsang and S. Pope, "User centric identity management," in *AusCERT Asia Pacific information technology security conference*. Citeseer, 2005, p. 77. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.1563&rep=rep1&type=pdf

[45] J. Werner, C. M. Westphall, and C. B. Westphall, "Cloud identity management: A survey on privacy strategies," *Computer Networks*, vol. 122, pp. 29–42, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128617301664

[46] S. Y. Lim, P. T. Fotsing, A. Almasri, O. Musa, M. L. M. Kiah, T. F. Ang, and R. Ismail, "Blockchain technology the identity management and authentication service disruptor: a survey," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 8, no. 4-2, pp. 1735–1745, 2018.

[47] A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel, "A survey on essential components of a self-sovereign identity," *Computer Science Review*, vol. 30, pp. 80–86, 2018.

[48] Q. Feng, D. He, S. Zeadally, M. K. Khan, and N. Kumar, "A survey on privacy protection in blockchain system," *Journal of Network and Computer Applications*, vol. 126, pp. 45–58, 2019.

[49] P. Mahalle, S. Babar, N. R. Prasad, and R. Prasad, "Identity management framework towards internet of things (iot): Roadmap and key challenges," in *International Conference on Network Security and Applications*. Springer, 2010, pp. 430–439. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-14478-3_43

[50] K.-Y. Lam and C.-H. Chi, "Identity in the internet-of-things (iot): New challenges and opportunities," in *International Conference on Information and Communications Security*. Springer International Publishing, 2016, pp. 18–26. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-3-319-50011-9_2.pdf

[51] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[52] T. Nandy, M. Y. I. B. Idris, R. Md Noor, L. Mat Kiah, L. S. Lun, N. B. Annuar Juma'at, I. Ahmedy, N. Abdul Ghani, and S. Bhattacharyya, "Review on security of internet of things authentication mechanism," *IEEE Access*, vol. 7, pp. 151 054–151 089, 2019.

[53] M. El-hajj, A. Fadlallah, M. Chamoun, and A. Serhrouchni, "A survey of internet of things (iot) authentication schemes," *Sensors*, vol. 19, no. 5, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/5/1141

[54] M.-O. Pahl and L. Donini, "Giving iot services an identity and changeable attributes," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019, pp. 455–461.

[55] R. Román-Castro, J. López, and S. Gritzalis, "Evolution and trends in iot security," *Computer*, vol. 51, no. 7, pp. 16–25, 2018.

[56] K. Zhao and L. Ge, "A survey on the internet of things security," in *2013 Ninth International Conference on Computational Intelligence and Security*, 2013, pp. 663–667.

[57] R. Roman, P. Najera, and J. Lopez, "Securing the internet of things," *Computer*, vol. 44, no. 9, pp. 51–58, 2011.

[58] H. A. Abdulghani, N. A. Nijdam, A. Collen, and D. Konstantas, "A study on security and privacy guidelines, countermeasures, threats: Iot data at rest perspective," *Symmetry*, vol. 11, no. 6, p. 774, 2019.

[59] M. Katagi, S. Moriai *et al.*, "Lightweight cryptography for the internet of things," *Sony Corporation*, vol. 2008, pp. 7–10, 2008. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.227.8445&rep=rep1&type=pdf

[60] Z.-K. Zhang, M. C. Y. Cho, Z.-Y. Wu, and S. W. Shieh, "Identifying and authenticating iot objects in a natural context," *Computer*, vol. 48, no. 08, pp. 81–83, 2015.

[61] A. R. Metke and R. L. Ekl, "Security technology for smart grid networks," *IEEE Transactions on Smart Grid*, vol. 1, no. 1, pp. 99–107, June 2010.

[62] J. Xia and Y. Wang, "Secure key distribution for the smart grid," *IEEE Transactions on Smart Grid*, vol. 3, no. 3, pp. 1437–1443, Sep. 2012.

[63] V. Pillitteri and T. Brewer, "Guidelines for smart grid cybersecurity," Tech. Rep., 2014-09-25 2014.

[64] R. Dupont and A. Enge, "Provably secure non-interactive key distribution based on pairings," *Discrete Applied Mathematics*, vol. 154, no. 2, pp. 270–276, 2006, coding and Cryptography. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166218X05002337

[65] V. Seferian, R. Kanj, A. Chehab, and A. Kayssi, "Identity based key distribution framework for link layer security of ami networks," *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 3166–3179, July 2018.

[66] A. Shostack, *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[67] ISO/IEC JTC 1/SC 27, "Iso/iec 15408 information security, cybersecurity and privacy protection — evaluation criteria for it security," 2022, iSO 15408:2022. [Online]. Available: https://www.iso.org/standard/72891.html

[68] K. Beckers, D. Hatebur, and M. Heisel, "A problem-based threat analysis in compliance with common criteria," in *2013 International Conference on Availability, Reliability and Security*, Sep. 2013, pp. 111–120.

[69] W. H. Wolf, *Computers as components*. Elsevier/Morgan Kaufmann, 2008.

[70] Recessim Wiki, "Landis+gyr residential meter," 2020, Accessed: Apr. 16, 2023. [Online]. Available: https://wiki.recessim.com/view/Landis\%2BGyr_Residential_Meter

[71] European Committee for Electrotechnical Standardization, "EN 13757-2: Communication systems for meters - part 2: Physical and link layer," 2018, accessed: Apr. 16, 2023. [Online]. Available: https://standards.cen.eu/dyn/www/f?p=204:110:0::::FSP_PROJECT,FSP_ORG_ID:62141,6135&cs=185A0912EBD1A2C1B62A03A7E80332078

[72] C. Gu, *Power On and Bootloader*. Berkeley, CA: Apress, 2016, pp. 5–25. [Online]. Available: https://doi.org/10.1007/978-1-4842-1919-5_2

[73] *Security requirements for cryptographic modules*, National Institute of Standards and Technology Std., may 2001.

[74] K. Markantonakis *et al.*, "Enhancing the conditional access module security in light of smart card sharing attacks," *Presentation, Information Security Group Smart Card Centre, Royal Holloway, University of London*, vol. 20, 2008.

[75] C. Johnson, M. Badger, D. Waltermire, J. Snyder, and C. Skorupka, "Guide to cyber threat information sharing," Oct. 2016.

[76] C. L. Smith and D. J. Brooks, "Chapter 3 - security risk management," in *Security Science*, C. L. Smith and D. J. Brooks, Eds. Boston: Butterworth-Heinemann, 2013, pp. 51–80. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123944368000035

[77] T. A. Johnson, *Cybersecurity: Protecting critical infrastructures from cyber attack and cyber warfare*. CRC Press, 2015.

[78] S. P. Skorobogatov, "Semi-invasive attacks: a new approach to hardware security analysis," University of Cambridge, Computer Laboratory, Tech. Rep., 2005.

[79] M. T. Rahman, Q. Shi, S. Tajik, H. Shen, D. L. Woodard, M. Tehranipoor, and N. Asadizanjani, "Physical inspection & attacks: New fron-

tier in hardware security," in *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*. IEEE, jul 2018, pp. 93–102.

[80] M. G. Rekoff, "On reverse engineering," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-15, no. 2, pp. 244–252, 1985.

[81] R. Torrance and D. James, "The state-of-the-art in ic reverse engineering," in *Cryptographic Hardware and Embedded Systems - CHES 2009*, C. Clavier and K. Gaj, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 363–381.

[82] R. C. Gilberg, R. M. Knowles, P. Moroney, and W. A. Shumate, "Secure integrated circuit chip with conductive shield," Jun. 12 1990, uS Patent 4,933,898.

[83] S. H. Weingart, "Physical security devices for computer subsystems: A survey of attacks and defenses," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2000, pp. 302–317.

[84] S. Manich, M. S. Wamser, and G. Sigl, "Detection of probing attempts in secure ics," in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*. IEEE, 2012, pp. 134–139.

[85] M. Nagata, "Exploring fault injection attack resilience of secure ic chips," in *2022 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 2022, pp. 11C–1.

[86] S. Skorobogatov, "How microprobing can attack encrypted memory," in *2017 Euromicro Conference on Digital System Design (DSD)*. IEEE, 2017, pp. 244–251.

[87] A. Mohammadi, M. Ebrahimi, A. Ejlali, and S. G. Miremadi, "Scfit: A fpga-based fault injection technique for seu fault model," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2012, pp. 586–589.

[88] O. Kömmerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors." *Smartcard*, vol. 99, pp. 9–20, 1999.

[89] C. O'Flynn, "Getting root on philips hue bridge 2.0," 2016.

[90] N. Timmers, A. Spruyt, and M. Witteman, "Controlling pc on arm using fault injection," in *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2016, pp. 25–35.

[91] M. Witteman and M. Oostdijk, "Secure application programming in the presence of side channel attacks," in *RSA conference*, 2008.

[92] S. Endo, Y. Li, N. Homma, K. Sakiyama, K. Ohta, and T. Aoki, "An efficient countermeasure against fault sensitivity analysis using configurable delay blocks," in *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2012, pp. 95–102.

[93] M. Nagata, T. Miki, and N. Miura, "Physical attack protection techniques for ic chip level hardware security," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 1, pp. 5–14, 2021.

[94] L. Zussa, A. Dehbaoui, K. Tobich, J.-M. Dutertre, P. Maurine, L. Guillaume-Sage, J. Clediere, and A. Tria, "Efficiency of a glitch detector against electromagnetic fault injection," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2014, pp. 1–6, iSSN: 1558-1101.

[95] N. Miura, D. Fujimoto, D. Tanaka, Y.-i. Hayashi, N. Homma, T. Aoki, and M. Nagata, "A local EM-analysis attack resistant cryptographic engine with fully-digital oscillator-based tamper-access sensor," in *2014 Symposium on VLSI Circuits Digest of Technical Papers*, Jun. 2014, pp. 1–2, iSSN: 2158-5636.

[96] Y. Araga, M. Nagata, H. Ikeda, T. Miki, N. Miura, N. Watanabe, H. Shimamoto, and K. Kikuchi, "A Thick Cu Layer Buried in Si Interposer Backside for Global Power Routing," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 9, no. 3, pp. 502–510, Mar. 2019.

[97] S. Bhunia and M. Tehranipoor, "Chapter 8 - side-channel attacks," in *Hardware Security*, S. Bhunia and M. Tehranipoor, Eds. Morgan Kaufmann, 2019, pp. 193–218. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128124772000137

[98] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, 2005.

[99] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems, "A practical implementation of the timing attack," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 1998, pp. 167–182.

[100] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual international cryptology conference*. Springer, 1999, pp. 388–397. [Online]. Available: https://link.springer.com/content/pdf/10.1007/3-540-48405-1_25.pdf

[101] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn, "Iot goes nuclear: Creating a zigbee chain reaction," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 195–212.

[102] J. Krämer, D. Nedospasov, A. Schlösser, and J.-P. Seifert, "Differential photonic emission analysis," in *Constructive Side-Channel Analysis and Secure Design*, E. Prouff, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–16.

[103] A. Schlösser, D. Nedospasov, J. Krämer, S. Orlic, and J.-P. Seifert, "Simple photonic emission analysis of aes," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2012, pp. 41–57.

[104] J. Krämer, "Why cryptography should not rely on physical attack complexity," *it-Information Technology*, vol. 59, no. 1, pp. 53–56, 2017.

[105] V. Rozic, B. Yang, W. Dehaene, and I. Verbauwhede, "Highly efficient entropy extraction for true random number generators on fpgas," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–6.

[106] C. S. Petrie and J. A. Connelly, "A noise-based ic random number generator for applications in cryptography," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 5, pp. 615–621, 2000.

[107] J. Senden, "Biasing a ring-oscillator based true random number generator with an electro-magnetic fault injuction using harmonic waves," Master's thesis, University of Twente, 2015.

[108] P. Bayon, L. Bossuet, A. Aubert, and V. Fischer, "Electromagnetic analysis on ring oscillator-based true random number generators," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2013, pp. 1954–1957.

[109] Y. Su, J. Wu, C. Long, and L. Wei, "Secure decentralized machine identifiers for internet of things," in *Proceedings of the 2020 The 2nd International Conference on Blockchain Technology*, 2020, pp. 57–62.

[110] M. Barr, "Memory types," *Embedded Systems Programming*, vol. 14, no. 5, pp. 103–104, 2001.

[111] U. Gatti, "One-time programmable memories for harsh environments," *Rad-hard Semiconductor Memories*, p. 151, 2019.

[112] C. M. Maxfield, "Chapter 15 - memory ics," in *Bebop to the Boolean Boogie (Third Edition)*, third edition ed., C. M. Maxfield, Ed. Boston: Newnes, 2009, pp. 193–212. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781856175074000152

[113] D. Kahng and S. M. Sze, "A floating gate and its application to memory devices," *The Bell System Technical Journal*, vol. 46, no. 6, pp. 1288–1295, jul 1967.

[114] C. M. Maxfield, "Chapter 16 - programmable ics," in *Bebop to the Boolean Boogie (Third Edition)*, third edition ed., C. M. Maxfield, Ed. Boston: Newnes, 2009, pp. 213–234. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781856175074000164

[115] R. F. Rizzolo, T. G. Foote, J. M. Crafts, D. A. Grosch, T. O. Leung, D. J. Lund, B. L. Mechtly, B. J. Robbins, T. J. Slegel, M. J. Tremblay *et al.*, "Ibm system z9 efuse applications and methodology," *IBM Journal of Research and Development*, vol. 51, no. 1.2, pp. 65–75, 2007.

[116] H. Divva, A. P. Chavan, and S. Krishnamurthy, "Design and verification of ecc scheme to optimize area and tester time in otp rom controller," in *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, 2019, pp. 151–155.

[117] J.-M. Schmidt, M. Hutter, and T. Plos, "Optical fault attacks on aes: A threat in violet," in *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2009, pp. 13–22.

[118] ——, "Optical fault attacks on aes: A threat in violet," in *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2009, pp. 13–22.

[119] M. Hutle and M. Kammerstetter, "Chapter 4 - Resilience Against Physical Attacks," in *Smart Grid Security*, F. Skopik and P. Smith, Eds. Boston: Syngress, Jan. 2015, pp. 79–112. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128021224000043

[120] S. Skorobogatov, "Physical attacks and tamper resistance," in *Introduction to Hardware Security and Trust*. Springer, 2012, pp. 143–173.

[121] M. Tunstall, *Smart Card Security*. Cham: Springer International Publishing, 2017, pp. 217–251. [Online]. Available: https://doi.org/10.1007/978-3-319-50500-8_9

[122] J. Jung, J. Cho, and B. Lee, "A secure platform for iot devices based on arm platform security architecture," in *2020 14th International Conference on Ubiquitous Information Management and Communication (IMCOM)*, 2020, pp. 1–4.

[123] L. Bossuet, M. Grand, L. Gaspar, V. Fischer, and G. Gogniat, "Architectures of flexible symmetric key crypto engines—a survey: From hardware coprocessor to multi-crypto-processor system on chip," *ACM Computing Surveys (CSUR)*, vol. 45, no. 4, pp. 1–32, 2013.

[124] S. Gueron, "Intel advanced encryption standard (aes) instructions set," *Intel White Paper, Rev*, vol. 3, pp. 1–94, 2010.

[125] I. ARM, "Armv8-a architecture reference manual," *URL: https://documentation-service. arm. com/static/60e6f8573d73a34b640e0cee*, 2015.

[126] L. Gaspar, V. Fischer, F. Bernard, L. Bossuet, and P. Cotret, "Hcrypt: a novel concept of crypto-processor with secured key management," in *2010 International Conference on Reconfigurable Computing and FPGAs*. IEEE, 2010, pp. 280–285.

[127] S. A. Rotondo, *Trusted Computing Group*. Boston, MA: Springer US, 2011, pp. 1331–1331. [Online]. Available: https://doi.org/10.1007/978-1-4419-5906-5_498

[128] S. L. Kinney, *Trusted Platform Module Basics: Using TPM in Embedded Systems*. USA: Newnes, 2006.

[129] T. C. Group, "Trusted platform module library part 1: Architecture," Trusted Computing Group, Tech. Rep. 01.59, Nov. 2019. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf

[130] *TPM 2.0 Mobile Common Profile*, Trusted Computing Group Technical Report 31, Dec. 2015. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TPM_2.0_Mobile_Common_Profile_v2r31_FINAL.pdf

[131] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against intel sgx," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1466–1482.

[132] S. Saab, P. Rohatgi, and C. Hampel, "Side-channel protections for cryptographic instruction set extensions," *Cryptology ePrint Archive*, 2016.

[133] Y. Lu, "Attacking hardware aes with dfa," *arXiv preprint arXiv:1902.08693*, 2019.

[134] T. C. Group, "Profile pc client specific trusted platform module tpm family 2.0," Trusted Computing Group, Tech. Rep. 1.3, Sep. 2021.

[135] *FIPS 140-3 - Security requirements for cryptographic modules*, National Institute of Standards and Technology Std., apr 2019.

[136] B. Pearson, C. Zou, Y. Zhang, Z. Ling, and X. Fu, "Sic 2: Securing microcontroller based iot devices with low-cost crypto coprocessors," in *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2020, pp. 372–381.

[137] Z. Zieliski, J. Chudzikiewicz, and J. Furtak, *An Approach to Integrating Security and Fault Tolerance Mechanisms into the Military IoT*. Cham: Springer International Publishing, 2019, pp. 111–128. [Online]. Available: https://doi.org/10.1007/978-3-030-02807-7_6

[138] R. Toegl, "Tagging the turtle: Local attestation for kiosk computing," in *Advances in Information Security and Assurance*, J. H. Park, H.-H. Chen, M. Atiquzzaman, C. Lee, T.-h. Kim, and S.-S. Yeo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 60–69.

[139] N. Kuntze, A. Fuchs, and C. Rudolph, "Reliable identities using off-the-shelf hardware security in manets," in *2009 International Conference on Computational Science and Engineering*, vol. 2. IEEE, 2009, pp. 781–786.

[140] G. Inc, "Introduction to secure elements," May 2018, Accessed: Mar. 21, 2022. [Online]. Available: https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Secure-Element-15May2018.pdf

[141] A. Umar and K. Mayes, *Trusted Execution Environment and Host Card Emulation*. Cham: Springer International Publishing, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-50500-8_18

[142] NXP, "P5cx012/02x/40/73/80/144 family," Jan. 2008.

[143] B. Lepojevic, B. Pavlovic, and A. Radulovic, "Implementing nfc service security–se vs tee vs hce," in *SYMORG Conference*, 2014.

[144] K. Mayes and T. Evans, *Smart Cards and Security for Mobile Communications*. Cham: Springer International Publishing, 2017, pp. 93–128. [Online]. Available: https://doi.org/10.1007/978-3-319-50500-8_4

[145] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer Science & Business Media, 2008, vol. 31.

[146] K. E. Mayes and K. Markantonakis, *Smart cards, tokens, security and applications*. Springer, 2008, vol. 1.

[147] V. Lomne, "Common criteria certification of a smartcard: a technical overview," in *CHES*, 2016.

[148] E. B. Sanjuan, I. A. Cardiel, J. A. Cerrada, and C. Cerrada, "Message queuing telemetry transport (mqtt) security: a cryptographic smart card approach," *IEEE Access*, vol. 8, pp. 115 051–115 062, 2020.

[149] Y. Jeon and Y. Kang, "Implementation of a lorawan protocol processing module on an embedded device using secure element," in *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*, 2019, pp. 1–3.

[150] B. S. S. B.V., "Bosch ip video and data security guidebook," Bosch, techreport 2.0, Apr. 2021. [Online]. Available: https://resources-boschsecurity-cdn.azureedge.net/public/documents/Data_Security_Guideb_Special_enUS_9007221590612491.pdf

[151] C. Lesjak, T. Ruprechter, J. Haid, H. Bock, and E. Brenner, "A secure hardware module and system concept for local and remote industrial embedded system identification," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 2014, pp. 1–7.

[152] C. Lesjak, T. Ruprechter, H. Bock, J. Haid, and E. Brenner, "Estado — enabling smart services for industrial equipment through a secured, transparent and ad-hoc data transmission online," in *The 9th International Conference for Internet Technology and Secured Transactions (ICITST-2014)*, 2014, pp. 171–177.

[153] R. N. Akram, P.-F. Bonnefoi, S. Chaumette, K. Markantonakis, and D. Sauveron, "Improving security of autonomous uavs fleets by using new specific embedded secure elements-a position paper," in *2nd AETOS international conference on "Research challenges for future RPAS/UAV systems", Bordeaux, France*, 2014.

[154] I. GlobalPlatform, "Tee system architecture," GlobalPlatform Technology, techreport GPD_SPE_009, 2018. [Online]. Available: https://globalplatform.org/wp-content/uploads/2018/09/GPD_TEE_SystemArch_v1.1.0.10-for-v1.2_PublicReview.pdf

[155] A. Vasudevan, E. Owusu, Z. Zhou, J. Newsome, and J. M. McCune, "Trustworthy execution on mobile devices: What security properties can my mobile platform give me?" in *International conference on trust and trustworthy computing*. Springer, 2012, pp. 159–178. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.220.220&rep=rep1&type=pdf

[156] I. GlobalPlatform, "Trusted user interface api," GlobalPlatform, techreport GPD_SPE_020, Jun. 2013. [Online]. Available: {https://globalplatform.org/wp-content/uploads/2013/06/GlobalPlatform_Trusted_User_Interface_API_v1.0.pdf}

[157] T. Alves, "Trustzone: Integrated hardware and software security," *White paper*, 2004.

[158] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016. [Online]. Available: http://css.csail.mit.edu/6.858/2020/readings/costan-sgx.pdf

[159] A. Rao, "Rising to the challenge - data security with intel confidential computing," Intel, Feb. 2022. [Online]. Available: https://community.intel.com/t5/Blogs/Products-and-Solutions/Security/Rising-to-the-Challenge-Data-Security-with-Intel-Confidential/post/1353141

[160] M. McReynolds, "Azure announces next generation intel sgx confidential computing vms," Nov. 2021. [Online]. Available: https://techcommunity.microsoft.com/t5/azure-confidential-computing/azure-announces-next-generation-intel-sgx-confidential-computing/ba-p/2839934

[161] S. Pinto and N. Santos, "Demystifying arm trustzone: A comprehensive survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[162] H. Yang and M. Lee, "Demystifying arm trustzone tee client api using op-tee," in *The 9th International Conference on Smart Media and Applications*, 2020, pp. 325–328.

[163] T. Firmware, "Open portable trusted execution environment," 2013. [Online]. Available: https://www.op-tee.org/

[164] B. McGillion, T. Dettenborn, T. Nyman, and N. Asokan, "Open-TEE – an open virtual trusted execution environment," in *2015 IEEE Trustcom/BigDataSE/ISPA*. IEEE, aug 2015.

[165] N. Zhang, H. Sun, K. Sun, W. Lou, and Y. T. Hou, "Cachekit: Evading memory introspection using cache incoherence," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 337–352.

[166] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "{ARMageddon}: Cache attacks on mobile devices," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 549–564.

[167] R. Guanciale, H. Nemati, C. Baumann, and M. Dam, "Cache storage channels: Alias-driven attacks and verified countermeasures," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 38–55.

[168] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou, "Truspy: Cache side-channel information leakage from the secure world on arm devices," *Cryptology ePrint Archive*, 2016.

[169] A. Machiry, E. Gustafson, C. Spensky, C. Salls, N. Stephens, R. Wang, A. Bianchi, Y. R. Choe, C. Kruegel, and G. Vigna, "Boomerang: Exploiting the semantic gap in trusted execution environments." in *NDSS*, 2017.

[170] Z. István, P. Rosero, and P. Bonnet, "Always-trusted iot—making iot devices trusted with minimal overhead," in *Proceedings of the 5th Workshop on System Software for Trusted Execution*, ser. SysTEX '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2.

[171] Intel, "linux-sgx," Github, 2015, Accessed: Apr. 21, 2022. [Online]. Available: https://github.com/intel/linux-sgx

[172] A. Nilsson, P. N. Bideh, and J. Brorsson, "A survey of published attacks on intel sgx," *arXiv preprint arXiv:2006.13598*, 2020.

[173] A. Brandão, J. S. Resende, and R. Martins, "Hardening cryptographic operations through the use of secure enclaves," *Computers & Security*, vol. 108, p. 102327, 2021.

[174] M. Schwarz, S. Weiser, and D. Gruss, "Practical enclave malware with intel sgx," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 16th International Conference, DIMVA 2019, Gothenburg, Sweden, June 19–20, 2019, Proceedings 16*. Springer, 2019, pp. 177–196.

[175] V. Shanbhogue, J. W. Brandt, and J. Wiedemeier, "Protecting information processing system secrets from debug attacks," Feb. 10 2015, uS Patent 8,955,144.

[176] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 557–574.

[177] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[178] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[179] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 142–157.

[180] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient {Out-of-Order} execution," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 991–1008.

[181] C. Canella, D. Genkin, L. Giner, D. Gruss, M. Lipp, M. Minkin, D. Moghimi, F. Piessens, M. Schwarz, B. Sunar, J. Van Bulck, and Y. Yarom, "Fallout: Leaking data on meltdown-resistant cpus," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 769–784. [Online]. Available: https://doi.org/10.1145/3319535.3363219

[182] S. Van Schaik, A. Milburn, S. Österlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida, "Ridl: Rogue in-flight data load," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 88–105.

[183] M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher, and D. Gruss, "Zombieload: Cross-privilege-boundary data sampling," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 753–768.

[184] H. Vill, "Sgx attestation process," 2017. [Online]. Available: https://courses.cs.ut.ee/MTAT.07.022/2017_spring/uploads/Main/hiie-report-s16--17.pdf

[185] J. Van Bulck, D. Oswald, E. Marin, A. Aldoseri, F. D. Garcia, and F. Piessens, "A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1741–1758.

[186] NIST, "National vulnerability database," 2022, Accessed: Apr. 29, 2022. [Online]. Available: https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=TrustZone&search_type=all&isCpeNameSearch=false

[187] D. Cerdeira, N. Santos, P. Fonseca, and S. Pinto, "SoK: Understanding the prevailing security vulnerabilities in TrustZone-assisted TEE systems," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, may 2020.

[188] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure:{SGX} cache attacks are practical," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[189] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "Cachezoom: How sgx amplifies the power of cache attacks," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 69–90.

[190] S. K. Bukasa, R. Lashermes, H. L. Bouder, J.-L. Lanet, and A. Legay, "How trustzone could be bypassed: Side-channel attacks on a modern system-on-chip," in *IFIP International Conference on Information Security Theory and Practice*. Springer, 2017, pp. 93–109.

[191] Z. Chen, G. Vasilakis, K. Murdock, E. Dean, D. Oswald, and F. D. Garcia, "{VoltPillager}: Hardware-based fault injection attacks against intel {SGX} enclaves using the {SVID} voltage scaling interface," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 699–716.

[192] S. Gueron, "A memory encryption engine suitable for general purpose processors," *Cryptology ePrint Archive*, 2016.

[193] C. Lesjak, D. Hein, and J. Winter, "Hardware-security technologies for industrial iot: Trustzone and security controller," in *IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2015, pp. 002 589–002 595.

[194] Z. Ling, H. Yan, X. Shao, J. Luo, Y. Xu, B. Pearson, and X. Fu, "Secure boot, trusted boot and remote attestation for arm trustzone-based iot nodes," *Journal of Systems Architecture*, vol. 119, p. 102240, 2021.

[195] J. Wang, Z. Hong, Y. Zhang, and Y. Jin, "Enabling security-enhanced attestation with intel sgx for remote terminal and iot," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 88–96, 2018.

[196] A. Durand, P. Gremaud, J. Pasquier, and U. Gerber, "Trusted lightweight communication for iot systems using hardware security," in *Proceedings of the 9th International Conference on the Internet of Things*, 2019, pp. 1–4.

[197] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[198] P. Tuyls, B. Škorić, S. Stallinga, A. H. Akkermans, and W. Ophey, "Information-theoretic security analysis of physical uncloneable functions," in *International Conference on Financial Cryptography and Data Security*. Springer, 2005, pp. 141–155.

[199] B. Škorić, P. Tuyls, and W. Ophey, "Robust key extraction from physical uncloneable functions," in *International Conference on Applied Cryptography and Network Security*. Springer, 2005, pp. 407–422.

[200] G. A. Fink, D. V. Zarzhitsky, T. E. Carroll, and E. D. Farquhar, "Security and privacy grand challenges for the internet of things," in *2015 International Conference on Collaboration Technologies and Systems (CTS)*. IEEE, 2015, pp. 27–34.

[201] Y. Atwady and M. Hammoudeh, "A survey on authentication techniques for the internet of things," in *Proceedings of the International Conference on Future Networks and Distributed Systems*, ser. ICFNDS '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3102304.3102312

[202] M. Mamdouh, A. I. Awad, A. A. Khalaf, and H. F. Hamed, "Authentication and identity management of ioht devices: Achievements, challenges, and future directions," *Computers & Security*, vol. 111, p. 102491, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821003151

[203] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: A tutorial," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1126–1141, 2014.

[204] H. Kang, Y. Hori, T. Katashita, M. Hagiwara, and K. Iwamura, "Cryptographie key generation from puf data using efficient fuzzy extractors," in *16th International conference on advanced communication technology*. IEEE, 2014, pp. 23–26.

[205] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "Fpga intrinsic pufs and their use for ip protection," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2007, pp. 63–80.

[206] U. Rührmair, H. Busch, and S. Katzenbeisser, "Strong pufs: models, constructions, and security proofs," in *Towards hardware-intrinsic security*. Springer, 2010, pp. 79–96.

[207] U. Rührmair and D. E. Holcomb, "Pufs at a glance," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–6.

[208] D. Nedospasov, J.-P. Seifert, C. Helfmeier, and C. Boit, "Invasive puf analysis," in *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2013, pp. 30–38.

[209] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Pro-*

ceedings of the 17th ACM conference on Computer and communications security, 2010, pp. 237–249.

[210] G. T. Becker, "On the pitfalls of using arbiter-pufs as building blocks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 8, pp. 1295–1307, 2015.

[211] N. Wisiol, C. Mühl, N. Pirnay, P. H. Nguyen, M. Margraf, J.-P. Seifert, M. van Dijk, and U. Rührmair, "Splitting the interpose puf: A novel modeling attack strategy," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 97–120, 2020.

[212] A. Vijayakumar and S. Kundu, "A novel modeling attack resistant puf design based on non-linear voltage transfer characteristics," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 653–658.

[213] A. Mahmoud, U. Rührmair, M. Majzoobi, and F. Koushanfar, "Combined modeling and side channel attacks on strong pufs." *IACR Cryptol. ePrint Arch.*, vol. 2013, p. 632, 2013.

[214] A. Vijayakumar, V. C. Patil, C. B. Prado, and S. Kundu, "Machine learning resistant strong puf: Possible or a pipe dream?" in *2016 IEEE international symposium on hardware oriented security and trust (HOST)*. IEEE, 2016, pp. 19–24.

[215] J. Delvaux, R. Peeters, D. Gu, and I. Verbauwhede, "A survey on lightweight entity authentication with strong PUFs," *ACM Computing Surveys*, vol. 48, no. 2, pp. 1–42, nov 2015.

[216] A. Mahmoud, U. Rührmair, M. Majzoobi, and F. Koushanfar, "Combined modeling and side channel attacks on strong pufs," *Cryptology ePrint Archive*, 2013.

[217] S. Tajik, E. Dietz, S. Frohmann, H. Dittrich, D. Nedospasov, C. Helfmeier, J.-P. Seifert, C. Boit, and H.-W. Hübers, "Photonic side-channel analysis of arbiter pufs," *Journal of Cryptology*, vol. 30, no. 2, pp. 550–571, Apr 2017. [Online]. Available: https://doi.org/10.1007/s00145-016-9228-6

[218] D. Merli, D. Schuster, F. Stumpf, and G. Sigl, "Semi-invasive em attack on fpga ro pufs and countermeasures," in *Proceedings of the Workshop on Embedded Systems Security*, ser. WESS '11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/2072274.2072276

[219] A. R. Korenda, F. Afghah, B. Cambou, and C. Philabaum, "A proof of concept SRAM-based physically unclonable function (PUF) key generation mechanism for IoT devices," in *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, jun 2019.

[220] C. Böhm, M. Hofer, and W. Pribyl, "A microcontroller sram-puf," in *2011 5th International Conference on Network and System Security*. IEEE, 2011, pp. 269–273.

[221] D. Nedospasov, J.-P. Seifert, C. Helfmeier, and C. Boit, "Invasive puf analysis," in *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2013, pp. 30–38.

[222] D. Mukhopadhyay, "Pufs as promising tools for security in internet of things," *IEEE Design & Test*, vol. 33, no. 3, pp. 103–115, 2016.

[223] M. A. Qureshi and A. Munir, "PUF-IPA: A PUF-based identity preserving protocol for internet of things authentication," in *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*, IEEE. IEEE, jan 2020, pp. 1–7.

[224] K. B. Frikken, M. Blanton, and M. J. Atallah, "Robust authentication using physically unclonable functions," in *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, pp. 262–277.

[225] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM conference on Computer and communications security - CCS '02*. ACM Press, 2002.

[226] D. Lim, J. Lee, B. Gassend, G. Suh, M. van Dijk, and S. Devadas, "Extracting secret keys from integrated circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 10, pp. 1200–1205, oct 2005.

[227] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM Press, 2010, pp. 237–249.

[228] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Controlled physical random functions," in *18th Annual Computer Security Applications Conference, 2002. Proceedings.* IEEE Comput. Soc, 2002.

[229] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Lightweight secure PUFs," in *2008 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, nov 2008.

[230] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas, "Slender PUF protocol: A lightweight, robust, and secure authentication by substring matching," in *2012 IEEE Symposium on Security and Privacy Workshops*. IEEE, may 2012.

[231] J. Delvaux and I. Verbauwhede, "Fault injection modeling attacks on 65 nm arbiter and RO sum PUFs via environmental changes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 6, pp. 1701–1713, jun 2014.

[232] Ü. Kocabaş, A. Peter, S. Katzenbeisser, and A.-R. Sadeghi, "Converse puf-based authentication," in *Trust and Trustworthy Computing*, S. Katzenbeisser, E. Weippl, L. J. Camp, M. Volkamer, M. Reiter, and X. Zhang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 142–158.

[233] M.-D. Yu, M. Hiller, J. Delvaux, R. Sowell, S. Devadas, and I. Verbauwhede, "A lockdown technique to prevent machine learning on PUFs for lightweight authentication," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 3, pp. 146–159, jul 2016.

[234] Y. Gao, H. Ma, S. F. Al-Sarawi, D. Abbott, and D. C. Ranasinghe, "PUF-FSM: A controlled strong PUF," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2017.

[235] A. Braeken, "Puf based authentication protocol for iot," *Symmetry*, vol. 10, no. 8, p. 352, 2018.

[236] U. Chatterjee, V. Govindan, R. Sadhukhan, D. Mukhopadhyay, R. S. Chakraborty, D. Mahata, and M. M. Prabhu, "Building PUF based authentication and key exchange protocol for IoT without explicit CRPs in verifier database," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 3, pp. 424–437, may 2019.

[237] M. Ebrahimabadi, M. Younis, and N. Karimi, "A PUF-based modeling-attack resilient authentication protocol for IoT devices," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[238] Z. Huang and Q. Wang, "A puf-based unified identity verification framework for secure iot hardware via device authentication," *World Wide Web*, vol. 23, no. 2, pp. 1057–1088, 2020.

[239] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede, "Helper data algorithms for PUF-based key generation: Overview and analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 889–902, jun 2015.

[240] A. Shamsoshoara, A. Korenda, F. Afghah, and S. Zeadally, "A survey on hardware-based security mechanisms for internet of things," *ArXiv. org*, 2019.

[241] B. Škorić, P. Tuyls, and W. Ophey, *Robust Key Extraction from Physical Uncloneable Functions*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 407–422.

[242] K. Kursawe, A.-R. Sadeghi, D. Schellekens, B. Skoric, and P. Tuyls, "Reconfigurable physical unclonable functions - enabling technology for tamper-resistant storage," in *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*. IEEE, 2009.

[243] G. Suh, C. O'Donnell, I. Sachdev, and S. Devadas, "Design and implementation of the AEGIS single-chip secure processor using physical random functions," in *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, 2005.

[244] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *2007 44th ACM/IEEE Design Automation Conference*, 2007, pp. 9–14. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4261134

[245] I. Eichhorn, P. Koeberl, and V. van der Leest, "Logically reconfigurable PUFs," in *Proceedings of the sixth ACM workshop on Scalable trusted computing - STC '11*. ACM Press, 2011.

[246] L. Zhang, Z. H. Kong, and C.-H. Chang, "PCKGen: A phase change memory based cryptographic key generator," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*. IEEE, may 2013.

[247] J. Zhang and G. Qu, "Physical unclonable function-based key sharing via machine learning for IoT security," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 8, pp. 7025–7033, aug 2020.

[248] IEEE, "Ieee standard for low-rate wireless networks," *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)*, pp. 1–800, 2020.

[249] D. Geelen, G. van Kempen, F. van Hoogstraten, and A. Liotta, "A wireless mesh communication protocol for smart-metering," in *2012 International Conference on Computing, Networking and Communications (ICNC)*, Jan 2012, pp. 343–349.

[250] Espressif Systems, "Esp32-c3 datasheet," March 2021, Accessed: Mar. 6, 2023. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf

[251] ——, "ESP32-H2," 2021, Accessed: Apr. 11, 2023. [Online]. Available: https://www.espressif.com/en/products/socs/esp32-h2

[252] ——, "Esp32-c3 datasheet," March 2021, Accessed: Mar. 6, 2023. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-c3_technical_reference_manual_en.pdf

[253] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.
[254] Ai-Thinker, "Esp-c3-13 specification," June 2021, Accessed: Mar. 6, 2023. [Online]. Available: https://docs.ai-thinker.com/_media/esp32/docs/esp-c3-13_specification.pdf
[255] Espressif Systems. (2016) ESP-IDF Startup API Guide. Accessed: Mar. 6, 2023. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/api-guides/startup.html
[256] S. Lyubka, "mdk," 2022, Accessed: Mar. 6, 2023. [Online]. Available: https://github.com/cpq/mdk
[257] Texas Instruments, "Spst cmos analog switches," Datasheet, 2006, Accessed: Mar. 21, 2023. [Online]. Available: https://www.ti.com/lit/ds/symlink/ts12a4514.pdf
[258] NewAE Technology Inc., "ChipWhisperer-Nano," Product page, 2022, Accessed: Mar. 21, 2023. [Online]. Available: https://rtfm.newae.com/Capture/ChipWhisperer-Nano/
[259] Riscure, "CVE-2019-17391," 2019, Accessed: Apr. 28, 2023. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-17391
[260] Saleae Inc., "Saleae logic 8 logic analyzer," 2023, Accessed: Mar. 21, 2023. [Online]. Available: https://eur.saleae.com/products/saleae-logic-8
[261] Future Technology Devices International Limited, "FTDI FT232H datasheet," Nov. 2019, Accessed: Mar. 21, 2023. [Online]. Available: https://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232H.pdf
[262] Espressif Systems, "Security advisory," Espressif Systems, Security Advisory AR2022-003, May 2022, Accessed: Apr. 26, 2023. [Online]. Available: https://bit.ly/3QbXaIV
[263] F.-X. Standaert, E. Peeters, and J.-J. Quisquater, "On the masking countermeasure and higher-order power analysis attacks," in *International Conference on Information Technology: Coding and Computing (ITCC'05)-Volume II*, vol. 1. IEEE, 2005, pp. 562–567.
[264] XGecu, "XGecu TL866II plus," Accessed: Apr. 11, 2023. [Online]. Available: http://www.autoelectric.cn/en/TL866_main.html
[265] Andromeda Research Labs, "Understanding in-circuit eeprom and microcontroller reading and programming," Accessed: Apr. 11, 2023. [Online]. Available: https://www.arlabs.com/incircuit.html
[266] National Security Agency, "Ghidra," 2019, Accessed: Apr. 11, 2023. [Online]. Available: https://ghidra-sre.org/
[267] Espressif Systems, "Esp32-arduino sketch and encryption with esp-idf," 2021, Accessed: Apr. 11, 2023. [Online]. Available: https://github.com/espressif/arduino-esp32/issues/5645
[268] S. Yoo and A. A. Jerraya, "Introduction to hardware abstraction layers for soc," *Embedded software for SoC*, pp. 179–186, 2003.
[269] L. Lathrop, S. Liebl, U. Raithel, M. Söllner, and A. Aßmuth, "Securing the internet of things from the bottom up using physical unclonable functions," *CLOUD COMPUTING 2020*, p. 44, 2020.
[270] Linaro, "Trusted firmware," 2013, Accessed: June 30, 2023. [Online]. Available: https://www.trustedfirmware.org/
[271] J. King, "HKG18-212 - Trusted Firmware M: Introduction," Linaro Connect - Hong Kong, Mar. 2018, Accessed: June 30, 2023. [Online]. Available: https://resources.linaro.org/en/resource/L7dqBYxKyyTPWk2vWHtbMj
[272] Linaro, "MCUBOOT," 2017, Accessed: June 30, 2023. [Online]. Available: https://github.com/mcu-tools/mcuboot