IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 26 May 2023, accepted 12 July 2023, date of publication 27 July 2023, date of current version 1 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3298833

SURVEY

Firmware Integrity Protection: A Survey

ANTOINE MARCHAND[1,2], YOUCEF IMINE[2,3], HAMZA OUARNOUGHI[2,3], TITOUAN TARRIDEC[1], AND ANTOINE GALLAIS[2,3]
[1]Orange Cyberdefense, 92983 Paris La Défense, France
[2]LAMIH, CNRS, UMR 8201, Université Polytechnique Hauts-de-France, 59313 Valenciennes, France
[3]INSA Hauts-de-France, 59313 Valenciennes, France

Corresponding author: Antoine Marchand (antoine.marchand@uphf.fr)

This work was supported by the Agence Nationale Recherche Technologie (ANRT) to Orange Cyberdefense, France, under Grant 2020/1361.

ABSTRACT Over the last years, firmware integrity protection has significantly been addressed in various contexts such as internet of things, vehicles, wireless sensor networks and other systems. However, due to variety of studied systems, the proposed approaches often make divergent and sometimes even conflicting security assumptions. In this regard, we propose in this article a complete survey of the most relevant approaches addressing the firmware integrity protection regardless the considered system or field of application. We first organize the approaches in three main categories, depending on the protection type: 1. secure update, 2. attestation, and 3. secure boot. We then propose two new taxonomies, the first one concerns secure update mechanisms and the second one considers secure boot. Moreover, we extend the scope of an existing taxonomy for attestation mechanisms by studying new approaches and discussing limitations. Finally, we identify open research challenges and then give suggestions and guidelines on how to address them.

INDEX TERMS Survey, firmware, integrity, secure update, attestation, secure boot.

I. INTRODUCTION

Firmware, these pieces of software rooted in hardware run trustfully and flawlessly, or do they? Nowadays, firmware have become ubiquitous and every single smart or not-so-smart device embeds at least one. That is why firmware are also a prime target for attackers. Indeed, as firmware often have very high privileges due to their low-level execution, their tampering may lead to complete platform compromise. Tampering with a firmware can therefore have several consequences depending on its application. For instance, while a malicious firmware on a computer can be used to steal data by compromising an encryption key, it can on other circumstances have severe consequences. For example, we can imagine what an attacker could do with a compromised pacemaker firmware.

Several attacks against firmware have been demonstrated in the past:

- Bootkit, which is a malicious software that runs at the firmware level (and often integrated into the firmware), has been shown feasible for different platforms. For instance, an article published in the Phrack Magazine [1] shows how to make and integrate a bootkit into old Phoenix and Award BIOS firmware while other articles [2], [3], [4], [5] show different approaches to implement a bootkit into a firmware using System Management Mode (SMM) [6]. After the disclosures of NSA's mass surveillance projects by Edward Snowden in 2013, we know that NSA developed bootkits targeting servers and network equipment [7].
- Several update systems were abused to install malicious firmware updates. In the Black Hat USA 2019, Alex Matrosov and Alexandre Gazet showed that Lenovo's T540p firmware update procedure lacks signature verification so attackers could potentially install any update they want to the embedded controller which can lead to denial of service or data exfiltration [8]. Moreover, [9] shows how to circumvent the update procedure of a Dell Latitude E6400[1] platform.

The associate editor coordinating the review of this manuscript and approving it for publication was Cong Pu.

[1]Running a firmware with revision A29.

- IoT devices are not exempt. In 2015, Chris Valasek and Charlie Miller remotely took control of a Jeep car [10]. They were able to remotely log into the Jeep display and multimedia system due to a lack of authentication and then reflash a processor which can communicate to the Controller Area Network (CAN) bus. Once they can write arbitrary messages to the CAN bus, they had control of the whole vehicle and can for instance turn the wipers on and off, prevent the car from braking or control the car's steering. The review of Kim et al. [11] also shows that it is possible to update the firmware of some Electronic Control Units (ECUs) without prior authentication.

A. MOTIVATION

Considering these attacks, the widespread of firmware and their high privileges, it is of utmost importance to protect their integrity against attackers. Therefore, several research work in the literature have been proposed to ensure security of firmware.

Among these works, some papers [12], [13], [14] focus only on confidentiality service aiming to protect the intellectual property of firmware providers and to enhance the quality of their services. Whereas, other works propose solutions for firmware integrity verification at the users level.

In this paper, we focus on the solutions aiming to ensure integrity protection since we believe that it covers more challenging firmware security problems on both the providers and the users levels. Moreover, these security problems are usually not limited to integrity service but also covers confidentiality and sometimes availability services. Therefore, it represents a more generalized context to review. For these reasons, we have reviewed numerous surveys which study this issue for a specific security mechanism or field of application (see Table 1).

Attestation was first studied by Steiner et al. [15]. In this survey, the authors present a taxonomy of attestation in the context of Wireless Sensor Networks (WSNs). In [19], the authors take a different approach and focus on hardware and hybrid attestation in the context of network infrastructures. In this context, the resources of the nodes that compose the system are not as constrained as in WSNs, so it is possible to use different methods that are often too costly in terms of money or resources to be used in WSNs. In [20], the authors study attestation in the more general context of embedded systems. Finally, in the survey of Kuang et al. [21], the authors study remote attestation in the context of IoT devices. They consider different adversary models in order to encompass as many approaches as possible.

About secure update, the studied surveys always consider the context of IoT. In [16], Kolehmainen presents challenges to securely update an IoT device as well as ways of solving them. The author then presents three approaches to do a secure update: SUIT, LWM2M and blockchain. Finally, the author proposes a high-level model of secure update. In [17], the authors first present prerequisites to perform a secure update before proposing several attack categories. Finally the authors present several centralized and decentralized approaches to perform a secure update. They also discuss the advantages and disadvantages as well as possible improvements of decentralized solutions. Finally, in [18], the authors present several secure firmware update mechanisms which are best suited for each class of IoT devices (low-end, medium-end and high-end devices). They also compare several client-server based (centralised) and blockchain-based (decentralised) solutions according to their features.

Secure boot has been studied by Wang et al. [22] in 2022. Through their comprehensive survey, the authors have presented an extensive and insightful overview of existing secure boot schemes for embedded devices. They have conducted an in-depth analysis of these schemes, carefully comparing them to highlight their respective strengths and weaknesses. Finally, they propose a new classification of existing secure boot methods divided into three categories: hardware, software, and hybrid methods.

As we can notice, the surveys cited above target mainly embedded technologies (IoT, WSNs). Our survey is different from previous works since we consider firmware in general, regardless of their field of application (IoT, WSN, smart grid, etc.) or on which platform they are used.

We also consider static code analysis and control flow protection as out of our scope. Finally, we have seen several papers about supply chains attacks, showing that they may impact firmware integrity [23], [24], [25]. However, we think that supply chain attacks are a research topic on their own and we will therefore not discuss them in this survey.

B. OUR CONTRIBUTIONS

The contributions of our paper can be summarized as follows:

- We provide a comprehensive and extensive study that investigates existing firmware integrity protection mechanisms.
- We propose a categorization of the aforementioned security mechanisms upon their characteristics.
- We analyse and compare these techniques to show their advantages and disadvantages.
- We identify several open research problems and we give some suggestions on how to address them.

In this article, we start by giving an overview of research trends and challenges for every field of application in II. Then, we survey different protection types in sections III, IV and V. In these sections, we first introduce useful concepts to understand each protection type and we define their assumptions, adversary models and goals. Then, we introduce a taxonomy which shows the differences between existing solutions for each protection type. Each taxon is detailed and illustrated with examples. Taxa in a same group are then discussed to exhibit their advantages and disadvantages. In VI

TABLE 1. Comparison of existing surveys in firmware integrity protection.

	Date	Field of application	Secure update	Attestation	Secure boot
Steiner et al. [15]	09/2016	WSN		●	
Antti Kolehmainen [16]	08/2018	IoT	●		
Bettayeb et al. [17]	03/2019	IoT	●		
Mtetwa et al. [18]	11/2019	IoT	●		
Sfyrakis et al. [19]	05/2020	Network Infrastructures		●	
Banks et al. [20]	05/2021	Embedded		●	
Kuang et al. [21]	10/2021	IoT		●	
Wang et al. [22]	03/2022	Embedded			●

we then discuss open research problems, analyze challenges to be solved and give guidelines on how to tackle them. Finally, we conclude this article in VII.

II. FIELDS OF APPLICATION

In this section we will first analyze research trend depending on application fields. In a second time, we will see challenges for designing a firmware integrity protection solution for each of these fields of application.

In the remainder of this article we will use the terms *general-purpose computer* and *proprietary project*, so it is necessary to define them first. We define a general-purpose computer as a versatile high-end system and thus not specialized in the execution of a specific task. This notion is opposed to embedded systems which are generally constrained in terms of resources and specialized in one or several precise tasks. A proprietary project is defined as a project owned by a private company as opposed to projects published in the scientific literature.

A. RESEARCH TREND

Figure 2 shows the distribution of the studied approaches by field of application. The category *embedded* groups all the works considering embedded systems but not mentioning the type of the targeted system. We can observe that the distribution of fields of application is unbalanced. Indeed, we notice that embedded systems represent more than three quarters of the published approaches while the works focusing on general-purpose computers are rather marginal. This disparity is even more significant as these works include so-called proprietary projects.

The solution targeting smart grids [26] has been considered separately. Indeed, even if smart meters can be seen as embedded systems, smart grids do not necessarily contain only smart meters and are not necessarily composed exclusively of embedded systems. The constraints, assumptions and adversary model can therefore be different from those of embedded systems depending on the approach and the considered grid.

Figure 1 shows a growing interest in the study of firmware integrity protection for the different fields of application. We can also see that approaches targeting WSNs are starting to be replaced by approaches targeting the IoT as of 2016. This can be explained by a change in terminology since these systems are similar in terms of firmware integrity protection challenges. Finally, we can notice a stagnation of approaches focusing on general-purpose computers from 1997 until 2012, the year from which work on this field of application really starts to develop. These observations show that nowadays firmware integrity protection is both an interesting scientific and industrial research topic for several fields of application.

B. CHALLENGES

1) EMBEDDED SYSTEMS

The development of an embedded system solution must take into account various constraints [27], [28]. Some of them are common to all embedded systems when others are specific to a subset of these systems. First, embedded systems are characterized by limited computing and memory resources. That is why, Park et al. [29] propose a new protocol which is based on a novel randomized hash function tailored for low resources CPU. Also, they need to be power efficient because their power source can be limited: photovoltaic cell or battery for example. For instance, Deng et al. [30] explain that battery can be easily exhausted by an attacker re-transmitting packets to the device, thus leading to a Denial of Service (DoS). Moreover, Jin et al. [31] propose an approach that eliminates redundant processing overhead in order to reduce power usage of battery-powered sensor nodes. Finally, these systems should stay cheap [32], [33] and sometimes small [33].

Embedded systems can be used in critical applications requiring continuous [34] and sometimes real-time [35] operation. This constraint requires that the system is neither interrupted nor disturbed by the used protection mechanism. This implies for example to ensure the compatibility between the update and the target hardware [36], [37], [38]. Indeed, the setup of an incompatible update can brick the system or cause a malfunction at least.

The systems used in the WSNs, IoT and smart grids contexts have a large number of nodes distributed in their environment [15], [34], [36], [39], [40]. It is therefore infeasible for most application scenarios to physically reach and check the integrity of each node. The verification process should then be done remotely [15]. The scalability of this process is therefore essential to be practical in this context. Another

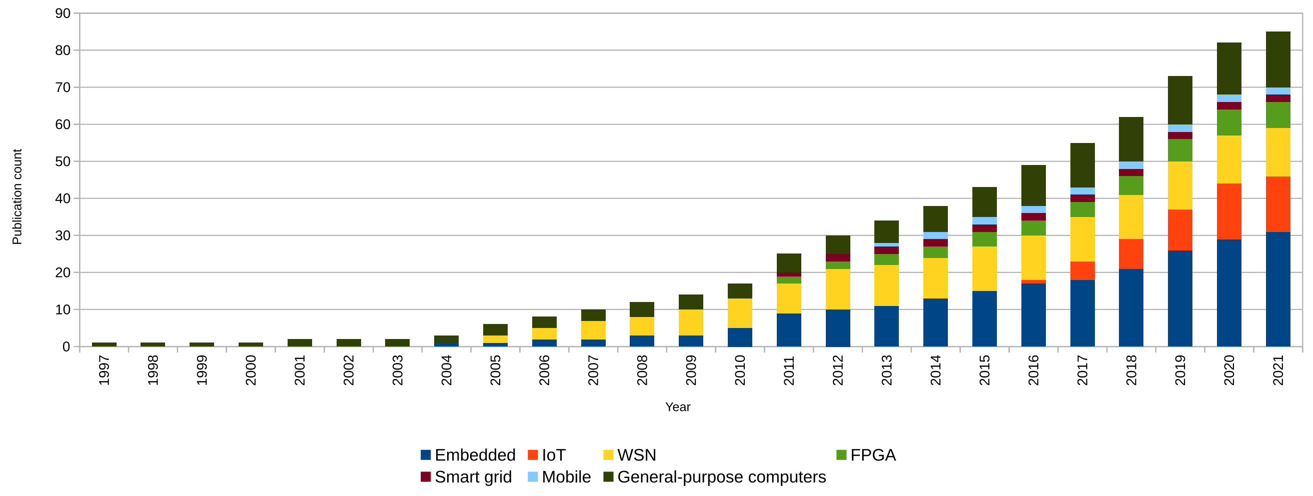


FIGURE 1. Histogram of publications count grouped by field of application.

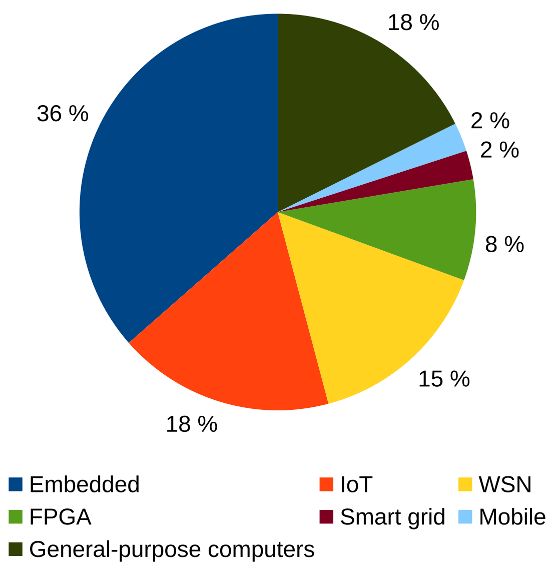


FIGURE 2. Distribution of publications grouped by field of application from 1997 to 2021.

problem encountered in these applications is the communication between the nodes. These nodes often communicate wirelessly which is challenging considering their large number and the unreliability of these communications [15], [16], [41], [42].

The approach of Gonzalez et al. [43] shows a specific challenge for smartphones used in the context of Bring Your Own Device (BYOD). In other words, the authors study smartphones used in both personal and professional environments. It is therefore important for the company to be able to guarantee that the operating system used comes from a trustworthy source and complies with the company's security policy. The authors explain that designing such a solution is challenging given the diversity in both smartphones and the used operating systems.

2) GENERAL-PURPOSE COMPUTERS

Physical attacks should be considered in the context of general-purpose computers [44], [45]. These systems are indeed often accessible and can be more easily subverted by an attacker. We can for example mention the Evil Maid attack [46] which consists in capturing the device while it is left unattended (for example in a hotel room) and then making the desired alterations on the system. We can also note that protecting the system from these attacks is seldom considered [47], [48], [49], [50].

General-purpose computers are not fixed monolithic systems. They can be extended with peripheral cards of various types and functions. However, these peripheral cards usually contain a firmware that can be used by an attacker to compromise the entire system. For example, the work of Duflot et al. [51] shows how compromising a network card allows to insert a backdoor into the operating system. Checking the integrity of the peripherals' firmware is therefore crucial to ensure that they do not use vulnerable and/or malicious firmware.

In the same way as embedded systems used in critical applications, general-purpose computers have sometimes to meet high availability constraints [52], [53]. This can for instance happen for servers or workstations. In this context, it is necessary, as mentioned previously, to ensure the availability of the protected system. The said protection must therefore under no circumstances compromise the good functioning of the system on which it is used.

C. OVERVIEW

The diversity of fields of application studied also implies a diversity of considered constraints and addressed challenges. This is why several approaches have been proposed in the state of the art. We can classify these approaches according to three main categories: secure update, attestation and secure boot of firmware (see Figure 4).

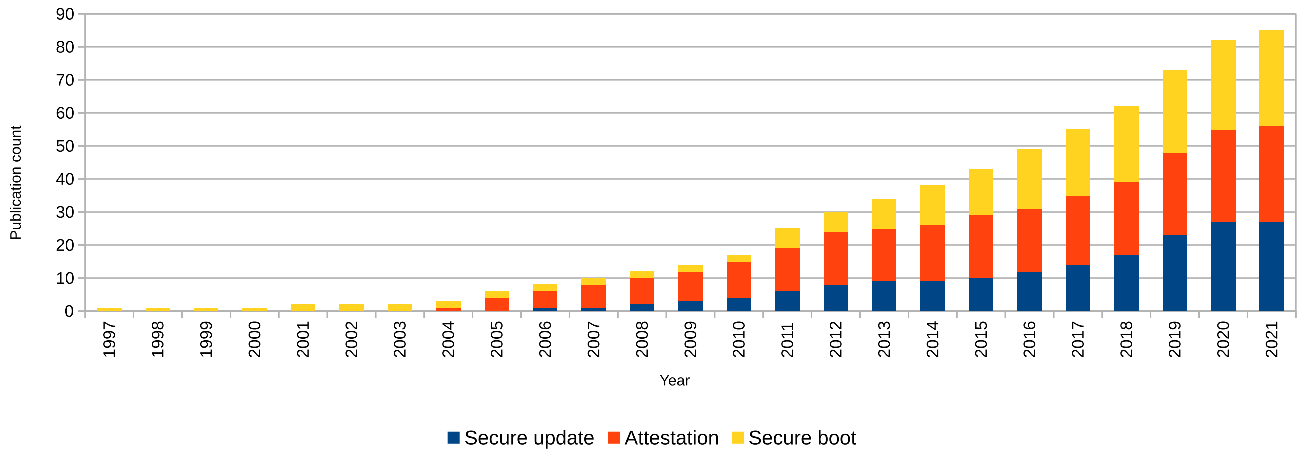


FIGURE 3. Histogram of publications count grouped by security type.

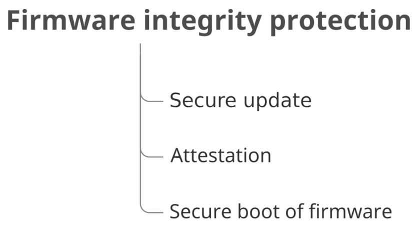


FIGURE 4. Studied firmware integrity protection types.

Figure 3 shows the evolution of the state of the art works over the years, depending on the type of security. We can notice that interest in this research area is growing for all types of security. In the rest of this survey, we detail each of these categories and we classify the associated approaches using a taxonomy specific to each of them.

We would like to draw reader's attention to the fact that, in this paper, we only consider the secure boot of firmware. As mentioned by some work [22], secure boot is a security mechanism checking the whole boot chain: from firmware to operating system, including drivers. Therefore, secure boot ensures the device's overall integrity and not only firmware's integrity. However, in this paper we study secure boot from the point of view of firmware integrity protection as described in the Unified Extensible Firmware Interface (UEFI) specifications [54] and in several papers [55], [56].

III. SECURE FIRMWARE UPDATE

In this section, we first formally describe secure firmware update mechanisms. Then, we review the main characteristics of existing secure update approaches and we propose a new taxonomy (shown in Figure 5) which identifies four major characteristics of secure firmware update mechanisms.

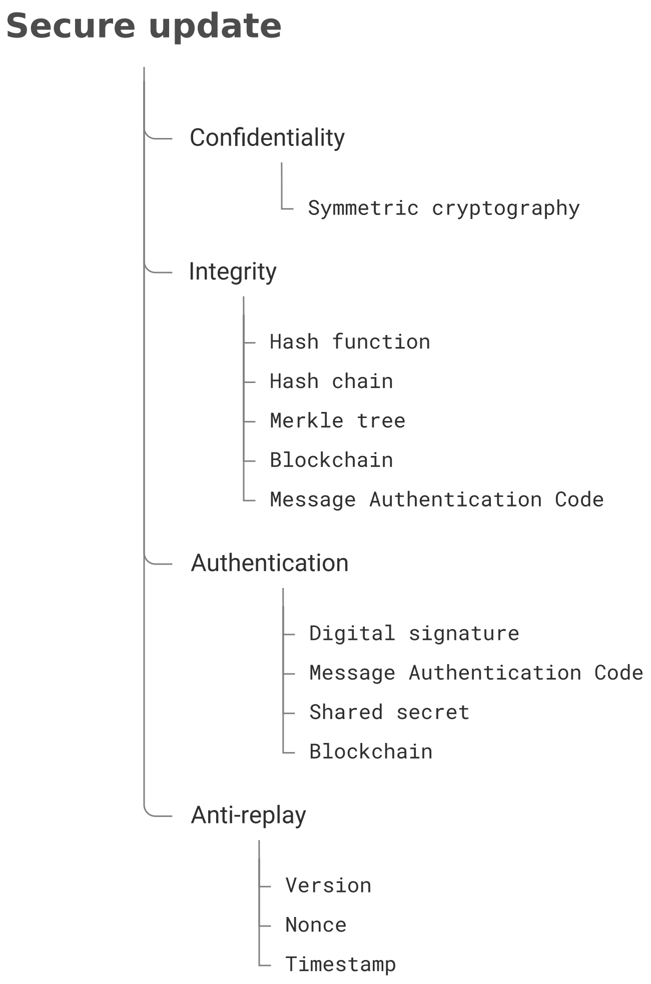


FIGURE 5. A taxonomy of secure firmware update mechanisms.

A. BACKGROUND

A secure update is a process by which a server $\mathcal{S}$ performs the firmware update of a client $\mathcal{C}$ via a communication link $\mathcal{L}$. The server $\mathcal{S}$ is commonly assumed to be safe [30], [34], [36], [41], [57] and $\mathcal{C}$ is assumed to be uncompromised before the update (we come back to this point in part III-A2). We note *Safe*($\mathcal{S}$) the proposition "the server $\mathcal{S}$ is safe", *Compromised*($\mathcal{C}$) the proposition "the client $\mathcal{C}$ is compromised before the update'' and $Secure(\mathcal{L})$ the proposition ''the communication link $\mathcal{L}$ is secure''.

This process can be generalized in the case where several hops are used. In this case, the update transits through N servers (noted $\mathcal{S}_i$) via N communication links noted $\mathcal{L}_i$. In this case, the update process is said to be ''secure'' if and only if:

$$\neg Compromised(\mathcal{C}) \wedge \bigwedge_{i=1}^{N} Safe(\mathcal{S}_i) \wedge \bigwedge_{i=1}^{N} Secure(\mathcal{L}_i) \qquad (1)$$

In the same way, the update of a set of M clients (with $M > 1$) by a server $\mathcal{S}$ via a communication link $\mathcal{L}$ is considered secure if and only if:

$$\bigwedge_{i=1}^{M} (\neg Compromised(\mathcal{C}_i)) \wedge Safe(\mathcal{S}) \wedge Secure(\mathcal{L}) \qquad (2)$$

1) CHALLENGES

A secure update mechanism must address one or more of the following challenges:

- **Confidentiality**: Prevent an attacker from being able to analyze the firmware. This prevents vulnerability research via binary analysis or reverse engineering and prevents the theft of intellectual property.
- **Integrity**: Prevent the installation of a corrupted update image.
- **Authentication**: Prevent an attacker to craft updates that would have been accepted by the client otherwise. The authentication mechanism makes sure only authorized entities can publish and install updates to the client.
- **Anti-replay**: Prevent an attacker that has previously captured a session to replay it. This prevents an attacker from reinstalling a legitimate but old - and therefore potentially vulnerable - version of the firmware.

The $\mathcal{L}$ link thus does not need to address all the challenges we have introduced to be secure. Indeed, these challenges are intended to be general and may be irrelevant depending on the threat model considered.

2) ADVERSARY MODEL

The papers we have studied often adopt the Dolev-Yao [58] model for communications between the server and the client [30], [34], [36], [39], [41]. That is, the attacker can eavesdrop, intercept, modify and inject packets from / to the network. However, the approach proposed by Yesilyurt et al. [59] is more restrictive and consider that the attacker can only eavesdrop the traffic on the network.

For embedded systems with limited computing power, it is sometimes assumed that attackers have more computing power than the client they have compromised [41].

Although the client is considered to be uncompromised prior to the update, some works consider that the client may be under certain conditions. For [30], attackers can compromise a client but they can only compromise the confidentiality objective. This assumes that attackers only have read access and can extract the encryption key stored within the client's memory. However, attackers do not seem to have write access to the client. That is why they can not tamper with the firmware - and therefore the update module - embedded in the client. In the same way, for [39], attackers cannot compromise the whole client since the bootloader is supposed to be safe and immutable. We can see that whatever the proposed solution, the client must remain uncompromised at least as far as the update module is concerned.

B. TAXONOMY

We have chosen to build our taxonomy upon the security goals to be achieved. We have therefore categorized existing solutions according to the mechanisms used to achieve the following security goals: confidentiality, integrity, authentication and anti-replay. We believe that this classification is relevant given the diversity of existing approaches. Indeed, they target various applications, each of them has specific constraints as we have mentioned in the section II. Moreover, they do not use the same adversary model and do not make the same assumptions. These approaches will therefore use different mechanisms to reach the security goals.

1) CONFIDENTIALITY

The updated firmware is usually transmitted over communication channels that are not necessarily secure. Therefore, it can be intercepted and analyzed by an attacker. Confidentiality may be required by manufacturers to protect their intellectual property or to prevent reverse engineering in case sensitive data are stored inside the firmware. In the case where confidentiality must be ensured, symmetric cryptography is used by all approaches we have studied. However, the encryption key is not always stored or generated the same way. Approaches which do not protect firmware confidentiality do most of the time not mention it in their security goals [26], [34], [37], [39], [60], [61] or do not take it into account in their adversary model [36]. However, we can consider the example of the work of Deng et al. [30] in which the authors explicitly state that they do not take into account the confidentiality aspect because of an orthogonal assumption with this objective. They make the assumption that the attacker is able to compromise a device completely and access all the information it contains. In this case, it is complex or even impossible to protect the confidentiality of the update since the attacker can, by assumption, access every encryption key stored within the device.

The approaches of Nilsson et al. [41], Thanh et al. [62] and Yesilyurt et al. [59] propose to pre-install a shared encryption key between the server and the client. In this case, each client has a different encryption key and the server must store them all. However, the preinstalled key can also be shared between all devices. For instance, the work of Kerliu et al. [63] relies on such design. Sharing a key between all devices is convenient as only one key needs to be stored but an attacker who is able to compromise one device is then able to compromise the whole system. Finally, Guillen et al. [64]

propose a protocol which is able to update the pre-installed keys into the device. Their protocol uses dedicated packets to update the keys independently of the firmware update. Moreover, keys version numbers are also stored into the device so that an attacker is unable to downgrade pre-installed keys to a previous version. In all cases described above, the key pre-installation should be done in a safe environment so that an attacker will not be able to get the encryption key.

Another method is to use a randomly generated session key each time it is needed. One way to do so is to use the TLS protocol [38], [65], [66], [67], [68]. These approaches benefit from the handshake procedure and therefore generate a different session key for each connection. The symmetric encryption algorithm used in this case is chosen during the handshake and therefore cannot be known in advance. However, an up-to-date implementation of TLS should only offer the following algorithms: AES-GCM [69], AES-CCM [70], Camellia-GCM [71], ARIA-GCM [72] or ChaCha20-Poly1305 [73]. The work of Braeken et al. [57] uses a Diffie-Hellman like key exchange scheme in which both clients generate a random secret key for each communication. Finally, the work of Choi et al. [74], Keleman et al. [75] and Feng et al. [76] use a temporary session key but do not specify an algorithm to use.

Another idea is to use a Physical Unclonable Function (PUF) to generate a key. Falas et al. [77] uses the output of a Public PUF (PPUF) stimulated by a randomly chosen challenge as the encryption key. In another of their work [78] they instead use the challenge as the encryption key and send the expected output to the client along with the encrypted update. However, in the worst case, this method requires the client to iterate through every possible challenge before finding the good one. Note that these approaches work only with PPUF as the server also needs to know the output of the PUF according to a given challenge. Finally, the approach of Prada-Delgado et al. [35] uses an SRAM PUF with helper data and error-correcting codes to generate the encryption key. In this approach, the output of the PUF is only used once. After every update, the previous PUF's output is derived using a key derivation function and the helper data are updated accordingly so that the PUF will generate the expected output the next time.

The approach of Siddiqui et al. [79] propose to update the encryption key which is stored into a TPM along with the firmware. In this approach, the current encryption key is used to encrypt the value of a PCR in the TPM (reflecting the state of the device) xor-ed with a random value which is generated by the TPM's random number generator. This approach allows to update the encryption key automatically and thus ensures that each device has its own key which is valid only for the next update.

Finally, we also note that some works do not specify the encryption algorithm to be used. This is the case of the works of Feng et al. [76], Keleman et al. [75], Nilsson et al. [41] and Choi et al. [74] which propose to use a symmetric encryption algorithm without specifying which one. These approaches propose a generic protocol which independent of the symmetric encryption algorithm used. This algorithm can therefore be chosen according to the specific constraints of each implementation context.

2) INTEGRITY

Integrity protection is a fundamental issue in a secure update system. It is crucial that the update received by the client is the same as the one sent by the server, in other words that the update received by the client is not corrupted. The methods shown hereafter allow to detect an accidental corruption of an update but are however inefficient to protect against an attacker who can intercept and modify the information exchanged between the server and the client. This is why an authentication mechanism is often used in conjunction with the methods described below. We will come back to the authentication methods in the sub-section III-B3.

The most basic solution is to use a hash function computed over the entire firmware update. In this case, the validity of the digest is usually validated before installing the update. However, the approach of Aschenbruck et al. [36] transmits the update hash to the client after sending it. The verification is therefore done after the update has been installed, which requires the deletion of the update and the rollback of the device in case of a corrupted update. The digest is most of the time computed over the update plaintext, however the approach of Siddiqui et al. [79] computes the hash over the encrypted one. The client can thus verify the integrity of the update before the decryption operation, which avoids a cryptographic operation in case of a corrupted update. Note that the approach of Aschenbruck et al. [36] uses SHA-1 which has been deprecated by NIST for digital signatures since 2013 [80]. Moreover, a practical collision on this algorithm was found in 2017 [81].

The update can also be split into several packets to avoid having to retransmit the complete update in case of an error. Splitting the update makes it possible to identify the source of the error more precisely and thus to retransmit only the corrupted block. This segmentation allows new approaches to protect the integrity of each block. It is now possible to use data structures such as hash chains [30], [41], [74] or Merkle trees [30], [39], [76]. The hash chains ensure the integrity of every block except the first one. This is why an authentication system of the first block is necessary to ensure the integrity and the authenticity of the whole chain. The Merkle trees have the advantage of allowing a chaotic reception of the elements constituting it. In other words, the packets can be received in disorder without affecting the final calculation. Moreover, the modification of a node imposes to recalculate only the hashes of the impacted branch contrary to a hash chain which requires the recalculation of all the hashes depending on the modified element. However, for the same number of blocks, a Merkle tree requires a larger number of hashes than a hash chain and imposes that the client receives an index of hashes *a priori*. In their work, Deng et al. [30] also propose a hybrid structure combining hash chain and Merkle tree. The update is divided into pages and then each page is itself divided into several packets. A Merkle tree is used to ensure the integrity of a page and then the root of each tree is added into a hash chain. This way, only the authentication of the first element of the hash chain is needed and we still benefit from the advantages of the Merkle tree within each page.

Some approaches propose to use a blockchain to store the updates metadata [38], [60], [66], [68], [82]. The firmware, too large to be stored in a block, is then externalized and the metadata allows the client to know the location of the file to be downloaded. The transfer of the file itself is done by BitTorrent [60], IPFS [38], [66] or a web platform managed by the vendor [68]. The integrity of the update is then ensured by a hash function whose output is compared with a reference stored into the metadata.

Integrity protection of the transmitted data can also be ensured by a Message Authentication Code (MAC). In this case it is important to note that integrity and authentication can be ensured by the same function. Indeed, a modification of the data will lead to an error during the verification of the MAC, whether this modification is fortuitous (corruption due to the transmission channel) or malicious (spoofing attack attempt). A valid MAC therefore implies that data are not corrupted and authentic, but if it is not the case, it is impossible to know the origin of the error. We will see in more details how MAC is used in III-B3.

Finally, some works do not protect the integrity of the update. For example, Katzir and Schwartzman [26] do not address this problem. Moreover, Yesilyurt et al. [59] use a checksum for transmitted packets but do not check the integrity of the update itself. These two approaches therefore offer little or no protection in case of update corruption.

3) AUTHENTICATION

The authenticity of the update is often verified using a digital signature (asymmetric cryptography) or a MAC (symmetric cryptography). In the case of digital signatures, the algorithm to be used is based either on the prime factorization of a large number problem [67], [78] or on the discrete logarithm problem [39]. The MAC can be one output of an Authenticated Encryption (AE) algorithm [35], [63], [64], [77] or be the result of a function dedicated to this purpose. The approaches using TLS [65], [67] take advantage of the algorithm selected during the handshake and thus use a MAC even if we cannot determine in advance which one will be used. The approach of Braeken et al. [57] uses both a MAC and a digital signature to authenticate the exchanges between the server and the client. A mutual authentication between the two parties is first established using digital signatures. Other messages are then authenticated using an Hash-based MAC (HMAC).

Authentication can also be provided by a blockchain. For example, Alexandros Nanopoulos and Projjal Gupta [38], [66] use transactions on an Ethereum blockchain to store information about updates. Clients can then verify the authenticity of the transaction using the public key of the issuer. In order to handle the multi-developers and key revocation problems, the authors say it is possible to set up a multi-signatures smart contract, a certification authority or a Web-of-Trust infrastructure.

The work of Yesilyurt et al. [59] uses a shared secret to mutually authenticate the server and the client. An authentication packet is encrypted with the preinstalled key and allows the client to verify the authenticity of the server. The operation is then repeated from the client to the server so that the mutual authentication is established.

Finally, the approach presented by Katzir and Schwartzman [26] use a pattern of frequency change of the electrical network. The detection of this pattern by the concerned devices activates their ‘‘reprogramming’’ mode for a given period of time (this period is called ‘‘window of opportunity’’). The update can then be sent and will be accepted as long as this window of opportunity is valid. The update authentication is ensured by the fact that attackers are not able to predict when this pattern will appear nor can they manipulate the frequency of the electrical network. The authors state that their approach can be used in conjunction with standard cryptographic methods to enhance security.

4) ANTI-REPLAY

A popular method to prevent the replay of the update is to add the firmware version along with other data. This version is authenticated as the rest of data. This method was first proposed by Nilsson et al. [41] in the context of vehicles but as also been used on other contexts such as WSN and other embedded systems showing that this method is viable for a large number of contexts. In all cases, the version number format is not specified but must be monotonically increasing, meaning that a newer version has a strictly higher value than a previous one. With this method, the code checking the update before installation by the client is designed to ensure that the version of the update received is higher than the one currently installed. Otherwise, the update is simply cancelled. Thus, an attacker replaying a previously captured session will not be able to make the client install an older version of the firmware. Changing the version is also not an option since this change will be detected by the authentication mechanism. The approach of Hu et al. [82] is a bit different. The authors propose to check the firmware version by a smart contract stored on an Ethereum blockchain. The device, which performs the update request, invoke the smart contract by making a transaction on the blockchain. When invoking the contract, the device sends arguments such as its model, the version of the firmware currently installed and the digital signature of these data. The smart contract is then able to verify the signature with the device's public key and can therefore check whether an update needs to be done or not. This method has the advantages of authenticating the client, disclosing the update metadata only when necessary and keeping track of all updates performed.

Another method we have seen is the use of a nonce. This method is not incompatible with the previous one, and it happens that they are both used together [35], [62]. Works based on TLS also use a nonce which is generated during the handshake. Finally, the paper written by Braeken et al. [57] relies on a Diffie-Hellman like key exchange scheme. Therefore, each client generates a secret number at the beginning of the session. These secrets numbers are generated randomly just as nonces are but must obviously be kept secret.

As mentioned in the section III-B3, Katzir and Schwartzman [26] use a ''window of opportunity'' system to reprogram devices. Thus, even assuming that attackers can eavesdrop the packets when updating a device, it is impossible for them to replay the session before the next window. However, this window is supposed to be unpredictable and unforgeable by the attacker. That is why, anti-replay is guaranteed. In the same idea we find the solution proposed by He et al. [68], the authors use the term ''time window'' in their paper. In their solution, the update creation timestamp is added with other metadata into a blockchain transaction. The client then checks if the update is not too old before installing it.

C. DISCUSSION

We presented a taxonomy and detailed the techniques encountered for each taxon. Each technique was then illustrated with examples. We also discussed the advantages and disadvantages of different techniques. Finally, Table 2 maps, in chronological order, a representative number of secure update mechanisms to the taxonomy illustrated in Figure 5. To the best of our knowledge, every secure update mechanism would fit on the proposed taxonomy.

The main observation we make is the scarcity, in the scientific literature, of secure firmware update mechanisms for general-purpose computers. We think that most of the cryptographic means used for embedded solutions could be used as building blocks for the implementation of a secure update system targeting general-purpose computers. Indeed, on general-purpose computers, these algorithms are already implemented in known libraries and their resources are much more important than those of embedded systems. It is therefore unlikely that computing capacity, memory or storage will be lacking. However, even if the same cryptographic algorithms could be used in this context, it is not the same for the solutions. Indeed, as we have seen in the section II, the considered attacks and challenges for embedded systems and for general-purpose computers differ. Therefore, a solution targeting general-purpose computers cannot be a simple copy of a solution targeting embedded systems. Finally, the work of Lee et al. [83] shows a system for updating the kernel of general-purpose computers, so perhaps it is possible to build on this work and extend the solution to firmware update.

IV. ATTESTATION

In this section, we first formally describe the attestation mechanisms. Then, we review a representative number of approaches. We also extend Steiner and Lupu's [15] (see Figure 6) classification of attestation approaches used in the context of WSNs to a new classification of existing solutions regardless the nature of the system. We will therefore study new approaches and discuss them from a new point of view. We will also see the limitations we have encountered when generalizing their taxonomy.

A. BACKGROUND

Attestation is a process by which an entity called *verifier* (denoted $\mathcal{V}$) wants to obtain a proof that another entity called *prover* (denoted $\mathcal{P}$) is in an expected state S, known by $\mathcal{V}$. For this, the attestation process relies on a timed challenge mechanism (see Figure 7). A challenge c is generated by $\mathcal{V}$ using a procedure *Challenge*() then sent to $\mathcal{P}$. The challenge generation time (noted t_c) is saved by $\mathcal{V}$ and the expected answer $r_{\mathcal{V}}$ is calculated. Once the challenge is received, $\mathcal{P}$ executes an attestation routine noted $Attest(S, c)$. This attestation routine takes as parameters the internal state of $\mathcal{P}$ as well as the challenge c and produces a response (noted $r_{\mathcal{P}}$) which is sent to $\mathcal{V}$. Finally, $\mathcal{V}$ computes the reception time of the answer (noted t_r) as well as the time taken by the attestation process noted $\delta = t_r - t_c$. The verifier can then verifies that $r_{\mathcal{P}} = r_{\mathcal{V}}$ and that δ is lower than a limit Δ. If this is the case, then $\mathcal{P}$ is considered to be in the expected state S. The generic attestation process is formalized in Figure 8. A summary of the terminology used can be found in Table 3.

Firmware integrity attestation is a subset of attestation processes and can be seen as the attestation of one or more memory regions. Thus, in our case the state S is a set of $N \in \mathbb{N}^*$ disjoint memory regions, noted $M = \{m_1, m_2, \cdots, m_N\}$ such that $\forall m \in M, m = \{0, 1\}^*$. These memory regions can contain static or dynamic data and can be contiguous or not.

1) ASSUMPTIONS

In the literature, we find many assumptions around attestation-based solutions which we discuss in what follows:

The verifier cannot be compromised by the attacker. In most cases, the verifier is a separate entity from the prover and does not always have the same architecture as the latter. In this case, the verifier is often considered safe. However, not all approaches rely on this assumption. In the context of WSNs, [84] proposes a distributed attestation protocol in which the nodes of the network can all play the role of the verifier. However, since the nodes are all assumed to be vulnerable, the protocol uses a consensus protocol so that the attestation response remains trustworthy. In this case, the trustworthiness of the attestation response depends directly on the number of nodes compromised by the adversary.

The verifier knows the hardware architecture of the prover. The hardware architecture of the prover will be decisive for the design of the attestation routine. Indeed, an attestation routine can use secrets stored in tamper-resistant hardware if the platform has any [85]. Otherwise, other processes can be used to ensure the secrets integrity. For example, the work of Schulz et al. [86] uses the output of a Physical

TABLE 2. Mapping of some existing secure firmware update mechanisms to the proposed taxonomy.

Approach	Application	Confidentiality	Integrity	Authentication	Anti-replay
Deng et al. [30]	WSN	N	HC and/or MT	DS	N
Nilsson et al. [41]	Vehicle	SC	HC	DS	Version
PETRA [34]	WSN	N	MAC	MAC	N
Costa et al. [37]	Embedded	N	HF	DS	N
Braeken et al. [57]	FPGA	SC	MAC	MAC	Nonce
Window [26]	Smart grid	N	N	Y	Y
Selective [36]	WSN	N	HF	DS	Version
SDRP [39]	WSN	N	MT	DS	Version
Thanh et al. [62]	FPGA	SC	MAC	MAC	Nonce
Crypto-Bootloader [64]	Embedded	SC	MAC	MAC	Version
Choi et al. [74]	Embedded	SC	HC	DS	Version
UpdaThing [65]	IoT	SC	MAC	DS	Nonce
Boohyung et al. [60]	IoT	N	BC + HF	DS	Nonce
Prada-Delgado et al. [35]	IoT	SC	MAC	MAC	Nonce
Código Network [66]	IoT	SC	BC + HF	BC	Nonce
SOTA [59]	IoT	SC	N	Shared secret	N
Feng et al. [76]	Embedded	SC	MT	MAC	Version
Dhobi et al. [67]	Embedded	SC	MAC	DS	Nonce
He et al. [68]	IoT	SC	BC + HF	BC	Nonce + Timestamp
Hu et al. [82]	IoT	N	BC + HF	BC + DS	Version
Keleman et al. [75]	Embedded	SC	HF	DS	Nonce
Falas et al. [77]	Embedded	SC	MAC	MAC	Version
Kerliu et al. [63]	WSN	SC	MAC	MAC	N
Falas et al. [78]	Embedded	SC	HF	DS	Version
Siddiqui et al. [79]	FPGA	SC	HF	DS	Nonce
Lo et al. [61]	IoT	N	HF	MAC	Version
Gupta [38]	IoT	SC	BC + HF	BC	Nonce

L*egend* :Y: Yes, N: No, SC: Symmetric Cryptography, HF: Hash Function, HC: Hash Chain, MAC: Message Authentication Code, MT: Merkle Tree, BC: BlockChain, DS: Digital Signature

TABLE 3. Attestation notation summary.

Terminology	Description
$\mathcal{P}$	Prover
$\mathcal{V}$	Verifier
S, $\tilde{S}$	State
c	Challenge
$r_{\mathcal{P}}$	Response computed by $\mathcal{P}$
$r_{\mathcal{V}}$	Response computed by $\mathcal{V}$
δ	Attestation duration
Δ	Attestation time limit
t_c	Challenge sending time
t_r	Response reception time
M	Set of memory regions
m	Memory region
$Attest(S, c)$	Compute the response
$Challenge()$	Generate a challenge
$Time()$	Get the current time
$P(X)$	Probability of X
ϵ	Infinitesimal quantity

Unclonable Function (PUF) in the calculation of a checksum. We can also cite SWATT [87] which relies on a careful design of the attestation procedure so that it is optimal for a specific hardware configuration.

The adversary has full software access. Many attestation protocols are designed for the context of embedded systems (IoT, WSNs, etc.). In this context, it is common that devices do not have tamper-resistant hardware to protect the software at the hardware level. This is why it is frequently assumed that an attacker has full access to the software (access to read and write).

The adversary has full control over the prover's communication. This is a direct consequence of the previous assumption. Indeed, if attackers have full access to the software, they can use the device's communication means and therefore intercept, inject or replay traffic from / to the network.

The adversary cannot modify the prover's hardware. It is generally assumed that the attacker cannot modify the prover's hardware. Attacks consisting of replacing the processor with a faster one or adding extra memory in order to store malicious code are therefore considered out of scope. Side-channel attacks are also sometimes considered out of scope [88], [89]. However, [32] allows the modification of the hardware which is outside the System-on-Chip (SoC). The SoC itself is considered safe because of the complexity and cost of this kind of attacks.

2) CHALLENGES

An attestation mechanism design must address several challenges. Indeed, according to [15], $Challenge()$ and $Attest(S, c)$ functions must follow specific design rules. The $Challenge()$ procedure must ensure the three following design principles:

1) **Authenticity**: $\mathcal{P}$ can verify that the challenge was issued by $\mathcal{V}$.

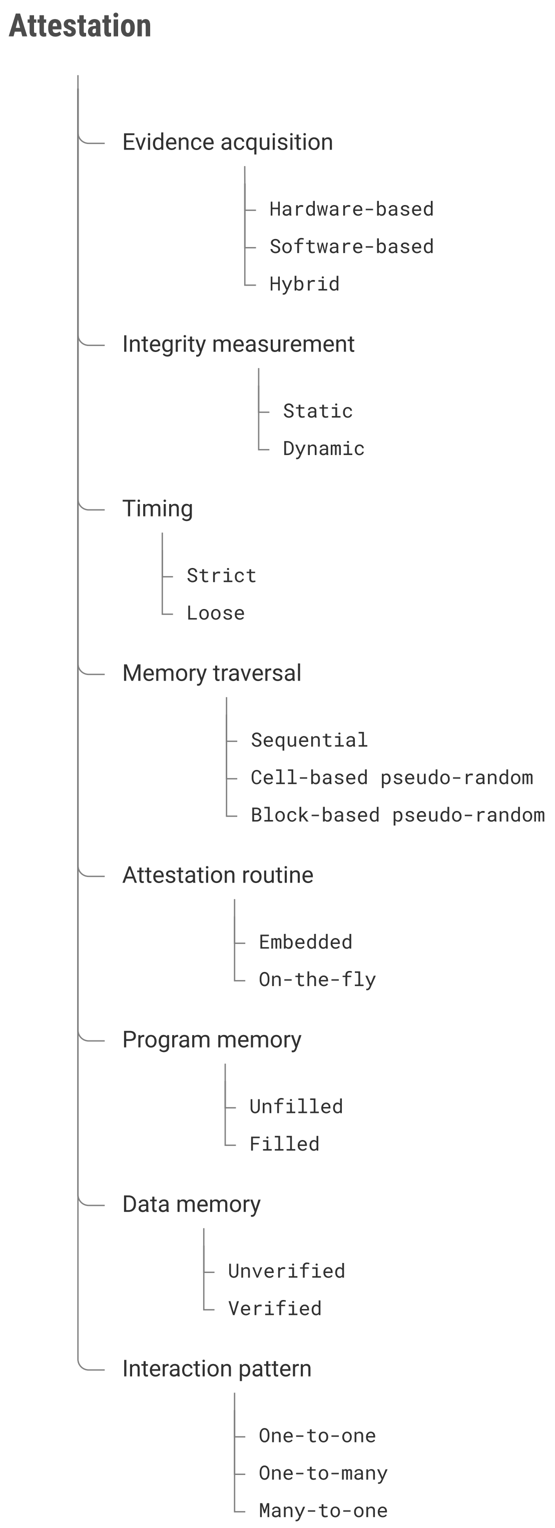


FIGURE 6. **A taxonomy of attestation mechanisms.**

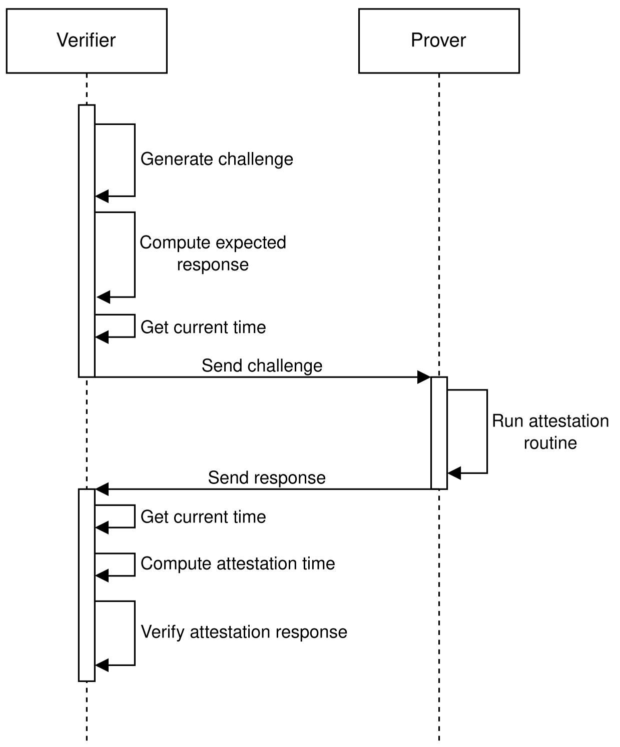


FIGURE 7. **Overview of an attestation process.**

$$
\begin{array}{rcl}
\mathcal{V} & : & c \leftarrow Challenge() \\
\mathcal{V} & : & r_{\mathcal{V}} = Attest(S, c) \\
\mathcal{V} & : & t_c \leftarrow Time() \\
\mathcal{V} \rightarrow \mathcal{P} & : & c \\
\mathcal{P} & : & r_{\mathcal{P}} \leftarrow Attest(S, c) \\
\mathcal{P} \rightarrow \mathcal{V} & : & r_{\mathcal{P}} \\
\mathcal{V} & : & t_r \leftarrow Time() \\
\mathcal{V} & : & \delta \leftarrow t_r - t_c \\
\mathcal{V} & : & (r_{\mathcal{V}} = r_{\mathcal{P}}) \wedge (\delta < \Delta)
\end{array}
$$

FIGURE 8. **A generic attestation procedure.**

2) **Freshness**: $\mathcal{P}$ can know when the challenge was generated and can therefore ignore old ones.
3) **Unpredictability**: An attacker must not be able to predict the challenge which will be generated by $\mathcal{V}$ even with the knowledge of all the previous challenges.

The first two principles are essential to prevent an attacker from conducting a Denial of Service (DoS) attack by forcing $Attest(S, c)$ to loop. The third principle is important to prevent an attacker from computing the result of $Attest(S, c)$ in advance. The attestation routine itself must respect the following five design rules:

1) **Authenticity**: $\mathcal{V}$ can verify that $r_{\mathcal{P}}$ comes from $\mathcal{P}$.
2) **Atomicity**: The execution of the attestation routine must not be interrupted. Otherwise, an attacker may modify the state S of $\mathcal{P}$ while the attestation routine is running and therefore may fake a good state.
3) **Unforgeability**: An attacker must not be able to compute $r_{\mathcal{P}}$. If the approach considered uses optimal implementation and thus strict timing constraints then the

attacker must not be able to compute $r_{\mathcal{P}}$ faster than $Attest(S, c)$.

4) **Dynamicity**: $r_{\mathcal{P}}$ should reflect the current running state of $\mathcal{P}$.
5) **Determinism**: $\mathcal{V}$ must be able to compute $r_{\mathcal{V}}$ such as, for an infinitesimal ϵ:

$$P(\exists S \neq \tilde{S}, Attest(S, c) = Attest(\tilde{S}, c)) \leq \epsilon$$

3) ADVERSARY MODEL

The objective of an attacker is to compromise $\mathcal{P}$ without being detected by the attestation procedure. The different types of attackers have already been studied by [20]. The attacks which can be carried out against an attestation protocol have been described in detail in the context of WSN and IoT respectively by [15] and [21]. However, we note that communicating with a faster machine than the client to compute $r_{\mathcal{P}}$ (this attack is called a proxy attack) is considered as being out of scope by some of the works which we study [48], [90], [91].

B. TAXONOMY

We have chosen to build our taxonomy upon common properties of remote attestation mechanisms we have derived from approaches we have reviewed. These properties are the following:

- **Evidence acquisition**: Is the protection software-based, hardware-based or hybrid?
- **Integrity measurement**: Does the approach check runtime properties of the device or not?
- **Timing**: Is the tolerance for elapsed time calculation strict or loose?
- **Attestation routine**: Is the attestation routine embedded or generated on-the-fly?
- **Memory traversal, program and data memories**: What are the attested memory regions and how is the memory traversal made?
- **Interaction pattern**: How do communicate the prover(s) and the verifier?

These properties are common to all remote attestation approaches we have studied. That's why we think that is it relevant to build our taxonomy upon them. Thus, every remote attestation mechanism should fit on the proposed taxonomy.

1) EVIDENCE ACQUISITION

In the literature, we have identified three methods used to acquire the evidence of the prover's integrity: software-based, hardware-based and hybrid.

Software-based methods rely on the attestation routine to produce an unforgeable result. To ensure a sufficient evidence level, software-based approaches use several techniques. First, the evidence can rely on the optimality of the attestation procedure [47], [48], [87], [91], [92]. Any modification of the procedure by an attacker will thus be detected by the verifier due to the higher than usual execution time. We discuss this point in detail in the section IV-B3. Then, other approaches propose to use an on-the-fly generated attestation routine [29], [92], [93]. These approaches are based on the assumption that the attacker will not have time to analyze the attestation routine in the allotted time since the routine is random and unpredictable by the attacker. Finally, several approaches propose to fill the unused memory with incompressible randomness and then to browse the memory to be attested in a pseudo-random fashion [42], [84], [90], [94].

The hardware-based approaches require dedicated security hardware that can in some cases be tamper-resistant. About tamper-resistant hardware, we can cite approaches using a Trusted Platform Module (TPM) [85], [95], [96] or a PUF [86], [97]. We have also seen approaches using a custom security module. These security modules can be either standalone modules [89], [98], [99] or a hardware extension to an existing processor [100], [101]. In the first case, a communication channel between the processor and the security module must be implemented. For example, the approach of Nunes et al. [98] uses Direct Memory Access (DMA) to exchange information between the two chips. The second method has the advantage to not expose this data bus, since it is integrated in a single chip. However, this second method is only applicable to processors with open hardware. Finally, the approach of Rawat et al. [102] mentions using a ''secure hardware space'' without specifying its nature (off-the-shelf or custom hardware). We have thus chosen to classify this approach as ''hardware-based'' on the assumption that the required hardware is custom. Hardware-based approaches increase the level of privileges needed by an attacker to compromise the system. These approaches are therefore able to protect the system against an attacker who has complete software control. However, the use of dedicated hardware has drawbacks such as higher costs and additional power consumption.

Finally, the hybrid approaches rely on non-custom and non-tamper resistant hardware modules. The hardware required by these approaches is most of the time a read-only memory (ROM) [32], [33], [103], [104]. The approach of Carpent et al. [33] additionally requires a Reliable Read-Only Clock (RROC) or a secure monotonic counter to authenticate and detect replayed or reordered verifier's attestation requests. Hybrid approaches offer a trade-off between hardware-based and software-based approaches in terms of cost, ease of deployment and security.

2) INTEGRITY MEASUREMENT

Attested memories can contain static data (ROM, program memories) or dynamic data (data memories, CPU registers, etc.). Static data attestation is the most common, but it does not allow the detection of attacks that rely solely on the modification of dynamic data: stack overflow or Return-Oriented Programming (ROP), for example.

In order to address these issues, some approaches attest dynamic data. First, we have encountered approaches that attest all memory regions containing dynamic data [47], [88], [94], [101]. Second, the approach of Garay et al. [92] attests only the program control locations such as the program stack, which allows to simplify the attestation. Indeed, with this method the verifier only needs to know some key structures of the program execution and not the whole content of data memories. Third, some approaches [47], [48], [88], [91], [95] attest only some registers of the CPU reducing again the knowledge required by the verifier. The attested registers contain addressing information (program pointer or data segment pointer for example) and therefore give information about the execution of the program at runtime. For example, an attacker wishing to hijack the execution of the program, to execute a function that should not be executed for example, will necessarily modify this kind of registers. However, there is a direct relationship between the verifier's knowledge and the security of the system. Indeed, the less the verifier is knowledgeable about the prover, the more the attacker's leeway is increased since the amount of attested data is reduced.

The attestation of both static and dynamic memories is crucial in order to ensure on the one hand that the program has not been modified and on the other hand that the program executes as it was mentioned to be. However, if the attestation of static data is straightforward, it is not the same for dynamic data since the verifier must know all the known good states of the prover's dynamic data. Indeed, the number of possible states grows exponentially with the number of possible branches in a program.

It should be noticed that this category is not the most relevant for the certification of the bitstream of an Field-Programmable Gate Array (FPGA). Indeed, the bitstream is not a program that would be executed sequentially by a processor at runtime but a configuration that is read and interpreted at power-up and that is not used afterwards. A bitstream is therefore static in essence and does not have dynamic elements such as the stack or the heap. This is why all the solutions attesting the bitstream of an FPGA will necessarily be classified as ''static''.

3) TIMING

No matter how remote attestation is implemented, the verifier only waits for the prover's response for a defined time. Some approaches use a strict timing condition while others use a looser one.

Approaches based on strict timing can avoid dedicated hardware since security is founded on the optimality of the attestation procedure. We note that most of them are software based [47], [48], [87], [91], [92]. However, this method brings up several problems that need to be discussed. First of all, producing an optimal attestation procedure is difficult and is specific to the considered architecture. It is therefore challenging to use this method in a system involving heterogeneous architectures. Moreover, this method is vulnerable to proxy attacks: an attacker may use a machine with higher computational capabilities than the prover in order to compute the attestation response faster than the latter. This allows the attacker to respond to the verifier within the time limit despite a non-optimal attestation routine. Finally, network latency must be carefully evaluated to avoid false positives and false negatives. However, the approach of Schellekens et al. [95] solves this problem by using a TPM. The TPM is used to generate and sign two timestamps on the prover side: one before the checksum execution and one just after the checksum execution. These two timestamps and their signature are sent to the verifier which thus has a proof of the execution time of the checksum procedure on the prover side. The latency of the network does not intervene in the calculation of the elapsed time, solving in fact this issue.

Approaches based on loose timing rely on the secrecy of keys used by cryptographic functions. The security level of these approaches is therefore inherent to the security of the cryptographic material storage. In this regard, several methods are used in the scientific literature. First of all, a ROM can be used [32], [88], [98], [100], [101], [103]. Note that in this case other mechanisms must be implemented to ensure the confidentiality of the cryptographic material if this is required. Other approaches use a dedicated security module, whether it is a TPM [85], [96] or another security module [89], [99], [102]. Vliegen et al. [97] use PUFs to generate the key used by their attestation protocol. Finally, the solution proposed by Shaneck et al. [93] uses a decryption code for the attestation routine. The key used by the decryption code is hidden in the memory of the device. Obfuscation techniques are also used to make the recovery of the cryptographic material even more complex. Therefore, the security of this approach relies only on the obfuscation of the decryption code and thus on security by obscurity. In other words, this approach is secure only as long as the attacker is unable to find the decryption key in memory.

4) MEMORY TRAVERSAL

The attestation procedures considered in this survey use the contents of the prover's memory as evidence of its trustworthiness. These procedures can then browse memory in two ways: sequentially or pseudo-randomly.

Approaches using a sequential memory traversal (see the column ''Memory traversal'' of Table 4) go through the memory of the prover in an iterative manner from start to end. This method allows for the traversal of the whole attested memory and has a linear time complexity as a function of the memory size. However, a sequential traversal is predictable by an attacker. This is why this method does not detect malicious code that would be able to relocate itself over time (called a roving malware). Indeed, when the attestation routine starts, the malware can be relocated to the end of the memory and moved back after the attestation routine passes through the original malware's location, thus effectively avoiding detection.

In order to protect against this attack, approaches using a pseudo-random traversal have been developed. Pseudo-random traversal techniques can be divided into two categories: cell-based pseudo-random and block-based pseudo-random. A cell-based traversal attests each address individually while a block-based traversal divides the memory into one or more blocks. Each block is attested randomly and then the addresses constituting the same block are attested sequentially. There are two important issues with pseudo-random traversals.

First, a particular attention must be given to the selection of the random number generation algorithm, otherwise the security level of the protection could be jeopardized [105], [106], [107]. We have identified several methods which are used in the scientific literature. A stream-cipher algorithm such as RC4 [31], [40], [48], [87], [93], [95] or RC5 [42], [84] is for example frequently used. The approach of Carpent et al. [33] uses a block-cipher in counter (CTR) mode. The approaches proposed by Li et al. [90] and Horsch et al. [91] are based on simple algorithms suited for devices with low computational capabilities and whose optimality may be important. Finally, the approach of Tan et al. [96] uses nonces sent by the verifier as pseudo-random numbers. Note that, in all of the above cases, the random number generator must be seeded with a value unknown to the attacker, otherwise the latter might be able to pre-compute the output of said generators.

Second, due to their random nature, these traversal techniques are probabilistic instead of being deterministic. Indeed, there is no certainty that each address will be attested at least once. This is why, in order to ensure with a high probability that each address will be attested at least once, these approaches are based on the result of the Coupon Collector's Problem [108]. This problem states that for a memory of size n it is necessary to perform $\mathcal{O}(n \ln n)$ memory access. In fact, some addresses end up being read several times, leading to an unnecessary overhead that can be reduced by a block-based traversal. Indeed, for a block of size b, only $\mathcal{O}(\frac{n \ln n}{b})$ memory accesses are necessary to attest each address at least once with high probability. It is therefore a trade-off between performance and security that must be considered for each implementation.

5) ATTESTATION ROUTINE

Most existing approaches have their attestation routine embedded in the prover's memory. However, a verifier can also generate an attestation routine each time it wishes to attest the prover. In the first case the attestation routine must be free of vulnerabilities and can be known by the attacker without compromising the security of the approach. In the second case, the security of the system is based on the assumption that attackers cannot guess in advance the instructions constituting the attestation routine and that they will therefore not have the time to analyze it in order to forge a valid response. To make attacker's task harder, various techniques are used in the literature. For example, the approach of Shaneck et al. [93] encrypts the attestation routine using a shared key between the prover and the verifier. The authors then add a decryption routine along the encrypted attestation routine. This decryption routine retrieves the key which can be hidden in the prover's memory or in the decryption routine itself. The program is also obfuscated and contains junk instructions, self-modifying code and uses opaque predicates in order to make the analysis more complex, whether it is automatic or manual.

We can also take the example of the solution proposed by Garay et al. [92] which is mainly built upon a technique called "program blinding". This technique consists in combining a small hand-written program with another randomly generated one so that the hand-written program keeps its properties but its output becomes difficult to predict. Moreover, the authors use a large set of possible randomly generated programs (called "agent" in the paper) in order to reduce the probability that an attacker in possession of all the generated agents could find by chance the one that will be used. Finally, the authors take care to generate agents with an irreducible Control-Flow Graph (CFG) so that any attempt to simplify the program will fail.

Finally, the approach of Park and al. [29] uses a randomly generated hash function. When the verifier wishes to attest the prover, a hash function is randomly generated and sent to the prover. The latter hashes the memory to be attested with this hash function and then returns the result to the verifier.

6) PROGRAM AND DATA MEMORIES

Program memory is not always used to its full capacity. Some memory areas can be vacant and thus be used by an attacker. In order to prevent this scenario, some solutions propose to fill this unused memory with random data [31], [42], [84], [90], [94], [96]. However, despite this, it is still possible for an attacker to compress the program memory in order to free up memory space. The latter contains mostly instructions that have low entropy and can therefore be compressed. This is why Vetter et al. [104] propose, in addition of filling the memory with random noise, to use a specific program memory able to store compressed instructions and to decompress them on-the-fly. As a result, the code becomes incompressible and an attacker is no longer able to free up memory space.

Depending on the architecture considered, the data memory can be shared with (von Neumann architecture) or separated from (Harvard architecture) the program memory. In both cases an attestation routine which does not check the data memory does not detect attacks which only modify this memory such as stack overflow or ROP. This is why some approaches also check the data memory [47], [88], [90], [92], [94], [101]. In this case the verifier must know not only the content of the program memory but also the content of the data memory which can be difficult as the program become more complex and the number of possible states increases. We also note that the approaches we have identified are designed for embedded systems or devices whose program remains simple enough so that the verifier can know the possible states of the data memory.

In the context of FPGAs, talking about data memory does not make sense. Indeed, the FPGA's bitstream is not a program running at runtime and as such does not have a data memory. The bitstream is used only once to configure the FPGA at the hardware level and then the memory containing this configuration data is not used until the next power cycle. The memory containing the bitstream can be seen as the program memory even if it is sometimes called differently. For example, Vliegen et al. [97] call this memory ''configuration memory''. In the specific case of their approach which uses a technique called ''partial reconfiguration'', the program memory can be partially rewritten but it is not always the case. The bitstream is indeed most of the time stored in an immutable memory from the point of view of the FPGA (the FPGA does not have the ability to write on this memory).

We have also studied works that do not make the distinction between program memory and data memory. For example, the ERASMUS project [101] mentions a memory area named ''RAM / Flash'' which seems to contain code and data. The SMATT project [40] mentions ''memory cells of the smart meter'' without specifying if these are memory areas containing code, data or both. That is why we cannot know if the data memory is checked or not. Similarly, the works of Schulz et al. [86], Carpent et al. [33] and Nunes et al. [98] do not specify the content of the attested memory areas.

7) INTERACTION PATTERN

The majority of the approaches encountered are based on ''one-to-one'' communication. A verifier communicates with a single prover in order to perform the attestation of the latter. However, this method has a linear time complexity with respect to the number of provers to attest. For example, Li et al. [47] use a ''one-to-one'' communication in order to attest several devices of a system. To do so, they attest each device sequentially, effectively leading to an $\mathcal{O}(n)$ complexity.

This is why a ''one-to-many'' communication can be used when many provers have to be attested. This technique takes advantage of the parallelization of the attestation routine execution to save time when attesting the complete system. We have seen three methods of parallelization in the literature. First, the approaches of Jin et al. [31], Asokan et al. [103], Tan et al. [96] and Ammar et al. [88] propose to do attestation chaining. This method consists in attesting hierarchically the provers of the system. Some provers thus play the role of verifier for the provers hierarchically below them. This hierarchy can be established according to different criteria depending on the application context: geographical proximity of the provers to the verifier or presence of a hardware module on the device for example. Then, the approach of Park et al. [40] randomly selects a fixed number of identical provers and sends to all of them the same challenge. All provers being strictly identical and having the same challenge, the answer received by the verifier must be the same regardless of the prover. The overhead is thus reduced since it becomes unnecessary to perform n times the checksum calculation. Finally, the approach proposed by Rawat et al. [102] transmits the attestation request to all provers at the same time. Each provers then uses its secret key to compute the answer that will be transmitted to the verifier. However, all these methods have the drawback of having a single point of failure: the verifier. Indeed, the verifier is unique and must be considered secure. Moreover, the chain attestation has the disadvantage of not allowing to attest finely the system. Indeed, as soon as a malicious node is attested, all the nodes that are hierarchically below it will also be considered malicious even if this is not the case.

In order to address the single point of failure issue, a ''many-to-one'' communication can be used. This method is introduced by Yang et al. [84] and allows to get rid of a trusted verifier. The authors present two distributed software-based attestation schemes that rely only on ''regular'' nodes and thus do not need a verifier. However, a drawback of this method is that the network must be dense. Indeed, if the network is not dense, attackers could take control of a sufficient number of nodes and could therefore compromise the consensus within the network. They would then be able to make healthy nodes look like corrupted nodes and vice versa.

C. DISCUSSION

In this section we extended Steiner et al. [15] classification of attestation approaches used in the context of WSNs to a new classification of existing solutions regardless the nature of the system. We have therefore studied approaches that they have not studied and we addressed these solutions from a new point of view. Furthermore, we have outlined some limitations we have encountered when generalizing their taxonomy. The Table 4 maps, to the taxonomy shown in Figure 6, the attestation mechanisms that we considered relevant for solving our problem. The table is sorted in chronological order.

V. SECURE BOOT

In this section, we first formally describe secure boot mechanisms. Then, we review the main characteristics of existing secure boot approaches and we propose a new taxonomy (shown in Figure 9) which identifies five major characteristics. We discuss each identified characteristic in more detail below.

We have split the projects from scientific literature and proprietary projects. We define a proprietary project as a project owned by a private company. We think that it is relevant to distinguish between scientific and private approaches. Indeed, private projects are mostly based on opaque software and/or hardware blocks as opposed to approaches published in the scientific literature. That is why we would need to reverse engineer these private projects if we would like to uncover their secrets. However, we do not have the right to carry out this work due to intellectual property protection laws. We therefore consider as true the assertions made in the technical documentation published by the manufacturers.

TABLE 4. Mapping of some existing attestation mechanisms to the proposed taxonomy.

Approach	Application	Evidence acquisition	Integrity measurement	Timing	Memory traversal	Attestation routine	Program memory	Data memory	Interaction pattern
SWATT [87]	Embedded	Software based	Static	Strict	CPR	Embedded	Unfilled	Unverified	One-to-one
PIV [29]	WSN	Software based	Static	Loose	SEQ	On-the-fly	Unfilled	Unverified	One-to-one
Shaneck et al. [93]	WSN	Software based	Static	Loose	CPR	On-the-fly	Unfilled	Unverified	One-to-one
Pioneer [48]	GPC	Software based	Dynamic	Strict	CPR	Embedded	Unfilled	Unverified	One-to-one
TEAS [92]	Embedded	Software based	Dynamic	Strict	CPR	On-the-fly	Unfilled	Verified	One-to-one
Proactive [94]	WSN	Software based	Dynamic	Loose	SEQ	Embedded	Filled	Verified	One-to-one
Distributed [84]	WSN	Software based	Static	Loose	BPR	Embedded	Filled	Unverified	Many-to-one
Schellekens et al. [95]	GPC	Hardware based	Dynamic	Strict	CPR	Embedded	Unfilled	Unverified	One-to-one
AbuHmed et al. [42]	WSN	Software based	Static	Loose	BPR	Embedded	Filled	Unverified	One-to-one
SBAP [90]	Embedded	Software based	Static	Loose	CPR	Embedded	Filled	Verified	One-to-one
USAS [31]	WSN	Software based	Static	Loose	CPR	Embedded	Filled	Unverified	One-to-many
Lightweight [86]	Embedded	Hardware based	Static	Loose	BPR	Embedded	Unfilled	-	One-to-one
VIPER [47]	GPC	Software based	Dynamic	Strict	CPR	Embedded	Unfilled	Verified	One-to-one
SMART [100]	Embedded	Hardware based	Static	Loose	SEQ	Embedded	Unfilled	Unverified	One-to-one
Compressed [104]	WSN	Hybrid	Static	Loose	CPR	Embedded	Filled	Unverified	One-to-one
SMATT [40]	Smart grid	Software based	Static	Loose	BPR	Embedded	Unfilled	-	One-to-many
SobTrA [91]	Embedded	Software based	Dynamic	Strict	CPR	Embedded	Unfilled	Unverified	One-to-one
Tan et al. [85]	WSN	Hardware based	Static	Loose	SEQ	Embedded	Unfilled	Unverified	One-to-one
SEDA [103]	Embedded	Hybrid	Static	Loose	SEQ	Embedded	Unfilled	Unverified	One-to-many
ERASMUS [101]	IoT	Hardware based	Dynamic	Loose	SEQ	Embedded	Unfilled	Verified	One-to-one
Schulz et al. [32]	IoT	Hybrid	Static	Loose	SEQ	Embedded	Unfilled	Unverified	One-to-one
SMARM [33]	IoT	Hybrid	Static	Loose	BPR	Embedded	Unfilled	-	One-to-one
MTRA [96]	IoT	Hardware based	Static	Loose	BPR	Embedded	Filled	Unverified	One-to-many
Vliegen et al. [97]	FPGA	Hardware based	Static	Loose	SEQ	Embedded	Unfilled	-	One-to-one
VRASED [98]	Embedded	Hardware based	Static	Loose	SEQ	Embedded	Unfilled	-	One-to-one
SIMPLE [88]	IoT	Software based	Dynamic	Loose	SEQ	Embedded	Unfilled	Verified	One-to-many
SHeFU [99]	IoT	Hardware based	Static	Loose	SEQ	Embedded	Unfilled	Unverified	One-to-one
Rawat et al. [102]	Vehicle	Hardware based	Static	Loose	SEQ	Embedded	Unfilled	Unverified	One-to-many
SRACARE [89]	Embedded	Hardware based	Static	Loose	SEQ	Embedded	Unfilled	Unverified	One-to-one

L*egend* :-: No information or irrelevant, SEQ: Sequential, CPR: Cell-based Pseudo-Random, BPR: Block-based Pseudo-Random
GPC: General-Purpose Computers

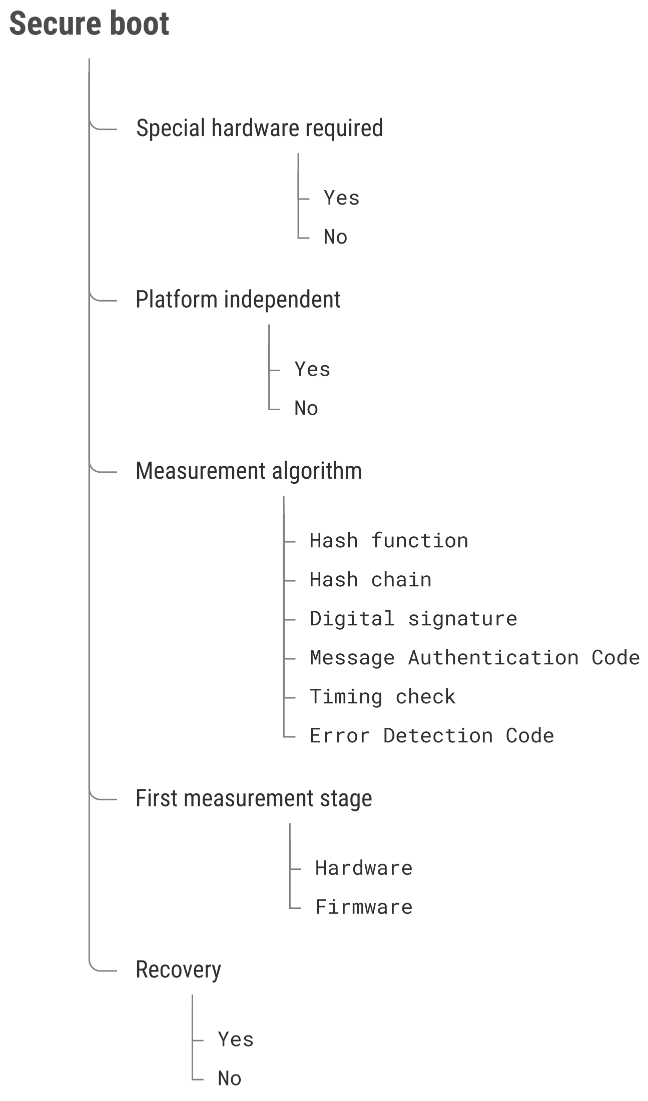


FIGURE 9. **A taxonomy of secure boot mechanisms.**

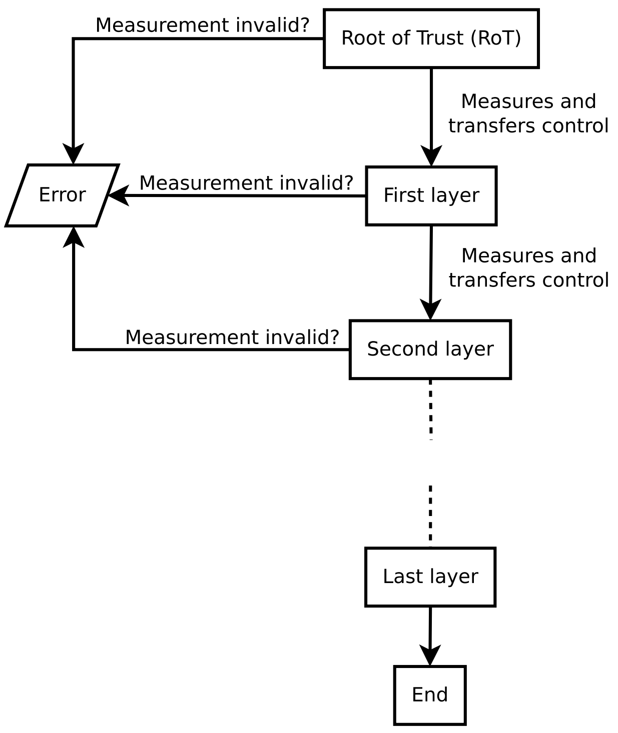


FIGURE 10. **Overview of a Chain of Trust (CoT).**

A. BACKGROUND

In the context of this survey, we consider that the security goal of secure boot is to verify the integrity of the firmware as described in the UEFI specifications [54]. This definition of secure boot is widely accepted in the security community [55], [56]. The literature overflows with various names such as ''trusted boot'', ''measured boot'', ''verified boot'' or ''authenticated boot''. For the sake of clarity, we will group all these names together under the name ''secure boot''.

Secure boot uses the principles of Root of Trust (RoT) and Chain of Trust (CoT) (see Figure 10). A RoT is the first element of a CoT. It is the element which is considered as *de facto* secure. In the case of secure boot this element can be a piece of software like a first stage bootloader [109], [110] or a hardware module like a TPM [46], [50], [111], [112], a smartcard [112], [113] or another hardware module [114], [115]. The next layer is measured by the current one then this measurement is verified. If the verification succeeds then the procedure continues with the next layer until all layers have been measured. Otherwise, the procedure prevents the system from booting.

Formally, let $L = \{L_1, L_2, \cdots, L_N\}$ a set of N layers and $M(L)$ a measurement function taking as argument the layer L to be measured and returning *True* if the measurement is valid, *False* otherwise. In this case, the validity V_i of each layer L_i is defined by the recurrence relation (3). The layer number zero represents the RoT and is therefore considered as *de facto* valid.

Note that the measurement function M, could need a reference value allowing it to the validity of the layer L passed in parameter. This reference value, named golden value, is voluntarily abstracted here since we want to keep the notations generic in order to cover as many cases as possible. We have also done the same for all the cryptographic material that M might need: public / private key, secret key, etc.

$$\begin{cases} V_0 = \top \\ V_{i+1} = V_i \wedge M(L_{i+1}) \end{cases} \tag{3}$$

1) CHALLENGES

A secure boot mechanism should address the following challenges:

- **Integrity**: Prevent an attacker to modify the firmware. In other words, the solution should ensure the bootstrapping chain's integrity.

- **Platform independence**: The solution should be easily implementable on heterogeneous machines from several manufacturers.
- **Resilience**: The solution should provide a way to recover from a firmware modification. The machine should be able to boot even if the firmware has been modified and must boot in a known good state.

2) ADVERSARY MODEL

The adversary's goal is to compromise the integrity of the firmware without being detected by the secure boot mechanism. In other words, the adversary's goal is to execute arbitrary code despite the presence of the protection.

The attacks that the attacker is allowed to perform depend directly on the privilege level of the protection. The protections proposed in the literature use either dedicated hardware (for instance [55], [112], [114], [116]) or a software extension (for instance [46], [109], [117]), see Table 5 for more information. Software level approaches do not protect against an attacker with hardware access to the system. However, even the approaches using some dedicated hardware do not address all hardware attacks. Indeed, [118] does not allow side-channel and fault injection attacks. In the same way, [119], [120] only allows the compromise of memories and buses which are external to the protection, the others are supposedly safe.

B. TAXONOMY

We have chosen to build our taxonomy upon common properties of secure boot mechanisms we have derived from approaches we have reviewed. These properties are the following:

- **Special hardware required**: Does the solution require the use of specific hardware?
- **Platform independent**: Can the solution be easily implemented on other platforms than the one it was designed for?
- **Measurement algorithm**: What are the algorithms used by the measurement method M?
- **First measurement stage**: At which abstraction level is the measurement function M called for the first time?
- **Recovery**: Does the proposed solution have a recovery/backup mechanism in case s firmware corruption is detected?

We think that this categorization is more relevant than basing our taxonomy upon security goals. Indeed, the only security goals we have identified is to ensure the integrity of the boot chain. Moreover, we think that the five characteristics we have identified are common to all secure boot approaches.

1) SPECIAL HARDWARE REQUIRED

The majority of the studied approaches require the use of specific hardware. Our study revealed five different categories of hardware: custom chip, specific processor, fuses, TPM and smartcards.

Custom chips can be open or closed hardware. Closed-hardware chips are used by private projects such as HP Sure Start [121], [122], [123], Intel Boot Guard [124], [125] or the Apple T2 chip [126]. This opacity, although understandable from the point of view of manufacturers' intellectual property protection, is in our opinion bad from the point of view of security since it is similar to security by obscurity. Indeed, approaches relying solely on security by obscurity are only secure as long as the algorithm remains unknown to the attacker. However, the attacker can carry out reverse engineering work in order to discover it. If the attacker succeeds in doing so, the security of the system is completely compromised. That is why some proprietary projects use open-hardware chips or open-source software. We can for instance cite the OpenTitan project [53] whose Verilog hardware description is published in a GitHub repository. Another example is the Olympus project [125], [127] whose specifications, source code and hardware are publicly available. We also found several open-hardware approaches documented in the scientific literature [118], [128], [129].

Usage of specific processor(s) can be explained in several ways. On the one hand, some projects have been designed for a single processor or a single family of processors. It is for example the case of the project of Lebedev et al. [116] which is interested in the protection of platforms using the Sanctum processor. On the other hand, some approaches use functionalities that are only offered by a limited number of processors. This is for example the case of the solution proposed by Gonzalez et al. [43] which uses TrustZone, a technology implemented only on some ARM processors.

Fuses are hardware components that can be written only once and whose value cannot be changed afterwards (once the fuse is "burnt", it cannot be restored to its original state). This technology is used to store critical information about the protection configuration. For example Jiang et al. [115] use a fuse to store cryptographic material, namely the golden values of the system. We can also talk about Samsung Knox [130] which uses a fuse to permanently store warranty expiration information if one of the bootstrap layers is not digitally signed by Samsung. The weakness of this technology is also its strength: the configuration becomes immutable and a new configuration requires to replace the concerned hardware components. If this is still possible for open hardware, it is not the case for closed hardware.

Several approaches rely on TPM. TPM can be used as storage for cryptographic material and only allow its reading under certain conditions as in the approach of Hudson et al. [46]. It can also be used to measure the integrity of the boot chain layers [50], [112], [124]. TPM have the advantage of being factory-integrated on recent general-purpose computers and can be implemented on a large number of others. However, one of the disadvantages of its use is its cost, which can be too high in some situations. This drawback is particularly limiting in the context of low cost embedded systems where the price of a TPM can sometimes be similar to the price of the system itself.

The storage of cryptographic material can also be done thanks to smartcards. The advantage of a smartcard compared to a TPM is its mobility. Indeed, users can carry it with them and use it easily on different workstations, whereas a TPM is fixed, soldered to the system. For example, the approach of Itoi et al. [113] uses a smartcard to verify the boot sequence of a workstation. First, the workstation sends the encrypted hash to be checked to the smartcard. Then. the smartcard decrypts it with its private key and verify it. Then, the smartcard generates the response (pass or fail), signs it with its private key and sends it back to the workstation. Finally, the workstation can, according to the response, continue or abort the bootstrapping procedure. Another example is the approach of Gonzalez et al. [43] which uses a Secure Element (which is a smartcard integrated into a mobile device) in the context of BYOD applications. The authors use the smartcard to verify the validity of the operating system (according to the company's policy). Once the operating system is verified, the user can access to the company's sensitive data (VPN access, public and private keys to authenticate users, etc.). Otherwise, these data are locked into the secure element and the operating system can not access them.

2) PLATFORM INDEPENDENT

We first need to define what we consider to be a "platform independent" approach. We have previously shown that some approaches require specific hardware in order to work. A "platform independent" approach is then defined as an approach that is compatible with almost all the platforms of its field of application despite the possible use of specific hardware blocks. The scientific literature proposes independent platform approaches that we have classified into several categories.

First of all, the approach of Yin et al. [110] is fully software-based. Their solution can therefore be implemented on any platform since no hardware is required and the proposed algorithms do not depend on specificities of a particular architecture. The approach of Mϋller et al. [131] uses a firmware decryption program and a security module implemented in an FPGA to decrypt the firmware itself. The encryption key is generated thanks to a dedicated PUF chip, external to the security module and communicating with the latter via an SPI bus. The authors specify that their solution should be implemented with the PUF into a single chip (ASIC) in order to prevent an attacker from obtaining the PUF response by sniffing the SPI bus. In the same way, the approach of Pocklassery et al. [132] uses the output of a PUF, implemented into a System-on-Chip (SoC), as a decryption key. This approach integrates the PUF and the rest of the security module into the same chip and is therefore not vulnerable to the previous attack. However, in both cases, the PUF is only used to generate a decryption key. It is therefore possible to use their approaches as is and replace the PUF by another key generation mechanism to have a fully software-based approach.

Then, some approaches [119], [120] implement a processor within an FPGA and use a part of its resources in order to implement a security module. Therefore, no additional hardware is needed since the FPGA was already used for the processor. Other approaches [111], [129] also use an FPGA but as a security module on its own. In other words, the FPGA is dedicated to security tasks (boot chain verification, recovery, etc.) and communicates with the processor that executes the firmware. These approaches therefore expose at least one communication bus between the processor and the security module. The data on this bus may be sniffed or altered by an attacker with physical access to the system. This attack is more difficult to do when everything is integrated into one single chip as the attacker would need to probe into on-chip busses. Finally, a drawback of these two methods is the need to consider the FPGA's bitstream safe as an attacker who is able to successfully modify the bitstream is able to modify the behavior of the security module.

3) MEASUREMENT ALGORITHM

Almost all approaches rely on one or more cryptographic functions: hash function, digital signature or MAC. In all cases, one or more pieces of data must be stored in a secure way in order to compare the calculated value to a known good value. The first method consists in calculating the hash of the not yet trusted layer and comparing it with a known good hash [49], [91], [109], [129], [132]. This method is the simplest to implement but requires to store a golden value per layer measured in a read-only memory. In order to reduce the amount of read-only memory required, it is possible to use a MAC instead of a hash function [115], [118], [131]. Thus the only data to be stored securely is the secret key which is the same for every layer. The reference MACs can be stored in a non-secure memory since the attacker is not able to calculate them without the secret key. In this case, the secret key memory must not only be read-only but the confidentiality of the key is also needed. The principle is essentially the same for the digital signature. In this case, the piece of data to be protected is the public key and the reference signatures can be stored in a non-secure memory.

The approaches we studied use different methods to store critical information related to the firmware integrity verification. The first solution is to use a ROM to ensure that this data is not modified by an attacker with full-software access. This ROM is either integrated inside a custom chip, as it is the case for the proprietary projects [52], [121], [122], [123], [126], or added as an external component as in the case of the AEGIS solution [117]. This solution has then been extended to add a smartcard support allowing to store sensitive data in a removable component bound to the user [113]. The approach of Huang et al. [112] also uses this method but uses a USB security module instead of a smartcard. These information can also be stored in fuses which ensures that these data cannot be modified even by an attacker with physical access to the system. The approaches of Devic et al. [119] and Rouget et al. [120] use an FPGA as a security module. The sensitive information are stored in the bitstream which is considered secure. Finally, the approach of Lebedev et al. [116] uses a PUF to generate a key which is then used to encrypt and decrypt the secret key of the device. This device's secret key can then be stored encrypted in an untrusted memory across reboots.

A TPM is also perfectly suited since it is designed for this application. Indeed, a TPM chip has registers designed to securely store hashes, called Platform Configuration Register (PCR). The specificity of these registers is that writing can only be done via an ''extend'' operation. During an ''extend'' operation, the value currently stored in the register is concatenated with the data to be extended and the result is then hashed to create the new value of the PCR. Each new value written in such a register therefore depends on the previous one. The final value of a PCR is thus a chain of hashes of all the pieces of data that have been measured since the TPM was powered up, thus ensuring the integrity of the entire chain if the last hash is equal to the known good value. This method is used by Trammell Hudson [46] and Avani et al. [50]. The main drawbacks of using an hardware TPM module are the usage of dedicated hardware increasing the cost of the final system, the reliance on a TPM chip manufacturer and the compatibility issues that may potentially exist with some platforms. Moreover, the approach of Khalid et al. [128] uses a custom security module that implements TPM's functionalities.

The approach of Li et al. [111] presents an Extended Trusted Platform Module (ETPM). This ETPM has all the functionalities of a TPM and adds some new ones: backup system, symmetric cryptography at hardware level and bus arbitration. The authors also present a secure boot procedure using their ETPM. Their approach is different from those presented in the previous paragraph. Indeed, the latter use the TPM via the firmware so the secure boot is actually managed by the firmware itself while the ETPM manages the secure boot procedure on its own.

The approach of Yin et al. [110] uses an Error Detecting Code (EDC) to ensure the integrity of each layer. The authors propose an implementation based on the CRC-16 algorithm. This approach, although using a similar principle to the one used by hash function based approaches, offers a lower level of security. Indeed, due to the linearity of CRC-16, an attacker is able to modify the firmware without changing the CRC-16 value. We therefore do not recommend to use this method since it offers a too low security level compared to other alternatives.

The approach of Wang et al. [114] is based on execution time calculation of several pieces of firmware code. To do this, the authors modify the firmware by adding hooks to it in order to know the elapsed time for a particular instructions block. A routine then checks that the calculated time corresponds to a golden value (within one epsilon) stored in memory. We have identified several drawbacks to this approach. First, a golden value must be stored in read-only memory for each instructions block we wish to analyze. Second, a profiling operation must be performed before the device is deployed in order to populate the memory with these golden values. Finally, if the entire firmware is not checked, it is technically possible for an attacker to slip through the cracks if it only modifies the parts of the code that are not protected.

4) FIRST MEASUREMENT STAGE

The first measurement stage corresponds to the level of abstraction performing the first measurement of the firmware. According to the approaches studied, we have identified two different stages for the verification: firmware and hardware.

Doing the first measurement at the firmware level has the advantage of requiring no additional hardware but only a modification of the firmware. However, it has the disadvantage of not protecting against an attacker who has complete control over the software in itself. Indeed, at least a software routine initiating the measurement must be considered safe. The attacker can then rewrite this software block to bypass the protection. In order to prevent this attack, these approaches rely on hardware mechanisms.

The first idea is to use a ROM to store the first piece of code which will be executed and which will do the first measurement [50], [91], [114], [116]. The approaches of Adnan et al. [109] and Haj-Yahya et al. [55] store the First Stage BootLoader (FSBL, ie. the first software to be executed by the device) into a small ROM located inside the processor or the security module. The FSBL can then measure the rest of the firmware which is located into a bigger unprotected memory. The AEGIS project [113], [117] uses the same idea but for general-purpose computers. However, general-purpose computers usually don't have a dedicated ROM space to store the FSBL. That is why this approach uses a dedicated ROM card to store the FSBL. The approach of Trammell Hudson [46] also targets general-purpose computers but uses the read-only protected areas mechanism offered by existing EEPROM instead of a dedicated ROM. This approach therefore doesn't need an external hardware component. The approach of Javier et al. [43] also stores the FSBL into a ROM but also uses a Trusted Execution Environment (TEE) and a Secure Element (smartcard) to store critical cryptographic material and software. The FSBL first verifies the integrity of the TEE operating system and then initializes the TEE with it. The TEE then verifies the integrity of the next stage before executing it. In case the next stage is not verified, the user can use its Secure Element to authenticate the system. This mechanism allows for instance a company to certify different operating systems in a bring your own device context.

The approaches proposed by Pocklassery et al. [132] and Müller et al. [131] store the RoT into a ROM and encrypt the rest of the firmware. However, the decryption key is generated by a PUF. The integrity of this key is then checked by the RoT and is then used to decrypt the rest of the firmware. Finally, the approach of Jiang et al. [115] uses not only a ROM to store the RoT but also fuses (which can in fact be seen as a small read-only memory) to store cryptographic material.

Doing the first measurement at the hardware level therefore offers a protection against attackers with full software access as the firmware does not need to be considered secure, only the hardware performing the measurement needs to be. Assuming that the firmware cannot change the behavior of the hardware security (which should never be the case), attackers necessarily need physical access to achieve their goals. This method is used by all proprietary projects (see proprietary projects in Table 5) and by approaches using an FPGA as a hardware security module [111], [118], [119], [128], [129].

5) RECOVERY

A recovery mechanism allows to restore the system to a trusted state in case of accidental or malicious corruption of the firmware. The majority of the approaches that we found do not implement a recovery mechanism. Among those which do, we can distinguish four different ways of implementing recovery.

The first implementation type uses a specific region within the same memory chip as the firmware one. This type of implementation does not require a specific hardware module since the existing hardware can be used as long as the memory has enough capacity to store both images. The main disadvantage is that an attacker with full software access could access the backup in addition to having access to the firmware. This method is used by the Olympus project [125], [127] and by the work of Yin et al. [110].

The second method consists in using a dedicated security hardware module and storing the backup in an additional memory located outside this module. This method has the advantage of physically isolating the backup from the firmware. Thus an attacker, even with full software access, will not be able to compromise the backup. However, this method does not offer protection against an attacker with physical access to the device. This idea is used by the approaches proposed by Webel et al. [52], Li et al. [111] and by the OpenTitan project [53].

The third method is similar to the previous one except that the backup memory is located inside the security hardware module. This technique is, in our opinion, the most robust from a security point of view because it also offers some protection against an attacker with physical access to the system. We note that this method is the most used among the approaches that we study [118], [121], [122], [123].

A last method, proposed by Arbaugh et al. [117] consists in externalizing the backup on a server and downloading it directly from it. This method has the advantage of not requiring any additional hardware since the software RoT can use the network hardware already available on the device to communicate with the server. The main drawback however is the inclusion in the Trusted Computing Base (TCB) of all the items needed to download the backup and the assumption that the server is secure.

C. DISCUSSION

In this section we presented a taxonomy and detailed the techniques encountered for each taxon. Each technique was then illustrated with examples. We also discussed the advantages and inconvenient of these techniques. Table 5 maps, in chronological order, a representative number of secure boot mechanisms to the taxonomy illustrated in Figure 9. To the best of our knowledge, every secure boot mechanism would fit on the proposed taxonomy.

VI. DISCUSSION: OPEN RESEARCH PROBLEMS

In this section, we first discuss the applicability of studied approaches. Then, we study open research problems and give directions on how to address them. We will first look at known attacks that can defeat some of the proposed protections. We will then study the potential contribution of new technologies to the resolution of the firmware integrity protection issue. Finally, we will then discuss mixing several protection mechanisms in a single solution.

A. APPLICABILITY

Throughout our study, we have identified some solutions based on mechanisms specific to one or more contexts. We will detail these solutions below and explain why they cannot be implemented for all contexts.

First of all, the solution proposed by Katzir and Schwartzman [26] is very specific. It is based on the principle of ‘‘window of opportunity’’ and require the frequency of the electrical network to change in order to update a device’s firmware. However, this solution relies on two prerequisites: 1. the device to be programmed is on the electrical network and 2. It is possible to vary the frequency of the said electrical network. However, this is only realistic in the context of an electrical network operators wishing to update devices on their own network. It is therefore very difficult to transpose this solution to another context.

Then, some attestation solutions are based on a one-to-many or many-to-one interaction pattern (see sub-subsection IV-B7). These solutions can only be used in the context of a system with several devices. These solutions are thus particularly suited for WSNs and for the IoT world but cannot be implemented when the considered system is made of a single device. This is typically the case for solutions targeting general-purpose computers. These solutions consider a system made up of a single machine and therefore cannot use the interaction patterns explained in this paragraph.

Then, we can notice that the studied approaches can be divided into two main categories: software solutions and hardware solutions. A software solution has the advantage of being less expensive since no hardware is required. It is also (in theory) compatible with any platform. However, these solutions cannot protect against an attacker with physical access to the system. In this case, only hardware solutions

TABLE 5. Mapping of some existing secure boot mechanisms to the proposed taxonomy.

	Approach	Application	Special hardware required	PI	MA	FMS	R
Scientific literature	**AEGIS** [117]	GPC	PROM board	N	DS	F	Y
	sAEGIS [113]	GPC	Smartcard	N	DS	F	N
	Devic et al. [119]	FPGA		Y	DS	H	N
	Adnan et al. [109]	Embedded		N	HF	F	N
	VMFA [110]	Embedded		Y	EDC	F	Y
	Reliable [111]	Embedded	FPGA + TPM	Y	Unknown	H	Y
	Khalid et al. [128]	Embedded	FPGA	Y	HF	H	N
	SobTrA [91]	Embedded		N	HF	F	N
	IOCheck [49]	GPC (peripherals)		N	HF	F	N
	Gonzalez et al. [43]	Mobile	Secure Element	N	DS	F	N
	SecBoot [120]	FPGA		Y	DS	F	N
	ConFirm [114]	Embedded	High Performance Counter (HPC)	N	Timing check	F	N
	Huang et al. [112]	GPC	TPM + USB smart card	N	DS	F	N
	Hudson et al. [46]	GPC	TPM	N	HC	F	N
	Jiang et al. [115]	Embedded	Fuse	N	DS + MAC	F	N
	Self-authenticating [132]	FPGA		Y	HF	F	N
	Müller et al. [131]	Embedded		Y	MAC	F	N
	Sanctum [116]	Embedded	Sanctum processor [133]	N	DS	F	N
	Haj-Yahya et al. [55]	Embedded	Code Authentication Unit (CAU)	N	DS	F	N
	Streit et al. [129]	Embedded	FPGA	Y	HF	H	N
	CARE [118]	Embedded	FPGA	N	MAC	H	Y
	SEDAT [50]	GPC	TPM	N	HC	F	N
Proprietary projects	**z15** [52]	GPC	IBM z15	N	DS	H	Y
	HP Sure Start [121]–[123]	GPC	Custom chip + Intel processor	N	DS	H	Y
	Intel Boot Guard [124], [125]	GPC	Custom chip + TPM + Intel processor	N	DS	H	N
	Apple T2 [126]	GPC	Custom chip	N	DS	H	N
	Samsung Knox [130]	Mobile	ARM processor + Fuse	N	DS	H	N
	Cerberus / Olympus [125], [127]	GPC (servers)	Custom chip + TPM + Fuse	N	DS	H	Y
	OpenTitan [53]	GPC (servers)	Custom chip + Fuse	N	DS	H	Y

H*eaderlegend* :PI: Platform Independent, MA: Measurement Algorithm, FMS: First Measurement Stage, R: Recovery

V*alueslegend* : Y: Yes, N: No, S: Static, D: Dynamic, H: Hardware, F: Firmware, HF: Hash Function, DS: Digital Signature, MAC: Message Authentication Code, HC: Hash Chain, EDC: Error Detecting Code, GPC: General-Purpose Computers

can be helpful. They are however more expensive and are sometimes incompatible with certain platforms. In fact, we have identified limitations in terms of compatibility regarding the hardware required by some approaches. For instance, the approach of Lebedev et al. [116] was designed specifically to be integrated within the Sanctum processor. Then, the solution of Gonzalez et al. [43] uses TrustZone, a technology implemented only on some ARM processors. Finally, Intel Boot Guard [124], [125] requires an Intel processor
to work.

These limitations also applies for proprietary solutions using closed hardware. This type of hardware is, by definition, a black box that we are forced to trust. This is similar to security by obscurity and is therefore opposed to the Kerckhoffs' principle [134], which is detrimental to the security of the system. Moreover, some fuses are sometimes burned in the production phase in order to pre-configure the protection. This process imposes to the user all or part of the configuration and can be problematic in case of firmware customization. Finally, this causes a problem in the event of a proven vulnerability at the hardware level, since it is impossible for the user to make any modification to the hardware or to the configuration. Users are therefore forced to rely on the product recall by the manufacturer in order to keep their system protected.

Finally, solutions based on hardware components add a production cost for each device. This additional cost may or may not be significant and prevent the implementation of a solution depending on the context considered. Indeed, a sensor or an IoT device are constrained in terms of manufacturing cost (see section II). Thus, adding hardware components to protect their firmware may be feasible in theory but impossible to implement due to the increased cost of the final product.

B. OPEN RESEARCH PROBLEMS

In this subsection we will pursue our specific analysis by focusing on the state-of-the-art shortcomings and potential threats associated to the proposed solutions.

1) KNOWN ATTACKS

We will see existing attacks against some of the mechanisms discussed earlier. These attacks show that some of the protection mechanisms discussed in this survey are vulnerable and should not be used for future projects unless the said attacks couldn't happen in the considered context. We will first discuss the time-of-check to time-of-use (TOCTOU) bugs. We will then discuss TPMs, the keystone of many approaches, yet vulnerable to several attacks.

The name "TOCTOU bugs" can sometimes be misleading. Indeed, the word "bug" may imply that TOCTOU can only occur at a software level. However, TOCTOU is a logical flaw introduced during the design of a system. It is therefore possible to find this kind of flaw at a software level as well as at a hardware level. Indeed, TOCTOU is caused by a race condition involving a check (for instance checking the credentials of a user) and the use of the result of that check (for instance granting the access to the user or not). This kind of logical flaw is therefore independent of the language used, of the implementation, and even of the hardware involved. Thus, it is possible to exploit a TOCTOU bug regardless of both the type of security (secure update, attestation, secure boot), the application (embedded, general-purpose computers, IoT, WSN, FPGA, etc.), and the abstraction level of the protection (software or hardware). In our case, a TOCTOU bug may occur when a measurement is performed because the integrity of the firmware is ensured at the time of the measurement. However, there is no guarantee that the firmware will not be modified later at the runtime. In other words, the firmware that we have checked may not be the same as the one that is running. The exploitation of this race condition has for example been shown to be feasible against Intel Boot Guard by Peter Bosch and Trammell Hudson [135]. Moreover, the survey of Steiner et al. [15] presents this logical flaw as an important issue in the context of WSN attestation. Finally, De Oliveira Nunes et al. [136] present RATA, an extension of their project VRASED [98] which protects the later against TOCTOU.

TPMs are vulnerable to several attacks. First of all, Jeremy Boone [137] has made a hardware module that intercepts packets sent and received by the TPM via the I2C bus.[2] The author shows that it is possible to attack the PCR extend operations and thus compromising the confidence that can be given to the measurements made by the TPM. Jeremy Boone also shows that the proposed module is able to substitute itself to the random number generator of the TPM, thus making deterministic any calculation based on this generator. Next, Han et al. [138] present two software attacks that allow an attacker to forge the values of the PCR registers, again challenging the trustfulness of the measurements. Finally, Moghimi et al. [139] present attacks against several different TPMs. They can extract keys from these chips using side-channel timing attacks. The authors also demonstrated how to recover a StrongSwan IPSec VPN server's authentication key via a network connection using such attacks. We think that in light of these facts, the use of TPM for new projects should receive increased attention.

The attacks described above show that the solutions proposed in the literature can be attacked by at least two different vectors: software and hardware. Thus, even if secure development of the firmware is an important step to avoid bugs at the software level, it is not sufficient to ensure the security of a system against an attacker with physical access.

2) NEW TECHNOLOGIES

Recently, there has been interest in using new technologies such as blockchain or Artificial Intelligence (AI) to verify the integrity of firmware.

[2]The author has chosen this bus for simplicity and specifies that the support of SPI and LPC buses is planned for a future release. That is why the chosen bus does not seem to impact the attack feasibility.

We have previously discussed the fact that some secure update approaches use a blockchain [38], [66], [68], [82]. However, we note that the use of this blockchain is limited to the update metadata storage. The firmware itself is indeed too large to be stored into a block and must therefore be externalized to another service. We therefore consider that the contribution of a blockchain in this context is questionable since it adds heaviness to the secure update system without providing any real benefit in terms of security. We think that it can however be useful when used for the update traceability and event logging for instance in a context of computer park management. Moreover, the solutions proposed by He et al. [68] and Hu et al. [82] are based on private blockchains and do not discuss the consensus algorithm used or how participating nodes are rewarded to improve the security of the blockchain. We therefore think that further studies should address these issues to highlight the advantages and disadvantages of existing solutions in the context of secure firmware updates using a private blockchain.

Artificial intelligence is used by the approach of Wang et al. [114]. The authors propose a machine learning algorithm allowing the detection of a malicious firmware. They propose two different implementations: a first one doing the computations locally and a second one doing the computations using a remote analyzer. AI can therefore be used in the context of secure boot to perform dynamic analysis of the system. However, we have identified several limitations to this approach. First of all, we must question the learning phase of the algorithm. It is indeed essential to be able to ensure the integrity of the firmware, without which it is possible that learning data are biased due to a malicious firmware [140]. Secondly, this approach is not very adaptive. A modification of the firmware (because of an update for example) requires to redo the whole learning process and recompute every golden value. Finally, the implementation performing the computations remotely requires the analyzer and the whole network stack to be safe, which increases the potential attack surface. Further work are thus expected on this subject, in order to discover solutions to overcome these limitations.

3) BLENDING EVERYTHING TOGETHER

In this survey, we studied three different types of protection: secure update, attestation and secure boot. However, few approaches combine these types of protection. The only example we have found in the literature is the SobTrA solution [91]. This software solution, targeting embedded systems, uses both an attestation and a secure boot mechanism. SobTrA first uses a timing-based attestation mechanism to verify the integrity of the untrusted device by making sure that a piece of code is executed untampered on it. This code can then act as an anchor for a chain of trust and bootstrap the platform thanks to a secure boot mechanism.

As we have seen throughout this paper, applications are diverse and their constraints, challenges, goals and threat models vary from one system to another. It is therefore not enough to combine several independent solutions to create a new solution using several types of protection. Designing such a solution requires a thorough study of the system to be protected. Moreover, combining several types of protection raises several questions which remain open for the moment: which combinations are interesting according to the field of application and the considered threat model? Do these protection mechanisms overlap? Can we have a sense of end-to-end security by using these three protection mechanisms?

VII. CONCLUSION

In this survey, we investigated three protection mechanisms (secure update, attestation and secure boot) used to ensure firmware integrity. We presented two new taxonomies that identify the main characteristics of secure update and secure boot solutions. We also extended the scope of a taxonomy used in the context of WSN attestation to all types of systems. We discussed the limitations we have encountered doing such a generalization as well. In addition, we presented the advantages and the disadvantages of each design choice made for these approaches. Finally, we have identified open research issues and have given guidelines on how to address them.

We have also highlighted the need for research taking into consideration physical attacks, open specifications and using generic hardware. Among the various research opportunities, designing a solution targeting general-purpose computers while taking into account the previously highlighted limitations appears promising. Such approaches should combine several security types, which to the best of our knowledge, has not been done so far.

REFERENCES

[1] aLS and Alfredo. (Jun. 2009). *Persistent BIOS Infection*. Phrack Magazine. Accessed: Nov. 22, 2021. [Online]. Available: http://phrack.org/issues/66/7.html

[2] BSDaemon, Coideloko, and D0nAnd0n. (Mar. 2008). *System Management Mode Hack: Using SMM for 'Other Purposes*. Phrack Magazine. Accessed: Nov. 22, 2021. [Online]. Available: http://phrack.org/issues/65/7.html

[3] F. Wecherowski and Core Collapse. (Jun. 2009). *A Real SMM Rootkit: Reversing and Hooking BIOS SMI Handlers*. Phrack Magazine. Accessed: Nov. 22, 2021. [Online]. Available: http://phrack.org/issues/66/11.html

[4] S. Embleton, S. Sparks, and C. Zou, "SMM rootkits: A new breed of OS independent malware," in *Proc. 4th Int. Conf. Secur. Privacy Commun. Netowrks*, Sep. 2008, pp. 1–12, doi: 10.1145/1460877.1460892.

[5] D. Oleksiuk. (Jul. 2015). *Building Reliable SMM Backdoor for UEFI Based Platforms*. Accessed: Nov. 22, 2021. [Online]. Available: http://blog.cr4.sh/2015/07/building-reliable-smm-backdoor-for-uefi.html

[6] (Nov. 2003). *Intel Platform Innovation Framework for EFI System Management Mode Core Interface Specification (SMM CIS)*. Intel Corporation. Accessed: Nov. 22, 2021. [Online]. Available: https://www.intel.com/content/dam/doc/reference-guide/efi-smm-cis-v091.pdf

[7] N. S. Agency. (Jun. 2008). *NSA ANT Catalog*. Accessed: Nov. 22, 2021. [Online]. Available: https://www.eff.org/files/2014/01/06/20131230-appelbaum-nsa_ant_catalog.pdf

[8] A. Matrosov and A. Gazet, *Breaking Through Another Side–Bypassing Firmware Security Boundaries From Embedded Controller*. Seattle, WA, USA: Black Hat USA, Aug. 2019.

[9] C. Kallenberg, J. Butterworth, X. Kovah, and S. Cornwell. (Mar. 2014). *Defeating Signed BIOS Enforcement*. The MITRE Corporation. Accessed: Nov. 22, 2021. [Online]. Available: https://www.mitre.org/sites/default/files/publications/defeating-signed-bios-enforcement.pdf

[10] C. Valasek and C. Miller, "Remote exploitation of an unaltered passenger vehicle," IOActive, Seattle, WA, USA, Tech. Rep. 1, 2015. Accessed: Nov. 22, 2021. [Online]. Available: https://ioactive.com/wp-content/uploads/2018/05/IOActive_Remote_Car_Hacking-1.pdf

[11] K. Kim, J. S. Kim, S. Jeong, J.-H. Park, and H. K. Kim, "Cybersecurity for autonomous vehicles: Review of attacks and defense," *Comput. Secur.*, vol. 103, Apr. 2021, Art. no. 102150. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820304235

[12] H. Mansor, K. Markantonakis, R. N. Akram, and K. Mayes, "Don't brick your car: Firmware confidentiality and rollback for vehicles," in *Proc. 10th Int. Conf. Availability, Rel. Secur.*, Aug. 2015, pp. 139–148.

[13] M. M. Hossain, S. Mohammad, J. Vosatka, J. Allen, M. Allen, F. Farahmandi, F. Rahman, and M. Tehranipoor, "HEXON: Protecting firmware using hardware-assisted execution-level obfuscation," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2021, pp. 343–349.

[14] B. Cyr, J. Mahmod, and U. Guin, "Low-cost and secure firmware obfuscation method for protecting electronic systems from cloning," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3700–3711, Apr. 2019.

[15] R. V. Steiner and E. Lupu, "Attestation in wireless sensor networks: A survey," *ACM Comput. Surv.*, vol. 49, no. 3, pp. 1–31, Sep. 2016, doi: 10.1145/2988546.

[16] A. Kolehmainen, "Secure firmware updates for IoT: A survey," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData)*, Jul. 2018, pp. 112–117.

[17] M. Bettayeb, Q. Nasir, and M. A. Talib, "Firmware update attacks and security for IoT devices: Survey," in *Proc. ArabWIC 6th Annu. Int. Conf. Res. Track*, Mar. 2019, pp. 1–6, doi: 10.1145/3333165.3333169.

[18] N. S. Mtetwa, P. Tarwireyi, A. M. Abu-Mahfouz, and M. O. Adigun, "Secure firmware updates in the Internet of Things: A survey," in *Proc. Int. Multidisciplinary Inf. Technol. Eng. Conf. (IMITEC)*, Nov. 2019, pp. 1–7.

[19] I. Sfyrakis and T. Gross, "A survey on hardware approaches for remote attestation in network infrastructures," 2020, *arXiv:2005.12453*.

[20] A. Sprogø Banks, M. Kisiel, and P. Korsholm, "Remote attestation: A literature review," 2021, *arXiv:2105.02466*.

[21] B. Kuang, A. Fu, W. Susilo, S. Yu, and Y. Gao, "A survey of remote attestation in Internet of Things: Attacks, countermeasures, and prospects," *Comput. Secur.*, vol. 112, Jan. 2022, Art. no. 102498. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821003229

[22] R. Wang and Y. Yan, "A survey of secure boot schemes for embedded devices," in *Proc. 24th Int. Conf. Adv. Commun. Technol. (ICACT)*, Feb. 2022, pp. 224–227.

[23] J. Harrison, N. Asadizanjani, and M. Tehranipoor, "On malicious implants in PCBs throughout the supply chain," *Integration*, vol. 79, pp. 12–22, Jul. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926021000304

[24] Z. Basnight, J. Butts, J. Lopez, and T. Dube, "Firmware modification attacks on programmable logic controllers," *Int. J. Crit. Infrastruct. Protection*, vol. 6, no. 2, pp. 76–84, Jun. 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1874548213000231

[25] F. E. McFadden and R. D. Arnold, "Supply chain risk mitigation for IT electronics," in *Proc. IEEE Int. Conf. Technol. Homeland Secur. (HST)*, Nov. 2010, pp. 49–55.

[26] L. Katzir and I. Schwartzman, "Secure firmware updates for smart grid devices," in *Proc. 2nd IEEE PES Int. Conf. Exhib. Innov. Smart Grid Technol.*, Dec. 2011, pp. 1–5.

[27] M. M. Ogonji, G. Okeyo, and J. M. Wafula, "A survey on privacy and security of Internet of Things," *Computer Science Review*, vol. 38, Nov. 2020, Art. no. 100312. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013720304123

[28] E. Schiller, A. Aidoo, J. Fuhrer, J. Stahl, M. Ziörjen, and B. Stiller, "Landscape of IoT security," *Comput. Sci. Rev.*, vol. 44, May 2022, Art. no. 100467. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013722000120

[29] T. Park and K. G. Shin, "Soft tamper-proofing via program integrity verification in wireless sensor networks," *IEEE Trans. Mobile Comput.*, vol. 4, no. 3, pp. 297–309, May 2005.

[30] J. Deng, R. Han, and S. Mishra, "Secure code distribution in dynamically programmable wireless sensor networks," in *Proc. 5th Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, 2006, pp. 292–300, doi: 10.1145/1127777.1127822.

[31] X. Jin, P. Putthapipat, D. Pan, N. Pissinou, and S. K. Makki, "Unpredictable software-based attestation solution for node compromise detection in mobile WSN," in *Proc. IEEE Globecom Workshops*, Dec. 2010, pp. 2059–2064.

[32] S. Schulz, A. Schaller, F. Kohnhäuser, and S. Katzenbeisser, "Boot attestation: Secure remote reporting with off-the-shelf IoT sensors," in *Computer Security ESORICS 2017*, S. N. Foley, D. Gollmann, and E. Snekkenes, Eds. Cham, Switzerland: Springer, Aug. 2017, pp. 437–455.

[33] X. Carpent, N. Rattanavipanon, and G. Tsudik, "Remote attestation of IoT devices via SMARM: Shuffled measurements against roving malware," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Apr. 2018, pp. 9–16.

[34] W. Itani, A. Kayssi, and A. Chehab, "PETRA: A secure and energy-efficient software update protocol for severely-constrained network devices," in *Proc. 5th ACM Symp. QoS Secur. Wireless Mobile Netw.*, Oct. 2009, pp. 37–43, doi: 10.1145/1641944.1641952.

[35] M. A. Prada-Delgado, A. Vázquez-Reyes, and I. Baturone, "Trustworthy firmware update for Internet-of-Thing devices using physical unclonable functions," in *Proc. Global Internet Things Summit (GIoTS)*, Jun. 2017, pp. 1–5.

[36] N. Aschenbruck, J. Bauer, J. Bieling, A. Bothe, and M. Schwamborn, "Selective and secure over-the-air programming for wireless sensor networks," in *Proc. 21st Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2012, pp. 1–6.

[37] L. P. Costa, R. Herrero, M. D. Biase, R. Nunes, and M. Zuffo, "Over the air download for digital television receivers upgrade," *IEEE Trans. Consumer Electron.*, vol. 56, no. 1, pp. 261–268, Feb. 2010, doi: 10.1109/TCE.2010.5439154.

[38] P. Gupta, "A decentralized approach towards secure firmware updates and testing over commercial IoT devices," 2020, *arXiv:2011.12052*.

[39] D. He, C. Chen, S. Chan, and J. Bu, "SDRP: A secure and distributed reprogramming protocol for wireless sensor networks," *IEEE Trans. Ind. Electron.*, vol. 59, no. 11, pp. 4155–4163, Nov. 2012.

[40] H. Park, D. Seo, H. Lee, and A. Perrig, "SMATT: Smart meter ATTestation using multiple target selection and copy-proof memory," in *Computer Science and Its Applications*, S.-S. Yeo, Y. Pan, Y. S. Lee, and H. B. Chang, Eds. Amsterdam, The Netherlands: Springer, Oct. 2012, pp. 875–887.

[41] D. K. Nilsson and U. E. Larson, "Secure firmware updates over the air in intelligent vehicles," in *Proc. Workshops IEEE Int. Conf. Commun. Workshops*, May 2008, pp. 380–384.

[42] T. AbuHmed, N. Nyamaa, and D. Nyang, "Software-based remote code attestation in wireless sensor network," in *Proc. GLOBECOM IEEE Global Telecommun. Conf.*, Nov. 2009, pp. 1–8.

[43] J. González, M. Hölzl, P. Riedl, P. Bonnet, and R. Mayrhofer, "A practical hardware-assisted approach to customize trusted boot for mobile devices," in *Information Security*, S. S. M. Chow, J. Camenisch, L. C. K. Hui, and S. M. Yiu, Eds. Cham, Switzerland: Springer, Oct. 2014, pp. 542–554.

[44] National Institute of Standards and Technology, "Security and privacy controls for information systems and organizations," Nat. Inst. Standards Technol., Gaithersburg, MA, USA, Tech. Rep. 800-53, Sep. 2020. Accessed: May 11, 2022. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf

[45] R. Ross, V. Pillitteri, K. Dempsey, M. Riddle, and G. Guissanie, "Protecting controlled unclassified information in nonfederal systems and organizations," Nat. Inst. Standards Technol., Gaithersburg, MA, USA, Tech. Rep. 800-171, Feb. 2020. Accessed: May 11, 2022. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-171r2.pdf

[46] T. Hudson. (Dec. 2016). *Bootstraping a Slightly More Secure Laptop*. 33C3 Works For Me. Accessed: Nov. 22, 2021. [Online]. Available: https://media.ccc.de/v/33c3-8314-bootstraping_a_slightly_more_secure_laptop

[47] Y. Li, J. M. McCune, and A. Perrig, "VIPER: Verifying the integrity of PERipherals' firmware," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, Oct. 2011, pp. 3–16, doi: 10.1145/2046707.2046711.

[48] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla, "Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems," in *Proc. 20th ACM Symp. Operating Syst. Princ.*, Oct. 2005, pp. 1–16, doi: 10.1145/1095810.1095812.

[49] F. Zhang, H. Wang, K. Leach, and A. Stavrou, "A framework to secure peripherals at runtime," in *Proc. 19th Eur. Symp. Res. Comput. Secur. (ESORICS)*. Wroclaw, Poland: Springer, Sep. 2014, pp. 219–238.

[50] A. Dave, M. Wiseman, and D. Safford, "SEDAT: Security enhanced device attestation with TPM2.0," 2021, *arXiv:2101.06362*.

[51] L. Duflot, Y.-A. Perez, and B. Morin, "What if you can't trust your network card?" Agence Nationale de la Sécurité des Systèmes d'information (ANSSI), Paris, France, Tech. Rep. 1, Sep. 2011. Accessed: Nov. 22, 2021. [Online]. Available: https://www.ssi.gouv.fr/uploads/IMG/pdf/paper.pdf

[52] T. Webel, O. Morlok, and D. Kiss, "Z15 selfboot and secure boot," *IBM J. Res. Develop.*, vol. 64, nos. 5–6, pp. 5:1–5:9, Sep. 2020.

[53] S. Johnson, D. Rizzo, P. Ranganathan, J. McCune, and R. Ho, "Titan: Enabling a transparent silicon root of trust for cloud," Google Cloud, Mountain View, CA, USA, Tech. Rep. 1, 2018. Accessed: Nov. 22, 2021. [Online]. Available: https://old.hotchips.org/hc30/1conf/1.14_Google_Titan_GoogleFinalTitanHotChips2018.pdf

[54] I. UEFI Forum. (Mar. 2021). *Unified Extensible Firmware Interface (UEFI) Specification Version 2.9*. UEFI Forum. Accessed: Nov. 22, 2021. [Online]. Available: https://uefi.org/sites/default/files/resources/UEFI_Spec_2_9_2021_03_18.pdf

[55] J. Haj-Yahya, M. M. Wong, V. Pudi, S. Bhasin, and A. Chattopadhyay, "Lightweight secure-boot architecture for RISC-V system-on-chip," in *Proc. 20th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2019, pp. 216–223.

[56] S. Sau, J. Haj-Yahya, M. M. Wong, K. Y. Lam, and A. Chattopadhyay, "Survey of secure processors," in *Proc. Int. Conf. Embedded Comput. Syst., Archit., Modeling, Simulation (SAMOS)*, Jul. 2017, pp. 253–260.

[57] A. Braeken, J. Genoe, S. Kubera, N. Mentens, A. Touhafi, I. Verbauwhede, Y. Verbelen, J. Vliegen, and K. Wouters, "Secure remote reconfiguration of an FPGA-based embedded system," in *Proc. 6th Int. Workshop Reconfigurable Commun.-Centric Syst. Chip (ReCoSoC)*, Jun. 2011, pp. 1–6.

[58] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 2, pp. 198–208, Mar. 1983.

[59] H. B. Yesilyurt, H. Aksu, S. Uluagac, and R. Beyah, "SOTA: Secure over-the-air programming of IoT devices," in *Proc. MILCOM IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2018, pp. 1–8.

[60] B. Lee and J.-H. Lee, "Blockchain-based secure firmware update for embedded devices in an Internet of Things environment," *J. Supercomput.*, vol. 73, no. 3, pp. 1152–1167, Mar. 2017, doi: 10.1007/s11227-016-1870-0.

[61] N.-W. Lo and S.-H. Hsu, "A secure IoT firmware update framework based on MQTT protocol," in *Proc. Int. Conf. Inf. Syst. Archit. Technol. (ISAT)*. Cham, Switzerland: Springer, Sep. 2020, pp. 187–198.

[62] T. Thanh, T. H. Vu, N. Van Cuong, and P. N. Nam, "A protocol for secure remote update of run-time partially reconfigurable systems based on FPGA," in *Proc. Int. Conf. Control, Autom. Inf. Sci. (ICCAIS)*, Nov. 2013, pp. 295–299.

[63] K. Kerliu, A. Ross, G. Tao, Z. Yun, Z. Shi, S. Han, and S. Zhou, "Secure over-the-air firmware updates for sensor networks," in *Proc. IEEE 16th Int. Conf. Mobile Ad Hoc Sensor Syst. Workshops (MASSW)*, Nov. 2019, pp. 97–100.

[64] O. Guillen, B. Nisarga, L. Reynoso, and R. Brederlow, "Crypto-Bootloader - Secure in-field firmware updates for ultra-low power MCUs," Texas Instrum., Dallas, TX, USA, Tech. Rep. 1, Sep. 2015. [Online]. Available: https://www.ti.com/lit/wp/slay041/slay041.pdf

[65] T. A. D. de Pinho, "UpdaThing: A secure and open firmware update system for Internet of Things devices," M.S. thesis, Dept. Inf. Syst. Comput. Eng., Técnico Lisboa, Lisbon, Portugal, Oct. 2016. Accessed: Nov. 22, 2021. [Online]. Available: https://fenix.tecnico.ulisboa.pt/downloadFile/1689244997256690/dissertation.pdf

[66] A. Nanopoulos, "Código network: A decentralized firmware update framework for IoT devices," M.S. thesis, Dept. Inform., Univ. Edinburgh, Edinburgh, Scotland, Aug. 2018. Accessed: Nov. 22, 2021. [Online]. Available: https://project-archive.inf.ed.ac.uk/msc/20182632/msc_proj.pdf

[67] R. Dhobi, S. Gajjar, D. Parmar, and T. Vaghela, "Secure firmware update over the air using TrustZone," in *Proc. Innov. Power Adv. Comput. Technol. (i-PACT)*, vol. 1, Mar. 2019, pp. 1–4.

[68] X. He, S. Alqahtani, R. Gamble, and M. Papa, "Securing over-the-air IoT firmware updates using blockchain," in *Proc. Int. Conf. Omni-Layer Intell. Syst.*, May 2019, pp. 164–171, doi: 10.1145/3312614.3312649.

[69] J. Salowey, A. Choudhury, and D. McGrew, *AES Galois Counter Mode (GCM) Cipher Suites for TLS*, document RFC 5288, RFC Editor, Internet Requests for Comments, Aug. 2008. Accessed: Nov. 22, 2021. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5288.txt

[70] D. McGrew and D. Bailey, *AES-CCM Cipher Suites for Transport Layer Security (TLS)*, document RFC 6655, RFC Editor, Internet Requests for Comments, Jul. 2012. Accessed: Nov. 22, 2021. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6655.txt

[71] S. Kanno and M. Kanda, *Addition of the Camellia Cipher Suites to Transport Layer Security (TLS)*, document RFC 6367, RFC Editor, Internet Requests for Comments, Sep. 2011. Accessed: Nov. 22, 2021. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6367.txt

[72] W. Kim, J. Lee, J. Park, and D. Kwon, *Addition of the ARIA Cipher Suites to Transport Layer Security (TLS)*, document RFC 6209, RFC Editor, Internet Requests for Comments, Apr. 2011. Accessed: Nov. 22, 2021. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6209.txt

[73] A. Langley, W. Chang, N. Mavrogiannopoulos, J. Strombergson, and S. Josefsson, *ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS)*, document RFC 7905, RFC Editor, Internet Requests for Comments, Jun. 2016. Accessed: Nov. 22, 2021. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7905.txt

[74] B.-C. Choi, S.-H. Lee, J.-C. Na, and J.-H. Lee, "Secure firmware validation and update for consumer devices in home networking," *IEEE Trans. Consum. Electron.*, vol. 62, no. 1, pp. 39–44, Feb. 2016.

[75] L. Keleman, D. Matić, M. Popović, and I. Kaštelan, "Secure firmware update in embedded systems," in *Proc. IEEE 9th Int. Conf. Consum. Electron. (ICCE-Berlin)*, Sep. 2019, pp. 16–19.

[76] W. Feng, Y. Qin, S. Zhao, Z. Liu, X. Chu, and D. Feng, "Secure code updates for smart embedded devices based on PUFs," in *Cryptology and Network Security*, S. Capkun and S. S. M. Chow, Eds. Cham, Switzerland: Springer, Nov. 2018, pp. 325–346.

[77] S. Falas, C. Konstantinou, and M. K. Michael, "A hardware-based framework for secure firmware updates on embedded systems," in *Proc. IFIP/IEEE 27th Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Oct. 2019, pp. 198–203.

[78] S. Falas, C. Konstantinou, and M. K. Michael, "Hardware-enabled secure firmware updates in embedded systems," in *VLSI-SoC: New Technology Enabler*, C. Metzler, P.-E. Gaillardon, G. De Micheli, C. Silva-Cardenas, and R. Reis, Eds. Cham, Switzerland: Springer, Jul. 2020, pp. 165–185.

[79] A. S. Siddiqui, Y. Gui, and F. Saqib, "Secure boot for reconfigurable architectures," *Cryptography*, vol. 4, no. 4, p. 26, Sep. 2020. [Online]. Available: https://www.mdpi.com/2410-387X/4/4/26

[80] E. Barker and A. Roginsky, "Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. 800-131A, Jan. 2011. Accessed: Nov. 22, 2021. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar1.pdf

[81] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full SHA-1," CWI Amsterdam Google Res., Amsterdam, The Netherlands, Tech. Rep. 1, Jan. 2017. Accessed: Nov. 22, 2021. [Online]. Available: https://shattered.io/static/shattered.pdf

[82] J.-W. Hu, L.-Y. Yeh, S.-W. Liao, and C.-S. Yang, "Autonomous and malware-proof blockchain-based firmware update platform with efficient batch verification for Internet of Things devices," *Comput. Secur.*, vol. 86, pp. 238–252, Sep. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740481831438X

[83] S. Lee and S. Yoo, "Tux: Trust update on Linux booting," in *Security and Trust Management*, S. K. Katsikas and C. Alcaraz, Eds. Cham, Switzerland: Springer, Oct. 2018, pp. 105–121.

[84] Y. Yang, X. Wang, S. Zhu, and G. Cao, "Distributed software-based attestation for node compromise detection in sensor networks," in *Proc. 26th IEEE Int. Symp. Reliable Distrib. Syst. (SRDS)*, Oct. 2007, pp. 219–230.

[85] H. Tan, W. Hu, and S. Jha, "A remote attestation protocol with trusted platform modules (TPMs) in wireless sensor networks.: Submitted for Wiley security and communication networks," *Secur. Commun. Netw.*, vol. 8, no. 13, pp. 2171–2188, Sep. 2015, doi: 10.1002/sec.1162.

[86] S. Schulz, A.-R. Sadeghi, and C. Wachsmann, "Short paper: Lightweight remote attestation using physical functions," in *Proc. 4th ACM Conf. Wireless Netw. Secur.*, Jun. 2011, pp. 109–114, doi: 10.1145/1998412.1998432.

[87] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "SWATT: Software-based attestation for embedded devices," in *Proc. IEEE Symp. Secur. Privacy*, May 2004, pp. 272–282.

[88] M. Ammar, B. Crispo, and G. Tsudik, "SIMPLE: A remote attestation approach for resource-constrained IoT devices," in *Proc. ACM/IEEE 11th Int. Conf. Cyber-Phys. Syst. (ICCPS)*, Apr. 2020, pp. 247–258.

[89] A. Dave, N. Banerjee, and C. Patel, "SRACARE: Secure remote attestation with code authentication and resilience engine," 2021, *arXiv:2101.06148*.

[90] Y. Li, J. M. McCune, and A. Perrig, "SBAP: Software-based attestation for peripherals," in *Trust and Trustworthy Computing*, A. Acquisti, S. W. Smith, and A.-R. Sadeghi, Eds. Berlin, Germany: Springer, 2010, pp. 16–29.

[91] J. Horsch, S. Wessel, F. Stumpf, and C. Eckert, "SobTrA: A software-based trust anchor for ARM cortex application processors," in *Proc. 4th ACM Conf. Data Appl. Secur. Privacy*, Mar. 2014, pp. 273–280, doi: 10.1145/2557547.2557569.

[92] J. Garay and L. Huelsbergen, "Software integrity protection using timed executable agents," in *Proc. ACM Symp. Inf., Comput. Commun. Secur.*, Jan. 2006, pp. 189–200.

[93] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim, "Remote software-based attestation for wireless sensors," in *Security and Privacy in Ad-Hoc and Sensor Networks*, R. Molva, G. Tsudik, and D. Westhoff, Eds. Berlin, Germany: Springer, Jul. 2005, pp. 27–41.

[94] Y.-G. Choi, J. Kang, and D. Nyang, "Proactive code verification protocol in wireless sensor network," in *Computational Science and Its Applications ICCSA*, O. Gervasi and M. L. Gavrilova, Eds. Berlin, Germany: Springer, Aug. 2007, pp. 1085–1096.

[95] D. Schellekens, B. Wyseur, and B. Preneel, "Remote attestation on legacy operating systems with trusted platform modules," *Sci. Comput. Program.*, vol. 74, nos. 1–2, pp. 13–22, 2008, doi: 10.1016/j.scico.2008.09.005.

[96] H. Tan, G. Tsudik, and S. Jha, "MTRA: Multi-tier randomized remote attestation in IoT networks," *Comput. Secur.*, vol. 81, pp. 78–93, Mar. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818307697

[97] J. Vliegen, M. M. Rabbani, M. Conti, and N. Mentens, "A novel FPGA architecture and protocol for the self-attestation of configurable hardware," ES&S imec-COSIC/ESAT, KU Leuven, Belgium, SPRITZ, Univ. Padua, Italy, Cryptol. ePrint Arch., Paper 2019/405, Apr. 2019. Accessed: Nov. 22, 2021. [Online]. Available: https://eprint.iacr.org/2019/405

[98] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, "VRASED: A verified hardware/software co-design for remote attestation," in *Proc. 28th USENIX Secur. Symp.*, Aug. 2019, pp. 1429–1446.

[99] M. M. Rabbani, J. Vligen, M. Conti, and N. Mentens, "SHeFU: Secure hardware-enabled protocol for firmware updates," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

[100] K. M. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, "SMART: Secure and minimal architecture for (establishing a dynamic) root of trust," in *Proc. NDSS*, Feb. 2012, p. 15. Accessed: Nov. 22, 2021. [Online]. Available: https://www.ics.uci.edu/ gts/paps/smart.pdf

[101] X. Carpent, N. Rattanavipanon, and G. Tsudik, "ERASMUS: Efficient remote attestation via self-measurement for unattended settings," 2017, *arXiv:1707.09043*.

[102] A. Rawat, M. Khodari, M. Asplund, and A. Gurtov, "Decentralized firmware attestation for in-vehicle networks," *ACM Trans. Cyber-Phys. Syst.*, vol. 5, no. 1, pp. 1–23, Dec. 2020, doi: 10.1145/3418685.

[103] N. Asokan, F. Brasser, A. Ibrahim, A.-R. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann, "SEDA: Scalable embedded device attestation," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 964–975, doi: 10.1145/2810103.2813670.

[104] B. Vetter and D. Westhoff, "Simulation study on code attestation with compressed instruction code," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops*, Mar. 2012, pp. 296–301.

[105] D. Evans, P. Bond, and A. Bement, "Security requirements for cryptographic modules," Nat. Inst. Standards Technol., Gaithersburg, MA, USA, Tech. Rep. 800-90C, May 2001. Accessed: May 12, 2022. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf

[106] E. Barker and J. Kelsey, "Recommendation for random number generation using deterministic random bit generators," Nat. Inst. Standards Technol., Gaithersburg, MA, USA, Tech. Rep. 800-90A, Jun. 2015. Accessed: May 12, 2022. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

[107] E. Barker and J. Kelsey, "Recommendation for random bit generator (RBG) constructions," Nat. Inst. Standards Technol., Gaithersburg, MA, USA, Tech. Rep. 800-90C, Apr. 2016. Accessed: May 12, 2022. [Online]. Available: https://csrc.nist.gov/CSRC/media/Publications/sp/800-90c/draft/documents/sp800_90c_second_draft.pdf

[108] M. Mitzenmacher and E. Upfal, *Probability and Computing : Randomized Algorithms and Probabilistic Analysis*. Cambridge, U.K.: Cambridge Univ. Press, Jan. 2005.

[109] L. H. Adnan, H. Hashim, Y. M. Yussoff, and M. U. Kamaluddin, "Root of trust for trusted node based-on ARM11 platform," in *Proc. 17th Asia Pacific Conf. Commun.*, Oct. 2011, pp. 812–815.

[110] H. Yin, H. Dai, and Z. Jia, "Verification-based multi-backup firmware architecture, an assurance of trusted boot process for the embedded systems," in *Proc. IEEE 10th Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Nov. 2011, pp. 1188–1195.

[111] J. Li, H. Zhang, and B. Zhao, "Research of reliable trusted boot in embedded systems," in *Proc. Int. Conf. Comput. Sci. Netw. Technol.*, vol. 3, Dec. 2011, pp. 2033–2037.

[112] C. Huang, C. Hou, H. Dai, Y. Ding, S. Fu, and M. Ji, "Research on Linux trusted boot method based on reverse integrity verification," *Sci. Program.*, vol. 2016, pp. 1–12, Jun. 2016.

[113] N. Itoi, W. A. Arbaugh, S. J. Pollack, and D. M. Reeves, "Personal secure booting," in *Information Security and Privacy*, V. Varadharajan and Y. Mu, Eds. Berlin, Germany: Springer, Jul. 2001, pp. 130–144.

[114] X. Wang, C. Konstantinou, M. Maniatakos, R. Karri, S. Lee, P. Robison, P. Stergiou, and S. Kim, "Malicious firmware detection with hardware performance counters," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 2, no. 3, pp. 160–173, Jul. 2016.

[115] H. Jiang, R. Chang, L. Ren, and W. Dong, "Implementing a ARM-based secure boot scheme for the isolated execution environment," in *Proc. 13th Int. Conf. Comput. Intell. Secur. (CIS)*, Dec. 2017, pp. 336–340.

[116] I. Lebedev, K. Hogan, and S. Devadas, "Invited paper: Secure boot and remote attestation in the sanctum processor," in *Proc. IEEE 31st Comput. Secur. Found. Symp. (CSF)*, Jul. 2018, pp. 46–60.

[117] W. A. Arbaugh, D. J. Farber, and J. M. Smith, "A secure and reliable bootstrap architecture," in *Proc. IEEE Symp. Secur. Privacy*, May 1997, pp. 65–71.

[118] A. Dave, N. Banerjee, and C. Patel, "CARE: Lightweight attack resilient secure boot architecturewith onboard recovery for RISC-V based SOC," 2021, *arXiv:2101.06300*.

[119] F. Devic, L. Torres, and B. Badrignans, "Securing boot of an embedded Linux on FPGA," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops Phd Forum*, May 2011, pp. 189–195.

[120] P. Rouget, B. Badrignans, P. Benoit, and L. Torres, "SecBoot—Lightweight secure boot mechanism for Linux-based embedded systems on FPGAs," in *Proc. 12th Int. Symp. Reconfigurable Commun.-Centric Syst. Chip (ReCoSoC)*, Jul. 2017, pp. 1–5.

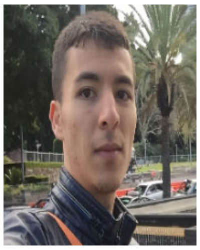
[121] HP Development Company, "Technical whitepaper HP sure start," HP Develop. Company., Palo Alto, CA, USA, Tech. Rep. 902696-002, May 2016, Accessed: Nov. 22, 2021. [Online]. Available: https://web.archive.org/web/20210225092618/http://h10032.www1.hp.com/ctg/Manual/c05163901

[122] HP Development Company, "Technical whitepaper—HP sure start Gen3," HP Develop. Company, Palo Alto, CA, USA, Tech. Rep. 4AA6-9339ENW, Sep. 2017. Accessed: Nov. 22, 2021. [Online]. Available: https://web.archive.org/web/20201109023448/https://www8.hp.com/h20195/v2/GetPDF.aspx/4AA6-9339ENW.pdf

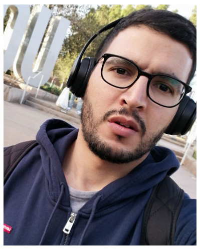
[123] HP Development Company, "Technical white paper—HP sure start," HP Develop. Company, Palo Alto, CA, USA, Tech. Rep. L49253-001, Jan. 2019, accessed 22 November 2021. [Online]. Available: https://web.archive.org/web/20201119103539/http://h10032.www1.hp.com/ctg/Manual/c06216928

[124] X. Ruan, *Boot With Integrity, or Don't Boot*. Berkeley, CA, USA: Apress, 2014, pp. 143–163, doi: 10.1007/978-1-4302-6572-6_6.
[125] J. Yao and V. Zimmer, *Building Secure Firmware—Armoring the Foundation of the Platform*, W. Spahr, S. McDermott, L. Berendson, and J. Vakili, Eds. Berkeley, CA, USA: Apress, Oct. 2020, doi: 10.1007/978-1-4842-6106-4.
[126] Apple Inc., "Apple T2 security chip—Security overview," Apple Inc., Cupertino, CA, USA, Tech. Rep., Oct. 2018. Accessed: Nov. 22, 2021. [Online]. Available: https://www.apple.com/ca/mac/docs/Apple_T2_Security_Chip_Overview.pdf
[127] B. Kelly, "Project Cerberus—Security architecture overview specification," Microsoft Corporation, Redmond, WA, USA, Tech. Rep. L49253-001, Aug. 2017. Accessed: Nov. 22, 2021. [Online]. Available: https://raw.githubusercontent.com/opencomputeproject/Project_Olympus/master/Project_Cerberus/Project%20Cerberus%20Architecture%20Overview.pdf
[128] O. Khalid, C. Rolfes, and A. Ibing, "On implementing trusted boot for embedded systems," in *Proc. IEEE Int. Symp. Hardware-Oriented Secur. Trust (HOST)*, Jun. 2013, pp. 75–80.
[129] F.-J. Streit, F. Fritz, A. Becher, S. Wildermann, S. Werner, M. Schmidt-Korth, M. Pschyklenk, and J. Teich, "Secure boot from non-volatile memory for programmable SoC architectures," 2020, *arXiv:2004.09453*.
[130] Samsung Research America, "Whitepaper: Samsung Knox security solution," Samsung Research Amer., Mountain View, CA, USA, Tech. Rep. 2.2, May 2017. Accessed: Nov. 22, 2021. [Online]. Available: https://images.samsung.com/is/content/samsung/p5/global/business/mobile/SamsungKnoxSecuritySolution.pdf
[131] K.-U. Müller, R. Ulrich, A. Stanitzki, and R. Kokozinski, "Enabling secure boot functionality by using physical unclonable functions," in *Proc. 14th Conf. Ph.D. Res. Microelectron. Electron. (PRIME)*, Jul. 2018, pp. 81–84.
[132] G. Pocklassery, W. Che, F. Saqib, M. Areno, and J. Plusquellic, "Self-authenticating secure boot for FPGAs," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Apr. 2018, pp. 221–226.
[133] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. 25th USENIX Conf. Secur. Symp.*, Berkeley, CA, USA: USENIX Association, Aug. 2016, pp. 857–874.
[134] A. Kerckhoffs, "La cryptographie militaire," *J. Sci. Militaires*, vol. 9, pp. 5–38, Jan. 1883. [Online]. Available: https://www.petitcolas.net/kerckhoffs/crypto_militaire_1_b.pdf
[135] P. Bosch and T. Hudson, "Now you see It. TOCTOU attacks against Boot-Guard," in *Proc. Hack Box Secur. Conf.* Amsterdam, The Netherlands, 2019, pp. 1–47. Accessed: Nov. 22, 2021. [Online]. Available: https://conference.hitb.org/hitbsecconf2019ams/materials/D1T1%20-%20Toctou%20Attacks%20Against%20Secure%20Boot%20-%20Trammell%20Hudson%20&%20Peter%20Bosch.pdf
[136] I. D. O. Nunes, S. Jakkamsetti, N. Rattanavipanon, and G. Tsudik, "On the TOCTOU problem in remote attestation," 2020, *arXiv:2005.03873*.
[137] J. Boone, "TPM genie: Interposer attacks against the trusted platform module serial bus," NCC Group, London, U.K., Tech. Rep. 1, Mar. 2018. Accessed: Accessed: Nov. 22, 2021. [Online]. Available: https://raw.githubusercontent.com/nccgroup/TPMGenie/master/docs/NCC_Group_Jeremy_Boone_TPM_Genie_Whitepaper.pdf
[138] S. Han, W. Shin, J.-H. Park, and H. Kim, "A bad dream: Subverting trusted platform module while you are sleeping," in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)* Baltimore, MD, USA: USENIX Association, Aug. 2018, pp. 1229–1246. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/han
[139] D. Moghimi, B. Sunar, T. Eisenbarth, and N. Heninger, "TPM-FAIL: TPM meets timing and lattice attacks," in *Proc. 29th USENIX Secur. Symp. (USENIX Secur.)* Baltimore, MD, USA: USENIX Association, Aug. 2020, pp. 2057–2073. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-tpm
[140] N. Pitropakis, E. Panaousis, T. Giannetsos, E. Anastasiadis, and G. Loukas, "A taxonomy and survey of attacks against machine learning," *Comput. Sci. Rev.*, vol. 34, Nov. 2019, Art. no. 100199. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013718303289

ANTOINE MARCHAND received the master's degree in computer science from Université Polytechnique Hauts-de-France (UPHF), in 2020, where he is currently pursuing the Ph.D. degree with the Laboratory of Industrial and Human Automation, Mechanics and Computer Science (LAMIH). He is also with Orange Cyberdefense, as part of a CIFRE Agreement. His work focus on computer security, more precisely on reverse engineering, bootkits, hardware attacks, and firmware integrity protection.

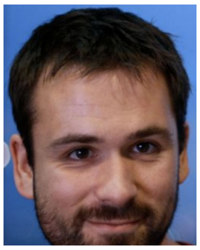


YOUCEF IMINE received the master's degree in networking and distributed systems from Abou Bekr Belkaid University, Algeria, in June 2016, and the Ph.D. degree in computer science and information technologies and systems from the University of Technology of Compiègne, France, in 2019. During his thesis, he worked under the supervision of Prof. Abdelmadjid Bouabdallah, on cloud computing security. He is an Associate Professor with INSA Hauts-de-France. His main works concern communication management and data security in emerging technologies, such as cloud and fog computing and the IoT, which include several challenges, such as data authentication, cryptographic access control, privacy-preserving and trust management, and communication reliability in heterogeneous IoT networks.



HAMZA OUARNOUGHI is an Associate Professor with the Computer Science Department, INSA Hauts-de-France, Valenciennes. He has been a Researcher with the Computer Science Department, LAMIH Laboratory, since 2018. From 2017 to 2018, he was a Temporary Lecturer and an Assistant Researcher with UTBM Belfort. From 2013 to 2017, he was a Ph.D. Student with the Lab-STICC Laboratory, UBO Brest. His research and teaching domains are performances modeling and optimization, cloud/fog/edge computing, computer architecture, and artificial intelligence.

TITOUAN TARRIDEC received the Engineer degree from Tcom Lille, in 2016. He received several offensive security certifications. He first received the OSCP, in 2018; and then the OSCE and the OSWE, in 2020. He has been a Pentester with the Ethical Hacking and Technical Assessment Department, Orange Cyberdefense, since 2016. His work at Orange Cyberdefense, mainly focus on red team penetration testing and anti-malware solutions analysis and bypass.



ANTOINE GALLAIS received the M.Sc. and Ph.D. degrees in computer science from the University of Lille, France, in 2004 and 2007, respectively. He is a Full Professor of computer science with INSA Hauts-de-France, Universitè Polytechnique Hauts-de-France, Valenciennes, France, with research activities in LAMIH, UMR CNRS 8201. He was an Associate Professor with the University of Strasbourg, France, from 2008 to 2019. His main research interests include wireless ad hoc and mesh networks, actuator and sensor networks, the Industrial Internet of Things, activity scheduling, routing and MAC protocols, mobile networks, fault-tolerance, cybersecurity, and performance evaluation.