J. Parallel Distrib. Comput. 119 (2018) 107–120

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A PUF-based hardware mutual authentication protocol

Mario Barbareschi [*], Alessandra De Benedictis, Nicola Mazzocca

Department of Electrical Engineering and Information Technology, University of Naples, Federico II, Naples, Italy

HIGHLIGHTS

- Mutual authentication protocol based on the Physical Unclonable Functions, PHEMAP.
- The protocol inherits PUF properties: unclonability, tamper-evident, uniqueness.
- PHEMAP is devised bearing in mind errors of the PUF and of communication.
- Its efficacy and efficiency are mathematically demonstrated and practically shown.
- PHEMAP introduces a novel enrollment procedure based on self-iteration.

ARTICLE INFO

Article history:
Received 23 October 2017
Received in revised form 1 February 2018
Accepted 11 April 2018
Available online 20 April 2018

MSC:
00-01
99-00

Keywords:
PUF
Mutual authentication
FPGA
Hardware security

ABSTRACT

Physically Unclonable Functions (PUFs) represent a promising security primitive due to their unclonability, uniqueness and tamper-evident properties, and have been recently exploited for device identification and authentication, and for secret key generation and storage purposes.

In this paper, we present PHEMAP (Physical Hardware-Enabled Mutual Authentication Protocol), that allows to achieve mutual authentication in a one-to-many communication scenario, where multiple devices are connected to a sink node. The protocol exploits the recursive invocation of the PUF embedded on the devices to generate sequences (chains) of values that are used to achieve synchronization among communicating parties.

We demonstrate that, under reasonable assumptions, PHEMAP is secure and robust against *man-in-the-middle* attacks and other common physical attacks. We discuss PHEMAP performance in several operation conditions, by measuring the efficiency of the protocol when varying some of the underlying parameters.

Finally, we present an implementation of PHEMAP on devices hosting an FPGA belonging to the Xilinx Zynq-7000 family and embedding an Anderson PUF architecture, and show that the computation and hardware overhead introduced by the protocol makes it feasible for commercial mid-range devices.

1. Introduction

Nowadays, smart devices are used in a wide range of application scenarios; they are highly interconnected and exchange and process a huge amount of data including personal information. With the high popularity that the Internet of Things (IoT) paradigm is recently achieving, we are witnessing not only the growth of business opportunities and the birth of new and complex applications that improve the user experience in several contexts, but also the unprecedented increase of sensitive data exposure that opens new risks for security and safety.

In this scenario, devices identification and authentication are fundamental requirements in order to protect user data and prevent *man-in-the-middle* and object emulation attacks. However, secure authentication is a challenging aspect in the considered application domains due to the nature of involved devices, which may be highly heterogeneous and possibly constrained from the point of view of the available hardware resources, and due to the spread of a wide variety of attacks targeting these devices.

Indeed, physical attacks against embedded digital devices are rapidly growing in complexity and effectiveness, jeopardizing even applications that adopt cryptographic functions as a security shield. These attacks include node cloning, node altering, side channel attacks and physical attacks [25,26] and others [14], aimed at disclosing the cryptographic secrets stored in and used by the devices. It is worth pointing out that successful attacks may not only cause the loss of sensitive data (violating confidentiality), but they may also enable a privilege escalation in the system (violating

* Corresponding author.
E-mail addresses: mario.barbareschi@unina.it (M. Barbareschi), alessandra.debenedictis@unina.it (A. De Benedictis), nimazzoc@unina.it (N. Mazzocca).

https://doi.org/10.1016/j.jpdc.2018.04.007
0743-7315/© 2018 Elsevier Inc. All rights reserved.

authentication) and cause damages that are not bounded within the device under attack.

Recently, Physically Unclonable Functions (PUFs) have been recognized as powerful hardware-based security primitives, able to guarantee attractive security properties thanks to their physical characteristics with a relatively low computational effort [33]. They basically operate a transformation of input binary strings into output binary strings by processing them through an integrated circuit that is affected by random imperfections imprinted by the manufacturing process (related for example to the behavior of internal capacitance and resistance). These imperfections confer uniqueness and unclonability properties on PUFs, and make PUF responses particularly suited to be used for the generation of cryptographic keys, since they are a good source of randomness, can be hardly guessed, and can be generated on demand when PUFs are powered-up, without the need for a non-volatile memory for their storage. Moreover, thanks to the above discussed properties, PUFs can be successfully adopted for the identification and authentication of embedded devices, as shown by several recent scientific results (such as [6,16,34,37] and others).

PUF-based authentication typically requires that the authenticator entity and the devices embedding the PUF share some knowledge on the output produced by the PUF when presented with specific stimuli. Authentication is carried out according to a *challenge–response* paradigm, and implies that the authenticator stores the shared set of input/output pairs (named *challenge–response pairs*) in order to perform device identity verification. Note that, in order to thwart common attacks and to allow for multiple authentication operations during time, the set of challenge–response pairs to store (along with related information needed to ensure PUF stability) is typically very large, with an inevitably negative impact on the performance.

If a mutual authentication is needed, even the devices must be able to keep track of the pairs shared with the counterpart in order to be able to verify its identity. Since in real applications devices are often resource-constrained, this makes mutual authentication harder to achieve, and therefore most of the existing PUF-based authentication schemes only allow to verify the identity of devices. This may be an issue in many applications, such as for example cyber–physical systems, where central entities typically control peripheral nodes provided with sensors and actuators that directly deal with critical data and systems. In this case in fact, nodes should be able to verify the identity of the requester before executing received instructions.

Paper Contribution. In this paper, we present Physical Hardware-Enabled Mutual Authentication Protocol (PHEMAP) designed to provide *mutual* authentication among the nodes involved in a communication scenario where a large number of (typically constrained) devices exchange control and data information with a central entity (sink node) provided with higher storage and communication capabilities.

The protocol exploits the recursive invocation of the PUF embedded on the devices to generate sequences (chains) of values that are used to achieve synchronization among communicating parties. Chains are generated (and then stored) by the sink node by accessing the devices' circuitry in a protected environment during an enrollment phase. Chain values are then used during the subsequent authentication phases by both the devices and the sink node to mutually prove their identity. The adoption of chains allows to simplify the management of challenge–response pairs at the sink node and enables mutual authentication without requiring a great storage and computation effort from the devices.

We demonstrate that, under reasonable assumptions, PHEMAP is secure and robust against *man-in-the-middle* attacks and other common physical attacks, such as tampering and differential power attacks. We discuss PHEMAP performance in several operation conditions, by measuring the efficiency of the protocol when varying some of the underlying parameters. Finally, we present an implementation of PHEMAP on a programmable device (Field-Programmable Gate Array — FPGA) belonging to the Xilinx Zynq-7000, and embedding an Anderson PUF architecture, and show that the computation and hardware overhead introduced by the protocol makes it feasible for commercial mid-range devices.

Organization of the Paper. The paper is structured as follows: in Section 2, we provide the needed background on PUFs and survey some research efforts related to PUF architectures and applications, with emphasis on existing PUF-based authentication schemes. In Section 3, we formally present the mutual authentication problem by discussing the main assumptions behind our work and by introducing the notions and formalism adopted in the PHEMAP protocol description. In Section 4, we provide a detailed explanation of the protocol phases and point out possible related issues and corresponding solutions. Section 5 is devoted to analyzing the security and performance of PHEMAP, while Section 6 illustrates a pragmatic implementation of the approach on a Xilinx Zynq-7000 FPGA. Finally, in Section 7, some final remarks are reported along with a brief discussion of the adoption of the main PHEMAP concepts in different scenarios from the one considered in this paper.

2. Background and related work

In this section, the needed background on PUFs is provided, along with a discussion of the existing solutions for PUF-based authentication.

2.1. Physically unclonable functions

PUFs are hardware primitives usually implemented in integrated circuits, whose physical micro-structure depends on unpredictable factors introduced during the manufacturing process that cannot be cloned or duplicated (i.e., it is practically infeasible to exactly reproduce them by controlling the fabrication process). From a functional point of view, PUFs are similar to one-way functions, since the mapping between inputs and outputs is repeatable but cannot be modeled by means of invertible mathematical functions [30]. The electrical stimulus applied to a PUF is usually referred to as the *challenge*, while the corresponding reaction is called the *response*. As anticipated, the set of challenge–response pairs (CRPs) characterizing a PUF instance represents a unique feature that can be used for its identification. Any malicious alteration of the PUF structure would be easily detected, thus making PUFs tamper-evident equipments [5].

In the literature, several PUF architectures for silicon devices have been proposed, which can be mainly categorized as *delay-based* or *memory-based*. Delay-based PUFs, such as the Arbiter PUF [28], the Ring Oscillator (RO) PUF [12,31,37], the Butterfly PUF [27], and the Anderson PUF [4], measure microscopic random delay variations characterizing specific paths in an electronic circuit. Instead, memory-based PUFs, such as the SRAM PUF [21], the D flip-flop PUF [39] and the STT-MRAM PUF [41], exploit unpredictable mismatches of the memory cells.

In order to estimate the quality of PUF architectures, several metrics have been introduced [31], including: (i) the PUF *uniqueness*, which is an indicator of the randomness introduced by the manufacturing process among different instances of the same PUF structure; (ii) the PUF *reliability*, which is a measure of how well the variations of environmental conditions (including supply voltage [42], temperature [4], and aging [24]) are tolerated by a PUF with respect to produced responses; (iii) the PUF *uniformity*, which estimates the distribution of logic-1s and logic-0s in the responses.

Clearly, high levels of uniqueness and reliability are desirable for a good PUF, while uniformity should be as close as possible to 50% in order to ensure random responses. Since many of the proposed PUFs do not achieve the desired quality, several post-processing techniques have been introduced, such as Fuzzy Extractors [15], and Majority Voters [29], which can be applied in order to improve a PUF performance. In particular, as for the Fuzzy Extractor technique, it involves two main primitives, namely (i) the error reconciliation, based on an error correction code, which recovers the reliability of the PUF responses, and (ii) the privacy amplification, which increases the information entropy of each response. Inevitably, the error reconciliation requires that devices have to store so called *helper data*, which however can be publicly exchanged since they are not useful to extract information about PUF responses [15].

PUFs can also be classified into *strong* PUFs and *weak* PUFs based on the number of unique challenges that they are able to process. A weak PUF can only support a limited amount of challenges, while strong PUFs are characterized by a large CRP set, such that the exhaustive measurement of all challenge–response pairs would result computationally infeasible (e.g., it has been demonstrated [20] that 100 bits-long challenges would ensure a negligible probability of success).

However, while an exhaustive measurement of CRPs is infeasible for strong PUFs, they are inherently susceptible to *modeling attacks* [35], which adopt complex machine learning techniques to obtain a numerical model of the PUF challenge–response relationship by relying on the knowledge of a limited amount of CRPs.

Therefore, strong PUFs particularly subject to these types of attacks, such as the Arbiter PUF, should be avoided or, alternatively, additional protection mechanisms should be adopted in order not to expose CRPs to potential attackers. Gassend et al. proposed in [18] a mechanism to obtain reliable responses that cannot be directly used to infer a mathematical model of PUFs. Such a mechanism, named Controlled PUF, hides the real PUFs behavior by means of a hashing function used to mask both the challenge and the response.

2.2. PUF-based authentication

Due to their properties, PUFs can be adopted in a wide range of security applications, such as device authentication and identification [16,37], cryptographic secret key generation and storage [3,30], and Intellectual Property protection [1,13]. In particular, both weak and strong PUFs can be exploited in order to authenticate individual devices without involving the adoption of cryptographic primitives and with a relatively low hardware overhead. As said, the CRPs characterizing a PUF represent its fingerprint and can be used to prove the identity of the device that embeds the PUF circuit. In order to enable authentication, the authenticating device and the authenticator node (called *verifier* hereafter) need to share some knowledge about the device PUF, i.e., one or more CRPs: in a basic authentication protocol, the verifier presents the device with a *known* challenge (i.e., with a challenge belonging to a known CRP) and compares the response provided by the device with the expected response. Clearly, in this scenario, an attacker may intercept the exchanged messages and record the overheard CRPs in order to use them later to carry out a *man-in-the-middle* attack.

In order to cope with this issue, authors in [37] proposed the scheme shown in Fig. 1, which is based on strong PUFs and turns out in an authentication mechanism composed of two distinct phases:

1. **Enrollment:** during initialization, a Trusted Third Party (TTP) (i.e., the verifier), in possession of the device embedding the PUF, applies (in a trusted environment) a significant number of randomly chosen challenges, and stores the corresponding responses in a secure database for future authentication operations;

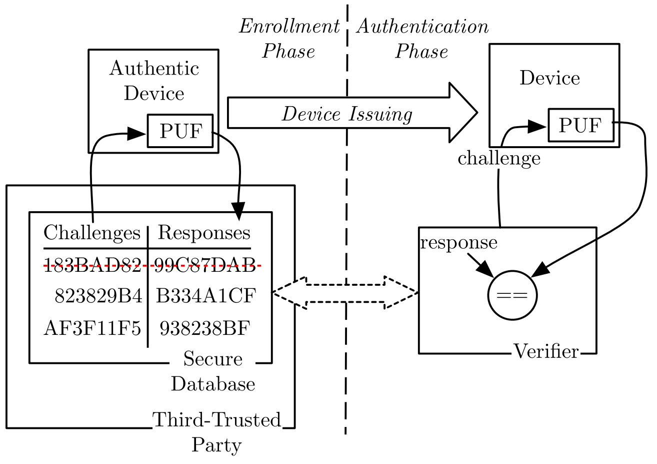


Fig. 1. PUF-based authentication mechanism.

2. **Verification:** at run-time, when the TTP needs to authenticate the device, it selects a challenge that has never been used before from the database, and remotely obtains a PUF response from the device. If the comparison between the response provided by the device and the one previously recorded succeeds, the verifier can prove that the device is authentic, since only the authentic device and the verifier know the CRP.

Note that, in this protocol, challenges and responses can be sent in cleartext during the verification phase, but challenges must be used only once to prevent malicious nodes from replaying old messages in order to perform *man-in-the-middle* attacks.

In [34], Rostami et al. proposed a more complex authentication scheme that does not involve the exposure of full PUF responses or of their transformations, but entails the exchange of random subsets of the PUF response strings. The scheme is based on the assumption that the verifier has access to a PUF compact model and is able to find a match between a sub-string and the corresponding full response to perform authentication.

Related to the obfuscation of PUF responses, the Authors of [6] proposed a device authentication scheme featuring a model-based PUF (i.e., a PUF for which it is possible to model the challenge–response relationship), used in combination with an encryption module. The model-based PUF is used to speed-up the enrollment phase, since the entire CRP set can be derived from the underlying model. In order to preserve the unclonability property of the PUFs, the model is kept hidden by applying an encryption algorithm to challenges, which are first encrypted and then exchanged with the devices for authentication purposes.

A model-based PUF was also used by Yu et al. to design the mutual authentication mechanism discussed in [45]. The proposed protocol exploits a very limited challenge set in order to reduce the amount of training data exposed to a potential attacker, and involves a 3-way clear-text CRP exchange between the verifier and the devices. Compared to this solution, as discussed in Section 4.3, PHEMAP allows to achieve authentication with only two messages (provided that an initialization has been carried out), and it is able to guarantee higher security since it does not rely on model-based PUFs. As a drawback, PHEMAP requires high-reliable PUFs, which can be obtained by paying the overhead of implementing a post-processing technique such as the Fuzzy Extractor.

Van Herrewege et al. illustrated in [40] a mutual authentication protocol based on the reverse Fuzzy Extractor scheme. Even if their approach resembles the PHEMAP scheme, they exploit a static and fixed identifier that resides within the device to achieve authentication, while the scheme we propose is free of any static information in addition to the PUF circuit. Moreover, even in this case, the number of messages to exchange in order to achieve mutual authentication is higher than in PHEMAP.

Finally, Aman et al. proposed in [2] a mutual authentication scheme for IoT applications. Their approach, as the two previously discussed, requires three messages to complete a mutual authentication. Moreover, by analyzing the protocol, it is easy to find that it is prone to a very simple attack: an attacker that is able to drop a device ID from the first message in fact, would be able to query the verifier for an arbitrary number of times, thus getting a partial view of its CRP set. Conversely, PHEMAP is formally demonstrated to be immune against common attack schemes, as widely discussed in Section 5.1.

3. Problem statement and formalization

As previously stated, the main objective of this paper is to present a PUF-based protocol for the mutual authentication of the nodes involved in a communication. In particular, we focus on a one-to-many communication scenario where a (typically large) set of constrained devices communicate with a sink node (the verifier), provided with higher computation, storage and communication capabilities, in order to exchange data and control information. In this scenario, we aim at providing an efficient solution to enable the mutual authentication among the devices and the verifier that does not involve the adoption of complex cryptographic primitives and does not introduce a high overhead compared to similar solutions. In the following, we provide a formalization of the mutual authentication problem and discuss the main concepts that are used by PHEMAP along with the main assumptions it founds on.

3.1. The mutual authentication problem

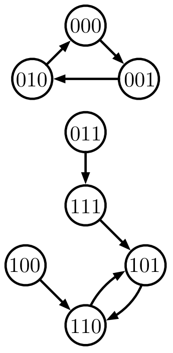
Let $\theta_D(\cdot)$ be the PUF associated with a device D, C be the set of challenges for $\theta_D(\cdot)$ and R be the set of corresponding responses. Let challenges and responses be N bits long, such that both a generic challenge $c \in C$ and the corresponding response $r = \theta_D(c)$ can be expressed as N-bit binary strings $\{0, 1\}^N$. The set Φ_D of the CRPs of the PUF $\theta_D(\cdot)$ includes 2^N pairs $\langle c \in C, r \in R\rangle$. Moreover, as each element of R is generated by mapping the set C through the PUF, R is a subset of C ($R \subseteq C$). Note that, in case the considered PUF does not produce responses with the same length as the submitted challenges, hash functions may be easily adopted to meet this assumption.

Fig. 2 shows a simple example PUF that works with 3-bit binary challenges and responses. As shown in the figure, a PUF can be represented by means of a table, which simply maps responses to challenges, or in form of a directed graph, whose vertices correspond to challenge and response values, and whose edges model the mapping between them through the PUF.

Let us consider the authentication protocol shown in Fig. 1. Let $\overline{C} \subset C$ be the set of challenges used by the verifier during the enrollment phase to obtain a subset of CRPs of the PUF $\theta_D(\cdot)$. The verifier is able to authenticate the device D if, given a challenge $c \in \overline{C}$, the response r computed by the device as $\theta_D(c)$ is equal to the response obtained by the verifier for c during enrollment. Conversely, the device D is able to authenticate the verifier if the latter proves to have enrolled the PUF $\theta_D(\cdot)$ by providing the correct response for a picked challenge in $\overline{C}$. However, bearing in mind how the enrollment phase is executed in the considered authentication protocol, the device D is not aware of all the CRPs extracted by the verifier during enrollment and, hence, is not able to authenticate it, unless additional mechanisms are implemented.

In order to overcome this issue, we devised a mechanism that provides devices with the knowledge needed to authenticate the

c	$\theta(c)$
000	001
001	010
010	000
011	111
100	110
101	110
110	101
111	101



$$R = \{000, 001, 010, 101, 110, 111\} \subseteq C$$

Fig. 2. Example of PUF representation.

verifier without requiring great storage and computation capabilities. Let us consider the graph-based representation of a PUF (refer to Fig. 2): note that, since $R \subseteq C$, it is possible to identify several paths in the graph due to the partial overlap between the challenge and response sets. If considering this property from a different point of view, these paths may be seen as sequences or *chains* of values obtained by recursively applying the PUF. Going back to the mutual authentication problem, we found that PUF chains could help simplify the management of challenge–response pairs for both the sink node and the devices. During enrollment in fact, the verifier can generate CRPs by following a chain rather than using random challenges, and can still use these CRPs to prove the identity of the device. On the way back, a device can authenticate the verifier by submitting a challenge taken from the chain and by verifying that the counterpart knows the corresponding response. Clearly, this approach requires that the verifier and the device are synchronized on the same chain, and the proposed protocol aims at achieving this synchronization. In the next section, we provide the formalization of PUF chains and introduce the concept of chain *sentinel*, which is fundamental in the design of the protocol to ensure protection against common attacks.

3.2. PUF chains and chain sentinels

Given the PUF $\theta_D(\cdot)$ associated with the device D, we define a *chain* a simple path in the graph representing $\theta_D(\cdot)$, i.e., a directed and connected sub-graph in $\theta_D(\cdot)$ that does not contain loops. Let $\gamma_{D,c_0,M} = \langle\theta_D(\cdot), c_0, M\rangle$ be a chain starting with the challenge c_0 (called the *chain root*) and containing M vertices, referred to as *links* hereafter.

Following the previous discussion, it is possible to give a different definition of chain, based on the recursive application of the PUF function. Let $\theta_D^n(\cdot)$ be the n-times iterated self composition of the PUF $\theta_D(\cdot)$. For example, the 2-times iterated self composition of the PUF is $\theta_D^2(\cdot) = \theta_D(\theta_D(\cdot))$. A chain $\gamma_{D,c_0,M}$ can be defined as the sequence of links $\{l_0, l_1, \dots, l_{M-1}\}$, where:

- $l_0 = c_0$;
- $l_i = \theta_D(l_{i-1}) = \theta_D^{(i)}(l_0), \qquad \forall i \in [1, M-1]$;
- $l_a \neq l_b, \qquad \forall l_a, l_b \in \gamma_{D,c_0,M},\ a \neq b.$

Fig. 3 shows two example chains for the function reported in Fig. 2, namely $\langle\theta(\cdot), 000, 3\rangle$ and $\langle\theta(\cdot), 011, 4\rangle$. The double-circled nodes in the figure represent the chains' roots.

As it will be clarified in Section 4, the proposed PHEMAP protocol relies on the set Γ_D of distinct chains extracted from $\theta_D(\cdot)$, such

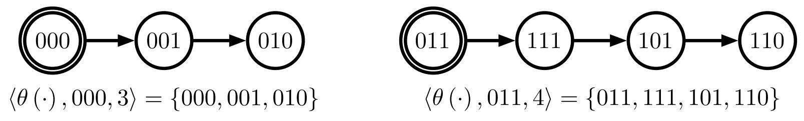

000
001
010
$\langle\theta(\cdot), 000, 3\rangle = \{000, 001, 010\}$
011
111
101
110
$\langle\theta(\cdot), 011, 4\rangle = \{011, 111, 101, 110\}$

Fig. 3. Two example chains extracted from the graph reported in Fig. 2.

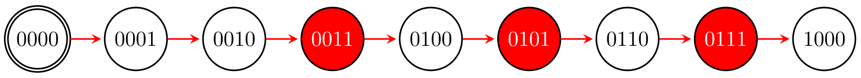


Fig. 4. $\gamma = \langle\theta(x) = x + 1, 0000, 9, 0011, 2\rangle$, i.e., a chain of a PUF $\theta(x) = x + 1$ obtained by an eight-times self composition. The sentinels (red circles) occur with a period of 2. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

that each link l_i is unique and belongs to one and only one chain γ_D. This property can be formalized as follows:

$$\gamma_{D,c_0,M} \cap \gamma_{D,c_0',L} = \varnothing, \qquad \forall \gamma_{D,c_0,M}, \gamma_{D,c_0',L} \in \Gamma_D,\ c_0 \neq c_0'.$$

Finally, let us introduce the concept of chain sentinels. Given a chain $\gamma_{D,c_0,M}$, a link σ_0 on the chain and a positive integer S, the links that appear in the chain in positions multiple of S starting from link σ_0 (*sentinel root*) are referred to as chain *sentinels*.

The definition of chain can be extended by specifying the period S of sentinels and the sentinel root σ_0 as follows: $\gamma_{D,c_0,M,S,\sigma_0} = \langle\theta_D(\cdot), c_0, M, S, \sigma_0\rangle$. In Fig. 4, the chain $\langle\theta(x) = x + 1, 0000, 9, 0011, 3\rangle$ is shown. As it will be discussed in the next section, sentinels play a fundamental role in the proposed mutual authentication protocol from the point of view of security.

Before going into the details of our proposal, in the next section we present the main assumptions it founds on, related both to the involved PUF architectures and to the capabilities of the nodes.

3.3. Protocol assumptions

For what regards the PUF architectures adopted for the protocol implementation, we assume that:

1. involved PUF circuits behave as *strong PUFs*, so that the exhaustive measurement of all CRPs would result computationally unfeasible;
2. involved PUFs are *tamper-resistant* and their interface can be accessed only by the device through its inner logic, except for the enrollment phase, when a proper external interface is enabled to be used by the verifier only;
3. involved PUFs are *reliable*, such that the probability of having a noisy response, due to unpredictable variations caused by changes in the environmental conditions, is sufficiently low.

It is worth pointing out that the above assumptions are reasonable and do not restrict the feasibility of the protocol in real scenarios. In particular, when strong PUFs are not available, a hardware component that combines a weak PUF and a symmetric cipher may be used [8], without changing the protocol as discussed below. The assumption of adopting reliable PUFs can be easily met by considering either an intrinsically reliable PUF such as the Anderson PUF [4] or by applying well-known techniques, such as error fuzzy extraction, which have been proven to be effective in improving PUF reliability [15]. Nevertheless, in Section 4.4, we will discuss the effects of unreliable responses on the protocol in order to prove the robustness of PHEMAP. For what regards tamper-resistance, such a property is intrinsically provided by PUF circuits, while the exclusive accessibility to inner logic of devices embedding PUFs can be achieved by disabling an external interface before the devices issue. For instance, if the PUF is embedded on an FPGA, the designer could produce two bitstreams: one in which the external interface is enabled, to be used during the enrollment phase and kept protected, and the other one with no external access to the PUF circuitry, to be used during normal operation.

For what regards the nodes capabilities and operations, we assume that:

1. the verifier enrolls PUFs in a trusted environment and stores the extracted CRPs in a secure database;
2. each device includes a tamper-proof secure perimeter, where sensitive information can be stored safely, such that it cannot be altered by a malicious user during the protocol execution;
3. both the verifier and the devices are equipped with a pseudo-random number generator, used for the protocol operations.

4. Mutual authentication protocol

The proposed mutual authentication protocol extends the simple authentication protocol discussed above by exploiting the chains extracted by the devices PUFs, which are used both in the enrollment and in the verification phases. In particular, during the *enrollment* phase, a subset of the chains associated with a given PUF is extracted by the verifier and used to enroll the corresponding device. During *verification*, chain links are used to generate authentication messages exchanged between the verifier and the device. Verification requires that the device and the verifier are synchronized on the same chain, therefore an *initialization* phase is needed whenever a new device is issued/reset or in particular cases that will be discussed later in Section 4.4. In the following subsections, the details of the enrollment, initialization and verification phases will be discussed.

4.1. Enrollment

During the enrollment phase, carried out in a protected environment, the verifier generates and stores a set of PUF chains for each device. A chain $\gamma_{D,c_0,M}$ is generated by picking a random challenge c_0 and by iteratively applying the PUF function $\theta_D(\cdot)$ to it for M times. The length M of each chain depends on the number of new links that can be generated by applying the PUF self composition to the selected challenges, so that each link appears only once over the extracted chains. The chain generation is performed on a set of T random challenges in order to have T different chains. Generated chains are stored together with their corresponding helper data, meant to be used for recovering noisy responses [15]. Clearly, the number of enrolled chains for each device and the number of devices that can be managed by a verifier depend on its storage capacity, since all the chains are stored locally.

The enrollment algorithm related to the generation of a single chain is reported below. It takes in input the set of chains previously extracted, the PUF function and the maximum chain length M_{max}, and provides in output a new chain starting from a random link c_0 and of length $M \leq M_{max}$.

Algorithm 1 Chain generation algorithm

```
Input: $\Gamma_D, M_{max}, \theta_D$
Output: $\gamma_{D,c_0,M} \notin \Gamma_D, M \leq M_{max}$
    $M \leftarrow 0$
    repeat
        $c_0 \leftarrow rand$
    until $c_0 \in \Gamma_D$
    $M \leftarrow M + 1$
    initialize $\gamma_{D,c_0,M}$
    $x \leftarrow c_0$
    while $M < M_{max}$ and $\theta_D(x) \notin \Gamma_D$ do
        $x \leftarrow \theta_D(x)$
        append $x$ to $\gamma_{D,c_0,M}$
        $M \leftarrow M + 1$
    end while
```

As mentioned, if a strong PUF is not available, an alternative implementation of the PUF circuit that combines a weak PUF with a cipher can be used. In this case, in the enrollment phase, the PUF is used to generate the cipher key, operation that can be done by the manufacturer who must also supply the related helper data. The enrollment phase carried out by the verifier consists in submitting challenges to the cipher interface rather than to the PUF interface. Note that this solution would not affect the mathematical unclonability of the PUF, since the key remains secret and no one is able to recover it to emulate the PUF behavior.

4.2. Initialization

As it will be discussed in the next subsection, chains generated during enrollment are used for verification in a revised version of the traditional challenge–response paradigm, where the authenticator entity submits a challenge to the supplicant corresponding to a link l_i of an enrolled chain and verifies that the received response matches with the link l_{i+1} that follows l_i on the chain. Note that, while it is easy for the verifier to pick a new challenge belonging to an enrolled chain and to verify the corresponding response provided by the device by simply following the chain, this operation is not trivial for a device. Based on the assumptions made before in fact, devices are provided with very limited storage and computation capabilities, and cannot keep track either of the chains that have been enrolled or of the links that have been already used in previous verification steps. In order to address these issues, we devised an initialization step that not only allows to achieve a first mutual authentication between the verifier and the device, but also enables them to synchronize on the link to use for future verifications.

Initialization is always launched by the verifier and involves the exchange of suitable messages built by manipulating a number of consecutive links on a chain by means of the use of XOR functions and nonces. As it will discussed in Section 5.1, the initialization protocol exploits the concept of sentinel introduced in Section 3.2 in order to ensure that only legitimate verifiers can achieve synchronization. Basically, being S the sentinel period, defined a-priori and embedded in both the devices and the verifier from the network set-up, initialization requires the knowledge of at least S consecutive links in order to build protocol messages. Hence, protection against attacks aimed at tampering with the protocol is ensured by the property that no more than $S - 1$ consecutive links are exposed during any of the phases of PHEMAP, so that an attacker is not able to forge messages and impersonate the verifier. The initialization protocol involves four phases, discussed in the following.

Phase 1: The verifier generates a random nonce n and sends

$$msg_1 = \left\{ l_i, \left(\bigoplus_{j=0}^{S-3} l_{i+1+j} \right) \oplus n, l_{i+S-1} \oplus n \right\} = \{l_i, v_1, v_2\}$$

to the device D.

Phase 2: The device D verifies:

$$\bigoplus_{j=0}^{S-2} \theta_D^{(j+1)}(l_i) == v_1 \oplus v_2.$$

If the comparison succeeds, the device generates a random nonce m and computes:

$$msg_2 = \left\{ \theta_D^{(S)}(l_i) \oplus m, \theta_D^{(S+1)}(l_i) \oplus m \right\} = \{d_1, d_2\}$$

which is sent to the verifier. Moreover, the device saves d_2 in its secure register.

Phase 3: The verifier computes:

$$l_{i+S} \oplus l_{i+S+1} == d_1 \oplus d_2.$$

If the comparison succeeds, the verifier authenticates the device and sends

$$msg_3 = \{l_i, l_{i+S+2} \oplus m\} = \{l_i, v_3\}$$

to it.

Phase 4: The device computes:

$$\theta_D^{(S+1)}(l_i) \oplus \theta_D^{(S+2)}(l_i) == v_3 \oplus d_2$$

If the comparison succeeds, the device authenticates the verifier and saves $\theta_D^{(S+2)}(l_i)$ in its secure register.

It is worth noting that, due to the structure of the protocol, S must be at least equal to 4, and therefore the initialization protocol involves at least 7 links including l_i. After initialization completion, the verifier and the device are both synchronized on the link $\theta_D^{(S+2)}(l_i)$. The next link $\theta_D^{(S+3)}(l_i)$ constitutes the first sentinel of the chain and will not be exchanged during the following operations.

Example. Fig. 5 illustrates an example of initialization protocol starting from the link l_0, with $S = 4$. In the first phase, the verifier uses the first 4 links to build msg_1, which contains l_0 in plain text, the XOR of l_1 and l_2 with a random nonce n, and the XOR of l_3 with the same nonce.

In the second phase the device performs the comparison but, even if it succeeds, the device cannot authenticate the verifier, as such a message may have been generated as a replay of a previous one. Hence, the device prepares msg_2, which contains additional two links (l_4 and l_5) in combination with a nonce m to prove the possession of the PUF used to compute the CRPs extracted by the verifier during the enrollment phase.

In the third phase, the verifier is able to easily prove that the received message contains proper links, and consequently to authenticate the device. To demonstrate, in turn, that it is in possession of the plain version of previous exchanged links (l_4, and l_5), the verifier sends msg_3, which includes the root link and the link l_6 XOR-ed with the nonce m generated by the device. Finally, in the last phase, the device authenticates the verifier if the message received corresponds to the expected value. At this point, the verifier and the device are both synchronized on the link l_6, while the next link l_7 is the first sentinel of the chain. □

4.3. Verification

The *verification* phase can be initiated by both the verifier and the device and consists in a simple link exchange that involves two consecutive links belonging to a chain extracted from the device PUF. Basically, being l_k the link sent by the initiator of the protocol, the other party must prove that it knows the subsequent link l_{k+1} on the chain to be authenticated. As mentioned, in order to prevent replay attacks, the CRPs used for authentication must be used only

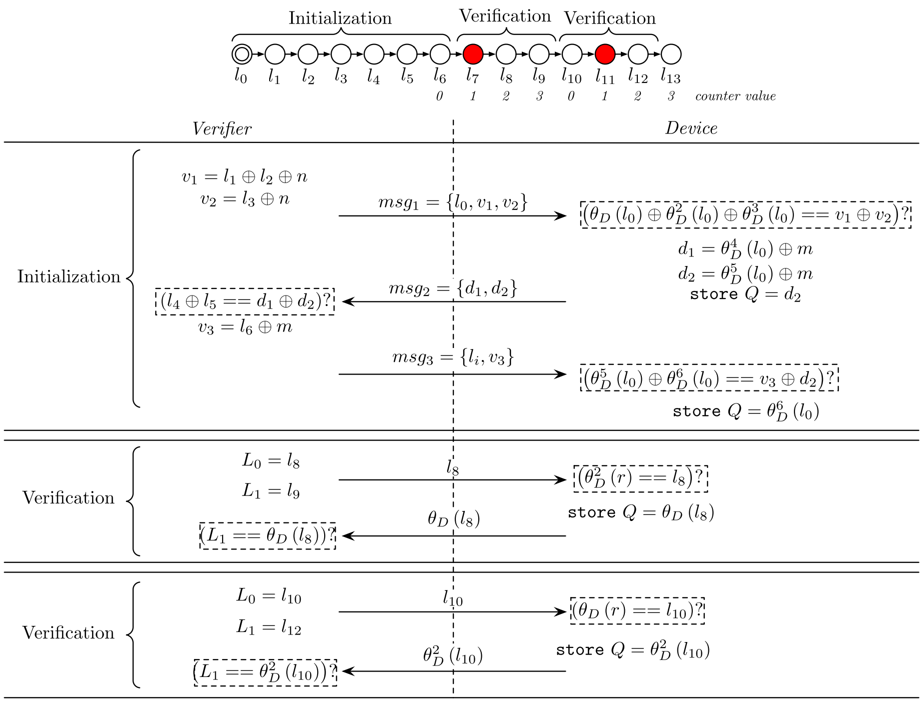

Initialization
Verification
Verification
l_0 l_1 l_2 l_3 l_4 l_5 l_6 l_7 l_8 l_9 l_{10} l_{11} l_{12} l_{13}
0 1 2 3 0 1 2 3 counter value
Verifier
Device
Initialization
$v_1 = l_1 \oplus l_2 \oplus n$
$v_2 = l_3 \oplus n$
$msg_1 = \{l_0, v_1, v_2\}$
$(\theta_D(l_0) \oplus \theta_D^2(l_0) \oplus \theta_D^3(l_0) == v_1 \oplus v_2)?$
$d_1 = \theta_D^4(l_0) \oplus m$
$d_2 = \theta_D^5(l_0) \oplus m$
store $Q = d_2$
$msg_2 = \{d_1, d_2\}$
$(l_4 \oplus l_5 == d_1 \oplus d_2)?$
$v_3 = l_6 \oplus m$
$msg_3 = \{l_i, v_3\}$
$(\theta_D^5(l_0) \oplus \theta_D^6(l_0) == v_3 \oplus d_2)?$
store $Q = \theta_D^6(l_0)$
Verification
$L_0 = l_8$
$L_1 = l_9$
l_8
$(\theta_D^2(r) == l_8)?$
store $Q = \theta_D(l_8)$
$\theta_D(l_8)$
$(L_1 == \theta_D(l_8))?$
Verification
$L_0 = l_{10}$
$L_1 = l_{12}$
l_{10}
$(\theta_D(r) == l_{10})?$
store $Q = \theta_D^2(l_{10})$
$\theta_D^2(l_{10})$
$(L_1 == \theta_D^2(l_{10}))?$

Fig. 5. Example of initialization and verification phases.

once, therefore links exchanged during verification must not have been exchanged before.

Right after initialization, the verifier and the device are synchronized on the same chain γ_D and are both aware of the last link that has been used in previous exchanges. Let us call the last exchanged link l_i for simplicity of notation. The verification protocol consists in a simple link exchange that involves the next two links starting from l_{i+1} (not including a sentinel), and can be initiated by both parties. Note that, as anticipated, sentinels must never be exchanged between the verifier and the device for security purposes. In order to support this feature, both parties have to implement an S-modulo counter that is reset after an initialization process and that is used to identify sentinels. In the case of the device, we assume that the counter resides in a secure perimeter together with the register used to keep status information.

When the mutual authentication protocol is initiated by the verifier, the following operations are performed:

1. Being l_i the last synchronized link and σ_0 the first sentinel identified after initialization, the verifier identifies the subsequent link of the chain $\gamma_{D,c_0,M,S,\sigma_0}$ that does not represent a sentinel (let us call it L_0) and sends it to the device D.
2. The device D, whose secure register Q contains the same synchronization link l_i, knows, based on the current value of its counter, if any sentinel has been encountered by the verifier and what value it has to expect in the received message (either $l_{i+1} = \theta_D(l_i)$ or $l_{i+2} = \theta_D^2(l_i)$). Therefore, the device compares L_0 with the proper value to check that what has been received is a valid response. If the comparison succeeds, the device authenticates the verifier and identifies the next link to store in the device secure register Q and to exchange. This will be the first non-sentinel link following the one just used for comparison.
3. The verifier knows, based on the current value of its counter, if the device skipped any sentinel and what value L_1 it has to expect in the received message (either $\theta_D(L_0)$ or $\theta_D^2(L_0)$). Hence, it is able to compare the received value with the right link on the chain and to authenticate the device if they correspond.

Example. In the example shown in Fig. 5, after initialization, both the verifier and the device are synchronized on l_6 and their counters are both set to zero. The counter reset status tells the verifier that the next link l_7 is a sentinel and must be skipped. Hence, $L_0 = l_8$ is selected, and the counter is set to two (i.e., two links have been considered).

On the device side, the counter reset status tells the device that the verifier has encountered a sentinel and that L_0 has to be compared with $\theta_D^2(Q) = \theta_D^2(l_6) = l_8$ and not with $\theta_D(Q) = l_7$. After authenticating the verifier, the device first increases the value of its counter by two (i.e., two links have been considered) and then selects the next link to exchange with the verifier to complete the verification step. In order to do this, the device checks the current value of its counter: since it is different from zero, the link obtained as $\theta_D(L_0) = \theta_D(l_8) = l_9$ is not a sentinel and can be used for the exchange. This value is then stored in the secure register and sent to the verifier, and the device counter is set to three.

The verifier receives the message from the device. Since the current value of the counter is two, the verifier knows that it has to compare the received value with the link that follows L_0 in the chain, namely $L_1 = l_9$. The comparison succeeds and the device is authenticated. The verifier counter is set to three.

Assume a new verification is required at this point, as shown in Fig. 5. The verifier counter is three, therefore the next link, $L_0 = l_{10}$, is not a sentinel and can be exchanged. The counter is updated to four, and it is thus reset.

Since the device counter is not zero, the received value $L_0 = l_{10}$ is compared with $\theta_D(Q) = \theta_D(l_9)$. After authenticating the verifier, the device increases its counter that becomes equal to zero. This value means that the value obtained by computing the PUF response for $\theta_D(l_{10})$ (i.e., l_{11}) is a sentinel and cannot be exchanged. Therefore, the device computes $\theta_D^2(l_{10}) = l_{12}$, which is stored in Q and sent to the verifier.

The verifier receives the message from the device. Since its counter value is zero, the verifier knows that the device has encountered a sentinel. Subsequently, the verifier skips the link that follows L_0 and compares the received value with l_{12}. □

When the authentication request comes from the device, similar operations to those described above are carried out: even in this case in fact, the two parties will exchange couples of subsequent non-sentinel links in order to achieve mutual authentication.

4.4. Chain links exhaustion and de-synchronization issues

As said, the chains associated with a device D are built by the verifier by iteratively applying the self composition of the PUF function embedded within D to a set of initial random challenges during the enrollment phase. Since a chain must not contain loops by definition, the process of link generation for a given chain γ ends whenever a link that was already present in γ or in any other chain extracted for D is found. This condition determines the maximum theoretical length M of each chain but, in real cases, the actual length may be determined by other constraints, such as the maximum available memory space to store the chains at the verifier or the maximum allowed computation time for the enrollment phase.

In any case, chains are a finite collection of links, and hence the above discussed protocols have to take into account their exhaustion since each authentication operation consumes two links and the initialization phase consumes at least 7 links. Luckily, chains are computed and stored only by the verifier, which can easily cope with exhaustion issues by simply triggering a new initialization phase in order to achieve synchronization on a new chain. Clearly, this results in a computation overhead, which will be discussed in Section 5 together with other related issues.

In addition to chain exhaustion, another condition may occur requiring specific attention, namely the de-synchronization of the verifier and the device. De-synchronization basically means that the verifier and the device do not share the same knowledge about the next link to use for protocol operations. De-synchronization might happen due to tampering attempts, temporary unavailability of either device/verifier or transmission channel, or due to a wrong response given by the PUF embedded into the device. In any case, de-synchronization may happen during both the initialization and the verification phases, and eventually results in a re-initialization. We will consider these two conditions separately in the following paragraphs.

De-synchronization during initialization. Based on the initialization procedure described above, three messages are exchanged between the verifier and the device, and three different de-synchronization cases due to message loss may occur depending on which one among msg_1, msg_2 or msg_3 gets lost.

In the first case, the verifier will simply not receive any reply from the device, from whose point of view the protocol has never started. The same applies to the second case, where again the verifier does not receive the expected message. Since none of the two parties has authenticated the counterpart yet, these two cases can be easily solved by re-starting the initialization procedure after the expiration of a timeout.

For what regards the third case, the verifier authenticates the device, but the device is not able to authenticate the verifier as well. This case may be detected by introducing a further *ACK message* to be sent by the device after the phase 4 of the initialization in order to acknowledge the authentication to the verifier. A simple solution would be the set-up of a timer at the verifier to trigger a re-initialization in case the ACK is not received on time. An alternative solution, which does not involve the additional ACK message (that would negatively affect the performance), is to set a timer at the device side right after the sending of msg_2: if no msg_3 is received before the timeout, the device can send an explicit request for re-initialization to the verifier (NACK).

Let us now consider the case of wrong responses generated by the device PUF during either phase 2 or phase 4 of the initialization protocol. In phase 2, the device invokes the embedded PUF two times, one during the verification that involves msg_1 and the other one during the building of msg_2. If any wrong PUF response makes the verification fail, no further operation will be carried out by the device: this situation falls in the second de-synchronization case previously discussed. If instead an error occurs when building msg_2, the verifier will not be able to authenticate the device, and will then trigger a re-initialization. Finally, if a wrong PUF response is generated during phase 4 of initialization, the involved verification step will fail and the device will not authenticate the verifier. If no explicit ACK/NACK is devised, the verifier will detect the de-synchronization only at the first following communication attempt, and will possibly trigger a re-initialization.

De-synchronization during verification. As discussed before, verification involves the exchange of two messages aiming to prove that the parties share the same knowledge on a given chain. When the verifier is the initiator of the verification protocol, two different de-synchronization cases may occur: (i) the message sent by the verifier does not properly reach the device (the verifier is synchronized on l_{i+1}, and the device on l_i); (ii) the message sent by the verifier reaches the device, but the answer of the device does not reach the verifier (the verifier is synchronized on l_{i+1}, and the device on l_{i+2}).

The above cases are indistinguishable from the point of view of the initiator, since the verifier will not receive the expected message from the counterpart. However, in the second case things are complicated by the fact that the mutual authentication is partially achieved by the device. A simple solution is again the set-up of a timer at the verifier and the re-start of the process. Note that, in order to prevent replay attacks, the same link cannot be exchanged more than once. Hence, for each new attempt to execute the verification protocol, the verifier will move ahead in the chain and select a new non-sentinel link. So, for example, at the second attempt, the verifier will send l_{i+3}, then l_{i+5} and so on, up to a maximum number of attempts, after which a re-initialization procedure will be launched. Consequently, the protocol shown in Section 4.3 must be slightly changed to enable the device identify the right link to use in the verification, starting from the value stored in its secure register (and for a maximum number of trials).

When de-synchronization is due to unstable PUF responses, two cases are possible. In the first case, a wrong response may be generated during the comparison with received L_0, resulting in an authentication rejection. No reply would be sent back to the verifier, thus falling in the previously discussed case (ii). In the second case, an error may occur during the computation of the subsequent link to send and store: the device would properly authenticate the verifier, but it would send and store a wrong value, thus compromising not only the current authentication step, but even the future ones. However, the verifier will easily detect the issue and eventually trigger a new initialization.

When the device is the initiator of the protocol, the situation is quite similar to the one just discussed: assuming that the device is provided with a timer, if the message containing the value $\theta_D(l_i)$ is not received by the verifier or the reply gets lost, at the timeout it will send $\theta_D^2(l_i)$, and then $\theta_D^3(l_i)$ and so on, for a maximum number of times. Consequently, as said above, the verification protocol must be slightly changed to enable the verifier identify the right link on the chain (starting from l_{i+1} and for a maximum of trials) that must be checked for correspondence with the value sent by the device. After a maximum number of trials, the device may explicitly request for a new initialization by sending a proper message to the verifier. Alternatively, it may be the verifier to detect the de-synchronization and to trigger the re-initialization.

In presence of PUF instability conditions, two cases may occur. In the first case, a wrong response may be extracted during the generation of the first link to be sent and stored: the authentication at the verifier will fail and a re-initialization will be eventually triggered. In the second case, an error may occur during the verification: the device will not authenticate the verifier and will not update its secure storage. Again, if no explicit ACK/NACK is devised, the verifier will detect the de-synchronization at the first communication attempt and will possibly trigger a re-initialization.

5. PHEMAP analysis

In this section, some remarks on the security and efficiency of the proposed PHEMAP protocol are reported.

5.1. PHEMAP security analysis

As anticipated, PHEMAP was designed to ensure mutual authentication between two communicating parties while preventing *man-in-the-middle* attacks aimed at interfering with the communication by means of the impersonation of either the verifier or a legitimate device.

While the unclonability property of PUFs inherently ensures that malicious devices cannot be legitimately authenticated by the verifier (since they would not be able to produce the PUF responses that have been enrolled), the case when an attacker tries to impersonate the verifier and to be authenticated by the devices is more tricky and is worth a deeper discussion. Therefore, in order to prove the security of the proposed authentication scheme, we formally demonstrate the impossibility, for a malicious verifier, to carry out the initialization phase and the verification phase, respectively.

Theorem 1. *Let D be a legitimate device embedding the PUF $\theta_D(\cdot)$, and let V be a malicious third-party that wants to impersonate the verifier. Even if V has been able to intercept all past exchanged messages between the device D and the legitimate verifier, it is not able to perform the initialization phase to get synchronized with D.*

Proof. W.r.t. initialization, node V may exploit the knowledge of the past communication between the verifier and the device either by replaying some old message or by trying to forge new messages from links previously exchanged (during either initialization or verification). Let us discuss the two cases separately:

Case A: *V replays initialization messages used in previous initialization attempts.* Let us consider a successfully completed initialization session involving messages msg_1, msg_2 and msg_3, built by using the nonces n and m according to the protocol discussed in Section 4.2, and let us assume that V first replays msg_1 to device D. Since D has a limited storage capacity and does not keep history information about previous initialization procedures (except for the last synchronization link, which however is not related to any chain from the point of view of the device), the first step of initialization will succeed. At this point, D will pick a random nonce m', different from the m used in the original msg_2, and will build a msg'_2 as a reply to V. Clearly, msg'_2 will rely upon the same links previously used in msg_2, but will be different from it due to the XOR-ing with nonce m'. Hence, V will not be able to build a proper message msg'_3 and the initialization will not be completed.

Case B: *V tries to forge initialization messages by exploiting past exchanged plain text links or links stored in the device memory*. Let us assume that V, aware of the protocol, has been able to collect a number of subsequent links, exchanged during past initialization and verification steps. Recall that, due to the presence of sentinels (that are never exchanged nor stored in the device memory), V is able to gather at most $S-1$ subsequent links. Since, by definition, initialization requires the knowledge of at least S subsequent links, V will not be able to carry it out successfully.

Let us assume now that, in addition to being able to collect all exchanged messages during a previous initialization, V is also able to leak the content of the device memory right after initialization. Consider the example of Fig. 5, where the last stored value is $l_6 = \theta_D^6(l_0)$. From msg_3, V is able to extract v_3 that has been obtained as $m \oplus l_6$, and can easily compute the nonce m with a XOR operation. By knowing m and msg_2, V is able to obtain also l_5 and l_4, for a total of three subsequent discovered links. However, as for the previous case, V will not be able to take advantage of this knowledge since initialization requires at least S subsequent links and verification involves new links. □

Theorem 2. *Let D be a legitimate device embedding the PUF $\theta_D(\cdot)$, and let V be a malicious third-party that wants to impersonate the verifier. Even if V has been able to intercept all past exchanged messages between the device D and the legitimate verifier, it is not able to get authenticated by the device.*

Proof. Let us assume that D has been previously initialized by the legitimate verifier, and that it is synchronized on the ith link l_i of chain γ. The malicious node V may either attempt to initiate the verification process or it may intercept an authentication request from the device D and impersonate the verifier to achieve mutual authentication. Let us discuss the two cases separately:

Case A: *V is the initiator of the verification.* Since D is synchronized on link l_i, it expects to receive $\theta_D(l_i)$ from V or, alternatively, it expects to receive $\theta_D(l_{i+3})$, $\theta_D(l_{i+5})$ and so on, if de-synchronization issues are managed as suggested in Section 4.4. However, by definition, link l_{i+1} (as well as l_{i+3}, l_{i+5} etc.) has never been exchanged before, nor it can be computed by V, which cannot be authenticated by D.

Case B: *D is the initiator of the verification*. When D is the initiator, it sends $\theta_D(l_i) = l_{i+1}$ to V and expects to receive $\theta_D^2(l_i) = l_{i+2}$ as a reply. However, based on the same motivations stated above, V cannot generate any successive link to the ones already exchanged and will not be able to complete the verification process. □

In addition to preventing *man-in-the-middle* attacks, PHEMAP was also designed to reduce, as much as possible, the leakage of links. An attacker able to read the content of the devices secure register would not be able to infer any information about the embedded PUF and the links that will be subsequently used for authentication, since only already exchanged links or masked links (by means of random nonce) are stored in the secure register. Moreover, new links are not exchanged unless necessary and they are always masked by nonces, thus minimizing the exposure of sensitive information that may be used, among others, to carry out modeling attacks against adopted PUFs. However, as remarked in Section 2, suitable mechanisms exist and may be adopted to mitigate the risk related to modeling attacks, such as Controlled PUFs and CRP hashing. As shown in Section 6, we included such techniques in our PHEMAP implementation.

Finally, with regard to physical security, we can observe that the protocol is quite resilient against Differential Power Analysis (DPA) attacks if a proper bit size is chosen for challenges and responses (e.g., equal to or greater than 128 bits), since it relies upon XOR operations involving entire bit strings represented by chain links and nonces. On the contrary, the adoption of AES encryption to mask links would have made the initialization procedure sensitive to DPA attacks, as AES relies on multiple XOR operations with short binary blocks (8 bits) [10]. Moreover, techniques such as the wave dynamic differential logic discussed in [38] may be successfully adopted in order to make the whole implementation of the protocol controller resilient to side-channel attacks.

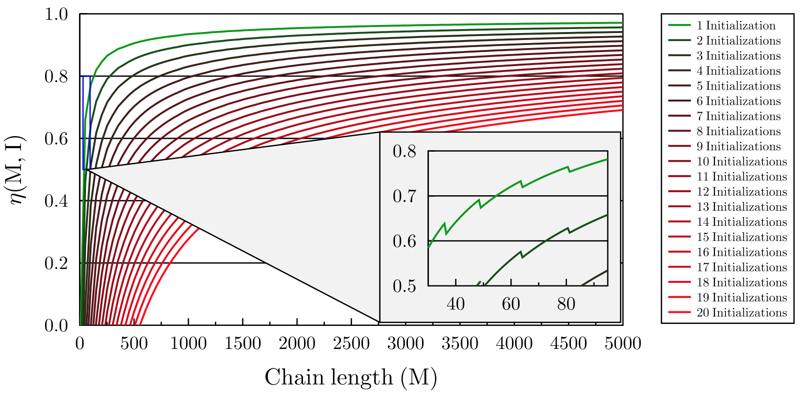


Fig. 6. Values of efficiency when increasing the number M of enrolled links and varying the number or re-initializations.

5.2. PHEMAP efficiency analysis

In order to evaluate the performance of PHEMAP, we conducted both a theoretical and an experimental analysis, aimed at estimating the efficiency of the protocol in terms of resource usage and at measuring the overhead introduced both in terms of memory occupancy and of latency.

For what regards *efficiency*, it is worth recalling that a subset of the generated links is actually wasted, since it is either used during initialization or simply skipped for security purposes (it is the case of sentinels). As pointed out in Section 4.4, chains have a finite length and links are subject to exhaustion, therefore the number of unused links should be minimized, even to reduce, as much as possible, the number of re-initialization operations to carry out.

The period S of sentinels determines the amount of links of a chain that can be used for authentication. Let M be the total number of enrolled links of a chain. The number of links that will be actually exchanged during authentication $M_A(M, S)$ can be obtained by calculating the following equation:

$$M_A(M,S) = M - \left\lfloor \frac{M}{S} \right\rfloor - (S+3)\,. \tag{1}$$

Eq. (1) basically states that, assuming that the initialization procedure has been successfully carried out and that no de-synchronizations occur, $M_A(M, S)$ is determined by subtracting, from the enrolled M links, the links used as sentinels and those used for the initialization procedure.

Such equation has a maximum when $S = \sqrt{M}$. By substituting this value in Eq. (1), it is possible to estimate the maximum of $M_A(M, S)$ and the corresponding maximum efficiency, obtained as the ratio between this value and the total number M of enrolled links of a chain. Eq. (2) reports the maximum efficiency $\eta(M, I)$ when a number I of re-initializations is carried out.

$$\eta(M,I) = \frac{M - \left\lfloor \frac{M}{\sqrt{M}} \right\rfloor - I \times \left(\sqrt{M}+3\right)}{M}. \tag{2}$$

Of course, the efficiency of the protocol and the corresponding overhead are influenced by the performance of the underlying network used to convey the protocol messages. Indeed, in presence of a noisy network with a high packet loss rate, the probability of de-synchronization is non negligible: the number I of re-initialization operations will likely increase with a resulting increase of the authentication latency. Fig. 6 shows the values of efficiency $\eta(M, I)$ when M increases in presence of a varying number I of re-initializations. As expected, each curve asymptotically tends to 1 with a speed that decreases with the increase of M, and the different curves denote a lower efficiency when the number of initializations increases. Clearly, better values of efficiency (i.e., values close to 1) are possible if the number of enrolled links is sufficiently high. To give an example, consider that in order to achieve an efficiency equal to 90.2% (without de-synchronization events), each chain should have at least $M = 484$ enrolled links (with a corresponding sentinel period equal to 22). Luckily, as it will be discussed in the next section, in a real implementation the number of links that it is possible to enroll is sufficiently high to ensure an acceptable value of efficiency.

For what concerns the overhead introduced by PHEMAP compared to the basic authentication scheme discussed in Section 2, it can be measured in terms of both the additional hardware required on the devices in order to support the protocol, and of the time needed to perform the authentication and initialization operations. As for the first point, in addition to the PUF circuit, a device will embed a volatile memory, an S-modulo counter, a random number generator and a minimal control logic to realize the protocol steps.

As for the second point, while the basic authentication scheme devises a single challenge–response algorithm, PHEMAP adds an additional link exchange step to enable the mutual authentication, plus the initialization phase, which however is expected to be performed only occasionally. The single verification step at the device involves the computation of the PUF response for a given challenge, which is straightforward for the device, plus the storage of binary data. For what regards the verifier, which is supposed to have greater capabilities, it has to store the generated chains; even if this is expensive from the storage point of view, it allows to perform the verification operation with a low computational effort since the links can be saved in a hash table.

6. Implementation and experimental evaluation

The proposed mutual authentication mechanism can be based on any strong PUF, provided that it is able to generate a response each time an authentication is needed. This requirement makes unattractive PUFs that generate a response only on the device power-up, such memory-based PUFs. Therefore, in order to practically realize a PHEMAP architecture, we used the Anderson PUF [4], which is a good scalable and reliable PUF available for different Xilinx FPGA families [7,46]. The original Anderson PUF can produce responses on-demand, but it works only as signature generator, so it is not suitable for being adopted in an authentication mechanism. Authors in [22] addressed this issue by proposing an architecture that incorporates one or more challenge bits, known as *enhanced Anderson PUF*. Based on this work and on the one described in [46], we implemented the enhanced Anderson PUF on the Xilinx Zynq-7000 XC7Z020 FPGA, a mid-range family that embeds a dual core ARM processing system and an Artix-7 FPGA fabric [43].

In order to meet the assumptions made in Section 3, we implemented an additional mechanism together with the enhanced

Anderson PUF to make it tamper-resistant and increase its reliability. In particular, we implemented the Controlled PUF mechanism described in [17], which includes a fuzzy extractor to recover the reliability [15]. As for the hashing algorithm, we adopted and implemented *Spongent* [9], which is a lightweight cryptographic hash function that generates a hash digest of 128 bits. Therefore, the implemented PUF has 128 bit-long challenges and responses. With regard to the error correction code (ECC) mechanism, we implemented a BCH [44] scheme tailored to the specific quality parameters of our Anderson PUF, following the methodology given in [20]. It is worth noting in fact that the failure rate of a PUF architecture can be controlled by tuning the ECC parameters, in particular the maximum amount of bits that can be recovered. Hence, we adopted *BCH(63,51)*, which encodes input blocks of 51 bits using 63 bits, and which is capable of detecting and correcting up to two random errors per block. Thanks to this architectural choice, the error probability associated with the implemented Anderson PUF turns out to be less than 10^{-10}.

Finally, the secure non-volatile memory needed by the protocol was implemented as a register residing into the FPGA Programmable Logic, which is a secure perimeter as highlighted by [36], while the control logic of the protocol was implemented through a hardware controller, responsible for generating PUF responses according to the Controlled PUF scheme and for executing the node-side protocol operations.

On the verifier side, the PHEMAP logic was implemented through a software application written in the C++ language, designed according to the client–server paradigm. The verifier application was configured as a server application, able to serve multiple requests coming from the devices and to manage a large database of enrolled chains. It was deployed on a workstation running an Arc Linux distribution, based on the 4.11.3 Linux kernel. The workstation was equipped with a 4 GHz Intel 6700 processor and a 32 GB 3.2 GHz DDR4 memory, and was connected to the Internet by a 10 Gb Ethernet link. As for the chains database, we simply used the file system offered by the operating system, without installing any database server in order to reduce the access overhead.

6.1. Experimental evaluation

In the following, we discuss some experiments carried out with the setup discussed in the previous section, in order to show both (i) the impact of the protocol implementation on the considered devices (in terms of time and resource demand), and (ii) the protocol performance when varying some of its parameters in different working conditions.

Measuring PHEMAP overhead on devices.
The primary operation carried out by a device is the submission of a challenge to the embedded PUF in order to obtain the corresponding response. The time to accomplish this operation, called *PUF querying time*, is influenced by the invocation of the *Spongent* primitive, which is executed two times inside the PUF calculation, since both the challenge and the PUF response have to be hashed according to the Controlled PUF mechanism, as previously discussed. On the other hand, the maximum querying throughput is limited by the saturation phenomenon of the Anderson PUF, which results in having responses biased towards logic-1 [19]. By taking into account this factor, we found that 464 total clock cycles are required to generate a link (cf. Table 1), which means that, by setting a clock frequency of 300 MHz, each PUF querying operation takes less than 2 μs for being completed.

For what regards the *hardware resources* needed to implement the PHEMAP logic on a device, as reported in Table 1, 4342 slice look-up tables (LUTs) and 1265 flip-flops are required in total. They correspond to 8.16% and 1.18% of the available resources, respectively, with the actual overhead of the protocol being 0.76% and 0.2%.

Table 1
PHEMAP overhead measured on Xilinx Zynq-7020 family devices.

	Slice LUTs	Flip-Flops	Clock cycles
Spongent	84	70	70
BCH	290	184	300
Anderson PUF	3936	984	16
PHEMAP Controller	32	27	8
Total	4342 (8.16%)	1265 (1.18%)	464

Analyzing PHEMAP overall performance.
The first reported experiment is aimed at identifying one of the main parameters of the PHEMAP protocol for the selected PUF implementation, namely the average maximum size M of the chains that can be enrolled by the verifier. In order to estimate this parameter, we implemented on the same FPGA ten different instances of the selected Anderson PUF. Then, for each PUF instance, we launched the enrollment of 100 different chains (starting from different root values) and recorded the corresponding size. In all the cases, more than 1.5×10^9 subsequent links on a single chain were generated before exhausting the verifier memory, without experiencing any collision. This was expected since the implemented Controlled PUF scheme and the Anderson PUF architecture itself, are able to imprint a very high entropy on PUF responses. This result suggests that, in real implementations, the achieved value of M enables to obtain high levels of efficiency (cf. Section 5.2). The actual value of M chosen for the protocol can be set according to other constraints, such as the desired mission time of the device under enrollment, the required authentication throughput, the communication channel reliability and the storage capability of the server.

As outlined in Section 5, the PHEMAP initialization phase is critical from the point of view of the protocol efficiency, since it not only causes a waste of chain links, which cannot be used for actual authentication operations, but it also impacts on the overall latency, especially in case of de-synchronization situations that require multiple executions of this phase. In order to evaluate this impact, we carried out two sets of experiments aimed at measuring the overall initialization latency in presence of an increasing packet loss rate and of an increasing sentinel period, respectively.

As for the first set of experiments, we considered the worst-case de-synchronization situation discussed in Section 4.4, in which the last protocol message (sent by the verifier) is lost. As discussed, this situation can be handled by taking into account an explicit NACK message sent by the device after a timeout expiration, which triggers a re-initialization. For our experiments, in order to select a reasonable timer duration, we measured the average round trip time *RTT* between the device and verifier and set the timer to $10 \times RTT$. We considered a packet loss probability varying uniformly in [0, 0.25] and carried out several experiments to measure the average latency of the initialization protocol (i.e., the time taken to complete an initialization process) over an IP network. The probability range was fixed by taking into account the fact that more than 25% of packet loss is unacceptable for any kind of application [32], therefore it made no sense to consider higher probabilities. Given the interval [0, 0.25], we picked different values of probability in such interval (with a step of 0.01), and ran one thousand initialization procedures for each loss probability value, measuring the resulting latency values and computing their average.

Latency was measured as the total amount of time spent to complete one initialization operation, including the protocol operations executed locally by the verifier and the device, the network transmission time, and the possible time spent waiting for the

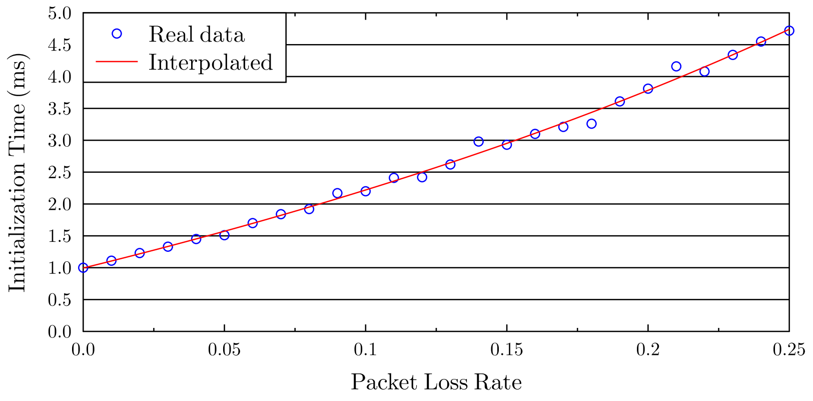


Fig. 7. Values of latency of the initialization protocol in presence of a varying packet loss rate.

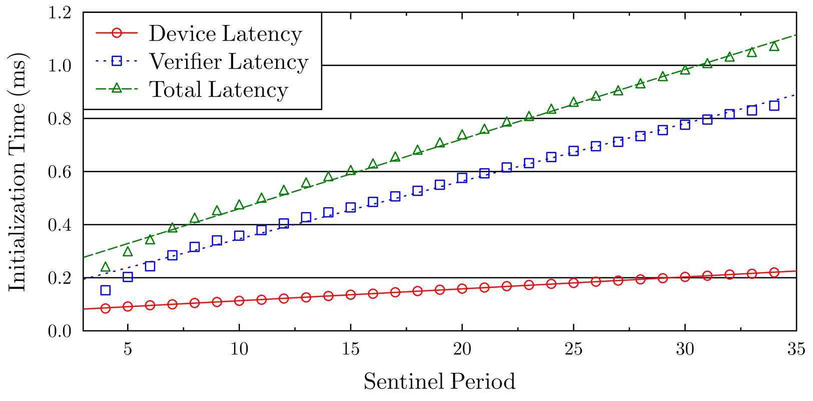


Fig. 8. Values of latency of the initialization protocol in presence of a varying period of sentinels.

timer expiration at the device when a message gets lost. Fig. 7 reports the average values of latency resulting from these experiments when increasing the packet loss probability as discussed. As shown, the measured trend has a hyperbolic behavior, which tends to infinite values of time when the packet loss rate approaches 1.

For what regards the second set of experiments, we analyzed the latency introduced by initialization operations at both the verifier and device side when varying the sentinel period, in order to evaluate the impact of this parameter on the complexity of the operations to perform. The experiment results are reported graphically in Fig. 8, which shows that the latency exhibits a linear behavior as the sentinel period grows. Moreover, as one can notice, the curve that represents the device latency (red line with circle markers) grows slower than the curve associated with the verifier latency (blue line with square markers). This difference can be explained by recalling that the verifier has to access the file system in order to recover the links involved in initialization (by means of a sequential access strategy), while the device simply queries the PUF circuit.

7. Conclusions and final remarks

Security is increasingly becoming a critical factor of embedded system design, since cryptography is often not enough to protect systems due to many physical attacks that are able to extract secret keys from devices. In this context, PUFs represent a breakthrough thanks to their unclonability, uniqueness and tamper-evident properties that allow to obtain, with a relatively low overhead, an excellent randomness source that can be successfully used for authentication, digital signature and key storage purposes.

Recently, several research efforts have been devoted to defining new PUF architectures, with particular focus on performance and quality, while only few secure schemes are currently available based on this technology. Our contribution was aimed at investigating the potential of PUFs as security enablers by means of the introduction of PHEMAP, a PUF-based mutual authentication protocol designed to provide mutual authentication in a one-to-many communication scenario.

In this paper, we formalized the mutual authentication problem and provided a detailed discussion of the proposed protocol, by illustrating the possible operational conditions and the related degenerate cases. In order to evaluate the effectiveness of the protocol, we carried out a qualitative security analysis, by demonstrating that it is able to thwart *man-in-the-middle* attacks and other common physical attacks. Moreover, we discussed the efficiency of the protocol in terms of resource usage and conducted several experiments, based on both simulation and measurements on a real implementation, aimed at evaluating the hardware overhead due to the implementation of the PHEMAP controller components and to the latency introduced by the protocol phases. The experimental results showed that the overhead introduced by the approach makes PHEMAP feasible to be implemented on a commercial mid-range device.

It is worth noting that, in the proposed solution, the verifier may represent a bottleneck and single-point-of-failure, since it has to manage every enrolled PUF chain. Although the existing performance and reliability issues may be tackled by resorting, for example, to cloud-based services, it may be interesting in a future work to investigate the introduction of a federated/distributed verification scheme that would help relieve the burden for the verifier.

7.1. Other applications of PHEMAP

Moreover, even if the application scenario discussed in this paper relates to the mutual authentication between two communicating parties, the PHEMAP concepts can be used even in other domains. For instance, since links are obtained as PUF responses, they are eligible to be used as cryptographic keys. Hence, two parties may exploit PHEMAP to synchronize the keys to be used in crypto-schemes, in order to guarantee confidentiality in communication. Moreover, in a more advanced scenario, the PHEMAP protocol may be used to implement a Moving Target Defense (MTD) approach consisting in continuously changing the cryptographic keys used for encrypting the communications. This would expose a changing attack surface to an attacker aimed at violating confidentiality and would increase the overall security level of the system [11,23].

In other application contexts, such as radio-frequency identification, PHEMAP can be used in a stateful configuration, i.e., by substituting the volatile memory with a non-volatile one. This would lead to a single initialization step.

References

[1] Y. Alkabani, F. Koushanfar, Active control and digital rights management of integrated circuit IP cores, in: Proceedings of the 2008 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, ACM, 2008, pp. 227–234.

[2] M.N. Aman, K.C. Chua, B. Sikdar, Physical unclonable functions for IoT security, in: Proceedings of the 2Nd ACM International Workshop on IoT Privacy, Trust, and Security, IoTPTS '16, ACM, New York, NY, USA, 2016, pp. 10–13. http://dx.doi.org/10.1145/2899007.2899013.

[3] D. Amelino, M. Barbareschi, E. Battista, A. Mazzeo, How to manage keys and reconfiguration in WSNs exploiting SRAM based PUFs, in: Intelligent Interactive Multimedia Systems and Services 2016, Springer International Publishing, 2016, pp. 109–119.

[4] J.H. Anderson, A PUF design for secure FPGA-based embedded systems, in: Proceedings of the 2010 Asia and South Pacific Design Automation Conference, IEEE Press, 2010, pp. 1–6.

[5] M. Barbareschi, Notions on Silicon Physically Unclonable Functions, Springer International Publishing, Cham, 2017, pp. 189–209. http://dx.doi.org/10.1007/978-3-319-44318-8_10.

[6] M. Barbareschi, P. Bagnasco, A. Mazzeo, Authenticating IoT devices with physically unclonable functions models, in: 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2015, pp. 563–567. http://dx.doi.org/10.1109/3PGCIC.2015.117.

[7] M. Barbareschi, P. Bagnasco, A. Mazzeo, Supply voltage variation impact on anderson PUF quality, in: 2015 10th International Conference on Design Technology of Integrated Systems in Nanoscale Era, DTIS, 2015, pp. 1–6. http://dx.doi.org/10.1109/DTIS.2015.7127361.

[8] M. Bhargava, K. Mai, An efficient reliable puf-based cryptographic key generator in 65 nm cmos, in: 2014 Design, Automation Test in Europe Conference Exhibition, DATE, 2014, pp. 1–6. http://dx.doi.org/10.7873/DATE.2014.083.

[9] A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varıcı, I. Verbauwhede, Spongent: A lightweight hash function, in: Cryptographic Hardware and Embedded Systems–CHES 2011, Springer, 2011, pp. 312–325.

[10] E. Brier, C. Clavier, F. Olivier, Correlation power analysis with a leakage model, in: Cryptographic Hardware and Embedded Systems-CHES 2004, Springer, 2004, pp. 16–29.

[11] V. Casola, A. De Benedictis, M. Albanese, A multi-layer moving target defense approach for protecting resource-constrained distributed devices, Adv. Intell. Syst. Comput. 263 (2014) 299–324. http://dx.doi.org/10.1007/978-3-319-04717-1_14.

[12] A. Cherkaoui, L. Bossuet, C. Marchand, Design, evaluation, and optimization of physical unclonable functions based on transient effect ring oscillators, IEEE Trans. Inf. Forensics Secur. 11 (6) (2016) 1291–1305. http://dx.doi.org/10.1109/TIFS.2016.2524666.

[13] A. Cilardo, M. Barbareschi, A. Mazzeo, Secure distribution infrastructure for hardware digital contents, IET Comput. Digit. Tech. 8 (6) (2014) 300–310. http://dx.doi.org/10.1049/iet-cdt.2014.0036.

[14] A. Cilardo, N. Mazzocca, Exploiting vulnerabilities in cryptographic hash functions based on reconfigurable hardware, IEEE Trans. Inf. Forensics Secur. 8 (5) (2013) 810–820. http://dx.doi.org/10.1109/TIFS.2013.2256898.

[15] Y. Dodis, L. Reyzin, A. Smith, Fuzzy extractors: How to generate strong keys from biometrics and other noisy data, in: Advances in Cryptology-Eurocrypt 2004, Springer, 2004, pp. 523–540.

[16] K.B. Frikken, M. Blanton, M.J. Atallah, Robust authentication using physically unclonable functions, in: Information Security, Springer, 2009, pp. 262–277.

[17] B. Gassend, D. Clarke, M. van Dijk, S. Devadas, Controlled physical random functions, in: Proceedings of the 18th Annual Computer Security Applications Conference, ACSAC '02, IEEE Computer Society, Washington, DC, USA, 2002, p. 149. http://dl.acm.org/citation.cfm?id=784592.784802.

[18] B. Gassend, M. van Dijk, D. Clarke, S. Devadas, Controlled physical random functions, in: Security with Noisy Data, Springer, 2007, pp. 235–253.

[19] P. Grabher, D. Page, M. Wójcik, On the (re) design of an FPGA-based PUF, IACR Cryptology EPrint Archive 2013 (2013) 195.

[20] J. Guajardo, S.S. Kumar, G.-J. Schrijen, P. Tuyls, FPGA Intrinsic PUFs and their Use for IP Protection, Springer, 2007.

[21] D.E. Holcomb, W.P. Burleson, K. Fu, Power-up SRAM state as an identifying fingerprint and source of true random numbers, IEEE Trans. Comput. 58 (9) (2009) 1198–1210. http://dx.doi.org/10.1109/TC.2008.212.

[22] M. Huang, S. Li, A delay-based PUF design using multiplexer chains, in: 2013 International Conference on Reconfigurable Computing and FPGAs, ReConFig, 2013, pp. 1–6. http://dx.doi.org/10.1109/ReConFig.2013.6732258.

[23] S. Jajodia, A.K. Ghosh, V. Swarup, C. Wang, X.S. Wang, Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats, vol. 54, Springer Science & Business Media, 2011.

[24] F. Kastensmidt, J. Tonfat, T. Both, P. Rech, G. Wirth, R. Reis, F. Bruguier, P. Benoit, L. Torres, C. Frost, Voltage scaling and aging effects on soft error rate in SRAM-based FPGAs, Microelectronics Reliability.

[25] P. Kocher, Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, in: Advances in Cryptology –CRYPTO'96, Springer, 1996, pp. 104–113.

[26] P. Kocher, J. Jaffe, B. Jun, Differential power analysis, in: Advances in Cryptology –CRYPTO'99, Springer, 1999, pp. 388–397.

[27] S.S. Kumar, J. Guajardo, R. Maes, G.J. Schrijen, P. Tuyls, Extended abstract: The butterfly PUF protecting IP on every FPGA, in: 2008 IEEE International Workshop on Hardware-Oriented Security and Trust, 2008, pp. 67–70. http://dx.doi.org/10.1109/HST.2008.4559053.

[28] D. Lim, J.W. Lee, B. Gassend, G.E. Suh, M. van Dijk, S. Devadas, Extracting secret keys from integrated circuits, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 13 (10) (2005) 1200–1205. http://dx.doi.org/10.1109/TVLSI.2005.859470.

[29] R. Maes, P. Tuyls, I. Verbauwhede, Intrinsic PUFs from flip-flops on reconfigurable devices, in: 3rd Benelux Workshop on Information and System Security, WISSec 2008, vol. 17, 2008.

[30] R. Maes, I. Verbauwhede, Physically unclonable functions: A study on the state of the art and future research directions, in: Towards Hardware-Intrinsic Security, Springer, 2010, pp. 3–37.

[31] A. Maiti, I. Kim, P. Schaumont, A robust physical unclonable function with enhanced challenge-response set, IEEE Trans. Inf. Forensics Secur. 7 (1) (2012) 333–345. http://dx.doi.org/10.1109/TIFS.2011.2165540.

[32] K.C. Mansfield, J.L. Antonakos, Computer networking from lans to wans: Hardware, Software, and Security, Course Technology, Cengage Learning, Boston.

[33] R. Pappu, B. Recht, J. Taylor, N. Gershenfeld, Physical one-way functions, Science 297 (5589) (2002) 2026–2030.

[34] M. Rostami, M. Majzoobi, F. Koushanfar, D.S. Wallach, S. Devadas, Robust and reverse-engineering resilient PUF authentication and key-exchange by substring matching, IEEE Trans. Emerg. Top. Comput. 2 (1) (2014) 37–49. http://dx.doi.org/10.1109/TETC.2014.2300635.

[35] U. Rührmair, J. Sölter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, S. Devadas, PUF modeling attacks on simulated and silicon data, IEEE Trans. Inf. Forensics Secur. 8 (11) (2013) 1876–1891. http://dx.doi.org/10.1109/TIFS.2013.2279798.

[36] L. Sanders, Secure Boot of Zynq-7000 All Programmable SoC (2013). https://www.xilinx.com/support/documentation/application_notes/xapp1175_zynq_secure_boot.pdf.

[37] G.E. Suh, S. Devadas, Physical unclonable functions for device authentication and secret key generation, in: Proceedings of the 44th Annual Design Automation Conference, ACM, 2007, pp. 9–14.

[38] K. Tiri, I. Verbauwhede, A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation, in: Proceedings of the Conference on Design, Automation and Test in Europe, vol. 1, IEEE Computer Society, 2004, p. 10246.

[39] V. van der Leest, G.-J. Schrijen, H. Handschuh, P. Tuyls, Hardware intrinsic security from D flip-flops, in: Proceedings of the Fifth ACM Workshop on Scalable Trusted Computing, ACM, 2010, pp. 53–62.

[40] A. Van Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A.-R. Sadeghi, I. Verbauwhede, C. Wachsmann, Reverse fuzzy extractors: Enabling lightweight mutual authentication for PUF-enabled RFIDs, in: Financial Cryptography and Data Security, Springer, 2012, pp. 374–389.

[41] E.I. Vatajelu, G.D. Natale, M. Barbareschi, L. Torres, M. Indaco, P. Prinetto, STT-MRAM-based PUF architecture exploiting magnetic tunnel junction fabrication-induced variability, ACM J. Emerg. Technol. Comput. Syst. 13 (1) (2016) 5.

[42] V. Vivekraja, L. Nazhandali, Circuit-level techniques for reliable physically uncloneable functions, in: 2009 IEEE International Workshop on Hardware-Oriented Security and Trust, 2009, pp. 30–35. http://dx.doi.org/10.1109/HST.2009.5225054.

[43] Xilinx Inc. Zynq-7000 All programmable SoC overview, advance product specification-ds190 (v1.11). June 7, 2017. https://www.xilinx.com/support/documentation/data_sheets/ds190Zynq7000Overview.pdf.
[44] M.-D.M. Yu, S. Devadas, Secure and robust error correction for physical unclonable functions, IEEE Des. Test Comput. 27 (1) (2010) 48–65.
[45] M.-D.M. Yu, M. Hiller, J. Delvaux, R. Sowell, S. Devadas, I. Verbauwhede, A lockdown technique to prevent machine learning on pufs for lightweight authentication, IEEE Trans. Multi-Scale Comput. Syst. 2 (3) (2016) 146–159.
[46] J. Zhang, Q. Wu, Y. Lyu, Q. Zhou, Y. Cai, Y. Lin, G. Qu, Design and implementation of a delay-Based PUF for FPGA IP protection, in: 2013 International Conference on Computer-Aided Design and Computer Graphics, 2013, pp. 107–114. http://dx.doi.org/10.1109/CADGraphics.2013.22.

Mario Barbareschi received the Ph.D. in Computer and Automation Engineering in 2015 and the Master Degree in Computer Engineering cum laude in 2012, both from the University of Naples Federico II, where he is currently working as a post-doctoral fellow. His research interests include Hardware Security and Trust, Cyber–Physical Security, Approximate Computing, memristor-based architectures and embedded systems based on the FPGA technology. He has authored more than 30 peer-reviewed papers published in leading journals and international conferences.

Alessandra De Benedictis received the Ph.D. degree in computer and automation engineering in 2013. She is a post-doctoral fellow at the Department of Electrical Engineering and Information Technology of the University of Naples Federico II. She has been working on the design and evaluation of secure architectures for the protection of distributed resource-constrained devices and she has recently started working on the negotiation and dynamic enforcement of security in clouds through the adoption of service level agreements.

Nicola Mazzocca received the M.Sc. degree in electronic engineering and the Ph.D. degree in computer engineering, both from the University of Naples Federico II, Italy. He is currently a Professor of High-Performance and Reliable Computing at the Computer and System Engineering Department of the University of Naples Federico II. He authored over 200 papers in international journals, books, and international conferences in the field of computer architecture, reliable and secure systems, distributed systems, and performance evaluation in high-performance systems.