Journal of Ambient Intelligence and Smart Environments 1 (2021) 1–5
IOS Press


# Refillable PUF Authentication Protocol for Constrained Devices

Arthur Desuert [a], Stéphanie Chollet [a,*], Laurent Pion [a] and David Hély [a]
[a] *LCIS, Université Grenoble Alpes, Grenoble, France*
*E-mails: arthur.desuert@grenoble-inp.fr, stephanie.chollet@grenoble-inp.fr, laurent.pion@grenoble-inp.fr, david.hely@grenoble-inp.fr*

**Abstract.** Connected devices are deployed at a rapid rate and in broad domains like home automation or industry, forming the Internet of Things. Those devices need to be secure and trusted to prevent malicious use. However some connected devices are low-cost, memory constrained, power constrained, etc.. This greatly limits the deployment of usual security solutions. As the absence of security is not acceptable, it is necessary to search for lightweight security solutions adapted to such devices. Physical Unclonable Function (PUF) technology can support new lightweight security mechanisms and several lightweight security protocols using this technology have been proposed over the years. Those protocols look promising, however there are still some unaddressed challenges which have slowed down a large scale adoption. This article presents the design of a new authentication protocol for constrained devices which takes into account those challenges. This protocol is implemented on a hardware platform used for connected devices development, which is then used to evaluate the security level and performances of the protocol in a realistic scenario. This evaluation shows that the protocol is secure and can meet industrial time constraints.

Keywords: Internet of Things (IoT), Physical Unclonable Function (PUF), security, authentication

## 1. Introduction

With the past and expected growth of the Internet of Things (IoT) and its applications [1], connected devices are largely available today. Those devices gather data from their surroundings and exchange them with remote platforms in the fog and/or in the cloud for further processing. Moreover, some devices are able to physically interact with their environment. Those devices capabilities can be used by pervasive applications to provide services to users while remaining invisible. In fact, those applications extend software systems into the physical world to offer smart environments (*i.e.*, at the same time ubiquitous, ambient, seamless and transparent) [2].

Pervasive applications work in close proximity to the physical world and potentially to human users, with expected deployment of those applications in domains such as smart-home, e-Health or Industry 4.0. Consequently, it is vital to integrate security considerations into the development process to ensure trust in pervasive environments, as any malfunction or misbehavior would not only disappoint the final user [2] but could lead to hazardous or harmful situations. An important security aspect is to guarantee the authenticity of connected devices from their manufacturing to their deployment as part of a smart environment and to allow this authenticity to be checked by pervasive applications working with the devices. This would avoid the intrusion of counterfeit or rogue devices which could perturb the operation of pervasive applications.

*Corresponding author. E-mail: stephanie.chollet@grenoble-inp.fr.

To authenticate IoT devices, it is possible to use protocols and standards commonly deployed on the Internet like HTTP[1], MQTT[2], AMQP[3], CoAP[4], etc. Most of these protocols use the Transport Layer Security (TLS) protocol [3] or the Datagram Transport Security Layer (DTLS) [4] to secure communications. They are usually associated to the Public Key Infrastructure (PKI) which manages authentication with X.509 digital certificates and asymmetric cryptography [5, 6]. Those solutions have already been deployed experimentally for pervasive applications [7], they bring a good security level but they are not adapted to some devices as they have a non negligible memory footprint, involve heavy computations, require the secure storage of long-term secrets and are costly to deploy. They are well suited for high-end devices, but not for low-cost and constrained devices. Complementary solutions are needed to perform lightweight authentications with a correct security level.

To tackle this issue, Physical Unclonable Functions (PUF) are a promising technology. First introduced by Pappu [8], a PUF is a hardware system which generates a unique fingerprint when it is manufactured. Like a function, a PUF accepts an input and generates a corresponding output which is influenced by the PUF fingerprint. This allows the identification of a PUF instance. PUF are interesting because they act as a secure storage for their fingerprint while using more simple circuits than dedicated storage solutions. It is very hard to duplicate a given PUF because its fingerprint is generated by unpredictable manufacturing noise. For those reasons, researches are conducted on this technology. One research area is the use of PUF to develop lightweight security protocols compatible with constrained devices, identification and authentication protocols for instance. Yet the domain is vast and among the various propositions, some challenges are still rarely addressed yet they are crucial to ensure the deployment in commercial solutions.

This article examines those challenges and presents the design of a new PUF-based authentication protocol for constrained devices. The protocol key features are the clear definition of involved actors, its integration into the device life-cycle from manufacturing to field deployment and its compatibility with industrial time constraints thanks to a new approach in the management of authentication data.

This article is organized as it follows: Section 2 presents current authentication schemes and their limitations in the context of low-cost constrained devices, then introduces the base principle and open challenges in using PUF technology for lightweight authentication. Section 3 presents the global approach to address the identified challenges, approach which led to the design of a new protocol. Section 4 explains how the protocol has been implemented on a hardware target and the design choices. Before the conclusion, Section 5 presents the evaluation of the protocol security level and performance metrics.

## 2. Problem definition

### *2.1. Authentication in IoT context*

Authentication of smart devices is a hot topic and a very active research field, particularly when the authentication takes place between a low-cost resource-constrained device and a resource-rich server, as highlighted by the plurality of proposed protocols [9]. The devices constraints make the implementation of cryptographic primitives and security functions a challenging task. However, those constraints are rarely quantified which makes it difficult to clearly grasp those limits. In an effort to clarify this situation, this article uses the terminology defined by the Internet Engineering Task Force (IETF) for constrained devices in the RFC 7228 [10]. This RFC introduces three classes of constrained devices based on their embedded storage for code and data:

- **Class 0 (C0)** is the most constrained class with less than 100 KiB of code and less than 10 KiB of data,
- **Class 1 (C1)** is an intermediate level with around 100 KiB of code and around 10 KiB of data,
- **Class 2 (C2)** is the less constrained class with around 250 KiB of code and 50 KiB of data.

[1] https://datatracker.ietf.org/doc/html/rfc7231
[2] https://mqtt.org/
[3] https://www.amqp.org/
[4] https://datatracker.ietf.org/doc/html/rfc7252

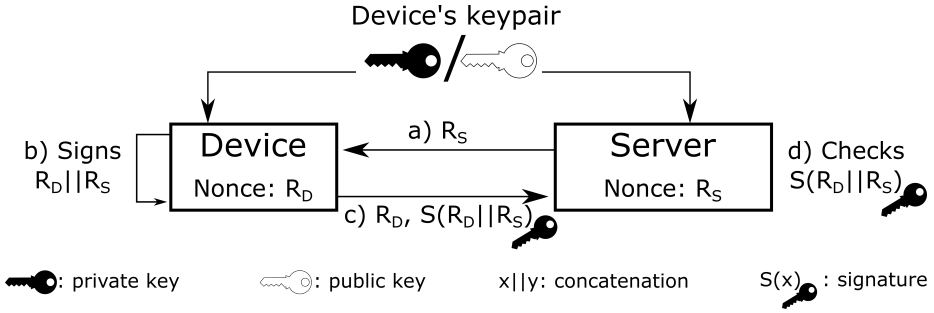


Fig. 1. One way authentication with asymmetric keys.

A device embedding more storage capabilities than class 2 limits is not considered constrained in terms of storage, an example being the Raspberry Pi boards[5]. However, such device can be constrained by limited energy supply. The RFC defines classes of energy limitation defined by the type of limitation the device has to deal with. The main target of this article is class 0 devices as they are the ones in need of lightweight solutions for authentication. Before presenting such solutions, two classical authentication methods are presented to understand their incompatibility with constrained devices.

The first method relies on asymmetric cryptography with the Public Key Infrastructure (PKI). Asymmetric cryptography works with key-pairs composed of a private key and a public key. The PKI is a set of norms, infrastructures and processes which aim at ensuring trust in public keys. To do so, additional data are associated to public keys thank to standards like X.509 certificates [5]. For each device to authenticate, a key-pair is generated. The private key is securely embedded in the device while the public key is inserted in a certificate describing the device identity. In this setup, authentication protocols have been normalized by the National Institute of Standards and Technology (NIST) [11]. A simplified version of a one-way authentication is illustrated by Fig. 1.

In Fig. 1, the server generates a random number used once (*nonce*) $R_S$ and sends it to the device (a). When receiving $R_S$, the device generates a random *nonce* $R_D$ and signs the concatenation ($||$) of both numbers with its private key (b). The device then sends back $R_D$ and the signature (c). It is assumed that the server recovers the device certificate which contains the public key and checks its validity using an appropriate way, as those steps are not illustrated in this example. Finally, using the nonce of the device and its public key, the server can check the signature and validate or not the authentication. This method is scalable because each device is assigned a private key and the associated public key can be distributed on large scales using the dedicated infrastructures of the PKI. However, asymmetric cryptography relies on heavy computations which is often incompatible with constrained devices in terms of power or timing. This method implies non negligible deployment costs, to either set up a private infrastructure or register to an existing one.

The second authentication method uses symmetric cryptography. The principle is similar to the previous method: for each device which needs authentication, a symmetric key is generated and embedded in the device in a secure manner. This key needs to be shared with servers willing to authenticate the smart device. A server can then send a challenge to the device which ciphers it with the shared key and sends it back for authentication. Symmetric cryptography operations are less costly in terms of computations than asymmetric operations, but the challenging part is sharing the secret key between the device and the server: it is possible to use a single key for several devices and servers but if only one device is compromised the whole network becomes vulnerable. A more secure way is to assign a unique key to every pair of device and server which need to communicate, but this requires additional management. In the end, the deployment costs are non negligible just like the asymmetric method.

Additionally, in both methods the smart device needs to securely store some data: either its private key or the shared secret key. Several solutions exist to ensure the secure storage of such sensitive data. They all come with different security levels and different costs. For instance, there are dedicated microprocessor chips called Secure Element (SE). A SE is specifically designed to store confidential data and to realize some cryptographic operations. It

[5]https://www.raspberrypi.org/

includes sensors to detect intrusion attempts and react accordingly, by erasing the internal storage for instance. It is designed to be tamper proof against hardware attacks such as power side channel [12] or fault injection attacks [13]. Those security properties are often qualified by a certification authority and must be compliant with security standards, such as the Common Criteria (CC) for instance. While with SE, one can reach a very high security level, it induces an extra costs to the system in terms of components and engineering [14].

Another solution for secure storage is to use a MicroController Unit (MCU) with embedded security features like a secure context, which splits the MCU in two distinct parts:

- a *normal context*, also called Rich Execution Environment (REE), where the main applications can run and interact with the outside world;
- and a *secure context*, also called Trusted Execution Environment (TEE), where sensitive operations can be executed and secrets securely stored with very limited interactions with the outside world.

Context switches can only happen in well-defined entry/exit points to prevent illegal accesses. This solution can be easier to implement but it is less secure than a SE because it does not always include anti-tampering protections [14].

Finally, combining a classical authentication method with a secure storage solution is often not viable for low-cost constrained devices. To tackle this issue, new technologies are being considered like Physical Unclonable Functions (PUF) [15].

### 2.2. PUF technology for authentication

A Physical Unclonable Function (PUF) is a hardware circuit which reacts to an input stimulus named the challenge by emitting an output stimulus named the response. The originality of a PUF is that its output not only depends on the challenge but on some intrinsic properties of the hardware circuits as well, such as the geometry of its transistors or the size of the hardware structure. Because those properties mainly depend on manufacturing process variations, it is statistically highly unlikely or almost impossible to produce identical PUF, hence its unclonable property.

PUF have others characteristics. First, there is the challenge space accepted by the PUF. It classifies PUF in two categories: *weak PUF* and *strong PUF*. A PUF is classified as *strong* if it has a large Challenge-Response Pairs (CRP) space, qualified as non-enumerable, and this CRP space scales exponentially with the PUF design size. Otherwise it is a *weak PUF* which has a limited CRP space, sometimes made up of a unique CRP. Another metric of interest is the error rate, to quantify the stability of PUF responses. This metric must be as low as possible to minimize the correction operations needed to have a stable response to a given challenge. Inter-PUF distance is critical as well when using PUF for authentication purposes. It consists of measuring, for a given challenge, the distance between several PUF responses. This allows to check the absence of collision between PUF responses which could lead to authentication issues. This metric must be close to 50%, as it represents uniform use of the response space.

They are several ways and technologies to create a PUF, depending on the integration constraints and required properties. The first PUF design, presented by R. Pappu in 2001 [8], used a laser and a specially crafted diffracting material. It was a strong PUF but it was quite challenging to integrate in electronic systems. Then, B. Gassend *et al.* presented in 2002 a silicon PUF [16], which is created using integrated circuits, making them much easier to interface with existing hardware. Silicon PUF can be split in two main categories: *delay-based PUF* and *memory-based PUF*. Delay-based PUF are based on signal propagation through circuit. Arbiter PUF [16] is an example of this type of PUF. It is a strong PUF but it requires very precise circuit design methodology to ensure optimal operation of the PUF. Memory-based PUF rely on memory circuits, like the SRAM PUF [17]. The main advantage is the use of already existing and available hardware to implement those PUF, but unfortunately most memory-based PUF are weak PUF which are less practical than strong PUF regarding authentication.

PUF circuits have some intrinsic security properties: first, their unclonability which guarantees an almost-nil risk of duplication. Then, they are often resistant to direct tamper attacks which can perturb the PUF operation, thus modifying its responses and preventing any data leak. Finally, PUF are active circuits, so no information is available when the device or the PUF is powered down, as opposed to non-volatile memories storing sensitive data. All those properties make PUF circuits good candidates for a secure storage or a secure secret generator with a lower

integration cost than the previously introduced solutions in Section 2.1. This secure function can be exploited in the design of low-cost authentication protocols offering a sufficient security level. A simple example of a PUF-based authentication procedure using Challenge-Response Pairs (CRP) is presented by G. E. Suh and S. Devadas in [18] and illustrated by Fig. 2.

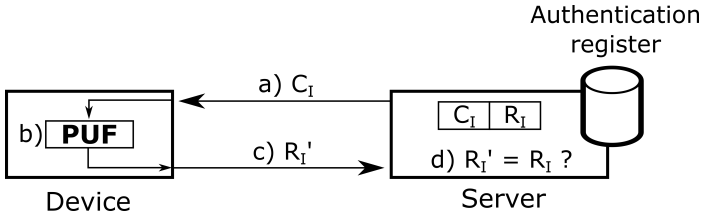


Fig. 2. Authentication procedure with previously initialized register.

The authentication register contains several CRP which were stored during a previous operation called the registration. To authenticate the device, the server selects a challenge in the authentication register and sends it to the device (a). The device presents the received challenge to its internal PUF (b) and sends the corresponding response back to the server (c). Finally, the server checks if the received response corresponds to the stored response in the register and validates or not the authenticity of the device (d).

Lightweight authentication using PUF technology is an active research field as shown by the plurality of proposed protocols in the past decade [19]. However, some challenges remain mostly unaddressed and constitute an obstacle to large-scale deployment in commercial and industrial products. Those challenges include:

- **the integration into the device life-cycle**. It is crucial to know when each phase has to be performed, particularly the registration phase which establishes trust in the device.
- **the management of the authentication register**. It is a critical equipment with the sensitive task of first registering devices in a secure environment and then ensuring the security and availability of those devices authentication data.
- **the management of authentication data**. This mostly concerns protocols which progressively delete authentication data, like Suh and Devadas' protocol. Such protocols need to address the case of a total exhaustion of authentication data. Moreover, such protocols need to gather enough authentication data to cover the whole device lifespan, which impacts the registration phase.

## 3. Methodology

### *3.1. Global approach*

Connected devices are becoming more common in several environments of human users: their home, their workplace, their city, etc. Pervasive applications rely on those devices to collect data or even to act upon the physical world to provide services to those users. It is then crucial to ensure that those devices can be trusted. A first step toward this goal is the authentication of devices. In this article, it is assumed that for each environment there is an equipment dedicated to the management of connected devices located in the fog [20]. This equipment is called gateway in the rest of the article. This gateway acts as an intermediary between devices and pervasive applications, with the ability to autonomously authenticate devices. Focusing on constrained devices, it is assumed that each device contains a PUF and each manufacturer of constrained devices manages an authentication register linked with those PUF. Based on those assumptions, this article proposes a new lightweight protocol, based on PUF, which enables mutual authentication between a gateway and a constrained device. Moreover, this protocol takes into account the identified limitations of previous protocols. In particular, the protocol is integrated into the device life-cycle; it clearly defines the entity in charge of the authentication register management and it offers a secure procedure to generate new authentication data while the device remains integrated in its environment.

This protocol includes three elements as illustrated in Fig. 3: the constrained device embedding a PUF, the users' environment gateway and the manufacturer's authentication register. The protocol features three phases integrated into the device life-cycle:

- **registration**: it is an initialization phase realized right after the device manufacturing. The goal is to gather authentication data using the embedded PUF of the device. After this phase, the device is ready to be integrated in the users' environment and used by pervasive applications.
- **authentication**: While a device is deployed in the users' environment and a pervasive application needs to interact with it, the device and the gateway exchange previously gathered authentication data in order to mutually authenticate.
- **secure refill**: This phase is triggered autonomously when a lack of authentication data is detected. The device and the gateway mutually authenticate, then open a secure channel, allowing the gateway to gather new authentication data from the device while it remains in the users' environment.

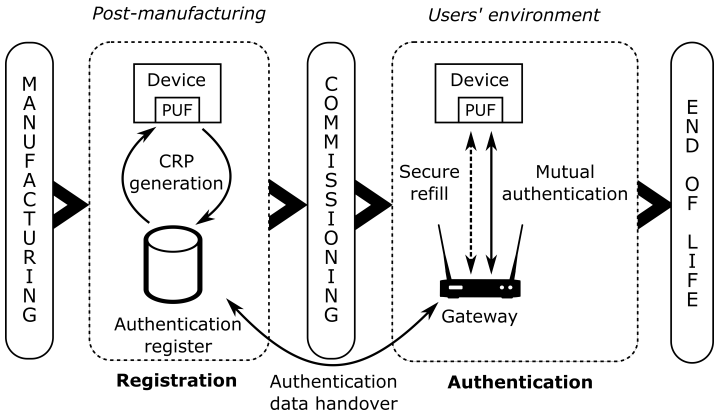


Fig. 3. Principles of the authentication protocol.

During the registration phase, the authentication register has an open access to the embedded PUF of the device to challenge it and to generate CRP in a secure environment. A strong PUF is required to have a wide challenge-response space. Once generated, the CRP are stored by the register in a secure manner and can be securely shared with the appropriate gateway. At the end of the registration phase, direct access to the PUF must be restrained. The authentication register can be either managed by the device manufacturer or delegated to a trusted third-party.

The authentication phase takes place after the device has been commissioned and integrated in the users' environment. The device reports to a gateway which checks its authenticity. To do that, the gateway queries the authentication register for the CRP of the device which are then used by the authentication protocol.

The secure refill phase takes place when the device authentication data are almost exhausted, after repeated use of the authentication phase. The gateway initiates an authentication and key derivation protocol which, when successfully completed, establishes a secure link between the device and the gateway. On the device side, the PUF can only be challenged through the secure link. At the end of the phase, the secure link is closed and the PUF access is restrained again.

### *3.2. Authentication protocol based on PUF*

To accurately describe the protocol, the following conventions are used:

- $x||y$ represents the concatenation of values $x$ and $y$.
- $x \oplus y$ represents the bit-wise exclusive OR (XOR) of values $x$ and $y$.
- $P(x)$ represents the response of a device PUF to challenge $x$.
- $H(x)$ represents the output of a secure hash function executed on $x$.

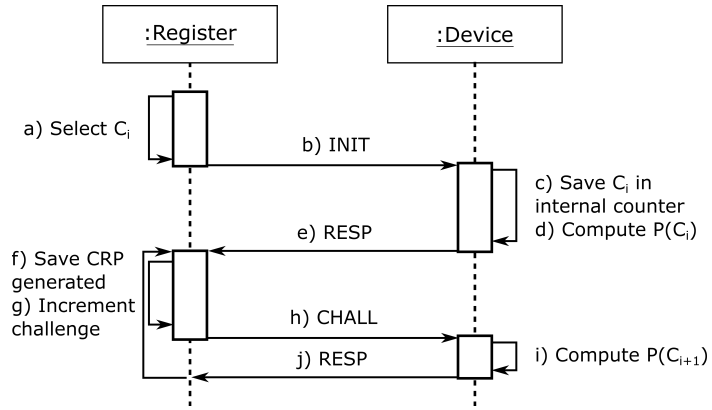


Fig. 4. Registration phase.

- $C(x; K = y)$ represents the output of a secure symmetrical cipher executed on the plaintext $x$ using $y$ as a key.
- $RM_T$, $GM_T$ and $DM_T$ represent messages sent respectively by the register (R), the gateway (G) and the device (D) with T indicating the message type.

The presentation of the process starts with the the two phases used to authenticate a device with a PUF, which are detailed in the section below, and then the secure refill phase is explained.

### *3.2.1. Registration phase*

The registration phase is the base of the trust in the device authenticity. Only the smart device and the authentication register are involved. This phase must be done in a secure environment as soon as possible after the device is manufactured. During this phase the authentication register creates and securely stores the device authentication table, a structure containing the generated authentication data. The authentication register starts by assigning a unique identifier $ID_{device}$ to the device. This identifier can be either generated by the PUF of the device, as the response to a fixed challenge, or generated by the authentication register and shared with the device which needs to store it. The authentication register creates a new authentication table associated with the identifier. Then, it begins the generation of CRP, as illustrated by Fig. 4.

The authentication register chooses a positive integer $C_i$ (a), the initial challenge. As the integer will be used by the device, this choice must take into account the way integers are handled by the device. If $I_{MAX}$ represents the greatest positive integer the device can handle, it is advised to choose $C_i$ in the interval $\{0; I_{MAX} \times 0.1\}$. This integer $C_i$ is then sent to the device using a $INIT$ message (b), which is defined by Eq. (1).

$$RM_{INIT} = INIT||C_i \tag{1}$$

The device, when receiving the message, uses the integer $C_i$ to initialize a counter (c). This counter is used to prevent rollback attacks and must be stored in non-volatile memory. The device also uses the integer $C_i$ as a challenge for its PUF system (d). The PUF response is sent back to the authentication register with a $RESP$ message (e), which is defined by Eq. (2).

$$DM_{RESP} = RESP||P(C_i) \tag{2}$$

Finally, the authentication register stores the generated CRP $(C_i; P(C_i))$ in the device authentication table (f). To generate the next CRP, the authentication register increments the integer value to obtain $C_{i+1}$ (g) and sends this value using a $CHALL$ message (h), whose definition is similar to the $INIT$ message. When receiving this message, the device uses the integer as a challenge for its PUF and sends back the response using a $RESP$ message. The counter of the device is left unchanged. The authentication register can generate the required amount of CRP by increasing the challenge integer and sending $CHALL$ messages.

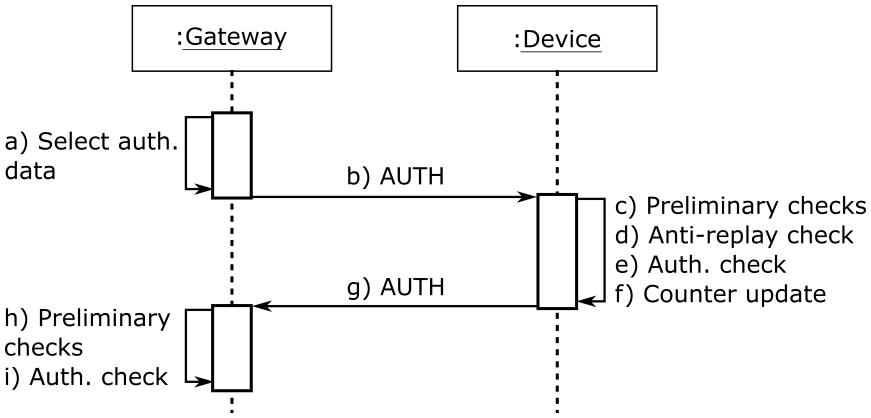


Fig. 5. Mutual authentication between a gateway and a device.

To close the registration phase, the register sends a *END* message to the device. This restrains the processing of the *INIT* and *CHALL* messages to prevent unauthorized generation of CRP which would disclose the device authentication data.

### 3.2.2. Authentication phase

Once the registration phase has been successfully carried out, the device can be sold and integrated in the users' environment to be used by pervasive applications. This is the commissioning transition. Once deployed, the device reports to a gateway. The authentication phase takes place between the device and the gateway, with the support of the authentication register. The objective is to reach mutual authentication between the gateway and the device. It is assumed that the gateway knows the authentication register and is able to authenticate and securely communicate with it.

The gateway starts by sending to the device a *ID_REQ* message. The device replies with a *ID_ANS* message, which is defined by Eq. (3).

$$GM_{ID\_ANS} = ID\_ANS||ID_{device} \tag{3}$$

With the device *ID*, the gateway can query the authentication register and recover the appropriate authentication table if the device has been properly registered. This is the handover transition illustrated in Fig. 3. Then, the gateway follows the protocol illustrated by Fig. 5.

The gateway selects in the authentication table a challenge $C_n$ and two responses $P(C_n)$ and $P(C_{n+1})$ (a). With these elements, the gateway computes its authentication message $M$, defined by Eq. (4). The two responses are combined with a XOR operation to avoid their transmission in plaintext and prevent PUF modeling attacks. This point is further discussed in Section 5.1. $M$ is then sent to the device using a $AUTH$ message (b), defined by Eq. (5). The gateway saves the device *ID* to keep track of the devices currently in an authentication procedure.

$$M = ID_{dev}||C_n||P(C_n) \oplus P(C_{n+1}) \tag{4}$$

$$GM_{AUTH} = AUTH||M||H(M) \tag{5}$$

Because the challenges are based on a counter, the device can compute $P(C_n)$ and $P(C_{n+1})$ from $C_n$. When receiving the message, the device performs preliminary checks (c) which consist of an integrity check using the message digest and a target *ID* check, to ensure the message is intended for the device. If those checks succeed, an anti-replay check (d) follows, which compares the provided challenge $C_n$ to the value stored by the counter of the

device. The check succeeds if $C_n$ is equal to or greater than the stored value. If one of those checks fails, the message is dropped. If the message is valid, the device computes the proof of authenticity of the gateway $P(C_n) \oplus P(C_{n+1})$ using its PUF and compares it to the proof included in the message (e). If the two proofs match, the gateway can be trusted and the device stores challenge $C_{n+4}$ in its counter (f), to avoid replay attacks. The device then computes its own authentication message $N$, defined by Eq. (6), and sends it to the gateway with a $AUTH$ message(g), defined by Eq. (7).

$$N = ID_{dev}||P(C_{n+2}) \oplus P(C_{n+3}) \tag{6}$$

$$DM_{AUTH} = AUTH||N||H(N) \tag{7}$$

When receiving the message, the gateway checks its integrity using $H(N)$, then it checks if $ID_{dev}$ corresponds to a device currently in authentication procedure (h). If one of those checks fails, the message is dropped. Otherwise, the gateway computes the proof $P(C_{n+2}) \oplus P(C_{n+3})$ using the responses stored in the authentication table and compares the result with the device proof of authenticity (i). If the proofs match, the device is authenticated by the gateway. To finish the authentication procedure, the gateway must delete the challenges $C_n$ to $C_{n+3}$ and their associated responses of the device authentication table, as the anti-replay counter of the device prevents their reuse.

At the end of the authentication phase, if all the steps succeeded, mutual authentication is reached between the gateway and the device and four CRP are removed from the device authentication table.

### *3.2.3. Secure refill phase*

To perform the authentication, CRP are used and then permanently deleted which leads to the exhaustion of the available authentication data for a given device. To prevent the total exhaustion of a device authentication data, as it would mean the impossibility to further authenticate the device, the protocol includes a third phase allowing the secure refill of authentication data without the need to take the device off the users' environment. A minimum of one CRP is needed to carry out the secure refill phase.

First, the gateway selects in its table a challenge $C_n$ and the corresponding response $P(C_n)$. With these elements the gateway computes an authentication message $M'$, defined by Eq. (8). The ciphered $P(C_n)$ plays the role of a random value (also called nonce). However, $P(C_n)$ does not have to be explicitly transmitted as the device can compute it with its PUF, which reduces the communication cost. The proof $M'$ is sent to the device using a $REFILL\_AUTH$ message, defined by Eq. (9). The gateway saves the device $ID$ in an internal structure to keep track of devices in a secure refill phase.

$$M' = ID_{dev}||C_n||C(P(C_n); K = P(C_n)) \tag{8}$$

$$GM_{REFILL\_AUTH} = REFILL\_AUTH||M'||H(M') \tag{9}$$

When receiving the message, the device performs the preliminary checks like a classical authentication procedure. Only the control of the proof of authenticity differs, as the device needs to cipher $P(C_n)$ using the symmetric cipher used by the gateway. If the proof of the gateway is valid, the device stores $P(C_n)$ in its volatile memory and computes its own authentication message $N'$, defined by Eq. (10). Then, the device sends back a $REFILL\_AUTH$ message defined by Eq. (11).

$$N' = ID_{dev}||C(C_n; K = P(C_n)) \tag{10}$$

$$DM_{REFILL\_AUTH} = REFILL\_AUTH||N'||H(N') \tag{11}$$

When receiving the message, the gateway performs the necessary checks to ensure that the message is valid and the device authenticity proof is correct. If every check completes with success, the gateway and the device are now mutually authenticated and they share the secret $P(C_n)$. This secret is used as a key for a symmetric cipher used to protect the next part of the refill procedure.

Once mutual authentication is reached the gateway can use the *INIT* and *CHALL* messages of the registration phase, enciphered with the shared secret. The device deciphers each message, computes the response associated to the given challenge, creates a *RESP* message with the response then enciphers it and sends it back. There is only one additional constraint compared to the initial registration phase: the gateway must choose an initial challenge greater than the value of the device counter, to maintain the anti-replay protection. Once a sufficient number of new CRP have been generated, the gateway terminates the refill phase with a final *END* message to the device. This triggers the removal of the shared secret on both sides.

To summarize, the protocol is made up of the following three phases:

- **registration**: this initial setup phase register devices and must be done in a secure environment.
- **authentication**: this phase provide lightweight mutual authentication while the device is deployed in the users' environment. Basic operations like XOR and additions are used for optimized runs on constrained devices.
- **secure refill**: this phase performs a new registration while the device is deployed in the users' environment. A more robust and standardized security principle is used with symmetric cryptography, which increases the footprint compared to the authentication phase but it remains compatible with constrained devices.

## 4. Prototyping

To validate the protocol, a demonstrator was developed with a single constrained device, a gateway and an authentication register. With those components, it is possible to register the device, to authenticate it and to securely generate new authentication data.

The demonstrator is composed of two elements: a LPC55S69-EVK development board from NXP playing the role of the device and a personal computer playing alternatively the role of the authentication register and the role of the gateway. Communication between those elements is assured by a serial link with the following properties: 115 200 bauds, 8 data bits and no parity bit. The demonstrator architecture is illustrated in Fig. 6.

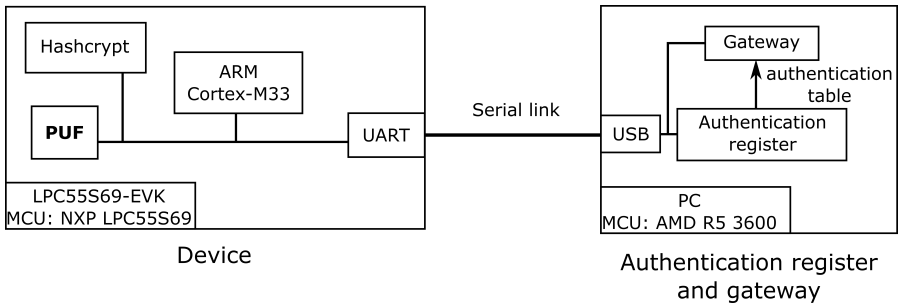


Fig. 6. Demonstrator architecture.

The device is implemented with a LPC55S69 microcontroller. This microcontroller is based on the ARM Cortex-M33 processor which is designed for IoT and embedded applications. Moreover, it is featuring a PUF subsystem based on SRAM PUF technology with built-in error correction mechanisms. The SRAM PUF being a weak PUF, the work presented in [21] is used to design the equivalent of a strong PUF by combining a weak PUF and a symmetric cipher. For the cipher, the Advanced Encryption Standard (AES) in Electronic Code Book (ECB) mode with an input/output block size of 128 bits and a key size of 128 bits is used. The LPC55S69 features a cryptographic accelerator, called Hashcrypt, supporting this algorithm with these parameters. This choice is not mandatory and any

symmetric cipher can be used, with respect to its security and performance attributes. The result of this combination is an emulated strong PUF accepting 128 bits challenges and computing 128 bits responses. The architecture of the emulated strong PUF is presented in Fig. 7.

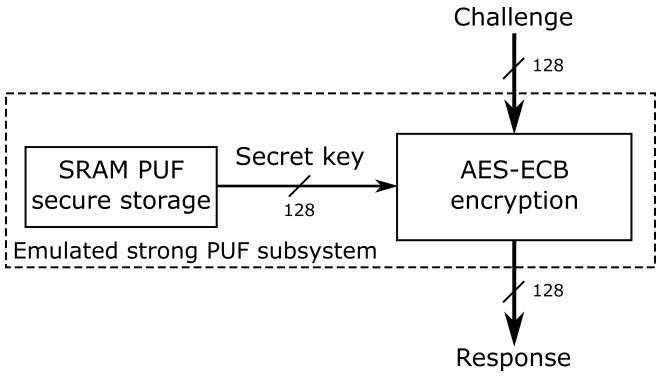


Fig. 7. Emulated strong PUF using a weak PUF and symmetric encryption.

In this architecture, the SRAM PUF acts as a secure storage for the key used by the AES-ECB cipher. The challenges are sent to the AES subsystem which enciphers them to produce the responses. Using the emulated strong PUF, the protocol operations are implemented following the specifications described in Section 3.2. The device software is written in C, using the interfaces provided by the NXP Software Development Kit.

The authentication register enrolls the device, then the gateway checks the authenticity of the device. The PC software is written in Python 3, using the pySerial module to enable serial communication with the device. The authentication table built by the authentication register is shared with the gateway. This authentication table is a very sensitive structure. The global security (storage, remote access and administration) of this register is outside the scope of this article but it is essential to take it into account when planning an on-field deployment of the protocol.

## 5. Protocol evaluation

### *5.1. Security*

Before presenting the security features of the protocol, the perimeter and attack model are defined. The security evaluation is focused on the device field use which covers the *authentication* and *secure refill* phases as illustrated by Fig. 3. It is supposed that the *registration* phase is performed in a secure environment. During field use it is supposed that the device and the gateway communicate over an insecure channel. Finally, it is supposed that the authentication register can be trusted, it holds an initial set of the device authentication data and it can securely communicate with the gateway. Concerning the attacker, his/her main goals are to successfully perform an authentication with a rogue equipment which can impersonate either a device or a gateway and to disturb the smooth operation of the protocol. To do so, it is supposed that the attacker follows the Dolev-Yao model [22] which means he/she can eavesdrop on the channel, intercept messages exchanged on the channel, replay them and create new messages.

Considering those hypothesis, the protocol has the following security features:

- **Authentication of the device**. A device is authenticated by the gateway only if it is able to present the appropriate authenticity proof composed of responses from the device PUF, in both the authentication phase and the secure refill phase. Without the device PUF, it is not possible to forge a valid proof.
- **Authentication of the gateway**. A gateway is authenticated by the device only if it can present an authentication proof composed of the device PUF responses. Except for the device which possesses the PUF, those responses are only known by the authentication register which performed the registration of the device and by the equipment trusted by the authentication register to have access to those data. If a gateway is in possession of

the device PUF responses, it means the gateway is trusted by the authentication register in charge of the device and can be trusted by the device itself. This is the case for both the authentication and secure refill phases.

- **Resistance against replay attacks**. Replay attacks are countered on the device side thanks to the counter which is updated for each valid authentication message received. An old valid message will fail the anti-replay check and thus will be dropped by the device. The same reasoning is valid for the secure refill phase, as the secure counter is updated when a valid authentication is received. On the gateway side, there is no individual counter associated to each device but as the device authentication table is progressively cleared of used CRP, a replayed message will not pass the check.
- **Protection against denial of service (DoS) attacks**. Denial of Service attacks against a device aim at perturbing its operation by flooding it with protocol messages. As the protocol requires a gateway to first authenticate when sending a message to a device, spam messages cannot achieve this. Therefore they cannot trigger further processing and the emission of a response by the device, which mitigates the denial of service. DoS attacks against a gateway are more delicate, as the gateway keeps track of the devices currently authenticating and only accept messages from those devices.
- **Protection against machine learning attacks**. Machine learning attacks aim at generating a numerical model of a given PUF instance using a subset of its CRP, a model which can correctly predict the PUF responses to arbitrary challenges with high probability [23]. To mitigates this threat, direct access to the PUF interface is deactivated once the registration phase is completed and reactivated only during the secure refill phase, after a successful authentication. Moreover, PUF responses are never exchanged in the clear to prevent passive collection of CRP by an eavesdropper. Those design choices increase the difficulty for an attacker to collect the adequate amount of CRP needed to successfully model the PUF.
- **Confidentiality of the refill phase**. The confidentiality relies on the use of a secure cipher and a temporary shared secret, similarly to the method used by the TLS Record protocol [3]. The shared secret is never exchanged in clear text and is linked to the device authenticity as it is built from a PUF response.

### *5.2. Memory footprint*

To assess if the protocol met its lightweight expectations, the protocol memory footprint was gathered and compared to another PUF-based authentication protocol [24], also implemented on a microcontroller. Memory footprints of a Transport Layer Security (TLS) library for embedded systems [25] were included as well to act as a reference of commercial authentication solutions. Other PUF-based protocols [26] could not be included as they were not implemented on a microcontroller target. The memory footprints comparison is presented in Table 1.

Table 1
Comparison of authentication protocols memory footprints.

| Protocol | Memory footprint (kB) |
|---|---|
| This article's protocol | 13 |
| Aysu's protocol [24] | 8.1 |
| Mbed TLS 1.3 with symmetric authentication [25] | 23 |
| Mbed TLS 1.3 with asymmetric authentication [25] | 53 |

Considering an hypothetical constrained device with 50 kB of memory dedicated to code, which can be considered a class 0 device, deploying a PUF protocol would leave more than 70% of memory space for application code. The TLS library with symmetric authentication could be deployed on such a device, leaving a bit more than 50% to application code, but the TLS library with asymmetric authentication could not be deployed at all. Those results illustrate the lightweight nature of PUF-based authentication protocols relative to the classical solutions.

### *5.3. Communication costs*

The communication costs of the protocol were evaluated and compared to similar PUF-based protocols [24, 26], as well as the TLS library still acting as a reference. The communication cost is defined as the length of

Table 2
Comparison of authentication protocols communication costs during the authentication phase.

| Protocol | | Communication cost (B) |
|---|---|---|
| This article's protocol | authenticator | 53 |
| | device | 49 |
| | **total** | 102 |
| Aysu's protocol [24] | authenticator | 32 |
| | device | 271 |
| | **total** | 303 |
| Barbareschi's protocol [26] | authenticator | 16 |
| | device | 16 |
| | **total** | 32 |
| Mbed TLS 1.3 with symmetric authentication [25] | **total** | 381 |
| Mbed TLS 1.3 with asymmetric authentication [25] | **total** | 1371 |

emitted messages. Depending on the communication media, emitting messages can be more energy-consuming than receiving them [27]. As a consequence communication costs should be as low as possible, particularly on the device side. The number of emitted bytes by the device and by the authenticator is given when available. Here, the authenticator refers to the equipment authentication the device. The comparison of communication costs is presented in Table 2.

Those results illustrate that PUF-based protocols have a lower communication cost than the TLS protocol. However this is not an absolute proof of the superiority of PUF protocols, as most of these protocols are still under development. Nevertheless, this provides promising outlooks for constrained devices and lightweight authentication.

### *5.4. Time metrics and registration*

Finally, time performances of the protocol were evaluated. The execution time was measured for each exchange of the protocol, an exchange consisting of a request message and its associated response message: for instance a *CHALL* message followed by a *RESP* message is considered an exchange. To measure these times, the Python gateway software was instrumented to track the relevant intervals and to export the results for processing. The average execution times are presented in Table 3.

Table 3
Authentication protocol's time metrics

| Phase | Exchange | Average time (s) |
|---|---|---|
| Registration | *INIT/RESP* | 1.1 |
| | *CHALL/RESP* | $3.0 \times 10^{-3}$ |
| Authentication | *ID_REQ/ID_ANS* | $5.5 \times 10^{-1}$ |
| | *AUTH/AUTH* | 1.1 |
| Secure refill | *REFILL_AUTH/REFILL_AUTH* | 1.1 |
| | *CHALL/RESP* | $6.4 \times 10^{-3}$ |

During the evaluation, it was noticed that the first exchange of the registration phase took a significant amount of time compared to subsequent exchanges. This is explained by the initialization of the registration phase with the initial *INIT/RESP* exchange. During this exchange the SRAM PUF is enabled, then the secret key is restored and transmitted to the AES block (see Fig. 7). Finally, the challenge is enciphered by the AES cipher and the response is sent back to the register. Once this initialization work is done, the following *CHALL/RESP* exchanges only do the challenge encryption part, explaining the important time difference.

Regarding the measures of the authentication phase, the $AUTH/AUTH$ exchange is similar in terms of PUF initialization to the $INIT/RESP$ exchange which explains the similar execution time. The $ID\_REQ/ID\_ANS$ exchange only requires the reconstruction of the ID using the SRAM PUF.

Regarding the secure refill phase, the mutual authentication takes the same amount of time to complete than the classical authentication phase. The generation of new CRP however is slower than the registration phase, which is explained by the use of encryption to ensure the confidentiality of the exchanges.

Time metrics were helpful to estimate the time needed to fully register a device. In that instance fully register means generating enough authentication data to cover the whole device lifespan. With an hypothetical frequency of one mutual authentication every minute between the device and the gateway, three device lifespans were evaluated. First, the required amounts of CRP were calculated and then, using the *registration* phase time metrics, the registration times were evaluated. Those results are presented in Table 4.

Table 4
Total registration time depending on the device lifespan.

| Device lifespan (y) | Amount of CRP (M) | Registration time (h) |
|---|---|---|
| 1 | 2.1 | 1.8 |
| 5 | 11 | 8.8 |
| 10 | 21 | 18 |

Those results highlight the challenge represented by the registration phase which needs to generate in one time the necessary authentication data for the whole device lifespan. Even for a one year lifespan, nearly 2 hours of registration time are needed. This takes a long time, not compatible with smart device manufacturing process. This implies as well an authentication register size of billions of records (millions of CRP for millions of devices), which needs to be managed by a manufacturer or a trusted third-party. Even at the scale of a gateway with tens to hundreds pervasive devices, this authentication data size could be an issue. It does not appear to be a realistic solution for massive production.

With this proposal of a refill phase, it is not necessary to generate all the CRP during the initial registration phase. Only a subset has to be generated, which is then handed over to the appropriate gateway to perform the authentications and secure refills over time when it is needed. As a consequence, the initial registration phase can be shortened to meet the industrial world constraints. To highlight this benefit, shorter device lifespans were considered with the possibility to renew those lifespans using the secure refill phase. Keeping an authentication frequency of one per minute, time evaluations of the initial registration and secure refill were computed with those shorter lifespans. The results are presented in Table 5.

Table 5
Initial registration time and secure refill time depending on the device lifespan.

| Device lifespan (d) | Amount of CRP | Registration time (s) | Refill time (s) |
|---|---|---|---|
| 1 | 1440 | 5.4 | 10.3 |
| 7 | $10 \times 10^3$ | 31 | 65 |
| 30 | $43 \times 10^3$ | 131 | 277 |

With this new approach, the device manufacturing extra-time for registration is below the minute for a lifespan of up to a week. This looks acceptable for an industrial process. In addition, the device will be unavailable recurrently for a short period of time during its field operation, in order to complete the secure refill phase. Around 10 seconds per day or less than 5 minutes per month, which seems reasonable for non-critical pervasive applications deployed in smart homes or smart cities environments. Those unavailability periods can even be synchronized with inactivity periods of the corresponding environment, like sleeping hours in the case of smart homes, to limit their impact.

Another advantage of the refill phase is that it allows the user system to run in an autonomous way once the initial handover is done. Indeed, after the gateway has recovered the first set of authentication data from the authentication register, no further contact is needed between the register and the gateway. This can be really interesting for airtight networks which are not connected to the Internet for instance. In such networks, the initial handover could be done with a physical access to the gateway.

## 6. Related Work

Research is active in the field of lightweight mutual authentication for IoT with various protocols being designed and developed.

Barbareschi *et al.* present in [26] a mutual authentication protocol using a graph-based representation of PUF systems. In their model, they consider the device and an additional equipment which is in charge of both the device registration in a secure environment and the device authentication once deployed. They do not explicitly mention when the registration and authentication should happen in the device life-cycle. Like this article protocol, their protocol requires the removal of used PUF responses to prevent replay attacks. However, the exhaustion of authentication data is only partially mentioned in their article. Their protocol does not feature a refill phase, they use only the registration phase to generate the devices authentication data with potential storage constraints depending on the number of managed devices. During the authentication phase, responses are exchanged in clear-text which can allow modeling attacks against the PUF of the device. They implemented their protocol using a Field-Programmable Gate Array (FPGA) as the device, a type of integrated circuit which can be configured using specific languages. As a result, they present a hardware-based implementation which is faster but more difficult to update than this article software-based implementation of the device side. Moreover, they used a 10 Gb data link to connect the FPGA with the computer in charge of the authentication, which seems unrealistic in a constrained setup.

Lee *et al.* design in [28] a mutual authentication protocol for low-cost wireless sensors. Their protocol does not require a large initial generation of CRP, however the registration is directly carried out by the gateway. The authors specify that the registration is carried out before the network is deployed, which makes difficult the addition of new sensors during the network lifespan. The protocol may become vulnerable to a Denial of Service (DoS) attack where an attacker can send several requests to authenticate to a device which will automatically respond as the requests are not authenticated. Their article does not include an implementation of the protocol.

Aysu *et al.* present a privacy-preserving mutual authentication protocol in [24]. To do so, they do not rely on device identifiers but they use an exhaustive search in the authentication data of the devices. Their trust model considers a set of devices and a trusted server. The server is in charge of enrolling the devices in a secure environment before they are deployed, and then it is in charge of authenticating the devices over an insecure channel. No further details are given on the life-cycle of the devices. Their protocol does not require the storage of large quantities of authentication data and their removal, so they are not concerned by exhaustion issues. In their protocol the server message to initiate the authentication is not authenticated itself, which can expose the devices to a DoS attack where an attacker impersonating the server sends several authentication messages to a device. This will trigger on the device the associated computations and the emission of response messages, increasing the energy consumption of the device. They present three implementations of their protocol: two fully software-based using a 16-bit MSP430 microcontroller and one software-based with a hardware co-processor. Similarly to this article, they connected the device to a computer with a serial link.

Van Herrewege *et al.* present in [29] a lightweight protocol for RFID readers and tags. They introduce the concept of *reverse fuzzy extractor* to reduce the calculations on the device side. Their protocol is based on the exchange of helper data, which are data used to ensure the reliability of PUF responses. As PUF responses are not transmitted in clear-text, their protocol does not remove used responses. As a consequence, their protocol is not concerned by the exhaustion of authentication data. Like this article, they consider three elements in their model: a RFID tag embedding a PUF, an authentication register managed by a trusted party and a field equipment authenticating the tag. They present the device implementation on a FPGA target but they do not mention the server side. Their protocol is not integrated in the life-cycle of the tag as no detail is given about the timing of the registration or the authentication. Like previous protocols, the message to initiate a device authentication is not authenticated itself. An attacker can then send several messages to a device to have it generate and send back several helper data associated with a PUF response. This weakness is corrected in a novel version of the protocol presented by Mael in [30], but the life-cycle of the device is still not taken into account.

## 7. Conclusion and Perspectives

For constrained devices to be successfully integrated in future smart and pervasive environments, protocols which offer a balance between security and resources consumption is needed. This article presented the design of a new protocol for authentication which meets this criteria. This protocol is lightweight in resources usage and enables mutual authentication between constrained devices and a more resourceful equipment managing the devices. The protocol key points are its clear definition of the involved actors, its integration into the life-cycle of devices from manufacturing to deployment in pervasive environments and its new approach on authentication data management supported by the refill procedure. This new protocol was implemented and validated using a hardware setup.

Using the hardware setup, the performances of the protocol were evaluated and compared to similar PUF-based protocols as well as an embedded TLS library. The results show close performances to similar PUF-based protocols and smaller memory and communication footprints than the TLS library. The protocol is secure against common attacks like replay attacks or PUF modeling attacks. This is in line with the lightweight and security targets of the protocol. Moreover, the refill feature of the protocol allows the initial registration to be performed in a reduced timespan, facilitating its integration into industrial processes and commercial products.

Future work will focus on improving the protocol and make it available on different hardware platforms for constrained devices. In addition, further work need to be done on the end-of-life management of the device to be able, for instance, to give the device a second life in another pervasive environment. Some work is needed to properly reset the device and to ensure the authentication data previously gathered cannot be used to compute new valid authentication data. Finally, a future goal is to make the protocol available in a pervasive environment to secure communications between pervasive applications and constrained devices which are part of this environment.

## Acknowledgements

This project has received funding from the Trust Chair of the Grenoble INP Fundation.

## References

[1] Cisco, Cisco Annual Internet Report (2018–2023), 2020. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[2] C. Escoffier, S. Chollet and P. Lalanda, Lessons learned in building pervasive platforms, in: *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, 2014, pp. 7–12, ISSN: 2331-9860. doi:10.1109/CCNC.2014.6866540.

[3] E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3, 2018. https://tools.ietf.org/html/rfc8446.

[4] E. Rescorla and N. Modadugu, Datagram Transport Layer Security Version 1.2, Request for Comments, RFC 6347, Internet Engineering Task Force, 2012, Num Pages: 32. doi:10.17487/RFC6347. https://datatracker.ietf.org/doc/rfc6347.

[5] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. Polk, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2008. https://datatracker.ietf.org/doc/html/rfc5280.

[6] T. Polk, R. Housley, S. Turner, D.R.L. Brown and K. Yiu, Elliptic Curve Cryptography Subject Public Key Information, Request for Comments, RFC 5480, Internet Engineering Task Force, 2009, Num Pages: 20. doi:10.17487/RFC5480. https://datatracker.ietf.org/doc/rfc5480.

[7] S. Chollet, L. Pion, N. Barbot and C. Michel, Secure IoT for a Pervasive Platform, in: *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2018, pp. 113–118. doi:10.1109/PERCOMW.2018.8480250.

[8] R.S. Pappu, Physical one-way functions, Thesis, Massachusetts Institute of Technology, 2001, Accepted: 2009-04-29T17:51:06Z. https://dspace.mit.edu/handle/1721.1/45499.

[9] M. El-hajj, A. Fadlallah, M. Chamoun and A. Serhrouchni, A Survey of Internet of Things (IoT) Authentication Schemes, *Sensors* **19**(5) (2019), 1141, Number: 5. doi:10.3390/s19051141. https://www.mdpi.com/1424-8220/19/5/1141.

[10] C. Bormann, M. Ersue and A. Keranen, Terminology for Constrained-Node Networks, 2014. https://datatracker.ietf.org/doc/html/rfc7228.

[11] N.I.o.S.a. Technology, Entity Authentication Using Public Key Cryptography, Technical Report, Federal Information Processing Standard (FIPS) 196 (Withdrawn), U.S. Department of Commerce, 1997, ISSN: 1997-0218. doi:10.6028/NIST.FIPS.196. https://csrc.nist.gov/publications/detail/fips/196/archive/1997-02-18.

[12] P. Kocher, J. Jaffe and B. Jun, Differential Power Analysis, in: *Advances in Cryptology — CRYPTO' 99*, M. Wiener, ed., Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 1999, pp. 388–397. ISBN 978-3-540-48405-9. doi:10.1007/3-540-48405-$1_2$5.

[13] P. Maistri, R. Leveugle, L. Bossuet, A. Aubert, V. Fischer, B. Robisson, N. Moro, P. Maurine, J.-M. Dutertre and M. Lisart, Electromagnetic analysis and fault injection onto secure circuits, in: *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, 2014, pp. 1–6, ISSN: 2324-8440. doi:10.1109/VLSI-SoC.2014.7004182.

[14] Global Platform, The Trusted Execution Environment: Delivering Enhanced Security at a Lower Cost to the Mobile Market, 2015. https://globalplatform.org/wp-content/uploads/2018/04/GlobalPlatform_TEE_Whitepaper_2015.pdf.

[15] U. Rührmair and D.E. Holcomb, PUFs at a glance, in: *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014, pp. 1–6, ISSN: 1558-1101. doi:10.7873/DATE.2014.360.

[16] B. Gassend, D. Clarke, M. van Dijk and S. Devadas, Silicon physical random functions, in: *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, Association for Computing Machinery, New York, NY, USA, 2002, pp. 148–160. ISBN 978-1-58113-612-8. doi:10.1145/586110.586132.

[17] D.E. Holcomb, W.P. Burleson and K. Fu, Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers, *IEEE Transactions on Computers* **58**(9) (2009), 1198–1210, Number: 9. doi:10.1109/TC.2008.212.

[18] G.E. Suh and S. Devadas, Physical Unclonable Functions for Device Authentication and Secret Key Generation, in: *2007 44th ACM/IEEE Design Automation Conference*, Association for Computing Machinery, San Diego, California, 2007, pp. 9–14, ISSN: 0738-100X. ISBN 978-1-59593-627-1. doi:10.1145/1278480.1278484.

[19] J. Delvaux, R. Peeters, D. Gu and I. Verbauwhede, A Survey on Lightweight Entity Authentication with Strong PUFs, Technical Report, 977, 2014. https://eprint.iacr.org/2014/977.

[20] B. Omoniwa, R. Hussain, M.A. Javed, S.H. Bouk and S.A. Malik, Fog/Edge Computing-Based IoT (FECIoT): Architecture, Applications, and Research Issues, *IEEE Internet of Things Journal* **6**(3) (2019), 4118–4149, Conference Name: IEEE Internet of Things Journal. doi:10.1109/JIOT.2018.2875544.

[21] M. Bhargava and K. Mai, An efficient reliable PUF-based cryptographic key generator in 65nm CMOS, in: *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014, pp. 1–6, ISSN: 1558-1101. doi:10.7873/DATE.2014.083.

[22] D. Dolev and A.C. Yao, On the security of public key protocols, in: *22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)*, 1981, pp. 350–357, ISSN: 0272-5428. doi:10.1109/SFCS.1981.32.

[23] U. Rührmair and J. Sölter, PUF modeling attacks: An introduction and overview, in: *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014, pp. 1–6, ISSN: 1558-1101. doi:10.7873/DATE.2014.361.

[24] A. Aysu, E. Gulcan, D. Moriyama, P. Schaumont and M. Yung, End-To-End Design of a PUF-Based Privacy Preserving Authentication Protocol, in: *Cryptographic Hardware and Embedded Systems – CHES 2015*, T. Güneysu and H. Handschuh, eds, Lecture Notes in Computer Science, Vol. 9293, Springer, Berlin, Heidelberg, 2015, pp. 556–576. ISBN 978-3-662-48324-4. doi:10.1007/978-3-662-48324-4$_2$8.

[25] G. Restuccia, H. Tschofenig and E. Baccelli, Low-Power IoT Communication Security: On the Performance of DTLS and TLS 1.3, *arXiv:2011.12035 [cs]* (2020), arXiv: 2011.12035. http://arxiv.org/abs/2011.12035.

[26] M. Barbareschi, A. De Benedictis and N. Mazzocca, A PUF-based hardware mutual authentication protocol, *Journal of Parallel and Distributed Computing* **119** (2018), 107–120. doi:10.1016/j.jpdc.2018.04.007. http://www.sciencedirect.com/science/article/pii/S0743731518302582.

[27] P.-M. Mutescu, A.I. Petrariu and A. Lavric, Wireless Communications for IoT: Energy Efficiency Survey, in: *2021 12th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*, 2021, pp. 1–4, ISSN: 2159-3604. doi:10.1109/ATEE52255.2021.9425330.

[28] Y.S. Lee, H.J. Lee and E. Alasaarela, Mutual authentication in wireless body sensor networks (WBSN) based on Physical Unclonable Function (PUF), in: *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2013, pp. 1314–1318, ISSN: 2376-6506. doi:10.1109/IWCMC.2013.6583746.

[29] A. Van Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A.-R. Sadeghi, I. Verbauwhede and C. Wachsmann, Reverse Fuzzy Extractors: Enabling Lightweight Mutual Authentication for PUF-Enabled RFIDs, in: *Financial Cryptography and Data Security*, A.D. Keromytis, ed., Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, pp. 374–389. ISBN 978-3-642-32946-3. doi:10.1007/978-3-642-32946-3$_2$7.

[30] R. Maes, Physically Unclonable Functions: Constructions, Properties and Applications (Fysisch onkloonbare functies: constructies, eigenschappen en toepassingen), PhD thesis, 2012. https://lirias.kuleuven.be/retrieve/192818.