PUF-Based Protocol for Securing Constrained Devices

Arthur Desuert, Stéphanie Chollet, Laurent Pion and David Hély
Univ. Grenoble Alpes, Grenoble INP, LCIS
F-26000 Valence, France
{firstname.lastname}@lcis.grenoble-inp.fr

***Abstract*—In networks of smart devices, managing the trust among devices is a key challenge to prevent malicious intrusions of rogue agents. One aspect of trust is to ensure the authenticity of each device. Authentication schemes already exist for resourceful nodes, but we lack solutions for constrained devices which are undoubtedly an important part of those networks. In this paper, we present a new PUF-based authentication protocol for smart devices with autonomy and cost constraints, which aims at filling the gap between current authentication protocols and no authentication at all. We also describe our first implementation of the protocol using a microcontroller of the NXP LPC55S6x family, which features secure storage and key generation using a SRAM PUF. Finally, we present our evaluation of the protocol's principle and performances to validate the feasibility of the solution in real-world applications.**

***Index Terms*—Internet of Things (IoT), Physically Unclonable Function (PUF), authentication**

I. Introduction

Lights, thermometers, cameras, smart-meters and even refrigerators! The number and the diversity of devices we connect to the Internet is growing everyday, forming the vast Internet-of-Things space. Those devices collect data from their surroundings and send them to dedicated platforms in the fog or the cloud for further processing. They can also act upon their environment. Using those connected devices, pervasive applications aim at offering services to users while remaining invisible to them. Pervasive means at the same time ubiquitous, ambient, seamless and transparent [1]. Those applications extend software systems into the physical world to offer an intelligent environment.

Pervasive applications work in close proximity to the real world and potentially to human users, with expected deployment of those applications in domains such as smart-home, smart-buildings or Industry 4.0. Consequently, it is vital to integrate security considerations in the development process to ensure trust in pervasive environments, as any malfunction or misbehavior would greatly disappoint the final user [1]. This is quite a challenging task, particularly concerning hardware security with smart devices highly heterogeneous in terms of cost, computational power or exposure to external threats. A first step toward ensuring this hardware security would be to guarantee the authenticity of those devices from their manufacturing to their deployment as part of an intelligent environment. This would avoid the intrusion of counterfeit or rogue devices which could perturb the pervasive application's operations.

Authentication solutions for Internet-of-Things (IoT) already exist based on the Public Key Infrastructure (PKI) with the use of asymmetric cryptography, device's certificates, micro vaults and secure protocols. They answer to some of our problems, but they also have limitations: they are glutton in terms of computational load and memory space which can negatively affect device's autonomy, they are not always easy to deploy and configure, and they are generally designed for high security needs. Those solutions are ideal for fields like e-Health or smart-grids. But for less critical applications which target cost-efficiency, an intermediate solution should be available which offers an acceptable level of security.

Physically Unclonable Functions (PUF) are quite recent hardware blocks first introduced in 2001 by Pappu [2]. A PUF can generate values based on physical imperfections made during the manufacturing of the PUF structure. Even with the same base model each created PUF carries its own imperfections, forming the PUF's fingerprint which influences the output of the block. Because this fingerprint is based on imperfections, it is almost impossible to reproduce the precise configuration of a given PUF instance which justifies the unclonable title of the hardware function. Thank to this property, PUF can be used as a secure generator for an unique identifier with lower integration costs than traditional solutions. But this is only one of the possible applications of this technology which can also be exploited in authentication protocols. This is precisely our focus in the rest of the paper.

We present a new authentication protocol using PUF technology for pervasive devices which takes into account potential constraints like power autonomy or computational speed and aims for cost-efficiency. We precisely describe the integration of our protocol in the device's life-cycle to ensure the device is genuine and can be trusted. We also discuss the link between the expected device's lifespan and the required amount of authentication data.

This paper is organized as it follows: Section II introduces some of the existing techniques to authenticate smart devices and underline the opportunity for lightweight authentication. Section III presents our global approach for lightweight authentication protocol using PUF technology. Section IV details the two phases of the protocol. Before the conclusion, implementation and validation are explained in Section V.

II. Background

A. Existing Authentication Techniques

Authentication of smart devices is a hot topic and some solutions are already commercially available to ensure the authenticity of devices before and during their operating time inside an IoT network.

One widely used authentication method involves asymmetrical cryptography certificates, very similar to the method used to authenticate servers in Web navigation. For each device to authenticate, an asymmetrical key pair is generated. The private key part is securely embedded in the device while a certificate is generated with the public key part. This certificate can be distributed to any equipment needing to authenticate the connected device. The authentication procedure is illustrated by the Fig. 1.

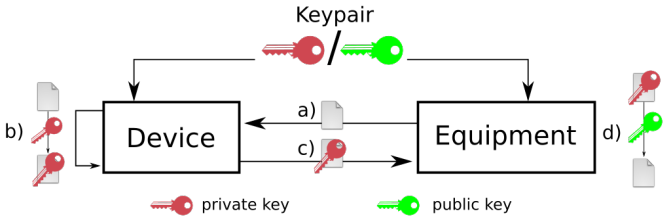

Fig. 1. Authentication with asymmetric keys.

The equipment generates a challenge and sends it to the device (a). The device ciphers the challenge using its private key (b) and sends it back (c). Finally the equipment verifies the challenge using the device's certificate (d). This method is efficient because the private key can be generated and stored inside the device, but it relies on asymmetric cryptography which performances might not compatible with very constrained devices in terms of power, timing or computation capacity. This is particularly a problem for devices with autonomy constraints as recurrent authentication could greatly shorten the device's lifespan.

To tackle this issue, it is possible to use symmetrical cryptography for authentication purposes. The principle is similar to the certificate method: for each device which needs authentication, a secret key is generated and embedded in the device in a secure manner. This key needs to be shared with equipment willing to authenticate the smart device, a gateway for instance. The equipment can then send a challenge to the device which ciphers it with its secret key and sends it back for authentication. Here, the tricky part is sharing the secret key between multiple devices: secure procedures are required to protect the key's confidentiality during its transfer to the devices. Key revocation and renewal is also a more complex operation because of the use of one single key for multiple devices: for instance, if one device is compromised it affects all the equipment sharing the same secret key until they are updated with a new key.

In both authentication scenarios, 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. 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 includes sensors to detect intrusion attempts and react accordingly, by erasing the internal storage for instance. It is also designed to be tamper proof against hardware attacks such as power side channel or fault injection attacks. Those security qualities are often qualified by a certification laboratory and must be compliant with security standards, evaluated with the Common Criteria (CC) for instance. One drawback of SE is the extra cost it induces to the system in terms of components and engineering [3].

Another solution for secure storage is to use a Micro-Controller 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 doesn't include anti-tampering protections [3].

Finally, the development of Physically Unclonable Functions (PUF) technology [4] has led to its use as a lightweight security feature and authentication protocols have been developed around it. We detail this in the next section.

B. PUF Technology for Authentication

A Physically Unclonable Function is a hardware block which reacts to an input stimulus called the challenge by emitting an output stimulus called the response. PUF's originality is that its output not only depends on the challenge but also on some intrinsic properties of the hardware components, such as the geometry of its transistors or the length of its circuits. 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 also 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. Another metric of interest is the error rate, to quantify the PUF responses' stability. 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 also critical 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 to create a PUF, depending on the integration constraints and required properties. The first PUF design, presented by R. Pappu in 2001 [2], 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 [5], 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 [5] is an example of this type of PUF. It is a strong PUF but it requires very precise circuit routing to ensure optimal operation of the PUF. Memory-based PUF rely on memory circuits, like the SRAM PUF [6]. The main advantage is the use of already existing hardware to implement those PUF, but most memory-based PUF are weak PUF which are less practical than strong PUF.

PUF circuits have some intrinsic security properties: first, their unclonability which guarantee 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 prevent 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. This secure storage function can be exploited in the design of low-cost authentication protocol offering a decent security level. A simple example of PUF-based authentication procedure is presented by G. E. Suh and S. Devadas [7], using Challenge Response Pairs (CRP), illustrated by Fig. 2.

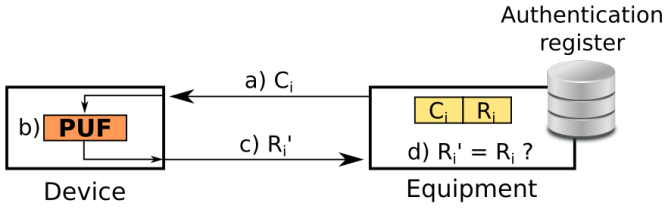

Fig. 2. Authentication procedure with previously initialized register.

The authentication register contains couples of CRP that have been stored during a previous operation called the enrollment. To authenticate the device, the equipment selects a challenge in the authentication register and sends it to the device (a). The device presents the challenge it received to its internal PUF (b) and sends the corresponding response back to the equipment (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).

To conclude, authenticating devices using PUF technology is possible and some research has been done in this direction. However, some blind spots remain which can prevent the large scale deployment of those solutions in commercial and industrial products. First, there is the management of the authentication register, a critical equipment which registers the constrained device, then ensures the secure storage of the gathered authentication data and its availability when it is needed to authenticate the device. Then, there is the problematic of authenticity management during the device lifespan and the amount of CRP needed to do this. Indeed, in Suh and Devadas' protocol the challenge and response are exchanged in the clear. To prevent malicious CRP reuse, the pair is immediately suppressed after the authentication which leads to the exhaustion of the device's authentication data.

III. Global Approach

In this section, we describe our global approach to address the subject of PUF-based authentication for constrained devices. We propose an authentication protocol supporting register management according to the device life-cycle. The device is a smart object which can be integrated in the end user's environment as part of a pervasive application. As this device can collect data about the user's environment or even modify this environment, it is crucial to authenticate it as a genuine. This device contains a PUF used to authenticate it in the system as a low-cost solution in opposition to the classical authentication solutions based on certificates.

Our protocol is based on three elements as illustrated in Fig. 3: the device including a PUF, the authentication register and the end user's gateway. We propose two phases in the device life-cyle for ensuring the authentication:

- *registration*: it is an initialization step to gather authentication data using the embed PUF of the device. After this step, the device is commissioned and ready to be used in a pervasive environment.
- *authentication*: the device exchanges an extract of authentication data previously gathered in order to be authenticated with the end user's gateway.

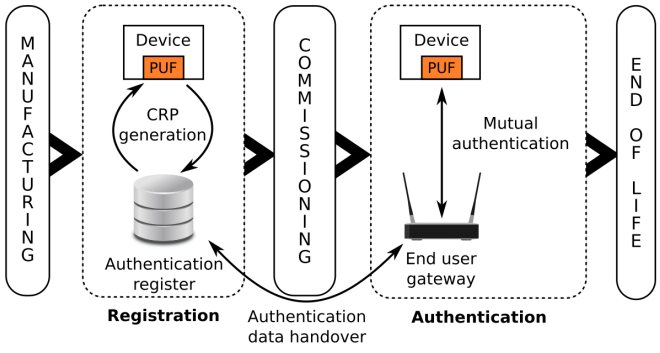

Fig. 3. Principles of the authentication protocol.

During the registration phase, the authentication register has an open access to the device's embedded PUF to challenge it

and to generate some CRP in a secure environment. A strong PUF or a weak PUF with derivation process 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 shared to the appropriate gateway. The authentication register can be managed directly by the device's manufacturer or delegated to a trusted third-party.

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

With this global view, we have precisely defined the participants of our authentication protocol, their interactions and who manages each of them. We have also showed where our protocol takes place in the device's life-cycle to ensure the device's genuineness. In the next section, we detail further each phase of the protocol.

IV. Authentication Protocol Based on PUF

To accurately describe our protocol, we define the following conventions:

- $x||y$ represents the concatenation of values x and y;
- $P(x)$ represents the response of a device's PUF to challenge x;
- $H(x)$ represents the output of a secure hash function executed on x;
- RM_T, GM_T and DM_T represent messages sent respectively by the register, the gateway and the device with T indicating the message's type.

In the next sections, we detail the two phases of our protocol used to authenticate device with a PUF.

A. Registration Phase

The registration phase is the base of the trust in the device's authenticity. Only the smart device and the authentication register are involved. This phase must be done in a secure environment and as soon as possible after the device production. During this phase the authentication register creates and securely stores the device's authentication table, a structure containing the generated authentication data. The authentication register starts by computing a unique identifier ID_{device} and sends it to the device using a $SAVE$ message, which is defined as it follows:

$$RM_{SAVE} = SAVE||ID_{device} \quad (1)$$

When the device receives this message, it stores the identifier in its internal memory. In parallel, the authentication register creates a new authentication table associated with the identifier. Then, the authentication register begins the generation of CRP, as illustrated by Fig. 4.

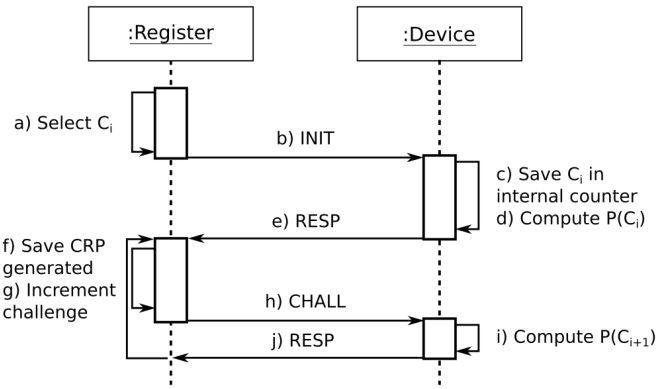

Fig. 4. Registration phase.

The authentication register chooses a positive integer C_i (a), the initial challenge, and sends it using a $INIT$ message (b), which is defined by:

$$RM_{INIT} = INIT||C_i \quad (2)$$

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 securely 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:

$$DM_{RESP} = RESP||P(C_i) \quad (3)$$

Finally, the authentication register stores the generated CRP $(C_i; P(C_i))$ in the device's 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), which 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 device's counter is left untouched. The authentication register can generate the required amount of CRP by incrementing the challenge integer and sending $CHALL$ messages.

At the end of the registration phase, the $INIT$ and $CHALL$ messages must be disabled to prevent unauthorized generation of CRP which would disclose the device's authentication data.

B. Authentication Phase

Once the registration phase has been successfully carried out, the device can be sold and integrated in a pervasive environment. This is the commissionning transition. Once in its pervasive environment, the device reports to a user's gateway in our global architecture. The authentication phase takes place between the device and the gateway, with the support of the authentication register. The authentication is mutual between the gateway and the device. We assume that the gateway knows the authentication register and is able to 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:

$$GM_{ID\_ANS} = ID\_ANS||ID_{device} \tag{4}$$

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.

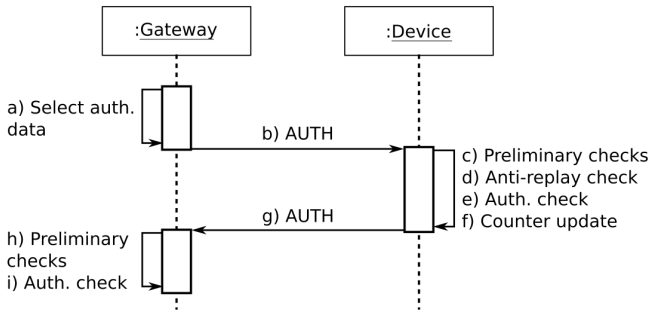

Fig. 5. Mutual authentication between a gateway and a device.

The gateway selects in the 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 (5), and sends it to the device using a $AUTH$ message (b), defined by (6). The gateway also saves the device's ID to keep track of the devices currently in an authentication procedure.

$$M = ID_{dev}||C_n||P(C_n) \oplus P(C_{n+1}) \tag{5}$$

$$GM_{AUTH} = AUTH||M||H(M) \tag{6}$$

Because the challenges are based on a counter, the device can compute $P(C_n)$ and $P(C_{n+1})$ from C_n. Moreover, we do not transmit responses in plain text to prevent machine learning attacks against the PUF [8]. When receiving the message, the device performs preliminary checks (c) which consist of an integrity check using the message's 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 device's secure counter. The check succeeds if C_n is equal or greater than the stored value. If one of those checks fails, the message is dropped. If the message is valid, the device can compute the gateway's authenticity proof $P(C_n) \oplus P(C_{n+1})$ using its PUF and compare it to the message's proof (e). If the proofs are equal, the gateway can be trusted and the device stores challenge C_{n+4} in its secure counter (f), to avoid replay attacks. The device then computes its own authentication message N, defined by (7), and sends it to the gateway with a $AUTH$ message(g), defined by (8).

$$N = ID_{dev}||P(C_{n+2}) \oplus P(C_{n+3}) \tag{7}$$

$$DM_{AUTH} = AUTH||N||H(N) \tag{8}$$

Once again, we don't transmit the responses in plain text for the underlined reasons. When receiving the message, the gateway checks its integrity using $H(N)$, then it checks if ID_{dev} matchs to a device currently in authentication procedure (h). If one of this checks fails, the message is dropped. Otherwise, the gateway computes $P(C_{n+2}) \oplus P(C_{n+3})$ using the responses stored in the authentication table and compares the result with the device's authenticity proof (i). If it is equal, the device is authenticated by the gateway. To finish the authentication procedure, the gateway must delete the entries C_n to C_{n+3} of the authentication table, as they can't be used anymore due to the anti-replay counter of the device.

At the end of the authentication phase, if all the steps succeeded, mutual authentication is reached between the gateway and the device.

V. Implementation and Validation

To validate our protocol, we developed a demonstrator with a single constrained device, a gateway and an authentication register. With those components, we are able to enroll, then to authenticate the device. The implementation details are presented in Section V-A. Then, we used this demonstrator to gather some metrics in order to evaluate our protocol. Those results are presented in Section V-B.

A. Implementation

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.

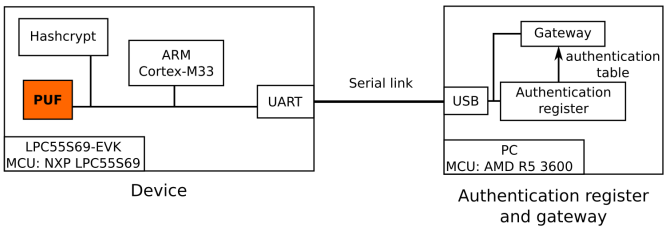

Fig. 6. Demonstrator architecture.

The device is implemented with the LPC55S69 MCU because it is based on the ARM Cortex-M33 processor which is designed for 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, we used the work presented in [9] to design

the equivalent of a strong PUF by combining a weak PUF and a symmetric encryption algorithm. For the symmetric encryption algorithm, we choose the Advanced Encryption Standard (AES) in Electronic Code Book (ECB) mode because the LPC55S69 also features a cryptographic accelerator, called Hashcrypt, supporting this algorithm. This choice is not mandatory and any symmetric algorithm can be used, with respect to its security and performance attributes. The architecture of emulated strong PUF is presented in Fig. 7.

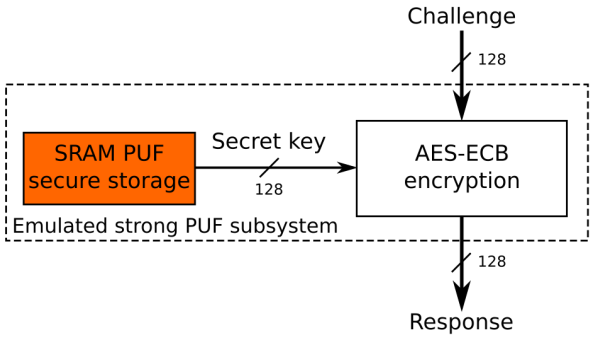

Fig. 7. Emulated strong PUF using a weak PUF and symmetric encryption.

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 ciphers them to produce the responses. Using the emulated strong PUF, the protocol operations are implementing following the specifications described in Section IV. The device's software is written in C, using the API provided by the NXP Software Development Kit.

The authentication register enrolls the device, then the gateway checks the authenticity of the device. The 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 to the gateway. This authentication table is a very sensitive structure. In our work, we don't actually concentrate our efforts on the security of this table. But for an on-field deployment of the protocol in a pervasive environment, it is essential to take it into account.

B. Validation

To evaluate our protocol, we chose to focus on time and memory usage metrics. Concerning time metrics, we measured the time of execution for each exchange of the protocol, an exchange consisting of a request message and its associated response message: for instance, during the registration phase, a $CHALL$ message followed by a $RESP$ message is considered an exchange. To measure these times, we instrumented our Python software to track the relevant intervals and export the results for processing. We measured the execution times for all the protocol exchanges. The average execution times are presented in Table I.

TABLE I
AUTHENTICATION PROTOCOL'S TIME METRICS

TIME METRICS	
Average $INIT/RESP$ exchange	1.1s
Average $CHALL/RESP$ exchange	3.0ms
Average ID_REQ/ID_ANS exchange	0.55s
Average $AUTH/AUTH$ exchange	1.1s

During our tests, it can be noticed that the first exchange of the registration phase takes a significant amount of time compared to subsequent exchanges. This $INIT/RESP$ exchange initializes the registration phase. During this step the SRAM PUF is enabled, then the secret key is restored and transmitted to the AES block. Finally, the challenge is encrypted by the AES cipher and the response is sent back to the register. Once this initialization work has been 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 explain the similar execution time. The ID_REQ/ID_ANS exchange only requires the reconstruction of the ID using the SRAM PUF. We judge those time results acceptable for non critical pervasive devices.

For memory metrics, we gathered the size of our client application loaded in the device and the size of the variables requiring a non-volatile storage. A summary of those metrics is presented in Table II.

TABLE II
AUTHENTICATION PROTOCOL'S MEMORY METRICS.

MEMORY METRICS	
Client application	24.8kB
Anti-rollback counter	4B
PUF activation code*	1192B
Device ID recovery code*	52B
Secret key recovery code*	52B

* *Variable specific to our PUF implementation.*

About the client application's size the code in charge of the protocol's logic takes around 5.7kB of space, the remaining code being the libraries provided by NXP to ease the development on the target. Among those libraries, PUF library and Hashcrypt library take respectively around 4.9kB and 200B of space.

The anti-rollback counter is crucial as it constrains the maximum number of authentications the device can carry out, constraining the device's lifespan too. In our implementation we chose a size of 32 bits for this counter, allowing around 1 billion authentications as the counter is incremented by 4 each time an authentication success. Even at a rate of one authentication every minute, it would take more than 100 years to reach the counter's limit. This is a more than sufficient limit for a device with a lifespan of 10 to 20 years. It must be noted that the anti-rollback counter must be stored in a secure part

of the non-volatile memory which prevent illegal access or tampering of the stored value.

The others variables listed are specific to our implementation. Precisely, they are specific to the PUF subsystem developed by NXP. The PUF activation code is produced at the initialization of the SRAM PUF and is needed each time the PUF is powered on to restore its internal signature. The device identifier (ID) and secret key recovery codes are generated when storing the mentioned values using the PUF secure storage. Those codes are needed to recover the stored values from the PUF once it is powered on and correctly initialized. As those codes don't leak sensitive information about the stored values, they can be stored in a non-secure part of the device's non-volatile memory.

Finally, we evaluated for different expected lifespans the amount of CRP needed to authenticate the device with a frequency of one mutual authentication every minute. Combined with the average registration request time, this allowed us to estimate the total registration time with our implementation. Those results are presented in Table III.

TABLE III
TOTAL REGISTRATION TIME DEPENDING ON THE DEVICE'S LIFESPAN.

Device's lifespan expected (year)	1	5	10
Required amount of CRP	2.1M	11M	21M
Estimated registration time (hour)	1.8	8.8	18

Those results highlight the challenge represented by the registration phase which need to generate in one time the necessary authentication data for the whole device's life. Even for a one year lifespan, nearly 2 hours of registration time are needed.

In summary, our protocol presents decent memory metrics and reasonable authentication time once on the field. The average registration time is also decent but the potential issue lies in the number of CRP needed to cover the whole device's lifespan. In the next section, we present other researches around PUF-based authentication.

VI. Related Work

Others protocols using PUF technology have been developed to achieve lightweight mutual authentication for IoT. Barbareschi *et al.* present in [10] an authentication protocol relying on *PUF chains*, a new way to generate CRP to ease the authentication on the device side. In their model they identify 3 equipment involved in the authentication: a trusted third-party in charge of the enrollment, an authentication equipment called the *verifier* and the device to authenticate. But they don't integrate their protocol in the device's life-cycle. Also, challenges and responses are exchanged in clear during their *verification* phase, which could allow modeling attacks against the device's PUF.

Van Herrewege *et al.* present in [11] a lightweight protocol for RFID readers and tags. They introduce their concept of *reverse fuzzy extractor* to reduce the calculations on the device's side. Their protocol is based on the exchange of helper data, which ara data used for the error correction of the PUF's response. Like us, they consider 3 players in their model: a database managed by the token issuer which is the equivalent of our register, a reader which plays the role of our gateway and a token which is similar to our smart device. But they don't describe how their protocol takes place in the token's life-cycle.

A more advanced protocol presented in [12] uses PUF with asymmetric cryptography to provide both authentication and confidentiality. They consider a more complex user's network than us, with not only the devices and their authentication server but also higher devices like routers. In their protocol, the enrollment is done by the authentication server in a secure way before the device join the network. This can be difficult to achieve if the server is already managing some devices. Moreover, this protocol was proven vulnerable by Braeken in [13]. The author presents the attacks on the protocol and proposes an alternative scheme which fixes the identified vulnerabilities, but doesn't develop further the enrollment procedure.

In [14], Öztürk *et al* focus on highly constrained RFID systems in pervasive environment. They don't use any encryption or hash primitive on the device, only two PUF circuits to respect their constraints. Like us, they designed their protocol to prevent modeling attacks, deliberately using noisy PUF responses and other countermeasures to increase the attack's difficulty. In their protocol the RFID reader is in charge of the initialization and enrollment of the token after its production, and also its authentication when deployed on the field.

Aysu *et al.* present a protocol which ensure the privacy of the authenticated devices in [15]. To do so, they don't rely on device identifiers like us but they use exhaustive search in the devices' authentication data. In their trust model they consider a trusted server in charge of the enrollment in a secure environment and then the authentication of the devices. No information is given about the timing of those phases in the device's life-cycle.

VII. Conclusion and Perspectives

In this paper, we presented the basis of lightweight authentication using the PUF technology. We identified some blind spots which can prevent the good development and integration of this kind of authentication in smart devices. Then, we presented a proposal including a global architecture and its associated authentication protocol to try on clarify those blind spots. We realized a demonstrator with a quite recent and dedicated to constraint systems NXP LPC55S69 microcontroller embedding a PUF block to prove the feasibility of the protocol.

Testing the demonstrator, we got reasonable results in terms of memory consumption and authentication time. This first step is encouraging us for new researches of lightweight security integration in devices with the objective of securing pervasive applications to ensure they can be trusted by users. However, our protocol still need improvements, on the authentication data handover for instance, before thinking about a large scale deployment.

AKNOWLEDGMENT

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

REFERENCES

[1] C. Escoffier, S. Chollet, and P. Lalanda, "Lessons learned in building pervasive platforms," in *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, Jan. 2014, pp. 7–12, iSSN: 2331-9860.

[2] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical One-Way Functions," *Science*, vol. 297, pp. 2026 – 2030, 2002.

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

[4] U. Rührmair and D. E. Holcomb, "PUFs at a glance," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2014, pp. 1–6, iSSN: 1558-1101.

[5] 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*, ser. CCS '02. New York, NY, USA: Association for Computing Machinery, Nov. 2002, pp. 148–160. [Online]. Available: https://doi.org/10.1145/586110.586132

[6] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers," *IEEE Trans. Computers*, vol. 58, no. 9, pp. 1198–1210, 2009. [Online]. Available: https://doi.org/10.1109/TC.2008.212

[7] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *Proceedings of the 44th Design Automation Conference, DAC 2007, San Diego, CA, USA, June 4-8, 2007*. IEEE, 2007, pp. 9–14. [Online]. Available: https://doi.org/10.1145/1278480.1278484

[8] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proceedings of the 17th ACM conference on Computer and communications security*, ser. CCS '10. New York, NY, USA: Association for Computing Machinery, Oct. 2010, pp. 237–249. [Online]. Available: https://doi.org/10.1145/1866307.1866335

[9] 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)*, Mar. 2014, pp. 1–6, iSSN: 1558-1101.

[10] M. Barbareschi, A. De Benedictis, and N. Mazzocca, "A PUF-based hardware mutual authentication protocol," *Journal of Parallel and Distributed Computing*, vol. 119, pp. 107–120, Sep. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731518302582

[11] 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*, ser. Lecture Notes in Computer Science, A. D. Keromytis, Ed. Berlin, Heidelberg: Springer, 2012, pp. 374–389.

[12] U. Chatterjee, R. S. Chakraborty, and D. Mukhopadhyay, "A PUF-Based Secure Communication Protocol for IoT," *ACM Transactions on Embedded Computing Systems*, vol. 16, no. 3, pp. 67:1–67:25, Apr. 2017. [Online]. Available: https://doi.org/10.1145/3005715

[13] A. Braeken, "PUF Based Authentication Protocol for IoT," *Symmetry*, vol. 10, no. 8, p. 352, Aug. 2018, number: 8 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2073-8994/10/8/352

[14] E. Öztürk, G. Hammouri, and B. Sunar, "Towards Robust Low Cost Authentication for Pervasive Devices," in *2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Mar. 2008, pp. 170–178.

[15] 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*, ser. Lecture Notes in Computer Science, T. Güneysu and H. Handschuh, Eds. Berlin, Heidelberg: Springer, 2015, pp. 556–576.