# An Improved PUF-Based Privacy-Preserving IoT Protocol for Cloud Storage

Cédrick De Pauw[a], Jan Tobias Mühlberg[b] and Jean-Michel Dricot[c]

*École Polytechnique de Bruxelles, Université Libre de Bruxelles, Av. Roosevelt 50, 1050 Bruxelles, Belgium*

Keywords:     SRAM PUF, Extended Lifetime, Supply Chain Security, Embedded Devices.

Abstract:     The IoT technology allows many types of personal data to be measured by many kinds of devices and sensors, and to be sent over the Internet for various applications. However, this data transmission has to be secure and the privacy of the users should ideally be preserved. In this work, we propose a SRAM PUF-based privacy-preserving IoT protocol for cloud storage based on an existing protocol from the literature. Proposals are made to increase the supply chain security of the PUF construction used by a device, to extend the secure lifetime of this device by increasing the number of keys it may generate and avoiding reboot-based attacks, and to allow a PUF construction to be used for different applications. These proposals only require changes on the device enrollment and on the master key generation procedure, leaving the PUF construction, the fuzzy extractor construction and the cryptographic key derivation unchanged. Benefits and limitations of this new protocol are evaluated and security objectives achieved with these proposals are analyzed.

## 1 INTRODUCTION

The Internet of Things (IoT) technology has significantly grown in popularity and raised new security and privacy issues. With the use of IoT devices, more and more personal data is exchanged and some manufacturers underestimate potential security threats in their product (Babaei and Schiele, 2019). Moreover, sensitive data may be exposed to third-party infrastructure during transport and processing, where this data may then be monetized without consent.

Strong cryptographic primitives and protocols may not be suited for IoT devices which should be inexpensive and often have limited resources in terms of computing power or battery autonomy (Babaei and Schiele, 2019; Ashur et al., 2018). Therefore, dedicated protocols are proposed in the literature. Common solutions for keys storage rely on power-backed volatile memory or non-volatile memory (NVM) (Mostafa et al., 2020), both solutions being vulnerable to attacks (Mostafa et al., 2020). Physically unclonable functions (PUFs) were proposed to address the key storage problem and to provide secure and lightweight authentication and session keys.

[a] https://orcid.org/0009-0002-1223-3285
[b] https://orcid.org/0000-0001-5035-0576
[c] https://orcid.org/0000-0002-8539-9940

PUFs are based on established concepts, i.e., identification or randomness extraction from the measurements of manufacturing variations, which have previously been used for document and hardware identification (Graybeal and McFate, 1989; Herder et al., 2014; Anderson, 2021; Rührmair, 2022).

Prior research proposes to use the PUF construction for one application only. For constrained devices, however, it would be beneficial to re-use a PUF construction for different applications, in comparison with storing distinct cryptographic keys in a NVM (Ferreira, 2022). Commonly, the PUF usage is also limited, e.g., to a certain number of encryptions. This approach limits the lifetime of the PUF and thereby often the life span of the device, in particular for devices that frequently send sensitive data. E.g., the protocol proposed by Ashur et al. (2018) relies on electronic fuses to prevent the reuse of memory segments that were previously used to generate keys. Here, the PUF construction is limited to one-time and one-application use with an electronic fuse being consumed on every reboot, which presents an attack vector and imposing limitations on the secure lifetime of a device. Finally, application service providers often perform the enrollment of the PUF, which poses a problem in case of a supply-chain compromise. In Ashur et al. (2018), master keys are extracted at the manufacturing site and provided to the user via

Quick Response (QR) codes. Although these QRs is sealed in a tamper-evident packaging, it could be compromised at earlier stages. Foreseeing options for the user to parameterize and extract new keys would thwart such attacks and extend the lifetime of devices.

**Contributions.** We propose improvements to Ashur et al. (2018) towards a privacy-preserving cloud storage protocol which takes the PUF construction lifetime into account, which allows the user to reset and manage their key generation parameters, and which can be used to protect data associated to distinct applications. We select Ashur et al. (2018)'s protocol for its privacy-preserving properties: unlinkability, anonymity and pseudonymity, confidentiality and plausible deniability. Our proposal inherits these properties and improves the master key generation process in terms of flexibility and sustainability.

Specifically, our proposal allows for all memory segments of the PUF to be reused in order to derive a unlimited number of keys, to roll out these keys after a reset, and hence to extend the secure lifetime of the device. This "unlimited" number of keys is possible due to the use of three distinct random numbers, a PUF response and different hash computations, making the key generation sequence hard to predict. The improved protocol shall further allow the user to reconfigure the key generation parameters for different applications. Finally, if enough memory is available on the device, it should be possible to simultaneously use the PUF construction for different applications.

The scope of this contribution is defined by the following objectives:

- Propose improvements to the PUF (and device) enrollment and the key generation procedure of Ashur et al. (2018)'s protocol;

- Perform an analysis of the security objectives achieved by this improved protocol.

## 2 BACKGROUND

This section introduces required PUFs concepts and presents Ashur et al. (2018), on which our work is based, in detail.

### 2.1 Physically Unclonable Functions

A *physically unclonable function* is a physical function built from an integrated circuit (IC) and relying on the physical variations introduced by the manufacturing process of some components (Mostafa et al., 2020; Herder et al., 2014). A PUF is used to generate challenge-response pairs (CRPs): as illustrated by

Figure 1, a *challenge* is a sequence of bits given as input to the function, while a *response* is a sequence of bits obtained as output from the function for a given challenge (Babaei and Schiele, 2019; Mostafa et al., 2020; Herder et al., 2014).
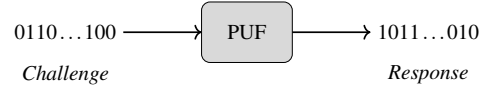


Figure 1: PUF challenge-response pair.

This section will introduce main PUF properties and the static random-access memory (SRAM) PUF architecture, on which the current work relies.

**PUF Properties.** PUFs obey two main properties: *identifiability* and *physical unclonability* (Maes, 2013; Schrijen, 2018; van der Leest and Tuyls, 2013). These properties are defined as follows (Maes, 2013).

**Identifiability.** *A PUF circuit can precisely be identified based on its generated CRPs.*

**Physical Unclonability.** *A PUF circuit can not be physically reproduced even if an adversary controls its manufacturing process.*

The identifiability of a PUF is only possible due to its *reproducibility*, also called *reliability* (Schrijen, 2018), and *uniqueness* (Maes, 2013; Schaller et al., 2014; Maiti et al., 2012). These properties are briefly defined and illustrated below (Maes, 2013).

**Reproducibility.** *A PUF circuit reproduces the same response for a given challenge with a high probability.*

Let the reference response to a challenge be the response obtained for the first trial. Let another response be obtained for any trial $i > 1$. The reproducibility is illustrated by Figure 2, where highlighted bits are bits ensuring this property.
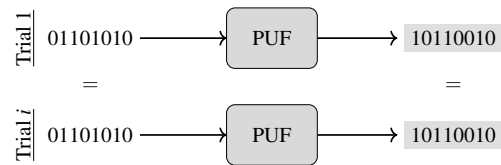


Figure 2: PUF reproducibility.

**Uniqueness.** *Two PUF circuits generate distinct responses for a common challenge with a high probability.*

Let us consider responses from any PUF $i$ and any PUF $j \neq i$ to the same challenge. The uniqueness is illustrated by Figure 3, where highlighted bits are bits ensuring this property.
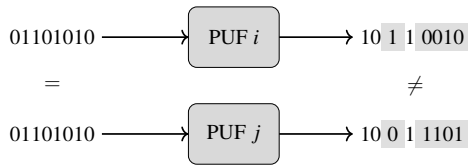
Figure 3: PUF uniqueness.

When the response of a PUF to a challenge is used for security applications, e.g. as a key, it is necessary to ensure additional properties (Maes, 2013; Maiti et al., 2012; Schaller et al., 2014).

**SRAM PUF.** A SRAM PUF is a silicon PUF which uses variations in the manufacturing of SRAM cells (Herder et al., 2014; Maes, 2013) as a source of randomness. The manufacturing of a SRAM cell induces physical differences between its two inverters (Maes, 2013) that will lead the SRAM cell to one of its two stable states on power-up, i.e. either "1" or "0". When the difference between inverters from a SRAM cell is small, the cell may be metastable (Maes, 2013) before getting its default value with a certain probability. Thus, an algorithm like *fuzzy extractor* can be used to provide a reliable PUF response extraction (Mostafa et al., 2020; Schrijen, 2018; Kang et al., 2013).

If such a PUF receives a challenge, the response may be generated by reading a set of cells determined by the challenge, e.g. a fixed number of cells selected based on a given address. It reads initial binary values from each selected SRAM cell on power-up to obtain a sequence of bits that will serve as the response from its CRP. SRAM PUF responses may be seen as a set of physically obfuscated keys (POKs) (Maes, 2013).

## 2.2 Ashur et al. (2018)'s Protocol

We build upon Ashur et al. (2018) and propose a cloud storage method in the following sections. The protocol uses a SRAM PUF as a basis for a privacy-preserving tracking system that allows for end-to-end encryption between IoT devices and application users. A new pair of ephemeral key and identity is generated for each message, by applying a key derivation function (KDF) with a key domain separator on a session key which is obtained from a hash chain.

**Context.** The tracking application using the selected protocol may be summarized as follows:

1. A tag is attached to personal belongings of a user and the user collects tag master keys by scanning a QR code provided with the tag;

2. The tag collects IDs from emitting beacons, used for the tracking, on Bluetooth Low Energy (BLE) at regular intervals;

3. The tag regularly sends, using Long Range (LoRa), the encrypted list of observed beacons, whose length is fixed, to dedicated LoRa Wide Area Network (LoRaWAN) gateways, first, and through a dedicated secure communication channel, eventually relying on intermediary servers and Transport Layer Security (TLS), to the application server using an ephemeral pseudonym derived from a PUF-based master key;

4. The corresponding user can collect location history data of its device from the application server (through Tor) and determine the device location.

**Protocol Description.** Ashur et al. (2018) propose to use a SRAM PUF based on a SRAM which is sufficiently large to be partitioned into $m > 1$ segments. Each segment produces a PUF response which is then used to generate a new master key $k_{tag}$, similarly to what Aysu et al. (2015) proposed and as illustrated by Figure 4. The latter is reproduced, and its uniformity and integrity are guaranteed, by using a robust version of the fuzzy extractor proposed by Kang et al. (2014).

At boot, a segment is selected and the corresponding $k_{tag}$ is used as the genesis of a hash chain of length $v$. Each hash value obtained from this hash chain is a session key $k_i$ which is given to a KDF together with a domain separator $d$ to generate a one-time pseudonym $p$ and a one-time encryption key $k_{AE}$. Once the KDF input is consumed, the next session key in the chain is computed. Once the end of the hash chain has been reached, the next SRAM segment is used to generate a new master key. The previous steps are also illustrated by Figure 4. To provide backward secrecy, the SRAM segment selection relies on $m$ fuses. Once a segment has been used, the corresponding fuse is destroyed and the segment becomes unreachable.

Each message $(p, c, t)$ sent to the server by the tag contains a one-time pseudonym $p$, the ciphertext $c$ and the message authentication tag $t$ generated from an authenticated encryption with associated data (AEAD), the associated data being the pseudonym. The user can send a request for the pseudonym $p$, and receives all entries for this pseudonym from the server. While the response may contain collisions, these cannot be decrypted.

The hash chains of the tag and the user terminal may become desynchronized. Knowing the transmission interval between messages, an upper bound on the number of messages sent since the last successful request can be computed and all pseudonyms related to the tag can be requested. This synchronization is performed, one request at a time, with a random delay between requests to avoid the linking between the tag and the user terminal and between pseudonyms.
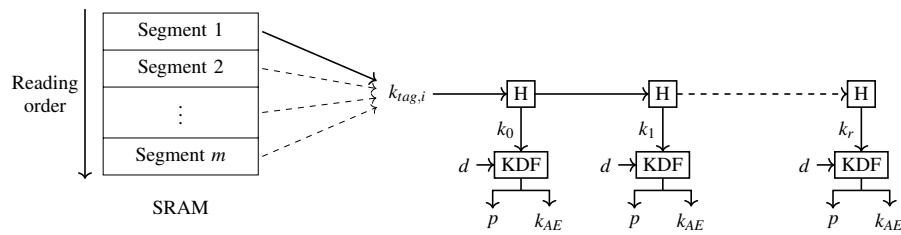
Figure 4: Memory segments are used in turn to produce a master key $k_{tag,i}$; the generation of one-time authenticated encryption keys $k_{AE}$ and corresponding pseudonyms $p$ from a master key is performed using a hash function $H$, a key derivation function *KDF* and a domain separator $d$ (Ashur et al., 2018).

**Limitations.** The main limitation of Ashur et al. (2018)'s protocol is the number of keys which may be generated. Even though it is not a problem in the context given by Ashur et al. (2018) — a lifetime of 5 years is obtained from 100 memory segments used as master keys, each generating 438 session keys and providing encryption for, at most, 18.25 days of records — it may not be sufficient for other applications. As such, the approach imposes a planned obsolescence for the PUF. Another issue is the method used to share keys with the user; the master keys are fixed and the user can not generate new keys if the previous ones were compromised or if an additional set of keys is required for a specific application.

# 3 PROPOSED SOLUTION

## 3.1 Threat Model

The application context from Ashur et al. (2018) is easily generalizable: sensing devices send data to remote terminals, eventually for distinct applications. The data passes through untrusted third-party servers before being fetched by the terminals.

Following the MITRE (2007) classification, our work aims to mitigate *communications* and, eventually, *supply chain* attacks. These domains of attack are non-exhaustively illustrated below:

**Communications.** A first possible attack on communications is a flooding attack. An adversary may perform a denial of service (DoS) attack by overflowing the server with a large quantity of packets and saturating either the server database or the channel. The user would have to download more entries, including fake ones, for each of their request. A second attack consists in the cryptanalysis of the traffic in order to find the content of messages and obtain cryptographic keys. If future generated keys depend on the keys which are found, the security of future communications is compromised.

**Supply Chain.** An adversary may interact with the device and its QR code before its delivery to the user. It would then be possible for the adversary to inject a spyware into the device, to collect master keys from the QR code or to alter it, or to clone the PUF if they act early in the supply chain.

## 3.2 Improved Protocol

Ashur et al. (2018)'s protocol can be "improved" by implementing together all the following proposals, which are described one by one to clarify their purpose. First, let $x_i$ be the key of size $K$, associated to segment $i$, being the result of a hash function $H$ implemented in the fuzzy extractor proposed by Kang et al. (2014), i.e. the key previously denoted by $k_{tag}$ in Ashur et al. (2018); it is referred as a "PUF response". Let also $k_{tag,i}$ be the master key associated to segment $i$. For simplicity, we assume that every hash function $H$ used in this work provides an output of $K$ bits.

**Proposal 1.** The first proposal allows the re-use of memory segments by cycling over them, i.e. jump to segment 1 after segment $m$ has been used. Since the re-use of previously generated master keys should be avoided, fresh master keys are generated on each cycle for each memory segment. Nonces help to achieve this goal, but they should be produced in a predictable way to retrieve data encrypted with generated keys.

Here, a *genesis nonce* $g_1$ is given as initial input to a hash chain. For each cycle $k$ and for each memory segment $i$, the hash value $r_{i,k}$ of $K$ bits is generated on the fly and a fresh master key $k_{tag,i,k} = x_i \oplus r_{1,i,k}$ is obtained, similarly to what is done with the one-time pad (OTP) encryption (Stallings, 2022). The result $k_{tag,i,k}$ is then used as a fresh master key (Algorithm 1).

**Proposal 2.** When iterating over the memory segments, an adversary may determine which segment generates the current master key if they know the iteration order of the segments. This second proposal consists in the pseudo-randomization of the order in which memory segments are visited. For each cycle,

```
generate g₁ = {0,1}^K;
foreach cycle k in cycles do until end of service
    foreach segment i in memory segments do
        // generate new nonce
        r_{1,i,k} ← H(g₁);
        // generate new master key
        k_{tag,i,k} ← x_i ⊕ r_{1,i,k};
        // overwrite with nonce
        g₁ ← r_{1,i,k};
    end
end
```

Algorithm 1: Master key re-freshing when cycling.

a nonce is generated using a hash chain and a *genesis nonce* $g_2$, as previously, and helps to determine the segment order. The randomization pseudo-code given by Algorithm 2 is inspired by a countermeasure against side-channel attacks proposed by Witteman (2017) and may be used for a memory providing a number of segments $m$ which is a power of two. However, it may be adapted, depending on the needs, when the number of segments is not a power of two.

```
generate g₂ = {0,1}^K;
foreach cycle k in cycles do until end of service
    // generate new nonce
    r_{2,k} ← H(g₂);
    for i from 0 to m − 1 do for each memory
      segment
        j ← (i + r_{2,k}) AND (m − 1);
        generate key from segment j;
    end
    // overwrite with nonce
    g₂ ← r_{2,k};
end
```

Algorithm 2: Cycle randomization, inspired by Witteman (2017).

**Proposal 3.** While the second proposal obfuscates the order in which memory segments are accessed, if an adversary manages to discover a PUF response and an iteration-ordered sequence of master keys from a cycle, they may XOR the known response with all master keys, apply $H$ on the results to obtain new potential hash values, XOR these values with the next master key in the captured sequence, for each master key, and repeat the process. The adversary succeeds once they retrieve a part of the $g_1$ hash chain and, hence, obtain associated PUF responses. This attack is possible because of the link between master keys in the generation procedure. A possible improvement is to perform the master key generation from the first proposal using $k_{tag,i,k} = x_i \oplus H(r_{1,i,k}||r_{2,k})$. This trick allows to break the apparent link between master keys.

**Proposal 4.** Previous proposals mask the PUF responses, cut links between the different master keys and make the segment order unpredictable for an adversary. However, to generate master keys on a terminal, PUF responses and corresponding genesis nonces must be stored on it. Thus, if they are retrieved by any malicious software on a terminal, the security of master keys dedicated to other applications resides in the ignorance of their genesis nonces. A method to prevent the leakage of the PUF responses, and hence a loss regarding the security of the master key generation for other applications, is to use *per-application PUF responses*. This method does not use responses as described in proposal 3 but replaces $x_i$ by $y_i$ with $y_i = H(x_i||r_{3,i})$, where $r_{3,i}$ is a nonce, dedicated to one memory segment $i$ and to one application, which should be computed with $r_{3,i} = H(s+i)$, where $s$ is a random value of $K$ bits used as a seed and $i$ is the segment index. By doing so, the pseudo-randomization of the segment order does not require to re-compute a hash chain for $r_{3,i}$ on each iteration, in the worst case. With this last proposal, the final expression to compute the master key is the following:

$$k_{tag,i,k} = y_i \oplus H(r_{1,i,k}||r_{2,k})$$
$$= H(x_i||H(s+i)) \oplus H(r_{1,i,k}||r_{2,k})$$

As a consequence, the **enrollment must be performed by the user** by connecting the device exploiting the PUF to their terminal. The application software exchanges genesis nonces with the device, and generates nonces if it does not have a random number generator, retrieves the PUF CRPs, the $i$-th response being $y_i$, and manages data securely.

Depending on the application, the genesis nonces may either be stored in NVM and be overwritten as the cursor moves along the hash chain, i.e. previous values cannot be accessed anymore, or a copy may be created and updated such that the original value may be retrieved from the NVM. However, the first option is more interesting for security reasons and is therefore recommended. Finally, in order to recover from a reboot, current iteration in the current cycle should be stored in the NVM.

# 4 DISCUSSION

## 4.1 System Properties

Benefits and limitations of our proposals are discussed below:

**Benefit 1 - Lifetime.** The lifetime of the PUF, as it is the case for the lifetime or the security of the tag,

is significantly extended by cycling over memory segments to generate new master keys.

**Benefit 2 - Supply Chain Security.** The security of the PUF supply chain is improved as no QR code allows one to collect all master keys. Instead, the enrollment is performed by the user themselves by connecting the PUF to the software managing the secure storage of the nonces and the CRPs.

**Benefit 3 - Increase Difficulty for the Adversary.**
By adding per-iteration nonces to the PUF responses, the discovering of these responses is made harder for the adversary. Moreover, they must retrieve the seed used for loop randomization and determine the current cycle to find the order in which memory segments are read.

**Benefit 4 - Helper Data Overhead.** The helper data overhead caused by the use of new nonces is reduced by using hash chains; only three random numbers are stored for an eventually large number of cycles over the memory segments.

**Benefit 5 - Per-Application Keys.** By generating two new genesis nonces with an "application-specific" enrollment, considering the device exploiting the PUF has enough memory to store additional helper data or the previous helper data is overwritten, it is possible to produce per-application keys using the same PUF responses.

**Limitation 1 - Synchronization Data Overhead.**
The fuzzy extractor used by Ashur et al. (2018), initially proposed by Kang et al. (2014), does not require to store any random number in the helper data it generates. Moreover, no constant has to be saved to allow the recovery of the current iteration after a reboot. These additional data require an extra space which is not negligible. However, thanks to the proposed use of the hashing chains to generate nonces, this overhead may be limited to $(3K + T)A$ bits, $A$ being the number of distinct applications relying on master keys generated from the PUF, $K$ being the number of output bits generated by the selected hash function, $T$ being the size in bits of the smallest word allowing to store the maximum number of iterations, i.e. the number of memory segments, per cycle (e.g. 1 byte is sufficient to iterate over maximum 256 memory segments, hence $t = 8$). This estimation only stands if no copy of the genesis nonces is stored, as previously recommended.

**Limitation 2 - Additional Computations.**
Additional computations are introduced to generate the new master keys. Note that the PUF construction and the fuzzy extractor are identical to the ones used in the original protocol design, and the method to derive pseudonyms and associated keys remains unchanged too. The additional computations can be evaluated in terms of hash computations. To generate new master keys for each cycle with our four proposals, one hash computation is required to obtain $r_{2,k}$ for the current cycle, one hash computation is required per memory segment to obtain $r_{1,i,k}$, and three hash computations are required per memory segment to obtain a master key with $k_{tag,i,k} = H(x_i||H(s+i)) \oplus H(r_{1,i,k}||r_{2,k})$. Thus, $4m + 1$ additional hash computations are required per cycle and per application, $m$ being the number of memory segments used by the PUF.

**Limitation 3 - Impact on Backward Secrecy.**
When a master key is discovered, the protocol from Ashur et al. (2018) limits the number of compromised messages to the hash chain length $L$. In the worst case, the improvement proposed in this work allows an adversary to discover $CL$ messages if they uncover the new PUF responses and the genesis nonces, where $C$ is the number of cycles performed over the memory segments.

**Limitation 4 - Storage of Genesis Nonces.** The two genesis numbers and the application seed must be stored in an NVM which may be read or manipulated if not secure. This may help the adversary to find PUF responses and impact backward secrecy, as described in the previous limitation. However, if genesis nonces are overwritten during the process, the forward secrecy should be preserved.

## 4.2 Security Analysis

The selected protocol provides several security services (Ashur et al., 2018; Nieles et al., 2017; Boyd et al., 2020; Shostack, 2014; Mees, 2020); none of these are altered by our proposals.

**Data Integrity.** *Any unauthorized information manipulation during a message transmission is detected.* This service is guaranteed thanks to the authenticated encryption.

**Data Origin and Entity Authentication.** *The source of the message is authentic.* Data integrity is required for this service. This service is guaranteed thanks to the authenticated encryption with associated data.

**Confidentiality.** *Unauthorized parties do not have access to the message content.* This service is guaranteed thanks to the authenticated encryption.

**Anonymity and Pseudonymity.** *It is impossible to distinguish an entity from other entities, or to link*

*an entity to an action or a message.* This service is guaranteed by the use of one-time pseudonyms, by introducing collisions between pseudonyms and by transferring all entries corresponding to a requested pseudonym on user's request.

**Unlinkability.** *It is impossible to link two or more entities, actions or messages.* This service is guaranteed by using one-time pseudonyms, by forcing the tag to send encrypted messages of fixed length at random intervals and by allowing the user to connect via Tor.

**Plausible Deniability.** *The identity of any actor, e.g. the source of a message, can not be proven by another entity.* This is performed thanks to the pseudonymity and the unlinkability.

**Backward Secrecy.** *The leakage of previously generated keys does not allow an adversary to discover future keys.* It is not possible to find keys related to unexplored memory segments, nor to find master keys of new cycles except if nonces and PUF responses are known.

**Forward Secrecy.** *The leakage of previously generated keys does not allow an adversary to discover other keys generated in the past.* It is not possible to find master keys of previous cycles, except if PUF responses and previous nonces are known.

# 5 RELATED WORK

Maes (2013) presents the different PUF properties in details and describes different PUF architectures. Two types of applications based on PUFs are illustrated: authentication and key generation.

Cambou et al. (2022) proposes a PUF-based method to generate session keys to establish a secure client-server channel based on password sets provided by the users. Zheng et al. (2023) proposes to use a trusted server as a starting point to provide peer-to-peer mutual authentication and encryption, and Guajardo et al. (2010) presents a combination of PUFs and biometrics to link patients and health monitoring devices to transmitted measurements for remote healthcare applications.

Lounis and Zulkernine (2021) analyses the security of fifteen IoT-dedicated authentication protocols, lists PUF-related threats and draws lessons from protocol misconceptions for future designs. Similarly, Mall et al. (2022) presents a survey of PUF-based authentication and key agreement (AKA) protocols for the IoT, wireless sensor networks and smart grids. It describes threats affecting them and compares their performance and their security.

Regarding applications, Skoric et al. (2007) and van der Leest and Tuyls (2013) propose to use a SRAM PUF as a secure key storage module, and Schrijen (2018) proposes a method based on a SRAM PUF to securely provide root keys to devices and to secure software images. Other works present anti-counterfeiting mechanisms (Schaller et al., 2014).

Rührmair (2022) provides a tutorial to achieve secret-free security and avoid PUF secret extraction risks. Examples illustrate how standard PUF constructions are not secret-free and why more advanced constructions present the secret-free property. Delvaux (2017) analyses the security of fuzzy extractors, as vectors of attacks on PUFs, and Müelich (2020) describes attacks on PUFs and solutions to prevent side-channel attacks on fuzzy extractor components.

Finally, Ferreira (2022) proposes a privacy-preserving authenticated key exchange protocol, based on a preshared pair of identity and master key, to derive new ephemeral identities and session keys.

# 6 CONCLUSION

We make four proposals to improve the privacy-preserving device tracking protocol by Ashur et al. (2018). These proposals improve the lifetime of the PUF, reduce risks related to the supply chain security and make the key generation less predictable for an adversary. We also enable a single PUF to generate keys for distinct applications, at the cost of a slightly higher power consumption and data overhead.

As future work, we envision to explore additional privacy-oriented objectives, e.g. *unobservability*, *content awareness* and *policy and consent compliance* (Deng et al., 2010). Unobservability — both anonymity regarding entities, actions and messages, and undetectability of dummy entities, actions and messages — is most relevant for application. Therefore, it would be interesting to propose a privacy-preserving mechanism to only allow authorized user to generate dummy traffic, while also rate-limiting this traffic. Cryptanalysis (Kraleva et al., 2023) of our proposals should be performed to evaluate if the generation of an unlimited number of keys is realistic.

## ACKNOWLEDGEMENTS

# REFERENCES

Anderson, R. (2021). *Security Engineering A Guide to Building Dependable Distributed Systems*. Wiley & Sons, Limited, John.

Ashur, T., Delvaux, J., Lee, S., Maene, P., Marin, E., Nikova, S., Reparaz, O., Rožić, V., Singelée, D., Yang, B., and Preneel, B. (2018). A privacy-preserving device tracking system using a low-power wide-area network. In *CANS*, pages 347–369. Springer.

Aysu, A., Gulcan, E., Moriyama, D., Schaumont, P., and Yung, M. (2015). End-to-end design of a puf-based privacy preserving authentication protocol. In Güneysu, T. and Handschuh, H., editors, *CHES 2015*, pages 556–576. Springer.

Babaei, A. and Schiele, G. (2019). Physical unclonable functions in the internet of things: State of the art and open challenges. *Sensors*, 19(14):3208.

Boyd, C., Mathuria, A., and Stebila, D. (2020). *Protocols for Authentication and Key Establishment*. Springer.

Cambou, B., Telesca, D., and Jacinto, H. S. (2022). PUF-protected methods to generate session keys. In *LNNS*, pages 744–764. Springer.

Delvaux, J. (2017). *Security Analysis Of PUF-Based Key Generation And Entity Authentication*. PhD thesis, KU Leuven.

Deng, M., Wuyts, K., Scandariato, R., Preneel, B., and Joosen, W. (2010). A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements. *Requirements Engineering*, 16(1):3–32.

Ferreira, L. (2022). Privacy-preserving authenticated key exchange for constrained devices. In *ACNS*, pages 293–312. Springer.

Graybeal, S. N. and McFate, P. B. (1989). Getting out of the starting block. *Scientific American*, 261(6):61–67.

Guajardo, J., Asim, M., and Petković, M. (2010). Towards reliable remote healthcare applications using combined fuzzy extraction. In *ISC*, pages 387–407. Springer.

Herder, C., Yu, M.-D. M., Koushanfar, F., and Devadas, S. (2014). Physical unclonable functions and applications: A tutorial. *Proceedings of the IEEE*, 102:1126–1141.

Kang, H., Hori, Y., Katashita, T., and Hagiwara, M. (2013). The implementation of fuzzy extractor is not hard to do : An approach using puf data. In *2013 SCIS*.

Kang, H., Hori, Y., Katashita, T., Hagiwara, M., and Iwamura, K. (2014). Cryptographic key generation from puf data using efficient fuzzy extractors. In *2014 ICACT*. IEEE.

Kraleva, L., Mahzoun, M., Posteuca, R., Toprakhisar, D., Ashur, T., and Verbauwhede, I. (2023). Cryptanalysis of strong physically unclonable functions. *IEEE OJ-SSCS*, 3:32–40.

Lounis, K. and Zulkernine, M. (2021). More lessons: Analysis of puf-based authentication protocols for iot. Cryptology ePrint Archive, Paper 2021/1509. https://eprint.iacr.org/2021/1509.

Maes, R. (2013). *Physically Unclonable Functions: Constructions, Properties and Applications*. Springer, 2013 edition.

Maiti, A., Gunreddy, V., and Schaumont, P. (2012). A systematic method to evaluate and compare the performance of physical unclonable functions. In *Embedded Systems Design with FPGAs*, pages 245–267. Springer.

Mall, P., Amin, R., Das, A. K., Leung, M. T., and Choo, K.-K. R. (2022). Puf-based authentication and key agreement protocols for iot, wsns, and smart grids: A comprehensive survey. *IEEE JIOT*, 9(11):8205–8228.

Mees, W. (2020). *Pragmatic cybersecurity*. Independently Published, Place of publication not identified.

MITRE (2007). Common Attack Pattern Enumeration and Classification. Last visit: 2023-05-15.

Mostafa, A., Lee, S. J., and Peker, Y. K. (2020). Physical unclonable function and hashing are all you need to mutually authenticate IoT devices. *Sensors*, 20(16):4361.

Müelich, S. (2020). *Channel coding for hardware-intrinsic security*. PhD thesis, Universität Ulm.

Nieles, M., Dempsey, K., and Pillitteri, V. Y. (2017). An introduction to information security. Technical report, National Institute of Standards and Technology.

Rührmair, U. (2022). Secret-free security: a survey and tutorial. *Journal of Cryptographic Engineering*.

Schaller, A., Arul, T., van der Leest, V., and Katzenbeisser, S. (2014). Lightweight anti-counterfeiting solution for low-end commodity hardware using inherent PUFs. In *TRUST*, pages 83–100. Springer.

Schrijen, G. J. (2018). Physical unclonable functions to the rescue: A new way to establish trust in silicon. In *2018 Embedded World*.

Shostack, A. (2014). *Threat modeling*. Wiley.

Skoric, B., Schrijen, G.-J., Tuyls, P., Ignatenko, T., and Willems, F. (2007). Secure key storage with PUFs. In *Security with Noisy Data*, pages 269–292. Springer.

Stallings, W. (2022). *Cryptography and Network Security Principles and Practice, Global Edition*. Pearson Education, Limited, 8 edition.

van der Leest, V. and Tuyls, P. (2013). Anti-counterfeiting with hardware intrinsic security. In *2013 DATE*, pages 1137–1142.

Witteman, M. (2017). Secure application programming in the presence of side channel attacks.

Zheng, Y., Liu, W., Gu, C., and Chang, C.-H. (2023). Puf-based mutual authentication and key exchange protocol for peer-to-peer iot applications. *IEEE TDSC*, 20(4):3299–3316.