Secure Lightweight Entity Authentication with Strong PUFs: Mission Impossible?

Jeroen Delvaux[1,2], Dawu Gu[2], Dries Schellekens[1] and Ingrid Verbauwhede[1]

[1] ESAT/COSIC and iMinds, KU Leuven, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{*jeroen.delvaux, dries.schellekens, ingrid.verbauwhede*}*@esat.kuleuven.be*
[2] CSE/LoCCS, Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai 200240, P.R. China
dwgu@sjtu.edu.cn

Abstract. Physically unclonable functions (PUFs) exploit the unavoidable manufacturing variations of an integrated circuit (IC). Their input-output behavior serves as a unique IC 'fingerprint'. Therefore, they have been envisioned as an IC authentication mechanism, in particular for the subclass of so-called strong PUFs. The protocol proposals are typically accompanied with two PUF promises: lightweight and an increased resistance against physical attacks. In this work, we review eight prominent proposals in chronological order: from the original strong PUF proposal to the more complicated converse and slender PUF proposals. The novelty of our work is threefold. First, we employ a unified notation and framework for ease of understanding. Second, we initiate direct comparison between protocols, which has been neglected in each of the proposals. Third, we reveal numerous security and practicality issues. To such an extent, that we can not support the use of any proposal in its current form. All proposals aim to compensate the lack of cryptographic properties of the strong PUF. However, proper compensation seems to oppose the lightweight objective.

Keywords: physically unclonable function, entity authentication, lightweight

1 Introduction

In this work, we consider a common authentication scenario with two parties: a **low-cost resource-constrained token** and a **resource-rich server**. Practical instantiations could be the following: RFID, smart cards and a wireless sensor network. One-way or possibly mutual entity authentication is the objective. The server has secure computing and storage at its disposal. Providing security is a major challenge however, given the requirements at the token side. Tokens typically store a secret key in non-volatile memory (NVM), using a mature technology such as EEPROM and its successor Flash, battery backed SRAM or fuses. Cryptographic primitives import the key and perform an authentication protocol. Today's problems are as follows. First, implementing cryptographic primitives in a resource-constrained manner is rather challenging. Second, an attacker can gain physical access to the integrated circuit (IC) of a token. NVM has proven to be vulnerable against physical attacks [19]: the secret is stored permanently in a robust electrical manner. Third, most NVM technologies oppose the low-cost objective. EEPROM/Flash requires floating gate transistors, resulting in additional manufacturing steps with respect to a regular CMOS design flow. Battery backed SRAM requires a battery. Circuitry to protect the NVM contents (e.g. a mesh of sensing wires) tends to be expensive.

Physically unclonable functions (PUFs) offer a promising alternative. Essentially, they are binary functions, with their input-output behavior determined by IC manufacturing variations. Therefore, they can be understood as a unique IC 'fingerprint', analogous to human biometrics. They might alleviate the aforementioned problems. Many PUFs allow for an implementation which is both resource-constrained and CMOS compatible. Furthermore, the secret is hidden in the physical structure of an IC, which is a much less readable format. Invasive attacks might easily destroy this secret, as an additional advantage. Several PUF-based protocols have been proposed, in particular for the subclass of so-called strong PUFs. We review the most prominent proposals: controlled PUFs [4–6], Öztürk et al. [15], Hammouri et al. [7], logically reconfigurable PUFs [10], reverse fuzzy extractors (FEs) [21], slender PUFs [14] and the converse protocol [11]. The novelty of our work is threefold. First, we employ a unified notation and framework for ease of understanding. Second, we initiate direct comparison between protocols, which has been neglected in each of the proposals. Third, we reveal numerous security and practicality issues. To such an extent, that we can not support the use of any proposal in its current form. All proposals aim to compensate the lack of cryptographic properties of the strong PUF. However, proper compensation seems to oppose the lightweight objective. The observations and lessons learned in this work might facilitate future protocol design.

The organization of this paper is as follows. Section 2 introduces notation and preliminaries. Section 3 analyzes the strong PUF protocols. Section 4 provides an overview of the protocol issues. Section 5 concludes the work. We operate at protocol level, considering PUFs as a black box. The protocol of Hammouri et al. is therefore discussed in Appendix B: a good understanding of PUF internals is required, as described in Appendix A.

2 Preliminaries

2.1 Notation

Binary vectors and matrices are denoted with a single bold character, e.g. $\boldsymbol{C} \in \{0,1\}^{1\times m}$. All vectors are row vectors. Their elements are selected with an index $i \geq 1$ between round brackets, e.g. $\boldsymbol{C}(1)$. The null vector is denoted as $\mathbf{0}$. Operations are the following: addition (XOR), e.g. $\boldsymbol{X} \oplus \boldsymbol{C}$, multiplication e.g. $\boldsymbol{E} \cdot \boldsymbol{H}^T$, concatenation e.g. $\boldsymbol{X}||\boldsymbol{C}$, and bit inversion, e.g. $\overline{\boldsymbol{R}}$. Variable assignment is denoted with an arrow, e.g. $d \leftarrow d-1$. Functions are printed in *italic*, with their input arguments between round brackets, e.g. Hamming weight $HW(\boldsymbol{R})$ and Hamming distance $HD(\boldsymbol{R_1}, \boldsymbol{R_2})$.

2.2 Physically Unclonable Functions: Black Box Description

The m-bit input and n-bit output of a PUF are referred to as challenge $\boldsymbol{C}$ and response $\boldsymbol{R}$ respectively. Unfortunately for cryptographic purposes, the behavior of the challenge-response pairs (CRPs) does not correspond with a random oracle. First, the response bits are not perfectly reproducible: noise and various environmental perturbations (supply voltage, temperature, etc.) result in **non-determinism**. The reproducibility (error rate) differs per response bit. Second, the response bits are **non-uniformly distributed**: bias and correlations are present. The latter might enable so-called modeling attacks. One tries to construct a predictive model for the PUF, given a limited set of training CRPs. Machine learning algorithms have proven to be successful [18].

PUFs are often subdivided in two classes, according to their number of CRPs. **Weak PUFs** offer few CRPs: their total content (2^{m+n} bits) is of main interest. Architectures typically consist of an array of identically laid-out cells (or units), each producing one response bit. E.g. the SRAM PUF [8] and the ring oscillator PUF[1] [20] are both very popular. The total bit-content scales roughly linear with the required IC area. Although there might be some correlation or a general trend among cells, a predictive model is typically of no concern. The response bits are mostly employed to generate a secret key, to be stored in volatile memory, in contrast to NVM. Post-processing logic, typically a fuzzy extractor (FE) [3], is required to ensure a reproducible and uniformly distributed key.

Strong PUFs offer an enormous number of CRPs, often scaling exponentially with the required IC area. They might greatly exceed the need for secret key generation and have been promoted primarily as lightweight authentication primitives. Architectures are typically able to provide a large challenge space (e.g. $m = 128$), but only a very small response space, mostly $n = 1$. CRPs are highly correlated, making modeling attacks a major threat. The most famous example is the arbiter PUF [12], described in appendix A.

2.3 Secure Sketch

The non-determinism of a PUF causes the regenerated instance of a response $\boldsymbol{R}$ to be slightly different: $\widetilde{\boldsymbol{R}} = \boldsymbol{R} \oplus \boldsymbol{E}$, with $HW(\boldsymbol{E})$ small. Secure sketches [3] are a useful reconstruction tool, as defined by a two-step procedure. First, public helper data is generated: $\boldsymbol{P} = Gen(\boldsymbol{R})$. Second, reproduction is performed:

---

[1] We consider the most usable read-out modes which avoid correlations, e.g. pairing neighboring oscillators.

$\boldsymbol{R} = \mathit{Rep}(\widetilde{\boldsymbol{R}}, \boldsymbol{P})$. Helper data $\boldsymbol{P}$ unavoidably leaks some information about $\boldsymbol{R}$, although this entropy loss is supposed to be limited. Despite the rather generic definition, two constructions dominate the implementation landscape, as specified below. Both the code-offset and syndrome construction employ a binary $[n, k, t]$ block code $\mathcal{C}$, with t the error-correcting capability. The latter construction requires a linear block code, as it employs the parity check matrix $\boldsymbol{H}$. Successful reconstruction is guaranteed for both constructions, given $HW(\boldsymbol{E}) \leq t$. Information leakage is limited to $n - k$ bits. The hardware footprint is asymmetric: *Gen* is better suited for resource-constrained devices than *Rep* [21].

	Gen	*Rep*
code-offset	Random $\boldsymbol{W} \in \mathcal{C}$	$\widetilde{\boldsymbol{W}} \leftarrow \widetilde{\boldsymbol{R}} \oplus \boldsymbol{P} = \boldsymbol{W} \oplus \boldsymbol{E}$
	$\boldsymbol{P} \leftarrow \boldsymbol{R} \oplus \boldsymbol{W}$	Error-correct $\widetilde{\boldsymbol{W}}$ to $\boldsymbol{W}$
		$\boldsymbol{R} \leftarrow \boldsymbol{P} \oplus \boldsymbol{W}$

	Gen	*Rep*
syndrome	$\boldsymbol{P} \leftarrow \boldsymbol{R} \cdot \boldsymbol{H}^T$	$\boldsymbol{S} \leftarrow \widetilde{\boldsymbol{R}} \cdot \boldsymbol{H}^T \oplus \boldsymbol{P} = \boldsymbol{E} \cdot \boldsymbol{H}^T$
		Determine $\boldsymbol{E}$
		$\boldsymbol{R} \leftarrow \widetilde{\boldsymbol{R}} \oplus \boldsymbol{E}$

3 Lightweight Authentication with Strong PUFs

We analyze all strong PUF authentication schemes in chronological order. One can read the sections in arbitrary order, although we highly recommend to read 3.1 first. All schemes employ two phases. The first phase is a one-time enrollment in a secure environment, following IC manufacturing. The server then obtains some information about the PUF, CRPs or even a predictive model via machine learning, to establish a shared secret. The destruction of one-time interfaces might permanently restrict the PUF access afterwards. The second phase is in-the-field deployment, where tokens are vulnerable to physical attacks. Token and server then authenticate over an insecure communication channel. In general: challenge $\boldsymbol{C}$ and response $\boldsymbol{R}$ are required to be of sufficient length, e.g. $m = n = 128$, to counteract brute-force attacks and random guessing.

For proper assessment, we define two reference authentication methods. Reference I employs a token with a secret key $\boldsymbol{K}$ programmed in NVM, as represented by Figure 1(a). Additional cryptographic logic performs the authentication. For ease of comparison, we opt for a hash function, hereby limiting ourselves to token authenticity only. The server checks whether a token can compute $a \leftarrow \mathit{Hash}(\boldsymbol{K}, \boldsymbol{N})$, with $\boldsymbol{N}$ a random nonce. Reference II employs PUF technology, potentially providing more physical security at a lower manufacturing cost. We employ a weak PUF[2] to generate a secret key, as represented by Figure 1(b). The reproducibility and non-uniformity issue are resolved in a sequential manner, using a FE. A secure sketch first ensures reproducibility. *Gen* is executed only once during enrollment. The public helper bits are stored by the server, or alternatively at the token side in insecure off-chip NVM. A hash function performs entropy compression, hereby compensating the non-uniformity of $\boldsymbol{R}$, in addition to the entropy loss caused by the helper data. One could generate a key as $\boldsymbol{K} \leftarrow \mathit{Hash}(\boldsymbol{R})$. We perform an optimization by merging this step with the authentication.

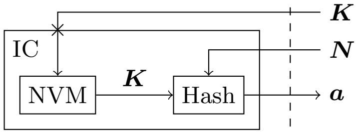
3.1 Naive Authentication

The most simple authentication method employs an unprotected strong PUF only [16], as shown in Figure 1(c). Figure 2 represents the corresponding protocol. The server collects a database of d arbitrary CRPs during enrollment. We assume the use of a true random number generator (TRNG). A genuine token should be able to reproduce the response for each challenge in the server database. Only an approximate match is required, taking PUF non-determinism into account: Hamming distance threshold ϵ implements this. To avoid token impersonation via replay, CRPs are discarded after use, limiting the number of authentications to d. Choosing e.g. $m = 128$, an attacker can not gather all CRPs and impersonate a token as such. Choosing e.g. $n = 128$, randomly guessing $\boldsymbol{R}$ is extremely unlikely to succeed.

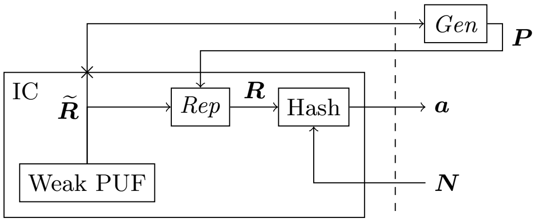
Modeling Attacks Strong PUFs are too fragile for unprotected exposure, as demonstrated by a history of machine learning attacks [18]. A predictive PUF model would enable token impersonation.

---

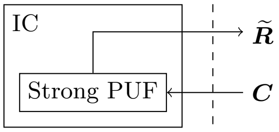
[2] Logic for generating challenges is implicitly present and might be as simple as reading out the full cell array.



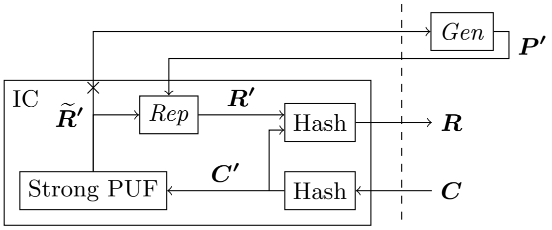
(a) Reference I: key in NVM.



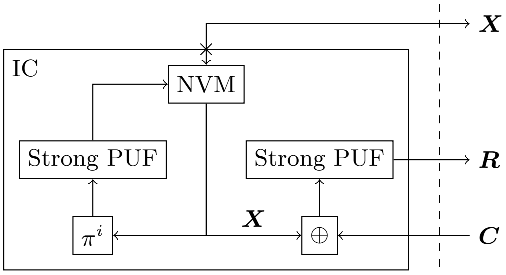
(b) Reference II: key via PUF and FE.



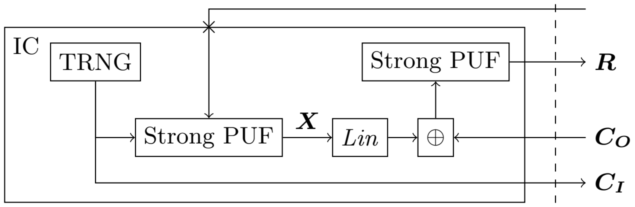
(c) Naive strong PUF [16].



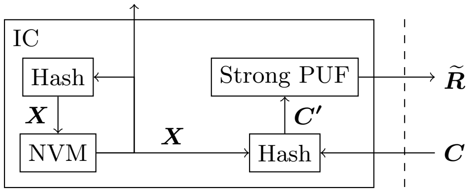
(d) Controlled PUF [4–6].



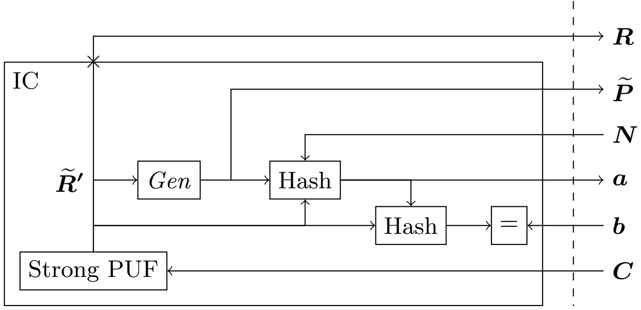
(e) Öztürk et al [15].



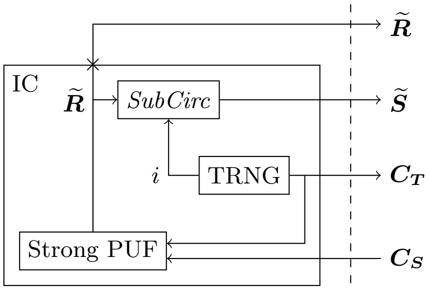
(f) Hammouri et al [7].



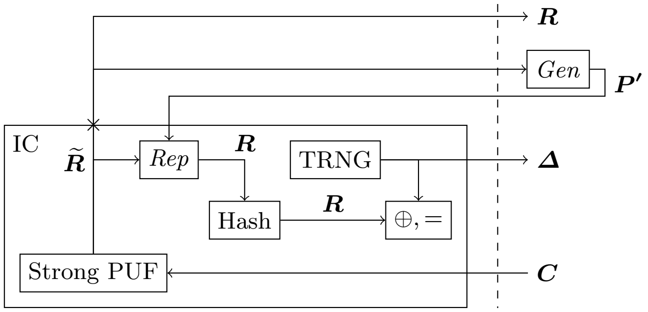
(g) Logically reconfigurable PUF [10].



(h) Reverse FE [21].



(i) Slender PUF [14].



(j) Converse [11].

Fig. 1. Token representation for all protocols and the two references. The following IC logic is not drawn: expansion of the strong PUF responses, intermediary registers (volatile) and control. A dashed line represent the interface with the server. One-time interfaces destructed after enrollment are marked by the symbol ×.

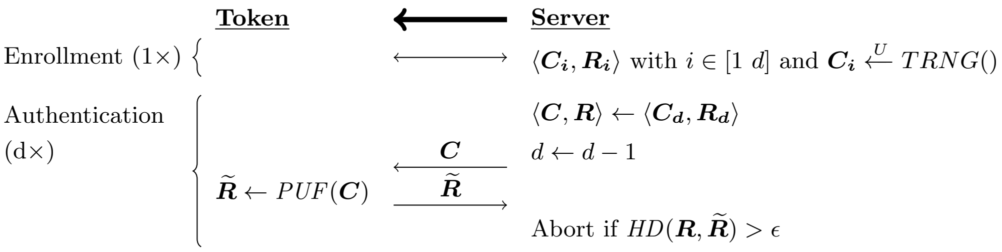


Fig. 2. Naive authentication protocol. The thick arrow points from verifier to prover.

So far, no architecture can claim to be practical, well-validated and robust against modeling. Two fundamental problems undermine the optimism for a breakthrough. First, strong PUFs extract their enormous amount of bits from a limited IC area only, hereby using a limited amount of circuit elements. A highly correlated structure is the unavoidable consequence. The arbiter PUF model in appendix A.2 provides some insights in this matter. Second, the more complicated the structure of the PUF, the more robust against modeling, but the less reproducible the response, as local non-determinism propagates to the PUF output. Appendix A.3 provides some insights in this matter.

Limited Response Space In practice, strong PUFs provide a limited response space only, often $n = 1$. Replicating the PUF circuit is a simple but unfortunately very expensive solution. The lightweight approach is to evaluate a list of n challenges, hereby concatenating the response bits. Various methods can be employed to generate such a list. The server could generate the list, requiring no additional IC logic, but resulting in a large communication overhead. A small pseudorandom number generator (PRNG), such as a linear feedback shift register (LFSR), is often employed. Challenge $\boldsymbol{C}$ is then used as a seed value: $\boldsymbol{R} \leftarrow PUF(PRNG(\boldsymbol{C}))$. A variety of counter-based solutions can be applied as well. All protocol proposals in the remainder of this work typically suggest a particular response expansion method. We make abstraction of this, except when there is a related security problem.

3.2 Controlled PUFs

Controlled PUFs [4–6] provide reinforcement against modeling via a cryptographic hash function (one-way). Two instances, preceding and succeeding the PUF respectively, are shown in Figure 1(d). Figure 3 represents the corresponding protocol[3]. The preceding instance eliminates the chosen-challenge advantage of an attacker. The succeeding instance hides exploitable correlations due to the PUF structure. The latter hash seems to provide full protection by itself, but requires the use of a secure sketch: its avalanche effect would trigger on a single bit flip in $\widetilde{\boldsymbol{R}}'$. CRP stored by the server are accompanied by helper data.

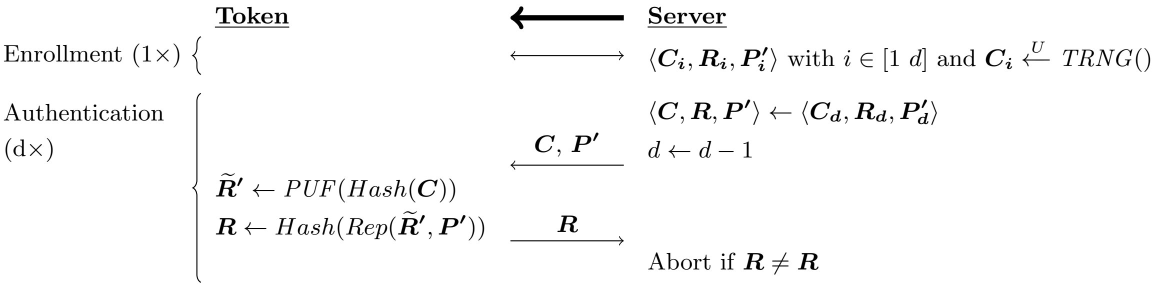


Fig. 3. Authentication with controlled PUFs.

---

[3] Controlled PUFs were proposed in a wider context than CRP-based token authentication only.

Inferior to reference II The proposal seems to be inferior to reference II. First, the PUF is required to have an enormous instead of modest input-output space. This is inherently more demanding, even if one would extend reference II with a few challenge bits to enable the use of multiple keys. Second, server storage requirements scale proportionally with the number of authentications, in contrast to constant-size.

3.3 Öztürk et al.

Öztürk et al. [15] employ two easy-to-model PUFs, as shown in Figure 1(e). Figure 4 represents the corresponding protocol. Both PUFs are assumed to possess a large response space, without defining challenge expansion logic however. The outer PUF foresees the token with CRP behavior. To prevent its modeling by an attacker, an internal secret $\boldsymbol{X}$ is XORed within the challenge path. A feedback loop, containing a (repeated) permutation and an inner PUF, is employed to update $\boldsymbol{X}$ continuously. During enrollment, the server has both read and write access to $\boldsymbol{X}$ via a one-time interface. This allows the server to construct models for either PUF, followed by an initialization of $\boldsymbol{X}$. The server has to keep track of $\boldsymbol{X}$, which is referred to as synchronization. The non-determinism of the inner PUF makes this non-trivial. One assumes an excellent match between the responses of the inner PUF and its model. At most one bit of $\boldsymbol{X}$ is assumed to be affected, in the seldom case of occurrence. An authentication failure (violation of ϵ) provides an indication thereof. One proposes a simple recovery procedure at the server side: bits of $\boldsymbol{X}$ are successively flipped until the authentication succeeds.

	Token	$\longleftarrow$	**Server**
Enrollment	Secure NVM: $\boldsymbol{X}$	$\longleftrightarrow$	Train models $\widetilde{PUF_I}$ and $\widetilde{PUF_O}$
(1×)			$\boldsymbol{X} \overset{U}{\leftarrow} TRNG()$
Authentication		$\xleftarrow{\boldsymbol{C}}$	$\boldsymbol{C} \overset{U}{\leftarrow} TRNG()$
(∞)	$\forall i : \boldsymbol{X}(i) \leftarrow PUF_I(\pi^i(\boldsymbol{X}))$		
	$\boldsymbol{R} \leftarrow PUF_O(\boldsymbol{X} \oplus \boldsymbol{C})$	$\xrightarrow{\boldsymbol{R}}$	
			$\forall i : \boldsymbol{X}(i) \leftarrow \widetilde{PUF_I}(\pi^i(\boldsymbol{X}))$
			$\boldsymbol{R'} \leftarrow \widetilde{PUF_O}(\boldsymbol{X} \oplus \boldsymbol{C})$
			Abort if $HD(\boldsymbol{R}, \boldsymbol{R'}) > \epsilon$

Fig. 4. Authentication protocol of Öztürk et al.

Issues Regarding $\boldsymbol{X}$ There are several issues related to the use of $\boldsymbol{X}$. First, it implicates the need for secure reprogrammable NVM, hereby undermining the advantages of PUFs. Either read or write access would enable an attacker to model the system, as during enrollment. Second, the synchronization effort is too optimistic. PUFs and their models typically have a 3% error rate. The server faces a continuous synchronization effort, not necessarily limited to a single error. Third, it enables denial-of-service attacks. An attacker can collect an (unknown) large number of CRPs, desynchronizing $\boldsymbol{X}$. Propagation of errors across authentications makes the recovery effort rapidly infeasible.

Feedback Loop Comments The permutation has to be chosen carefully. Repetition is bound to occur, as determined by the order k: $\pi^k = \pi$. This would implicate $\boldsymbol{X}$ to have identical bits, opposing its presumed non-uniformity. A simple simulation confirms the significance for 64-bit vectors. The estimated probability of a randomly chosen permutation to have $k \leq 63$ equals $\approx 9\%$. Furthermore, the need for the inner PUF is questionable. First, its non-determinism poses a limit on the complexity of the feedback loop, and hence the modeling resistance of the overall system. Second, the outer PUF and the initialization of $\boldsymbol{X}$ already induce IC-specific behavior. Using a cryptographic hash function as feedback would resolve all foregoing comments. The resulting system would then be remarkably similar to a later proposal: logically reconfigurable PUFs [11].

3.4 Logically Reconfigurable PUF

Logically reconfigurable PUFs [10] were proposed in order to make tokens recyclable, hereby reducing electronic waste. An internal state $\boldsymbol{X}$ is therefore mixed into the challenge path, as shown in Figure 1(e). Read access to $\boldsymbol{X}$ is allowed, although not required. There is no direct write access, although one can perform an update: $\boldsymbol{X} \leftarrow Hash(\boldsymbol{X})$. Figure 5 represents the authentication protocol.

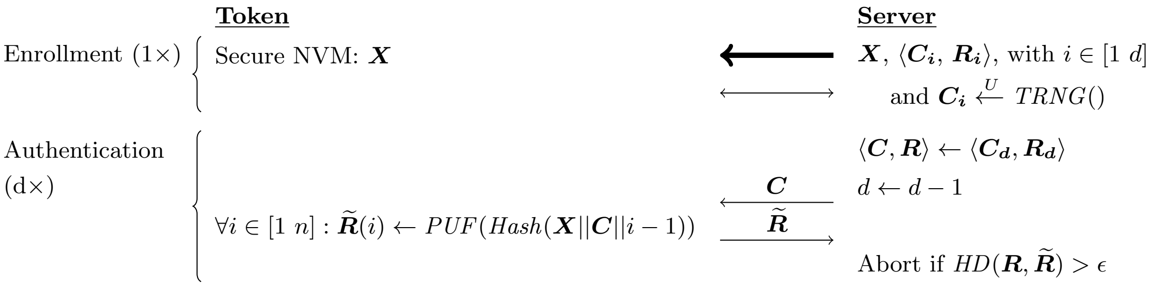


Fig. 5. Authentication protocol for logically reconfigurable PUFs.

Exacting PUF Requirements The proposal does not aim to prevent PUF modeling attacks, despite providing forward/backward security proofs with respect to $\boldsymbol{X}$. Therefore, the practical value of the proposal is rather questionable: the protocol can not be instantiated due to lack of an appropriate strong PUF.

Issues related to NVM The proposal requires reprogrammable write-secure NVM, which is not free of issues. First, it undermines a main advantages of PUFs: low-cost manufacturing. Second, it enables denial-of-service attacks. An attacker can update $\boldsymbol{X}$ one or more times, invalidating the CRP database of the server. The proposal does not describe an authentication mechanism for the reconfiguration.

3.5 Reverse Fuzzy Extractor

The reverse FE proposal [21] provides mutual authentication, in contrast to previous work. The term 'reverse' highlights that *Gen* and not *Rep* is implemented as token hardware, as shown in Figure 1(h). As such, one does benefit of the lightweight potential of *Gen*. Figure 6 represents the protocol[4]. One author proposed a modified version in [13], with a reversal of authentication checks stated to be the main difference.

PUF Non-Determinism Non-determinism is essential to avoid replay attacks. For tokens, one does reuse the PUF for this purpose, hereby avoiding the need for a TRNG. A lower bound for the PUF non-determinism is imposed, in order to avoid server impersonation. However, this seems to be a very delicate balancing exercise in practice, opposing more conventional design efforts to stabilize PUFs. The need for environmental robustness (perturbations of outside temperature, supply voltage, etc.) magnifies this opposition. An attacker might immerse the IC in an extremely stable environment, hereby minimizing the PUF error rate. For genuine use however, one might have to provide correctness despite large perturbations, corresponding to the maximum error rate. The modified protocol proposal [13] does not discuss the foregoing contradiction, although its use of a token TRNG provides a resolution.

---

[4] The use of a token identifier (public, could be stored in insecure off-chip NVM) is omitted for simplicity, as it seems to have no impact on security. The server has been maintained as protocol initiator.

	Token	$\boldsymbol{\longleftrightarrow}$	**Server**
Enrollment (1×)		$\longleftrightarrow$	$\langle \boldsymbol{C_i}, \boldsymbol{R_i}\rangle$ with $i \in [1\ d]$ and $\boldsymbol{C_i} \overset{U}{\leftarrow} \mathit{TRNG}()$
Authentication (∞)			$\langle \boldsymbol{C}, \boldsymbol{R}\rangle \leftarrow \langle \boldsymbol{C_j}, \boldsymbol{R_j}\rangle$ with $j \in [1\ d] \overset{U}{\leftarrow} \mathit{TRNG}()$
		$\xleftarrow{\boldsymbol{C}, \boldsymbol{N}}$	$\boldsymbol{N} \overset{U}{\leftarrow} \mathit{TRNG}()$
	$\widetilde{\boldsymbol{R}} \leftarrow \mathit{PUF}(\boldsymbol{C})$		
	$\widetilde{\boldsymbol{P}} \leftarrow \mathit{Gen}(\widetilde{\boldsymbol{R}})$		
	$\boldsymbol{a} \leftarrow \mathit{Hash}(\boldsymbol{N}, \widetilde{\boldsymbol{R}}, \widetilde{\boldsymbol{P}})$	$\xrightarrow{\widetilde{\boldsymbol{P}}, \boldsymbol{a}}$	
			$\widetilde{\boldsymbol{R}} \leftarrow \mathit{Rep}(\boldsymbol{R}, \boldsymbol{P})$
			Abort if $\mathit{Hash}(\boldsymbol{N}, \widetilde{\boldsymbol{R}}, \widetilde{\boldsymbol{P}}) \neq \boldsymbol{a}$
		$\xleftarrow{\boldsymbol{b}}$	$\boldsymbol{b} \leftarrow \mathit{Hash}(\boldsymbol{a}, \widetilde{\boldsymbol{R}})$
	Abort if $\mathit{Hash}(\boldsymbol{a}, \widetilde{\boldsymbol{R}}) \neq \boldsymbol{b}$		

Fig. 6. Reverse FE protocol (token identifier omitted).

PRNG Exploitation The proof-of-concept implementation employs a 64-bit arbiter PUF in combination with a PRNG to expand its response space: $\widetilde{\boldsymbol{R}} \leftarrow \mathit{PUF}(\mathit{LFSR}(\boldsymbol{C}))$. The concatenation of 255 arbiter response bits forms a single response $\boldsymbol{R}$. A [n=255, k=21, t=55] BCH code has been employed, allowing for an efficient implementation of *Gen*. One generates 7 responses (and helper sets) from a single $\boldsymbol{C}$, aiming at a security level of $7k > 128$ bits.

The proposal does not provide an explicit warning to refuse fixed points of the LFSR, e.g. $\boldsymbol{C} = \boldsymbol{0}$. This would have been appropriate however, in order to avoid a trivial server impersonation attack. Fixed points will result in either $\widetilde{\boldsymbol{R}} = \boldsymbol{0}$ or $\widetilde{\boldsymbol{R}} = \overline{\boldsymbol{0}}$, assuming stability for one particular response bit. An attacker can guess $\boldsymbol{b}$ with probability 1/2. Another issue, which is far less obvious and hence more serious, is described next.

The proposal raises the concern of repeated helper data exposure. An attacker might collect helper data $\{\widetilde{\boldsymbol{P}}_1, \widetilde{\boldsymbol{P}}_2, \ldots\}$ for the same challenge $\boldsymbol{C}$, trying to retrieve $\boldsymbol{R}$. Therefore, one does recommend the syndrome construction, as there is provably no additional leakage with respect to the traditional $n - k$ entropy loss[5]. However, the following threat has been overlooked: helper data exposure for correlated CRPs, as enabled by the PRNG.

We first consider the (unrealistic but desired) case of a perfectly deterministic PUF. Helper data exposure can be understood via an underdetermined system of linear equations $\boldsymbol{H} \cdot \boldsymbol{R}^T = \boldsymbol{P}^T$, having $n - k = 234$ equations for $n = 255$ unknowns. Challenge $\boldsymbol{C_1}$ provided by the server is employed as the LFSR initial value, resulting in $n = 255$ challenges after expansion, leading to a response $\boldsymbol{R} = \big(r_1\ r_2\ \ldots\ r_{255}\big)$. One could easily construct a challenge $\boldsymbol{C_2}$ leading to a response $\boldsymbol{R} = \big(r_2\ r_3\ \ldots\ r_{256}\big)$. Subsequently, both system of equations can be merged, as shown below. There are $2(n - k) = 468$ equations for $n + 1 = 256$ unknowns, which seems to be solvable at first sight.

$$\begin{pmatrix} \boldsymbol{H} & \boldsymbol{0}^T \\ \boldsymbol{0}^T & \boldsymbol{H} \end{pmatrix} \big(r_1\ r_2\ \ldots\ r_{256}\big)^T = \begin{pmatrix} \boldsymbol{P_1}^T \\ \boldsymbol{P_2}^T \end{pmatrix}$$

However, equation dependencies have to be considered. Therefore, we distinguish two types of block codes. First, the case of a non-cyclic code. Our experiments indicate that there are n independent equations for $n + 1$ unknowns, leading to a successful attack. Helper data leakage can not distinguish between $\boldsymbol{R}$ and $\overline{\boldsymbol{R}}$, since $r_i \oplus r_j = \overline{r_i} \oplus \overline{r_j}$. Second, the case of a cyclic code (e.g. BCH). Our experiments indicate that there are $n - k + 1$ independent equations for $n + 1$ unknowns, which does not benefit an attacker. However, an alternative method does circumvent this limitation. Arbiter PUFs can be modeled with only a few thousand CRPs, using machine learning techniques. By repeatedly merging helper data, one can obtain a system of $n - k + q$ independent equations for $n + q$ unknowns, given an arbitrary $q \geq 1$. We choose e.g. $q = 10000$, including both training and verification CRPs. Subsequently, machine

---

[5] Apart from security, one does not mention that the code-offset construction seems less practical: it requires a random codeword.

learning is performed in a brute-force manner, guessing the $k = 21$ unknowns, which seems to be feasible. The correct guess would result in the observable event of a high modeling accuracy.

We now consider a PUF which is not perfectly deterministic. To minimize this inconvenience, an attacker could ensure the IC's environment to be very stable. Code parameters, which are normally chosen in order to maintain robustness against a variety of environmental perturbations (supply voltage, outside temperature, etc.), become relatively relaxed then. Subsequently, we consider a transformation $\boldsymbol{H}^\star = \boldsymbol{T} \cdot \boldsymbol{H}$, which selects linear combinations of the rows of $\boldsymbol{H}$. The transformation is chosen so that the rows of $\boldsymbol{H}^\star$ do have a low Hamming weight. Many algorithms could find a suitable transformation. Transforming $\boldsymbol{H}$ to reduced row echelon form, possibly after permuting its columns, might be a good start for instance.

Suboptimal Even with $d = 1$, the protocol still allows for an unlimited number of authentications. Token impersonation via replay has already been prevented by the use of nonce $\boldsymbol{N}$. This observation could result in numerous protocol simplifications, as has been acknowledged in the proposing paper. However, one does not state clearly that there would a simplification for the PUF as well: the enormous input-output space is no more required, even a weak PUF could be employed. Despite all the former, one still promotes the use of $d > 1$, arguing that it offers an increased side channel resistance. We argue that this countermeasure might not outweigh the missed advantages and that there might be numerous more rewarding countermeasures. The modified protocol proposal [13] does not discuss the foregoing matter, although it uses $d = 1$, hereby providing a weak PUF with implicit challenge as an example.

3.6 Slender PUF

The slender PUF proposal [14] includes three countermeasures against modeling, while avoiding the need for cryptographic primitives, as clear from Figure 1(i). First, one employs a strong PUF with a high resistance, which is modeled during enrollment via auxiliary one-time interfaces. One does propose the XOR arbiter PUF (see appendix A.3). Second, the exposure of $\boldsymbol{R}$ is limited to random substrings $\boldsymbol{S}$, hereby obfuscating the CRP link. The corresponding procedure *SubCirc* treats the bits in a circular manner. Third, server and token both contribute to the challenge via their respective nonces $\boldsymbol{C_S}$ and $\boldsymbol{C_T}$, counteracting chosen-challenge attacks. Figure 7 represents the protocol.

$$\begin{array}{llcl}
 & \underline{\textbf{Token}} & \boldsymbol{\longleftarrow} & \underline{\textbf{Server}} \\
\text{Enrollment } (1\times)\ \{ & & \longleftrightarrow & \text{Train model } \widetilde{PUF} \\
\text{Authentication} & & \xleftarrow{\boldsymbol{C_S}} & \boldsymbol{C_S} \overset{U}{\leftarrow} TRNG() \\
(\infty) & \boldsymbol{C_T} \overset{U}{\leftarrow} TRNG() & & \\
 & \widetilde{\boldsymbol{R}} \leftarrow PUF(\boldsymbol{C_S}, \boldsymbol{C_T}) & & \\
 & j \in [1\ n] \overset{U}{\leftarrow} TRNG() & & \\
 & \widetilde{\boldsymbol{S}} \leftarrow SubCirc(\widetilde{\boldsymbol{R}}, j) & \xrightarrow{\boldsymbol{C_T},\ \boldsymbol{S}} & \\
 & & & \widetilde{\boldsymbol{R}} \leftarrow \widetilde{PUF}(\boldsymbol{C_S}, \boldsymbol{C_T}) \\
 & & & \text{Abort if } \forall j : HD(SubCirc(\widetilde{\boldsymbol{R}}, j), \widetilde{\boldsymbol{S}}) > \boldsymbol{\epsilon}
\end{array}$$

Fig. 7. Slender PUF protocol.

PRNG Exploitation The proof-of-concept implementation employs a PRNG to expand the PUF response space: $\widetilde{\boldsymbol{R}} \leftarrow PUF(PRNG(\boldsymbol{C_S}, \boldsymbol{C_T}))$. However, the employed construction might allow for token impersonation: $PRNG(\boldsymbol{C_S}, \boldsymbol{C_T}) = LFSR(\boldsymbol{C_S}) \oplus LFSR(\boldsymbol{C_T})$. We assume an identical feedback polynomial for both LFSRs, which is the most intuitive assumption[6]. A malicious token might then

---

[6] The proof-of-concept implementation employs 128-bit LFSRs, with $\boldsymbol{C_S}$ and $\boldsymbol{C_T}$ as the initial states, without specifying feedback polynomials. Furthermore, FPGA implementation results (Table III) strongly suggest the use of identical feedback polynomials.

return $\boldsymbol{C_T} \leftarrow \boldsymbol{C_S}$, resulting in an expanded challenge $\boldsymbol{C'} = \boldsymbol{0}$. The server's PUF model outputs either $\widetilde{\boldsymbol{R}} = \boldsymbol{0}$ or $\widetilde{\boldsymbol{R}} = \overline{\boldsymbol{0}}$. So provided a substring $\boldsymbol{S} = \boldsymbol{0}$, authentication does succeed with a probability 1/2. Via replay, one could increase the probability to 1. We require eavesdropping on a single genuine protocol execution: $\boldsymbol{C_S^{(1)}}$, $\boldsymbol{C_T^{(1)}}$ and $\boldsymbol{S^{(1)}}$. The malicious prover gets authenticated with the following: $\boldsymbol{C_T^{(2)}} \leftarrow \boldsymbol{C_S^{(2)}} \oplus \boldsymbol{C_S^{(1)}} \oplus \boldsymbol{C_P^{(1)}}$ and $\boldsymbol{S^{(2)}} \leftarrow \boldsymbol{S^{(1)}}$. The expanded challenges are equal.

Exacting PUF Requirements The PUF requirements are rather exacting and partly opposing. On one hand, the PUF should be easy-to model, requiring a highly correlated structure. On the other hand, CRP correlations do enable statistical attacks, due to the lack of cryptographic primitives. Although the XOR arbiter PUF seems to offer this delicate balance, it comes at a price: several one-time interfaces[7], high response non-determinism and a limited modeling accuracy. Furthermore, the user's control regarding the challenge list should be highly restricted: statistical attacks are very powerful if an attacker has a (partial) chosen-challenge advantage. The PRNG-TRNG construction should accomplish this goal. Statistical attacks exploit the knowledge of a function $P(r_u = r_v) = f(\boldsymbol{C}_u, \boldsymbol{C}_v)$ for a certain strong PUF architecture. CRPs with $|P(r_u = r_v) - 1/2| > 0$ are correlated and hence exploitable. One might be able to retrieve index j as such[8], circumventing the *SubCirc* countermeasure. This subsequently enables machine learning attacks, as the CRP link has been restored. The proposal plots f for the arbiter PUF and its XOR variant based on simulations. However, we believe the simulation results of the XOR variant to be erroneous, and provide a correction in appendix A.4. Although the correlations have been underestimated, the larger XOR variants might still offer security.

3.7 Converse Authentication

Figures 1(j) and 8 represent the converse authentication protocol [11]. The authentication is one-way, in a less conventional setting where tokens verify the server. The difference vector (XOR) between two responses is denoted as $\boldsymbol{\Delta}$. The restriction $\boldsymbol{\Delta} \neq 0$ prevents an attacker from impersonating the server, choosing $\{\boldsymbol{C_i} = \boldsymbol{C_j}, \boldsymbol{P_i} = \boldsymbol{P_j}\}$. Optionally, one can extend the protocol with the establishment of a session key $Hash(\boldsymbol{R_i} || \boldsymbol{R_j})$. The attacker capabilities are restricted: invasive analysis of the prover IC is assumed to be impossible. Furthermore, one assumes an eavesdropping attacker, trying to impersonate the server given a log of genuine authentication transcripts.

	Token	$\longrightarrow$	**Server**
Enrollment (1×)		$\longleftrightarrow$	$\langle \boldsymbol{C_i}, \boldsymbol{P_i}, \boldsymbol{R_i} \rangle$ with $i \in [1\ d]$ and $\boldsymbol{C_i} \overset{U}{\leftarrow} TRNG()$
Authentication (∞)	$\boldsymbol{\Delta} \neq 0 \overset{U}{\leftarrow} TRNG()$	$\xrightarrow{\boldsymbol{\Delta}}$	
		$\xleftarrow{\boldsymbol{C_i}, \boldsymbol{P_i}, \boldsymbol{C_j}, \boldsymbol{P_j}}$	Find i, j such that $\boldsymbol{R_i} \oplus \boldsymbol{R_j} = \boldsymbol{\Delta}$ If none, choose random i, j.
	$\boldsymbol{R_i} \leftarrow Hash(Rep(PUF(\boldsymbol{C_i}), \boldsymbol{P_i}))$ $\boldsymbol{R_j} \leftarrow Hash(Rep(PUF(\boldsymbol{C_j}), \boldsymbol{P_j}))$ Abort if $\boldsymbol{R_i} \oplus \boldsymbol{R_j} \neq \boldsymbol{\Delta}$		

Fig. 8. Converse authentication protocol.

---

[7] Chains are modeled separately via machine learning, before XORing takes place.

[8] An identical attack could result in full key-recovery for the protocol predecessor: the pattern matching key generator [17], which serves as an alternative for the fuzzy extractor (reference II). This risk has not been in acknowledged in either paper. We note that another statistical attack of fundamentally different nature has recently been published for the predecessor [2].

Attacker Overly Restricted The attacker capabilities are unclear and overly restricted to be practical. First, restricting invasion would automatically extend to physical attacks in general. The need for PUFs is hereby severely reduced. It is not clear whether an attacker is allowed to freely query the server, as exploited hereafter. We argue that this should be possible, as the protocol is initiated by the token.

Scalability Issues The probability of success for a randomly guessing attacker would be $1/2^n$, when trying to impersonate the server. In certain sense, the server faces a similar probability, posing an upper limit on n for practicality reasons. Successful authentication of the server relies on the availability of a given $\boldsymbol{\Delta}$ within its database. A scalable method to (construct and) search within a database was not discussed and seems far from obvious. Assume a cumbersome trial and error procedure as a search method: a pairwise selection has a probability of $1/2^n$ to be usable. Memory requirements are then mild compared to the search workload, as there are $\frac{d(d-1)}{2}$ pairwise selections, although still enormous in comparison to all other proposals. A database of size $d = 2^{25}$ is mentioned to be realistic. A major threat for server impersonation is the following. An attacker might query the server and construct a personal database: $\langle \boldsymbol{\Delta_i}, \boldsymbol{C_{i1}}, \boldsymbol{P_{i1}}, \boldsymbol{C_{i2}}, \boldsymbol{P_{i2}} \rangle$. Authentication will succeed, although a session key can not be retrieved if present.

Predecessor Issues Although the protocol is in essence identical to a direct predecessor [1], it is not described in terms of modifications. Nevertheless, a few issues have (quietly) been resolved. We observe five modifications. First, *Rep* is acknowledged to require helper data. Second, one introduces the restriction $\boldsymbol{\Delta} \neq \boldsymbol{0}$. However, this event occurs so seldom that an explicit check could be regarded as overhead. Third, $\boldsymbol{\Delta}$ is generated by a TRNG instead of a PRNG. To avoid replay, the latter construction would require secure NVM, hereby undermining the advantages of PUFs. Fourth, the protocol is initiated by a token instead of the server. Unfortunately, this enables exhaustion of the server database as described before. Fifth, PUF responses are reinforced by a cryptographic hash function. In its absence, we see a direct opportunity for prover impersonation in the occasional case that $HW(\boldsymbol{\Delta}) < t$. Consider an arbitrary response $\boldsymbol{R} = Rep(PUF(\boldsymbol{C}), \boldsymbol{P})$. For both secure sketch constructions, an attacker might be able to produce a given $\boldsymbol{\Delta}$:

Code-offset construction: $\boldsymbol{R} \oplus \boldsymbol{\Delta} = Rep(PUF(\boldsymbol{C}), \boldsymbol{P} \oplus \boldsymbol{\Delta})$
Syndrome construction: $\boldsymbol{R} \oplus \boldsymbol{\Delta} = Rep(PUF(\boldsymbol{C}), \boldsymbol{P} \oplus H * \boldsymbol{\Delta})$

4 Overview and Discussion

Tables 1 and 2 provide an overview of Section 3. We do not support the use of any strong PUF proposal in its current form, given the numerous amount of issues. However, we do not object the use of both weak PUF methods: reference II and the modified reverse FE proposal [13]. We now discuss the strong PUF issues.

Two proposals rely on **secure reprogrammable NVM**: Öztürk et al. and logically reconfigurable PUFs. The need for reprogramming undermines a major advantage which PUFs might offer: low-cost manufacturing. The secrecy of the NVM state in the former proposal undermines a second advantage as well: an improved resistance against physical attacks. Furthermore, the hardware footprint of the latter proposal is not expected to improve with respect to reference I. Note that reference I requires one-time programmable NVM only: e.g. fuses can be used. Finally, updating the NVM state was identified as a denial-of-service attack for both proposals.

PUF responses are not perfectly reproducible. Protocols employ two approaches to overcome this issue: error correction and error tolerance. Unfortunately, two proposals struggle with the error tolerance approach. PUF non-determinism is greatly underestimated in Öztürk et al.: the protocol synchronization effort is presented too optimistic. We believe the proposal of Hammouri et al. to be non-functional because of internal error propagation.

In practice, **strong PUFs do have a small output space**. There are various methods to resolve this issue, all imposing a certain efficiency burden. However, protocols proposals have very little attention for this topic. Even though system security is not necessarily unaffected. We specified attacks for the

	Reference I	Reference II	Naïve	Controlled	Öztürk et al.	Hammouri et al.	Reconfiguration	Reverse FE	Slender	Converse
Weak PUF	×	✓	×	×	×	×	×	×	×	×
Strong PUF[1]	×	×	✓	✓	✓[2]	✓[2]	✓[3]	✓[4]	✓[5]	✓
NVM	✓	×	×	×	✓[6]	×	✓[6]	×	×	×
TRNG	×	×	×	×	×	✓	×	×	✓	✓
Gen	×	×	×	×	×	×	×	✓	×	×
Rep	×	✓	×	✓	×	×	×	×	×	✓
Hash	✓	✓	×	✓	×	×	✓	✓	×	✓
1× interface	✓	✓	×	✓	✓	✓	×	✓	✓	✓
Other	×	×	×	×	✓	✓	✓	×	✓	✓
Token authenticity	✓	✓	✓	✓	✓	✓	✓	✓	✓	×
Server authenticity	×	×	×	×	×	×	×	✓	×	✓
# Authentications	∞	∞	d	d	∞	∞	d	∞	∞	∞
CRPs	×	×	✓	✓	×	×	✓	✓	×	✓
Model	×	×	×	×	✓	✓	×	×	✓	×
Key	✓	✓	×	×	×	×	×	×	×	×

[1] Including logic to expand the response space.
[2] Easy-to-model.
[3] Robust against modeling.
[4] Non-determinism lower bound.
[5] Both easy-to-model and hard-to-model
[6] Reprogrammable.

Table 1. For all protocols: token hardware (top segment), the authenticity provided (middle segment) and the secret stored by the server (bottom segment).

Protocol	**Issues**
Controlled	- Inferior to reference II.
Öztürk et al.	- NVM undermines the advantages of PUFs. - Synchronization effort is presented too optimistic. - Denial-of-service attack. - *Choice of permutation requires care.*
Hammouri et al.	- Bizarre inner PUF, prone to modeling deficiencies. - Non-functional: internal error propagation. - Bizarre modeling procedure: contradictive and overcomplicated.
Reconfiguration	- Exacting PUF requirement: secure against modeling. - NVM undermines the advantages of PUFs. - Denial-of-service attack.
Reverse FE	- Exacting PUF requirements to counteract replay attacks. - *PRNG Exploitation.* - No need for strong PUF.
Slender PUF	- *PRNG Exploitation.* - Exacting PUF requirements to counteract statistical attacks.
Converse	- Attacker model too restricted. - Scalability issues, leading to server impersonation. - Predecessor Issues.

Table 2. Issues revealed in this work. Implementation dependant issues are printed in *italic.*

proof-of-concept implementations of the reverse FE and slender PUF proposal, both exploiting the challenge expansion PRNG.

Strong PUFs are insecure against modeling, in practice. Two proposals are therefore too demanding: logically reconfigurable PUFs and the slender PUF protocol. Although the latter offers some countermeasures, there is an opposing requirement for the PUF to be easy-to-model, leading to a delicate balancing exercise. In general, proposals not reinforced by a cryptographic hash function are much more likely to be vulnerable.

Several proposal rely on a **secure TRNG**. Tampering with its randomness opens new perspectives for a physical attack. Its use seems to be unavoidable however if server authentication is a must, as assumed for the converse protocol, to avoid replay attacks. The reverse FE proposal extracts its non-determinism from the PUF instead, which has been identified as a delicate balancing exercise. Two protocols without server authentication employ their TRNG as a modeling countermeasure: Hammouri et al. and slender PUF protocol.

All proposals aim to provide lightweight entity authentication, which addresses a highly relevant need. However, in many use cases, there will be accompanying security needs: message confidentiality and integrity, privacy, etc. We did not consider **protocol extensibility** in this work, although it might be of interest when designing a new protocol. References I and II, which employ a secret key, might benefit from a huge amount of scientific literature. Like-minded, the establishment of a session key has been proposed as an extension for the converse protocol.

5 Conclusion

Various protocols utilize a strong PUF to provide lightweight entity authentication. We described all such proposals using a unified notation, hereby creating a first overview and initializing direct comparison as well. We defined two reference authentication methods, to identify the misuse of PUFs. Our analysis revealed numerous security and practicality issues. Therefore, we do not recommend the use of any strong PUF proposal in its current form. More fundamental research is required, aiming to create a truly strong PUF with great cryptographic properties. If not, secure lightweight authentication might become a mission impossible. The observations and lessons learned in this work will facilitate future protocol design.

Acknowledgment

The authors greatly appreciate the support received. The European Commission through the ICT programme under contract FP7-ICT-2011-317930 HINT. The Research Council of KU Leuven: GOA TENSE (GOA/11/007), the Flemish Government through FWO G.0550.12N and the Hercules Foundation AKUL/11/19. The national major development program for fundamental research of China (973 Plan) under grant no. 2013CB338004. Jeroen Delvaux is funded by IWT-Flanders grant no. 121552. The authors would like to thank Anthony Van Herrewege (KU Leuven, ESAT/COSIC) for his valuable comments.

References

1. Das, A., Kocabaş, Ü., Sadeghi, A.-R., Verbauwhede, I.: *PUF-based secure test wrapper design for cryptographic SoC testing.* In: Design, Automation & Test in Europe (DATE), IEEE, pp. 866–869, Mar. 2012.
2. Delvaux, J., Verbauwhede, I.: *Attacking PUF-Based Pattern Matching Key Generators via Helper Data Manipulation* In: CT-RSA 2014.
3. Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.: *Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data.* SIAM J. Comput., vol. 38, no. 1, pp. 97–139, Mar. 2008.
4. Gassend, B., Clarke, D.E., van Dijk, M., Devadas, S.: *Silicon physical random functions.* In: ACM Conference on Computer and Communications Security, pp. 148–160, Nov. 2002.
5. Gassend, B., Clarke, D.E., van Dijk, M., Devadas, S.: *Controlled Physical Random Functions.* In: Annual Computer Security Applications Conference (ACSAC), pp. 149–160, , Dec. 2002.
6. Gassend, B., van Dijk, M., Clarke, D.E., Torlak, E., Devadas, S., Tuyls, P.: *Controlled physical random functions and applications.* In: ACM Trans. Inf. Syst. Secur., vol. 10, no. 4, 2008.
7. Hammouri, G., Öztürk, E., Sunar B.: *A tamper-proof and lightweight authentication scheme.* In: Journal Pervasive and Mobile Computing, vol. 6, no. 4, 2008.
8. Holcomb, D.E., Burleson, W.P., Fu, K.: *Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers.* In: IEEE Trans. Computers, vol. 58, no 9, 2009.
9. Hospodar, G., Maes, R., Verbauwhede, I.: *Machine Learning Attacks on 65nm Arbiter PUFs: Accurate Modeling poses strict Bounds on Usability.* In: Workshop on Information Forensics and Security (WIFS), IEEE 2012, pp. 37–42, Dec. 2012.
10. Katzenbeisser, S., Kocabaş, Ü., van der Leest, V., Sadeghi, A.-R., Schrijen G.-J., Schröder, H., Wachsmann, C.: *Recyclable PUFs: Logically Reconfigurable PUFs.* In: Cryptographic Hardware and Embedded Systems (CHES), pp. 374–389, Sep 2011.
11. Kocabaş, Ü., Peter, A., Katzenbeisser, S., Sadeghi, A.-R.: *Converse PUF-Based Authentication.* In: Trust and Trustworthy Computing (TRUST), LNCS, vol. 7344, pp. 142–158, Jun. 2012.
12. Lee, J.W., Lim, D., Gassend, B., Suh, G.E., van Dijk, M., Devadas, S.: *A technique to build a secret key in integrated circuits for identification and authentication applications.* In: VLSI Circuits, 2004 Symposium on, pp. 176–179, Jun. 2004
13. Maes, R.: *Physically Unclonable Functions: Constructions, Properies and Applications.* PhD Thesis, KU Leuven, Chapter 5, 2012.
14. Majzoobi, M., Rostami, M., Koushanfar, F., Wallach, D.S., Devadas, S.: *Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching.* In: IEEE Symposium on Security and Privacy (SP), pp. 33–44, May 2012.

15. Öztürk, E., Hammouri, G., Sunar B.: *Towards Robust Low Cost Authentication for Pervasive Devices.* In: IEEE Conference on Pervasive Computing and Communications (PerCom), Mar. 2008.
16. Pappu, R.: *Physical One-Way Functions.* PhD Thesis, MIT, Chapter 9, 2001.
17. Paral, Z., Devadas, S.: *Reliable and Efficient PUF-Based Key Generation Using Pattern Matching.* In: Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on, pp. 128–133, Jun. 2011.
18. Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., Schmidhuber, J.: *Modeling attacks on physical unclonable functions.* In: Computer and Communications Security (CCS), 2010 ACM conference on, pp. 237–249, Oct. 2010.
19. Skorobogatov, S.: *Semi-invasive attacks - a new approach to hardware security analysis.* Technical Report UCAM-CL-TR-630, University of Cambridge, Computer Laboratory, Apr. 2005.
20. Suh, G.E., Devadas, S.: *Physical unclonable functions for device authentication and secret key generation.* In: Design Automation Conference (DAC), pp. 9–14, Jun. 2007.
21. Van Herrewege, A., Katzenbeisser, S., Maes, R., Peeters, R., Sadeghi, A.-R., Verbauwhede, I., Wachsmann, C.: *Reverse Fuzzy Extractors: Enabling Lightweight Mutual Authentication for PUF-Enabled RFIDs.* In: Financial Cryptography and Data Security (FC), LNCS, vol. 7397, pp. 374–389, Feb. 2012.

A Arbiter PUF

A.1 Architecture

Arbiter PUFs [12] quantify manufacturing variability via the propagation delay of logic gates and interconnect. The high-level functionality is represented by Figure 9(a). A rising edge propagates through two paths with identically designed delays, as imposed by layout symmetry. Because of nanoscale manufacturing variations however, there is a delay difference Δt between both paths. An arbiter decides which path 'wins' the race ($\Delta t \lessgtr 0$) and generates a response bit r.

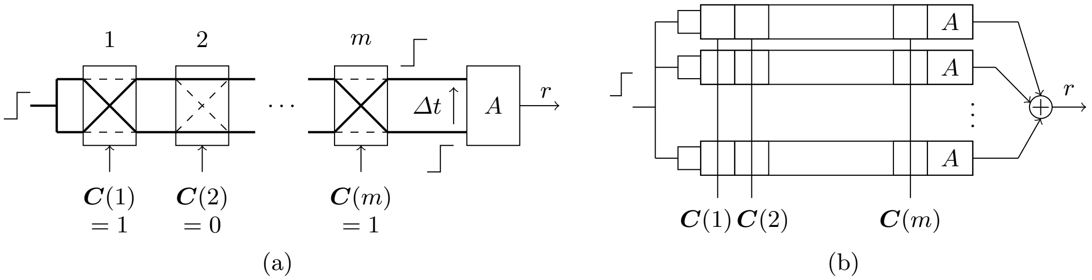


Fig. 9. Arbiter PUF (a) and its XOR variant (b).

The two paths are constructed from a series of m switching elements. The latter are typically implemented with a pair of 2-to-1 multiplexers. Challenge bits determine for each stage whether path segments are crossed or uncrossed. Each stage has a unique contribution to Δt, depending on its challenge bit. Challenge vector $\boldsymbol{C}$ determines the time difference Δt and hence the response bit r. The number of CRPs equals 2^m. The response reproducibility differs per CRP: the smaller $|\Delta t|$, the easier to flip side because of various physical perturbations.

A.2 Vulnerability to Modeling Attacks

Arbiter PUFs show additive linear behavior, which makes them vulnerable to modeling attacks. A decomposition in individual delay elements is given in Figure 10. Both intra- and inter- switch delays contribute to Δt, as represented by white and gray/black squares respectively. We incorporate the latter in their preceding switches, without loss of generality, to facilitate the derivation of a delay model.

In the end, delay elements are important as far as they generate delay differences between both paths. Therefore, each stage can be described by two delay parameters only: one for each challenge bit state, as illustrated in Figure 11. The delay difference at the input of stage i flips in sign for the crossed configuration and is incremented with δt_i^1 or δt_i^0 for crossed and uncrossed configurations respectively.

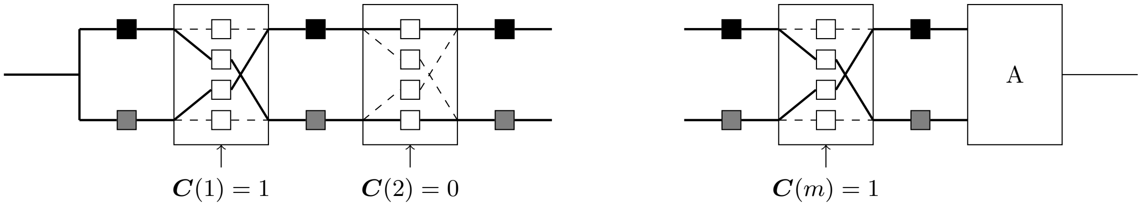


Fig. 10. Arbiter PUF decomposed in individual delay elements, represented by small squares and prone to manufacturing variability. The interconnecting lines have zero delay.

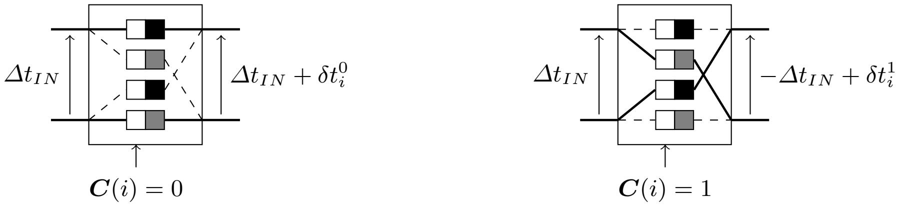


Fig. 11. Arbiter stage

The impact of a δt on Δt is incremental or decremental for an even and odd number of subsequent crossed stages respectively. By lumping together the δt's of neighboring stages, one can model the whole arbiter PUF with $m+1$ independent parameters only (and not $2m$). A formal expression for Δt is as follows [18]:

$$\Delta t = \boldsymbol{\gamma}\boldsymbol{\tau}^T \text{ with } \boldsymbol{\tau} = \frac{1}{2}\begin{pmatrix} \delta t_0 & \delta t_1^0 - \delta t_1^1 \\ \delta t_1^0 + \delta t_1^1 & + \ \delta t_2^0 - \delta t_2^1 \\ \vdots & \\ \delta t_{m-1}^0 + \delta t_{m-1}^1 & + \ \delta t_m^0 - \delta t_m^1 \\ \delta t_m^0 + \delta t_m^1 & \end{pmatrix}^T \text{ and } \boldsymbol{\gamma} = \begin{pmatrix} 1 - 2(\boldsymbol{C}(1) \oplus \boldsymbol{C}(2) \oplus \ldots \oplus \boldsymbol{C}(m)) \\ 1 - 2(\boldsymbol{C}(2) \oplus \ldots \oplus \boldsymbol{C}(m)) \\ \vdots \\ 1 - 2\boldsymbol{C}(m) \\ 1 \end{pmatrix}^T .$$

Vector $\boldsymbol{\gamma} \in \{\pm 1\}^{1\times(m+1)}$ is a transformation of challenge vector $\boldsymbol{C}$. Vector $\boldsymbol{\tau} \in \mathbb{R}^{(q+1)\times 1}$ contains the lumped stage delays. The more linear a system, the easier to learn its behavior. [9]. By using $\boldsymbol{\gamma}$ instead of $\boldsymbol{C}$ as ML input, a great deal of non-linearity is avoided. The non-linear threshold operation $\Delta t \lessgtr 0$ remains however. Only 5000 CRPs were demonstrated to be sufficient to model non-simulated 64-stage arbiter PUFs with an accuracy of about 97% [9].

A.3 XOR Variant

Several variants of the arbiter PUF increase the resistance against ML. They introduce various forms of non-linearity for this purpose. We only consider the XOR variant. The response bits of multiple arbiter chains are XORed to a single response bit, as shown in Figure 9(b). All chains have the same challenge as input. The more chains, the more resistance against against ML: the required number of CRPs and the computation time both increase rapidly [18]. However, the reproducibility of r decreases with the number of chains as well: each additional chain injects non-determinism into the overall system. A practical limit on the ML resistance is hence imposed.

A.4 CRP Correlation Function: Enabling Statistical Attacks

Machine learning attacks exploit CRP correlations in a implicit manner. Statistical attacks benefit from their explicit exploitation, hereby assuming the knowledge a function $P(r_1 = r_2) = f(\boldsymbol{C_1}, \boldsymbol{C_2})$. Such a function has already been determined for the arbiter PUF and its XOR variant in [14] via simulations. However, we believe the simulation results for the XOR variant to be erroneous. Therefore, we repeat

PUF	$P(r_1 = r_2)$
Arbiter	$f = P(\lvert t_{SAME}\rvert > \lvert t_{DIFF}\rvert)$
	with $t_{SAME} \sim \mathcal{N}(0, 2\sqrt{m-h+1}\sigma_1)$
	and $t_{DIFF} \sim \mathcal{N}(0, 2\sqrt{h}\sigma_1)$
	$= 1 - 4\int_0^\infty \phi(t, 2\sqrt{m-h+1}\sigma_1)\varPhi(-t, 2\sqrt{h}\sigma_1)dt$
	$= 1 - 4\int_0^\infty \phi(t, \sqrt{m-h+1})\varPhi(-t, \sqrt{h})dt$
2-XOR arbiter	$f^2 + (1-f)^2$
3-XOR arbiter	$f^3 + 3f(1-f)^2$
4-XOR arbiter	$f^4 + 6f^2(1-f)^2 + (1-f)^4$

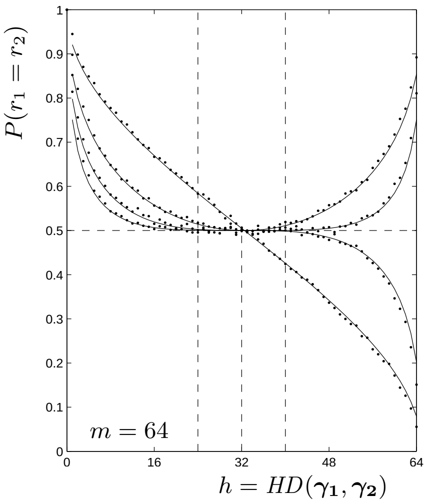


Fig. 12. Correlations for the arbiter PUF and its XOR variant. Dots represent simulations results. The mathematical model, drawn continuously although of discrete nature, matches reasonably well. Vertical dashed lines enclose 99% of the data for randomly chosen challenges. The more chains being XORed, the better one approximates the ideal behavior $f = 1/2$, but the larger the response non-determinism.

the simulation experiment and we derive an analytical model as well, as shown in Figure 12. Let $\mathcal{N}(\mu, \sigma)$ denote a normal distribution with mean μ and standard deviation σ. Let $\phi(x, \sigma)$ and $\varPhi(x, \sigma)$ denote the probability density function and cumulative distribution function respectively, assuming $\mu = 0$. We assume all δt's to have a distribution $\mathcal{N}(0, \sigma_1)$, which is a common and well-validated practice in previous work. Although not fully correct[9], we then assume the elements of $\boldsymbol{\tau}$ to have a distribution $\mathcal{N}(0, 2\sigma_1)$. We introduce the variable $h = HD(\boldsymbol{\gamma_1}, \boldsymbol{\gamma_2})$.

B Hammouri et al.

Hammouri et al. [7] employ two strong PUFs, as shown in Figure 1(f). Both PUFs are modeled during enrollment. The outer PUF is an arbiter PUF. The inner PUF is a custom architecture based on the arbiter PUF. *Lin* compensates the non-linearity of the outer arbiter PUF, apart from the final thresholding step: $\varDelta t$ is a linear function of $Lin(\boldsymbol{X})$. Its form is relatively simple, as shown below. Figure 13 represents the authentication protocol.

	Token	$\longleftarrow$	**Server**	
Enrollment (1×)		$\longleftrightarrow$	Train models $\widetilde{PUF_I}$ and $\widetilde{PUF_O}$	
Authentication (∞)	$\boldsymbol{C_I} \xleftarrow{U} TRNG()$	$\xrightarrow{\boldsymbol{C_I}}$		
		$\xleftarrow{\boldsymbol{C_O}}$	$\boldsymbol{C_O} \xleftarrow{U} TRNG()$	
	$\widetilde{\boldsymbol{X}} \leftarrow PUF_I(\boldsymbol{C_I}, PRNG())$			$\forall i \in [1\ n]$
	$\widetilde{\boldsymbol{R}}(i) \leftarrow PUF_O(Lin(\widetilde{\boldsymbol{X}}) \oplus \boldsymbol{C_O})$	$\xrightarrow{\widetilde{\boldsymbol{R}}(i)}$		
			$\widetilde{\boldsymbol{X'}} \leftarrow \widetilde{PUF_I}(\boldsymbol{C_I}, PRNG())$	
			$\widetilde{\boldsymbol{R'}}(i) \leftarrow \widetilde{PUF_O}(Lin(\widetilde{\boldsymbol{X'}}) \oplus \boldsymbol{C_O})$	
			Abort if $HD(\widetilde{\boldsymbol{R}}, \widetilde{\boldsymbol{R'}}) > \epsilon$	

Fig. 13. Authentication protocol of Hammouri et al.

---

[9] We neglect dependencies within $\boldsymbol{\tau}$ and we also ignore the different form of $\boldsymbol{\tau}(1)$ and $\boldsymbol{\tau}(m+1)$

$$Lin(\boldsymbol{X}) = \begin{pmatrix} \boldsymbol{X}(1) \oplus \boldsymbol{X}(2) \\ \boldsymbol{X}(2) \oplus \boldsymbol{X}(3) \\ \vdots \\ \boldsymbol{X}(m-1) \oplus \boldsymbol{X}(m) \\ \boldsymbol{X}(m) \end{pmatrix}^T .$$

B.1 Inner PUF: Bizarre Architecture, Prone to Modeling Deficiencies

Architecture The inner PUF is a custom architecture based on the arbiter PUF. One proposes a rather bizarre extension of the challenge space. Out of two individual chains, one aims to construct a single reconfigurable chain. For each stage, one out of two switching elements is selected, hereby introducing a second challenge $\boldsymbol{S}$. The proposal ignores the need to describe the reconfiguration logic: Figure 14 (right) provides a generic schematic, including both intra- and inter-switch delays. Via reconfiguration, one aims to provide a large response space. One does evaluate $\boldsymbol{C_I}$ for a fixed list of configurations vectors $\{\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots\}$, hereby concatenating the response bits. The configuration vectors are generated by a PRNG, having $\boldsymbol{S_1}$ as initial state. Note that one could have provided a large response space with a regular arbiter PUF as well, given the use of a PRNG.

$$\boldsymbol{\tau} = \tfrac{1}{2} \left(\begin{array}{ccc} \delta t_0 & & \cancel{\delta t_1^0 - \delta t_1^1} \\ \delta t_1^0 + \delta t_1^1 & + & \cancel{\delta t_2^0 - \delta t_2^1} \\ & \vdots & \\ \delta t_{m-1}^0 + \delta t_{m-1}^1 & + & \cancel{\delta t_m^0 - \delta t_m^1} \\ \delta t_m^0 + \delta t_m^1 & & \end{array} \right)^T$$

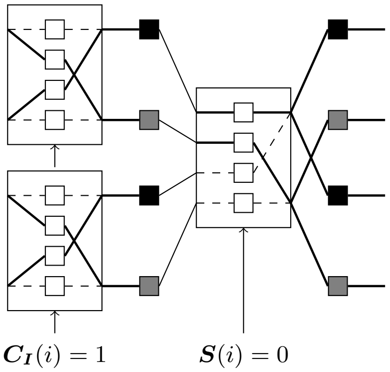


Fig. 14. Reconfigurable stage of the inner PUF

Prone to Modeling Deficiencies The architecture is required to be easy-to-model: the server has to construct a model during enrollment. The overall structure is additive, as for a regular arbiter PUF, and therefore we expect modeling to be feasible. Although one could have derived a generic (but complicated) delay model, possibly leading to an efficient modeling method, the authors do propose a shortcut. The overall delay model is supposed to be separable in terms of the individual chains. One does construct a model $\boldsymbol{\tau_U}$ for the upper chain, applying $\boldsymbol{S} = \boldsymbol{0}$. Similar, one obtains a model $\boldsymbol{\tau_L}$ for the lower chain, applying $\boldsymbol{S} = \overline{\boldsymbol{0}}$. The overall model $\boldsymbol{\tau}(\boldsymbol{S})$ selects elements from both vectors: $\boldsymbol{\tau}(i) = \boldsymbol{S}(i)\boldsymbol{\tau_L}(i) + \overline{\boldsymbol{S}(i)}\boldsymbol{\tau_U}(i)$. The variability of intra-stage delays is being neglected, justified by placing the stages far apart in the circuit lay-out. An approximating delay model for upper/lower chain is derived as well, as shown in Figure 14 (left).

Although it might be possible to make all the former workable (there is no proof-of-concept implementation), the approach is strongly lay-out dependent and prone to modeling deficiencies. Apart from being area consuming: positioning stages far apart might not be sufficient to justify separability. Intra- and inter-stage variations originate from CMOS transistors and metal interconnect respectively. We highly doubt that the former would be per se negligible with respect to the latter. It is possible though to enhance separability: upsizing transistor of the switches, inserting minimum-sized inverter chains in between switches, etc. Furthermore, one should distinguish the metal interconnect before and after the selection logic, as shown in Figure 14 (right). Variability of the latter would undermine the separability.

We stress that all these complications could have been avoided easily, e.g. by using a regular arbiter PUF.

B.2 Bizarre Modeling Procedure: Contradictive and Overcomplicated

During enrollment, the server constructs a model for both the inner and outer PUF. Reading out $\boldsymbol{X}$, the output of the inner PUF, via a one-time interface would have made things easy. This would allow to model both PUFs separately. However, one designed a rather complicated procedure to model the inner PUF via the output of the outer PUF. Therefore, one introduces a function *Lin* to linearize the outer PUF. We note that this is rather contradictive: it degrades the overall modeling resistance. Fixing $\boldsymbol{C_O} = \mathbf{0}$, one obtains a system $r \leftarrow PUF_O(Lin(\boldsymbol{X}))$. Error propagation from $\boldsymbol{X}$ to r is very limited then. An error in bit $\boldsymbol{X}(i)$ would flip the sign of $\boldsymbol{\gamma}(i)$ only, corresponding to $h = 1$ in Figure 12. During enrollment, one can force the PRNG to maintain either $\mathbf{0}$ or $\overline{\mathbf{0}}$ as its state, allowing to model the upper and lower chain separately. This requires some sort of destructive interface, similar to our proposal to read out $\boldsymbol{X}$ directly. Depending on $\boldsymbol{C_I}$, $\boldsymbol{X}$ will be either $\mathbf{0}$ or $\overline{\mathbf{0}}$, apart from potential noisiness, hereby maximizing h. As a consequence, one can distinguish either case, as r is very likely to flip. This enables modeling of the inner PUF. Although some samples might be problematic: the pair $\boldsymbol{X} = \mathbf{0}/\overline{\mathbf{0}}$ occasionally result in $\Delta t \approx 0$, maintaining r in a noisy state.

B.3 Non-functional: Error Propagation

We believe the proposal to be non-functional because of internal error propagation. The minimization of h does not hold for the system $r \leftarrow \widetilde{PUF_O}(Lin(\widetilde{\boldsymbol{X'}}) \oplus \boldsymbol{C_O})$, in the general case that $\boldsymbol{C_O} \neq \mathbf{0}$. A single error in $\boldsymbol{X}$ will flip the sign of r with a probability close to 1/2. This could have been avoided by implementing the system $r \leftarrow PUF_O(Lin(\boldsymbol{X} \oplus \boldsymbol{C_O}))$ instead.