Recyclable PUFs: Logically Reconfigurable PUFs⋆

Stefan Katzenbeisser[1], Ünal Kocabaş[1], Vincent van der Leest[2], Ahmad-Reza Sadeghi[3], Geert-Jan Schrijen[2], Heike Schröder[1], and Christian Wachsmann[1]

[1] Technische Universität Darmstadt (CASED), Germany
{katzenbeisser,busch}@seceng.informatik.tu-darmstadt.de
{unal.kocabas,christian.wachsmann}@trust.cased.de

[2] Intrinsic-ID, Eindhoven, The Netherlands
{vincent.van.der.leest,geert.jan.schrijen}@intrinsic-id.com

[3] Technische Universität Darmstadt and Fraunhofer SIT Darmstadt, Germany
ahmad.sadeghi@trust.cased.de

Abstract. Physically Unclonable Functions (PUFs) are security primitives that exploit intrinsic random physical variations of hardware components. In the recent years, many security solutions based on PUFs have been proposed, including identification/authentication schemes, key storage and hardware-entangled cryptography. Existing PUF instantiations typically exhibit a static challenge/response behavior, while many practical applications would benefit from reconfigurable PUFs. Examples include the revocation or update of "secrets" in PUF-based key storage or cryptographic primitives based on PUFs.
In this paper, we present the concept of Logically Reconfigurable PUFs (LR-PUFs) that allow changing the challenge/response behavior without physically replacing or modifying the underlying PUF. We present two efficient LR-PUF constructions and evaluate their performance and security. In this context, we introduce a formal security model for LR-PUFs. Finally, we discuss several practical applications of LR-PUFs focusing on lightweight solutions for resource-constrained embedded devices, in particular RFIDs.

1 Introduction

The rapid evolution of information technology has drastically changed the shape of information systems. Computing devices are becoming increasingly smaller and highly distributed. Moreover, embedded computing platforms such as smartphones and sensors undergo a rapid development becoming progressively more sophisticated with regard to their computational, storage and interface capabilities. At the same time, the growing popularity and widespread use of these systems, along with the fact that they are increasingly deployed to process and store privacy-sensitive and security-critical data, makes them attractive targets for different kinds of software and hardware attacks. In particular, resource-constrained embedded devices, like RFIDs, are typically vulnerable to physical attacks.

In this context, Physically Unclonable Functions (PUFs), promise a cost-efficient alternative to existing hardware protection mechanisms. PUFs enable unique device identification and authentication (see, e.g., [46,56,44,48]), binding software and hardware components to devices (see, e.g., [17,24,16,10]), secure storage of, e.g., cryptographic secrets (see, e.g., [59,33]), and can be directly integrated into cryptographic algorithms [3] and remote attestation protocols [50]. Today, there are already some PUF-based security products aimed for the market, including RFID, IP-protection and anti-counterfeiting solutions [58,20].

Existing PUF implementations typically exhibit a static challenge/response behavior, while many practical applications would benefit from reconfigurable PUFs, whose challenge/response

⋆ A preliminary version of this report that has been presented at CHES'11 [22], and a more condensed version has been published in the Journal of Cryptographic Engineering [23]. This version additionally provides detailed security proofs.

behavior can be dynamically changed, i.e., *reconfigured*, after deployment. For instance, applications of PUF-based key storage [59,33] and PUF-based cryptographic primitives [3] may require revoking or updating previous secrets derived from the PUF, which could be easily realized with a reconfigurable PUF. Another example are solutions to prevent downgrading of software by binding the software to a certain hardware configuration [26,10]. Moreover, when PUF-based wireless access tokens (see, e.g., [46,56,44,48,58]) are re-used/recycled, the new users of the token shall not be able to retrieve access rights and/or to obtain privacy-sensitive information of the previous users of the token (see, e.g., [60,21,4]).

However, all known implementations of physically reconfigurable PUFs rely on optical mechanisms, reconfigurable hardware (i.e., FPGAs), or novel memory technologies [26], which all have serious drawbacks in practice. In particular, optical PUFs cannot be easily integrated into standard integrated circuits and often require expensive and error-prone evaluation equipment, while FPGA-based solutions cannot be realized with non-reconfigurable hardware (i.e., ASICs) that is commonly used in practice [36]. In this context, several attempts to emulate physically reconfigurable PUFs have been made. One of the first proposals in this direction was integrating a floating gate transistor into the delay lines of an arbiter PUF, which allows physically changing the challenge/response behavior of the PUF based on a logical state maintained in non-volatile memory [32]. Other approaches restrict access to the interface of the PUF and use part of the PUF challenge to emulate reconfigurability [33,26], which, however, works only for PUFs with a large challenge space.

Our goal and contributions. In this paper, we present the concept of *Logically Reconfigurable PUFs* (LR-PUFs), a practical alternative to physically reconfigurable PUFs. LR-PUFs amend a PUF with a stateful control logic that changes the challenge/response behavior of the LR-PUF according to its internal logical state without physically replacing or modifying the underlying PUF.[4]

We present and evaluate two different constructions for LR-PUFs. The results of our performance measurements show that the implementation overhead of the logical reconfiguration on top of a physical PUF is rather small. Further, we introduce a formal security model for LR-PUFs and prove that our constructions are secure. More precisely, we show that, when instantiated by an appropriate PUF under reasonable assumptions, our LR-PUFs can achieve both *forward-* and *backward-unpredictability*: The former assures that responses measured before the reconfiguration event are invalid thereafter, while the latter assures that an adversary with access to a reconfigured PUF cannot estimate the PUF behavior before reconfiguration. Finally, we demonstrate how LR-PUFs could be deployed for re-usable (recyclable) access tokens, such as electronic transit tickets, and discuss other envisaged applications of LR-PUFs. Note that, although the constructions of LR-PUFs as proposed in this paper seem to be similar to Controlled PUFs [13], they have very different objectives: In contrast to Controlled PUFs, LR-PUFs do not aim to prevent modeling attacks on PUFs but provide a practical way to *enable reconfigurability* for existing, typically static PUF constructions. We will elaborate on this aspect in Section 3.

Outline. The rest of the paper is structured as follows: After providing background information on Physically Unclonable Functions (PUFs) in Section 2, we present the concept of Logically Reconfigurable PUFs (LR-PUFs) in Section 3. We show two concrete LR-PUF constructions in Section 4, describe their implementation and evaluate their performance in Section 5, and formally prove their security in Section 6. In Section 7, we show how LR-PUFs could be used to realize recyclable access tokens and discuss several other potential use cases of LR-PUFs. Finally, we conclude in Section 8.

[4] A similar concept has been independently proposed by Lao et al. [28]. However, they do not provide a (formal) security model and do not discuss the assumptions underlying their constructions.

2 Background: Physically Unclonable Functions (PUFs)

A Physically Unclonable Function (PUF) is a noisy function that is embedded into a physical object, e.g., an integrated circuit [45,2]. When queried with a *challenge* w, a PUF generates a *response* $y \leftarrow \texttt{PUF}(w)$ that depends on both w and the unique device-specific intrinsic physical properties of the object containing $\texttt{PUF}()$. Since PUFs are subject to noise (e.g., environmental variations), they return slightly different responses when queried with the same challenge multiple times.

In literature, PUFs are typically assumed to be *robust*, *physically unclonable*, *unpredictable* and *tamper-evident*, and several approaches to heuristically quantify and formally define their properties have been proposed (see [2] for a comprehensive overview). Robustness means that, when queried with the same challenge multiple times, the same PUF will always return the same response. Physical unclonability means that it is infeasible to physically produce two PUFs that cannot be distinguished based on their challenge/response behavior, which cannot be achieved by (cryptographic) algorithms. Unpredictability requires that it is infeasible to predict the PUF response to a given, previously unknown challenge, even if the PUF can be adaptively queried for a certain number of times. Since this is the most interesting property for cryptographic applications of PUFs (see, e.g., [3,36,2]), we will formally define unpredictability later, when we prove the security of our LR-PUF constructions. Tamper-evidence means that any attempt to physically access the PUF irreversibly changes its challenge/response behavior. This is an important issue for practical deployment since it allows the detection of invasive hardware attacks, to which embedded devices are typically exposed to in practice.

A broad variety of different PUF constructions exists (see [36] for an overview). The most appealing ones for integration into electronic circuits are electronic PUFs. The most prominent examples of electrical PUFs include *delay-based PUFs* that exploit race conditions (arbiter PUFs [29,44,34]) and frequency variations (ring oscillator PUFs [14,55,37]) that can be found in integrated circuits; *memory-based PUFs* that are based on the instability of volatile memory cells like SRAM [18,19], flip-flops [35,30] and latches [54,25]; and *coating PUFs* [57], which are based on the capacitance caused by a special dielectric coating applied to the chip that houses the PUF.

Note that the number of unique responses of a memory-based PUF is limited by the number of its memory cells. Moreover, it has been shown that most delay-based PUFs are subject to model building attacks that allow simulating the PUF in software (see, e.g., [29,44,34,47]). To counter this problem, additional primitives must be used: Controlled PUFs [13] use cryptography in hardware to hide the actual response of the underlying PUF, which prevents model building attacks. This requires the link between the PUF and the crypto component as well as the crypto component itself to be protected against invasive and side channel attacks.

3 Logically Reconfigurable PUFs

A logically reconfigurable PUF (LR-PUF) is a PUF whose challenge/response behavior depends on both the physical properties of the PUF and the logical state maintained by a control logic. The challenge/response behavior of the LR-PUF can be dynamically changed after it has been deployed by updating its state.

3.1 System Model

An LR-PUF combines a conventional physically unclonable function and a control logic circuit. As shown in Figure 1, the control logic maintains a state S, which is stored in non-volatile

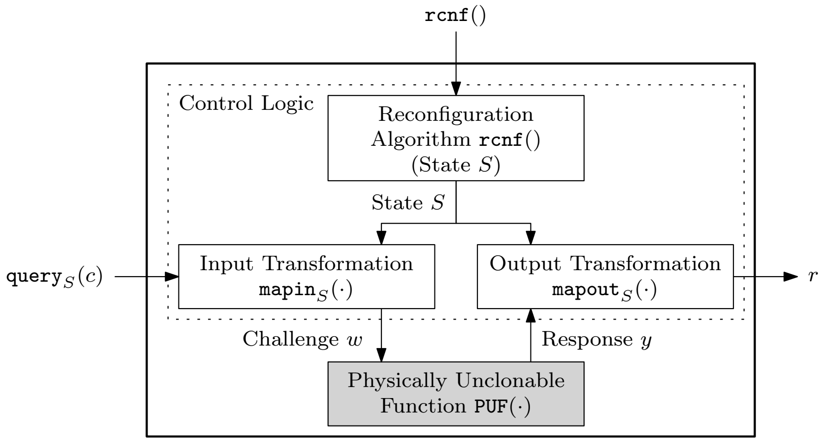

Fig. 1. Generic Logically Reconfigurable PUF construction

memory, and provides an algorithm $\texttt{query}_S()$ for querying, and $\texttt{rcnf}()$ for reconfiguring the LR-PUF. The algorithm $\texttt{query}_S()$ consists of an input transformation function $\texttt{mapin}_S()$ and an output transformation function $\texttt{mapout}_S()$: $\texttt{query}_S(c)$ computes $w \leftarrow \texttt{mapin}_S(c)$, evaluates $y \leftarrow \texttt{PUF}(w)$, and returns $r \leftarrow \texttt{mapout}_S(y)$. The algorithm implementing $\texttt{rcnf}()$ reconfigures the LR-PUF by changing the current state S to a new independent state $S' \leftarrow \texttt{rcnf}()$.

Note that the generic LR-PUF construction depicted in Figure 1 can be seen as a generalization of controlled PUFs [13]. Controlled PUFs aim to *hide* the challenge/response behavior of the underlying PUF to the adversary to prevent model building attacks [13,47] by applying an appropriate $\texttt{mapin}()$ and/or $\texttt{mapout}()$ function. In contrast, LR-PUFs aim to *enable reconfigurability* for conventional non-reconfigurable PUFs after they have been deployed by entangling an updatable state with the challenges and/or responses of the underlying PUF.

3.2 Assumptions and Adversary Model

Following the common assumptions on PUFs, we assume that the underlying PUF of the LR-PUF is physically unclonable and unpredictable (see Section 2). Further, the algorithms $\texttt{mapin}()$, $\texttt{mapout}()$ and $\texttt{rcnf}()$ are publicly known. Moreover, the adversary $\mathcal{A}$ is assumed to *know* the current and all previous states S of the LR-PUF but *cannot change* S to a value of its choice (e.g., an old LR-PUF state).

3.3 Requirements

To prevent the adversary $\mathcal{A}$ from changing the LR-PUF state to a specific value, the following must be ensured: (1) $\texttt{rcnf}()$ cannot be manipulated such that it generates predictable states, and (2) the non-volatile memory cells storing the LR-PUF state cannot be set to specific values (e.g., by hardware attacks). The first requirement can be achieved by implementing the reconfiguration function using a fault injection aware design at a reasonable performance penalty [38,1]. Moreover, although fault injection attacks against non-volatile memory (e.g., EEPROM or Flash) are known [51], it seems to be difficult in practice to perform invasive attacks that change the content of specific non-volatile memory cells without affecting the content of the surrounding cells [52]. Hence, in practice it should be infeasible for an adversary to write a specific value (e.g., an old LR-PUF state) into the non-volatile memory of the LR-PUF. In particular, due to the increasing complexity of modern embedded systems and the fact that technology nodes are progressively getting smaller, the amount of precision and the quality of the equipment required to successfully perform such attacks renders them uneconomical in most practical applications (e.g., electronic ticketing).

3.4 Security Objectives

Like a conventional PUF, an LR-PUF should be robust, physically unclonable and unpredictable (see Section 2). In particular, it should be infeasible for the adversary $\mathcal{A}$ to predict the response to a challenge of an LR-PUF for a certain state, even if $\mathcal{A}$ knows the responses to this challenge for *other* (e.g., old) states. Moreover, in most applications of reconfigurable PUFs, it must be infeasible to set the state of the LR-PUF to a specific value, which would allow resetting the LR-PUF to a previous state and may help the adversary to predict LR-PUF responses.

We first informally summarize the security requirements of LR-PUFs below and later give formal definitions in Section 6.

- *Backward unpredictability:* $\mathcal{A}$ cannot predict the response of the LR-PUF for a *previous* state S (i.e., before reconfiguration) to a challenge that has not been queried for the *previous* state, even if $\mathcal{A}$ knows an adaptively chosen set of challenge/response pairs of the LR-PUF for the previous state and can adaptively obtain challenge/response pairs of the LR-PUF for the current state.
- *Forward unpredictability:* $\mathcal{A}$ cannot predict the response of an LR-PUF for the *current* state S to a challenge that has not yet been queried for the *current* state, even if $\mathcal{A}$ knows an adaptively chosen set of challenge/response pairs of the LR-PUF for the previous state and can adaptively obtain challenge/response pairs of the LR-PUF for the current state (except for the challenge in question).
- *Non-resettability:* $\mathcal{A}$ cannot set the state of the LR-PUF to a specific value.

4 LR-PUF Constructions

In this section, we present two instantiations of our generic LR-PUF construction described in Section 3. The first construction is optimized for the fast generation of LR-PUF responses, while the second construction aims for the area constraints of low-cost devices and provides a trade-off between response generation time and the area size, i.e., the number of gates required.

4.1 Speed-optimized LR-PUF Construction

Our first construction uses a PUF with a large challenge and a large response space and implements the control logic based on a single collision-resistant hash function. The challenge space must be large since otherwise it may be possible to create a complete challenge/response pair (CRP) database, which allows simulating the PUF. A large response space is a fundamental security requirement in many applications such as PUF-based identification/authentication [59,33] and PUF-based block-ciphers [3], where it is crucial that the PUF response to a formerly unknown challenge can be guessed with negligible probability only.

Our first construction is shown in Algorithm 1 and works as follows: Upon $\mathtt{query}_S(c)$, the control logic computes $w \leftarrow \mathsf{Hash}(S\|c)$ and returns $y \leftarrow \mathtt{PUF}(w)$. To reconfigure the LR-PUF, $\mathtt{rcnf}()$ sets the LR-PUF state to $S \leftarrow \mathsf{Hash}(S)$.

The collision-resistance property of the hash function ensures the unpredictability property of the LR-PUF (see Section 2), as we will show later in the formal security analysis. Note that the LR-PUF state is just used to parameterize the hash function and thus needs not to be secret. Hence, to reconfigure the LR-PUF it is sufficient to hash the previous LR-PUF state to obtain a new and independent state (assuming the hash function is collision-resistant).

Alg. 1 Speed-optimized LR-PUF

$\texttt{query}_S(c)$	
$w \leftarrow \mathsf{Hash}(S\|c)$	// $\texttt{mapin}_S(c)$
$y \leftarrow \texttt{PUF}(w)$	
$r \leftarrow y$	// $\texttt{mapout}_S(y)$
Return r	
$\texttt{rcnf}()$	
$S \leftarrow \mathsf{Hash}(S)$	

Alg. 2 Area-optimized LR-PUF

$\texttt{query}_S(c)$	
for $j = 0$ to n **do**	// $\texttt{mapin}_S(c)$
$w_j \leftarrow \mathsf{Hash}(S\|c\|j)$	// $\texttt{mapin}_S(c)$
$y_j \leftarrow \texttt{PUF}(w_j)$	
endfor	
$r \leftarrow (y_0\| \dots \|y_n)$	// $\texttt{mapout}_S(y)$
Return r	
$\texttt{rcnf}()$	
$S \leftarrow \mathsf{Hash}(S)$	

Most PUF constructions that support a large challenge space (e.g., arbiter PUFs [29,44,34]) have only a small response space. Typically several of these PUFs are implemented and evaluated in parallel on the same challenge to generate a large number of PUF response bits in a short amount of time. However, this approach significantly increases the amount of area required for the overall PUF implementation. Hence, we propose a second, area-optimized LR-PUF construction that can be implemented with only one single PUF with a large challenge but small response space.

4.2 Area-optimized LR-PUF construction.

The intuition of our second construction is very similar to the speed-optimized construction of Section 4.1. However, it uses only a single PUF with a large challenge but small response space that is evaluated sequentially n times to generate an n bit LR-PUF response. While this approach increases the time required to generate a large number of PUF response bits, it requires significantly less area than the speed-optimized construction of Section 4.1, providing a trade-off between area consumption and response generation speed.

The underlying PUF of the LR-PUF must be sequentially queried with different challenges to generate a large number of different (and ideally) independent response bits. This can be achieved by including a counter j as additional input to the hash function that now generates a sequence of PUF challenges w_j from the LR-PUF challenge c and the current LR-PUF state S. The corresponding PUF responses y_j are then concatenated to form the response r of the LR-PUF.

Our second construction is depicted in Algorithm 2 and works as follows: On $\texttt{query}_S(c)$, the control logic of the LR-PUF computes $\texttt{mapin}_S(c)$ as $w_j \leftarrow \mathsf{Hash}(S\|w\|j)$ for $j \in \{0, \dots, n\}$, evaluates $y_j \leftarrow \texttt{PUF}(w_j)$, and $\texttt{mapout}_S()$ finally returns $r \leftarrow (y_0\| \dots \|y_n)$. To reconfigure the LR-PUF, $\texttt{rcnf}()$ sets the LR-PUF state to $S \leftarrow \mathsf{Hash}(S)$.

5 Implementation and Performance Evaluation

Both constructions presented in Section 4 are based on PUFs with a large challenge space. The only existing electronic PUFs that provide this feature seem to be arbiter PUFs [29,15]. The hash function of the control logic can be implemented efficiently by using a lightweight block cipher.

We implemented a prototype of both of our LR-PUF constructions on a Xilinx Spartan-6 FPGA board. We instantiated the underlying PUF based on arbiter PUFs that support 64 bit challenges and generate 1 bit responses, following the approach in [53]. The hash function of the control logic is based on the PRESENT block cipher [5] in Davies-Meyer mode [27]. Both resulting LR-PUF implementations use 80 bit challenges and generate 64 bit responses.

Optimization	Response time in clock cycles	Area consumption in slices (gate equivalents) Control logic	Arbiter PUF	Total
Speed	1,069	166	4,288	4,454 (25,265 GE)
Area	64,165	358	67	425 (6,902 GE)

Table 1. Performance results of the LR-PUF constructions presented in Section 4.

We evaluated our implementation with regard to response generation speed and area consumption. Our results are summarized in Table 1. The second column shows the time in number of clock cycles required to compute an LR-PUF response r. The remaining columns show the number of slices required to implement the control logic, the PUF and the overall construction. The area estimation does not include the non-volatile memory for storing the LR-PUF state, which cannot be implemented on FPGA. Our results show that the area-optimized construction requires only about 27% of the area (in gate equivalents) of the speed-optimized construction but is 60 times slower.

Note that our implementation is meant to demonstrate the feasibility of our approach and to obtain performance results. Due to the technical constraints of FPGAs, our implementation does not cover the non-volatile memory for storing the LR-PUF state, which is emulated by providing the state as an input to the FPGA. A secure implementation of our constructions must be based on PUFs providing the unpredictability property (see Section 2). The only known PUFs that have this property seem to be controlled PUFs [13]. A typical controlled PUF prevents model building attacks by hashing the response [13] of its underlying PUF, which hides the response from the adversary [47]. Since PUF responses are noisy, some form of error correction (e.g., a fuzzy extractor [9]) must be applied to the PUF response before it is hashed. Note that our LR-PUF construction already includes a hash function that could also be used for the controlled PUF. Therefore, only the error correction mechanism must be implemented when building an LR-PUF based on a controlled PUF. Moreover, the non-volatile memory and control logic should be protected against fault-injection attacks, e.g., by applying the techniques described in [38,1].

6 Security Definitions and Evaluation

In this section we formally define the LR-PUF security properties of *forward-* and *backward-unpredictability* and show that both are fulfilled by the constructions proposed in Section 4. To this end, we first formalize the security property of unpredictability of a standard PUF.

Along the lines of [2], we define unpredictability of a PUF in terms of an *unpredictability game* between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. $\mathcal{A}$ is first given a PUF and is allowed to query it at most q times. This step allows to model adversaries that are able to learn challenge/response pairs (CRPs) either by direct physical access to the interface of the PUF or by eavesdropping on messages containing PUF challenges and responses. At the end of the game, $\mathcal{A}$ is required to output a (non-trivial) valid pair of a PUF challenge and response.

Unpredictability Game of a PUF

Setup: The challenger $\mathcal{C}$ issues the $\mathtt{PUF}$ to the adversary $\mathcal{A}$.

Queries: Proceeding adaptively, $\mathcal{A}$ queries the $\mathtt{PUF}$ at most q times on challenges w_i ($1 \leq i \leq q$). For each query, $y_i \leftarrow \mathtt{PUF}(w_i)$ is given to $\mathcal{A}$.

Output: Finally, $\mathcal{A}$ outputs a challenge/response pair (w^*, y^*).

Let Q denote the set of all challenges issued by $\mathcal{A}$. We say that $\mathcal{A}$ wins the game, if y^* is a valid PUF response to $\mathtt{PUF}(w^*)$ and $w^* \notin Q$. Conversely, a PUF is unpredictable, if no efficient adversary $\mathcal{A}$ is able to win the game with significant success probability:

Definition 1. *A PUF is (q, ε)-unpredictable, if no probabilistic polynomial time adversary $\mathcal{A}$ that makes at most q queries to the LR-PUF can win the unpredictability game with a probability greater than ε.*

Next, we define backward- and forward-unpredictability of an LR-PUF in terms of a two-stage game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. In the first stage, $\mathcal{A}$ is given oracle access (i.e., access to the interface) of the LR-PUF, from which $\mathcal{A}$ can obtain challenge/response pairs (CRPs) at will. This stage models the ability of $\mathcal{A}$ to obtain challenges and responses (with respect to a fixed internal LR-PUF state) by passive eavesdropping. We also give $\mathcal{A}$ access to the internal LR-PUF state in order to model hardware attacks against the LR-PUF implementation. Once $\mathcal{A}$ has learned enough CRPs, the challenger performs the reconfiguration operation and finally gives $\mathcal{A}$ oracle access to the reconfigured LR-PUF such that $\mathcal{A}$ can obtain CRPs of the reconfigured LR-PUF. At the end of the game, $\mathcal{A}$ outputs a prediction (c^*, r^*) of an LR-PUF challenge/response pair.

More formally, $\mathcal{A} = (\mathcal{A}_L, \mathcal{A}_C)$ consists of two probabilistic polynomial time algorithms, where $\mathcal{A}_L$ interacts with the LR-PUF before reconfiguration and $\mathcal{A}_C$ thereafter. $\mathcal{A}$ engages in the following experiment:

Backward- and Forward-Unpredictability Game of an LR-PUF

Setup: The challenger $\mathcal{C}$ sets up an LR-PUF by choosing a random state S, which is given to the adversary $\mathcal{A} = (\mathcal{A}_L, \mathcal{A}_C)$.

Phase I: $\mathcal{A}_L$ is allowed to call $\texttt{query}_S()$ of the LR-PUF up to q_L times. At the end of phase I, $\mathcal{A}_L$ stops and outputs a log file *st* that is used as input to $\mathcal{A}_C$. We denote with Q_L the set of challenges issued by $\mathcal{A}_L$ during phase I.

Reconfiguration: $\mathcal{C}$ reconfigures the LR-PUF by calling $\texttt{rcnf}()$, which updates the internal LR-PUF state to S'.

Phase II: $\mathcal{A}_C$ is initialized with log file *st* from $\mathcal{A}_L$ and the LR-PUF state S'. $\mathcal{A}_C$ is allowed to query the reconfigured LR-PUF $\texttt{query}_{S'}()$ up to q_C times on arbitrary challenges. We denote with Q_C the set of challenges issued by $\mathcal{A}_C$ during phase II.

Output: Finally, $\mathcal{A}_C$ outputs a challenge/response pair (c^*, r^*) of the LR-PUF.

Depending on whether we consider backward- or forward-unpredictability, we can state different conditions of an adversary being successful: $\mathcal{A}$ wins the *backward-unpredictability* game if r^* is a valid LR-PUF response to $\texttt{query}_{S'}(c^*)$ and $c^* \notin Q_C$. Thus, once the LR-PUF has been reconfigured, the adversary cannot output a (non-trivial) challenge/response pair for the *reconfigured* LR-PUF. Conversely, $\mathcal{A}$ wins the *forward-unpredictability* game if r^* is a valid LR-PUF response to $\texttt{query}_S(c^*)$ and $c^* \notin Q_L$. Thus, an adversary, who has access to a reconfigured LR-PUF cannot predict (non-trivial) responses of the LR-PUF *before* reconfiguration happened. We say that an LR-PUF is backward- (resp. forward-) unpredictable, if no efficient adversary $\mathcal{A}$ can win the game with significant success probability:

Definition 2. *An LR-PUF is (q_L, q_C, ε)-backward unpredictable (resp. forward-unpredictable), if no probabilistic polynomial time adversary $\mathcal{A}$ that makes at most q_L queries in phase I and at most q_C queries in phase II, is able to win the backward-unpredictability (resp. forward-unpredictability) game with a probability greater than ε.*

Both constructions of Section 4 achieve backward- and forward- unpredictability:

Proposition 1. *The speed-optimized LR-PUF construction shown in Section 4.1 is (q_L, q_C, ε)-backward unpredictable (resp. forward-unpredictable), if* $\mathsf{Hash}()$ *is collision-resistant and the underlying PUF is $(q_L + q_C, \varepsilon)$-unpredictable.*

Proposition 2. *The area-optimized LR-PUF construction shown in Section 4.2 is* (q_L, q_C, ε)*-backward unpredictable (resp. forward-unpredictable), if* $\mathsf{Hash}()$ *is collision-resistant and the underlying PUF is* $(n(q_L + q_C), \varepsilon)$*-unpredictable.*

In the following we only provide a proof sketch, while the detailed proofs can be found in Appendix A.

The proofs of both propositions follow from the standard reductionist approach. In particular, we show that any adversary $\mathcal{A}$ against the LR-PUF can be converted into an adversary $\mathcal{B}$ that either breaks the collision-resistance of the hash function $\mathsf{Hash}()$ or the unpredictability of the underlying physical PUF (Definition 1). To this end, $\mathcal{B}$ simulates $\mathcal{A}$: Whenever $\mathcal{A}$ makes an LR-PUF query $\mathtt{query}_S(c)$, $\mathcal{B}$ simulates the response y to this query, i.e., $\mathcal{B}$ computes $w \leftarrow \mathtt{mapin}_S(c)$ using the (known) internal LR-PUF state S, evaluates the physical PUF on w and returns the obtained response $y \leftarrow \mathtt{PUF}(w)$ to $\mathcal{A}$. Once the simulation stops, it can easily be seen that either a hash collision or a valid prediction of a challenge/response pair of the physical PUF can be extracted from $\mathcal{A}$'s output.

7 Applications

7.1 LR-PUF-based Authentication Tokens

Electronic payment and ticketing has been gradually introduced in many countries over the past few years (see, e.g., [42,7,40]). Typically, these systems are using RFID-enabled tokens and provide different types of electronic transit tickets. Given the typically large number of tickets used in an electronic transit ticket system and the costs per token (typically between 1-3 Euros), from an economic perspective it may be worthwhile to consider recycling of RFID-based tickets. In fact, some ticketing systems (e.g., the Dutch transportation system [43]) allow recharging RFID-based tickets with money and returning used tickets to the vendor with possible restitution of preloaded money left on the ticket. Moreover, many U.S. and European governments make manufacturers and importers of electronic products responsible for the disposal of their products when discarded by the consumer (see, e.g., [6,11]). In this context, recyclable tokens can help to save waste disposal costs and to reduce the amount of electronic waste. In this section, we discuss how LR-PUFs could be used to enhance the security of electronic ticketing and payment systems while at the same time enabling secure and privacy-preserving recycling of used RFID-tickets.

There are several proprietary solutions for electronic tickets in practice. Most of them are based on widely used RFID tokens, where the most prominent example is the MiFare family produced by NXP Semiconductors [41]. There are several hard- and software attacks against MiFare Classic tokens [39,49,12], which use a proprietary encryption algorithm that has been completely broken [8]. However, other MiFare products are claimed not to be affected. A recent attack on MiFare Classic 4K chipcards concerns the Dutch electronic payment and transit ticket system [43]: Using a MiFare compatible card reader and a software from the Internet, an average user can add debit to his RFID-based transit ticket without being detected [61,31].

In this context, PUFs could provide a cost-effective security mechanism: Authentication based on PUFs can prevent copying and manipulating the information (i.e., the debit of the RFID-based ticket and/or the user's rights) by cryptographically binding this data to the physical characteristics of the underlying RFID chip. Existing PUF-based authentication schemes (see, e.g., [46,19,44,48,58]) typically assume each device, i.e., each token $\mathcal{T}$, to be equipped with a PUF, whereas the verifier $\mathcal{V}$ maintains a database $\mathcal{D}$, i.e., a set of challenge/response pairs (CRPs) of each ticket. In the authentication protocol, $\mathcal{V}$ chooses a random challenge from $\mathcal{D}$ and sends it to $\mathcal{T}$, which then returns some response. $\mathcal{V}$ accepts if the response of $\mathcal{T}$ matches the one in $\mathcal{D}$.

Using LR-PUFs instead of non-reconfigurable PUFs would allow for cost-effective, secure and privacy-preserving recycling of RFID-based tickets: By reconfiguring the LR-PUF all information and access rights bound to $\mathcal{T}$ are securely "erased", which cannot be achieved with non-reconfigurable PUFs. However, reconfiguring the LR-PUF invalidates the CRP database $\mathcal{D}$ of $\mathcal{V}$, which means that after each reconfiguration of $\mathcal{T}$ a new CRP database must be established. To counter this problem, $\mathcal{V}$ could know the LR-PUF state S of each token and maintain a CRP database $\mathcal{D}'$ of the PUF underlying the LR-PUF, which can be seen as the "authentication secrets" of the token. This is common in ticketing applications because usually the verifier is the ticket issuer who typically knows the authentication secrets of all tokens. Since the algorithms of the control unit, i.e., the input and output transition functions `mapin`() and `mapout`(), respectively, and the state update algorithm `rcnf`(), are publicly known, $\mathcal{V}$ could use $\mathcal{D}'$ to recompute the LR-PUF response for any state of $\mathcal{T}$ and compare it to the response sent by $\mathcal{T}$. $\mathcal{V}$ accepts if the response of $\mathcal{T}$ matches the one recomputed based on $\mathcal{D}'$ and S.

7.2 LR-PUF-based RFID-enabled Luggage Tags

Many airlines have started replacing paper-based tickets with electronic tickets. However, they still print luggage tags, which are increasingly equipped with disposable RFID chips. The purpose of these chips is to ease the tracking of individual luggage in the process of loading. However, RFID-enabled labels could be read out even without visual contact. This may allow several attacks ranging from copying luggage tags to smuggle in additional luggage in the name of another passenger. Moreover, RFID-enabled luggage tags may disclose personal information on their owner (e.g., name, number of luggage pieces and/or luggage weight), which could be used to track the user on the airport or provides useful information to luggage thieves. To solve these problems, travellers could purchase or rent a more powerful LR-PUF-enabled RFID token that is put into the luggage or that could even be embedded into new generations of suitcases. Each time the traveller checks in, his RFID-based tag is reconfigured by the airline attendant, which securely erases the previous information stored on it. This prevents tracking the traveler for more than one flight and impedes misrouting of luggage due to old travel information. Further, to avoid illegitimate tracking of travellers, the RFID-enabled luggage tag after could be reconfigured or temporarily disabled once the passenger leaves the baggage claim area.

7.3 Other Applications Envisaged

One can find many other applications that could take advantage of LR-PUFs. Examples include, secure deletion and/or update of cryptographic secrets in PUF-based key storage [59,33] and PUF-based cryptographic primitives [3], where the reconfiguration of the PUF ensures that old secrets cannot be retrieved any more. Another example are solutions to prevent downgrading of software [26] by binding the software to the PUF, where reconfiguring the PUF invalidates the old software version such that only the latest version can be used. A concrete LR-PUF-based instantiation of such a system has been recently presented in [10].

8 Conclusion

We presented the concept and formalization of logically reconfigurable PUFs (LR-PUFs), which utilize a control logic to enable dynamic reconfigurability for existing, typically static PUFs without physically replacing or modifying them. We introduced two different constructions for LR-PUFs: Our first construction is optimized for response generation speed, while the second one aims for resource-constrained embedded devices, like RFID tags. Furthermore, we have shown that both constructions achieve the security properties of backward- and forward unpredictability, which are two desirable properties in the context of PUF-based cryptographic applications like key storage and device identification. Finally, we showed how LR-PUFs could be applied in the context of recyclable (access) tokens to enhance the security properties of existing solutions while providing a means for secure recycling of PUF-based access tokens. Future work includes the design and implementation of LR-PUF based security solutions for (privacy-preserving) device authentication and IP protection.

Acknowledgements

We thank Dries Schellekens (K.U. Leuven) for his helpful comments, Patrick Koeberl (Intel) and Jérôme Quevremont (Thales) for several useful discussions on hardware attacks and use cases, and Timm Korte (Sirrix AG) for providing us his implementation of PRESENT. This work has been supported in part by the European Commission under grant agreement ICT-2007-238811 UNIQUE.

References

1. Akdemir, K.D., Wang, Z., Karpovsky, M.G., Sunar, B.: Design of cryptographic devices resilient to fault injection attacks using nonlinear robust codes. Fault Analysis in Cryptography (2011)
2. Armknecht, F., Maes, R., Sadeghi, A.R., Standaert, F.X., Wachsmann, C.: A formal foundation for the security features of physical functions. In: IEEE Symposium on Security and Privacy. pp. 397–412. IEEE Computer Society (May 2011)
3. Armknecht, F., Maes, R., Sadeghi, A.R., Sunar, B., Tuyls, P.: Memory leakage-resilient encryption based on physically unclonable functions. In: Advances in Cryptology (ASIACRYPT). LNCS, vol. 5912, pp. 685–702 (2009)
4. Armknecht, F., Sadeghi, A.R., Visconti, I., Wachsmann, C.: On RFID privacy with mutual authentication and tag corruption. In: International Conference on Applied Cryptography and Network Security (ACNS). LNCS, vol. 6123, pp. 493–510. Springer (2010)
5. Bogdanov, A., Knudsen, L., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: Cryptographic Hardware and Embedded Systems (CHES). LNCS, vol. 4727, pp. 450–466. Springer (2007)
6. Californians Against Waste: E-waste laws in other states. `http://www.cawrecycles.org/issues/ca_e-waste/other_states` (April 2011)
7. Calypso Networks Association: Website. `http://www.calypsonet-asso.org/` (August 2011)
8. Courtois, N.T., Nohl, K., O'Neil, S.: Algebraic attacks on the Crypto-1 stream cipher in MiFare Classic and Oyster Cards. Cryptology ePrint Archive, Report 2008/166 (2008)
9. Dodis, Y., Reyzin, L., Smith, A.: Security with Noisy Data, chap. Fuzzy Extractors, pp. 79–99. Springer-Verlag (2007)
10. Eichhorn, I., Koeberl, P., van der Leest, V.: Logically reconfigurable PUFs: Memory-based secure key storage. In: ACM Workshop on Scalable Trusted Computing (ACM STC). ACM Press (2011)
11. European Commission: Waste electrical and electronic equipment website. `http://ec.europa.eu/environment/waste/weee/index_en.htm` (April 2011)
12. Garcia, F.D., de Koning Gans, G., Muijrers, R., van Rossum, P., Verdult, R., Schreur, R.W., Jacobs, B.: Dismantling MiFare classic. In: Jajodia, S., Lopez, J. (eds.) 13th European Symposium on Research in Computer Security (ESORICS). LNCS, vol. 5283, pp. 97–114. Springer Verlag (2008)
13. Gassend, B., Clarke, D., van Dijk, M., Devadas, S.: Controlled physical random functions. In: Computer Security Applications Conference. pp. 149–160. IEEE Computer Society (2002)
14. Gassend, B., Clarke, D., van Dijk, M., Devadas, S.: Silicon physical random functions. In: ACM Conference on Computer and Communications Security (ACM CCS). pp. 148–160 (2002)
15. Gassend, B., Lim, D., Clarke, D., van Dijk, M., Devadas, S.: Identification and authentication of integrated circuits: Research articles. Concurr. Comput. : Pract. Exper. 16(11), 1077–1098 (2004)
16. Guajardo, J., Kumar, S., Schrijen, G.J., Tuyls, P.: Brand and IP protection with physical unclonable functions. In: IEEE International Symposium on Circuits and Systems, ISCAS 2008. pp. 3186–3189 (May 2008)
17. Guajardo, J., Kumar, S., Schrijen, G.J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. In: Cryptographic Hardware and Embedded Systems, CHES 2007. Lecture Notes in Computer Science, vol. 4727, pp. 63–80. Springer Berlin / Heidelberg (2007)

18. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. In: Workshop on Cryptographic Hardware and Embedded Systems (CHES). LNCS, vol. 4727, pp. 63–80 (2007)
19. Holcomb, D.E., Burleson, W.P., Fu, K.: Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. In: Conference on RFID Security (RFIDSec) (2007)
20. Intrinsic ID: Product webpage. `http://www.intrinsic-id.com/products.htm` (April 2011)
21. Juels, A.: RFID security and privacy: A research survey. Journal of Selected Areas in Communication 24(2), 381–395 (February 2006)
22. Katzenbeisser, S., Kocabaş, U., van der Leest, V., Sadeghi, A.R., Schrijen, G.J., Schröder, H., Wachsmann, C.: Recyclable PUFs: Logically reconfigurable PUFs. In: Workshop on Cryptographic Hardware and Embedded Systems (CHES). vol. 6917, pp. 374–389. Springer Berlin / Heidelberg, Berlin, Heidelberg (Sep 2011)
23. Katzenbeisser, S., Kocabaş, U., van der Leest, V., Sadeghi, A.R., Schrijen, G.J., Wachsmann, C.: Recyclable PUFs: logically reconfigurable PUFs. Journal of Cryptographic Engineering pp. 1–10 (Sep 2011)
24. Kumar, S., Guajardo, J., Maes, R., Schrijen, G.J., Tuyls, P.: Extended abstract: The butterfly PUF protecting IP on every FPGA. In: IEEE International Workshop on Hardware-Oriented Security and Trust, HOST 2008. pp. 67–70 (June 2008)
25. Kumar, S., Guajardo, J., Maes, R., Schrijen, G.J., Tuyls, P.: Extended abstract: The butterfly PUF protecting IP on every FPGA. In: IEEE Workshop on Hardware-Oriented Security and Trust (HOST). pp. 67–70 (2008)
26. Kursawe, K., Sadeghi, A.R., Schellekens, D., Tuyls, P., Scoric, B.: Reconfigurable physical unclonable functions — Enabling technology for tamper-resistant storage. In: IEEE International Workshop on Hardware-Oriented Security and Trust (HOST). pp. 22–29. IEEE Computer Society, San Francisco, CA, USA (2009)
27. Lai, X., Massey, J.: Hash functions based on block ciphers. In: Rueppel, R. (ed.) Advances in Cryptology (EUROCRYPT). LNCS, vol. 658, pp. 55–70. Springer (1993)
28. Lao, Y., Parhi, K.K.: Novel reconfigurable silicon unclonable functions. In: Workshop on Foundations of Dependable and Secure Cyber-Physical Systems (FDSCPS) (April 11 2011)
29. 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 application. In: Symposium on VLSI Circuits. pp. 176–159 (2004)
30. van der Leest, V., Schrijen, G.J., Handschuh, H., Tuyls, P.: Hardware intrinsic security from D flip-flops. In: ACM Workshop on Scalable Trusted Computing (ACM STC). pp. 53–62 (2010)
31. Letter from Dutch minister on OV-chipkaart: `https://zoek.officielebekendmakingen.nl/dossier/32440/kst-23645-415.html`
32. Lim, D.: Extracting Secret Keys from Integrated Circuits. Master's thesis, MIT, MA, USA (May 2004)
33. Lim, D., Lee, J.W., Gassend, B., Suh, G.E., van Dijk, M., Devadas, S.: Extracting secret keys from integrated circuits. IEEE Transactions on VLSI Systems 13(10), 1200–1205 (2005)
34. Lin, L., Holcomb, D., Krishnappa, D.K., Shabadi, P., Burleson, W.: Low-power sub-threshold design of secure physical unclonable functions. In: ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED). pp. 43–48 (2010)
35. Maes, R., Tuyls, P., Verbauwhede, I.: Intrinsic PUFs from flip-flops on reconfigurable devices. In: Workshop on Information and System Security (WISSec). p. 17 (2008)
36. Maes, R., Verbauwhede, I.: Physically unclonable functions: A study on the state of the art and future research directions. In: Sadeghi, A.R., Naccache, D. (eds.) Towards Hardware-Intrinsic Security, pp. 3–37. Information Security and Cryptography, Springer Berlin Heidelberg (2010)
37. Maiti, A., Casarona, J., McHale, L., Schaumont, P.: A large scale characterization of RO-PUF. In: IEEE Symposium on Hardware-Oriented Security and Trust (HOST). pp. 94–99 (2010)
38. Monnet, Y., Renaudin, M., Leveugle, R.: Designing resistant circuits against malicious faults injection using asynchronous logic. IEEE Trans. Comput. 55, 1104–1115 (September 2006)
39. Nohl, K., Plötz, H.: MiFare — Little security despite obscurity. `http://events.ccc.de/congress/2007/Fahrplan/events/2378.en.html` (2007)
40. NXP Semiconductors: MiFare applications. `http://www.mifare.net/applications/` (August 2011)
41. NXP Semiconductors: MiFare smartcard ICs. `http://www.mifare.net/products/smartcardics/` (February 2011)
42. Octopus Holdings: Website. `http://www.octopus.com.hk/en/` (August 2011)
43. OV-Chipkaart: Website. `http://www.ov-chipkaart.nl/` (April 2011)
44. Öztürk, E., Hammouri, G., Sunar, B.: Towards robust low cost authentication for pervasive devices. In: IEEE International Conference on Pervasive Computing and Communications (PERCOM'08). pp. 170–178. IEEE Computer Society (March 2008)
45. Pappu, R.S., Recht, B., Taylor, J., Gershenfeld, N.: Physical one-way functions. Science 297, 2026–2030 (2002)
46. Ranasinghe, D.C., Engels, D.W., Cole, P.H.: Security and privacy: Modest proposals for low-cost RFID systems. Auto-ID Labs Research Workshop (September 2004)

47. Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., Schmidhuber, J.: Modeling attacks on physical unclonable functions. In: ACM conference on Computer and communications security (ACM CCS). pp. 237–249 (2010)
48. Sadeghi, A.R., Visconti, I., Wachsmann, C.: PUF-enhanced RFID security and privacy. Workshop on Secure Component and System Identification (SECSI) (2010)
49. Schreur, R.W., van Rossum, P., Garcia, F., Teepe, W., Hoepman, J.H., Jacobs, B., de Koning Gans, G., Verdult, R., Muijrers, R., Kali, R., Kali, V.: Security flaw in MiFare Classic. `http://www.sos.cs.ru.nl/applications/rfid/pressrelease.en.html` (March 2008)
50. Schulz, S., Sadeghi, A.R., Wachsmann, C.: Short paper: lightweight remote attestation using physical functions. In: Proceedings of the fourth ACM conference on Wireless network security. pp. 109–114. WiSec '11, ACM, New York, NY, USA (2011)
51. Skorobogatov, S.: Semi-invasive attacks — A new approach to hardware security analysis. Technical Report UCAM-CL-TR-630, University of Cambridge, 15 JJ Thomson Avenue, Cambridge CB03 0FD, UK (April 2005)
52. Skorobogatov, S.: Local heating attacks on Flash memory devices. In: IEEE International Workshop on Hardware-Oriented Security and Trust (HOST'09). pp. 1–6. IEEE Computer Society (July 27 2009)
53. Soybali, M., B. Ors, G.S.: Implementation of a PUF circuit on an FPGA. In: IFIP International Conference on New Technologies Mobility and Security. pp. 1–5. IEEE Computer Society (2011)
54. Su, Y., Holleman, J., Otis, B.: A 1.6pJ/bit 96% stable chip-ID generating circuit using process variations. In: IEEE International Solid-State Circuits Conference (ISSCC). pp. 406–611 (2007)
55. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: Design Automation Conference. pp. 9–14 (2007)
56. Tuyls, P., Batina, L.: RFID-tags for anti-counterfeiting. In: The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13–17, 2005, Proceedings. Lecture Notes on Computer Science (LNCS), vol. 3860, pp. 115–131. Springer Verlag (2006)
57. Tuyls, P., Schrijen, G.J., Škorić, B., van Geloven, J., Verhaegh, N., Wolters, R.: Read-proof hardware from protective coatings. In: Workshop on Cryptographic Hardware and Embedded Systems (CHES). LNCS, vol. 4249, pp. 369–383 (2006)
58. Verayo, Inc.: Product webpage. `http://www.verayo.com/product/products.html` (April 2011)
59. Škorić, B., Tuyls, P., Ophey, W.: Robust key extraction from physical uncloneable functions. In: Applied Cryptography and Network Security (ACNS). LNCS, vol. 3531, pp. 407–422 (2005)
60. Weis, S.A., Sarma, S.E., Rivest, R.L., Engels, D.W.: Security and privacy aspects of low-cost radio frequency identification systems. In: Proc. of PerCom. LNCS, vol. 2802, pp. 50–59. Springer (2003)
61. Wikipedia: OV-Chipkaart (August 2011), `http://en.wikipedia.org/wiki/OV-chipkaart`

A Security Proofs

A.1 Speed-optimized LR-PUF Construction

In this section, we prove Proposition 1, i.e., the forward- and backward unpredictability of our speed-optimized LR-PUF construction described in Section 4.1. In the following, we only consider attacks against backward-unpredictability. Adapting the proof for attacks against forward-unpredictability is straightforward.

Proof. Given an adversary $\mathcal{A} = (\mathcal{A}_L, \mathcal{A}_C)$ that breaks the backward-unpredictability of the speed-optimized LR-PUF construction described in Section 4.1 with non-negligible probability, we construct an adversary $\mathcal{B}$ that (1) breaks the unpredictability of the underlying physical PUF or (2) finds collisions for the collision-resistant hash function $\mathsf{Hash}()$ with the same success probability as $\mathcal{A}$.

Suppose that $\mathcal{A}$ adaptively queries LR-PUF challenges c_i and obtains the corresponding LR-PUF responses r_i for $1 \leq i \leq (q_C + q_L)$. Recall that $r_i = \mathtt{PUF}(w_i)$ with

$$w_i = \begin{cases} \mathsf{Hash}(S||c_i) & \text{for } 1 \leq i \leq q_C \\ \mathsf{Hash}(S'||c_i) & \text{for } (q_C+1) \leq i \leq (q_C + q_L). \end{cases}$$

We denote with (c^*, r^*) $\mathcal{A}$'s prediction of a challenge/response pair of the LR-PUF, where r^* is a valid response to $\mathtt{PUF}(w^*)$ with $w^* = \mathsf{Hash}(S'||c^*)$. We distinguish two types of adversaries:

Type-1 adversary: $\mathcal{A}$ is a type-1 adversary if there exists an index i such that $w_i = w^*$ and $1 \leq i \leq (q_L + q_C)$.

Type-2 adversary: Otherwise, $\mathcal{A}$ is a type-2 adversary.

Simulation. $\mathcal{B}$ first chooses a random LR-PUF state S, hands it over to $\mathcal{A}_L$ and runs a black-box simulation of the challenger $\mathcal{C}$ of the backward-unpredictability game (see Definition 2). When $\mathcal{A}_L$ sends a challenge c_i, $\mathcal{B}$ simulates the corresponding LR-PUF response as follows: $\mathcal{B}$ sets $w_i \leftarrow \mathsf{Hash}(S||c_i)$, queries the physical PUF $r_i \leftarrow \mathtt{PUF}(w_i)$, stores (c_i, w_i, r_i) in some (initially empty) list $\mathcal{L}$, and forwards r_i to $\mathcal{A}_L$. At some point $\mathcal{A}_L$ stops and outputs some log file *st*. After obtaining *st*, $\mathcal{B}$ changes the LR-PUF state S to a fresh, randomly chosen state S' in order to reconfigure the LR-PUF. Then, $\mathcal{B}$ initializes $\mathcal{A}_C$ with S' and *st*, and continues to simulate $\mathcal{C}$ in a black-box manner. When $\mathcal{A}_C$ sends a challenge c_i, $\mathcal{B}$ simulates the response of the reconfigured LR-PUF as follows: $\mathcal{B}$ sets $w_i \leftarrow \mathsf{Hash}(S'||c_i)$, queries the physical PUF on $r_i \leftarrow \mathtt{PUF}(w_i)$, stores (c_i, w_i, r_i) in $\mathcal{L}$, and forwards r_i to $\mathcal{A}_C$. At some point $\mathcal{A}_C$ stops and returns a challenge/response pair (c^*, r^*).

Type-1 adversary. At the end of the simulation, $\mathcal{B}$ parses the list $\mathcal{L}$ and records the index $1 \leq i \leq (q_L + q_C)$ for which $w_i = w^*$. Since $\mathcal{A}$ is a successful type-1 adversary and $\mathcal{B}$ performs a perfect simulation, such an index exists. Even though $w^* = w_i$, we have by the assumptions of the game (see Definition 2) that $c^* \neq c_i$. Thus, $\mathcal{B}$ found a collision of $\mathsf{Hash}()$, which contradicts the collision resistance property of hash function $\mathsf{Hash}()$.

Type-2 adversary. At the end of the simulation, $\mathcal{A}_C$ outputs a challenge/response pair (c^*, r^*) of the LR-PUF. Recall that $w^* = \mathsf{Hash}(S'||c^*)$. Suppose that $\mathcal{A}$ is a type-2 adversary. Since $\mathcal{B}$ performs a perfect simulation, there exists *no* index i such that $w_i = w^*$. Thus, $\mathcal{B}$ has never queried w^* to the physical PUF and thus, (w^*, r^*) is a valid prediction of a challenge/response pair of the underlying PUF, which contradicts the unpredictability of the physical PUF. The success probability of $\mathcal{B}$ is equal to the success probability of $\mathcal{A}$ and $\mathcal{B}$ makes $q_C + q_L$ queries to the physical PUF. This proves the proposition. □

A.2 Area-optimized LR-PUF Construction

In this section, we prove Proposition 2, i.e., the forward- and backward unpredictability of our area-optimized LR-PUF construction presented in Section 4.2. In the following, we only consider attacks against backward-unpredictability. Adapting the proof for attacks against forward-unpredictability is straightforward. The main idea of the proof is similar to the proof for the speed-optimized LR-PUF construction.

Proof. Given an adversary $\mathcal{A} = (\mathcal{A}_L, \mathcal{A}_C)$ that can break the backward-unpredictability of the area-optimized LR-PUF construction described in Section 4.2 with non-negligible probability, we construct an adversary $\mathcal{B}$ that (1) breaks the unpredictability of the underlying PUF or that (2) finds collisions for the collision-resistant hash function $\mathsf{Hash}()$ with the same success probability as $\mathcal{A}$.

Suppose that $\mathcal{A}$ adaptively queries challenges c_i and obtains the corresponding LR-PUF responses r_i for $1 \leq i \leq (q_L + q_C)$. Recall that $r_i = y_i^1, \ldots, y_i^n$ with $y_i^j = \mathtt{PUF}(w_i^j)$ where

$$w_i^j = \begin{cases} \mathsf{Hash}(S||c_i||j) & \text{for } 1 \leq i \leq q_L \text{ and } 1 \leq j \leq n \\ \mathsf{Hash}(S'||c_i||j) & \text{for } (q_L + 1) \leq i \leq (q_L + q_C) \text{ and } 1 \leq j \leq n. \end{cases}$$

Let (c_*, r_*) be the adversary's prediction of the LR-PUF, where $r_* = y_*^1, \ldots, y_*^n$ and $y_*^j \leftarrow \mathtt{PUF}(w_*^j)$ with $w_*^j = \mathsf{Hash}(S'||c_*||j)$. We distinguish two adversaries:

Type-1 adversary: $\mathcal{A}$ is a type-1 adversary if there exist indices i, j, k such that $w_i^j = w_*^k$ with $1 \leq i \leq (q_L + q_C)$ and $1 \leq j, k \leq n$.

Type-2 adversary: $\mathcal{A}$ is a type-2 adversary if no such indices exist.

Simulation. $\mathcal{B}$ first chooses a random state S for the LR-PUF, which is given to $\mathcal{A}_L$. Then, $\mathcal{B}$ runs a black-box simulation of the challenger $\mathcal{C}$ of the backward-unpredictability game (see Definition 2). When $\mathcal{A}_L$ sends a challenge c_i, $\mathcal{B}$ simulates the corresponding LR-PUF response as follows: $\mathcal{B}$ sets $w_i^j \leftarrow \mathsf{Hash}(S||c_i||j)$ and queries the underlying physical PUF on w_i^j for $1 \leq j \leq n$. In turn, $\mathcal{B}$ obtains $r_i = y_i^1, \ldots, y_i^n$, where $y_i^j = \mathtt{PUF}(w_i^j)$, stores (c_i, w_i^j, y_i^j) for $1 \leq j \leq n$ in some (initially empty) list $\mathcal{L}$, and forwards r_i to $\mathcal{A}_L$. At some point $\mathcal{A}_L$ stops and returns some log file *st*. After obtaining *st*, $\mathcal{B}$ changes the LR-PUF state S to a fresh, randomly chosen state S' in order to reconfigure the LR-PUF. Then, $\mathcal{B}$ initializes $\mathcal{A}_C$ with S' and *st*, and continues to simulate $\mathcal{C}$ in a black-box manner. When $\mathcal{A}_C$ sends a challenge c_i, $\mathcal{B}$ simulates the corresponding response of the reconfigured LR-PUF as follows: $\mathcal{B}$ sets $w_i^j \leftarrow \mathsf{Hash}(S'||c_i||j)$ for $1 \leq j \leq n$ and queries the physical PUF on w_i^j, so that $y_i^j \leftarrow \mathtt{PUF}(w_i^j)$. In turn, $\mathcal{B}$ obtains $r_i = y_i^1, \ldots, y_i^n$, stores (c_i, w_i^j, y_i^j) for $1 \leq j \leq n$ in $\mathcal{L}$, and forwards r_i to $\mathcal{A}_C$. At some point $\mathcal{A}_C$ stops and returns a challenge/response pair (c_*, r_*).

Type-1 adversary. At the end of the simulation, $\mathcal{B}$ parses the list $\mathcal{L}$ and records indices i, j, k for which $w_i^j = w_*^k = \mathsf{Hash}(S'||c_*||k)$ with $1 \leq i \leq (q_L + q_C)$ and $1 \leq j, k \leq n$. Since $\mathcal{A}$ is a successful type-1 adversary and $\mathcal{B}$ performs a perfect simulation, such indices exist. Thus, $\mathcal{B}$ has found a collision of the hash function $\mathsf{Hash}()$, since by the assumption on the game, $c_i \neq c_*$. However, this is a contradiction to the collision-resistance property of the hash function $\mathsf{Hash}()$.

Type-2 adversary. At the end of the simulation, $\mathcal{A}_C$ returns a valid challenge/response pair $\big(c_*, (y_*^1, \ldots, y_*^n)\big)$ of the LR-PUF. $\mathcal{B}$ computes $w_*^j \leftarrow \mathsf{Hash}(S'||c_*||j)$ for all $1 \leq j \leq n$ and records the index j of the first element w_*^j that is *not* in $\mathcal{L}$. Since $\mathcal{A}$ is a type-2 adversary and the simulation by $\mathcal{B}$ is perfect, this index exists. Finally, $\mathcal{B}$ outputs (w_*^j, y_*^j) as a valid prediction of a challenge/response pair of the physical PUF, which has not been queried before. This contradicts the unpredictability property of the physical PUF. The success probability of $\mathcal{B}$ equals that of $\mathcal{A}$. Furthermore, $\mathcal{B}$ makes $n(q_L + q_C)$ queries to the physical PUF. This proves the proposition. $\square$