# Foundations of Reconfigurable PUFs

Jonas Schneider and Dominique Schröder

Saarland University

**Abstract.** A Physically Unclonable Function (PUF) can be seen as a source of randomness that can be challenged with a stimulus and responds in a way that is to some extent unpredictable. PUFs can be used to provide efficient solutions for common cryptographic primitives such as identification/authentication schemes, key storage, and hardware-entangled cryptography. Moreover, Brzuska et al. have recently shown, that PUFs can be used to construct UC secure protocols (CRYPTO 2011). Most PUF instantiations, however, only provide a static challenge/response space which limits their usefulness for practical instantiations. To overcome this limitation, Katzenbeisser et al. (CHES 2011) introduced Logically Reconfigurable PUFs (LR-PUFs), with the idea to introduce an "update" mechanism that changes the challenge/response behaviour without physically replacing or modifying the hardware.
In this work, we revisit LR-PUFs. We propose several new ways to characterize the unpredictability of LR-PUFs covering a broader class of realistic attacks and examine their relationship to each other. In addition, we reconcile existing constructions with these new characterizations and show that they can withstand stronger adversaries than originally shown. Since previous constructions are insecure with respect to our strongest unpredictability notion, we propose a secure construction which relies on the same assumptions and is almost as efficient as previous solutions.
This work appeared as part of the Bachelor thesis of Jonas Schneider submitted at Saarland University on September 11, 2013.

**Keywords:** Physically Unclonable Functions, Logically Reconfigurable, Tamper-resistance.

## 1 Introduction

Physically Unclonable Function (PUFs) are non-programmable hardware tokens that can be challenged with a stimulus and output responses that are unpredictable. The unpredictable output of the PUFs results from the manufactory process and cannot be controlled even by the producer itself. PUFs are extremely useful to build cryptographic applications, such as e.g., identification/authentication schemes, key storage, and hardware-entangled cryptography, and also to obtain protocols that are secure in Canetti's UC framework as shown by Brzuska et al. [4]. Most PUF instantiations, however, only provide a static challenge/response space which limits their usefulness for practical instantiations.

To overcome this limitation, Katzenbeisser et al. [8] introduced Logically Reconfigurable PUFs (LR-PUFs), with the idea to introduce an "update" mechanism that allows to change the input/output behaviour of a PUF. In this work, we revisit LR-PUFs presenting several ways to characterize the unpredictability, we examine their relationship to each other, and we show that previous constructions can withstand stronger adversaries than originally shown.

### 1.1 Background and Related Work

Physically Unclonable Functions were proposed as *Physical One-Way Functions* [12]. They consist of a physical device which can be challenged with a stimulus and responds in a way that is to some extent unpredictable.

- The PUF provides unpredictable, but robust responses. This means the response for a given challenge does not vary beyond a typically low bound, but it should be not be possible to predict the response for a stimulus that has not yet been applied.
- The PUF is not clonable, i.e., one cannot produce a device which exhibits the same response behavior. This goes even as far as not being able to recreate the same behavior if one has physical access to the device itself and not just a list of challenge-response pairs.

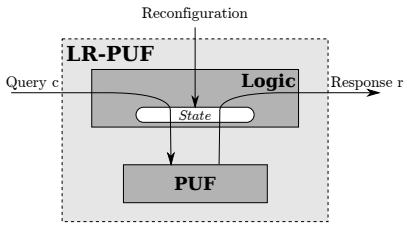


Fig. 1 Schematic of a generic Logically Reconfigurable PUF construction.

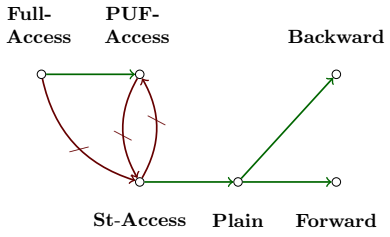


Fig. 2 The relations between the unpredictability notions we introduce.

A third property that is usually cited is tamper-evidence, which is closely related to unclonability. These properties are derived from imprecisions in the manufacturing process of some other object, such as differing gate delays in an integrated circuit. For a survey on the multitude of different PUF constructions, we refer the reader to [10]. A formal description of these properties has been the subject of many research efforts. An in-depth treatment to the definition of these properties that proposes a game-based framework for the description of PUF properties and even PUF creation is given in [2]. Brzuska et al. propose an entropy based characterization of the unpredictability property and examine how PUFs can be integrated into the UC-security framework in [4]. We use their formalization of PUFs as families of distributions.

The tamper-evidence property forbids most PUF designs to have a stimulus-response behavior which is anything but static. In applications where PUFs serve as physical tokens for e.g., access control this can be a disadvantage. Consider for example that the PUF-token should be transferable to a different person. The traditional PUF designs do not allow this, unless the new owner of the PUF should be allowed to carry the same credentials as the previous owner. Thus, there have been efforts to construct reconfigurable PUFs, the first being Controlled Physical Random Functions [6], and a more recent one being Reconfigurable PUFs, short rPUFs [9].

Both of these approaches have their own limitations. Controlled PUFs effectively limit the PUF response space a single user can draw from, thereby lessening security. Physically reconfigurable rPUFs require a potentially costly physical reconfiguration process, and there are no guarantees regarding the effectiveness of that process.

A solution which aims to sidestep these limitations are Logically Reconfigurable PUFs (LR-PUFs) [8]. In this approach, reconfiguration leaves the physical device untouched and is instead performed on a piece of state, which is stored together with the PUF. The stimulus mechanism of the PUF is encapsulated in a query algorithm, which processes challenges by entangling them in some way with the current state of the device. See Figure 1 for a visual representation of the LR-PUF concept. The idea is a combination of the state and the challenge to perform a *logical* reconfiguration, in which a new state is chosen instead of altering the actual physical device. This preserves the original input/output characteristics of the PUF, but does not require physical manipulation of the device.

### 1.2 Applications

PUFs have a wide range of applications such as key extraction and authentication [18, 15, 16, 17], remote attestation [15], and tamper-proof and fault injection resilient implementations of cryptographic primitives [14, 3, 1]. Most of these applications assume that the PUFs are somewhat ideal in the sense that they support large challenge and/or response spaces. Since most of the known PUF instantiations do not fulfil these properties, LR-PUFs seem to be better suited. Another application of LR-PUFs are electronic fare systems for public transport as suggested in [8]. In this setting, an access token is equipped with an LR-PUF, that serves to authenticate the passenger at the entry points to the transport system and can be used to secure a credit stored on the token. The reconfiguration capability of the device enables the easy reuse of tokens, as reconfiguration of the LR-PUF is (ideally) equivalent to physically replacing the device without causing the cost of a new device.

Another interesting application of LR-PUF technology is presented in [5]. Here, the LR-PUF is used to provide secure key storage and helps to prevent cloning and downgrading of embedded software authenticated using the stored keys. In this application, software is bound to an embedded device by encrypting it with a device-specific key. This key is generated by querying a PUF that is

part of the device, making the key dependent on the unique properties of the PUF in each device. This prevents cloning of the software to a new device, as the key on a cloned device will be generated differently. The reconfigurability is used in the event of a software update, to prevent downgrading to an older version of the software. In this event, a new key to encrypt the updated software is derived, and the old software version will no longer be useful, because the old key can no longer be retrieved from the LR-PUF.

### 1.3 Contribution

We revisit the LR-PUFs as introduced in [8] and present several ways to charaterize the unpredictability notion. We reconsider existing constructions with respect to these new measures, and we propose a novel construction that is secure w.r.t. our strongest notion of unpredictability. In the following, we discuss each contribution more in detail.

*Definitions* We introduce four different notions of unpredictability. The first one is called **Plain**-unpredictability and it is a natural extension of backward/forward-unpredictability of [8]. The basic idea of this definition is to allow the adversary to reconfigure the PUF several times. The second notion, called **St-Access**-unpredictability, removes the assumption that the state is stored in a tamper-evident manner and allows the adversary to directly write the state. The third notion, called **PUF-Access**-unpredictability, models the case where the adversary manages to bypass the query and reconfiguration logic and where it gains direct access to the PUF. The fourth notion, called **Full-Access**-unpredictability, combines **PUF-Access**-unpredictability and **St-Access** unpredictability in the sense that the adversary has direct access to the PUF and is allowed to set the state maliciously. Perhaps surprisingly there is an obstacle when trying to compare the power of state-setting adversaries to PUF-access adversaries. The issue is that a PUF-access adversary might be able to completely precompute the behavior of an LR-PUF given the current state, which makes both notions incomparable. A visual representation of these relations is given in Figure 2, where an arrow $\mathsf{A} \rightarrow \mathsf{B}$ denotes, that notion $\mathsf{A}$ implies notion $\mathsf{B}$ and an arrow $\mathsf{A} \not\rightarrow \mathsf{B}$ means that notion $\mathsf{A}$ does not imply notion $\mathsf{B}$.

*Analysis* In Appendix D we give a comprehensive security analysis of the "speed-optimized" and the "area-optimized-construction" from [8] w.r.t. our unpredictability notions. The former employs a collision resistant hash function both to combine state information and query and to generate a new state frome the old one. The latter uses an identical reconfiguration algorithm, but is geared towards PUFs with small area, i.e., small input range by providing a query mechanism that involves iteratively constructing a response from smaller subqueries. (For full definitions, please refer to Figure 7). Our analysis shows that both constructions are **St-Access**-unpredictable. Previously, it was only known that both constructions are backward (resp. forward) unpredictable. The practical consequences of this result is that the scheme remains secure, even if the state is *not* stored in a

tamper-evident manner. On the negative side we show that both constructions are not secure against adversaries that have direct access to the PUF. In fact, our result here is more general, showing that any LR-PUF cannot satisfy this notion where access to the underlying PUF makes the query and reconfiguration algorithms completely predictable to the adversary.

*Construction* We propose a simple LR-PUF construction that is **Full-Access**-unpredictable. Our scheme can be seen as a randomized variant of the "speed-optimized" construction from [8] with the difference being that our reconfiguration algorithm samples a fresh state $\mathsf{st}$ upon reconfiguration and it evaluates the underlying PUF on $w \leftarrow \mathsf{Hash}(\mathsf{st} \mid\mid c)$. This construction relies on the same computational assumptions as the scheme of [8], it is almost as efficient, but it satisfies both **Full-Access**- and **St-Access**-unpredictability.

### 1.4 Outline

In Section 2 we give some background by reviewing a formalization of Physically Unclonable Functions and present our formalization of LR-PUFs. Section 3 introduces the new unpredictability notions we propose and the relations among them. Section 4 contains the specification of a construction which achieves the strongest of our unpredictability notions.

## 2 Logically Reconfigurable PUFs

### 2.1 Notations

In this paper, we will consider only PPT algorithms, some of which will have access to a PUF (see section B for a short discussion of the implications of PUF access on computational hardness assumptions). Let $\mathcal{A}$ be such an algorithm. We denote with $y \leftarrow_{\$} \mathcal{A}(x)$ that $y$ is the output that was (possibly probabilistically) computed by $\mathcal{A}$ on input $x$. If $\mathcal{A}$ is a deterministic algorithm, we write $y \leftarrow \mathcal{A}(x)$ to mean that $y$ is the deterministic output of $\mathcal{A}$ on input $x$. The core of our definitions is formed by experiments and if $\mathsf{EXAMPLE}$ is such an experiment we write "$\mathsf{EXAMPLE} = x$" to mean that upon finishing, the experiment $\mathsf{EXAMPLE}$ outputs $x$. For two bit strings $x, y \in \{0,1\}^n$ let further $\mathsf{dis}(x, y)$ denote the Hamming distance of $x$ and $y$.

### 2.2 Physically Unclonable Functions

A Physically Unclonable Function (PUF) is a noisy function that is realized through a physical object [12]. The PUF can be queried with a challenge $c$ and answers with a response $r$. The output of the PUF is noisy meaning that querying the PUF twice with the same challenge yields most likely two different but closely related responses. In the following we recall the definition of PUFs and their main security property given in [4].

**Definition 1 (Physically Unclonable Functions).** *Let $\rho$ be the dimension of the range of the PUF responses of the PUF family, and let $d_{noise}$ be a bound on the PUF's noise. A pair $\mathcal{P} = (\mathsf{S}, \mathsf{E})$ is a family of $(\rho, d_{noise})$-PUFs if it satisfies the following properties:*

**Index Sampling.** *Let $\mathcal{I}_\lambda$ be an index set. The sampling algorithm $\mathsf{S}$ outputs, on input the security parameter $1^\lambda$, an index $\mathsf{id} \in \mathcal{I}_\lambda$. We do not require that the index sampling can be done efficiently. Each index $\mathsf{id} \in \mathcal{I}_\lambda$ corresponds to a set $\mathcal{D}_{\mathsf{id}}$ of distributions. For each challenge $c \in \{0,1\}^\lambda$, $\mathcal{D}_{\mathsf{id}}$ contains a distribution $\mathcal{D}_{\mathsf{id}}(c)$ on $\{0,1\}^{\rho(c)}$. We do* not *require that $\mathcal{D}_{\mathsf{id}}$ has a short description or an efficient sampling algorithm.*

**Evaluation.** *The evaluation algorithm $\mathsf{E}$ gets as input a tuple $(1^\lambda, \mathsf{id}, c)$, where $c \in \{0,1\}^\lambda$. It outputs a response $r \in \{0,1\}^{\rho(\lambda)}$ according to distribution $\mathcal{D}_{\mathsf{id}}$. It is* not *required that $\mathsf{E}$ is a PPT algorithm.*

**Bounded Noise.** *For all indices $\mathsf{id} \in \mathcal{I}$, for all challenges $c \in \{0,1\}^\lambda$, we have that when running $\mathsf{E}(1^\lambda, \mathsf{id}, c)$ twice, then for any two outputs $r_1, r_2$ that are produced the Hamming distance $\mathsf{dis}(r_1, r_2)$ is smaller than $d_{noise}(\lambda)$.*

**Unpredictability of PUFs** Loosely speaking, a PUF is unpredictable if it is difficult to predict the response of the PUF to a given, previously unknown challenge. This intuition is formalized in an experiment where the adversary can adaptively query the PUF on challenges of its choice and wins if it can predict the response to a fresh challenge of its choice, within the bound $d_{noise}$. Fresh means that the adversary did not query the PUF on this challenge.

**Definition 2 (PUF-Unpredictability).** *A family of PUFs $\mathcal{P} = (\mathsf{S}, \mathsf{E})$ is* unpredictable *if for any PPT algorithm $\mathcal{A}$ the probability that the experiment $\mathsf{PRE}^{\mathcal{P}}_{\mathcal{A}}(\lambda)$ evaluates to 1 is negligible (in the security parameter $\lambda$), where*

***Experiment*** $\mathsf{PRE}^{\mathcal{P}}_{\mathcal{A}}(\lambda)$
$\mathsf{id} \leftarrow \mathsf{S}(1^\lambda)$
$(c^*, r^*) \leftarrow \mathcal{A}^{\mathsf{E}}(\mathsf{id})$
$r \leftarrow \mathsf{E}(c^*)$
*Return 1 iff* $\mathsf{dis}(r, r^*) \leq d_{noise}$ *and $c^*$ is fresh.*

For the sake of simplicity we use this game based definition of unpredictability. A formulation with respect to entropy contained in the PUF responses is given in [4]. A comprehensive and more in-depth game-based formulation of PUF properties is found in [2].

### 2.3 Definition of Logically Reconfigurable PUFs

In practice, many PUF instances have only a restricted challenge and response space, such that after a certain number of queries they cannot be used anymore. The basic idea of Logically Reconfigurable PUF (LR-PUFs) is to extend the PUF by a control logic that allows to change the challenge and response behavior of the system. Our definition is similar to the one of [8].

**Definition 3 (Logically Reconfigurable PUFs).** *Let $\mathcal{P} = (\mathsf{S}, \mathsf{E})$ be a family of $(\rho, d_{noise})$-PUFs. A logically reconfigurable PUF (LR-PUF) with black-box access to $\mathcal{P}$ is a tuple of efficient algorithms $\mathcal{L} = (\mathsf{Setup}^{\mathsf{S},\mathsf{E}}, \mathsf{Query}^{\mathsf{E}}, \mathsf{Rcnf}^{\mathsf{E}})$, which satisfies the following properties*

**Setup.** *The* $\mathsf{Setup}$ *algorithm takes as input the security parameter $1^\lambda$. It outputs an index $\mathsf{id} \in \mathcal{I}$, determining the underlying PUF from $\mathcal{P}$ and an initial state $\mathsf{st} \in \{0,1\}^{\ell(\lambda)}$ of the LR-PUF. We require that $\ell(\lambda) \geq \lambda$.*

**Query mechanism.** *The $\mathsf{Query}_{\mathsf{st}}$ algorithm takes as input a challenge $c \in \{0,1\}^\lambda$ and outputs a response $r \in \rho(\lambda)$.*

**Reconfiguration.** *The $\mathsf{Rcnf}_{\mathsf{st}}$ algorithm updates the state of the LR-PUF to a new state $\mathsf{st}' \in \{0,1\}^{\ell(\lambda)}$ which is (possibly probabilistically) computed from the old state $\mathsf{st}$. The new state is also output.*

*The three algorithms may interact with the underlying PUF family via the oracles $\mathsf{E}$ and $\mathsf{S}$. We will often omit giving the oracle access explicitly. Additionally, we assume that the noise of the LR-PUF responses is bounded in the same way as the noise of the underlying PUF's responses.*

*Remark 1.* The setup algorithm of almost all constructions in this paper is the same and consists of the following steps. $\mathsf{Setup}(1^\lambda)$ generates the underlying PUF $\mathsf{id} \leftarrow_\$ \mathsf{S}(1^\lambda)$, chooses a string $\mathsf{st} \leftarrow_\$ \{0,1\}^{\ell(\lambda)}$ uniformly at random, and outputs $(\mathsf{id}, \mathsf{st})$. In the following, unless stated otherwise, all constructions will use this standard setup algorithm.

**Unpredictability of LR-PUFs** Ideally, an LR-PUF in one specific state should be as unpredictable as its underlying physical PUF, so the internal state of the LR-PUF can be seen as a mapping from LR-PUF queries to PUF queries that is ideally a permutation. Reconfiguration then constitutes a "shuffling" of this mapping, such that a completely new permutation is reached. To formalize this, the authors of [8] propose two complimentary notions of unpredictability:

**Forward-unpredictability:** The reconfiguration changes the mapping in such a way, that knowledge about the previous state does not enable an adversary to predict the challenge-response behavior for the reconfigured LR-PUF.

**Backward-unpredictability:** The reconfiguration reveals no additional information about the old internal state, i.e., after reconfiguration an adversary should not be able to predict the challenge-response behavior for the old state.

We provide a formal characterization of these properties as derivatives of our plain unpredictability notion (see Definition 4).

## 3 New Notions of Unpredictability

In this section we extend the original unpredictability notion by considering strengthened adversaries. We show how the new notions relate to each other and in which scenarios their consideration might be beneficial.

### 3.1 Multiple Reconfigurations

In [8], the unpredictability experiments revolve around a single reconfiguration process. However, an adversary might witness several reconfigurations and thereby deduce some information about the influence of the state on the LR-PUFs behavior. This motivates our first unpredictability definition, which is an extension of the backward/forward-unpredictability properties to multiple reconfigurations of the LR-PUF. To this end, we provide the adversary access to a reconfiguration oracle, which invokes the reconfiguration algorithm.

Let $\mathsf{Rcnf}$ denote the reconfiguration oracle for an LR-PUF $\mathcal{L}$ with current state $\mathsf{st}$. The oracle $\mathsf{Rcnf}$ accepts two kinds of inputs: $\perp$, upon which $\mathsf{Rcnf}_{\mathsf{st}}$ is invoked, and $\mathsf{st}'$ upon which the internal state of $\mathcal{L}$ is set to $\mathsf{st}'$. The latter input functionality allows the adversary to program the state and is only available to state-setting adversaries (see Appendix 3.2). Let $\mathcal{S}$ denote the list of states the adversary obtains over the course of an experiment, be it through $\mathsf{Setup}$ or the oracle $\mathsf{Rcnf}$. Further, let $\mathsf{Query}$ denote the query oracle, which takes as input a state $\mathsf{st}$ and a challenge $c$ and returns $\mathsf{Query}_{\mathsf{st}}(c)$ to the adversary. The adversary can only invoke the query oracle with states stored in $\mathcal{S}$.

**Definition 4 (Plain-Unpredictability).** *An LR-PUF* $\mathcal{L} = (\mathsf{Setup}, \mathsf{Query}_{\mathsf{st}}, \mathsf{Rcnf}_{\mathsf{st}})$ *is* unpredictable *if for any PPT adversary* $\mathcal{A}$ *the probability that the experiment* $\mathsf{PLAIN}^{\mathcal{L}}_{\mathcal{A}}(\lambda)$ *evaluates to 1 is negligible (in the security parameter* $\lambda$*), where*

***Experiment*** $\mathsf{PLAIN}^{\mathcal{L}}_{\mathcal{A}}(\lambda)$
$(\mathsf{id}, \mathsf{st}) \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{st}^*, c^*, r^*) \leftarrow \mathcal{A}^{\mathsf{Query}, \mathsf{Rcnf}(\perp)}(1^\lambda, \mathsf{st})$
$r \leftarrow \mathsf{Query}_{\mathsf{st}^*}(c^*)$
*Output 1 iff* $\mathsf{dis}(r, r^*) \leq d_{noise}$ *and* $c^*$ *was not previously queried to* $\mathsf{Query}(\mathsf{st}^*, \cdot)$ *and* $\mathsf{st}^* \in \mathcal{S}$.

*Remark 2.* We can obtain the backward- and forward-unpredictability notions described in [8] by considering restricted adversaries that invoke $\mathsf{Rcnf}(\perp)$ only once, setting and obtaining the new state $\mathsf{st}'$, in the following only query $\mathsf{Query}(\mathsf{st}', \cdot)$.

**Corollary 1.** *Let* $\mathcal{L}$ *be a* **Plain***-unpredictable LR-PUF, then* $\mathcal{L}$ *is also backward- and forward-unpredictable.*

**Separation of Plain- and Backward/Forward-Unpredictability** In what follows we show that **Plain**-unpredictability is strictly stronger than the previous notions. Let $\mathbf{Backward}^{\mathcal{L}}_{\mathcal{A}}(\lambda)$ (resp. $\mathbf{Forward}^{\mathcal{L}}_{\mathcal{A}}(\lambda)$) denote $\Pr\left[\mathsf{PLAIN}^{\mathcal{L}}_{\mathcal{A}}(\lambda) = 1\right]$ where $\mathcal{A}$ is a backward-unpredictability adversary (resp. forward-unpredictability adversary) as described above. We separate the security notions with the following two propositions.

**Proposition 1.** *If collision-resistant hash functions relative to a PUF exist (cf. Appendix B), then there exist backward-unpredictable LR-PUFs, which are not* **Plain***-unpredictable.*

The basic idea of our counterexample is to let the adversary learn a prediction by calling the reconfiguration oracle. This prediction, however, only helps him in combination with the evaluation oracle, that outputs the evaluation on a point *different* from the challenge, if the query contains a specific prediction. More precisely, we store a pair $(u, v)$ in the state. Then we modify the query algorithm, whose input is a challenge $c$, such that it evaluates the PUF on $\left(1^\lambda \oplus c \| \mathsf{st}'\right)$ if $c = \mathsf{E}(u)$. Clearly, in our construction the attacker can never invoke the query oracle on $u$ and thus, cannot exploit this part directly. However, whenever the attacker queries the reconfiguration oracle, it obtains this answer through the state.

Proof. Let $\mathcal{P} = (\mathsf{S}, \mathsf{E})$ be an unpredictable PUF in the sense of Definition 2, i.e., for any PPT adversary $\mathcal{A}$ there is a negligible function $\varepsilon'(\lambda)$ with $\Pr[\mathsf{PRE}^{\mathcal{P}}_{\mathcal{A}}(\lambda) = 1] = \varepsilon'(\lambda)$. Using $\mathcal{P}$ we construct a new LR-PUF $\mathcal{L} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Rcnf})$, where the algorithms are defined in Figure 3.

$\mathsf{Setup}(1^\lambda)$
- $\mathsf{id} \leftarrow \mathsf{S}(1^\lambda)$
- $\mathsf{st}' \leftarrow_\$ \{0,1\}^{\ell(\lambda)}$
- $u \leftarrow_\$ \{0,1\}^\lambda$
- $v \leftarrow_\$ \{0,1\}^{\rho(\lambda)}$
- **Return** $(\mathsf{id}, u \,||\, v \,||\, \mathsf{st}')$

$\mathsf{Query}_{\mathsf{st}}(c)$
- Parse $\mathsf{st}$ as $u \,||\, v \,||\, \mathsf{st}'$
- **IF** $\mathsf{dis}(c, \mathsf{E}(u)) \leq d_{noise}$
  - $w \leftarrow \mathsf{Hash}(1^\lambda \oplus c \,||\, \mathsf{st}')$
  - **Return** $\mathsf{E}(w)$
- **End IF**
- $w \leftarrow \mathsf{Hash}(c \,||\, \mathsf{st}')$
- **Return** $\mathsf{E}(w)$

$\mathsf{Rcnf}_{\mathsf{st}}$
- Parse $\mathsf{st}$ as $u \,||\, v \,||\, \mathsf{st}'$
- $r \leftarrow_\$ \{0,1\}^\lambda$
- **Return** $r \,||\, \mathsf{E}(u) \,||\, \mathsf{Hash}(\mathsf{st}')$

Fig. 3 This construction is backward-unpredictable, but not **Plain**-unpredictable.

First, we show that $\mathcal{L}$ retains backward-unpredictability. Suppose that $\mathcal{A}$ is an adversary against its backward-unpredictability. Recall that $\mathcal{A}$ receives the initial state $\mathsf{st}_0 = (u_0 \,||\, v_0 \,||\, \mathsf{st}'_0)$, and it may invoke the query oracle adaptively on that state. In the next step the PUF is reconfigured, after which the adversary can again ask the query oracle on challenges of its choice. The new state is $\mathsf{st}_1 = (u_1 \,||\, v_1 \,||\, \mathsf{st}'_1)$. Eventually, the adversary outputs its prediction $(c^*, r^*)$ for the initial state $\mathsf{st}_0$, succeeding if the prediction is valid and for a fresh challenge $c^*$.

Let $\mathbf{Pre}^{\mathcal{P}}_{\mathcal{A}}(\lambda) := \Pr[\mathsf{PRE}^{\mathcal{P}}_{\mathcal{A}}(\lambda) = 1]$ and let $\mathsf{bad}$ denote the event, that the adversary in the first query phase issues a query $\hat{c}$ with $\hat{c} = \mathsf{E}(u_0)$. In this event, the adversary will receive a response for a challenge it did not issue, namely $\hat{r} = \mathsf{Query}_{\mathsf{st}_0}\left(1^\lambda \oplus \hat{c}\right)$. Upon reconfiguration, it will be revealed to the adversary that

$$\mathsf{dis}(\hat{c} = v_1, \mathsf{E}(u_0)) \leq d_{noise},$$

giving it the opportunity to output $(\mathsf{st}_0, 1^\lambda \oplus \hat{c}, \hat{r})$ as a succeeding prediction. Because $u_0$ is drawn at random from the challenge space of the underlying PUF and the adversary does not have access to the PUF, it holds that $\Pr[\mathsf{bad}] \leq \mathbf{Pre}^{\mathcal{P}}_{\mathcal{A}}(\lambda) = \varepsilon'(\lambda)$.

Consider the case that the event $\mathsf{bad}$ does not happen. In this case, the construction's $\mathsf{Query}$ algorithm behaves identical to the $\mathsf{Query}$ algorithm of the speed optimized construction $\mathsf{speed}$ from [8]. Here we can use that $\mathsf{speed}$ was shown to be **Backward**-unpredictable in [8]. Let $\mathbf{Backward}^{\mathsf{speed}}_{\mathcal{A}}$ be the probability that $\mathsf{PLAIN}^{\mathsf{speed}}_{\mathcal{A}}(\lambda) = 1$.

**Lemma 1 (Proposition 1, [8]).** *If the underlying PUF $\mathcal{P}$ is unpredictable to adversaries that issue polynomially many queries, then the* $\mathsf{speed}$ *construction is* **Backward***-unpredictable to these adversaries.*

The difference in the reconfiguration algorithm does not harm this claim, as the same proof works with the minor modification that the reduction against the unpredictability of $\mathcal{P}$ must issue one additional query to it's PUF oracle during the reconfiguration phase. Using Lemma 1 it follows:

$$\begin{aligned}
&\left|\mathbf{Backward}^{\mathcal{L}}_{\mathcal{A}}(\lambda) - \mathbf{Backward}^{\mathsf{speed}}_{\mathcal{A}}(\lambda)\right| = \left|\Pr\left[\mathsf{PLAIN}^{\mathcal{L}}_{\mathcal{A}}(\lambda) = 1 | \mathsf{bad}\right] \Pr\left[\mathsf{bad}\right]\right. \\
&\left. + \Pr\left[\mathsf{PLAIN}^{\mathcal{L}}_{\mathcal{A}}(\lambda) = 1 | \neg\mathsf{bad}\right] (1 - \Pr\left[\mathsf{bad}\right]) - \Pr\left[\mathsf{PLAIN}^{\mathsf{speed}}_{\mathcal{A}}(\lambda) = 1\right]\right| \\
&= \Pr\left[\mathsf{bad}\right] + \Pr\left[\mathsf{bad}\right] \mathbf{Backward}^{\mathsf{speed}}_{\mathcal{A}}(\lambda) \leq \mathbf{Pre}^{\mathcal{P}}_{\mathcal{A}}(\lambda) + \mathbf{Pre}^{\mathcal{P}}_{\mathcal{A}}(\lambda)\mathbf{Backward}^{\mathsf{speed}}_{\mathcal{A}}(\lambda) \\
&\leq \mathsf{negl}(\lambda).
\end{aligned}$$

As mentioned above $\mathcal{L}$ leaks information about the challenge-response-behavior of the previous state upon reconfiguration, thus enabling an attacker to make a valid prediction if it is able to query the old state again. Specifically we construct an adversary $\mathcal{A}$ which makes 1 query to $\mathsf{Query}$, invokes $\mathsf{Rcnf}$ once and for which $\Pr[\mathsf{PLAIN}_{\mathcal{A}}(\lambda) = 1] = 1$. Upon receiving the security parameter and the initial state $\mathsf{st}_0 = u_0 \,||\, v_0 \,||\, \mathsf{st}'_0$, $\mathcal{A}$ immediately invokes $\mathsf{Rcnf}$ receiving $\mathsf{st}_1 = r \,||\, \mathsf{E}(u_0) \,||\, \mathsf{Hash}(\mathsf{st}'_0)$. Given $c' := \mathsf{E}(u_0)$, the adversary can call $\mathsf{Query}(\mathsf{st}_0, c')$ to receive $r^* = \mathsf{Query}_{\mathsf{st}_0}(1^\lambda \oplus c')$.

$\mathcal{A}$ then returns $(\mathsf{st}_0, 1^\lambda \oplus c', r^*)$. This will be a valid output, since $\mathcal{A}$ has never queried $\mathsf{Query}(\mathsf{st}_0, \cdot)$ on $1^\lambda \oplus c'$. □

**Proposition 2.** *If collision-resistant hash functions relative to a PUF (cf. Appendix B) exist, then there exist forward-unpredictable LR-PUFs, which are not* **Plain***-unpredictable.*

The proof is analogous to the one of Proposition 1, as the same construction describe there is also forward-unpredictable and is thus omitted.

### 3.2 State-setting Adversaries

The authors of [8] assume that the state is stored in a tamper-evident manner and therefore an attacker cannot set the state of the LR-PUF to arbitrary values. We believe that there are many plausible scenarios where tamper-evident storage of the state is too expensive and where the adversary might be able to change

the state, even though the internal physical PUF is tamper-evident. Therefore, we propose the following unpredictability notion, in which an adversary can set the internal state of the LR-PUF. As mentioned above, this is formalized via the type of inputs to the reconfiguration oracle $\mathsf{Rcnf}$ which are arbitrary states that are to be set as the new state of the LR-PUF.

**Definition 5 (St-Access-Unpredictability).** *An LR-PUF* $\mathcal{L} = (\mathsf{Setup}, \mathsf{Query}_{\mathsf{st}}, \mathsf{Rcnf}_{\mathsf{st}})$ *is* unpredictable for a state-setting adversary *if for any PPT adversary* $\mathcal{A}$ *the probability that the experiment* $\mathsf{ST\text{-}ACCESS}^{\mathcal{L}}_{\mathcal{A}}(\lambda)$ *evaluates to 1 is negligible (in the security parameter* $\lambda$*), where*

***Experiment*** $\mathsf{ST\text{-}ACCESS}^{\mathcal{L}}_{\mathcal{A}}(\lambda)$
$(\mathsf{id}, \mathsf{st}) \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{st}^*, c^*, r^*) \leftarrow \mathcal{A}^{\mathsf{Query},\mathsf{Rcnf}}(1^\lambda, \mathsf{st})$
$r \leftarrow \mathsf{Query}_{\mathsf{st}^*}(c^*)$
*Output 1 iff* $\mathsf{dis}(r, r^*) \leq d_{noise}$ *and* $c^*$ *was not previously queried to* $\mathsf{Query}(\mathsf{st}^*, \cdot)$ *and* $\mathsf{st}^* \in \mathcal{S}$.

The state-setting adversary can be thought of as bypassing the reconfiguration algorithm, thus security against state-setting adversaries should be considered a property of the $\mathsf{Query}$ mechanism.

*Remark 3.* It is easy to see that an LR-PUF construction satisfying this notion of unpredictability must also be **Plain**-unpredictable (Definition 4). Any adversary against **Plain**-unpredictability is also a valid adversary in the $\mathsf{ST\text{-}ACCESS}$-unpredictability experiment, which simply does not invoke the reconfiguration oracle on an input other than $\perp$.

**Corollary 2.** *Let* $\mathcal{L} = (\mathsf{Setup}, \mathsf{Query}_{\mathsf{st}}, \mathsf{Rcnf}_{\mathsf{st}})$ *be* **St-Access**-*unpredictable. Then* $\mathcal{L}$ *is also* **Plain**-*unpredictable.*

The inverse relationship does, however, not hold.

**Proposition 3.** *There exist* **Plain**-*unpredictable LR-PUF constructions, which are not* **St-Access**-*unpredictable.*

Proof. To show the separation, we consider a construction which has a "vulnerable" state, i.e., a state which does not support a secure reconfiguration. A state-setting adversary can then prepare the LR-PUF to have that state and get an advantage through the defective reconfiguration algorithm. An adversary without state-setting capabilities, however, would have to wait for that state to occur in a chain of honest reconfigurations to get any advantage, as long as the reconfiguration is working correctly for any other state.

Let $\mathcal{L}' = (\mathsf{Setup}', \mathsf{Query}'_{\mathsf{st}}, \mathsf{Rcnf}'_{\mathsf{st}})$ be a **Plain**-unpredictable LR-PUF. Using $\mathcal{L}'$ we define $\mathcal{L} := (\mathsf{Setup}, \mathsf{Query}'_{\mathsf{st}}, \mathsf{Rcnf}_{\mathsf{st}})$, where the algorithms are defined in Figure 4.

Let $\mathsf{bad}$ denote the event that the state of the LR-PUF will become $(\mathsf{st}' \,||\, r)$ with $\mathsf{st}' = r$ in an experiment where $\mathsf{Rcnf}$ is queried at most $q_r$ times. In this case, the reconfiguration algorithm reveals a possibly unqueried LR-PUF response.

$\mathsf{Setup}(1^\lambda)$
$(\mathsf{id}, \mathsf{st}) \leftarrow \mathsf{Setup}'(1^\lambda)$
$\mathsf{st}' \leftarrow_\$ \{0,1\}^{\rho(\lambda)}$
**Return** $(\mathsf{id}, \mathsf{st}' \,||\, 0^{\rho(\lambda)})$

$\mathsf{Rcnf}_{\mathsf{st}' \,||\, r}$
**IF** $\mathsf{st}' = r$ **THEN**
$\mathsf{st}' \leftarrow_\$ \{0,1\}^{\rho(\lambda)}$
$r \leftarrow \mathsf{Query}'_{\mathsf{st}'}(0^\lambda)$
**ELSE**
$\mathsf{st}' \leftarrow_\$ \{0,1\}^{\rho(\lambda)}$
$r \leftarrow_\$ \{0,1\}^{\rho(\lambda)}$
**Return** $\mathsf{st}' \,||\, r$

Fig. 4 This construction is **Plain**-unpredictable, but not **St-Access**-unpredictable.

Let $\mathsf{st}_0, \ldots, \mathsf{st}_{q_r-1}$ be the states which are generated in the experiment. Then, we calculate the probability that this event happens as

$$\begin{aligned}\Pr[\mathsf{bad}] &= 1 - \Pr\left[\forall i \in \{0,\ldots,q_r-1\} : \mathsf{st}_i \neq 0^{\ell(\lambda)}; \mathsf{st}_i \leftarrow_\$ \{0,1\}^{\ell(\lambda)}\right] \\ &= 1 - \prod_{i=0}^{q_r-1} 1 - \frac{1}{2^{\ell(\lambda)}} = 1 - \left(1 - \frac{1}{2^{\ell(\lambda)}}\right)^{q_r} \leq 1 - \left(1 - \frac{1}{2^\lambda}\right)^{q_r} \\ &\leq 1 - \left(1 - \frac{q_r}{2^\lambda}\right) = \frac{q_r}{2^\lambda}\end{aligned}$$

Let $\mathcal{A}$ be an adversary against the **Plain**-unpredictability of $\mathcal{L}$. Unless the event $\mathsf{bad}$ occurs, $\mathcal{A}$ is playing in the **Plain**-unpredictability experiment of $\mathcal{L}'$. This means, unless $\mathsf{bad}$ occurs, $\mathcal{L}$ is as **Plain**-unpredictable as $\mathcal{L}$.

Because $\mathcal{A}$ is a PPT algorithm, there is a polynomial bound on the number of reconfiguration queries $\mathcal{A}$ can make. Thus the probability of the event $\mathsf{bad}$ is negligible in $\lambda$. This means:

$$\begin{aligned}&\left|\mathbf{Plain}^{\mathcal{L}'}_{\mathcal{A}}(\lambda) - \mathbf{Plain}^{\mathcal{L}}_{\mathcal{A}}(\lambda)\right| = \left|\Pr[\mathsf{bad}] + (1 - \Pr[\mathsf{bad}])\mathbf{Plain}^{\mathcal{L}}_{\mathcal{A}}(\lambda) - \mathbf{Plain}^{\mathcal{L}}_{\mathcal{A}}(\lambda)\right| \\ &= \left|\Pr[\mathsf{bad}] - \Pr[\mathsf{bad}]\mathbf{Plain}^{\mathcal{L}}_{\mathcal{A}}(\lambda)\right| \leq \Pr[\mathsf{bad}] \leq \frac{q_r}{2^\lambda} \leq \mathsf{negl}(\lambda).\end{aligned}$$

Because we have assumed $\mathcal{L}'$ to be **Plain**-unpredictable, this shows that $\mathcal{L}$ is **Plain**-unpredictable.

The construction is not ST-ACCESS-unpredictable, since the adversary may set the LR-PUF state to $0^\lambda$ and by invoking $\mathsf{Rcnf}(\perp)$ subsequently, learn a new state $\mathsf{st}'$ and the response $r$ for the query $0^\lambda$ in that state. □

### 3.3 Direct Access Adversaries

Another assumption made in [8] is that the attacker cannot bypass the Query mechanism and thus, does not have direct access to the embedded PUF. In

the real world, however, it might be that case that the attacker finds a way to stimulate the physical PUF directly, circumventing the control logic of the LR-PUF. In what follows, we remove this assumption by giving the adversary direct access to the embedded PUF as well.

**Definition 6 (PUF-Access-Unpredictability).** *An LR-PUF* $\mathcal{L} = (\mathsf{Setup}, \mathsf{Query}_{\mathsf{st}}, \mathsf{Rcnf}_{\mathsf{st}})$ *is* unpredictable for an adversary with direct PUF access *if for any PPT adversary* $\mathcal{A}$ *the probability that the experiment* $\mathsf{PUF\text{-}ACCESS}^{\mathcal{L}}_{\mathcal{A}}(\lambda)$ *evaluates to 1 is negligible (in the security parameter* $\lambda$*), where*

***Experiment*** $\mathsf{PUF\text{-}ACCESS}^{\mathcal{L}}_{\mathcal{A}}(\lambda)$
$(\mathsf{id}, \mathsf{st}) \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{st}^*, c^*, r^*) \leftarrow \mathcal{A}^{\mathsf{Rcnf}(\bot),\mathsf{E}}(1^\lambda, \mathsf{st})$
*Set LR-PUF state to* $\mathsf{st}^*$ *using* $\mathsf{Rcnf}(\mathsf{st}^*)$
$\mathsf{st}' \leftarrow \mathsf{Rcnf}_{\mathsf{st}^*}$
$r \leftarrow \mathsf{Query}_{\mathsf{st}'}(c^*)$
*Output 1 iff* $\mathsf{dis}(r, r^*) \leq d_{noise}$.

Because an LR-PUF construction might rely solely upon the PUF itself to perform reconfiguration and querying, an adversary that has access to the PUF may be able, given the current state of the PUF, to compute challenge-response pairs for all the following states the LR-PUF will have.

**Proposition 4.** *If collision-resistant hash functions relative to a PUF exist, then there exists a* **Plain***-unpredictable LR-PUF construction, which is not* **PUF-Access***-unpredictable.*

Proof. Consider the speed-optimized construction given in [8]. We will later show, that this construction is **St-Access**-unpredictable (cf. Proposition 9). Because **St-Access**-unpredictability implies **Plain**-unpredictability, the construction is also **Plain**-unpredictable. The construction is not PUF-ACCESS-unpredictable, since an attacker $\mathcal{A}$ which has access to a PUF evaluation oracle $\mathsf{E}$ can simulate both Query and Rcnf, thus enabling the creation of valid challenge-response pairs, while bypassing the Query and Rcnf oracles. □

Perhaps surprisingly there is an obstacle when trying to compare the power of state-setting adversaries to PUF-access adversaries. As described above, a PUF-access adversary might be able to completely precompute the behavior of an LR-PUF given the current state. Thus the definition of **PUF-Access**-unpredictability demands the adversary predict a challenge response pair not for the state, which it finally outputs, but for the state which results from the reconfiguration based on that state. This excludes bypassing the Rcnf oracle and enables the definition to capture the unpredictability gain provided by the Rcnf algorithm.

**Proposition 5.** *Unpredictability against state-setting adversaries is not comparable to unpredictability against PUF-access adversaries, i.e.,*

*i) There exists an LR-PUF which is* **PUF-Access***-unpredictable but not* **St-Access***-unpredictable.*
*ii) If collision-resistant hash functions w.r.t. PUFs exist, there exist LR-PUFs, which are* **St-Access***-unpredictable but not* **PUF-Access***-unpredictable.*

The proof of this proposition can be found in Appendix A.

### 3.4 Full Access Adversaries

A combination of the previous scenarios provides the PUF access adversary with the possibility to set the internal state. This is intuitively the strongest notion, as it provides the adversary with essentially complete control over the LR-PUF during the query phase of the experiment.

**Definition 7 (Full-Access-Unpredictability).** *An LR-PUF* $\mathcal{L} = (\mathsf{Setup}, \mathsf{Query}_{\mathsf{st}}, \mathsf{Rcnf}_{\mathsf{st}})$ *is* unpredictable for a state-setting adversary with PUF access *if for any PPT adversary* $\mathcal{A}$ *the probability that the experiment* $\mathsf{FULL\text{-}ACCESS}^{\mathcal{L}}_{\mathcal{A}}(\lambda)$ *evaluates to 1 is negligible (in the security parameter* $\lambda$*), where*

***Experiment*** $\mathsf{FULL\text{-}ACCESS}^{\mathcal{L}}_{\mathcal{A}}(\lambda)$
$(\mathsf{id}, \mathsf{st}) \leftarrow \mathsf{Setup}^{\mathsf{S},\mathsf{E}}(1^\lambda)$
$(\mathsf{st}^*, c^*, r^*) \leftarrow \mathcal{A}^{\mathsf{Rcnf},\mathsf{E}}(1^\lambda, \mathsf{st})$
*Set LR-PUF state to* $\mathsf{st}^*$ *using* $\mathsf{Rcnf}(\mathsf{st}^*)$
$\mathsf{st}' \leftarrow \mathsf{Rcnf}_{\mathsf{st}^*}$
$r \leftarrow \mathsf{Query}_{\mathsf{st}'}(c^*)$
*Output 1 iff* $\mathsf{dis}(r, r^*) \leq d_{noise}$.

*Remark 4.* As **Full-Access**-unpredictability is an immediate extension of **PUF-Access**-unpredictability, it is easy to see the following corollary is true.

**Corollary 3.** *Let* $\mathcal{L}$ *be a* **Full-Access***-unpredictable LR-PUF. Then* $\mathcal{L}$ *is also* **PUF-Access***-unpredictable.*

However, the **PUF-Access**-unpredictability adversary is strictly weaker than the state-setting **Full-Access**-unpredictability adversary.

**Proposition 6.** *There are LR-PUF constructions which are* **PUF-Access***-unpredictable, but not* **Full-Access***-unpredictable.*

Proof. Let $\mathcal{L}' = (\mathsf{Setup}', \mathsf{Query}'_{\mathsf{st}}, \mathsf{Rcnf}'_{\mathsf{st}})$ be a **PUF-Access**-unpredictable LR-PUF with $\rho(\lambda) = \ell(\lambda)$. Using $\mathcal{L}'$ we define $\mathcal{L} := (\mathsf{Setup}, \mathsf{Query}'_{\mathsf{st}}, \mathsf{Rcnf}_{\mathsf{st}})$, where the algorithms are defined in Figure 5.

Let $\mathsf{bad}$ denote the event that the state of the LR-PUF will at some point become $(\mathsf{st}' \,||\, r)$ with $\mathsf{st}' = r$ in an experiment with $q_r$ invocations of $\mathsf{Rcnf}$. Again, as the **PUF-Access**-unpredictability adversary can only reconfigure the PUF honestly, it holds that

$$\Pr[\mathsf{bad}] \leq \frac{q_r}{2^\lambda}.$$

$\mathsf{Setup}(1^\lambda)$
$(\mathsf{id}, \mathsf{st}) \leftarrow \mathsf{Setup}'(1^\lambda)$
$\mathsf{st}' \leftarrow_\$ \{0,1\}^{\rho(\lambda)}$
**Return** $(\mathsf{id}, \mathsf{st}' \,||\, 0^{\rho(\lambda)})$

$\mathsf{Rcnf}_{\mathsf{st}' \,||\, r}$
**IF** $\mathsf{st}' = r$ **THEN**
$\mathsf{st}' := 0^{\rho(\lambda)}$
$r \leftarrow \mathsf{Query}'_{\mathsf{st}'}(0^\lambda)$
**ELSE**
$\mathsf{st}' \leftarrow_\$ \{0,1\}^{\rho(\lambda)}$
$r \leftarrow \{0,1\}^{\rho(\lambda)}$
**Return** $\mathsf{st}' \,||\, r$

Fig. 5 This construction is **PUF-Access**-unpredictable but not **Full-Access**-unpredictable.

So a reduction against the **PUF-Access**-unpredictability of the underlying LR-PUF $\mathcal{L}'$, that simply forwards all the oracle queries to its own challenger provides a perfect simulation of the **PUF-Access**-unpredictability challenger unless $\mathsf{bad}$ occurs. Thus, for an arbitrary **PUF-Access** adversary $\mathcal{A}$,

$$\begin{aligned}
&|\mathbf{PUF\text{-}Access}^{\mathcal{L}'}_{\mathcal{B}}(\lambda) - \mathbf{PUF\text{-}Access}^{\mathcal{L}}_{\mathcal{A}}(\lambda)| \\
&\qquad = \Big|\Pr[\mathsf{bad}] + (1 - \Pr[\mathsf{bad}])\mathbf{PUF\text{-}Access}^{\mathcal{L}}_{\mathcal{A}}(\lambda) \\
&\qquad\quad - \mathbf{PUF\text{-}Access}^{\mathcal{L}}_{\mathcal{A}}(\lambda)\Big| \\
&\qquad = \Big|\Pr[\mathsf{bad}] - \Pr[\mathsf{bad}]\mathbf{PUF\text{-}Access}^{\mathcal{L}}_{\mathcal{A}}(\lambda)\Big| \\
&\qquad = \Big|\Pr[\mathsf{bad}](1 - \mathbf{PUF\text{-}Access}^{\mathcal{L}}_{\mathcal{A}}(\lambda))\Big| \\
&\qquad \leq \Pr[\mathsf{bad}] \\
&\qquad \leq \frac{q_r}{2^\lambda} \leq \mathsf{negl}(\lambda).
\end{aligned}$$

The construction is not **Full-Access**-unpredictable, since a state-setting adversary can learn a valid query for a reconfigured state, by choosing a state with $\mathsf{st}' = r$. □

Since **Full-Access**-unpredictability implies **PUF-Access**-unpredictability and we because have seen that **St-Access**- and **PUF-Access**-unpredictability do not imply each other (see Proposition 5), **Full-Access**-unpredictability can also not follow from **St-Access**-unpredictability.

**Corollary 4.** *There exists a* **St-Access***-unpredictable LR-PUF construction, which is not secure w.r.t.* **Full-Access***-unpredictability.*

## 4 Construction

In this section we present our construction that fulfills the **Full-Access** notion of unpredictability we defined in Section 3. Our scheme can be seen as a randomized version of the speed-optimized construction from [8] with the difference that the

reconfiguration algorithm chooses a fresh state uniformly at random (instead of computing it as the hash of the old state). Afterwards, we show that the reconfiguration algorithm must be randomized in order to achieve our strongest notion of unpredictability.

$\mathsf{Query}_{\mathsf{st}}(c)$
$w \leftarrow \mathsf{Hash}(\mathsf{st} \,||\, c)$
$y \leftarrow \mathsf{E}(w)$
**Return** $y$

$\mathsf{Rcnf}_{\mathsf{st}}$
$\mathsf{st} \leftarrow_{\$} \{0,1\}^{\ell(\lambda)}$
**Return** st

Fig. 6 The full LR-PUF construction.

**Theorem 1.** *The* full *construction (Figure 6) is* **Full-Access***-unpredictable.*

Proof. Let $\mathcal{A}$ be a PPT **Full-Access**-adversary against full. At the end of the experiment $\mathsf{FULL\text{-}ACCESS}^{\mathsf{full}}_{\mathcal{A}}(\lambda)$, this adversary will output a triple $(\mathsf{st}^*, c^*, r^*)$ and will win if $r^* = \mathsf{Query}_{\mathsf{st}'}(c^*)$, where $\mathsf{st}'$ is obtained by invoking $\mathsf{Rcnf}_{\mathsf{st}^*}$. If $\mathcal{A}$ can be sure that full will be queried with $c^*$ on some specific state $\hat{\mathsf{st}}$ it can easily predict the response $r^*$ using the PUF-oracle $\mathsf{E}$. Thus, fix $\hat{\mathsf{st}} \in \{0,1\}^{\ell(\lambda)}$ and let state-hit denote the event that $\hat{\mathsf{st}}$ is the output of $\mathsf{Rcnf}_{\mathsf{st}^*}$. Because in full, the $\mathsf{Rcnf}_{\mathsf{st}}$ algorithm draws the new state independently from the old one and at random it holds that

$$\begin{aligned}\Pr[\mathsf{state\text{-}hit}] &= \Pr[\mathsf{st} = \hat{\mathsf{st}}; \mathsf{st} \leftarrow_{\$} \mathsf{Rcnf}_{\mathsf{st}^*}()] \\ &= \frac{1}{2^{\ell(\lambda)}} \leq \frac{1}{2^{\lambda}}.\end{aligned}$$

In the case that the newly reconfigured device does not have the state desired by the adversary, let predict denote the event that the adversaries prediction is valid even for this state. Then

$$\begin{aligned}\Pr[\mathsf{predict}] &= \Pr[r = r^*; r \leftarrow \mathsf{E}(\mathsf{Hash}(\mathsf{st} \,||\, c^*)), \mathsf{st} \leftarrow_{\$} \{0,1\}^{\ell(\lambda)}] \\ &= \Pr[r = r^*; r \leftarrow \mathsf{E}(\mathsf{Hash}(c)), c \leftarrow_{\$} \{0,1\}^{\ell(\lambda)+\lambda}] \\ &\leq \mathbf{Pre}^{\mathcal{P}}_{\mathcal{A}}(\lambda).\end{aligned}$$

Pulling the two cases together yields the overall **Full-Access**-predictability of full:

$$\begin{aligned}\mathbf{Full\text{-}Access}^{\mathsf{full}}_{\mathcal{A}}(\lambda) &= \Pr[\mathsf{state\text{-}hit}] + (1 - \Pr[\mathsf{state\text{-}hit}]) \Pr[\mathsf{predict}] \\ &\leq \frac{1}{2^{\lambda}} + \mathbf{Pre}^{\mathcal{P}}_{\mathcal{A}}(\lambda).\end{aligned}$$

As the underlying PUF was assumed to be unpredictable, this means that $\mathbf{Full\text{-}Access}^{\mathsf{full}}_{\mathcal{A}}(\lambda)$ is negligible, i.e., full is **Full-Access**-unpredictable. □

**Proposition 7.** *The* full *construction is* **St-Access***-unpredictable.*

PROOF. As the construction's Query algorithm is the same as the speed-construction's (see Figure 7), and the Rcnf-algorithm cannot be used in any advantageous way by an adversary, the construction is **St-Access**-unpredictable as long as the speed-construction is **St-Access**-unpredictable. This is shown in Proposition 9. □

## 5 Conclusion

In this paper, we have reconsidered the concept of Logically Reconfigurable PUFs, an extension of the PUF primitive with applications in embedded devices for access control or object tracking. We have given a formal definition of LR-PUFs and presented several new notions of unpredictability, which help to classify constructions according to the scenarios they could be employed in. An evaluation of two previously given construction has shown these constructions to withstand stronger adversaries than initially shown. Finally, we have given a new construction that can handle the strongest adversaries defined in this work and we have seen that these notions create an interesting separation between such constructions that rely on deterministic reconfiguration algorithms and such that randomize reconfiguration.

## Acknowledgements

Dominique Schröder was supported by the German Federal Ministry of Educationand Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA – www.cispa-security.org) and also by an Intel Early Career Faculty Honor Program Award. Finally, we thank the reviewers for their valuable comments.

# Bibliography

[1] K. D. Akdemir, Z. Wang, M. Karpovsky, and B. Sunar. Design of cryptographic devices resilient to fault injection attacks using nonlinear robust codes. In M. Joye and M. Tunstall, editors, *Fault Analysis in Cryptography*, Information Security and Cryptography, pages 171–199. Springer Berlin Heidelberg, 2012.

[2] F. Armknecht, R. Maes, A.-R. Sadeghi, F.-X. Standaert, and C. Wachsmann. A formalization of the security features of physical functions. In *2011 IEEE Symposium on Security and Privacy*, pages 397–412, Berkeley, California, USA, May 22–25, 2011. IEEE Computer Society Press.

[3] F. Armknecht, R. Maes, A.-R. Sadeghi, B. Sunar, and P. Tuyls. Memory leakage-resilient encryption based on physically unclonable functions. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 685–702, Tokyo, Japan, Dec. 6–10, 2009. Springer, Berlin, Germany.

[4] C. Brzuska, M. Fischlin, H. Schröder, and S. Katzenbeisser. Physically uncloneable functions in the universal composition framework. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 51–70, Santa Barbara, CA, USA, Aug. 14–18, 2011. Springer, Berlin, Germany.

[5] I. Eichhorn, P. Koeberl, and V. van der Leest. Logically reconfigurable pufs: Memory-based secure key storage. In *Proceedings of the sixth ACM workshop on Scalable trusted computing*, pages 59–64. ACM, 2011.

[6] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Controlled physical random functions. In *In Proceedings of the 18th Annual Computer Security Conference*, 2002.

[7] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. CRC Press, Boca Raton, Fla, 2008.

[8] S. Katzenbeisser, Ü. Koçabas, V. van der Leest, A.-R. Sadeghi, G. J. Schrijen, H. Schröder, and C. Wachsmann. Recyclable PUFs: Logically reconfigurable PUFs. In B. Preneel and T. Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 374–389, Nara, Japan, Sept. 28 – Oct. 1, 2011. Springer, Berlin, Germany.

[9] K. Kursawe, A. Sadeghi, D. Schellekens, B. Skoric, and P. Tuyls. Reconfigurable physical unclonable functions - enabling technology for tamper-resistant storage. In *Hardware-Oriented Security and Trust, 2009. HOST '09. IEEE International Workshop on*, pages 22–29, 2009.

[10] R. Maes and I. Verbauwhede. Physically unclonable functions: A study on the state of the art and future research directions. In A.-R. Sadeghi and D. Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 3–37. Springer Berlin Heidelberg, 2010.

[11] R. Ostrovsky, A. Scafuro, I. Visconti, and A. Wadia. Universally composable secure computation with (malicious) physically uncloneable functions. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 702–718. Springer Berlin Heidelberg, 2013.

[12] R. S. Pappu. Physical one-way functions. PhD thesis, 2001.

[13] U. Rührmair, J. Sölter, and F. Sehnke. On the foundations of physical unclonable functions. Cryptology ePrint Archive, Report 2009/277, 2009. http://eprint.iacr.org/.

[14] A.-R. Sadeghi, I. Visconti, and C. Wachsmann. PUF-Enhanced RFID Security and Privacy. In *Secure Component and System Identification (SECSI)*, Cologne, Germany, Apr. 2010.

[15] S. Schulz, A.-R. Sadeghi, and C. Wachsmann. Short paper: lightweight remote attestation using physical functions. In *Proceedings of the fourth ACM conference on Wireless network security*, WiSec '11, pages 109–114, New York, NY, USA, 2011. ACM.

[16] G. E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, pages 9–14, New York, NY, USA, 2007. ACM.

[17] P. Tuyls and L. Batina. RFID-Tags for Anti-counterfeiting. In D. Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 115–131. Springer Berlin Heidelberg, 2006.

[18] B. Škorić, P. Tuyls, and W. Ophey. Robust key extraction from physical uncloneable functions. In J. Ioannidis, A. Keromytis, and M. Yung, editors, *Applied Cryptography and Network Security*, volume 3531 of *Lecture Notes in Computer Science*, pages 407–422. Springer Berlin Heidelberg, 2005.

## A Proof of Proposition 5

Proof. Both subproofs will recycle constructions from previous proofs.

Let us first show *i)*. In the proof of Proposition 3 we have seen a construction, which is **Plain**-unpredictable but not **St-Access** unpredictable. We will now show that the same construction (construction Figure 4) is not only **Plain**-unpredictable but even **PUF-Access**-unpredictable as long as the underlying LR-PUF is also **PUF-Access**- and not only **Plain**-unpredictable. In this case, it is possible to perform the same reduction of **PUF-Access**-unpredictability of the construction to **PUF-Access**-unpredictability of the underlying LR-PUF as before for **Plain**-unpredictability, with the addition that PUF-Oracle queries will be forwarded to the challenger for **PUF-Access**-unpredictability of the underlying LR-PUF. The claim then follows.

For the proof of *ii)*, consider again the speed-optimized construction $\mathsf{speed}$. In Proposition 9 we will show that this construction is **St-Access**-unpredictable. It is, however, not **PUF-Access**-unpredictable since the reconfiguration algorithm is deterministic (see Section 4). □

## B PUFs and Collision-Resistant Hash Functions

As some of the LR-PUF constructions we will discuss use collision resistant hash functions, we will first have to define this primitive.

**Definition 8 (Hash function).** *Let $\ell : \mathbb{N} \to \mathbb{N}$ be a polynomial. A pair of PPT algorithms $(\mathsf{Gen}, \mathsf{Hash})$ is called a hash function if:*

$\mathsf{Gen}$. *Takes as input a security parameter $1^\lambda$ and returns an index $\mathsf{id}$ from some index set $\mathcal{I}_\lambda$.*

$\mathsf{Hash}$. *Takes as input an index $\mathsf{id}$ and a bit string $x \in \{0,1\}^*$. It returns an output string $\mathsf{Hash}(\mathsf{id}, x) \in \{0,1\}^{\ell(\lambda)}$. We set $\mathsf{Hash}(x) := \mathsf{Hash}(\mathsf{id}, x)$.*

What follows is the usual definition of collision resistance against a PPT adversary [7], modified to account for possible advantages an adversary might have through PUF Access. For a discussion of the possible complexity theoretic implications of PUF access, please refer to [4, 11].

**Definition 9 (Collision resistance w.r.t. PUFs).** *Let $\mathcal{P} = (\mathsf{S}, \mathsf{E})$ be a PUF family. A hash function is called collision-resistant with respect to $\mathcal{P}$, if for all PPT algorithms $\mathcal{A}$ that have black-box access to $\mathcal{P}$ the probability that the experiment $\mathsf{COLL}_{\mathcal{A}}^{\mathcal{P}}$ evaluates to 1 is negligible, where*

***Experiment*** $\mathsf{COLL}_{\mathcal{A}}^{\mathcal{P}}(\lambda)$
$\quad \mathsf{id} \leftarrow \mathsf{Gen}(1^{\lambda})$
$\quad (x, x') \leftarrow \mathcal{A}^{\mathsf{S},\mathsf{E}}(\mathsf{id})$
*Return 1 iff* $\mathsf{Hash}(x) = \mathsf{Hash}(x')$ *and* $x \neq x'$.

### B.1 PUFs and Asymptotic Security

From a practitioners perspective, the asymptotic formulation of unpredictability presented in this paper can seem problematic, as it does not quantify exactly how much security a given construction provides. In [13] there is also an argument made against asymptotic security claims about PUFs which suggests that such claims are not meaningful. The core of the argument is that PUFs are finite functions and an adversary could just have a hard-coded table of challenge response pairs for a given PUF. We believe this criticism is not applicable in our case, as we define PUFs as families of functions from which one is sampled in the unpredictability experiment. As the probability is also taken over the randomness of this sampling process, it is unlikely that the adversary will have the lookup table *for this specific PUF* hard-coded. As discussed in [4] it is not clear, that such assumptions still hold in the presence of adversaries that have access to a PUF. Therefor, we restrict the use of hash functions in our constructions to those which are still collision-resistant, even in the presence of PUF-power adversaries. We believe this assumption is well-founded, as intuitively a PUF does not provide an obvious advantage in finding collisions for a hash function. Additionally one could make an empirical argument for the validity of the assumption. If PUFs could provide an advantage in solving computational problems believed to be intractable for PPT algorithms, there should be real world adversaries that would use this advantage.As we do not currently know of such real world adversaries it does not seem far fetched to assume that PUFs are "safe" with respect to computational hardness assumptions. This is the approach taken in [11].

## C Deterministic Reconfiguration Algorithms

As seen before, the PUF-access variants of unpredictability exclude LR-PUF constructions where access to the underlying PUF makes the Query and Rcnf algorithms completely predictable to the adversary. We will now show that an LR-PUF construction, which is, in this sense, deterministic cannot achieve PUF-access unpredictability.

**Proposition 8.** *A LR-PUF construction* $\mathcal{L} = (\mathsf{Setup}, \mathsf{Query}_{\mathsf{st}}, \mathsf{Rcnf}_{\mathsf{st}})$*, where* $\mathsf{Rcnf}_{\mathsf{st}}$ *is deterministic cannot achieve* **PUF-Access***-unpredictability.*

Proof. Let $\mathcal{L} = (\mathsf{Setup}, \mathsf{Query}_{\mathsf{st}}, \mathsf{Rcnf}_{\mathsf{st}})$ be a construction where $\mathsf{Rcnf}_{\mathsf{st}}$ is deterministic. An adversary in the **PUF-Access**-unpredictability experiment can proceed as follows: Upon receiving the initial state $\mathsf{st}$, the adversary can compute the reconfiguration itself, giving it access to the next state $\mathsf{st}'$ without invoking the reconfiguration oracle. Thus, $\mathcal{A}$ can precompute-challenge response pairs $(c^*, r^*)$ for $\mathsf{st}'$ using the PUF-Oracle. It can then output $(\mathsf{st}, c^*, r^*)$ as its prediction. Because $\mathsf{Rcnf}$ is deterministic, the challenger will also compute $\mathsf{st}' \leftarrow \mathsf{Rcnf}_{\mathsf{st}}$ and the prediction $(c^*, r^*)$ will be valid for that state. □

As **PUF-Access**-unpredictability is implied by **Full-Access**-unpredictability, the following corollary follows immediately.

**Corollary 5.** *An LR-PUF construction* $\mathcal{L} = (\mathsf{Setup}, \mathsf{Query}_{\mathsf{st}}, \mathsf{Rcnf}_{\mathsf{st}})$*, where* $\mathsf{Rcnf}_{\mathsf{st}}$ *is deterministic cannot achieve* **Full-Access***-unpredictability.*

Of course, these claims exclude LR-PUF constructions which involve more than one PUF, as the **PUF-Access**- and **Full-Access**-unpredictability experiments only provide access to a single underlying PUF. It is, however, straightforward to extend the definition to multiple PUFs. This can also be intuitively motivated, as a multi-PUF construction, which does not secure access to one of the employed PUFs is unlikely to secure access to the other PUFs.

*Remark 5.* As generation of proper randomness on an highly embedded system such as a public transport access token seems impractical, this result establishes that the **Full-Access**-unpredictability notion might not be achievable for all application scenarios. However, in scenarios, where reconfigurations occur infrequently the negative effect of randomness that comes from a weak source is likely to be tolerable. Additionally, if the degree of sophistication of the device is high enough, there could already be a circuit in the device implementing a hash function such as SHA-2, which may be used to extract some randomness in a heuristic fashion.

## D Revisiting Earlier Constructions

Based on the newly proposed unpredictability notions it is worthwhile to revisit the original LR-PUF constructions given in [8]. We show that they provide unpredictability in more adverse settings than originally demonstrated.

### D.1 Speed-Optimized-Construction

Let us first consider the *speed-optimized* implementation of LR-PUFs given in [8]. The basic idea of the construction is to use a collision-resistant hash function to compound the state and the LR-PUF challenge into one PUF stimulus. The

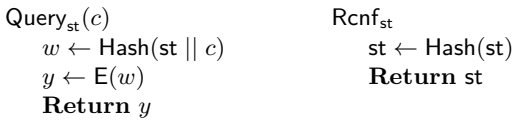


Fig. 7 The speed-optimized construction speed from [8].

formal description of the algorithms is shown in the left part of Figure 7. As the reconfiguration algorithm is deterministic, this construction cannot achieve **PUF-Access**- or **Full-Access**-unpredictability (see Section 4), however, we can show that it is **St-Access**-unpredictable, which is an improvement on the unpredictability result given in [8].

**Proposition 9.** *The* speed *construction is* **St-Access***-unpredictable.*

Proof. The proof of **St-Access**-unpredictability of the construction is similar to the security proof of the speed-optimized construction from [8], reducing **St-Access**-unpredictability to the unpredictability of the underlying PUF and collision-resistance of the hash function. Let $\mathcal{A}$ be a **St-Access**-unpredictability adversary against $\mathcal{L}$ and assume w.l.o.g. that $\mathcal{A}$ never asks the same query twice. After fixing $\mathcal{A}$ and its randomness, it is easy to see that $\mathcal{A}$ falls into one of two categories:

- $\mathcal{A}$ finds a collision for $\mathsf{Hash}$, i.e., the prediction of $\mathcal{A}$ has the form $(\mathsf{st}^*, c^*, \hat{r})$, where $\hat{r} = \mathsf{E}(w)$ and $w = \mathsf{Hash}(\mathsf{st}^* \,||\, c^*) = \mathsf{Hash}(\hat{\mathsf{st}} \,||\, \hat{c})$ for some challenge $\hat{c}$ that $\mathcal{A}$ has queried for a state $\hat{\mathsf{st}}$ during the experiment.
- $\mathcal{A}$'s prediction has a fresh response, i.e., it has the form $(\mathsf{st}^*, c^*, r^*)$ where $\mathsf{Hash}(\mathsf{st}^* \,||\, c^*) \neq \mathsf{Hash}(\hat{\mathsf{st}} \,||\, \hat{c})$ for all challenges $\hat{c}$ that $\mathcal{A}$ has made and all states $\hat{\mathsf{st}} \in \mathcal{S}$.

To see that the construction is **St-Access**-unpredictable, consider the same reduction $\mathcal{B}$ to the underlying PUF's unpredictability and the collision-resistance of the hash function that was used to show **Backward**-unpredictability of the construction in [8]. Assuming that $\mathcal{A}$ is a successful **St-Access**-adversary, $\mathcal{B}$ keeps a state st locally and forwards $\mathcal{A}$'s queries to its own oracles, while storing the queries in a list. As $\mathcal{A}$ is successful, the reduction can be used to find a valid prediction of the underlying PUF or a collision for the hash function, both of which violate the assumptions about the underlying primitives. The additional reconfiguration oracle that the **St-Access** adversary has access to is not an issue for $\mathcal{B}$, it can be simulated by updating the internal state st to $\mathsf{Hash}(\mathsf{st})$.

□

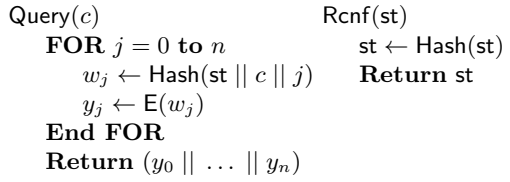


Fig. 8 The area-optimized construction area from [8].

### D.2 Area-Optimized-Construction

Beside the speed-optimized construction, [8] also propose an area-optimized construction, in which a small-range PUF is stimulated repeatedly on different sub-challenges derived from the original challenge. The sub-challenge responses are then assembled to one larger response to the original challenge. The formal description is given in the right part of Figure 7. We will now see that the construction *can* be **St-Access**-unpredictable, but the degree of unpredictability depends on the choice of underlying PUF and the iteration count $n$. Note, that a noise bound $d_{noise}$ on the full response means that the underlying PUF should not produce responses that are noisier than $\frac{d_{noise}}{n}$.

**Proposition 10.** *If there exist collision-resistant hash functions with respect to PUFs, and the underlying PUF is unpredictable, then the* area*-construction is* **St-Access***-secure.*

Proof. Consider a successful **St-Access**-unpredictability adversary against the area-construction. W.l.o.g. we assume that this adversary never asks the same query twice. Such an adversary can only be successful if it can break the collision-resistance property of the employed hash function or predict the underlying PUF. There are two extreme cases and many mixed ones. The first extreme case is that the adversary finds a challenge $c^*$ such that for a set of previous challenges $c_1, \ldots, c_n$ and states $\mathsf{st}_1, \ldots, \mathsf{st}_n$ and all $j \in \{1, \ldots, n\}$ $\mathsf{Hash}(\mathsf{st}^* \mid\mid c^* \mid\mid j) = \mathsf{Hash}(\mathsf{st}_j \mid\mid c_j \mid\mid j)$, i.e., the adversary can find $n$ collisions for the hash function. The other extreme case is that the adversary issues a valid prediction tuple $(\mathsf{st}^*, c^*, r^*)$ where for all $j \in \{1, \ldots, n\}$, there is no collision, which means the adversary has predicted the underlying PUF on $n$ fresh challenges. In between these extreme cases are those in which for some $j$, there is a colliding pair $(\mathsf{st}_j, c_j)$ which is known to the adversary and for the remaining the adversary predicts the underlying PUF. Let $\mathsf{Coll}(i)$ denote the probability that the probability that the adversary finds $i$ collisions. Then for any $i$, $\mathsf{Coll}(i) \leq \Pr\left[\mathsf{COLL}_{\mathcal{A}}^{\mathsf{Hash}} = 1\right]$. Let further $F(i)$ denote the probability that the adversary predicts $i$ sub-responses, either by PUF prediction or finding a collision.

Then we can formulate $\mathbf{St\text{-}Access}_{\mathcal{A}}^{\mathsf{area}}(secparam) = F(n)$ as a recurrence relation:

$$F(i) = \begin{cases} \mathsf{Coll}(i) + (1 - \mathsf{Coll}(i))\mathbf{Pre}_{\mathcal{A}}^{\mathcal{P}}(\lambda)F(i-1), & \text{if } i > 0, \\ 1, & \text{otherwise} \end{cases}$$

Using the bound for $\mathsf{Coll}(i)$ we can bound $F(i)$:

$$F(i) \leq \Pr\left[\mathsf{COLL}_{\mathcal{A}}^{\mathsf{Hash}} = 1\right] + \mathbf{Pre}_{\mathcal{A}}^{\mathcal{P}}(\lambda)F(i-1).$$

Inductively it can be shown that for all $m \in \mathbb{N}$,

$$F(m) \leq \Pr\left[\mathsf{COLL}_{\mathcal{A}}^{\mathsf{Hash}} = 1\right]\left(\sum_{i=0}^{m-1} \mathbf{Pre}_{\mathcal{A}}^{\mathcal{P}}(\lambda)^i\right) + \mathbf{Pre}_{\mathcal{A}}^{\mathcal{P}}(\lambda)^m.$$

As $\mathbf{St\text{-}Access}_{\mathcal{A}}^{\mathsf{area}}(\lambda) = F(n)$, where $n$ is the iteration count of construction's loop, it follows that

$$\mathbf{St\text{-}Access}_{\mathcal{A}}^{\mathsf{area}}(\lambda) \leq \Pr\left[\mathsf{COLL}_{\mathcal{A}}^{\mathsf{Hash}} = 1\right]\left(\sum_{i=0}^{n-1} \mathbf{Pre}_{\mathcal{A}}^{\mathcal{P}}(\lambda)^i\right) + \mathbf{Pre}_{\mathcal{A}}^{\mathcal{P}}(\lambda)^n.$$

Because we have assumed that the underlying PUF is unpredictable this probability is negligible in $\lambda$ for any fixed $n$ that is polynomial in $\lambda$. □