D1.2: Security architectures, protocol design and evaluation principles for anti-counterfeiting/anti-tampering solutions

Project number:	238811
Project acronym:	**UNIQUE**
Project title:	Foundations for Forgery-Resistant Security Hardware
Start date of the project:	01.09.2009
Duration:	30 months
Deliverable type:	Document
Deliverable reference number:	238811/ D1.2 | v1.0
Deliverable title:	Security architectures, protocol design and evaluation principles for anti-counterfeiting/anti-tampering solutions
WP contributing to deliverable:	WP1
Due date:	2011-02-28 (M18)
Actual submission date:	2011-02-28
Responsible organisation:	TUD
Authors:	Vincent van der Leest, Geert-Jan Schrijen (IID), Stefan Katzenbeisser, Heike Schröder, Christian Wachsmann (TUD), Patrick Koeberl (INTEL)
Abstract:	This document is deliverable D1.2 from WP1 of the FP7 project UNIQUE. The goal of this deliverable is to summarize appropriate security architectures and its components as well as the protocol design and evaluation principles for anti-counterfeiting and anti-tampering solutions. The deliverable describes the main principles for the development of new technologies in course of the UNIQUE project as well as a set of evaluation principles.
Keywords:	Security Architectures, Protocol Design
Dissemination level:	Public
Revision:	v1.0
Instrument:	STREP
Thematic Priority:	ICT

Contents

1 Introduction

The goal of this deliverable is to summarize appropriate security architectures and its components as well as the protocol design and evaluation principles for anti-counterfeiting and anti-tampering solutions. The deliverable describes the main principles for the development of new technologies in course of the UNIQUE project as well as a set of evaluation principles. On top of this, it contains security models for the analysis of the concrete schemes developed in WP2 for hardware anti-tampering/anti-counterfeiting solutions and recommendations for future research in WP2.

Outline. Deliverable D1.2 consists of a theoretical and a practical part. In the theoretical part, we introduce and formally define logically reconfigurable physically unclonable functions (LR-PUFs) (see Section 3). Roughly speaking, an LR-PUF has the ability to change its challenge/response behavior after deployment in a completely unpredictable way by using a logical reconfiguration mechanism on top of a common PUF. Furthermore, we present a generic LR-PUF construction and a formal security model for such kind of PUFs following the methodologies of modern cryptography. Moreover, we show two different concrete LR-PUF instantiations, one of them specifically tailored to the constrained resources of embedded devices (e.g., RFID chips), and prove their security.

In the practical part we consider two application scenarios using LR-PUFs: recyclable tokens (see Section 4) and hardware/software-binding (see Section 5). In each case, we give a detailed use case description, analyze existing solutions, and present a security architecture for the respective application scenario. Finally, in Section 6 we outline an evaluation framework and analyse the security requirements for LR-PUFs.

2 Preliminaries and Notation

2.1 Notation

For a finite set S, $|S|$ denotes the size of set S whereas for an integer (or a bit-string) n the term $|n|$ means the bit-length of n. The term $s \in_R S$ means the assignment of a uniformly chosen element of S to variable s. Let A be a probabilistic algorithm. Then $y \leftarrow \mathsf{A}(x)$ means that on input x, algorithm A assigns its output to variable y. The term $[\mathsf{A}(x)]$ denotes the set of all possible outputs of A on input x. $\mathsf{A}_K(x)$ means that the output of A depends on x and some additional parameter K (e.g., a secret key). Let E be some event (e.g., the result of a security experiment), then $\Pr[E]$ denotes the probability that E occurs. Let l be a security parameter (e.g., the bit length of a secret key). Probability $\epsilon(l)$ is called *negligible* if for all polynomials f it holds that $\epsilon(l) \leq 1/f(l)$ for all sufficiently large l. Probability $1 - \epsilon(l)$ is called *overwhelming* if $\epsilon(l)$ is negligible.

2.2 Physically Unclonable Functions (PUFs)

We start by giving a definition of a physically unclonable fucntion (PUF).

Definition 1 (Physically Unclonable Functions) *A family of physically unclonable functions* $\mathsf{PUF} = (\mathsf{Assemb}, \mathsf{Eval})$ *is a set of physical realizations of a family of probabilistic algorithms that fulfills the following algorithmic and physical properties.*

Assembling *The assembling process* Assemb *is both a physical and an algorithmic procedure. The input is the security parameter* 1^n *(e.g., the dimensions of the PUF specifying the bitlength of challenges and responses) and the output is a physically uncloneable function* PUF.

Evaluation *The evaluation procedure* Eval *maps an input* $c \in \{0,1\}^n$ *to an output* $r \in \{0,1\}^m$. *The input* c *is called* stimulus *or* challenge, *and the output* r *is referred to as* response.

Since we operate on physical measurements we have to take care of their natural conditions. This is, measurements are noisy by nature and thus, the measurements of the same stimuli will result in different but closely related responses. In some application, such as authentication, it is enough to apply a distance function to deal with noisy measurements. In other applications, however, a distance function is not enough because a noise-free response with a uniform distribution (such as cryptographic keys) is required. To deal with this problem, *fuzzy extractors* of Dodis et al. are applied – a secure form of error correction that enables a reliable extraction of a uniform key from a noisy non-uniform input. We refer the interested reader to [DRS04] for a comprehensive discussion and formal definitions. In the following, we write $r \leftarrow \mathsf{PUF}(c)$ to indicate that r is a legitimate response to $\mathsf{PUF}(c)$.

We now turn to the formalization of the security properties of PUFs. The first property reflects the fact that it is not possible to produce two devices that implement the same challenge/response behavior.

Definition 2 (Uncloneability) *We say that the PUF is* uncloneable *if there exists no efficient procedure* clone *that, given the device* D*, builds another physical object* $D' \neq D$ *such that both objects implement the same challenge/response behavior.*

Next, we turn to the definition of unpredictability. Unpredictability states that nobody should be able to predict the response to a given challenge without evaluating the PUF. The difference to "regular" unpredictable functions is that an adversary may indeed be able to predict some values. That is, some PUFs map closely related measurements to closely related responses. Therefore we require that the prediction has to be non-trivial. In modern cryptography, the security objectives are typically defined as a security experiment (game), where a polynomial bounded adversary can interact with a set of oracles (algorithms) that model the capabilities of the adversary. Hence, we now formalize unpredictability by defining the following game, where $\mathcal{A}$ denotes the adversary: The adversary first collects several PUF measurements up to a certain moment where he has no longer access to the device. Subsequently, the adversary has to output a valid challenge and response pair (c^*, r^*) such that the (e.g., Hamming) distance between its prediction and all previous measurements is at least ϵ. If this conditions is fulfilled and if $r^* \leftarrow \mathsf{PUF}(c^*)$, then the adversary wins the game.

Game PRE$(\mathsf{PUF}, \mathcal{A}, n)$

Setup The challenger generates a PUF by running the assembling procedure $\mathsf{Assemb}(1^n)$ to obtain a physically unclonable function PUF.

Queries Proceeding adaptively, the adversary may query the PUF-oracle $\mathcal{O}_{\mathsf{PUF}}$ q-times on a challenge $c \in \{0,1\}^n$. It returns $r \leftarrow \mathsf{PUF}(c)$ to the adversary and adds c to a list Q.

Output Eventually, the adversary outputs a pair (c^*, r^*). The game outputs 1 if the distance function $\mathsf{dist}(c^*, c) > \epsilon$ for all $c \in Q$ and $r^* \leftarrow \mathsf{PUF}(c^*)$.

We define $\mathsf{adv}_{\mathcal{A}}$ be the probability that $\mathbf{Prob}[\mathbf{PRE}(\mathsf{PUF}, \mathcal{A}, n) = 1]$.

Definition 3 *[Unpredictability] A physically unclonable function* PUF *represented by the procedures* Assemb *and* PUF *is* unpredictable *with respect to the game* $\mathbf{PRE}(\mathsf{PUF}, \mathcal{A}, n)$ *if for all polynomial-time algorithms* $\mathcal{A}$ *running in time* t*, the probability* $\mathbf{Prob}[\mathbf{PRE}(\mathsf{PUF}, \mathcal{A}, n) = 1]$ *is at most* $\delta_{\mathbf{PRE}}$*.*

2.3 Collision-resistant Hash-Functions

We define collision-resistant hash functions according to Katz-Lindell [KL07]:

Definition 4 *A hash function is a pair of probabilistic polynomial-time algorithms* $\Pi = (\mathsf{hGen}, H)$ *satisfying the following:*

1. hGen *is a probabilistic algorithm which takes as input a security parameter* 1^n *and outputs a key* k*. We assume that* 1^n *is implicit in* k*.*

2. *There exists a polynomial* ℓ *such that* H *takes as input a key* k *and a string* $x \in \{0,1\}^*$ *and outputs a string* $H^k(x) \in \{0,1\}^{\ell(n)}$ *(where* n *is the value of the security parameter implicit to* k*).*

We simplify the notation by writing $H(m) := H(k, m)$. Given a hash function $\Pi = (\mathsf{hGen}, H)$, an adversary $\mathcal{A}$, and a security parameter n. We define the following collision-finding game:

Game HASH-COL$(\mathcal{A}, \Pi, n)$

Setup A key k is generated by running $\mathsf{hGen}(1^n)$.

Output The adversary $\mathcal{A}$ is given k and outputs x, x'. The output of the game is defined to be 1 if and only if $x \neq x'$ and $H^k(x) = H^k(x')$. In such a case we say that $\mathcal{A}$ has found a collision.

Definition 5 *A hash function* Π *represented by the procedures* (hGen, H) *is* collision-resistant *if for all efficient adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\mathbf{Prob}[\mathbf{HASH\text{-}COL}(\mathcal{A}, \Pi, n) = 1] \leq \mathsf{negl}(n).$$

3 Logically Reconfigurable PUFs

3.1 Overview

In many applications, it would be desirable to have a reconfigurable PUF, e.g., in order to allow the key derived from a PUF to be updated. In [KSS+09] the authors suggest a number of possible reconfigurable PUFs, that might be useful for this purpose. In theory, these PUFs should be reconfigurable in a physical way to become completely different and independent new PUFs in comparison to the situation before reconfiguration.

In practice however, it turns out that all of the PUFs from [KSS+09] have severe downsides. These downsides have been discussed in UNIQUE Deliverable D2.1 [UNI10b] and have proven to be considerable obstacles for implementing reconfigurable PUFs in practical applications. Hence, either physical constraints or high cost prevent the use of physically reconfigurable PUFs in Integrated Circuits.

However, for some use cases, it suffices to use a PUF that is not physically reconfigurable, but that can only be reconfigured on a logical level. Therefore, we introduce the concept of logically reconfigurable PUF (LR-PUF), for which use case examples are described later in Section 4 and 5.

An LR-PUF can be constructed by using a regular PUF combined with a control logic and non-volatile (NV) memory, as depicted in Figure 1. In this construction the NV-memory

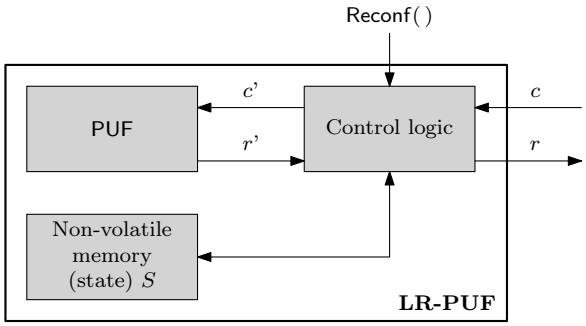

Figure 1: Generic logically reconfigurable PUF construction

stores a state for the LR-PUF. This state is used by the control logic in combination with the (set of) challenge/response pair(s) from the regular PUF (c' and r' in Figure 1) to create a unique (set of) challenge/response pair(s) (c and r in the figure) for that specific state S of the LR-PUF. When the control logic receives a reconfigure command, a 'completely new PUF' is created by the control logic by updating the state S in the NV-memory. This means that a new set of challenge/response pairs (c and r) is obtained, even though the PUF is not physically altered (c' and r' remain the same). In section 3.3 it is explained how the control logic from the LR-PUF should be constructed, in order to have a mathematically proven secure implementation. The challenge/response pair(s) of an LR-PUF can either be used for identification/authentication schemes or for secure key storage. For both of these application scenarios, a use case example can be found later on in this document.

The typical attack scenarios against LR-PUFs can be summarized as follows:

- In case of identification/authentication schemes, an attacker will attempt to derive information about c' and r' of the physical PUF in the LR-PUF scheme. The protocols

used in these schemes should make this impossible for an attacker. The security proof of the proposed protocols can be found in section 3.2.

- In case of secure key storage, an attacker will attempt to obtain the secret key that is constructed based on the LR-PUF response r. Note that there is an important difference with key storage in NV-memory: reading the state does not provide information about the secret key. Therefore, an attacker cannot gain information about the secret key by simply reading the memory content (which would have been a non-invasive attack). If the secret key derived from the LR-PUF is used in the secure way as proposed in [TB06], it should be impossible for an attacker to obtain this key.

- An attacker might want to change the state S in the NV-memory back to an old reconfiguration. In order to be secure against this type of attack, the LR-PUF will have to provide a certain level of tamper-resistance. More in detail, an adversary must not be able to write a specific value (e.g., an old LR-PUF state) into the NV-memory.

Note that the NV-memory of the LR-PUF cannot be written from outside the LR-PUF component. Hence, as discussed in Section 6.2 the only way to write data to this memory is performing an invasive attack. However, using an invasive attack to write a certain value into NV-memory is generally difficult in practice. This means that it will be extremely difficult for an attacker to change the state of the LR-PUF back to a previously used state.

Further, note that denial-of-service attacks can always be performed physically on a device. This can never be prevented, since it is possible to physically break/destroy a device. Using the value of the state to perform a denial-of-service attack does not add value for an attacker, since breaking/destroying the device will be easier and therefore preferred by the attacker.

The remainder of this section provides, besides a security proof for the LR-PUF, two possible constructions for this new PUF type.

Definition 6 (Reconfigurable Physically Unclonable Function) *A tuple* $\mathsf{rPUF} = (\mathsf{rAssemb}, \mathsf{rEval}, \mathsf{Reconfig})$ *is a family of reconfigurable physically unclonable functions that fulfills the following algorithmic and physically properties.*

Assembling *The assembling process* rAssemb *is both a physically and an algorithmic procedure. The input is the security parameter* 1^n *and the output is a* rPUF.

Evaluation *The algorithm* rEval *maps an input* $c \in \{0,1\}^n$ *to an output* r *of bit length* m. *The input* c *is called stimulus (challenge), and the output* r *is referred to as response.*

Reconfiguration *The reconfiguration process* Reconfig *takes as input an* rPUF *and outputs a reconfigured* rPUF'.

3.2 Formal Security Model

Intuitively, the security of a reconfigurable physically unclonable function should say that every efficient algorithm that queries the LR-PUF several times is not able to predict the response to a (chosen) challenge *after* the LR-PUF has been reconfigured. In particular this means that such algorithms may indeed be able to predict a response *before* the reconfiguration algorithm has been run, but not afterwards. Consider for example that an adversary learns the challenge-response behavior of a PUF, e.g., of a PUF with a small CRP space.

3.2.1 Unpredictable LR-PUFs

A possible notion of unpredictability of LR-PUFs could be that an attacker queries the LR-PUF a certain number of times and commits to some arbitrary challenge before getting access to the reconfiguration of the LR-PUF. Thus, the attacker commits to a challenge *without* having access to the reconfigured PUF. Although this definition may allow very efficient LR-PUF instantiations, it may not cover the case where the attacker is able to predict a response to a challenge *after* having access to the LR-PUF. Consider for example the situation where the reconfiguration algorithm changes only a *fraction* of the LR-PUF. This could mean that only some CRPs of the reconfigured PUF are fresh while some of them are unchanged. But without having access to the reconfigured PUF the attacker may not be able to exploit this obvious weakness of the PUF.

We formalize this intuition in the following game where the adversaries advantage, $\mathsf{adv}^{unpre}_{\mathsf{rPUF},\mathcal{A}}$, is defined as follows: Let $\mathcal{A} = (\mathcal{A}_L, \mathcal{A}_C)$ be an adversary modeled as a pair of efficient algorithms. $\mathcal{A}$ consists of a learning adversary $\mathcal{A}_L$ and a provocation adversary $\mathcal{A}_C$. Each attacker interacts with the LR-PUF rPUF in the following two phases:

Setup The challenger generates a LR-PUF by calling the assembling procedure, i.e., $\mathsf{rPUF} \leftarrow \mathsf{rAssemb}(1^n)$.

Phase I: Learning The algorithm $\mathcal{A}_L$ is initialized with a fresh random tape and queries rPUF up to q times. At the end of the learning phase, $\mathcal{A}_L$ stops and it outputs a state *st*.

Revocation The challenger reconfigures the rPUF to rPUF' using the reconfiguration algorithm, i.e., $\mathsf{rPUF}' \leftarrow \mathsf{reconf}(\mathsf{rPUF})$.

Phase II: Provocation The algorithm $\mathcal{A}_C$ is initialized with the state *st* and it has access to rPUF'. The attacker $\mathcal{A}_C$ may query rPUF' on arbitrary challenges.

Output At the end of the provocation phase $\mathcal{A}_C$ may output a pair (c^*, r^*).

The adversary $\mathcal{A} = (\mathcal{A}_L, \mathcal{A}_C)$ wins the game if $r^* \leftarrow \mathsf{rPUF}'(c^*)$ and $\mathcal{A}_C$ has never queried rPUF' on c^*.

Definition 7 *Let $|C|$ denote the set of all challenges and let $|R|$ denote the set of all responses. A reconfigurable physically unclonable function $\mathsf{rPUF} : \{0,1\}^{|C|} \rightarrow \{0,1\}^{|R|}$ is unpredictable if any adversary $\mathcal{A}$ running in time t, querying the rPUF at most q_L times and rPUF' at most q_C times wins the above game with probability at most $\frac{1}{|\mathcal{R}|} + \epsilon$, where ϵ is negligible in n.*

3.3 Instantiations

3.3.1 Construction 1

The idea of our first construction is to add a reconfiguration method on top of a regular PUF in order to transform the physically unclonable function into a LR-PUF.

More precisely, the reconfiguration method consists of a n-bit string s (reconfigure state) and a collision resistant hash function H. Now, the reconfiguration mechanism evaluates the PUF on a modified challenge $w' \leftarrow H(c||s)$ derived via the collision-resistant hash function. To reconfigure the LR-PUF, the reconfiguration algorithm simply picks a fresh value of the reconfigure state in order to change the input/output behavior.

Construction 1 *Let* $\mathsf{PUF} = (\mathsf{Assemb}, \mathsf{Eval})$ *be a physically unclonable function. We construct a reconfigurable physically unclonable function* $\mathsf{rPUF} = (\mathsf{rAssemb}, \mathsf{rEval}, \mathsf{Reconfig})$ *as follows:*

Assembling *On input of the security parameter* 1^n *the algorithm* rAssemb *runs the assembling algorithm of the physically unclonable function to obtain a PUF, i.e.,*$\mathsf{PUF} \leftarrow \mathsf{Assemb}(1^n)$. *It chooses a value* $s \leftarrow \{0,1\}^n$ *at random and picks a collision-resistant hash function* H *by running the algorithm* $\mathsf{hGen}(1^n)$. *The algorithm* rAssemb *then returns* $(\mathsf{rPUF}, \mathsf{PUF}, s, H)$.

Evaluation *On input of the challenge* c *the algorithm* rEval *runs the* PUF *evaluation algorithm* Eval *on the modified challenge* $w' = H(c||s)$ *of bit length* n *which maps* w' *to the output* $r \in \{0,1\}^m$.

Reconfiguration *On input* rPUF *the reconfiguration process* Reconfig *modifies the current state* $s \in \{0,1\}^n$ *to the state* $s' \in \{0,1\}^n$.

Theorem 3.1 *If PUF is an unpredictable physically unclonable function and* H *is a collision-resistant hash function, then Construction 1 is an unpredictable reconfigurable physically unclonable function.*

Proof The main idea of our proof is as follows: Suppose to the contrary that Construction 1 is insecure. We then construct an attacker $\mathcal{B}$ that breaks the unpredictability of the underlying PUF or that finds collisions in the collision–resistant hash function H. But before describing the algorithm $\mathcal{B}$ we distinguish between two types of attacker that $\mathcal{A}$ can emulate. Suppose that $\mathcal{A}$ adaptively asks for challenges $(c_1, ..., c_L, c_{L+1}, ...c_C)$ and obtains the corresponding response $(q_1, ..., q_L, q_{L+1}, ...q_C)$. Recall that $q_i \leftarrow \mathsf{rPUF}(w_i)$ with

$$w_i := \begin{cases} H(s||c_i) \text{ for } i = c_1, ..., c_L \\ H(s'||c_i) \text{ for } i = c_{L+1}, ..., c_C \end{cases}$$

and denote by (c^*, r^*) with $r^* \leftarrow \mathsf{rPUF}(w^*)$ and $w^* = H(s'||c^*)$ the prediction for the rPUF by $\mathcal{A}$. We say that

Type-1 attacker The algorithm $\mathcal{A}$ is a type-1 attacker, denoted by $\mathcal{A}_{col}$, if there exists an index i such that $w_i = w^*$ with $r^* \leftarrow \mathsf{rPUF}(w^*)$ and $w^* = H(s'||c^*)$ and $i \in \{1, \ldots, L, L+1, \ldots, C\}$.

Type-2 attacker The algorithm $\mathcal{A}$ is a type-2 attacker, denoted by $\mathcal{A}_{pred}$, if there exists no index i such that $w_i = w^*$ with $r^* \leftarrow \mathsf{rPUF}(w^*)$ and $w^* = H(s'||c^*)$ and $i \in \{1, \ldots, L, L+1, \ldots, C\}$.

In the following we show for each type of attacker, $\mathcal{A}_{col}, \mathcal{A}_{pred}$, how to construct a suitable simulator $\mathcal{B}_{col}, \mathcal{B}_{pred}$. At first, the algorithm $\mathcal{B}$ tosses a coin to guess if $\mathcal{A}$ is a type-1 or a type-2 attacker. Subsequently, $\mathcal{B}$ proceeds as follows:

Type-1 attacker On input key k the algorithm $\mathcal{B}_{col}$ firstly chooses a collision-resistant hash function H, i.e., $H \leftarrow \mathsf{hGen}(1^n)$. Subsequently, it chooses a value $s \leftarrow \{0,1\}^n$ and sets $\mathsf{rPUF}(\cdot) := \mathsf{PUF}(H(s||\cdot))$. Now $\mathcal{B}_{col}$ runs a black-box simulation of $\mathcal{A}_{col}$.

If $\mathcal{A}_L$ sends a challenge c_i to its rPUF oracle, the algorithm $\mathcal{B}_{col}$ computes the answer as follows: It computes $w_i = H(s||c_i)$, queries its oracle on $q_i \leftarrow \mathsf{rPUF}(w_i)$, stores (c_i, w_i, q_i) in some (initially empty) list $\mathcal{L}$, and forwards q_i to the attacker $\mathcal{A}_L$. At some point $\mathcal{A}_L$ stops outputting some state information st.

After obtaining st, the algorithm $\mathcal{B}_{col}$ changes the reconfiguration value s to a fresh and randomly chosen value $s' \leftarrow \{0,1\}^n$ in order to reconfigure rPUF. According to our construction we denote by rPUF' the reconfigured PUF. It then executes $\mathcal{A}_C$ as a black-box on input st.

If $\mathcal{A}_C$ sends a challenge c_i to its rPUF' oracle, the algorithm $\mathcal{B}_{col}$ generates the answer as follows: It sets $w_i = H(s'||c_i)$, queries its oracle on $q_i \leftarrow \mathsf{rPUF}(w_i)$, stores (c_i, w_i, q_i) in some list $\mathcal{L}$, and forwards q_i to the attacker $\mathcal{A}_C$. At some point $\mathcal{A}_C$ stops outputting a challenge-response pair (c^*, r^*).

Finally, the algorithm $\mathcal{B}_{col}$ parses $(w_1, ..., w_L, w_{L+1}, ..., w_C)$ and it outputs the first element w_i (together with q_i) with $w_i = w^*$. If such an element does not exists, then $\mathcal{B}_{col}$ aborts.

Suppose that $\mathcal{A}_{col}$ is a type-1 attacker that succeeds with non-negligible probability. Observe that both algorithms are efficient and that $\mathcal{B}_{col}$ performs a perfect simulation from $\mathcal{A}_{col}$'s point of view. Then there exists an index i such that $w^* = w_i$. But if $w^* = w_i$, then $\mathcal{B}_{col}$ has found a collision in the hash function H. However, this is a contradiction to the collision-resistance of the hash function and thus, such an attacker cannot exist.

Type-2 attacker On input security parameter n the algorithm $\mathcal{B}_{pred}$ firstly chooses a reconfiguration value $s \leftarrow \{0,1\}^n$ and a collision-resistant hash function H, i.e., $H \leftarrow \mathsf{hGen}(1^n)$. Then, $\mathcal{B}_{col}$ runs a black-box simulation of $\mathcal{A}_{pred}$.

If $\mathcal{A}_L$ sends a challenge c_i to its rPUF oracle, the algorithm $\mathcal{B}_{pred}$ sets $w_i = H(s||c_i)$. Subsequently, it queries its oracle on w_i, obtains q_i, stores (c_i, w_i, q_i) in some (initially empty) list $\mathcal{L}$, and forwards q_i to the attacker $\mathcal{A}_L$. At some point $\mathcal{A}_L$ stops outputting some state information st.

After obtaining st, the algorithm $\mathcal{B}_{pred}$ changes the reconfiguration value s to a fresh randomly chosen value $s' \leftarrow \{0,1\}^n$ in order to reconfigure the LR-PUF. According to our construction we denote by rPUF' the reconfigured PUF. It then runs a black-box simulation of $\mathcal{A}_C$ on input st (with access to rPUF'). It then runs a black-box simulation of $\mathcal{A}_C$.

If $\mathcal{A}_C$ sends a challenge c_i to its rPUF oracle, the algorithm $\mathcal{B}_{pred}$ sets $w_i = H(s'||c_i)$. Subsequently, it queries its external PUF-oracle on w_i, obtains q_i, stores (c_i, w_i, q_i) in $\mathcal{L}$, and forwards q_i to the attacker $\mathcal{A}_C$. At some point $\mathcal{A}_C$ stops outputting a challenge-response pair (c^*, r^*).

Finally, the algorithm $\mathcal{B}_{pred}$ outputs some (w^*, r^*) with $w^* = H(s'||c^*)$.

Suppose that $\mathcal{A}_{pred}$ is a type-2 attacker and succeeds with non-negligible probability. Observe that both algorithms are efficient and that $\mathcal{B}_{pred}$ performs a perfect simulation from $\mathcal{A}_{pred}$'s point of view. Then there exists *no* index i such that $w^* = w_i$. But if no such index exists, $\mathcal{B}_{pred}$ has never queried w^* to its oracle and thus, w^* is a prediction. However, this is a contradiction to the unpredictability of PUFs and thus, such an attacker cannot exists. ∎

3.3.2 Construction 2

In some application it could be possible that the PUF only possesses a small response space, e.g., $r \in \{0,1\}$. Consequently, we cannot apply our Construction 1 since it supports a large space of both challenges and responses.

From there we propose a second construction which deals with a large challenge space and a small response space. Intuitively, our second construction is similar to Construction 1 except that we blow up the response length by repeating the PUF evaluation step many times. We formalize the intuition in the following construction:

Construction 2 *Let* $\mathsf{PUF} = (\mathsf{Assemb}, \mathsf{Eval})$ *be a physically unclonable function with range* $2^{|R|}$. *We construct a reconfigurable physically unclonable function* $\mathsf{rPUF} = (\mathsf{rAssemb}, \mathsf{rEval}, \mathsf{Reconfig})$ *with range* $2^{|R|*x}$ *as follows:*

Assembling *On input of the security parameter* 1^n *the algorithm* rAssemb *runs the assembling algorithm of the physically unclonable function to obtain a PUF, i.e.,* $\mathsf{PUF} \leftarrow \mathsf{Assemb}(1^n)$. *It chooses a value* $s \leftarrow \{0,1\}^n$ *at random, an extension factor* $x \in \{0,1\}^n$, *and a hash function* $H \leftarrow \mathsf{hGen}(1^n)$. *The algorithm* rAssemb *then returns* $(\mathsf{rPUF}, \mathsf{PUF}, s, x, H)$.

Evaluation *On input of the challenge* c, *the extension factor* x, *and the state* s, *the algorithm* rEval *sets* $w^j = H(c||s||j)$ *and runs the* PUF *evaluation algorithm on* w^j, *i.e.,* $q^j \leftarrow \mathsf{Eval}(w^j)$ *for* $j = 1, \ldots, x$. *It then returns the* $x \times m$*-bit response* $q = (q^1, \ldots, q^x)$.

Reconfiguration *On input* rPUF *the reconfiguration process* Reconfig *picks* $s' \leftarrow \{0,1\}^n$ *uniformly at random and sets* $s \leftarrow s'$.

Theorem 3.2 *If the physically unclonable function* PUF *is unpredictable and* H *is a collision-resistant hash function, then Construction 2 is an unpredictable reconfigurable physically unclonable function.*

Proof The main idea is as follows: Suppose to the contrary that Construction 2 is insecure. We then construct an attacker $\mathcal{B}$ that (1) breaks the unpredictability of the underlying PUF or that (2) finds collisions in the collision–resistant hash function H. But before describing the algorithm $\mathcal{B}$ we distinguish between two types of attacker that $\mathcal{A}$ can emulate. Suppose that $\mathcal{A}$ adaptively asks for challenges $(c_1, ..., c_L, c_{L+1}, ..., c_C)$ and obtains the corresponding response $(q_1, ..., q_L, q_{L+1}, ..., q_C)$. Recall that $q_i = q_i^1, ..., q_i^x$ with $q_i^j \leftarrow \mathsf{rPUF}(w_i^j)$ where

$$w_i^j := \begin{cases} H(s||c_i||j) \text{ for } i = c_1, ..., c_L, j = 1, ..., x \\ H(s'||c_i||j) \text{ for } i = c_{L+1}, ..., c_C, j = 1, ..., x \end{cases}.$$

Let (c_*, r_*) be the adversary's prediction where $r_* = r_*^1, ..., r_*^x$ and $r_*^j \leftarrow \mathsf{rPUF}(w_*^j)$ with $w_*^j = H(s'||c_*||j)$. We say that

Type-1 attacker $\mathcal{A}$ is a type-1 attacker, denoted by $\mathcal{A}_{col}$, if there exists indices i, j, k such that $w_i^j = w_*^k$ with $w_*^j = H(s'||c^*||j)$ and $i \in \{1, \ldots, L, L+1, \ldots, C\}, j, k \in \{1, ..., x\}$.

Type-2 attacker $\mathcal{A}$ is a type-2 attacker, denoted by $\mathcal{A}_{pred}$, if no such indices exist.

In the following we show for each type of attacker, $\mathcal{A}_{col}, \mathcal{A}_{pred}$, how to construct a suitable simulator $\mathcal{B}_{col}, \mathcal{B}_{pred}$. At first, the algorithm $\mathcal{B}$ tosses a coin to guess if $\mathcal{A}$ is a type-1 or a type-2 attacker. Subsequently, $\mathcal{B}$ proceeds as follows:

Type-1 attacker On input key k the algorithm $\mathcal{B}_{col}$ firstly chooses a collision-resistant hash function H, i.e., $H \leftarrow \mathsf{hGen}(1^n)$. Then, it chooses a value $s \leftarrow \{0,1\}^n$ and sets $\mathsf{rPUF}(\cdot) := \mathsf{PUF}(H(s||\cdot||j))$. Now, $\mathcal{B}_{col}$ runs a black-box simulation of $\mathcal{A}_{col}$.

If $\mathcal{A}_L$ sends a challenge c_i to its rPUF oracle, the algorithm $\mathcal{B}_{col}$ computes the answer as follows: It computes $w_i^j = H(s||c_i||j)$ for $j = 1, ..., x$ and queries its external PUF-oracle on w_i^j. It obtains $q_i = q_i^1, ..., q_i^x$, and stores (c_i, w_i^j, q_i^j) in some (initially empty) list $\mathcal{L}$, and forwards q_i to the attacker $\mathcal{A}_L$. At some point $\mathcal{A}_L$ stops outputting some state information st.

After obtaining st, the algorithm $\mathcal{B}_{col}$ changes the reconfiguration value s to a fresh randomly chosen value $s' \leftarrow \{0,1\}^n$ in order to reconfigure the rPUF. It then executes $\mathcal{A}_C$ as a black-box on input st. According to our construction we denote by rPUF' the reconfigured PUF. It then runs a black-box simulation of $\mathcal{A}_C$ on input st (with access to rPUF').

If $\mathcal{A}_C$ sends a challenge c_i to its rPUF' oracle, the algorithm $\mathcal{B}_{col}$ computes the answer as follows: It computes $w_i^j = H(s'||c_i||j)$ for $j = 1, ..., x$ and queries its external PUF-oracle on w_i^j, i.e., $q_i^j \leftarrow \mathsf{rPUF}(w_i^j)$. It then obtains $q_i = q_i^1, ..., q_i^x$, stores (c_i, w_i^j, q_i^j) in some list $\mathcal{L}$, and forwards q_i to the attacker $\mathcal{A}_C$. At some point $\mathcal{A}_C$ stops outputting a challenge-response pair (c_*, q_*).

The algorithm $\mathcal{B}_{col}$ now parses $((w_1^1, ..., w_1^x), ..., (w_L^1, ..., w_L^x), (w_{L+1}^1, .., w_{L+1}^x), ..., (w_C^1, ..., w_C^x))$ and outputs the first element $w_i^j = w_*^k$ with $w_*^j = H(s'||c^*||j)$ and $i \in \{1, \ldots, L, L+1, \ldots, C\}, j, k = 1, ..., x$. If such an element does not exists, then $\mathcal{B}_{col}$ aborts.

For the analysis assume that $\mathcal{A}_{col}$ is a type-1 attacker that succeeds with non-negligible probability. Observe that both algorithms are efficient and that $\mathcal{B}_{col}$ performs a perfect simulation from $\mathcal{A}_{col}$'s point of view. Then there exists an index i such that $w_i^j = w_*^k$. But if $w_i^j = w_*^k$, then $\mathcal{B}_{col}$ has found a collision in the hash function H. However, this is a contradiction to the collision-resistance of the hash function and thus, such an attacker cannot exists.

Type-2 attacker On input the security parameter n the algorithm $\mathcal{B}_{pred}$ proceeds as follows: $\mathcal{B}_{pred}$ chooses a reconfiguration value $s \leftarrow \{0,1\}^n$ and a collision-resistant hash function H. It then runs a black-box simulation of $\mathcal{A} = (\mathcal{A}_L, \mathcal{A}_C)$.

Now, consider the first phase of the unpredictability game. If $\mathcal{A}_L$ sends a challenge c_i to its rPUF oracle, the algorithm $\mathcal{B}_{pred}$ computes the answer as follows: It sets $w_i^j = H(s||c_i||j)$ for $j = 1, ..., x$ and it sends these values to its external PUF oracle. It turn, $\mathcal{B}_{pred}$ obtains $q_i = q_i^1, ..., q_i^x$, stores (c_i, w_i^j, q_i^j) in some (initially empty) list $\mathcal{L}$, and forwards q_i to the attacker $\mathcal{A}_L$. At some point $\mathcal{A}_L$ stops outputting some state information st.

After obtaining st, the algorithm $\mathcal{B}_{pred}$ updates the reconfiguration value s to fresh and randomly chosen value $s' \leftarrow \{0,1\}^n$ in order to reconfigure the rPUF. According to our construction we denote by rPUF' the reconfigured PUF. It then runs a black-box simulation of $\mathcal{A}_C$ on input st (with access to rPUF').

If $\mathcal{A}_C$ sends a challenge c_i to its rPUF' oracle, the algorithm $\mathcal{B}_{pred}$ proceeds as follows: It sets $w_i^j = H(s'||c_i||j)$, queries its external PUF-oracle on w_i^j, and in turn it obtains $q_i = q_j^1, ..., q_j^x$. Finally, it stores (c_i, w_i^j, q_i^j) in $\mathcal{L}$, and forwards q_i to the attacker $\mathcal{A}_C$. At some point $\mathcal{A}_C$ stops outputting a challenge-response pair (c_*, r_*).

Finally, the adversary $\mathcal{B}_{pred}$ computes $w_*^j = H(s'||c_*||j)$. It then outputs the first element w_*^j that is not $((w_1^1, ..., w_1^x), ..., (w_L^1, ..., w_L^x), (w_{L+1}^1, .., w_{L+1}^x), ..., (w_C^1, ..., w_C^x))$.

For the analysis assume that $\mathcal{A}_{pred}$ is a type-2 attacker that succeeds with non-negligible probability. Observe that both algorithms are efficient and that $\mathcal{B}_{pred}$ performs a perfect simulation from $\mathcal{A}_{col}$'s point of view. Then, the success probability (denoted by win) of the algorithm $\mathcal{B}_{pred}$ can be calculated as follows:

$$\begin{aligned}\Pr[\mathcal{B}\ \mathsf{win}] &= \Pr[\mathcal{A}\ \mathsf{win}] \cdot \Pr[\mathcal{B}\ \mathsf{win}|\mathcal{A}\ \mathsf{win}] + \Pr[\mathcal{A}\ \neg\mathsf{win}] \cdot \Pr[\mathcal{B}\ \mathsf{win}|\mathcal{A}\ \neg\mathsf{win}] \\ &= \left(\frac{1}{|R| \cdot x} + \epsilon(n)\right) \cdot 1 + \left(1 - \left(\frac{1}{|R| \cdot x} + \epsilon(n)\right)\right) \cdot \frac{1}{|R|} \\ &= \frac{1}{|R|} + \frac{|R| - 1}{|R|^2 \cdot x} + \frac{\epsilon(n) \cdot (|R| - 1)}{|R|}\end{aligned}$$

The algorithm $\mathcal{B}_{pred}$ predicts the PUF with probability $\frac{1}{|R|} + \delta(n)$, where $\delta(n) = \frac{|R|-1}{|R|^2 \cdot x} + \frac{\epsilon(n) \cdot (|R|-1)}{|R|}$ is non-negligible and > 0. Since $|R| >= 1$ both terms $\frac{|R|-1}{|R|^2 \cdot x}$ and $\frac{\epsilon(n) \cdot (|R|-1)}{|R|}$ are > 0. ∎

4 Recyclable Tokens

Token-based access control systems are a pervasive technology that is used in many different applications. An emerging trend in practice is to use wireless tokens based on radio frequency identification (RFID) systems. These systems allow for the fully automatic identification and authentication of objects and their users and are increasingly employed in *security-critical* areas and in various applications where *privacy-sensitive* information is entrusted to tokens.

However, despite their benefits, RFID-enabled systems also pose challenging security and privacy risks (see, e.g., [WSRE03, Jue06, ASVW10]). Indeed, RFID tokens typically are computationally and memory constrained devices without protection against physical tampering [Atm11, NXP11]. Hence, hardware attacks that reveal the secrets of tokens impede the use of cryptographic authentication schemes and allow for cloning and forgery of tokens. Moreover, existing privacy-protecting schemes cannot be applied directly to RFID-based tokens due to their high computational complexity. Hence, designing economically efficient RFID-enabled access control systems that are resistant to hardware attacks and that preserve the privacy of their users, is a challenging open problem.

One approach to solve this problem are physically unclonable functions (PUFs, see Section 2.2). The security properties of PUFs can be used to prevent cloning and impersonation of PUF-enabled tokens without adding expensive secure or trusted components to the token. In literature, several approaches to PUF-based authentication and identification have been proposed (see, e.g., [REC04, TB06, OHS08, SVW10]), while commercial PUF-enabled RFID tokens are already on the market [Ver11].

In many applications of token-based access control systems (e.g. ski passes, gift certificates, transportation tickets), the tokens are recycled. This means that, in case a user does not need his token any longer, the token is returned to the operator of the access control system where it is reissued and later given to a new user. This approach is economically efficient since the operator does not need to buy new tokens. Moreover, this is environment-friendly since integrated circuits are difficult to recycle. However, when recycling a token, it must be ensured that the security objectives of the operator of the access control system and the previous, current and future users of that token are not violated. For instance, the new user of a reissued token should not be able to illegitimately obtain the access rights of a previous user or gain any privacy-sensitive information about the previous or future users of that token.

Existing PUF-based authentication schemes either integrate the PUF in the authentication protocol or use it to bind a cryptographic secret to the hardware of a specific token. Note that existing PUFs are static components that typically cannot be updated without significantly changing the security and functional properties of the PUF. However, reissuing a PUF-enabled token in a secure way, i.e., such that the security and privacy properties of the corresponding identification/authentication scheme are preserved for previous and future users of that token, usually would require replacing or changing the PUF component on the token, which is not feasible in practice. This can be realized by using reconfigurable PUFs, which have been introduced in Section 3.

In this section, we describe a token-based access control system that supports recyclable unclonable PUF-enabled tokens. Our approach extends existing PUF-based authentication schemes by integrating logically reconfigurable PUFs (see Section 3). The integration of LR-PUFs is not straightforward. In particular, the efficient verification of LR-PUF responses is a challenging problem. Previous approaches using a database of PUF challenge-response pairs (CRPs) cannot be applied directly to LR-PUFs since reconfiguring an LR-PUF changes

its challenge-response behavior and thus invalidates the CRP database, while having a CRP database for each reconfiguration state is inefficient and impractical. However, the verifier must be able to verify the outputs of the reconfigured LR-PUF while this must be infeasible for the adversary.

4.1 Use Case Description

A token-based access control system consists of a *system operator* $\mathcal{O}$, a set of *tokens* $\mathcal{T}$, and a set of *verifiers* $\mathcal{V}$, as depicted in Figure 2. $\mathcal{O}$ is the entity that initializes and maintains

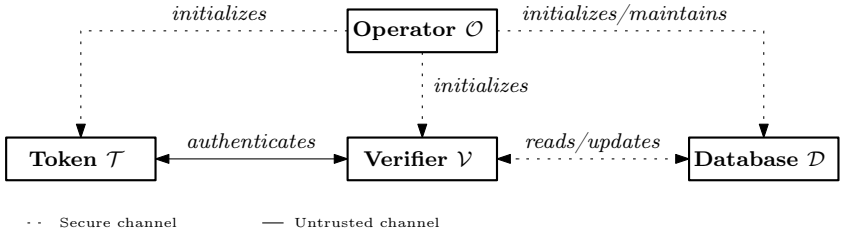

Figure 2: Token-based access control system

the token-based authentication system. Hence, each token and verifier is initialized by $\mathcal{O}$ before they are deployed in the system. Tokens and verifiers are called *legitimate* if they have been initialized by $\mathcal{O}$. In many applications $\mathcal{T}$ is an embedded device with constrained computing and memory capabilities that is either equipped with a wired or radio interface (e.g., a smartcard or an RFID chip). Usually $\mathcal{T}$ is attached to some object or carried by a user of the access control system. $\mathcal{V}$ is a stationary or mobile computing device that interacts with $\mathcal{T}$ when it is inserted into some card reader or gets into the radio reading range of $\mathcal{V}$. The main purpose of this interaction usually is the authentication of $\mathcal{T}$ to $\mathcal{V}$. Depending on the use case, $\mathcal{V}$ may also authenticate to $\mathcal{T}$ and/or obtain additional information like the identity of $\mathcal{T}$. $\mathcal{V}$ can have a sporadic or permanent online connection to some backend system $\mathcal{D}$, which typically is a database maintaining detailed information on all tokens in the system. This database is initialized and maintained by $\mathcal{O}$ and can be read and updated by $\mathcal{V}$.

Trust and adversary model. The system operator $\mathcal{O}$ maintains the token-based access control system, and thus is considered to behave correctly. Moreover, we assume that tokens $\mathcal{T}$ and verifiers $\mathcal{V}$ are initialized in a secure environment that cannot be accessed by the adversary. Likewise, $\mathcal{T}$ is reinitialized in the same secure environment before being reissued. However, $\mathcal{O}$ may be curious and collect user information (see, e.g., [WSRE03, Jue06, ASVW10]). In case $\mathcal{T}$ and $\mathcal{V}$ communicate over a radio link, every entity can eavesdrop and manipulate this communication, even from outside the nominal reading range of $\mathcal{V}$ and $\mathcal{T}$ [KW06]. Thus, the adversary can be every (potentially unknown) entity. Besides the communication between $\mathcal{T}$ and $\mathcal{V}$, the adversary can also obtain useful auxiliary information (e.g., by visual observation) on whether $\mathcal{V}$ accepted $\mathcal{T}$ as a legitimate token [JW06, Vau07]. Moreover, each token is assumed to be equipped with a LR-PUF as defined in Section 3.

Security objectives. The major security objectives are to prevent the creation of illegitimate (forged) tokens that are accepted by honest verifiers, prevent impersonation and copying (cloning) of legitimate tokens, and to make sure that it is impossible to permanently prevent

users from using legitimate tokens by exploiting deficiencies in the underlying protocols (denial-of-service attack) [BvLdM06]. In case of reusable tokens, another major threat would be the impersonation of previous or future users of a token. The most pressing privacy risk concerns the tracking of users (and their identities), which allows the creation and misuse of detailed user profiles. For instance, detailed movement profiles can leak sensitive information on the personal habits and interests of the user of a token. This is of particular importance in access control systems using wireless tokens [WSRE03, Jue06, ASVW10]. Thus, an RFID-enabled token-based access control system should provide anonymity as well as untraceability of tokens even if all the information stored on a token (e.g., its authentication secret) has been disclosed. Anonymity means the confidentiality of the identity of a token whereas untraceability refers to the unlinkability of the communication of a token.

The security requirements for a token-based access control system supporting recyclable tokens can be summarized as follows:

- *Unforgeability of tokens:* Only tokens that have been initialized by the system operator must be accepted by honest verifiers.
- *Availability:* Only the system operator can revoke and reinitialize tokens.
- *Revocability:* A revoked token must not be accepted by the verifiers.
- *Impersonation resistance:* The user of a reinitialized token must not be able to pretend to be one of the previous users of that token. Moreover, the new user of a token should not be able to make use of the access rights that have been granted to that token for previous users. Further, an old user of a reinitialized token must not be able to pretend to be the new user of that token.
- *Confidentiality:* The user of a reinitialized token must not be able to obtain any private information (e.g., travel behavior) of previous and future users of that token.
- *Anonimity:* Only the system operator should be able to link the identity of the owner to the corresponding token.
- *Untraceability:* Only the system operator should be able to link the transactions of a token, even in case all information stored on the token has been disclosed.

4.2 Analysis of Existing Solutions

To prevent cloning of a token, it must be infeasible to determine its authentication secrets by both attacking the corresponding authentication protocol as well as by physically attacking the token. One solution to counter cloning attacks is to employ established physical protection mechanisms that aggravate reading out the memory (e.g., the cryptographic authentication keys) of a token [SA02, NPSQ03]. However, this would dramatically increase the price of tokens and render them inappropriate for most commercial applications. A more economic solution to prevent cloning can be realized by using physically unclonable functions (PUFs, see Section 2.2). In general, there are three different approaches to PUF-based authentication of hardware tokens, which will be discussed in the following.

Authentication Directly Based on PUF

PUFs can be used directly to identify and to authenticate hardware tokens as depicted in Figure 3. Therefore, the token $\mathcal{T}$ must be equipped with a PUF but does not need to provide

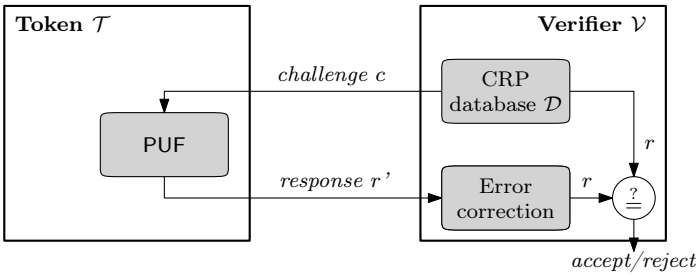

Figure 3: Authentication directly based on PUF

other security functionalities. The verifier $\mathcal{V}$ needs access to a database $\mathcal{D}$ that contains a subset of challenge-response pairs (CRPs) of the PUF of each token. To identify/authenticate $\mathcal{T}$, $\mathcal{V}$ picks a random challenge c from $\mathcal{D}$ and sends it to $\mathcal{T}$. On receipt of c, $\mathcal{T}$ replies with the response $r \leftarrow \mathsf{PUF}(c)$ of its PUF. $\mathcal{V}$ accepts $\mathcal{T}$ only if there is a tuple $(ID, c, r) \in \mathcal{D}$ and returns the identity ID of $\mathcal{T}$. Otherwise, $\mathcal{T}$ is rejected.

Existing solutions. One of the first proposals following this concept has been introduced by [REC04]. This approach has later been implemented on an RFID-enabled token and its security and usability has been analyzed in [DSP+08]. A privacy-preserving variant of the scheme in [REC04] offering untraceability of tokens has been proposed in [BR07]. The solutions presented in [BR07, CGP+09] address the requirement of the verifier having permanent access to the CRP database, which can be problematic in certain practical scenarios. To overcome the requirement of an online database at the verifier side, [OHS08, HOS08] introduce the concept of simulatable PUFs. A simulatable PUF is a PUF that can be described by a mathematical model that allows for the algorithmic computation of the PUF response to any given PUF challenge. According to [OHS08, HOS08], this mathematical model can only be derived from special PUF measurements that can only be taken during the manufacturing process of the PUF, which seems to be infeasible to achieve in practice. Moreover, simulatable PUFs seem to contradict the unclonability property of PUFs. To set up the mathematical model of the PUF, additional measurements must be taken during the PUF production, which requires some kind of special interface to the PUF. This interface is very likely to be probe pads for the wafer prober, which are not be bonded out to the package pins. It is assumed in [OHS08, HOS08] that this interface is permanently disabled such that it cannot be accessed any more after PUF deployment. However, in face of sophisticated reverse engineering techniques, this assumption seems to be quite strong in practice. In fact, the reverse engineering effort required actually seems to be relatively low, since it should be obvious which pads have not been bonded out. Hence, the interface to the PUF could be reactivated after deployment and used by the adversary to obtain the mathematical model and to simulate the PUF.

Discussion. Direct PUF-based identification/authentication is perfectly suited for applications that must use cost-efficient and hence typically resource-constrained tokens. However, a drawback of the CRP database approach is that CRPs cannot be reused since this would

enable replay attacks and allow for the tracing of tokens. Hence, in this case, the number of token authentications is limited by the number of challenge-response pairs in the verifier's database $\mathcal{D}$.

Authentication Based on Cryptography and PUF-based Key Storage

PUFs can be used to securely store data, e.g., a cryptographic secret [vTO05, LLG+05]. Such a PUF-based key storage can be combined with arbitrary standard cryptographic authentication protocols, as shown in Figure 4. Before deployment, token $\mathcal{T}$ must be initialized in a secure

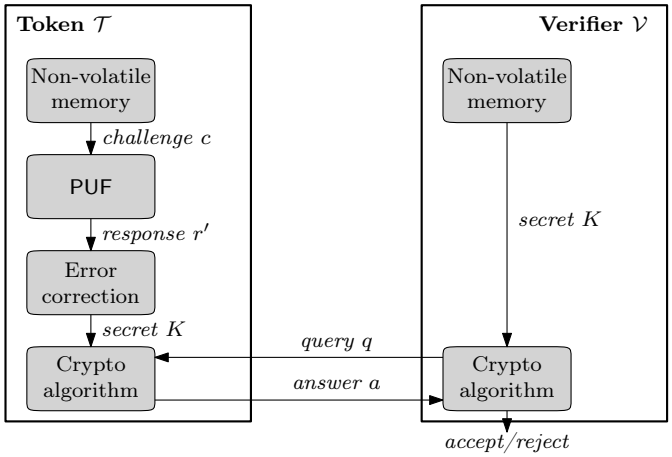

Figure 4: Authentication based on cryptography and PUF-based key storage

environment. During the initialization process a secret key K is established between $\mathcal{T}$ and the verifier $\mathcal{V}$. However, instead of storing the authentication secret of a token $\mathcal{T}$ in tamper-resistant memory, the PUF is used to reconstruct the secret whenever it is needed. Therefore, $\mathcal{T}$ stores in its non-volatile memory the challenge c from which the authentication secret K of $\mathcal{T}$ is derived. Whenever K must be used, $\mathcal{T}$ queries its PUF with c to obtain the corresponding PUF response r'. Since r' typically is slightly different on each read-out of the PUF and not a uniformly distributed value, r' cannot be used directly as a cryptographic secret. Thus, error correction and privacy amplification must be applied to r' to derive K (see Section 2.2). In practice, this can be achieved by fuzzy extractors [DRS04]. Finally, K can be used in a standard authentication protocol to authenticate $\mathcal{T}$ to the verifier $\mathcal{V}$. Since K is inherently hidden within the physical structure of the PUF of $\mathcal{T}$, obtaining K by hardware-related attacks is supposed to be intractable for real-world adversaries.

Existing solutions. The authors of [TB06] propose to use a PUF-based key storage to protect the authentication secrets of RFID-enabled tokens. A privacy-preserving variant of this approach has been presented in [SVW10].

Discussion. The advantage of this approach is that it can be used with any existing authentication scheme and does not require any modifications to the hard- and software of existing verifiers. According to [TB06], a PUF-based key storage can be implemented with less than 1000 gates, which is feasible with current RFID chips. However, besides the PUF and the

error correction scheme, the token must additionally contain the implementation of a cryptographic authentication scheme. Although there are many optimized cryptographic algorithms and implementations specifically designed for resource-constrained devices, the efficient integration of both a PUF-based key storage and a cryptographic scheme on an RFID chip is a challenging task in practice. Moreover, since all the different components shown in Figure 4 are crucial for the security of the authentications scheme, these building blocks and the busses connecting them must be implemented such that they are resistant to hardware attacks. In practice, this assumption is difficult to achieve. Only recently, the authors of [KS10] have shown the feasibility of side-channel attacks against error correction mechanisms as used in PUF-based key storage.

Authentication Based on Hardware-entangled Cryptography

Another approach to PUF-based authentication is hardware-entangled cryptography. This means that the PUF is used as an integral component of the cryptographic authentication scheme (see Figure 5). More in detail, the token $\mathcal{T}$ implements a hardware-entangled crypto-

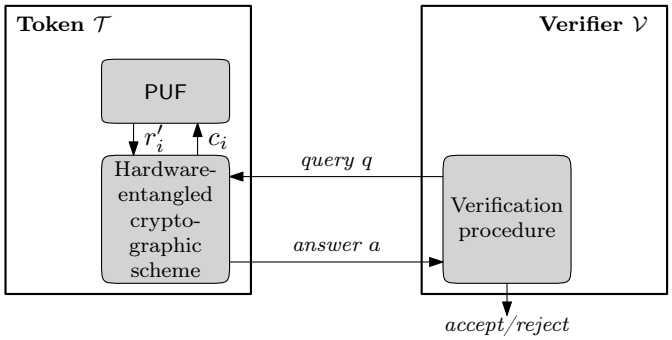

Figure 5: Authentication based on hardware-entangled cryptography

graphic scheme that is based on a PUF. To authenticate $\mathcal{T}$, the verifier $\mathcal{V}$ sends a query q to $\mathcal{T}$, which then replies with an answer a that is computed using the hardware-entangled cryptographic scheme on input q. During this computation, one or more challenges c_i to the PUF are generated and the corresponding responses r_i' are included into the subsequent processing steps that compute a. To verify a, $\mathcal{V}$ must be in possession of an appropriate verification procedure. In case the answer a returned by $\mathcal{T}$ is valid, $\mathcal{V}$ accepts $\mathcal{T}$. Otherwise $\mathcal{T}$ is rejected.

Existing solutions. Currently, the only known hardware-entangled cryptographic scheme is a PUF-based block cipher that has been presented in [AMS+09]. However, since the encryption and decryption can only be performed by the device containing the underlying PUF, this scheme cannot be used as a basis of an hardware-entangled cryptographic authentication scheme.

Discussion. The problem of hardware-entangled cryptography based on PUFs is that the underlying computations can only be performed by the device containing the PUF. Hence, it is difficult for the verifier to check whether the answer of the token has been computed correctly. To build authentication schemes from PUF-based cryptographic primitives, some kind of *publicly verifiable* PUF is needed. Such a verifiable PUF would allow for checking

whether a certain response to some challenge has actually been generated by a particular PUF without having access to the PUF or the corresponding CRP database. However, the design of such PUFs currently is an open research problem.

4.3 Security Architectures for Recyclable Tokens

In this section, we describe how to realize recyclable secure and privacy-preserving access tokens. Our solution extends existing approaches to PUF-enabled token-based access control systems by using logically reconfigurable PUFs (see Section 3). More precisely, we adapt the direct PUF-based authentication and the PUF-based key storage approach to support LR-PUFs. Hereby, it must be carefully considered that the integration of the LR-PUF does not subvert the security and privacy features of existing approaches. Moreover, for practical applicability it is crucial that the verification procedure of the verifier is still efficient.

Authentication Directly Based on PUF

We adapt the general concept of direct PUF-based authentication by replacing the PUF of the token $\mathcal{T}$ by a LR-PUF. To revoke and reissue $\mathcal{T}$, the LR-PUF of $\mathcal{T}$ is reconfigured, which (ideally) corresponds to replacing the PUF of $\mathcal{T}$ with a different one. However, by reconfiguring the LR-PUF of $\mathcal{T}$, the CRP database $\mathcal{D}$ of the verifier $\mathcal{V}$ is invalidated. To counter this problem, instead of having a database of CRPs of the LR-PUF, $\mathcal{V}$ is given access to a database containing the CRPs for the *real* PUF underlying the LR-PUF construction and the LR-PUF state of each token. Since the algorithms of the control unit (i.e., the input and output transition functions and the state update procedure) are publicly known, $\mathcal{V}$ can use the CRPs and state information of the database to recompute the LR-PUF response for a given challenge and any state of $\mathcal{T}$. However, this requires $\mathcal{V}$ and the LR-PUF to be synchronized w.r.t. the LR-PUF state.

Note that this approach is very efficient to implement on the token side since, besides the LR-PUF construction, no other components are required on the token. In scenarios, where severely resource-contained tokens must be used, the optimized LR-PUF construction presented in Section 3.3 is particularly suitable.

Authentication Based on Cryptography and PUF-based Key Storage

We adapt the approach that combines PUF-based key storage and cryptographic authentication techniques by replacing the PUF of the PUF-based key storage with a LR-PUF. To revoke and reissue a token $\mathcal{T}$, the LR-PUF of $\mathcal{T}$ is reconfigured, which (ideally) corresponds to replacing the PUF of $\mathcal{T}$ with a different one. This invalidates the old authentication secret K of $\mathcal{T}$ and creates a new authentication secret $K' \neq K$. More in detail, when $\mathcal{T}$ attempts to reconstruct its authentication secret after reconfiguration, the LR-PUF will return a different response $\tilde{r}' \neq r'$. Hence, applying the error correction to $\tilde{r}'$ leads to a different authentication secret $\tilde{K} \neq K$. To verify $\mathcal{T}$, the verifier $\mathcal{V}$ must know the new authentication secret $\tilde{K}'$ of $\mathcal{T}$. Therefore, the reinitialization procedure of $\mathcal{T}$ must include a key transport mechanism to securely transfer $\tilde{K}'$ to $\mathcal{V}$. This can be realized by a standard key transport protocol. Ideally, this protocol is based on the cryptographic primitives of the cryptographic algorithm used for the authentication of $\mathcal{T}$ to $\mathcal{V}$ and does not require additional components.

The advantage of this approach is that it does not require any modifications to the verifiers. However, as discussed in Section 4.2, the implementation on the token side is rather complex

and the security of this approach is based on assumptions that are difficult to achieve in practice.

5 HW/SW-Binding

In many of today's electronic products, embedded software plays an important role. Crucial functional parts of a device are implemented in hardware (i.e., in integrated circuits) but more and more higher layers (e.g., drivers, control, user interface) are implemented in embedded software or firmware that is running on an internal processor or micro-controller. This keeps the design flexible. Changing the software is easier and less costly than making new hardware. Furthermore it makes the design cycle shorter since bugs can easily be repaired in later software upgrades.

An advantage of software flexibility is the possibility to use it for device differentiation. The functionality of a device can be changed by updating the software image with a version that enables more features. Depending on the type of device, such an update mechanism can be implemented in the field. If for example the user pays an additional fee, he can download an updated software version and activate new features on his device. The product differentiation feature can however be abused by attackers. They will try to upgrade devices by simply copying the software of one device to another device. This way the attacker can counterfeit a more expensive product, without paying for it (counterfeiting scenario). In other cases software needs to be upgraded in order to fix certain security holes in a system. Once such a hole is discovered, the manufacturer wants to update the software of the devices in order to fix the security problem. After the bug fix has been performed attackers might try to downgrade the software to a previous version and exploit the security hole to their advantage (security patch scenario).

5.1 Use Case Description

System model. The HW/SW-binding system from this use case consists of a *system operator* $\mathcal{O}$, different versions of *software* $\mathcal{S}$, and a set of *devices* $\mathcal{D}$. $\mathcal{O}$ is the entity that initializes and maintains the HW/SW-binding system. Each $\mathcal{D}$ that is supplied by $\mathcal{O}$ contains some *legitimate* version of $\mathcal{S}$. After supplying $\mathcal{D}$ it is possible for $\mathcal{O}$ to change (or update) the version of $\mathcal{S}$ remotely in a secure and authenticated manner. This update can either be performed on all supplied $\mathcal{D}$ or on a subset of $\mathcal{D}$, which has been selected by $\mathcal{O}$. Different versions of $\mathcal{S}$ can either differentiate between the functionality that is allowed by $\mathcal{O}$ on $\mathcal{D}$ or new versions of $\mathcal{S}$ can be released in order to patch (security) bugs in older versions of $\mathcal{S}$. Finally, $\mathcal{D}$ is the hardware that has been supplied by $\mathcal{O}$, which can run the version of $\mathcal{S}$ that $\mathcal{O}$ allows for that specific $\mathcal{D}$.

Trust and adversary model. The system operator $\mathcal{O}$ maintains the system, and is thus considered to behave correctly. Moreover, we assume that $\mathcal{D}$ and $\mathcal{S}$ are initialized in a secure environment. However, the version of $\mathcal{S}$ on a legitimate $\mathcal{D}$ should be able to be changed remotely by $\mathcal{O}$ in a secure way.

An adversary in this use case is anyone who attempts to run a version of $\mathcal{S}$ on hardware in a manner that has not been authorized by $\mathcal{O}$. It is unauthorized if an adversary attempts to run any version of $\mathcal{S}$ on hardware that has not been approved by $\mathcal{O}$. Another possible attack is to run a different version of $\mathcal{S}$ on an authorized $\mathcal{D}$ than the version that has been allowed by $\mathcal{O}$. This can either be a version of $\mathcal{S}$ with more functionality or an older version of $\mathcal{S}$ with known (security) bugs.

Security objectives. If $\mathcal{S}$ can be copied from one $\mathcal{D}$ to another, this could lead to a serious loss in revenue for $\mathcal{O}$. Consider the situation where $\mathcal{O}$ supplies smart-cards for pay-TV applications. These cards ($\mathcal{D}$) run a certain version of $\mathcal{S}$, which allows the users of the smart-card to watch a selected set of pay-TV channels. If it is possible to copy $\mathcal{S}$ from one $\mathcal{D}$ to another, it will become possible to copy $\mathcal{S}$ from a card with the most complete pay-TV package and copy this $\mathcal{S}$ to other $\mathcal{D}$, which have not been authenticated to watch these channels. This way attackers can watch all channels with any $\mathcal{D}$, even if they do not pay for this functionality. Therefore, $\mathcal{O}$ could sustain a big loss in revenue.

Furthermore, consider a security sensitive application that runs $\mathcal{S}$ on $\mathcal{D}$. In case this application becomes compromised due to a breach of the security of $\mathcal{S}$, it should be possible for $\mathcal{O}$ to patch $\mathcal{S}$ (remotely). After the security patch has been performed, it should be impossible for an attacker to downgrade $\mathcal{S}$ to the previous bugged version. If an attacker is able to downgrade $\mathcal{S}$, this will cause a security risk that can lead to loss of revenue or reputational damage for $\mathcal{O}$. For certain applications it might even lead to reliability or safety issues.

The security requirements for HW/SW-binding can be summarized as follows:

- *Software authenticity:* Prevent running software on unauthorized hardware, i.e., an adversary should not be able to copy and use the software on another device.
- *Downgrading resistance:* Prevent downgrading of software, i.e. an adversary should not be able to downgrade the software to a previous version, which may have known (security) bugs.

5.2 Analysis of Existing Solutions

Existing anti-counterfeiting methods. Systems exist in which software is bound to specific hardware by encrypting parts of the software with a unique key or identifier that is permanently stored in the device. Only the device that has the correct key is able to decrypt the correct software. Copying the software to a second device will not work since the cryptographic key is different on the second device. This could be used to prevent counterfeiting.

Problem of existing methods. There is however a problem with the existing method. The system can be circumvented if the cryptographic key or identifier can be copied from one system to another. Therefore the key must be stored in a relatively expensive secure non-volatile memory.

5.3 Security Architectures for HW/SW-Binding

Solutions to the problems of existing methods. In order to solve the problem of secure key storage, we propose to use a PUF. PUFs can be used to securely generate cryptographic keys on a device and in a cost-effective way. PUF keys are derived "on the fly" based on the intrinsic characteristics of the hardware and are never stored permanently in a device. Since the PUF key is only available in the very limited amount of time when it is used in a cryptographic computation, it is very difficult to copy such a key. Invasive attacks will not leak information on the PUF key but instead will probably destroy the PUF rendering the hardware useless and the software uncopyable. By encrypting parts of the software with the cryptographic key derived from the PUF, the software can be securely bound to a specific

device. Since PUFs are unique and unclonable, the key that is derived from the PUFs will not be reproducible on another device. Therefore, using a PUF to generate the key that encrypts the software prevents counterfeiting by copying the software from one device to another.

Problem of security patches. Even if the key is securely generated by a PUF and can therefore not be copied, this does not prevent an attacker from downgrading a system to a previous software version. The reason is that the PUF, which has been used to derive the key for the previous software version, produces the same response and therefore the same key for the new version. Because the derived key has not changed an attacker might be able to downgrade the software to its previous (bugged) version. In other words: like the new version, the old version of the software can still be bound to the hardware.

Solution for security patches. In order to prevent the possibility of downgrading software to a previous version, we propose to use a logically reconfigurable PUF as defined in Section 3.3. When a new software version is distributed to the device, the PUF is reconfigured. After reconfiguration, the PUF has a new and unpredictable challenge-response behavior. This results in the fact that the cryptographic key that is derived from the PUF is also renewed. If the previous software is now copied back to the device, it cannot be decrypted because the PUF key has changed. Note that in this application it is important that the reconfiguration cannot be undone. In other words, the reconfiguration must be one-way.

Definition of upgrade (and reconfigure) protocols. In order to support software downgrading it is necessary that in each software-update step the PUF has to be reconfigured. The update could be encoded by a key that is derived from the new PUF state. Thus, a reconfiguration before installing the update is enforced.

6 Evaluation Principles

Raising the assurance level of hardware security solutions is a core objective of the UNIQUE project. One way to view *assurance* is as a metric that provides a confidence level that an implemented solution will perform as intended, even in the presence of adversaries. Determining the assurance level of a system is a challenging task. Security solutions are composed of many individual protections, which are often interdependent and exist at differing abstraction levels, spanning silicon, algorithms, protocols and systems as well as including non-technological aspects such as operational policies and economics. Adversaries will try to attack the system in ways that are typically different from those the designers had envisioned, for example by attempting to manipulate the system environment into atypical states. These aspects necessitate a holistic and multi-disciplinary approach to security engineering.

This section is motivated by the need to analyse the security requirements and assumptions early on in the design process. More comprehensive evaluation and validation activities are the focus of WP3. In WP3, a baseline security evaluation using a subset of the Common Criteria V3.1 [Com11a] will assess whether the solutions within UNIQUE meet the prescribed Common Criteria evaluation assurance level EAL5. This baseline will act at the foundation for further WP3 work.

In this section we will outline a basic evaluation framework which can inform the WP3 activities, as well as providing input into WP2 which focuses on design and implementation. We will then analyse the security architectures and protocols defined in previous sections, in order to extract the security requirements both explicit and implicit. Any assumptions made are investigated as these often inform the attack strategy of the adversary. Since a practical realisation of the developed security solutions is the goal of UNIQUE, this must be done with reference to implementation considerations.

6.1 Evaluation Framework

The difficulty of evaluating whether a system meets a prescribed assurance level is reflected in the significant time and cost associated with a Common Criteria or FIPS140-2 [Fed] evaluation. In the context of the Unique project a complete Common Criteria evaluation is neither appropriate (since evaluations are intended to be applied to a commercial product), nor possible due to time and cost constraints. We outline here an evaluation framework which can inform the evaluation and validation activities of WP3. The scope is restricted to the system embedding the LR-PUF.

LR-PUF primitives The primitives used in the LR-PUF constructions of Section 3 must be evaluated with a focus on practical realisations. Implicit security requirements and assumptions must be extracted and depending on the overall system threat model, the resistance to invasive attacks and side-channel analysis may be evaluated. The security requirements and assumptions of LR-PUFs are analysed in Section 6.2.

LR-PUF construction The algorithms and protocols defining the LR-PUF functionality must evaluated. A formal methodology is appropriate here. Security proofs for the algorithms which define the LR-PUF behaviour are presented in Section 3.2. Further work is required to extended these to include the LR-PUF state transition behaviour. The security implications of an attacker observing or modifying an internal communication pathway between the LR-PUF

primitives should be evaluated. Any decision to mitigate such an attack must be informed by the system threat model.

Implementation Correctness Evaluating whether the implementation matches the specification is key to a achieving a meaningful assurance level. Although the Common Criteria evaluates some aspects of the development process, such as configuration management, fundamentally it evaluates the specification, not the implementation. Noting that evaluating the correctness of the implementation is important, the capability to move seamlessly from a formal security algorithm or protocol specification to a hardware friendly design representation does not exist today. The result is a verification and evaluation gap which unless closed introduces the possibility of weakened security. Formal analysis tools are gaining momentum in the IC design community, where tools employing formal methods [Syn] are employed to evaluate the equivalence of hardware representations at differing levels of abstraction. Such an approach has potential in closing the evaluation gap, particularly in the control unit primitive of the LR-PUF.

6.2 Analysis of Security Requirements

The high level security requirements of a token-based access control system supporting recyclable tokens are made explicit in Section 4.1. These requirements will be the foundation of any security evaluation, and in some cases map directly onto standard Common Criteria security functional requirements (SFR), one example being the FMT_REV: Revocation SFR [Com11b]. With the exception of *Unforgeability* and *Impersonation Resistance*, these requirements will not be analysed further since they have implications which span many aspects of the overall system which are outside of the scope of this deliverable.

Instead we will focus on the LR-PUF constructions introduced in Section 3 which is build on a number of lower level primitives. We will review these primitives to extract the implicit security requirements and assumptions. Any evaluation framework must validate these in addition to the explicit security requirements of Section 4.1 as a necessary first step.

Physically Unclonable Functions In Section 2.2 a formal definition of a physically unclonable function is given. Definition 1 is sufficiently broad to include all known PUF architectures including those with small challenge-response spaces. However, the LR-PUF constructions of Section 3 and the security architectures for recyclable tokens of Section 4.3, strongly point towards a large challenge-response space PUF implementation for both LR-PUF instantiations of Section 3.3. In practice this means an arbiter PUF [GCvDD02b]. The two remaining PUF definitions must be examined in this context.

Definitions 2, 3 and the high level requirement *Unforgeability* are closely related. If it is possible to predict the response to a given challenge without evaluating the PUF then it is possible to clone the PUF, by emulating it with a software model for example. Model building attacks on the arbiter PUF were introduced in [GLC+04]. Here an attacker collects a subset of PUF CRPs and attempts to derive a numerical model from this data, allowing the prediction of responses to arbitrary challenges with reasonable probability. Variants of the arbiter PUF have been proposed to increase the resistance to modelling attacks, such as the the controlled PUF [GCvDD02a], the feed forward arbiter PUF [GLC+04] and the XOR arbiter PUF [SD07]. In [RSS+10] a variety of machine learning techniques are applied to the arbiter PUF and its variants.

With the above in mind it seems prudent that any arbiter PUF based design considers model building attacks for inclusion in the threat model and evaluation framework. The LR-PUF constructions prevent a chosen challenge attack[1] on the underlying PUF, while allowing the raw PUF responses to be read. However, since the hash primitive, LR-PUF state and challenge/state concatenation details are publicly known, the attacker will have knowledge of which challenge was applied thus increasing the feasibility of a modelling attack. Note that the formal security model in Section 3.2 acknowledges the existence of algorithms that can predict the response to a chosen LR-PUF (not PUF) challenge *before* the LR-PUF has been reconfigured. Mitigations can include appropriate selection of the PUF security parameter 1^n from Definition 1 in order to increase the computational complexity of a modelling attack and sizing parameter q_L of Definition 7 such that the number of CRPs required for a model building attack is prohibitive. Using more resilient arbiter PUF variants such as the XOR arbiter PUF and implementing a hardware delay to limit the number of LR-PUF evaluations that can be made per unit time are other options.

LR-PUF State Memory The LR-PUF construction requires non-volatile memory (NVM) in order to store the state. The state storage requirements are low, presumably 128 bits, while the memory must be multiple-time programmable (MTP) to support reconfiguration. Typical embedded flash or EEPROM capacities are a minimum of 4 Kb [TSM11], ruling these out (unless the excess capacity can be otherwise utilised). These technologies also have the disadvantage of requiring additional wafer processing steps which has cost implications. They also tend to lag the leading edge technology nodes by a number of generations. NVM technologies which can be implemented in standard CMOS have been proposed since at least the early '90s [OAT94], although it is only relatively recently that they have become widely available. These technologies are collectively known as logic NVM in the industry. An MTP variant of this technology is available [Syn11] at densities from 128 bits upwards making it a good fit for the LR-PUF application.

As discussed in Section 3.1, the security requirements of the NVM are simplified in that the security of the LR-PUF is not dependent on keeping the LR-PUF state secret. It should be noted that this is dependent on the underlying PUF being in practice unclonable. Assuming that this is the case, an attacker may want to change the LR-PUF state back to an old configuration. If successful, the high level *Impersonation Resistance* requirement would not be met. The details of how state is updated is not considered in this deliverable, however mitigations against this attack can range from state consistency checks at the verifier $\mathcal{V}$ to hardware level protection mechanisms implemented at the token $\mathcal{T}$ to prevent NVM updates to arbitrary states. A TPM-like 'extend' mechanism may be appropriate here [Tru03], where the TPM platform configuration registers are updated with the SHA-1 hash of the previous register data concatenated with the new data.

An attacker may attempt to bypass all higher level protection mechanisms by mounting an invasive attack on the NVM itself, in an attempt to change the contents to an arbitrary state. The storage mechanism used in the NVM under consideration relies on storing charge on a floating gate, a technique also common to EEPROM and flash technologies. Although known attacks on NVM focus on extracting memory content rather than changing memory content, successful modification of EEPROM bits using optical techniques is demonstrated in [Sko05b]. However, such techniques are more suited to single or localised bit fault injection

[1] A chosen challenge attack is one where the attacker is free to apply an arbitrary PUF challenge.

rather than the modification of a complete memory and probably not feasible for modern CMOS technology nodes due to sub-optical wavelength feature sizes. For the well funded and motivated attacker, the use of a focussed ion beam (FIB) allows charge to be deposited or removed from individual bit locations. Increasing the number of bit locations that an invasive attacker must modify is one simple way to increase the effort level and probability of error. Other mitigations include ensuring that the NVM floating gates are buried under metal routing or shielding, although attacks from the silicon backside can bypass these protections.

Memory remanence in NVM technologies should be considered. Data recovery in a limited number of flash devices is reported in [Sko05a]. The techniques developed are not general and therefore may not be applicable to the NVM technologies considered above, however the possibility of data remanence should be considered particularly in the recyclable access token use case.

LR-PUF Control Logic The generic LR-PUF shown in Figure 1 combines a control logic unit with the PUF and NVM primitives. The function of this unit is to implement the algorithms specified in the LR-PUF constructions of Section 3.3. The main security requirement here is for a collision resistant hash function as specified in Section 2.3. In practice, selecting the hash function will involve analysing existing hardware implementations in terms of silicon area and performance, a task which has been addressed in UNIQUE Deliverable D2.1 [UNI10b]. The selection must also be informed by up-to-date findings from the security research community.

The control unit is also responsible for LR-PUF state management, resulting in some implicit security requirements. The first of these is that the state is read-only at the LR-PUF interface. This can be enforced at the hardware level, for example by ensuring that there is no physical write path to the state storage. A sophisticated invasive attacker may be able to bypass such a protection, for example by performing silicon FIB edits. Any decision to include invasive attacks in the threat model must be informed by the properties of the complete system, including non-technological aspects such as the value of the assets being protected and operational policies. Mitigations against invasive attacks are reviewed in UNIQUE Deliverable D1.1 [UNI10a]. In the context of a cost sensitive application such as a recyclable access token, mitigations at the silicon level would likely be restricted to protective coatings, passivation layers and passive shields implemented in top level metal. Note that a design decision as fundamental as choosing a parallel data path over a serial one can raise the effort level for the invasive attacker.

The second implicit security requirement relates to the LR-PUF state transition function. In order to prevent an attacker from predicting future LR-PUF states, the transition function should be unpredictable which results in a random number generator (RNG) requirement. An attacker may be able to bias the RNG by environmental, means such raising the die temperature outside of normal limits. Mitigating against this type of attack in a cost constrained application is challenging, in the recyclable access token it could be mitigated at the operational level since token are reconfigured in a trusted environment.

References

[ABKR08] Mikhail J. Atallah, Eric D. Bryant, John T. Korb, and John R. Rice. Binding software to specific native hardware in a vm environment: the puf challenge and opportunity. In *VMSec '08: Proceedings of the 1st ACM workshop on Virtual machine security*, pages 45–48, New York, NY, USA, 2008. ACM.

[AMS+09] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, B. Sunar, and Pim Tuyls. Memory leakage-resilient encryption based on physically unclonable functions. In *Advances in Cryptology - ASIACRYPT*, volume 5912 of *LNCS*, pages 685–702, 2009.

[ASVW10] Frederik Armknecht, Ahmad-Reza Sadeghi, Ivan Visconti, and Christian Wachsmann. On rfid privacy with mutual authentication and tag corruption. In Jianying Zhou and Moti Yung, editors, *Proceedings of the 8th International Conference on Applied Cryptography and Network Security (ACNS'10)*, volume 6123 of *LNCS*, pages 493–510. Springer, June 22–25 2010.

[Atm11] Atmel Corporation. RFID and smart identification solutions for dozens of markets. `http://www.atmel.com/products/overview_wireless.asp`, 2011.

[BR07] Leonid Bolotnyy and Gabriel Robins. Physically unclonable function-based security and privacy in RFID systems. In *Proceedings of PERCOM*, pages 211–220. IEEE Computer Society, 2007.

[BvLdM06] Mike Burmester, Tri van Le, and Breno de Medeiros. Provably secure ubiquitous systems: Universally composable RFID authentication protocols. In *Proc. of SecureComm*, pages 1–9. IEEE Computer Society, 2006.

[CGP+09] Pier Francesco Cortese, Francesco Gemmiti, Bernardo Palazzi, Maurizio Pizzonia, and Massimo Rimondini. Efficient and Practical Authentication of PUF-based RFID Tags. Technical Report RT-DIA-150-2009, Università degli studi Roma Tre, Dipartimento di Informatica e Automazione, June 2009.

[Com11a] Common Criteria Portal. `http://www.commoncriteriaportal.org/`, 2011.

[Com11b] Common Criteria v3.1 Part 2: Security Functional Requirements. `http://www.commoncriteriaportal.org/files/ccfiles/CCPART2V3.1R3.pdf`, 2011.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances in Cryptology - EUROCRYPT*, volume 3027 of *LNCS*, pages 523–540, 2004.

[DSP+08] Srinivas Devadas, Edward Suh, Sid Paral, Richard Sowell, Tom Ziola, and Vivek Khandelwal. Design and implementation of PUF-based unclonable RFID ICs for anti-counterfeiting and security applications. In *IEEE International Conference on RFID 2008, Las Vegas, NV, USA, 16–17 April, 2008*, pages 58–64. IEEE Computer Society, 2008.

[Fed] Federal Information Processing Standard 140-2 - Security Requirements for Cryptographic Modules. `http://csrc.nist.gov/groups/STM/cmvp/standards.html#02`.

[GCvDD02a] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Controlled physical random functions. In *ACSAC '02: Proceedings of the 18th Annual Computer Security Applications Conference*, page 149, Washington, DC, USA, 2002. IEEE Computer Society.

[GCvDD02b] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Silicon physical random functions. In *ACM Conference on Computer and Communications Security*, pages 148–160, New York, NY, USA, 2002. ACM Press.

[GLC+04] Blaise Gassend, Daihyun Lim, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Identification and authentication of integrated circuits: Research articles. *Concurr. Comput. : Pract. Exper.*, 16(11):1077–1098, 2004.

[GMS09] M. Gora, A. Maiti, and P. Schaumont. A flexible design flow for software ip binding in commodity fpga. In *IEEE Fourth International Symposium on Industrial Embedded Systems - SIES 2009, Ecole Polytechnique Federale de Lausanne, Switzerland, July 8 - 10, 2009*, pages 211–218. IEEE, July 2009.

[HOS08] Ghaith Hammouri, Erdinç Öztürk, and Berk Sunar. A tamper-proof and lightweight authentication scheme. *Pervasive and Mobile Computing*, 4(6), December 2008.

[Jue06] Ari Juels. RFID security and privacy: A research survey. *Journal of Selected Areas in Communication*, 24(2):381–395, February 2006.

[JW06] Ari Juels and Stephen A. Weis. Defining strong privacy for RFID. ePrint, Report 2006/137, 2006.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[KS10] Deniz Karakoyunlu and Berk Sunar. Differences template attacks on PUF enabled cryptographic device. Presented at the IEEE Workshop on Information Forensics and Security (WIFS'10), December 2010.

[KSS+09] Klaus Kursawe, Ahmad-Reza Sadeghi, Dries Schellekens, Pim Tuyls, and Boris Scoric. Reconfigurable physical unclonable functions – enabling technology for tamper-resistant storage. In *2nd IEEE International Workshop on Hardware-Oriented Security and Trust - HOST 2009*, pages 22–29, San Francisco, CA, USA, 2009. IEEE.

[KW06] Ilan Kirschenbaum and Avishai Wool. How to build a low-cost, extended-range RFID skimmer. ePrint, Report 2006/054, 2006.

[LLG+05] Daihyun Lim, Jae W. Lee, Blaise Gassend, G. Edward Suh, Marten van Dijk, and Srini Devadas. Extracting secret keys from integrated circuits. *IEEE Transactions on VLSI Systems*, 13(10):1200–1205, 2005.

[NPSQ03] Michael Neve, Eric Peeters, David Samyde, and Jean-Jacques Quisquater. Memories: A survey of their secure uses in smart cards. In *Proceedings of the Second*

IEEE International Security in Storage Workshop, October 31, 2003, pages 62–72. IEEE Computer Society, 2003.

[NXP11] NXP Semiconductors. MIFARE smartcard ICs. `http://www.mifare.net/products/smartcardics/`, February 2011.

[OAT94] K. Ohsaki, N. Asamoto, and S. Takagaki. A single poly eeprom cell structure for use in standard cmos processes. *Solid-State Circuits, IEEE Journal of*, 29(3):311–316, March 1994.

[OHS08] Erdinç Öztürk, Ghaith Hammouri, and Berk Sunar. Towards Robust Low Cost Authentication for Pervasive Devices. In *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM'08)*. IEEE Computer Society, March 2008.

[REC04] Damith C. Ranasinghe, Daniel W. Engels, and Peter H. Cole. Security and privacy: Modest proposals for low-cost rfid systems. Auto-ID Labs Research Workshop, September 2004.

[RSS+10] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and J "urgen Schmidhuber. Modeling attacks on physical unclonable functions. In *CCS 2010: Proceedings of the 17th ACM conference on Computer and communications security*, pages 237–249, New York, NY, USA, 2010. ACM.

[SA02] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In *4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002), Redwood Shores, CA, USA, August 13–15, 2002, Revised Papers*, volume 2523 of *LNCS*, pages 31–48. Springer Verlag, 2002.

[SD07] G. Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Design Automation Conference*, pages 9–14, New York, NY, USA, 2007. ACM Press.

[Sko05a] Sergei Skorobogatov. Data remanence in flash memory devices. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 339–353. Springer Berlin / Heidelberg, 2005.

[Sko05b] Sergei P. Skorobogatov. Semi-invasive attacks – A new approach to hardware security analysis. Technical Report UCAM-CL-TR-630, University of Cambridge, Computer Laboratory, April 2005.

[SVW10] Ahmad-Reza Sadeghi, Ivan Visconti, and Christian Wachsmann. Puf-enhanced rfid security and privacy. 2nd Workshop on Secure Component and System Identification (SECSI'10), April 26–27 2010.

[Syn] Synopsys formal equivalence checker. `http://www.synopsys.com/Tools/Verification/FormalEquivalence/Pages/default.aspx`.

[Syn11] Synopsys NVM Products. `http://www.synopsys.com/IP/EmbeddedMemories/Pages/AeonNoveaNvm.aspx`, 2011.

[TB06] Pim Tuyls and Lejla Batina. RFID-tags for anti-counterfeiting. In *The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13–17, 2005, Proceedings*, volume 3860 of *Lecture Notes on Computer Science (LNCS)*, pages 115–131. Springer Verlag, 2006.

[Tru03] Trusted Computing Group. TCG TPM specification 1.2, 2003. `http://www.trustedcomputinggroup.org`.

[TSM11] TSMC Embedded Memory. `http://www.tsmc.com/download/brochures/2010_Embedded_Memory.pdf`, 2011.

[UNI10a] UNIQUE project. Deliverable D1.1: Requirements, Threat Models and Report on Building Blocks for Hardware Security, 2010.

[UNI10b] UNIQUE project. Deliverable D2.1: Description and design of intrinsic hardware security, crypto and hardware entangled crypto components, 2010.

[Vau07] Serge Vaudenay. On privacy models for RFID. In *Proc. of ASIACRYPT*, volume 4833 of *LNCS*, pages 68–87. Springer, 2007.

[Ver11] Verayo, Inc. Verayo website — product page. `http://www.verayo.com/product/products.html`, February 2011.

[vTO05] Boris Škorić, Pim Tuyls, and Wil Ophey. Robust key extraction from physical uncloneable functions. In *Applied Cryptography and Network Security (ACNS)*, volume 3531 of *LNCS*, pages 407–422, 2005.

[WSRE03] Stephen A. Weis, Sanjay E. Sarma, Ronald L. Rivest, and Daniel W. Engels. Security and privacy aspects of low-cost radio frequency identification systems. In *Proc. of PerCom*, volume 2802 of *LNCS*, pages 50–59. Springer, 2003.